IDENTIFYING INDIVIDUAL ANIMALS USING RANKING, VERIFICATION, AND CONNECTIVITY

By

Jonathan P. Crall

A Dissertation Submitted to the Graduate

Faculty of Rensselaer Polytechnic Institute

in Partial Fulfillment of the

Requirements for the Degree of

DOCTOR OF PHILOSOPHY

Major Subject: COMPUTER SCIENCE

Examining Committee:

_____

Dr. Charles Stewart, Dissertation Adviser

_____

Dr. Barbara Cutler, Member

_____

Dr. Bülent Yener, Member

_____

Dr. Richard Radke, Member

Rensselaer Polytechnic Institute
Troy, New York

June 2017
(For Graduation August 2017)

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGMENT

Over the past seven years, I have devoted nearly all of my time and energy into learning the skills and developing the systems that have allowed me to make this contribution. I would've been unable to achieve this feat without the support and guidance of others.

First and foremost, I must thank my partner: Dana Cardona. Her love, support, and feedback has motivated and sustained me during my time as a graduate student. In our five years together I have found more stability and purpose than I thought was possible.

I am grateful to my parents for their ongoing encouragement and for fostering my curiosity. Thank you for all that you have done and continue to do.

Also, to my friend Lucas Cotterell: it was in a conversation with Luc that I decided that I could and would pursue a doctorate degree. His long-lasting friendship has helped me achieve this goal.

Next, I thank my labmates Peter Honig, Zach Jablons, Jason Parham, and Hendrik Weideman. Our discussions have been invaluable and are responsible for a significant portion of my understanding of computer vision, machine learning, and neural networks.

I graciously thank my advisor — Chuck Stewart — whose guidance helped me navigate my research, grow professionally and personally, and whose writing style has caused me to develop a fondness for em dashes.

To my committee members: Barb Cutler, Rich Rake, and Bülent Yener, whose suggestions, comments, and challenges successfully pushed me from my candidacy to the completion of my dissertation. I am thankful to them and to the other RPI professors who have taken the time to share their insight and discuss research problems with me.

I am immensely grateful to Bill and Naomi Hoffman for all they have done to open this path for me. I am filled with appreciation for them and the many others at Kitware who acted as teachers and mentors to me.

ABSTRACT

In this thesis we address the problem of identifying individual animals using images in the context of assisting an ecologist in performing a population census. We are motivated by events like the "Great Zebra Count" where thousands of images of zebras and giraffes were collected in Nairobi National Park over two days. By grouping all images that contain the same individual we can census these populations. This problem is challenging because images are collected outdoors and contain occlusion, lighting, and quality variations and because the animals exhibit viewpoint and pose variations.

Our first contribution is an algorithm that ranks a database of images by their similarity to a query. A manual reviewer inspects only the top few results for each query — significantly reducing the search space — and determines if the animals match. Using this algorithm alone, we analyzed the images from the Great Zebra Count and performed a population census. Our second contribution is a verification algorithm that determines the probability that two images are from the same animal, that they are not, or that there is not enough to decide. This algorithm is used with the ranking algorithm to re-rank results and automatically verify high confidence image pairs.

Our third contribution is a semi-automatic graph identification algorithm. The approach represents each image as a node in the graph and incrementally forms edges between nodes determined to the same animal. The ranking and verification algorithms are used to search for candidate edges and estimate their probability of matching. Based on these probabilities, edges are prioritized for review and placed in the graph when they are automatically verified or manually reviewed. Redundant connections are added to detect and recover from errors. A termination criterion determines when identification is finished. Using the graph algorithm we perform a population census on the scale of the Great Zebra Count using less than 25% of the manual reviews required by the original method.

1. INTRODUCTION

1.1 IMAGE-BASED IDENTIFICATION APPLIED TO POPULATION ECOLOGY

Population ecology relies on estimating the number of individual animals that inhabit an area [1]. Estimating a population size is done in two phases: data collection and analysis. Data are collected as sets of *sighting* and *resighting* observations. A sighting is the first observation of an individual, and a resighting is a subsequent observation of a previously sighted individual. The observed data are then analyzed using software such as "program MARK" [2], [3] or Wildbook that applies statistical models such as the Lincoln-Petersen index [4], Jolly-Seber model [5], [6], or other related models [7]–[9]. For an ecologist recording that an individual has been observed is simple, but determining if that observation is a sighting or a resighting can be challenging. This requires the ecologist to identify the individual by comparing against all other observations in the data set.

Current methods to estimate a population size are limited by the data collection phase [10], [11]. The statistical population models require an observation sample size that grows with the size of the population being studied [4]. As the number of observations increases so does the difficulty of determining identity. Thus, the scope of a population study is limited by the number of raw observations that can be made, and by the rate of determining the individual identity within a set of observations. Overcoming these limitations is of particular importance to wildlife preservation because population statistics are necessary to guide conservation decisions [12].

Consider images as a source of sight-resight observations. There are numerous advantages. Many observations can be made rapidly and simultaneously, due to the simplicity and availability of cameras. Recording an observation is as cheap and simple as taking a picture. Camera traps can be employed for autonomous data collection. In a wildlife conservancy or national park, observations can be crowd-sourced by gathering images from safari tourists and citizen scientists. Images can be accumulated and stored in a large dynamic dataset of observations that could grow by thousands of images each

day. However, the challenge of identifying the individuals in the images remains. Manual methods are infeasible due to the rapid rate at which images can be collected. Therefore, we must turn towards computer vision based methods.

This thesis develops the foundation of the image analysis component of the "Image Based Ecological Information System" (IBEIS). The purpose of this system is to gain ecological insight from images using computer vision. We focus on estimating the size of a population of animals as just one example of ecological insight that might be gained from images. Thus, we come to the core problem addressed in this thesis: image-based identification of individual animals.

1.2 CHALLENGES OF ANIMAL IDENTIFICATION

In animal identification we are given a database of images. This database may initially be empty. Each image is cropped to a bounding box around an animal of interest and labeled with that animal's identity. For a new query image, the goal is to determine if any other images of the individual are in the database. If the query is matched, it is added to the database as a resighting of that individual. If the query is not matched, then it is added as a new individual.

In this work we focus on identifying individuals of species with distinguishing textures. Examples include zebras, giraffes, humpback whales, lionfish, nautiluses, hyenas, whale sharks, wildebeest, wild dogs, jaguars, cheetahs, leopards, frogs, toads, snails, and seals. The primary species that we will consider in this thesis are plains and Grévy's zebras, but we will maintain a secondary focus on Masai giraffes and humpback whales. The difficulty of animal identification depends on the distinctiveness of the visual patterns that distinguish an individual from others of its species. In addition, the images we identify are collected "in the wild" and therefore contain occlusion, distracting features, variations in viewpoint and image quality.

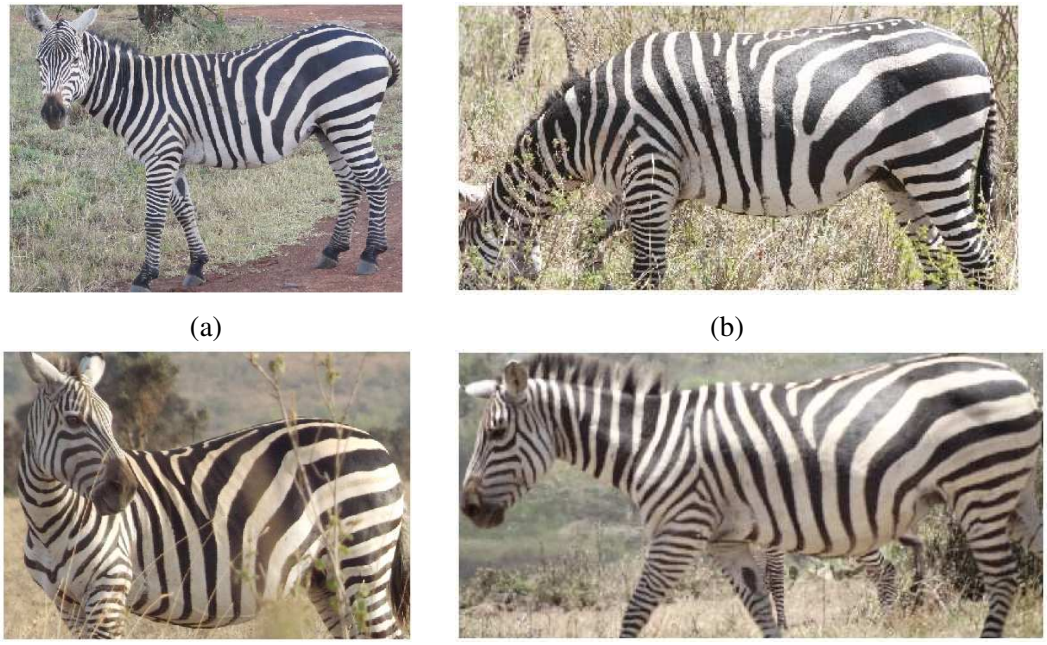
This section will present several examples to illustrate the challenges faced in animal identification. The discussion will begin with the challenges posed by the three primary species. Then problems common to all species will be described. These will be illustrated using plains zebras because they are the most challenging species considered in this thesis.

1.2.1 Distinguishing textures of each species

The plains zebra — shown in Figure 1.2 — is challenging to visually identify because individuals have relatively few distinguishing texture features. For most plains zebras, the majority of distinctive information lies in a small area on the front shoulder. Figure 1.4 illustrates that the patterns that distinguish two individuals can be subtle, even when the features are clearly visible. The matching difficulty greatly increases when features are partially occluded, the viewpoint changes, or the image quality is poor.

In contrast, Masai giraffes and Grévy's zebras, shown in Figure 1.6 and Figure 1.8 respectively, have an abundance of distinctive features. Distinctive textures that are unique to each individual are spread across the entire body of a Masai giraffe. For a Grévy's zebra there is a high density of distinguishing information above both front and back legs, as well as a moderate density of distinctive textures along the side of the body. The high density of distinctive textures in Masai giraffes and Grévy's zebras increases the likelihood that the same distinctive features can be seen from different viewpoints. Even so, the problem is still difficult due to "in the wild" conditions such animal pose, occlusion, and image quality.

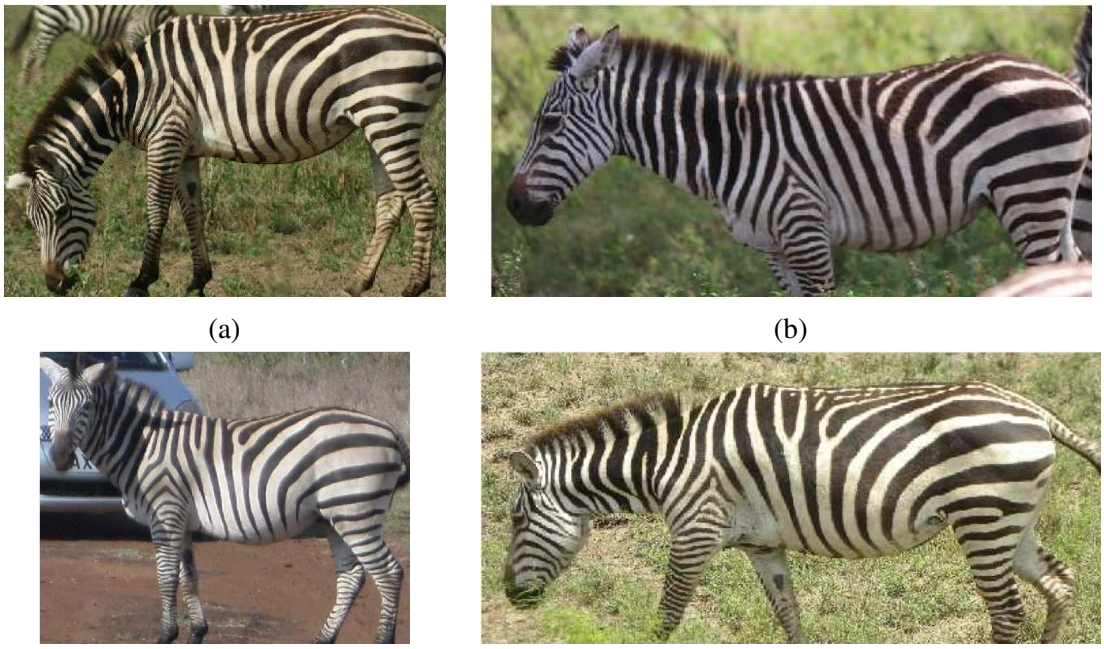
There are some species, like Humpback whales, where some individuals may contain distinguishing textures while others may lack them entirely. This means that only a subset of humpback whales will be able to be identified with the texture based techniques that we will consider in this thesis. However, other cues — like the shape of the notches along the trailing edge of the fluke — can be used to distinguish between different individuals. The work of Weideman and Jablons [13] addresses identifying humpback whales using trailing edge shape features. The example in Figure 1.10 illustrates individual humpback whales with and without distinctive textures.



(a) (b)

(c) (d)

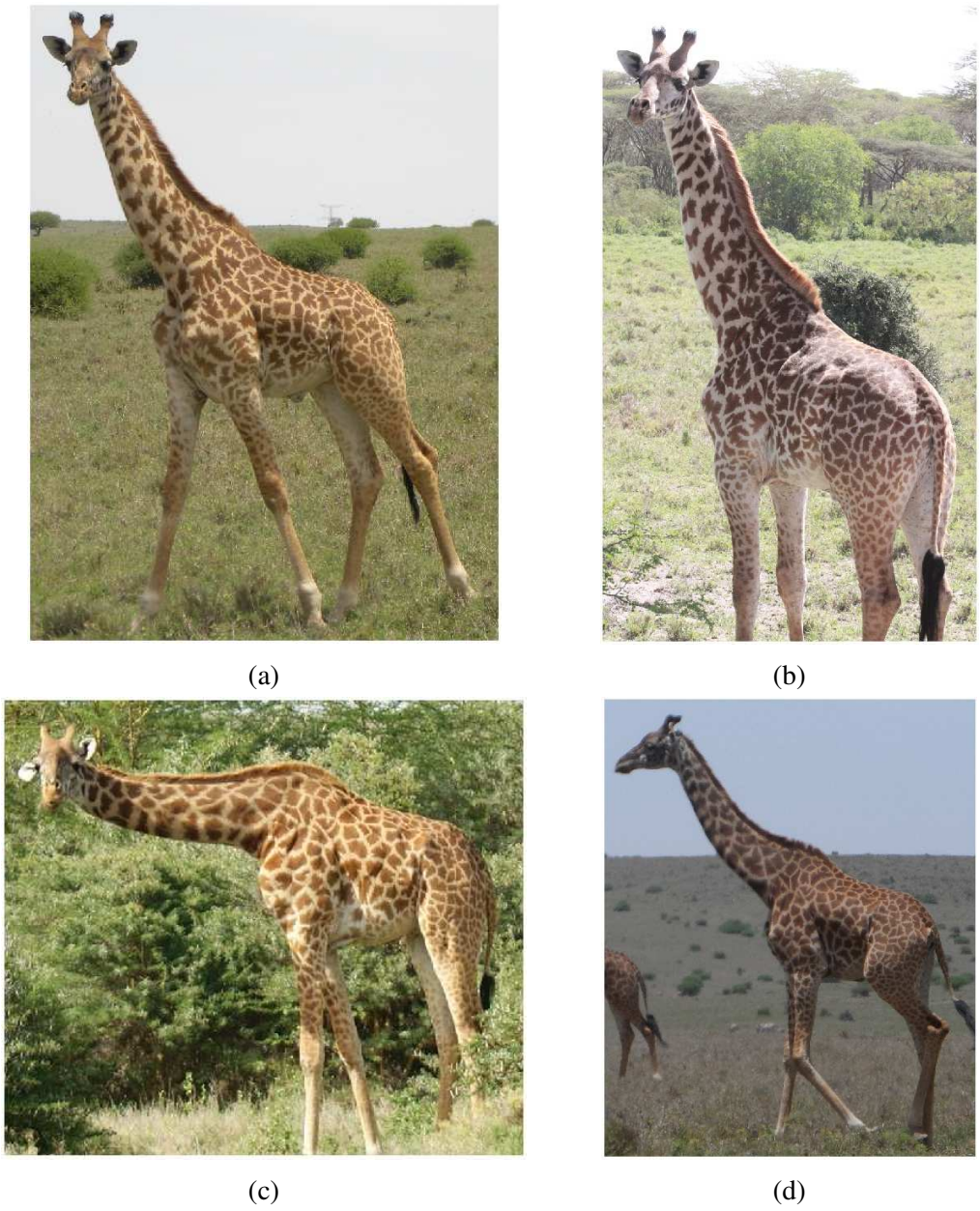
Figure 1.2: **Distinguishing features for plains zebras** For a plains zebra, the most distinguishing features tend to be located on the upper shoulder. Other distinguishing features are typically be found on the face and side. Image pairs (1.1a and 1.1b) and (1.1c and 1.1d) depict the same individual. These photos were taken during the GZC [14].



(a) (b)

(c) (d)

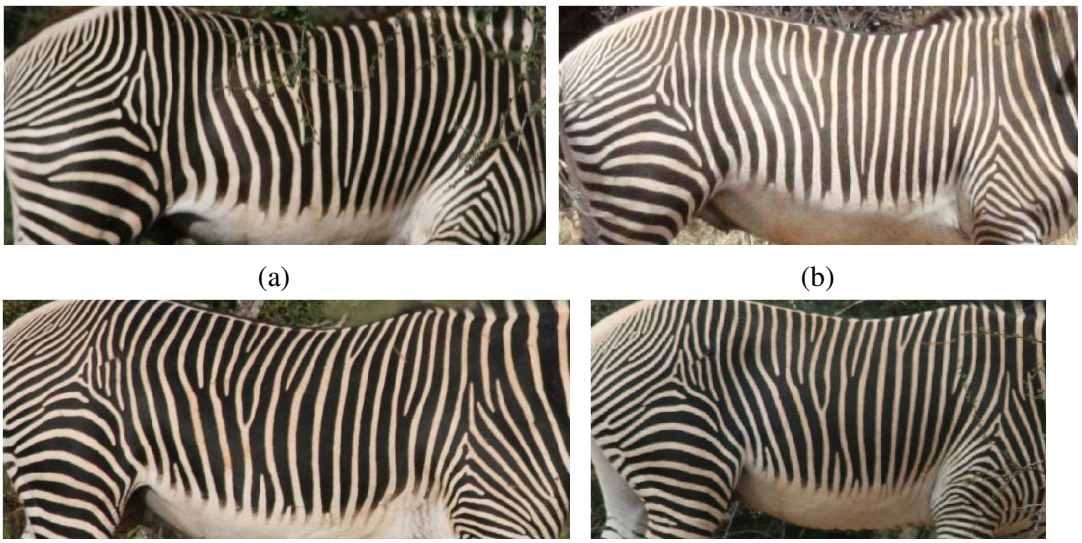
Figure 1.4: **Visually similar plains zebras** Different plains zebras sometimes have visual similarities that can be difficult to distinguish. There are three individuals in these four images. The images in (1.3a and 1.3d) depict the same individual. Dissimilarities can be seen on the lower thigh of images (1.3c and 1.3d), as well as on the front shoulder of images (1.3a and 1.3b).



(a) (b)

(c) (d)

Figure 1.6: **Distinguishing features for Masai giraffes** Masai giraffes have an abundance of features distinctive to each individual. There are two individuals seen in images pairs (1.5a and 1.5b) and (1.5c and 1.5d). Note that the numerous features make it initially difficult for a human to match giraffes. In contrast, this is easier for algorithms. These photos were taken during the GZC [14].



(a) (b)

(c) (d)

Figure 1.8: **Distinguishing features for Grévy's zebras** A Grévy's zebra's most distinctive features are above the front and rear legs. Useful, but less distinctive information can be seen on the side of the body. Image pairs (1.7a and 1.7b) and (1.7c and 1.7d) depict the same individual.

(a) (b) (c)

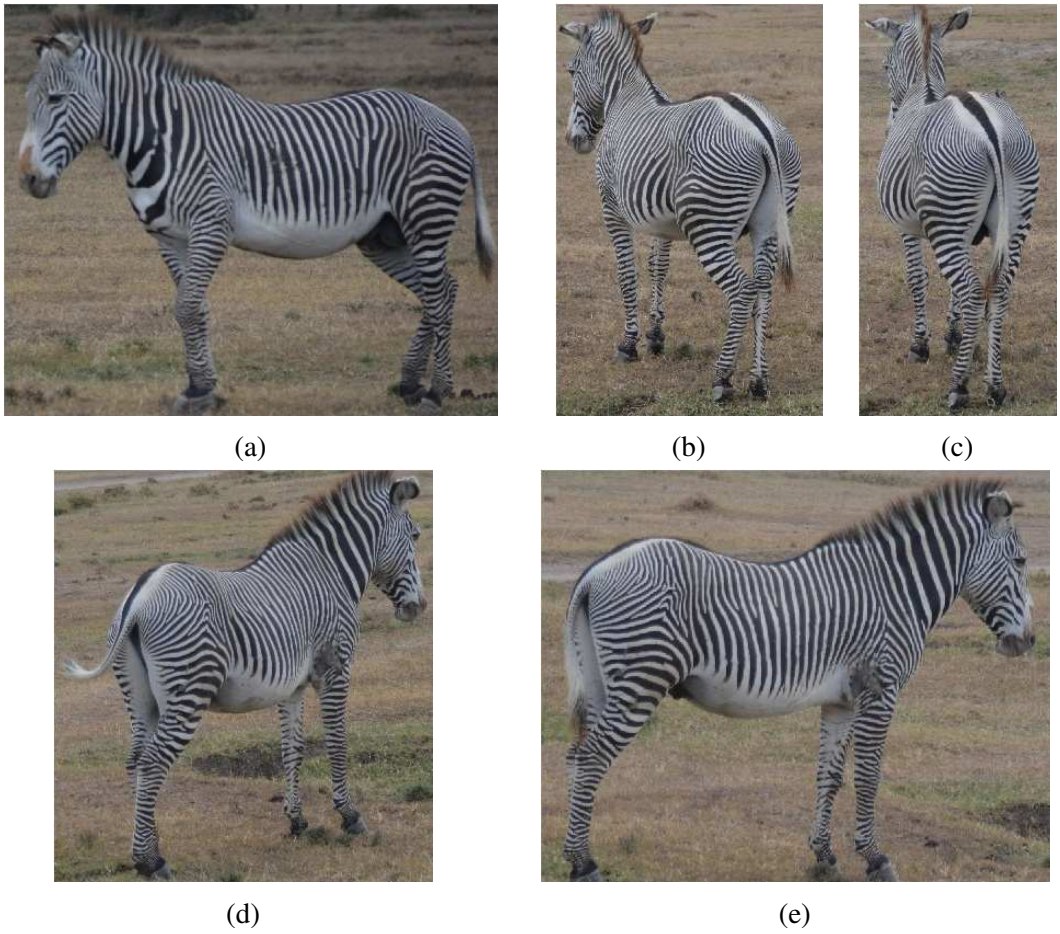
Figure 1.10: **Distinguishing features for humpback whales** A humpback whale can be identified by the texture patterns on the fluke or using the shape of the notches along the edge of the fluke. Note that some humpbacks (like the on seen in 1.9b) do not have any texture patterns on their fluke. The pair of images (1.9a and 1.9c) depict the same individual. These images were collected from FlukeBook[15].

1.2.2 Viewpoint and pose

One of the most difficult challenges faced in the animal identification problem is viewpoint. Animals are seen in a variety of poses and viewpoints, which can cause distinctive features to appear distorted. The patterns on the left and right sides of animals are almost always asymmetric. Therefore, matches can only be established using overlapping viewpoints and only if the viewpoints are distinctive. Some viewpoints, such as the backs of plains zebras, lack distinguishing information as shown in Figure 1.12. The effect of pose and viewpoint variation can be seen in Figure 1.14 and Figure 1.16.

(a) (b) (c) (d)

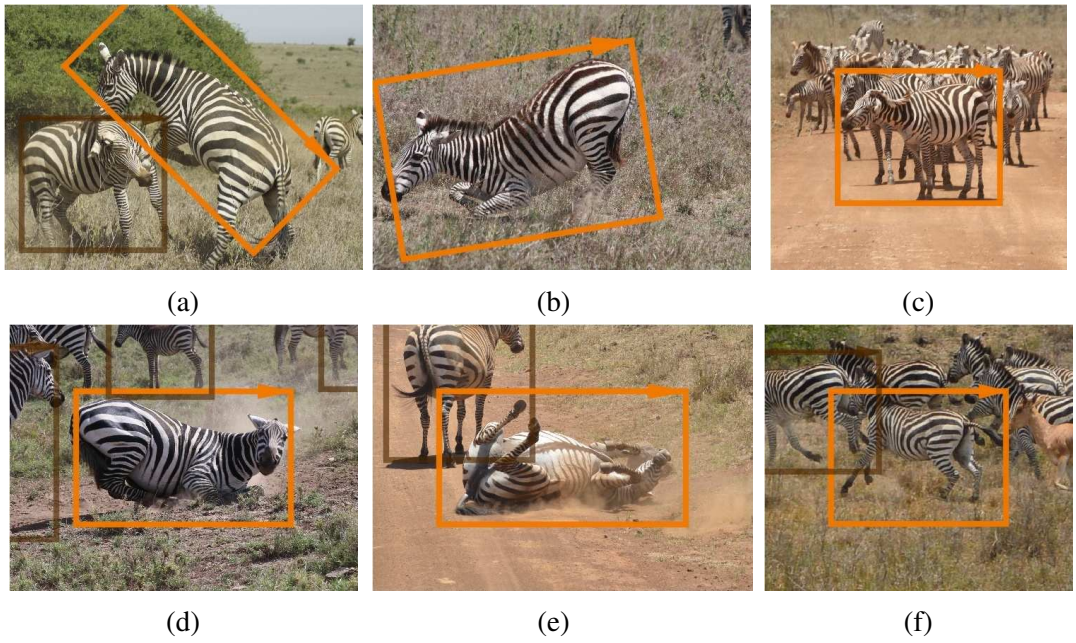
Figure 1.12: **Back viewpoints of plains zebras** The backs of plains zebras have very little distinguishing information. All the above images are different individuals. These photos were taken during the GZC [14].



(a) (b) (c)

(d) (e)

Figure 1.14: **Examples of viewpoint variations** This shows the viewpoint variations of an individual Grévy's zebra. It would not be possible to match image (1.13a and 1.13e) without information from images showing intermediate views. These photos were taken directly by our team.



(a) (b) (c)

(d) (e) (f)

Figure 1.16: **Examples of challenging pose variations** Animals can appear in a wide variety of poses. To clearly see the different poses, the images are shown with surrounding context. During identification the image is cropped to the bounding box shown around each animal. These photos were taken during the GZC [14].

1.2.3 Occluders and distractors

Because images of animals are often taken "in the wild", other objects in the image can act as *occluders* or *distractors*. Objects such as grass, bushes, trees or other animals, can act as occluders by partially obscuring the features that distinguish one individual from another. The appearance of the other animals nearby can be distracting because features from these animals will match different animals in the database. These *distractors* may also be from non-animal features when multiple pictures are taken against the same background as animals move through the same field of view. Several examples of occlusions and distractors are illustrated in Figure 1.18.

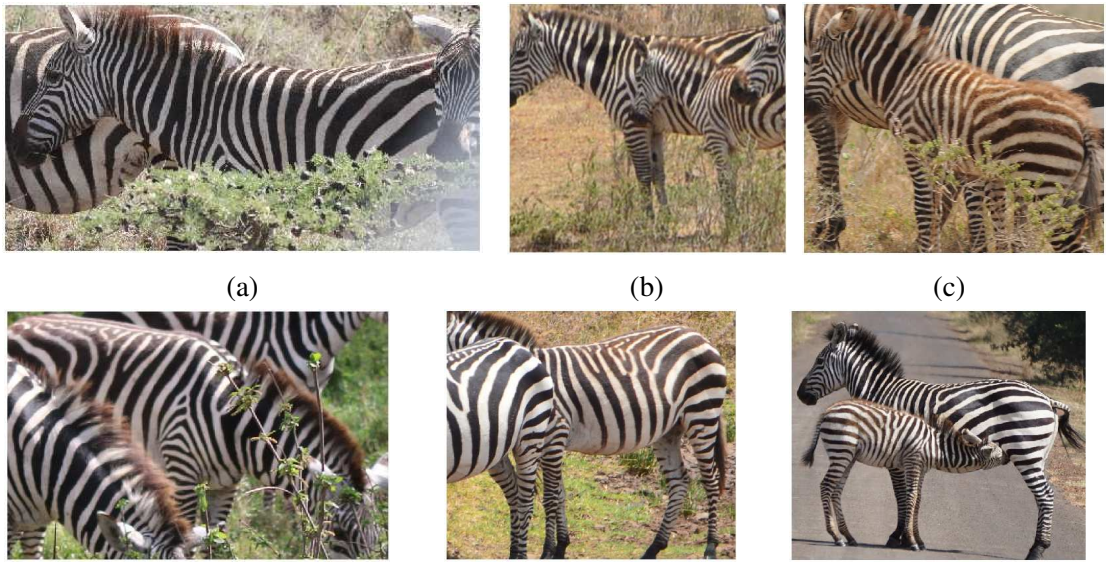
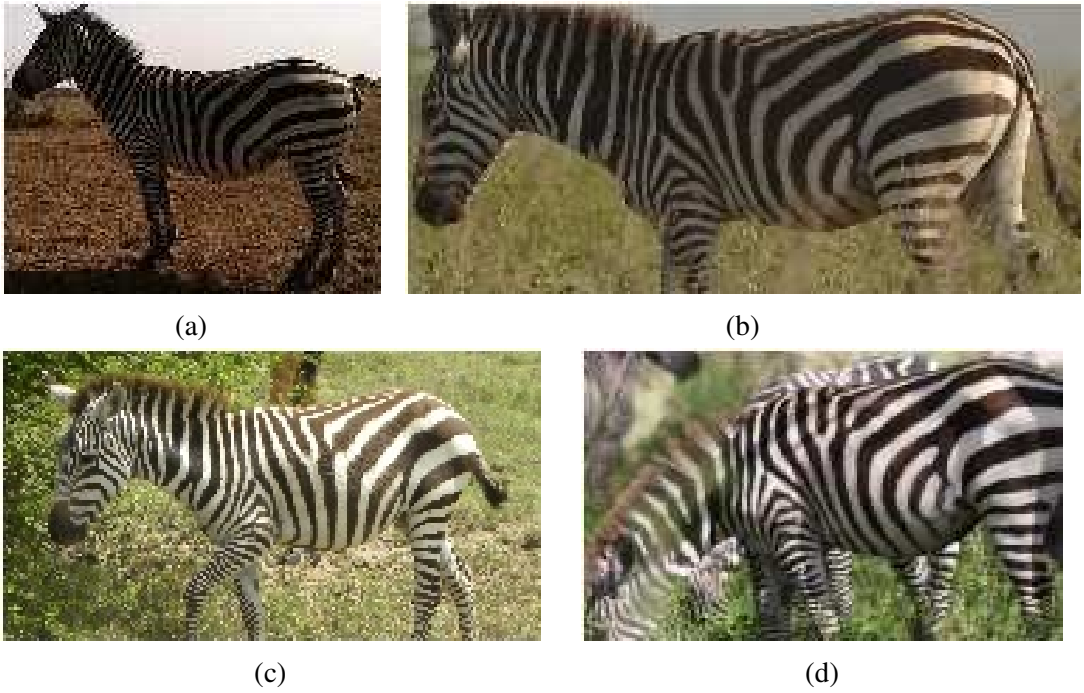


Figure 1.18: **Examples of occlusion and distractors** Occlusions can obfuscate or remove distinctive feature entirely. Secondary animals can introduce new distinctive features that do not belong to the primary animal. Images like this can cause other images of the secondary animal to the primary animal. These photos were taken during the GZC [14].

1.2.4 Image quality

Image quality is influenced by lighting, shadows, the camera used, image resolution, and the size of the animal in the image. Outdoor images will naturally have large variations in illumination. Different cameras can produce visual differences between images of an object. Images taken out of focus, from far away, or with a non-steady camera

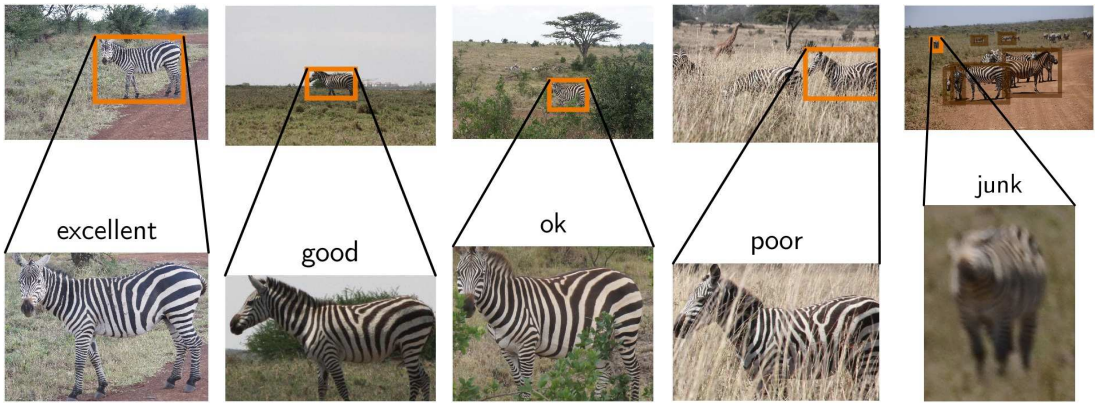
can cause animals to appear blurred. The effects of outdoor shadow and illumination are illustrated in Figure 1.20. Figure 1.22 illustrates five categories of image quality that will be described later in Section 1.3.2.3.



(a) (b)

(c) (d)

Figure 1.20: **Examples of different lighting conditions** Shadow and illumination can cause variations in the underlying image intensity and gradients. This can make it more difficult to localize repeatable keypoints and describe the underlying texture patterns. These photos were taken during the GZC [14].



(a) (b) (c) (d) (e)

Figure 1.22: **Examples of different image qualities** The bottom row shows the cropped images that correspond to the bounding boxes in the top row. Each column shows different qualities: (1.21a) an excellent quality image taken from a short distance, (1.21b) a good quality image with minor shadow and taken from a medium distance, (1.21c) an ok quality image due to minor occlusion, (1.21d) a poor quality image due to major occlusion, (1.21e) a junk quality image due to considerable blur. These photos were taken during the GZC [14].

1.2.5 Aging and injuries

The appearance of an individual changes over time due to aging and other factors including injuries. An example of the difference between a juvenile and adult zebra is shown in Figure 1.24. An example of how injuries can both remove distinctive features and add new ones is shown in Figure 1.26.

(a) (b)

Figure 1.24: **Examples of visual differences caused by age** The left and right images show the adult and juvenile appearance of the same individual. As an animal ages its appearance changes mainly in color and texture with some minor shape and scale differences. These photos were taken in Ol' Pejeta conservancy.

(a) (b)

Figure 1.26: **Examples of visual differences caused by injuries** Injuries can obscure features on an animal as well as creating new ones. The left image shows a wounded animal, and the right image shows an animal with a distinguishing scar. The left photo was taken during the GZC [14]. The right photo was taken in Ol' Pejeta conservancy.

1.3 THE GREAT ZEBRA COUNT

To further illustrate the problems addressed in this thesis, we consider the "Great Zebra Count" (GZC), held at Nairobi National Park on March 1st and 2nd, 2015 [14]. This event was designed with two purposes in mind: (1) to involve citizens in the scientific data collection effort, thereby increasing their interest in conservation, and (2) to determine the number of plains zebras and Masai giraffes in the park.

1.3.1 Data collection

Volunteer participants — each with his or her own camera — arrived by car at the park. Some cars had more than one photographer. Each car was assigned a route to drive

through the park. We attached a GPS dongle to each car to record time and location throughout the drive. Correlating this with the time stamp on each image (after adding a correction offset for each camera) allowed us to determine the geolocation of each image. Each photographer was given instructions guiding them toward taking quality images of the left sides of the animals they saw. When the cars returned — some after just an hour or two, others after the whole day — the images were copied from the cameras, a small sample of each photographer's images was immediately processed to illustrate what we would do with the data, and the entire set of images was stored for further processing. The result of this crowd-sourced collection event was a $48\,\mathrm{GB}$ dataset consisting of 9406 images.

1.3.2 Data processing

After the event, the entire collection of images was processed using a preliminary version of the system in order to generate the final count. The preliminary system followed the workflow of: (1) occurrence grouping, (2) animal detection, (3) viewpoint and quality labeling, (4) intra-occurrence matching, (5) master database identification, (6) consistency checks, and (7) population estimation. Here, we provide a brief overview of each step involved in the processing of the GZC image data, and then we will describe the challenges that arose.

1.3.2.1 Occurrence grouping

The images were first divided into *occurrences* — a standard term defined by the Darwin Core [16] to denote a collection of evidence (*e.g.* images) that an organism exists within defined location and time-frame. In the scope of this application, an occurrence is a cluster of images taken within a small window of time and space. Images are grouped into occurrences using the GPS and time data. Details are provided in Appendix A.

These computed occurrences are valuable measurements for multiple components of the IBEIS software. At its core an occurrence describes *when* a group of animals was seen and *where* that group was seen. From this starting point other algorithms can address questions like: *how many* animals there were, *who* an animal is, *who else* is an animal with, and *where else* have these animals been seen?

Furthermore, there are computational and algorithmic benefits to first grouping images into an occurrence. One benefit is that an occurrence can be used as a semantic processing unit to distribute manageable chunks of work to users of the system. Another is that occurrences can be used to improve the results of identification. Typically, there will be only a few individuals within an occurrence, and it is not uncommon for each individual to photographed multiple times and from multiple viewpoints. This redundancy in images will be exploited in Chapter 5.

1.3.2.2 Animal detection

Before matching begins each image is cropped to focus on a particular animal and remove background distractors. A detection algorithm localizes animals within the images. Each verified detection generates an *annotation* — a bounding box around a single animal in an image. An example illustrating detection of plains zebras is shown in Figure 1.28. In the GZC each detection was manually verified before becoming an annotation, but recent work introduces an automatic verification mechanism and reduces the need for complete manual review. The details of the detection algorithm are beyond the scope of this thesis, and are described in the work of Parham [17], [18].

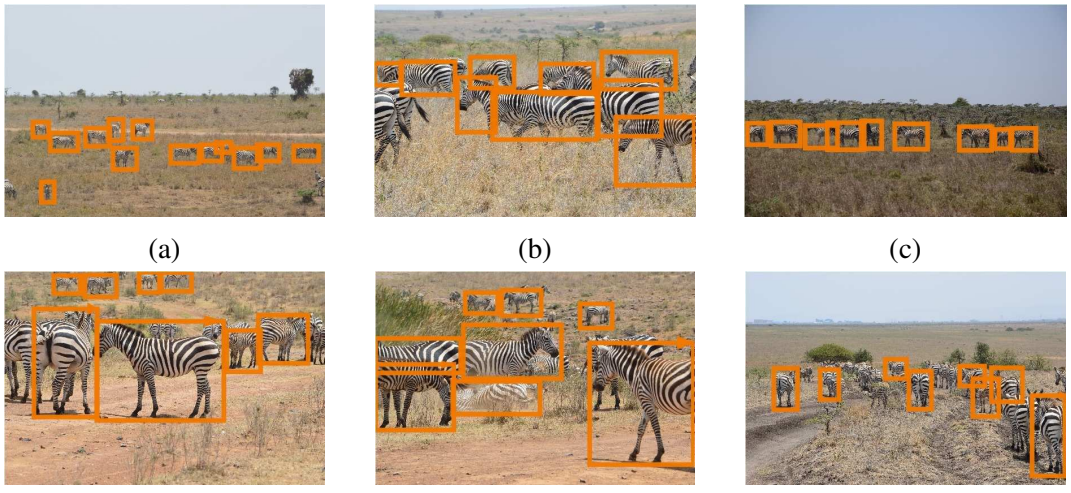


Figure 1.28: **Detection of plains zebras** These photos were taken during the GZC [14]. Detections were automatically suggested and manually verified before being accepted.

1.3.2.3 Viewpoint and quality labeling

When determining the number of animals in a population it is important to account for factors that can lead to over-counting. If two annotations of the same individual are not matched, then that individual will be counted twice. This could happen due to factors such as viewpoint and quality. For example, one annotation showing only the left side of an animal and another annotation showing only the right side the same animal cannot be matched because they are *incomparable*. The two annotations are comparable when they share regions with distinguishing patterns that can be put in correspondence. Viewpoint is the primary reason that two annotations are not comparable. However, other factors like image quality and heavy occlusion can corrupt distinguishing patterns rendering the annotation unidentifiable — not comparable with any other annotation. We must define what it means for two annotations to be comparable before we can estimate a population size.

Determining if an individual can be identified is analogous to the notion of a marked-individual [4]. For an annotation to be identifiable the patterns that can distinguish it from the rest of the population must be clear and visible, otherwise the annotation may not be able to find or be compared to potential matches. This means an annotation is only identifiable if (1) the image quality is high enough, and (2) it has a viewpoint that is comparable to all potential matches.

To address this challenge we label each annotation with 5 discrete quality labels and 8 discrete viewpoint labels. The quality labels we define are: *junk*, *poor*, *ok*, *good*, and *excellent*. The *junk* label is given to annotations that almost certainly will not be able to be identified, and *poor* labels are given to annotations that will likely be unidentifiable for a computer vision algorithm. The *good* and *excellent* labels are given to clear, well illuminated annotations with little to no occlusion with *excellent* being reserved for the best of the best. All other annotations are labeled as *ok*. The viewpoint labels we define are: *front*, *front-left*, *left*, *back-left*, *back*, *back-right*, *back*, and *front-right*. Note, that additional viewpoint labels like *up* and *down* may be necessary for animals such as lionfish or turtles. However, the 8 labels we use are sufficient for animals like zebras and giraffes because they are most commonly seen in upright positions.

In an effort to ensure that all annotations used in the GZC were comparable, we

did not include any annotation that had junk or poor qualities. We also did not include annotations not labeled with a left or frontleft viewpoint to account for limitations in the initial ranking algorithm. All labelings of viewpoint and quality were generated manually. Since then, we have trained viewpoint and quality classifiers using this manual data. Automatic detection of quality and viewpoint is discussed in the work of Parham [17].

1.3.2.4 Matching within each occurrence

Animals often have multiple redundant views within an occurrence, each of which can be the same, better, or complementary to other views. The images in Figure 1.30 illustrate redundant and complementary views of an individual in an occurrence. Merging all of an individual's views is a challenge, but also potentially an advantage as we can exploit redundancy to better handle missing features, subtle viewpoint changes, and occlusions.

We exploit this redundancy to gain the benefit of complementary views by matching all annotations within an occurrence in a process called *intra-occurrence matching*. In the GZC, each annotation was queried against all other annotations in its occurrence, returning a ranked list of candidate matches. The person running the software made the final decisions about which annotations match. Details about the ranking algorithm are given in Chapter 3.

The result of intra-occurrence matching is a set of *encounters*. An encounter is a group of annotations that were matched within an occurrence. Each encounter is either (1) the first sighting an individual or a (2) resighting. The task now becomes to determine which of these is the case by identifying each encounter against a master database.

1.3.2.5 Matching against the master database

To determine if an encounter is a new sighting or a resighting of an individual, it is matched against the master database in a process called *master database matching*. Before matching begins the master database is prepared for search. For each name in the master database a subset of *exemplar* annotations is chosen to represent the appearance of that individual. The exemplars are indexed using a search data structure.

After the master database has been prepared, the ranking algorithm is able to issue a subset of the encounter's annotations as a query. The result is a ranked list of exemplars

(a) (b) (c)

Figure 1.30: **Multiple images in an occurrence** These images were taken within an occurrence and demonstrate redundant and complementary features. Features on the shoulders are somewhat redundant in images (1.29a) to (1.29c) because they are all under approximately constant illumination and are seen from the same angle. Images (1.29a and 1.29c) have complementary features because the viewpoint of the animal has shifted slightly. These photos were taken during the GZC [14].

that are visually similar to the encounter. The top exemplars in the ranked list are used as candidate matches. Then, the candidate matches are reviewed, and the encounter is either merged into an existing master-name or added to the master database as a new master-name.

1.3.2.6 Consistency checks

When merging encounters into the master database it is possible that mistakes were made. Two error cases commonly occur.

(1) A *split case* occurs when a set of annotations from two or more different animals is incorrectly labeled with the same name. The main cause of this error is when distracting features are matched causing the annotations to appear visually similar.

(2) A *merge case* occurs when two sets of annotations from the same animal are incorrectly labeled with different names. This is caused by an algorithm or human error where a query encounter was not correctly matched to the database exemplars.

These errors usually occur because the query and database annotations have a low degree of *comparability* (*e.g.* differences in viewpoint or low quality). Of course, if no visual overlap exists between the two sets — such as one set exclusively from the left side and another exclusively from the right — nothing can be done. This is why the animal must be seen from a predetermined view in order to be counted. In the GZC this is the left side.

In the GZC suspect individuals were flagged for split checks using various heuristics

such as the number of annotations in the name or the apparent speed of the animal's movement as GPS and time data. To check a flagged individual we used the ranking algorithm to search for pairs of annotations with low matching scores that belong to the flagged name. Low similarity between two annotations within a name suggested that an error had occurred. These low scoring results were then manually reviewed. When breaking apart split cases, care was taken to account for the fact that right and left images should not match. Likewise, care was taken to ensure that an intermediate annotation linking two disjoint annotations has enough information to establish the link.

Merge checks issue all exemplars as queries against all other exemplars. High similarity between two different names suggested that a match was missed. These high scoring results were manually reviewed. More sophisticated error detection and recovery will be discussed in Section 5.5.

1.3.2.7 Population estimation

The final step for the GZC workflow was to estimate the number of animals in the park. Using the identification algorithm we defined which annotations were sightings and which were resightings. Because we were using a preliminary version of the system we were conservative in defining when an animal was sighted by only using the left and frontleft annotations with quality labels of ok, good, or excellent. Each individual that met these criteria was counted as a sighting. If a sighted individual had an annotation from both days, then we counted that individual as resighted.

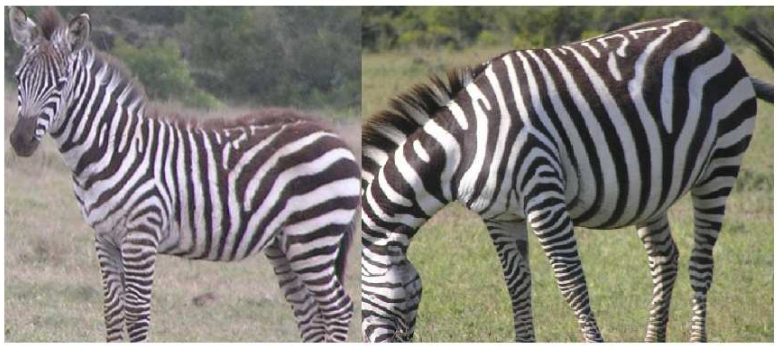
1.3.3 Processing challenges

Our experience with the Great Zebra Count has highlighted a number of challenges that must be addressed if this system is to be applied in future events. These challenges include the number of manual reviews required, the detection of and recovery from manual errors, and the overall lack of a systematic identification framework.

Perhaps the greatest challenge faced during the GZC was the considerable amount of time that was required to manually verify identification results. It can take several seconds to manually verify if a pair of annotations is a correct match even if the results are presented in a ranked list. This task is illustrated in Figure 1.32. Requiring the manual

verification of each result is untenable for a system that accepts thousands of new images a day. The lack of a systematic approach for identification meant that whenever two annotations were matched, the name labels of all annotations of those names were changed. This made it difficult to tease apart errors when they occurred. Furthermore, manual errors (likely caused by fatigue from the large number of manual reviews) resulted in numerous identification errors cases that were not able to be detected and resolved until the end of the process. Reviews of results were also done in order of matching scores regardless of previous decisions, causing the manual reviewer to inefficiently review redundant results between the same individual. Additionally, no stopping criterion for reviews was defined resulting in an ad hoc approach to determining when all matches were found.

Motivated by these observations we seek to develop a semi-automatic approach to animal identification. This approach will should be governed by a system that reduces the number of manual reviews and is able to detect and recover from errors, and determine when to stop searching for new matches.



(a) Rank 1

(b) Rank 2

(c) Rank 3

Figure 1.32: **Examples of top ranked matches** Each pair is a one-vs-one comparison. All the left images are the same query image. Each image on the right is a candidate match. The match in (1.31a) is correct and the other matches are incorrect. However, a ranked list may contain more than one correct match. These photos were taken during the GZC [14].

1.4 APPROACH

The problem addressed in this thesis is to identify individual animals "in the wild" and to count the individuals in a population. We are given a set of images containing annotations of the same species. The images are collected in an uncontrolled environment and likely contain imaging challenges such as occlusion, distracting features, viewpoint variations, pose variations, and quality variations. Furthermore, the images may be collected either over many years or over just a few days as in the GZC. Each annotation is labeled with time, GPS, quality, and viewpoint. We may also be given an initial partial name labeling of the annotations — *e.g*. in the case where we identify a new set of annotations against a previously identified set — but this need not be the case. We want to label each annotation with a *name* that uniquely identifies the individual. In other words, our task is to label all annotations from the same individual with the same name and give annotations from different individuals different names. After this is complete, the resulting database will contain the information needed to estimate the size of the population using techniques from sight-resight statistics.

The first step of the identification process is a ranking algorithm. The inputs to the algorithm are a single query annotation and a set of database annotations. Sparse patch-based features are localized in all annotations, and a descriptor vector is extracted for each feature. The descriptors of the database annotations are indexed for fast nearest neighbor search. We then find a set of matches in the database for each descriptor in the query annotation. The matches are scored based on visual similarity, distinctiveness within the database, and likelihood of belonging to the foreground. Matches are combined across multiple exemplar annotations to produce a matching score for each name in the database, resulting in a ranked list of results for each query.

We then extend the ranking algorithm by developing a classifier able to automatically review its results. First, we construct a pairwise feature that captures relationships between two annotations using local feature correspondence and global properties such as time and GPS. Then, we learn a classifier to predict if a pair of annotations — *i.e*. a result in the ranked list — is correct or incorrect.

In the final part of our approach, we place the problem of animal identification in a graph framework able to systematically guide the identification process. This is done

by placing each annotation in a graph as a vertex and placing labeled edges between annotations to represent how they are related. Using the graph framework we will be able to detect and recover from errors by taking advantage of multiple images seen of each individual.

We evaluate the ranking, verification, and graph identification algorithm by performing experiments on two main databases of plains zebras and Grévy's zebras. Some additional experiments are also performed on databases of Masai giraffes and humpback whales. First, the ranking experiments test the algorithm's ability to find potential matches of an individual animal over large periods of time, different viewpoints, different sized databases, and different numbers of exemplars. Then, the verification experiments will test the extent to which the correct results from the ranking algorithm can be separated from the incorrect results using our learned classifier. Finally, the graph identification experiments will demonstrate the algorithm's ability to reduce the number of required manual reviews and recover from errors. We determine the configuration of each algorithm that works best for identifying each species.

1.5 ORGANIZATION

This thesis is organized as follows: Chapter 2 describes related work. The focus is on the details of techniques used in the system, while an overview is given for those which are indirectly related. Chapter 3 describes the ranking algorithm for identifying individual animals, one annotation at a time, against a database of exemplars. This chapter includes an experimental evaluation of the ranking algorithm. This is the algorithm that was used in the GZC. Chapter 4 addresses the problem of semi-automatic verification of results from the ranking algorithm. Chapter 5 combines the ranking and verification algorithm into a semi-automatic framework that detects and corrects errors while reducing the number of manual reviews. Chapter 6 concludes this thesis and summarizes its contributions.

2. RELATED WORK

To address the individual animal identification problem we draw upon related research in feature detection [19]–[21], feature description [22]–[27], approximate nearest neighbor search [28], [29], instance recognition [30]–[36], face verification [37]–[41] fine-grained recognition [42]–[44], category recognition [45]–[49], and convolutional neural networks [26], [27], [50]–[52].

At a high level, the main questions addressed by the aforementioned research can be summarized as: How should image features be detected? How should detected image features be represented? How much invariance should features have? Should image features be quantized? How should image features be matched? How should feature matches be aggregated? How should feature matches be scored? How should all of this be done accurately? How should all of this be done efficiently? Answers to these questions address many of the challenges to animal identification previously introduced in Section 1.2.

This chapter summarizes literature relevant to addressing these questions in the context of animal identification. The outline of this chapter is as follows: Section 2.1 will discuss feature detection. Section 2.2 will discuss feature description. Section 2.3 will discuss approximate nearest neighbor algorithms. Sections 2.4 to 2.6, will discuss approaches to recognition. Section 2.7 will discuss convolutional neural networks.

2.1 IMAGE FEATURE DETECTION

Before an image can be analyzed, it must be broken down into smaller components. An image's visual appearance can be captured using a combination of local image patterns — *patch-based features*. The most informative patch-based features are typically centered on simple image structures such as junctions, corners, edges, and blobs [20]. If these features can be reliably detected, localized, and described then image matching can be posed as a problem in matching sets of features. This section describes work related to detecting features in an image, and Section 2.2 will discuss how detected features are then described.

The region where a feature is detected is called a *keypoint*. The simplest definition of a keypoint is just an xy-location in an image. However, images contain information at multiple scales; therefore a keypoint is typically associated with a scale. The scale of a keypoint is a non-negative real number that defines the level of detail at which to interpret the underlying image information. A keypoint with a scale can be thought of as a circular region with a radius that is the scale multiplied by some constant (*e.g.* $3\sqrt{3}$ is the constant used to determine a keypoint's radius in [21]). To account for changes in viewpoint and pose, it is also common to augment features with an orientation and shape. Adding these properties is said to add invariance to the feature. Invariant features can provide similar descriptions of the same semantic image region under different viewing conditions. However, adding invariance can cause features to lose distinguishing information [19]–[22].

Many detectors have been developed to detect patch-based feature keypoints [19], [20]. Algorithms such as Harris, SUSAN, and FAST detect corners [53]–[56]. Blobs and corners can be detected with Hessian [57], [58] or difference of Gaussians [22], [59] detectors. There are also region-based detectors: maximally stable extremal regions [60], saliency based methods [61] and superpixel-based methods [62], [63]. Some applications choose to skip keypoint detection and use a uniform grid of dense features [64]–[66]. Other applications, such as face recognition, use specialized keypoint detectors [39], [67]. There currently exists no principled method for selecting the appropriate feature detector. Different feature detectors perform differently given the application [20].

This section describes the representation of an image over multiple scales, the detection of features to sub-pixel and sub-scale accuracy, and the adaption of features to specify orientation and shape. We focus on the Hessian-based keypoint because it has been experimentally shown to be a reliable choice for instance recognition [20].

2.1.1 Scale-space

Scale-space theory describes image features as existing at multiple scales [68]. The same point on an object seen close up appears quite different compared to when it is at a distance. For example, a zebra in the distance may appear to have two stripes that are connected, but when the animal appears closer it becomes clear that the stripes are actually disconnected. Multi-scale detection is formalized by the theory of scale-space,

which parameterizes a continuous signal, f, with a scale, σ. The original signal is said to exist at scale 0. Convolving the original signal with a Gaussian kernel produces coarser scales.

Let f be a continuous 2-dimensional signal that defines an image. Let vector $\mathbf{x} = \begin{bmatrix} x \\ y \end{bmatrix}$ be a location in the image. The function $g(\sigma)$ is the isotropic 2D Gaussian kernel. The scale-space representation of a continuous image (for any non-zero scale) takes the form: $I(\mathbf{x}, \sigma) = g(\sigma) * f(\mathbf{x})$, where $*$ is the convolution operator. However, we do not have access to a continuous representation of an image. Therefore, in practice, the continuous Gaussian kernel is replaced with the discrete Gaussian kernel. This can be efficiently implemented as a discrete convolution with the 1-dimensional discrete Gaussian kernel in the x-direction and then in the y-direction, because the discrete Gaussian kernel is separable in orthogonal directions [68]. Using the definition of an image at a single scale the next step is to represent an image at multiple scales.

2.1.1.1 Gaussian pyramid

The discrete scale-space representation of an image is efficiently implemented using a Gaussian pyramid. A scale-space pyramid consists of L levels. Each level covers an octave. Starting from the base of each level with scale parameter σ the next octave is reached when σ doubles. There are s intervals represented within each octave. A Gaussian pyramid is illustrated in Figure 2.1.

The pyramid's base, $I(\mathbf{x}, 1) = g(1) * I_{\texttt{raw}}(\mathbf{x})$ is the $\ell = 0^{\text{th}}$ level of the pyramid, and is computed by blurring the original image (sometimes with small initial blurring) with $\sigma = 1$. Subsequent levels of the pyramid are produced by doubling sigma, thus the ℓ^{th} level of the pyramid is $I(\mathbf{x}, 2^\ell)$.

A property of discrete scale-space is that after appropriate smoothing downsampling the image by half is equivalent to doubling sigma. Let $I_\ell(\mathbf{x})$ denote the raw image downsampled by a factor of 2^ℓ using Lanczos resampling. Now, each level of the pyramid can be written as $I(\mathbf{x}, 2^\ell) = g(1) * I_\ell(\mathbf{x})$. Given the raw image at level ℓ, the scale corresponding to σ can be written as a relative scale $\sigma_\ell = \sigma / 2^\ell$. Thus, a discrete image at any scale can be efficiently computed as:

$$I(\mathbf{x}, \sigma) = g(\sigma_\ell) * I_\ell(\mathbf{x}) \tag{2.1}$$

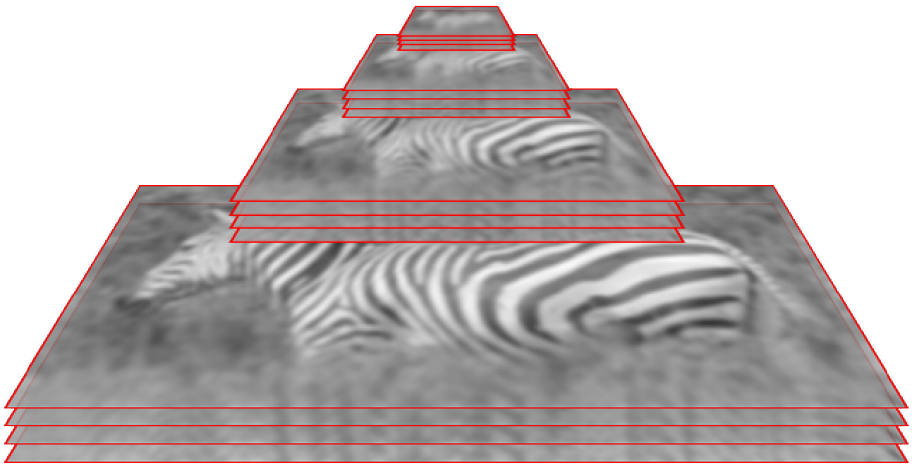


Figure 2.1: **A scale space pyramid** A Gaussian pyramid is used as a scale space representation of an image. A property of scale space is that doubling the scale is equivalent to downsampling the image by half. The set of images of a specific size correspond to an octave. The images within each octave are the intervals.

Discrete convolution is applied using a window of size $\lfloor 6\sigma_\ell + 1 \rfloor + (1 - (\lfloor 6\sigma_\ell + 1 \rfloor \mod 2))$. Interpolation between discrete values of $\mathbf{x}$ is used to sample intensity at sub-pixel accuracy.

A scale between two levels of the pyramid is called an interval. Typically, s intervals — with relative scales $2^{0/s}, 2^{1/s}, \ldots 2^{s/s}$ — are computed to represent the octave between level ℓ and $\ell+1$. If differences between scales are needed, then the scales $2^{-1/s}$ and $2^{1+1/s}$ are also computed [22].

2.1.2 Hessian keypoint detection

Hessian-based keypoint detection searches for extrema of the Hessian operator in both space and scale [57],[58]. The Hessian detector can qualitatively be viewed as a blob detector, but it also detects corners which may appear as blobs in scale-space [20]. The Hessian keypoint detector will compute a response value for each point in scale space indicating how blob-like each pixel is. The extrema of this response defines a set of Hessian keypoints. Post processing removes non-robust keypoints and localizes all other keypoints to sub-pixel and sub-scale accuracy.

2.1.2.1 Hessian response

Let subscripts denote the partial derivatives of the image intensity (*e.g.* I_x is the first x derivative, I_{xx} is the second x derivative, and I_{xy} is the first derivative in both x and y). The Hessian is a matrix of second order partial derivatives and is defined at each point in scale space.

$$\mathbf{H}(\mathbf{x}, \sigma) = \begin{bmatrix} I_{xx}(\mathbf{x}, \sigma) & I_{xy}(\mathbf{x}, \sigma) \\ I_{xy}(\mathbf{x}, \sigma) & I_{yy}(\mathbf{x}, \sigma) \end{bmatrix} \tag{2.2}$$

The initial response of the detector at each point is the determinant of the Hessian matrix. This response is computed for level and every pixel in the scale-space pyramid. At coarser scales the Hessian response weakens, so to ensure that responses between scales are comparable, the initial response is scale normalized by multiplying with σ^2. (See [69] for more details about the choice of this normalization factor.) The extrema of this space defines a set of candidate keypoints, $\mathcal{P}'$.

$$\mathcal{P}' = \underset{\mathbf{x},\sigma}{\operatorname{argextrema}} \left(\sigma^2 \operatorname{Det} \left(\mathbf{H}(\mathbf{x}, \sigma) \right) \right) \tag{2.3}$$

A point in this 3D space is a maxima/minima if its scale normalized value is greater/less than the scale normalized values of all its neighbors in the pyramid — *i.e.* the 8 neighbors in its current interval, its 9 neighbors in the next interval, and its 9 neighbors in the previous interval.

2.1.2.2 Edge filtering

Edge responses are not robust — *i.e.* the same point can not be localized reliably in two views of the same scene — due to their elongated nature. Because of this, the extrema that appear too edge-like are filtered using a threshold $t_{\texttt{edge}}$ which is compared to the ratio of the Hessian's squared trace and the determinant.

$$\mathcal{P} = \left\{ (\mathbf{x}, \sigma) \in \mathcal{P}' \;\middle|\; \frac{\operatorname{Tr} \left(\mathbf{H}(\mathbf{x}, \sigma) \right)^2}{\operatorname{Det} \left(\mathbf{H}(\mathbf{x}, \sigma) \right)} > t_{\texttt{edge}} \right\} \tag{2.4}$$

2.1.2.3 Sub-pixel and sub-scale localization

To compensate for the discrete nature of pixel images, each keypoint detection is localized to sub-pixel and sub-scale accuracy. The importance of feature localization is demonstrated in [70], where descriptors were computed on the normalized vectors of patch gradients using only principal component analysis (PCA) [71]. Despite the simplicity of the descriptors the authors were still able to effectively match two images due to the robust localization of the features.

Sub-pixel and sub-scale localization transforms a keypoint $\mathbf{p}_0$ into $\mathbf{p}^*$ using an iterative process. At each iteration i, a 2nd order Taylor expansion, centered at $\mathbf{p}_i = (\mathbf{x}_i, \sigma_i)$, approximates the scale normalized Hessian response: $T_i(\mathbf{x}, \sigma) \approx \sigma^2 \text{ Det } (\mathbf{H}(\mathbf{x}, \sigma))$. The keypoint is updated to the position of the maximum response of the Taylor expansion: $\mathbf{p}_{i+1} = \underset{\mathbf{p}}{\text{argmax}}\ T_i(\mathbf{p})$. This process iterates until convergence. If the process does not converge before a threshold number of iterations, the keypoint is deemed not robust and thrown out.

2.1.3 Affine adaptation

So far, the keypoints we have described correspond to circular regions where the pixel radius is some multiple of the scale. To account for small affine changes seen in non-planar objects (like zebras), the shape of each circular keypoint is adapted into an ellipse.

An affine shape $\mathbf{A} = \begin{bmatrix} a & 0 \\ c & d \end{bmatrix}$ is estimated (as a lower triangular matrix) for each keypoint using an iterative technique involving the second moment matrix [19],[72],[73]. The affine shape matrix transforms an ellipse into a unit circle. Note that because the matrix is lower triangular one of its eigenvectors points in the downward direction. Thus, the shape has no influence on the orientation of the keypoint. For each point in scale space the second moment matrix is evaluated as:

$$\mathbf{M}(\mathbf{x}, \sigma) = \begin{bmatrix} I_x^2(\mathbf{x},\sigma) & I_x(\mathbf{x},\sigma) I_y(\mathbf{x},\sigma) \\ I_x(\mathbf{x},\sigma) I_y(\mathbf{x},\sigma) & I_y^2(\mathbf{x},\sigma) \end{bmatrix} \tag{2.5}$$

The goal is to "stabilize" each keypoint shape by searching for the transformation, $\mathbf{A}^*$, that causes the second moment matrix to have equal eigenvalues. For each keypoint,

its elliptical shape is initialized as a circle $\mathbf{A}_0 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$. For each iteration i:

(1) Compute the second moment matrix, $\mathbf{M}(\mathbf{A}_i\mathbf{x}, \sigma)$, at the warped image patch.

(2) Check if the keypoint shape is stable. A keypoint shape is stable if the eigenvalue ratio of the second moment matrix is close to 1. If the keypoint has been stable for two consecutive iterations, then accept $\mathbf{A}^* \leftarrow \mathbf{A}_i$ and stop iteration. Otherwise, if the number of iterations, i, is greater than some threshold, then stop and discard the keypoint.

(3) Update the affine shape using the rule $\mathbf{A}_{i+1} = \mathbf{M}(\mathbf{A}_i\mathbf{x}, \sigma)^{\frac{1}{2}}\mathbf{A}_i$.

The matrix $\mathbf{A}$ only defines the transformation from an ellipse to a circle. The standard representation of an ellipse is a conic of the form $\mathbf{E} = \mathbf{A}^T\mathbf{A}$. This means that $\mathbf{A}$ is only defined up to an arbitrary rotation [19], [21]. Thus, we can freely rotate $\mathbf{A}$ into a lower triangular matrix. This ensures that one of its eigenvectors is pointing downwards — *i.e.* in the direction of the "gravity vector" [21]. Making use of the gravity vector removes a dimension of invariance. To allow for the specification of keypoint orientation, the keypoint representation can be extended with a parameter θ that defaults to 0.

2.1.4 Orientation adaptation

The keypoint orientation is defined using the parameter θ. By default, the orientation of a keypoint can be assumed to be aligned with the "gravity vector" — *i.e.* $\theta = 0$ [21]. Otherwise, an orientation must be computed. A common method for determining a keypoint's orientation is to use the dominant gradient orientation. In theory adapting the orientation to match the dominant gradient will cause a computed keypoint description to be invariant to rotations.

To compute a keypoint's dominant orientation the pixels around a keypoint vote into a fine-binned orientation histogram [22]. A pixel's vote is weighted by its gradient magnitude multiplied by its Gaussian weighted distance to the keypoint center. The dominant orientation $\theta \in [0, 2\pi)$ is chosen as the peak of this histogram. If there is more than a single peak it is common to create a copy of the keypoint for each maxima in this histogram. This process is illustrated in Figure 2.3.

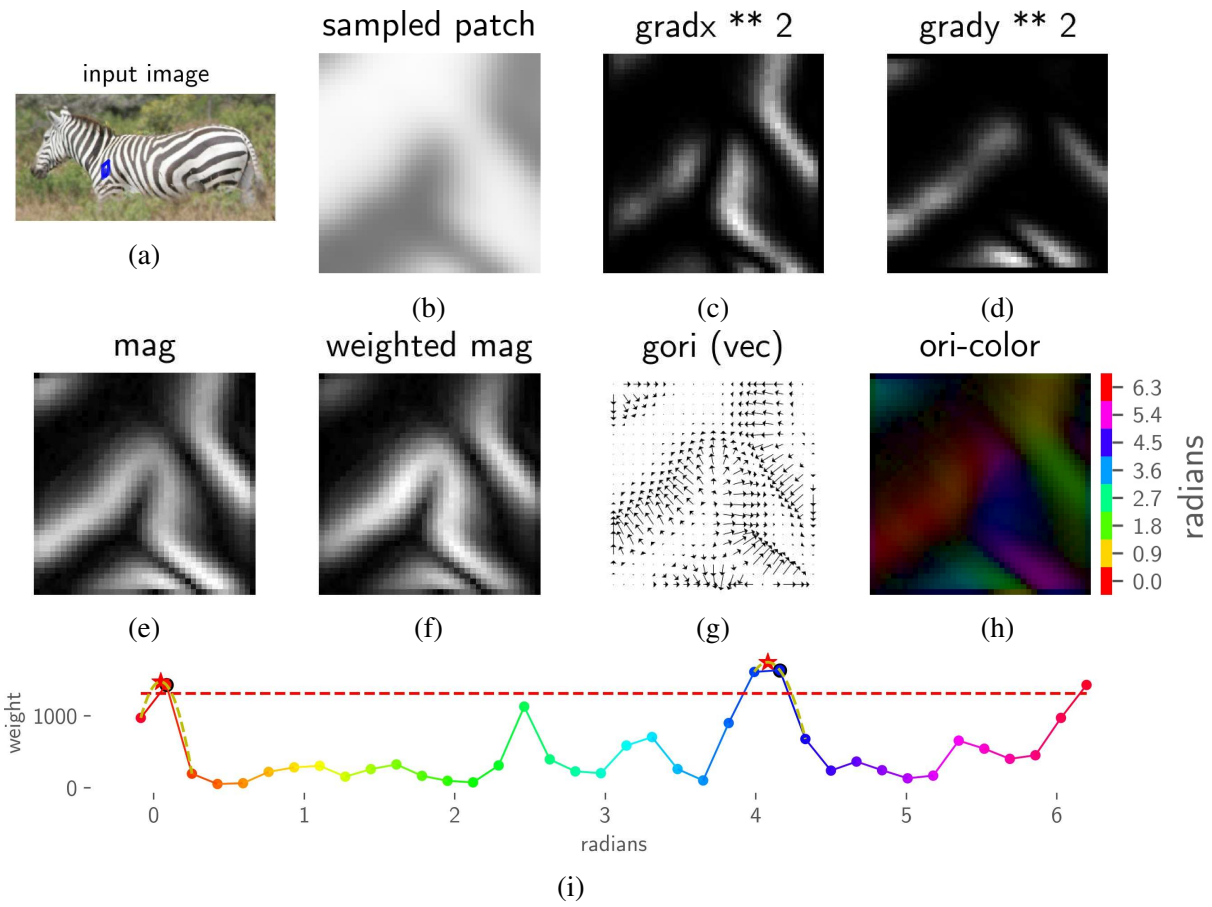


Figure 2.3: **Computing the dominant gradient orientation** The top row shows: (2.2a) the input image with a single elliptical keypoint, (2.2b) the normalized keypoint, (2.2c and 2.2d) the squared x and y image derivatives. The middle row shows: (2.2e) the gradient magnitudes, (2.2f) the Gaussian weighted gradient magnitude, (2.2g and 2.2h) the orientation at each pixel. The final row (2.2i) shows the histogram of weighted orientations. The starred positions show the dominant gradient orientations localized to sub-orientation accuracy.

2.1.5 Discussion — detector and invariance choices

To identify individual animals, features must be detected in distinguishing areas of an animal. For a feature to be useful, it must be detected in the multiple images of the same individual despite variations in viewpoint, pose, lighting, and quality. In our baseline algorithm we choose to use a Hessian based detector [21], [69] because it generally produces a large number of features and has been experimentally shown to be repeatable, accurate, and adaptable to multiple degrees of invariance [20].

Once a keypoint is detected, it is described using a keypoint description algorithm.

It is desirable for a keypoint description to be invariant to small changes in viewpoint, pose, and lighting. Accurate localization of a keypoint in scale and space helps to ensure that similar images contain similar features. Sometimes, it is beneficial to further localize a keypoint in shape and orientation, thus adding invariance to the feature. However, if too much invariance is used, it may not be possible to distinguish between semantically different features.

It is a challenge to choose the correct level of invariance when computing features. Often an application chooses one of two extremes. Consider the computation of keypoint orientation. Standard methods for orientation invariance assume patches can freely rotate, when in fact they may be constrained to be consistent with the orientation of surrounding patches [22]. On the other side extreme is the "gravity vector" [21], which globally enforces all keypoint to have a downward orientation. This may be a safe assumption when working with features from images of rigid objects taken in an upright position. It may not be correct when dealing with non-rigid objects like zebras. In our experiments in Section 3.5.5 we test different degrees on invariance. This test includes a novel method that achieves a middle ground between full orientation invariance and the gravity vector.

2.2 IMAGE FEATURE DESCRIPTION

Once each feature has been localized its visual appearance must be described before it can be matched. The goal of feature description is to encode raw image data into a vector — *i.e.* a *descriptor*. To represent the visual appearance of a keypoint a descriptor vector should have the following properties: (1) two visually similar patches produce vectors with a small metric distance and (2) visually dissimilar patches have vectors with large distances between them.

Constructing such a descriptor vector has been a core problem throughout the history of computer vision. The first texture descriptor robust to small image transformations was the scale invariant feature transform (SIFT) descriptor first published in 1999 [22], [74]. Since then, other hand-crafted algorithms have been proposed. However, results have always been at least comparable to the SIFT descriptor, and SIFT is still an effective and widely used hand-designed descriptors [23], [75]–[79]. A promising direction for outperforming the SIFT descriptor is descriptor learning [24], [25], [80];

specifically descriptor learning using deep neural networks [51], [81], [82]. This section first describes the basic SIFT algorithm and then provides an overview of alternatives that have been proposed to SIFT. Work related to learning descriptor vectors using deep neural networks is discussed later in Section 2.7.

2.2.1 SIFT

The SIFT descriptor is a 128 dimensional vector that summarizes the spatial distribution of the gradient orientations in an image patch [22]. To describe a keypoint with a SIFT descriptor, the keypoint's image data is warped using the affine transform of the scale space gradient image into a normalized reference frame (typically 41×41 pixels). For a descriptor to be useful in matching it is important that the keypoint is properly localized before a descriptor is computed [70]. Because it is not always possible to perfectly localize a keypoint, the SIFT descriptor aggregates information into a soft-histogram. Allowing data to contribute to multiple bins helps the SIFT descriptor to be robust to small localization errors and viewpoint variations. Distance between two SIFT descriptors is typically computed using the Euclidean distance. The SIFT descriptor of a patch is visualized in Figure 2.5.

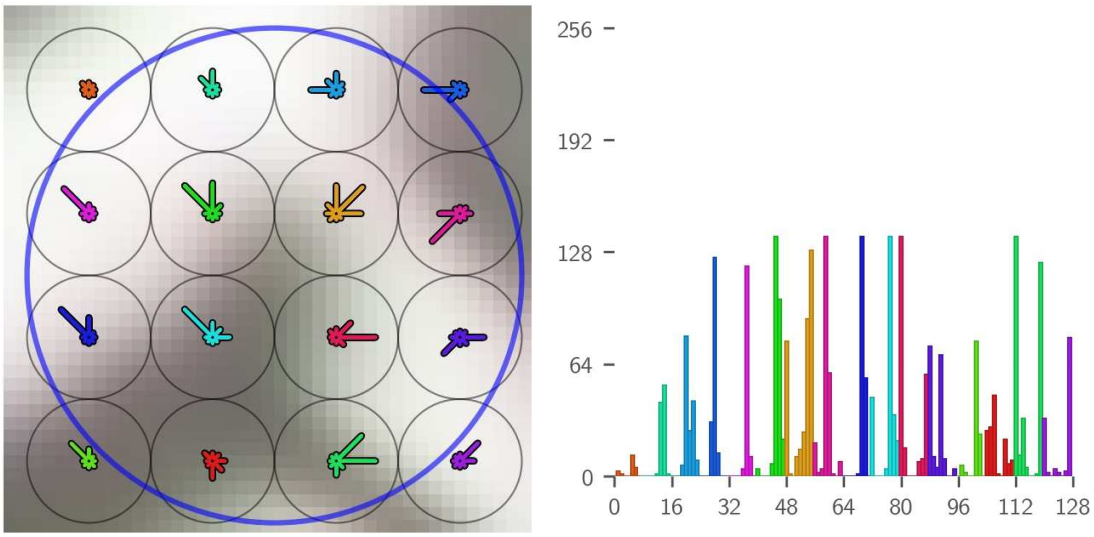
The structure of a SIFT descriptor is as follows: A 4×4 regular grid is superimposing over the normalized patch. Each of the 16 spatial grid cells contains an orientation histogram discretized into 8 bins. The SIFT descriptor is the concatenation of all orientation histograms, resulting in a single $16 \times 8 = 128$ dimensional vector.

The patch information populates the SIFT descriptor as follows: For every pixel, the patch gradient (the derivative in the x and y direction) is computed. Next, each pixel computes its gradient magnitude and orientation. Each pixel then casts a weighted vote. The bin that a pixel votes into is computed from its xy-location and gradient orientation. The weight of a pixel's vote is based on its gradient magnitude and Gaussian weighted distance to the patch center. To be robust to small localization errors, a pixel's vote is split via trilinear interpolation (x-location, y-location, and orientation) into the orientation histograms of the pixel's nearest grid cells as well as neighboring orientation bins in each grid cell's orientation histogram.

Once voting is completed a SIFT descriptor is normalized to account for lighting

differences between images. First, the vector is L2-normalized to unit length, which makes the descriptor invariant to linear changes in intensity. Then, a heuristic — that truncates each dimension to a maximum value of 0.2 — is applied to increase robustness to non-linear changes in illumination. Finally, the vector is renormalized.

For storage considerations the resulting 512-byte floating-point (float32) descriptor is typically cast as an array of unsigned 8-bit integers (uint8), resulting in a 128-byte descriptor. To reduce the impact of this quantization a trick is to multiply by 512 instead of 255 and then truncate values to 255 before converting from a float to a uint8. Even though each component is 8-bits and therefore can only store a maximum value of 255, value overflow is not likely to occur because of truncation, L2-normalization, and properties of natural images.



(a) (b)

Figure 2.5: **Example of a SIFT descriptor** (2.4a) shows a SIFT feature superimposed over pixels it describes. (2.4b) shows the same SIFT descriptor as a flat histogram. Notice the correspondence between the colors of the histogram bars.

2.2.2 Other descriptors and SIFT extensions

Even more than a decade after its original publication, SIFT remains a popular descriptor for patch-based matching because it is versatile, unsupervised, widely available, and easy to use. The principles used to guide the construction of the SIFT descriptor — particularly the use of aggregated gradients — have inspired many variants, extensions,

and new techniques [23], [76], [83]. Hand crafted alternatives to SIFT have been developed that are faster to compute and more efficient to store, but these alternatives do not significantly outperform SIFT's matching accuracy on general data [22], [23], [78]. This subsection provides a brief overview of these alternatives.

The use of aggregated gradient information in SIFT has been adapted for use in other computer vision problems such as detection and scene classification. The GIST descriptor is a low dimension descriptor used for scene classification that coarsely summarizes rough appearance of an entire image [84], [85]. The histogram of oriented gradients (HoG) descriptor is a high dimensional descriptor used in detection. The HoG descriptors describes the shapes of objects in an image [83]. Like the SIFT descriptor, the HoG descriptor illustrates the value of gradient-based image descriptions and has inspired extensions such as the discriminatively trained parts model [86].

As a general single-scale patch-based descriptor, the matching accuracy of SIFT has not been significantly outperformed on general datasets. One attempt at an improved general descriptor is the gradient location-orientation histogram (GLOH) descriptor [23]. GLOH uses a similar structure to SIFT but replaces the rectangular-bins with log-polar bins. GLOH did achieve higher matching accuracy on some datasets, but it was not by a significant margin. Despite the lack of generic success, hand-crafted SIFT variants have been successful when applied to specific tasks. Colored SIFT variants such as opponent-SIFT are valuable in category recognition tasks, where a color difference could be the distinguishing factor between categories [87] Combining multiple SIFT descriptors over different scales has also shown moderate improvements. The scale-less SIFT descriptor combines SIFT descriptors computed at multiple scales into a single descriptor. It has been shown to produce more accurate dense correspondences than representing each scale with an individual descriptor [88].

Efficiency is one area where SIFT has been significantly outperformed. An approximation to SIFT called speeded up robust features (SURF) is a fast approximation to SIFT based on integral images that achieves similar accuracy using a smaller 64 dimensional descriptor [76]. The DAISY descriptor uses a similar binning structure to GLOH, but uses convolutions with Gaussian kernels to quickly aggregate gradient histograms [89]. Binary descriptors such as local binary patterns (LBP) [90], [91], local derivative patterns [92],

and their variants such as BRIEF [75], BRISK [77], and FREAK [78] also quickly compute compact distinctive descriptors. Binary descriptors are built using multiple pairwise comparisons of average image intensity at predetermined locations. This results in a small descriptor that effectively represents aggregated gradient information.

Machine learning is able to outperform the matching accuracy of SIFT, however these techniques require training data to adapt to each new problem domain. Learned descriptors make use of the same aggregated gradient information used in the construction of SIFT descriptors. The Liberty, Yosemite, and Notre-Dame buildings datasets are standard datasets for descriptor learning [93]. Error on these datasets is measured using false positive rate at 95% recall (FPR95). The baseline SIFT error on this dataset is 27.02%. The configuration of a DAISY descriptor is learned in [25] and achieves an error of 15.16% on the buildings datasets. In [24], large scale non-convex optimization is used to learn a spatial pooling configuration of log-polar bins, a dimensionality reduction matrix, and a distance metric to further reduce the FPR95 error to 10.98%. The current state-of-the-art error of 4.56% on the buildings dataset is achieved using a convolutional neural network [26].

2.2.3 Discussion — descriptor choices

In our application we use the SIFT [22] as our baseline descriptor because it is one of the most widely used and well-known descriptors. SIFT describes images patches in such a way that small localization errors do not significantly impact the resulting representation. Exploration of alternative convolutional descriptors is discussed later in Section 2.7.

2.3 APPROXIMATE NEAREST NEIGHBOR SEARCH

In computer vision applications it is often necessary to search a database of high dimensional vectors [29], [94]–[97]. Patch descriptor vectors like SIFT are constructed such that the distance (under some metric) between vectors is small for matching patches and large for non-matching patches. Thus, finding matching descriptor vectors is often framed as a nearest neighbor search problem [22]. It becomes prohibitively expensive to perform exact nearest neighbor search as the size of the database increases. Therefore,

approximate algorithms — which can trade off a small amount of accuracy for substantial speed-ups — can be used instead.

2.3.1 Kd-tree

A *kd-tree* is a data structure used to index high dimensional vectors for fast approximate nearest neighbor search [98]. A kd-tree is an extension of a binary tree to multiple dimensions. Each non-leaf node of the tree is assigned a dimension and threshold value. The node splits data vectors between the left and right children by comparing the value of the data vector at the assigned dimension to the assigned threshold.

2.3.1.1 Building a kd-tree index

Indexing a set of vectors involves first choosing a dimension and threshold to split the data into two partitions. Then this procedure is recursively applied to each of the partitions. A common measure for choosing the dimension is to choose the dimension with the greatest variance in the data. The threshold is then selected as the median value of the chosen dimension.

2.3.1.2 Augmenting a kd-tree index

It is possible to augment an existing kd-tree by adding and removing vectors. Addition of a vector to a kd-tree is performed by appending the point to its assigned leaf. Removal of points from a kd-tree is done using lazy deletion — *i.e*. by masking the removed data. To avoid tree imbalance, a kd-tree is re-indexed after the number of points added or removed passes a threshold. Any masked point is deleted whenever the tree is re-indexed.

2.3.1.3 Searching a kd-tree index

Searching for a query point's exact nearest neighbor in a kd-tree has been shown to take expected logarithmic time for low ($k < 16$) dimensional data [99]. However, for higher dimensional data this same method takes nearly linear time [100]. This is because a query point and its nearest neighbor might be on opposite sides of a partition. Therefore, searching for nearest neighbors is typically done by approximate search using a priority queue [101]. A priority queue orders nodes to further search based on their distance to the

query vector. The search returns the best result after a threshold number of checks have been made.

Search accuracy is improved by using multiple randomized kd-trees [28]. If a single kd-tree has a probability of failure p, then m independently constructed trees have a p^m probability of failure. For each kd-tree a random Householder matrix is used to efficiently rotate the data. Using a random rotation preserves distances between rotated vectors but does not preserve the dimension of maximum variance. This means that each of the m kd-trees yields a different partitioning of the data, although it is not guaranteed to be independent. When searching multiple random kd-trees, a single priority queue keeps track of the next nearest bin boundaries to search over all the trees.

2.3.2 Hierarchical k-means

Another tree-based method for approximate nearest neighbor search is the hierarchical k-means. Each level in the hierarchical k-means tree partitions the data using the k-means algorithm [102] with a small value of k (*e.g.* 3). To query a new point it moves down the tree into the bin of the closest centroid at each level until it reaches a leaf node. Hierarchical k-means was one of the first techniques used to define a visual vocabulary [31] — a structure used for indexing and quantizing large amounts of descriptors.

2.3.3 Locality sensitive hashing

A hashing-based method for approximate nearest neighbor search is locality-sensitive hashing (LSH). This method is able to search a dataset of vectors for approximate nearest neighbors in sub-linear time [95], [96], [103]–[105]. LSH trades off a small amount of accuracy for a large query speed-up. A database is indexed using M hash tables. Each hash table uses its own randomly selected hash function. For each hash table, a query vector computes its hash and adds the database vectors it collided with to a shortlist. The shortlist is sorted by distance and returned as the approximate nearest neighbors.

2.3.4 FLANN

The fast library for approximate nearest neighbors (FLANN) is a software package built to quickly index and search high dimensional vectors [29]. The FLANN package

implements efficient algorithms for hierarchical k-means, kd-trees, and LSH. It also implements a hybrid between the k-means and kd-tree, as well as configuration optimization, to select the combination of algorithms that best reaches the desired speed/accuracy trade-off for a given dataset. Configuration optimization is performed using the Nelder-Mead downhill simplex method [106] with cross-validation.

2.3.5 Product quantization

Product quantization is a method for speeding up approximate nearest neighbor search of a set of high dimensional vectors [107], [108]. Each vector is split up into a set of sub-vector components. For each component, the sub-vectors are separately quantized using a codebook/dictionary/vocabulary. The pairwise squared distances between centroids in the vocabulary are stored in a lookup table. To comparing the distance between two vectors first each vector is split into sub-vectors, next the sub-vectors are quantized, and then the squared distances between quantized sub-vectors are read from the lookup table. The approximated squared distance between these two vectors is the sum of the squared distances between the quantized sub-vectors.

2.3.6 Discussion — choice of approximate nearest neighbor algorithm

In our single annotation identification algorithm a query descriptor searches for its nearest neighbor in a database containing all descriptors from all exemplars. Each annotation contains $\sim 10^4$ features, which are described with 128-component SIFT descriptors. Searching exact nearest neighbors becomes prohibitive when hundreds or thousands of images are searched. Thus, we turn towards approximate nearest neighbor algorithms. In this thesis all of our approximate nearest neighbors are found using the multiple kd-tree implementation in the FLANN package [29]. Using the configuration optimization built into the FLANN package, we have found that multiple kd-trees provide more accurate feature matches for our datasets than those computed by hierarchical k-means trees or LSH. In addition to being fast and accurate, multiple kd-trees support efficient addition and removal of points, which is needed in a dynamic setting [28].

2.4 INSTANCE RECOGNITION

There are many variations on the problem of visual recognition such as: specific object recognition (*e.g*. CD-covers) [22], [30], [31], location recognition [33], [35], [36], person re-identification [109]–[111], face verification/recognition [37]–[41], [112], category recognition [45]–[48], and fine-grained recognition [42]–[44]. The different types of recognition problems lie on a spectrum of specificity with respect to the objects they attempt to recognize. On one end of the spectrum, *instance recognition* techniques — like scene recognition or face verification — search for matches of the same exact object. On the other end of the spectrum category recognition algorithms — like car, bird, dog, and plane detectors — look for the same type of objects. Other problems sit — like fine-grained recognition where the goal might be to recognize specific subspecies of dog (*e.g*. German shepherd, golden retriever, boxer, beagle, . . .) — somewhere in the middle. Animal identification is closest to the instance recognition side of the spectrum, but the proposed solution draws upon techniques from other forms of recognition.

The discussion in this section focuses on instance recognition. The next two sections will discuss category recognition and fine-grained recognition.

2.4.1 Spatial verification

Before discussing specific techniques in instance recognition, we describe work related to spatial verification. Most instance recognition techniques initially match local image features without using any spatial information [22], [30], [32], [113]. This results in pairs of images with spatially inconsistent feature correspondences. Spatially inconsistent matches are illustrated in Figure 2.6. Inconsistent features are removed using *spatial verification*, a process based on the random sample consensus (RANSAC) algorithm [114].

RANSAC has come to refer to a family of iterative techniques to sample inliers from a noisy dataset that are consistent with some model [114]–[117]. In the context of spatial verification the model is an affine transformation matrix, and the dataset is a set of feature correspondences [22], [32], [118]–[120]. At each iteration of RANSAC a small subset of points is sampled from the original dataset and used to fit a hypothesis model. All other data points are tested for consistency with the hypothesis model. A score is assigned to the hypothesis model based on how well the out of sample data fit the

model (*e.g.* the number of transformed points that are within a threshold distance of their corresponding feature). After a certain number of iterations the process stops and returns the hypothesized model with the highest score as well as the inliers to that model.

When RANSAC returns a large enough set of inliers (with respect to some threshold), the hypothesis model it is generally considered to be a "good fit". In this case a more complex model — that may be more sensitive to outliers — can be fit. In spatial verification, it is common to use the RANSAC-inliers to estimate a homography transformation [121], 311–320. The homography is then used to estimate a new set of refined inliers, and these are returned as the spatially verified feature correspondences.

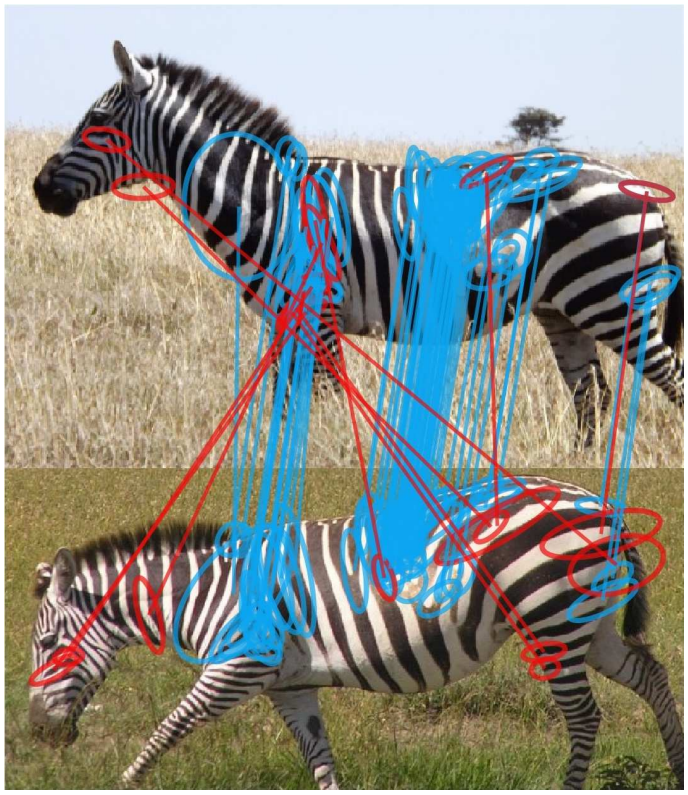


Figure 2.6: **Before and after spatial verification** This shows the matches before and after spatial verification. Inconsistent matches are shown in red. Consistent matches are shown in blue. Notice that not all spatially consistent matches are correct.

2.4.2 Lowe's object recognition

Lowe's introduction of SIFT descriptors includes an algorithm for recognizing objects in a training database and serves as an instance recognition baseline [22]. A single kd-tree indexes all database image descriptors. Approximate nearest neighbor search of the kd-tree is performed using the best-bin-first algorithm [101]. For a query image, each keypoint is assigned to its nearest neighbor as a match. The next nearest neighbor (belonging to a different object) is used as a normalizer — a feature used to measure the distinctiveness of a match. Any match with a ratio of distances to the match and the normalizer greater than threshold $t_{\texttt{ratio}}{=}0.8$ is filtered as not distinctive. Features likely to belong to the same object are clustered using a Hough Transform, and then clusters of features are spatially verified with a RANSAC approach [114].

2.4.3 Bag-of-words instance recognition

One of the most well-known techniques in instance recognition is the *bag-of-words* model introduced to computer vision by Sivic and Zisserman [30], [118]. The bag-of-words model addresses instance recognition using techniques from text retrieval. An image is cast as a text document where the image patches (detected at keypoints and described with SIFT) are the words. The concept of a visual word is formalized using a visual vocabulary. A *visual vocabulary* is defined by clustering feature descriptors traditionally constructed using k-means [102] (however more recent methods have learned vocabularies using neural networks [52]). The centroids of the clusters represent the *visual words* in the vocabulary. These centroids are used to quantize descriptor space. A feature in an image is assigned (quantized) to the visual word that is the feature's approximate nearest neighbor in the vocabulary. This means that each descriptor vector can be represented using just a single number — *i.e.* its index in the vocabulary. Vocabulary indices are used to construct an inverted index, which allows multiple feature correspondences to be made using a single lookup.

Given a visual vocabulary, the bag-of-words algorithm consists of two high level steps: (1) an offline indexing step and (2) an online search step. The offline step indexes a database of images for fast search. First each descriptor in each database image is assigned to its nearest visual word. An inverted index is constructed to map each visual

word to the set of database features assigned to that word. Each database feature is assigned a weight based on its term frequency (tf). Finally, each word in the vocabulary is assigned a weight based on its inverse document frequency (idf). The online step searches for the images in the database that are visually similar to the query image. First, each descriptor in the query image is assigned to its visual word, and the term frequency of each visual word in the query image is computed. Then, the inverted index is used to build a list of all images that share a visual word with the query. For each matching image, the sum of the tf-idf scores of the corresponding features is used as the image score. Finally, the ranked list of images is returned. These steps are now described in further detail.

2.4.3.1 The inverted index

The visual vocabulary allows for a constant length image representation. An image is represented as a histogram of visual words called a bag-of-words. A bag-of-words histogram is sparse because each image contains only a handful of words from a vocabulary. The sparsity of these vectors allows for efficient indexing using an inverted index. An inverted index maps each word to the database images that contain the word. Therefore, when a feature in a new query image is quantized the inverted index looks up all the database features that it matches to. A new feature correspondence is created for each database feature the inverted index maps to. For each correspondence a feature matching score is computed. Because all the word assignments and feature correspondences are known, the scores of all matching images can be efficiently computed by summing the scores of their respective feature correspondences.

2.4.3.2 Vocabulary tf-idf weighting

Each word in the database is weighted by its inverse document frequency (idf), and each individual descriptor is weighted by its term-frequency (tf) [30]. The idea behind the idf weight is that words appearing infrequently in the database are discriminative and should receive higher weight. The idea behind the tf weight is that words occurring more than once in the same image are more important.

2.4.3.3 Formal bag-of-words scoring

Let $\mathcal{X}$ be the set of descriptor vectors in an image. We also use $\mathcal{X}$ to refer to the image in general. Descriptor space is quantized using a visual vocabulary where $\mathcal{C}$ is the set of word centroids and $w_{\mathbf{c}}$ is the weight of a specific word. Let $\mathcal{X}_{\mathbf{c}} \subset \mathcal{X}$ be the set of descriptors in an image assigned to visual word $\mathbf{c}$. Let $q(\mathbf{x})$ be the function that maps a vector to a visual word. We overload notation to also let $q(\mathcal{X})$ map a set of descriptors into a set of visual words.

The tf-idf weighting of a single word $\mathbf{c}$ in the vocabulary is computed as follows: Let N be the number of images in the database. Let $N_{\mathbf{c}}$ be the number of images in the database that contain word $\mathbf{c}$. $|\mathcal{X}|$ is the number of descriptors in an image, and $|\mathcal{X}_{\mathbf{c}}|$ is the number of descriptors quantized to word $\mathbf{c}$ in that image. The idf weighting of word $\mathbf{c}$ is:

$$w_{\mathbf{c}} = \mathrm{idf}(\mathbf{c}) = \log\left(N/N_{\mathbf{c}}\right) \tag{2.6}$$

The tf weighting of a word $\mathbf{c}$ in an image $\mathcal{X}$ is:

$$\mathrm{tf}(\mathcal{X}, \mathbf{c}) = \frac{|\mathcal{X}_{\mathbf{c}}|}{|\mathcal{X}|} \tag{2.7}$$

Similarity between bag-of-words vectors is computed using the weighted cosine similarity. It is only necessary to sum the scores of matching features because the weight of a word that is not in both a query and database image is 0. The tf-idf similarity between two images can be written as

$$\mathrm{sim}(\mathcal{X}, \mathcal{Y}) = \sum_{\mathbf{c} \in q(\mathcal{X}) \cap q(\mathcal{Y})} \mathrm{tf}(\mathcal{X}, \mathbf{c})\, \mathrm{tf}(\mathcal{Y}, \mathbf{c})\, \mathrm{idf}(\mathbf{c}) \tag{2.8}$$

or equivalently

$$\mathrm{sim}(\mathcal{X}, \mathcal{Y}) = \frac{1}{|\mathcal{X}||\mathcal{Y}|} \sum_{\mathbf{c} \in \mathcal{C}} w_{\mathbf{c}} \sum_{\mathbf{x} \in \mathcal{X}_{\mathbf{c}}} \sum_{\mathbf{y} \in \mathcal{Y}_{\mathbf{c}}} 1 \tag{2.9}$$

The second formulation unifies the bag-of-words model with other vocabulary-based methods in the SMK framework, which will be discussed later in Section 2.4.8.

2.4.3.4 Extensions to bag-of-words

The main strength and the main weakness of vocabulary-based matching is its use of quantization. Quantization allows for large databases of images to be searched very rapidly [31]. However, quantization causes raw descriptors to lose much of their discriminative information [48], [122]. When a high-dimensional feature vector is quantized, it only encodes the presence of a word in a single number. This number is as descriptive as the partitioning of descriptor space, which is quite coarse in 128 dimensions, even with a large vocabulary. Several methods have been developed to help reduce errors caused by quantization.

Soft-assignment helps avoid quantization errors by mapping each raw descriptor to multiple words [122]. Another way to reduce quantization error is to use a finer partitioning of descriptors space [32]. Approximate hierarchical clustering and approximate k-means have been used to build vocabularies with up to 1.6×10^7 words [31], [32], [123]. Alternative similarity measures for descriptor quantization are also explored in [123]. A projection matrix for SIFT descriptors is learned in [124] to preserve information that would be lost in quantization.

Because the tf-idf weighting was originally designed for text recognition, it does not take into account challenges that occur in image recognition such as bursty features — a single feature that appears in an image with a higher than term expected frequency (*e.g.* bricks on a wall or vertical stripes on a zebra). Strategies for accounting for burstiness involve penalizing frequently occurring features by removing multiple matches to the same feature, using inter-image normalization, and using intra-image normalization. [125].

Query Expansion is a way to increase the recall of retrieval techniques and recover from tf-idf failure [119], [120], [126], [127]. After an initial query, all spatially verified feature correspondences are back-projected onto the query image. Then the query is re-issued. A model of "confusing features" — features more likely to belong to the background — can be used to filter out matches that should not be back projected onto the query image. Query expansion enriches the query with intermediate information that may help retrieve other viewpoints of the query image. However, because this technique requires at least one correct result in the ranked list, it only improves recall for queries that already have high accuracy.

One method to improve the performance of bag-of-words search is to remove non-useful features. It is found that as few as 4% of the features can be used in location recognition without loss in accuracy [128]. This related work defines a useful feature as one that is robust enough to be matched with a corresponding feature and stable enough to exist in multiple viewpoints. Thus, these useful features are computed as those that produced a spatially verified match to a correct image. Any feature that does not produce at least one spatially verified match is removed. Removing non-robust features both saves space and improves matching accuracy.

2.4.4 Min hash

Min-hashing is the analog of locality-sensitive hashing for sets. Min-hashing has been applied as an instance recognition technique for near-duplicate image detection [129], logo recognition [130], large scale image search [131], scene recognition [132], and unsupervised object discovery [133], [134].

The basic idea is to represent an image as a set of hashes based on permutations of a visual vocabulary. Recognition is accomplished by performing a lookup for each hash. Collisions are returned as the recognition results. Like LSH, the primary advantage of using min hash for instance recognition is its speed.

2.4.5 Hamming embedding

Hamming embedding is an extension of the bag-of-words framework that reduces the information lost in quantization by assigning each descriptor a small binary vector [33], [125], [135]. Each visual word $\mathbf{c}$, is assigned a $d_b \times d$ random orthogonal projection matrix $\mathbf{P_c}$, where d is the number of descriptor dimensions and d_b is the length of the binary code. A set of d_b thresholds, $\mathbf{t_c} \in \mathbb{R}^{d_b}$, is pre-computed for each word using the descriptors used to form the visual word cluster. These descriptors are projected using the word's random orthogonal matrix, and the median value of each dimension is chosen as that dimension's threshold.

When any descriptor, $\mathbf{d}$, is assigned to a word $\mathbf{c}$ it is also assigned a binary Hamming code, $\mathbf{b}$. To compute the binary Hamming code the descriptor is projected using the word's orthogonal matrix, $\mathbf{b}' = \mathbf{P_c d}$, and then each dimension is binarized based on a

threshold, $b_i = (b'_i > t_{\mathbf{c}i})$.

When a query descriptor, $\mathbf{d}$, is assigned to a word, $\mathbf{c}$, it is matched to all database descriptors belonging to that word. Each match is then assigned a score. First, the Hamming distance, h_d, is computed between the binary signature of the query and database descriptors. If the Hamming distance of the match is not within threshold, h_t, the score of the match is 0 and does not contribute to bag-of-words scoring. Otherwise, the score is the word's squared idf weight multiplied by a Gaussian falloff based on the Hamming distance. Using the inverted index, each image is scored by summing the scores of the descriptors that matched that image. The image scores are used to define a ranked list of results.

2.4.6 Fisher vector

A Fisher vector is an alternative to a bag-of-words [136], [137]. Like bag-of-words, Fisher vector representations have been used in both instance and category recognition [138]–[146]. Instead of training discrete visual vocabulary using the cluster centers of k-means, a Fisher vector encoding uses a continuous Gaussian mixture model (GMM). The number of Gaussian components in the GMM is similar to the number of words in a vocabulary. An image is encoded using the GMM by computing the likelihood of each feature with respect to the GMM. Likelihoods for different components of the GMM are aggregated using a soft-max function. Often, each component of this vector $\mathbf{v}$ is then power law normalized with fixed constant $0 \leq \beta < 1$. Power law normalization is a simple post processing method written as $v_i = \text{sign}\,(v_i)\,|v_i|^\beta$ [35]. Fisher vectors produce a much richer representation than normal bag-of-words vector because each descriptor is assigned to a continuous mixture of words rather than a single word.

It is noted in [136] that using Fisher vectors for instance recognition is similar to tf-idf. Normalized Fisher vectors down-weight frequently occurring GMM components — *i.e*. words with low idf weights. Furthermore, Fisher vector representations are well suited for compression, which allows scaling to large image collections.

2.4.7 VLAD — vector of locally aggregated descriptors

A vector of locally aggregated descriptors (VLAD) is similar to a Fisher vector descriptor — in fact it is a discrete analog of a Fisher vector [35], [137]. Like Fisher vectors, VLAD has been used in the context of both instance and category recognition [147]–[149]. VLAD still computes a visual vocabulary and assigns each feature to its nearest word, but instead of only recording presence or absence of a word, each feature computes the residual vector from the centroid of its assigned word. The residual vectors are summed to produce one constant length vector per word. All summed residuals are concatenated to produce a constant length image representation. Aggregation of the residual vectors allows for an accuracy similar to bag-of-words methods to be obtained, but using a smaller vocabulary ($\sim 10^1 - \sim 10^2$ words). Like Fisher vectors, VLAD descriptors are also power-law normalized [35].

There have been many extensions of the VLAD descriptor. The value of PCA, whitening, and negative evidence was shown in [147]. The MultiVLAD scheme is inspired by [150], and allows for retrieval of smaller objects that appear in larger images [149]. The basic idea is that VLAD descriptors are tiled in 3×3 grids. An integral image [151] of unnormalized VLAD descriptors is used to represent many possible tiles.

A vocabulary adaptation scheme is also introduced in [149]. The vocabulary is updated when a new image is added to the VLAD inverted index. This is performed by updating any word centroid $\mathbf{c}$ to $\mathbf{c}'$, where $\mathbf{c}'$ is the average of all the descriptors currently assigned to that word. The residuals of the affected words are recomputed and re-aggregated into updated VLAD descriptors.

Recently, NetVLAD — a convolutional variant of the VLAD descriptor — has been introduced [52], [152]. NetVLAD uses deep learning with a triplet loss function to simultaneously learn both the patch-based descriptors and the vocabulary. This convolutional approach shows large improvements (a 19% improvement on Oxford 5k) over previous state-of-the-art image retrieval techniques.

2.4.8 SMK — the selective match kernel

The selective match kernel (SMK) encapsulates the vocabulary-based techniques such as bag-of-words, Hamming embedding, VLAD, and Fisher vectors into a unified

framework [34], [36], [79], [113]. SMK provides a framework that "bridges the gap" between matching-based (here a match refers to a feature correspondence) approaches and aggregation-based approaches. The scores of matching-based approaches such as Hamming embedding and bag-of-words are based on establishing individual features correspondences. In contrast, the scores of aggregation approaches such as VLAD and Fisher vectors are computed from compressed image representations, where the individual features are not considered.

An advantage of a matching-based approach like Hamming embedding is that it can define a selectivity function. A selectivity function down weights individual feature correspondence with low descriptor similarity. Aggregation schemes have been shown to have their own advantages. Aggregated approaches like VLAD allow for matching applications to scale to a large number of images because each image is indexed with a compressed representation. Furthermore, aggregation-based approaches have been shown to provide better matching results on many datasets because they implicitly down weight bursty features [36], [113].

In the SMK framework a matching function and selectivity function are chosen. Different selections of these functions can implement and blend desirable attributes of the aforementioned frameworks. The matching function assigns correspondences between query and database descriptors. The choice of the matching function determines whether the resulting kernel is aggregated or non-aggregated. The selectivity function weights a correspondence's contribution to image similarity. It also can apply either power-law like normalization or hard thresholding in order to down weights correspondences with low visual similarity. One advantage of the SMK framework is that the selectivity function can be used in aggregated matching. In this case the selectivity function is applied to all correspondences assigned to a particular word.

2.4.9 Face recognition and verification

Face recognition is a specific form of instance recognition with the goal of recognizing individual human faces [38], [153]. Related to face recognition is the problem of face verification. In contrast to face recognition, face verification takes two unlabeled images and decides if they show the same face or different faces [41]. Clearly these techniques

are complementary because highly ranked results from a face recognition algorithm can be verified as true or false by a face verification algorithm.

Due to the specific nature of this problem specialized features detectors are often used. Facial feature detectors localize facial-landmarks such as the eye, mouth, and nose center and corner locations [39], [67]. Local texture-based descriptors such as Gabor filters [154]–[156] and local binary patterns (LBP) [40], [157] are extracted at detected facial regions [158]. Facial recognition researchers have also developed global descriptors — such as eigenfaces [159], Fisherfaces [160], and neural network based descriptions [41], [161]. — that represent the entire face. Recently, algorithms using both local and global representations computed using deep convolutional neural networks have shown state-of-the-art performance on both machine and human verification and recognition benchmarks [41].

In face recognition, each face image is encoded into a single vector. A function is trained to classify an unseen test image as an individual from the database of known faces. Many techniques are used in the literature to retrieve or classify a face. Examples of these techniques are: neural networks [41], [159], sparse coding [162], [163], principal component analysis (PCA) [164], Fisher linear discriminant (FLD) [165], linear discriminant analysis (LDA) [166], and support vector machines (SVMs) [167], [168].

Before the neural network revolution [50], sparse coding was one of the most popular techniques to retrieve faces [162], [163], [169], [170]. Sparse coding attempts to reconstruct unlabeled test vectors by searching for a linear combination of basis vectors from an over-complete labeled training database. Coding-based techniques are very similar to vocabulary-based methods. A codebook, dictionary, and vocabulary all are used to build image-level vector representations by quantizing raw features.

Another interesting technique is the Tom-vs-Pete classifier [39]. Given a set of N individuals (classes), a set of Tom-vs-Pete classifiers are used for both verification and indexing. At each facial landmark, k Tom-vs-Pete classifiers are computed. A single Tom-vs-Pete classifier is a linear SVM trained on a single corresponding feature for a single pair of classes. *E.g.* all the nose descriptors from class T and class P make up the SVM training data, and the learned SVM classifies a new nose feature as T-ish or P-ish. A descriptor vector for a single face is made by selecting 5000 out of the total

$k\binom{N}{2}$ classifiers and concatenating the signed distances from all the classifiers' separating hyperplanes. This descriptor facilitates both search and verification. A pair of face descriptor vectors can be verified as either a correct or incorrect match by constructing a new vector. The new vector is constructed by concatenating the element-wise product and difference of the two descriptor vectors. Then this new vector is classified using a radial basis function SVM.

One of the most recent advances in face verification and recognition is the DeepFace system [41]. The DeepFace system implements face verification using the following pipeline: (1) detect, (2) align, (3) represent, and (4) classify. Specialized facial point detectors and a 3D face model are used to register a 3D affine camera to an RGB-image. The image is then warped into a "frontalized" view using a piecewise affine transform. A face is represented as the 4096 dimensional output of a deep 7 layer convolutional neural network that exploits the aligned nature input images. An 8^{th}layer is used in supervised training where each output unit corresponds to a specific individual. At test time the L2-normalized output of the network is used as the feature representation. In a supervised setting, a χ^2-SVM is trained to recognize the individuals in a training dataset using the descriptor vectors produced by the network. In an unsupervised setting an ensemble of classifiers is used. The ensemble is composed of the output of a Siamese network [37] and several non-linear SVM classifiers with different inputs. The inputs are deep representations — the activations of a deep neural network's output layer — of the 3D aligned RGB-image, the 2D aligned RGB-image (generated using a simpler model based on similarity transforms), and an image comprised of intensity, magnitude, and orientation channels. Each input was fed through four deep networks each with different initialization seeds. DeepFace achieves an accuracy of 0.9735 on the Labeled Faces in the Wild dataset [38], which is comparable to the human performance measured at 0.975. When using unaligned faces the ROC score drops to 0.879, which demonstrates that alignment is very important for handling the problem of viewpoint in face verification.

2.4.10 Person re-identification

The person re-identification problem is typically posed in the context of locating the same person within a few minutes or hours from low-resolution surveillance

video [109]–[111],[171]. Common approaches to person re-identification typically transform images into a fixed length mid-level vector representation and a learned distance metric is used to compare representations. Mid-level representations can be built from color and texture histograms or extracted using a convolutional neural network. The distance metric is commonly learned as a Mahalanobis distance using linear discriminant analysis [171]. However, alternative approaches using dictionary learning [110] have also been shown to work well. Improvements to baseline can be achieved by conditioning person descriptors on viewpoint and pose [111]. Recently both features and distance metric have been learned using neural networks [109].

2.4.11 Discussion — instance recognition

Most instance recognition techniques use an indexing scheme based on a visual vocabulary [32], [33], [113], [147], [149], [172]–[174]. However, our baseline approach for animal identification does not use a visual vocabulary. This is because a visual vocabulary quantizes the raw features in the image and thus removes some of their discriminative ability [48], [122]. We have found this quantization to cause a noticeable drop in performance. Many aspects of our baseline algorithm are similar to Lowe's recognition algorithm [22], which does not quantize descriptors. The guiding principles of matching, filtering based on distinctiveness, filtering based on spatial consistency, and scoring are shared with our approach. However, our approach features several improvements to this algorithm. Furthermore, animal identification is a dynamic problem with specific domain-based concerns — such as quality and viewpoint in natural images — and requires innovation beyond Lowe's recognition algorithm.

Even though we would prefer to retain the discriminative information contained in raw descriptors, quantized image search has the ability to scale beyond our current suites [113], [136], [173]. In the future it may be necessary to investigate a VLAD-based SMK framework as a quantized alternative to our matching algorithm. Techniques such as soft-assignment [122] and learned vocabularies [123] could be used to reduce quantization errors. It is necessary to update the vocabulary as new images are added to the system. This issue could be addressed using the vocabulary adaptation technique in [149]. However, in this research we are more focused on the problem of verifying identifications

to reduce manual effort. As such we leave the scalable search issue for future work.

Facial recognition is similar to the problem of animal identification. Both problems seek to identify individuals. Some techniques used for face verification such as the Siamese network [37], [41] can be extended to the scope of animal identification. However, there is a much more mature literature on face recognition that has resulted in easily accessible and specialized algorithms for face feature detection and — most importantly — for face alignment. Individual animal identification does not have such a corpus of knowledge. We do not have access to highly specialized animal part detectors and alignment algorithms. Furthermore, we would like our algorithms to generalize to multiple species, so we would like to avoid over-specialized approaches. These are some reasons why convolutional neural networks will not make a prominent appearance in this thesis. Other reasons involve the size of our datasets. The recent NetVLAD network was trained using training datasets with $10,000$ to $90,000$ images [52]. We simply do not have this much labeled data. However, one goal of this thesis is to develop techniques that will help bootstrap labeled datasets of this size. Future research should investigate these deep learning techniques so they can be used after enough data has been collected for a specific species.

While the problem of animal identification and person re-identification are conceptually similar — sharing challenges such as lighting, pose, and viewpoint variation — differences in data collection creates the need for different solutions in practice. In contrast to the low-resolution image captured by surveillance cameras, the images used in animal identification are often manually captured by scientists in the field using high resolution DSLR cameras, and the goal is to match individuals over longer periods of time (years). Furthermore, re-identification techniques commonly focus on aggregate features that emphasize clothing, color, texture, and the presence of objects such as coats and backpacks, while in the animal id problem, it is often subtle localized variations in patterns on the skin and fur that distinguish individuals.

2.5 CATEGORY RECOGNITION

Different types of image recognition lie at different points on a spectrum of specificity. If instance recognition is at one end of the spectrum, then category recognition is at

the other. The goal of a *category recognition* algorithm is to assign a categorical class label to a query image [82],[175]–[179]. The categories often have visual appearances with a high degree of intra-class variance. *E.g.*, a recliner and a bench both belong to the chair category. Image representations and similarity measures are constructed to account for this. Despite this, techniques in category recognition have many similarities to instance recognition techniques. Until the neural network revolution [50], most category recognition techniques have been based on vocabulary methods [49], [50], [82], [180], [181] similar to those discussed in Section 2.4.3. This section first provides a brief overview of this literature. Then, we discuss naive Bayes classification techniques [47], [48] that play a large role in our baseline animal identification algorithms.

2.5.1 Vocabulary-based methods for category recognition

After vocabulary-based techniques demonstrated success in instance recognition, these techniques were quickly adapted and applied to category recognition [180]. Thus, there are many similarities — and some differences — in the techniques used to address these two problems. One difference is the size of the visual vocabulary. Instance recognition tends to require huge vocabularies ($\sim 10^5$ — $\sim 10^7$ words) to achieve a fine sampling of descriptor space [31], [32]. In contrast, category recognition uses smaller vocabulary sizes ($\sim 10^4$ words) to more coarsely sample descriptor space [46]. However, the vocabularies used in instance recognition have decreased in size with the advent of aggregated representations like VLAD and the Fisher vector [49], [149].

A second difference is how similarity between images is computed. In instance recognition the similarity between bag-of-word vectors is computed using a weighted cosine similarity. However, in category-recognition intra-class variation requires more sophisticated similarity measurements. Here, image similarity is computed using SVMs with different either linear or non-linear kernels such as χ^2, earth mover's distance, Hellinger, and Jensen-Shannon [46], [182], [183].

A third difference is the way that spatial information is used. Instead of filtering correspondences using spatial verification, spatial information is incorporated into category recognition algorithms using spatial pyramids [45], [184]. A spatial pyramid sub-divides an image into a hierarchy of grids. Max pooling is often used to select only the strongest

features in each spatial region [185], [186]. Each section of the image is encoded using the vocabulary and images are scored based on matches in each region.

2.5.1.1 Enhancements to category recognition

There are a wide variety of extensions and enhancements for image classification techniques based on bag-of-words, such as soft assignment of visual-words [187] and vocabulary optimization [188]. Numerous matching kernels — both linear and non-linear — have been developed such as kernel PCA, histogram intersection, and SVM square root bag-of-words vectors [189]–[191].

Generalized coding schemes improve performance over a bag-of-words image encoding. Vocabularies can be seen as codebooks or dictionaries in coding-based image classification techniques such as sparse coding and locally constrained linear coding [181], [188], [192]–[194]. Many coding schemes learn both the centroids and the function that quantizes a raw descriptor into a word [181], [188], [189], [192]–[194]. Techniques other than k-means are used to create vocabularies such as mean shift [192], coordinate descent with the locally constrained linear code criterion [188], and random forests [138]. Fisher vectors with linear classifiers have been found to outperform non-linear bag-of-words based SVM classifiers by using an L1-based distance measure and careful L2 and power-law normalization of descriptors [191], [195].

2.5.2 Naïve Bayes classification

The naïve Bayes nearest neighbor (NBNN) classifier is a simple non-parametric algorithm for category recognition that does not quantize descriptor vectors [48]. Boiman responds to the dominance of complex non-linear category recognition algorithms in the field [182], [196] by showing that simple techniques can compete with complex methods for category recognition. Boiman's paper also provides insight into the magnitude of information loss resulting from quantization.

Previous to [48], nearest neighbor classifiers had shown underwhelming accuracy in category recognition [45], [196], [197]. This was shown to be a result of using image-to-image distance. To remedy this, NBNN aggregates the information from multiple images by swapping the image-to-image distance for an image-to-class distance.

In NBNN, features of each class are indexed for fast nearest neighbor search, typically with a kd-tree [98]. For each feature, $\mathbf{d}_i$, in a query image, the algorithm searches for the feature's nearest neighbors in each class, $\mathrm{NN}_C(\mathbf{d}_i)$. The result of the algorithm is the class, C, that minimizes the image-to-class distance. In other words, the class of a query image is chosen by searching for the class that minimizes the total distance between each query descriptor and the nearest database descriptor in that class. This is expressed in the following equation:

$$C = \underset{C}{\operatorname{argmin}} \sum_{i=1}^{n} ||\mathbf{d}_i - \mathrm{NN}_C(\mathbf{d}_i)||^2 \tag{2.10}$$

This formulation where each descriptor is assigned to only the single nearest neighbor has been shown to be a good approximation to the minimum image-to-class Kullback-Leibler divergence [48] — a measure of how much information is lost when the query image is used to model the entire class.

2.5.3 Local naïve Bayes nearest neighbor

Local naïve Bayes nearest neighbor (LNBNN) is an improved version of the NBNN algorithm in both accuracy and speed [47]. In the original NBNN formulation a search is executed find each query descriptor's nearest neighbor in the database for each class separately. In contrast, the LNBNN modification searches all database descriptors simultaneously and ignores classes that do not return descriptor matches.

Each descriptor $\mathbf{d}_i$ in the query image searches for its nearest $K+1$ neighbors, $\{\mathbf{d}_1, \ldots, \mathbf{d}_K, \mathbf{d}_{K+1}\}$ over all classes. The first K neighbors are used as matches. The last neighbor is used as a normalizing term to weight the query descriptor's distinctiveness. Let $(\mathbf{d}_i, \mathbf{d}_j)$ be a matching descriptor pair, and let C be the class of $\mathbf{d}_j$. The score of each match is computed as the distance to the match subtracted from the distance to the normalizer.

$$s_{i,C} = ||\mathbf{d}_j - \mathbf{d}_K|| - ||\mathbf{d}_i - \mathbf{d}_j|| \tag{2.11}$$

The score of a class C is the sum of all the descriptor scores that match to it.

2.5.4 Discussion — class recognition

Progress in category recognition is generally made using techniques that allow classes with high intra-class variance to have lower matching scores. This is of little value to an instance recognition application, therefore we do not investigate most of the techniques in this section. However, the LNBNN [47] approach is interesting to us because it is a simple algorithm that does not suffer from quantization artifacts. NBNN and LNBNN [47], [48] never achieved state-of-the-art performance in image classification, however they have produced competitive results using simple techniques.

The simplicity of the techniques allowed for the authors to gain insight into visual recognition. Due to its simplicity and the insight that quantization significantly reduces the descriptive power of SIFT features, we adopt LNBNN as the baseline algorithm for animal identification.

2.6 FINE-GRAINED RECOGNITION

Fine-grained recognition is a problem more general than instance recognition, but more specific than category recognition [42]–[44]. Given an object of a known category, such as a bird, the goal of fine-grained recognition is to sub-classify the object into a fine-grained category such as a blackbird or a raven [198].

Algorithms for fine-grained recognition typically start by localizing the object and its parts with a detection algorithm [83] and parts-based models. Parts are segmented to remove background noise using algorithms like GrabCut [199]. Classification is performed locally on aligned parts as well as globally on the entire body and aggregated to yield a final classification.

Because fine-grained recognition lies on the same spectrum as instance recognition and category recognition it is not surprising that many of the same techniques — like Fisher vectors — are used [146]. Recently convolutional models have been successfully applied to fine-grained recognition [200]–[203].

2.6.1 Discussion — fine-grained recognition

The goal of fine-grained recognition is somewhat similar to animal identification. Fine-grained recognition localizes subtle information to distinguish between two similar

species, whereas animal identification localizes subtle information to distinguish between two similar individuals. However, the domains of species and individuals are dissimilar enough that off the shelf techniques for fine-grained recognition would need to be adapted before identification could be performed. One interesting avenue of research would be to use a parts model [86] as in [44], to align individuals before they are compared.

2.7 DEEP CONVOLUTIONAL NEURAL NETWORKS

Convolutional networks have been around for over more than two decades [204], [205]. However, they did not receive major attention from computer vision researchers until 2012 when a deep convolutional neural network (DCNN) [50] outperformed the best support vector machines (SVMs) [206] by over 10% in the ImageNet category recognition challenge [82]. Since then, many successful category recognition techniques based on DCNNs have been published [207]–[214]. DCNNs have also been shown to produce excellent results when applied to other computer vision problems such as: instance recognition [51], [52], [152], [215]–[217], fine-grained recognition [200], [201], [218], detection [219]–[221], face verification [41], [222], [223], and learning similarity between feature patches [26], [27], [27], [224], [225]. The sudden success of deep nets has been attributed (1) a larger volume of available of training data, and (2) implementations using faster GPUs [50].

Several techniques are employed to increase accuracy, reduce over-fitting, and reduce training time. Data augmentation is used to artificially increase the amount of training data [226]–[228]. The dropout technique has been shown to reducing over-fitting [229], [230]. At training time outputs of hidden units are randomly suppressed which forces the network to learn a more robust representation. It has been shown that dropout can be viewed as a form of model averaging [231]. Rectified linear units (ReLU) have been shown to be a faster alternative to the standard sigmoid activation functions [229], [232]. A ReLU is similar to a hinge function and simply outputs the signal of a unit if it is positive and outputs a zero otherwise. Leaky rectified linear units (LReLU) further improve network accuracy by including a "leakiness" term while maintaining the speed of ReLUs [233]. While a ReLU strictly suppresses a feature activation if it is negative a LReLU returns a small negative signal (by multiplying by a constant)

instead of zero.

A deep neural network is constructed by stacking several layers of units (neurons) together. Data is used to initialize the activations of an input layer, and the information is forward propagated through the network. Weights are chosen to optimize a loss function — *e.g.* categorical cross-entropy error or triplet loss [112] — which is chosen to depend on the application. Optimization of the loss function is performed using back-propagation [234] — typically using mini-batches and stochastic gradient descent with momentum [235]. Traditionally each layer in a neural network is fully connected — each pair of units between the previous layer and the current layer has its own edge weight — to the previous layer. However, in computer vision networks are constructed using convolutional layers.

A DCNN connects the input layer to a stack of convolutional layers [50]. A convolutional layer differs from a fully connected layer in that it is sparsely connected and that most of the edge weights between layers are shared [204], [205], [236]. Each convolutional layer is broken into several channels. Each channel is given its own weight matrix with a fixed width and height. This matrix of weights is convolved with the input layer to produce a feature activation map, one for each channel. Convolutional layers often use several pooling layers that aggregate information over a small area, reduce the size of the feature map, and increase robustness to transformations. Common pooling operations are max-pooling [50], [236] and maxout [237]. The convolutional layers may also be connected to a stack of fully connected layers. In this case, hierarchies of feature maps are built in the low level convolutional layers, and then fully connected layers learn decision boundaries between these features [238].

Because of weight sharing, convolutional networks must learn significantly fewer parameters than fully connected networks. This allows convolutional networks to be trained much faster. Fewer weights also acts as a form of regularization for the network. Intuitively learned convolutional filters are similar to Gabor filters [239], which are a naturally suited for extracting features from images. Even without learning weights, convolutions can be used to extract powerful features for matching [65]. The popular SIFT and HoG features [240] can even be implemented as convolutional networks. Despite the lack of hard theoretical insight into the inner workings of these networks, their

empirical performance cannot be denied.

2.7.1 Discussion — deep convolutional neural networks

Because of the astounding success of convolutional networks in almost every area in computer vision, we have investigated their use in animal identification. Specifically, we have investigated two approaches.

The first approach used deep convolutional feature descriptors as a replacement for the SIFT [22] descriptor following the patch-based scheme in [26]. The basic idea is to have two patches fed through the same (Siamese) [37] architecture and then compare their resulting encodings. This comparison can be as simple as Euclidean distance, or as complex as a learned distance measure. Training can be performed on pairs of patches, labeled as correct or incorrect, using the discriminative loss function [241]. Unfortunately, due to issues with the quality and quantity of our training data our convolutional replacements for the SIFT descriptor have not been successful.

The second approach aimed to use Siamese networks to directly compare two images of an animal to determine if they were the same or different, similar to the method used in DeepFace [41] for face verification. However, without the large training datasets and specialized alignment procedures used in DeepFace, we were unable to produce promising results.

Due to these issues, this thesis does not further pursue techniques based on DCNNs. We include this discussion to note the potential of deep learning applied to animal identification and to strongly suggest further investigation of these techniques in the future research. Of particular interest for future research is the matching technique presented in [242]. This method is particularly interesting because it learns to match and align images by mimicking a classic computer vision pipeline while using only synthetic training data. This may be able to overcome the issues mentioned above, however further investigation is needed.

3. IDENTIFICATION USING A RANKING ALGORITHM

This chapter addresses the problem of computer-assisted animal identification, where an algorithm suggests likely possibilities, but a human reviewer always makes the final decision. Given, a single annotation depicting an unknown animal and a database of previously identified annotations, the task is to determine a ranking of database individuals most likely to match the unknown animal. A human reviewer manually determines which — if any — of the top ranked results are correct.

An *annotation* is a rectangular region of interest around a specific animal within an image. Each annotation given a name label denoting its individual identity. A *name* refers to a group of annotations known to be the same individual. The identification process assigns a name label to an unknown annotation either as (1) the name label of a matched database annotation or (2) a new name label if no matches are found.

The ranking algorithm is based on the feature correspondences between a query annotation and a set of database annotations. In each annotation a set of patch-based features is detected at keypoint locations. Then the visual appearance of each patch is described using SIFT [22]. A nearest neighbor algorithm establishes a set of feature correspondences between a query and the annotations in the database. A scoring mechanism based on local naïve Bayes nearest neighbor (LNBNN) [47] produces a score for each feature correspondence. These scores are then aggregated into a single score for each name in the database, producing a ranked list of names. If the top ranked name has a "high" score, then it is likely to be the same individual depicted in the query annotation. If the top ranked name has a "low" score, then it is likely that the query individual is not depicted in any database annotation. An example ranked list returned by the algorithm is illustrated in Figure 3.2. For each result in the ranked list we decide if it is a correct or incorrect match. In the baseline algorithm this identification decision is left to a user. However, in the next chapter we will consider using an algorithm to automatically verify results returned in this ranked list.

The outline of this chapter is as follows: Section 3.1 discusses the initial processing of an annotation which involves image normalization, feature extraction, and feature

weighting. Sections 3.2 and 3.3 describe the baseline ranking and scoring algorithm. The first of these sections focuses on establishing feature correspondences, and the second focuses on filtering the correspondences. Section 3.4 describes the process for selecting exemplars. Section 3.5 provides an experimental evaluation of the ranking algorithm. Section 3.6 summarizes this chapter.

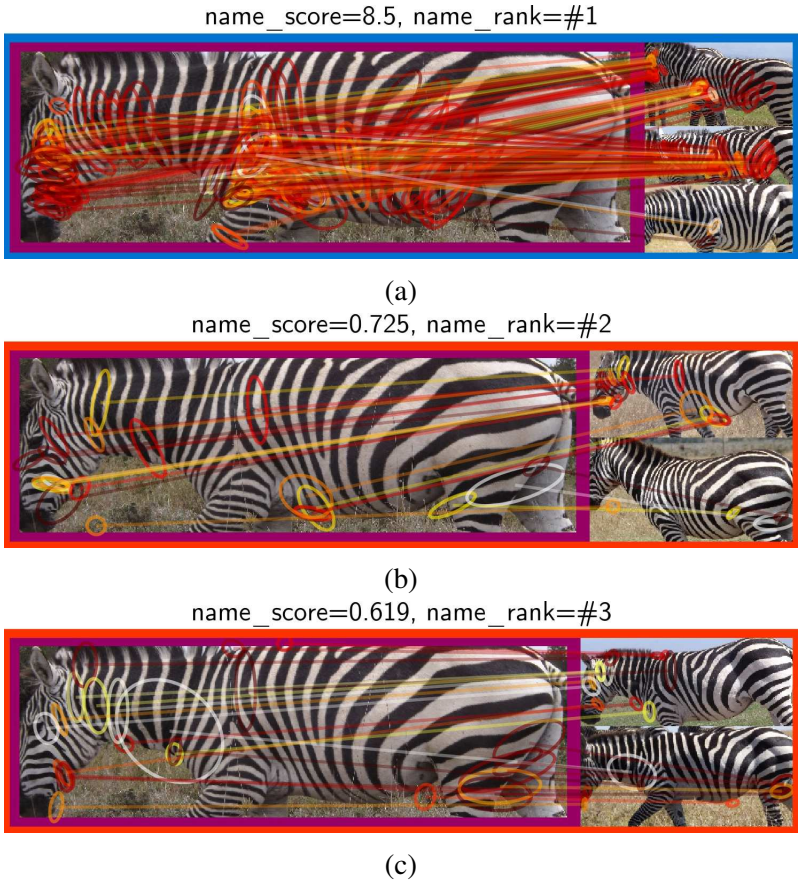


Figure 3.2: **Ranked matches** These are the top three results from the ranking algorithm. In each row, the query annotation is on the left and the exemplars of the matched name are on the right. Notice that the number of exemplars for each database name varies. The top-ranked match in (3.1a) is correct. The other ranks in (3.1b and 3.1c) are incorrect. The overall matching score is shown on the top of each result. The feature correspondences are overlaid on each result and colored by score.

3.1 ANNOTATION REPRESENTATION

For each annotation in the database we (1) normalize the image geometry and intensity, (2) compute features, (3) and weight the features. Image normalization rotates, crops, and resizes an annotation from its image. This helps to remove background clutter and roughly align the annotations in pose and scale. The extracted and normalized region is referred to as a *chip*. Then, a set of features — a keypoint and descriptor pair — is computed. Keypoints are detected at multiple locations and scales within the chip, and a texture-based descriptor vector is extracted at each keypoint. Finally, each feature is assigned a probabilistic weight using a foregroundness classifier. This helps remove the influence of background features.

3.1.1 Chip extraction

Each annotation has a bounding box and an orientation specified in a previous detection step. For zebras and giraffes, the orientation of the chip is chosen such that the top of the bounding box is roughly parallel to the back of the animal.

A chip is extracted by jointly rotating, scaling, and cropping an annotation's parent image using Lanczos resampling [243]. The scaling resizes the image such that the cropped chip has approximately 450^2 pixels and it maintains the aspect ratio of the bounding box. If specified in the pipeline configuration, adaptive histogram equalization [244] is applied to the chip, however this is not used in the experimental evaluation presented later in this chapter.

3.1.2 Keypoint detection and description

Keypoints are detected within each annotation's chip using a modified implementation of the Hessian detector described in [21] and reviewed in Section 2.1. This produces a set of elliptical features localized in space, scale, shape, and orientation. Each keypoint is described using the SIFT [22] descriptor that was reviewed in Section 2.2. The resulting unordered set of keypoint-descriptor pairs are an annotation's features. Further details about the keypoint structure are given in Section 3.1.4.

We choose a baseline feature detection algorithm that produces affine invariant keypoints with the gravity vector. Affine invariant (*i.e.* shape adapted) keypoints detect ellip-

tical patches instead of circular ones. We choose affine invariant keypoints because the animals we identify will be seen from many viewpoints. Because all chips have been rotated into an upright position, we assign all keypoints a constant orientation — this is the gravity vector assumption [21]. However, these baseline settings may not be appropriate for all species.

It is important to select the appropriate level of invariance for each species we identify. In our experiments in Section 3.5.5, we will vary several parameters related to invariance in keypoint detection. To determine if affine invariance is appropriate for animal identification we experiment with both circular and elliptical keypoints. We also experiment with different levels of orientation invariance. The gravity vector assumption holds in the case of rigid non-poseable objects (*e.g.* buildings), if the image is upright. Clearly, for highly poseable animals, this assumption is more questionable. However, full rotation invariance (using dominant gradient orientations) has intuitive problems. Patterns (like "V" and "Λ") that might contribute distinguishing evidence that two annotations match, would always appear identical under full rotation invariance. Ideally orientation selection would be made based on the pose of the animal.

We introduce a simple orientation heuristic to help match keypoints from the same animal in the presence of small pose variations. Instead of extracting a single keypoint in the direction of gravity or multiple keypoints in the directions of the dominant gradient orientation we extract 3 descriptors at every keypoint: one in the direction of gravity, and the other two offset at $\pm 15^{\circ}$ from the gravity direction. This provides a middle ground between rotation invariance and strict use of the gravity vector. Using this heuristic, it will be more likely to extract similar descriptors from two annotations of the same animal seen from slightly different poses.

3.1.3 Feature weighting

In animal identification, there will often be many annotations containing the same background. Photographers may take many photos in a single place and camera traps will contribute many images with the same background. Without accurate background masking, regions of an annotation from different images containing the same background may strongly match and outscore matches to correct individuals. An example illustrat-

ing two different individuals seen in front of the same distinctive background is shown in Figure 3.3. To account for this, each feature is given a weight based on its probability of belonging to the foreground — its "foregroundness". This weight is used indicate the importance of a feature in scoring and spatial verification.

Foregroundness is derived from a species detection algorithm developed by Parham [17]. The input to the species detection algorithm is the annotation's chip, and the output is an intensity image. Each pixel in the intensity image represents the likelihood that it is part of a foreground object.

A single feature's foregroundness weight is computed for each keypoint in an annotation as follows: The region around the keypoint in the intensity image is warped into a normalized reference frame. Each pixel in the normalized intensity patch is weighted using a Gaussian falloff based on the pixel's distance from the center of the patch. The sum of these weighted intensities is the feature's foregroundness weight. The steps of feature weight computation are illustrated in Figure 3.5.

Figure 3.3: **A scenery match** This is an example of two different animals appearing in front of the same distinctive background, illustrating the importance of background downweighting. The matching regions are highlighted.

3.1.4 Keypoint structure overview

The keypoint of a feature is represented as: $\mathbf{p} = (\mathbf{x}, \mathbf{A}, \theta)$, The vector $\mathbf{x} = \begin{bmatrix} x \\ y \end{bmatrix}$ is the feature's xy-location. The scalar θ is the keypoint orientation. The lower triangular matrix $\mathbf{A} = \begin{bmatrix} a & 0 \\ c & d \end{bmatrix}$ encodes the keypoint's shape and scale. This matrix skews and scales a keypoint's elliptical shape into a unit circle. A keypoint is circular when $a = d$ and $c = 0$. The keypoint scale is related to the determinant of this matrix and can be extracted as:

(a) (b) (c)

Figure 3.5: **Foregroundness weights** (3.4a) shows the annotation's cropped chip. This chip is passed to the species detector. (3.4b) shows the species detector outputs an intensity image indicating the likelihood that each pixel belongs to the foreground. (3.4c) shows the weighted sum of the intensity under each feature is used as that feature's foregroundness score.

$\sigma = \frac{1}{\sqrt{\text{Det}(\mathbf{A})}} = \frac{1}{\sqrt{ad}}$. All of this information can be encoded in a single affine matrix.

3.1.4.1 Encoding keypoint parameters in an affine matrix

It will be useful to construct two transformations that encode all keypoint information in a single matrix. The first, $\mathbf{B}$, maps a keypoint in an annotation into a normalized reference frame — the unit circle. The second transformation, $\mathbf{B}^{-1}$ is the inverse, which warps the normalized reference frame back onto the keypoint. To construct $\mathbf{B}$, the keypoint is centered at the origin $(0, 0)$ using translation matrix, $\mathbf{T}$. Then $\mathbf{A}$ is used to skew and scale the keypoint into a unit circle. Finally, the keypoint's orientation is normalized by rotating $-\theta$ radians using a rotation matrix $\mathbf{R}$.

$$\mathbf{B} = \mathbf{RAT} = \begin{bmatrix} \cos(-\theta) & -\sin(-\theta) & 0 \\ \sin(-\theta) & \cos(-\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a & 0 & 0 \\ c & d & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -x \\ 0 & 1 & -y \\ 0 & 0 & 1 \end{bmatrix} \tag{3.1}$$

The construction of $\mathbf{B}^{-1}$ is performed similarly.

$$\mathbf{B}^{-1} = \mathbf{T}^{-1}\mathbf{A}^{-1}\mathbf{R}^{-1} = \begin{bmatrix} 1 & 0 & x \\ 0 & 1 & y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{1}{a} & 0 & 0 \\ -\frac{c}{ad} & \frac{1}{d} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.2}$$

3.1.4.2 Extracting keypoint parameters from an affine matrix

During the spatial verification step, described in Section 3.3, keypoints are warped from one image into the space of another. It will be useful to extract the keypoint parameters from an arbitrary keypoint matrix. This will allow us to directly compare properties of corresponding features. Given an arbitrary affine matrix $\mathbf{B}^{-1}$ representing keypoint $\mathbf{p}$, we show how the individual parameters $(\mathbf{x}, \sigma, \theta)$ can be extracted. First consider the components of $\mathbf{B}^{-1}$ by simplifying the right side of Equation (3.2).

$$\mathbf{B}^{-1} = \begin{bmatrix} e & f & x \\ g & h & y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \frac{1}{a}\cos(\theta) & -\frac{1}{a}\sin(\theta) & x \\ \frac{1}{d}\sin(\theta) - \frac{c}{ad}\cos(\theta) & \frac{1}{d}\cos(\theta) + \frac{c}{ad}\sin(\theta) & y \\ 0 & 0 & 1 \end{bmatrix} \tag{3.3}$$

The position, scale, and orientation can be extracted from an arbitrary affine keypoint shape matrix $\mathbf{B}^{-1}$ as follows:

$$\begin{aligned} \mathbf{x} &= \begin{bmatrix} x \\ y \end{bmatrix} \\ \sigma &= \sqrt{\text{Det}\left(\mathbf{B}^{-1}\right)} \\ \theta &= \left(-\arctan2\left(f, e\right)\right) \mod 2\pi \end{aligned} \tag{3.4}$$

3.2 MATCHING AGAINST A DATABASE OF INDIVIDUAL ANIMALS

To identify a query annotation, it is matched against a database of known names. A name is a set of annotations known to depict the same animal. The basic matching pipeline can be summarized in three steps: (1) establish feature correspondences, (2) score feature correspondences, and (3) aggregate feature correspondence scores across the names. Correspondences between a query annotation's features and *all* database annotation features are established using an approximate nearest neighbor algorithm. This step also establishes a normalizing feature which is used to measure the distinctiveness of a query feature. Each feature correspondence is scored based on the feature weights established in the previous section and a measure of the distinctiveness of the query feature. The feature correspondence scores are then aggregated into a *name score* for each name in the database. The name scores induce a ranking on names in the database where

database names with higher ranks are more likely to be correct matches.

3.2.1 Establishing initial feature correspondence

3.2.1.1 Offline indexing

Before feature correspondences can be established, an offline algorithm indexes descriptors from all database annotations for fast approximate nearest neighbor search. All database descriptor vectors are stacked into a single array of vectors, $\mathcal{D}$, and these descriptors are indexed by an inverted index. The inverted index maps each descriptor in the stacked array back to its original annotation and feature. This database array is indexed for nearest neighbor search using a forest of kd-trees [28] using the FLANN library [29], both of which were reviewed in Section 2.3. This allows for the efficient implementation of a *neighbor index function* $\mathrm{NN}(\mathcal{D}, \mathbf{d}, K)$ that returns the indices in $\mathcal{D}$ of the K approximate nearest neighbors of a query feature's descriptor $\mathbf{d}$.

3.2.1.2 Approximate nearest neighbor search

Matching begins by establishing multiple feature correspondences between each query feature and several visually similar database features. For each query descriptor vector $\mathbf{d}_i \in \mathcal{X}$ the $K + K^*$ approximate nearest neighbors are found using the neighbor index function. These neighbors sorted by ascending distance are:

$$\mathrm{NN}(\mathcal{D}, \mathbf{d}_i, K + K^*) \equiv [j_1, \ldots, j_K, \ldots, j_{K+K^*}] \tag{3.5}$$

The K nearest neighbors, $[\mathbf{d}_{j_1}, \ldots, \mathbf{d}_{j_K}]$, are the initial feature correspondences to the i^{th} query feature. The remaining K^* neighbors, $\left[\mathbf{d}_{j_{K+1}}, \ldots, \mathbf{d}_{j_{K^*}}\right]$, are candidate normalizers for use in LNBNN scoring.

3.2.1.3 Normalizer selection

A single descriptor $\mathbf{d}_i^*$ is selected from the K^* candidate normalizers and used in computing the LNBNN score for all (up to K) of the i^{th} query descriptor's correspondences in the database. The purpose of a normalizing descriptor is to estimate the local density of descriptor space, which can be interpreted as a measure of the query descriptor's distinctiveness with respect to the database. The normalizing descriptor is chosen

as the most visually similar descriptor to the query that is not a correct match. In other words, the query descriptor's normalizer should be from an individual different from the query. The intuition is there will not be any features in the database that are close to distinctive features in the query except for the features that belong to the correct match.

The selection process described in the original formulation of LNBNN is to simply choose the $K + 1^{\text{th}}$ nearest neighbor, which amounts to setting $K^* = 1$. The authors of LNBNN find that there is no benefit to using a higher value of K^* [47]. However, this does not account for the case when some or all the $K+1^{\text{th}}$ nearest neighbor belongs to the same class as one of the nearest K neighbors. Therefore, we employ a slightly different selection process. To motivate our selection process, consider the case when there are more than K images of the same individual from the same viewpoint in the database and a distinctive feature from a new annotation of that individual is being scored. In this case K correspondences will be correctly established a distinctive the query feature and K database features. However, if the normalizer is chosen as the $K + 1$ neighbor, then these correspondences will be inappropriately downweighted.

To gain an intuition of the LNBNN scoring mechanism, consider the example in Figure 3.7. The figure shows the case where $K = 3$ and $K^* = 1$. In this case there are two examples (3.6b and 3.6c) of the query individual in the database. Even though there is an incorrect match, the LNBNN scores of the correct matches are an order of magnitude higher than the score for the incorrect match. Now, consider the case where the number of correct matches in the database is greater than K by setting $K = 1$. In this case the normalizing descriptor is the "same" feature as the query feature and the nearest match drops from 0.066 to 0.007.

To avoid this case, a normalizing feature is carefully chosen to reduce the possibility that it belongs to a potentially correct match. More formally, the normalizing descriptor is chosen to be the descriptor with the smallest distance to the query descriptor that is not from the same name as any of the chosen correspondences. Let $\mathcal{N}_j$ be the name associated with the annotation containing descriptor $\mathbf{d}_j$. Let $\mathscr{N}_i \equiv \{\mathcal{N}_j \mid j \in \mathrm{NN}(\mathcal{D}, \mathbf{d}_i, K)\}$ be the set of names matched by the i^{th} query feature. The descriptor that normalizes all matches

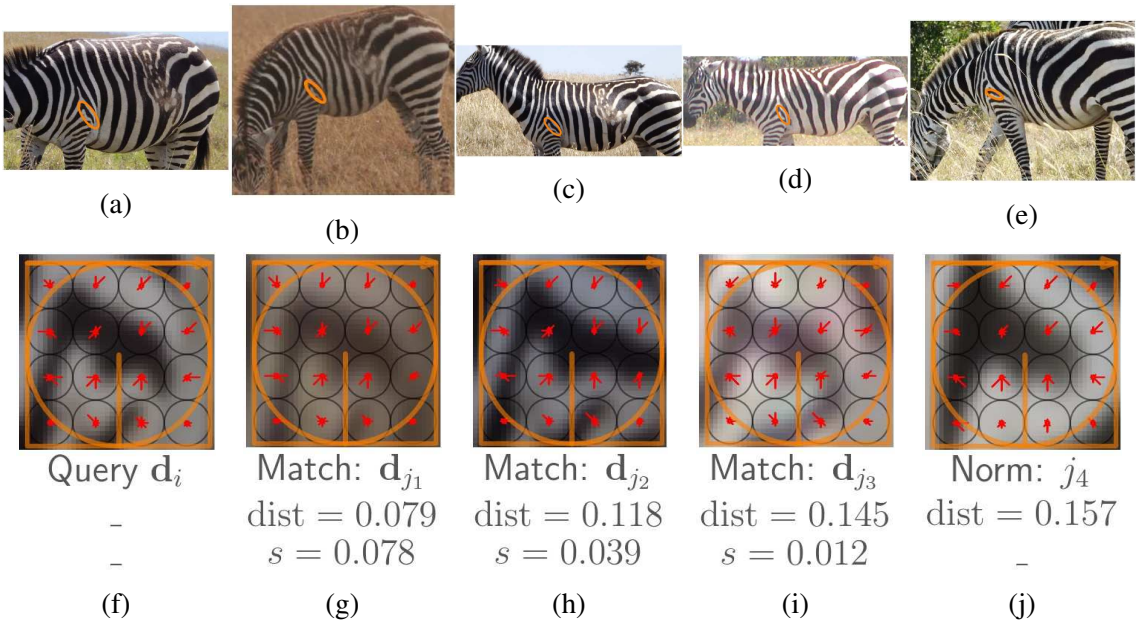


Figure 3.7: **LNBNN feature correspondence scoring** This shows the nearest four neighbors of a distinctive query feature (3.6f). The bottom row shows the warped and normalized features with their SIFT descriptors overlaid. The top row shows the annotation from which each feature was extracted. The first two neighbors (3.6g and 3.6h) are correct matches, the third neighbor (3.6i) is an incorrect match, and the fourth neighbor (3.6j) is used as an LNBNN normalizer to score the first three matches.

of query descriptor $\mathbf{d}_i$ is:

$$\mathbf{d}_i^* \equiv \underset{\mathbf{d}_j \in \left[\mathbf{d}_{j_{K+1}}, \ldots, \mathbf{d}_{j_{K^*}}\right]}{\operatorname{argmin}} ||\mathbf{d}_j - \mathbf{d}_i||^2 \mid \mathcal{N}_j \notin \mathscr{N}_i \tag{3.6}$$

3.2.2 Feature correspondence scoring

Each feature correspondence is given a score representing how likely it is to be a correct match. While the L2-distance between query and database descriptors is useful for ranking feature correspondences based on visual similarity, the distinctiveness of the match is more useful for ranking the query annotation's similarity to a database annotation [22], [47], [245]. However, highly distinctive matches from other objects — like background matches — do not provide relevant information about a query annotation's identity and should not contribute to the final score. Therefore, each feature correspondence is scored using a mechanism that combines both distinctiveness and likelihood that

the object belongs to the foreground. For each feature correspondence $m = (i, j)$ with query descriptor $\mathbf{d}_i$ and matching database descriptor $\mathbf{d}_j$, several scores are computed which are then combined into a single feature correspondence score $s_{i,j}$.

3.2.2.1 LNBNN score

The LNBNN scoring mechanism measures the difference in similarity between the query descriptor and (1) its match and (2) its normalizer $\mathbf{d}_i^*$. This serves as an estimate of local feature density and measures the distinctiveness of the feature correspondence. A match is distinctive when the query-to-match distance is much smaller than the query-to-normalizer distance, *i.e*. the local density of descriptor space around the query is sparse. The LNBNN score of a feature match is computed as follows:

$$s_{\texttt{LNBNN}} \equiv \frac{||\mathbf{d}_i - \mathbf{d}_i^*|| - ||\mathbf{d}_i - \mathbf{d}_j||}{Z} \tag{3.7}$$

All descriptors used in this calculation are L2-normalized to unit length — *i.e*. to sit on the surface of a unit hypersphere. The Z term normalizes the score to ensure that it is in the range $[0, 1]$. If descriptor vectors have only non-negative components (as in the case of SIFT [22]), then the maximum distance between any two L2-normalized descriptors is $Z{=}\sqrt{2}$. If descriptors vectors have negative components (like those that might extracted from a deep convolutional neural network [26]), then the maximum distance between them is $Z{=}2$.

3.2.2.2 Foregroundness score

To reduce the influence of background matches, each feature correspondence is assigned a score based on the foregroundness of both the query and database features. The geometric mean of the foregroundness of query feature, w_i, and database feature, w_j, drives the score to 0 if either is certain to be background.

$$s_{\texttt{fg}} \equiv \sqrt{w_i w_j} \tag{3.8}$$

3.2.2.3 Final feature correspondence score

The final score of the correspondence (i, j) captures both the distinctiveness of the match and the likelihood that the match is part of the foreground.

$$s_{i,j} \equiv s_{\texttt{fg}} s_{\texttt{LNBNN}} \tag{3.9}$$

3.2.3 Feature score aggregation

So far, each feature in a query annotation has been matched to several features in the database and a score has been assigned to each of these correspondences based on its distinctiveness and foregroundness. The next step in the identification process is to aggregate the scores from these patch-based correspondences into a single *name score* for each name in the database. Note that this name-based definition of scoring is a key difference between animal identification and image retrieval, where a score is assigned to each image in the database. In animal identification the analogous concept is an *annotation score* — a score assigned to each annotation in the database [32]. This distinction between a score from a query annotation to a database annotation is important because the goal of the application is to classify a new query annotation as either a known name or as a new name, not to determine which annotations are most similar.

This subsection presents two mechanisms to compute name scores. The first mechanism is annotation-based and computes a name score in two steps. This mechanism aggregates feature correspondence scores into an annotation score for each annotation in the database. Then the annotation scores are aggregated into a score for each name in the database. The second mechanism is feature-based. This mechanism aggregates feature correspondence scores matching multiple database annotations directly into a name score. These mechanisms are respectively similar to the image-to-image distance and the image-to-class distance described in [48].

3.2.3.1 The set of all feature correspondences

All scoring mechanisms presented in this subsection are based on aggregating scores from features correspondences. The set of all feature correspondences for a query anno-

tation $\mathcal{X}$ is expressed as:

$$\mathcal{M} \equiv \{(i,j) \mid \mathbf{d}_i \in \mathcal{X} \textbf{ and } j \in \mathrm{NN}(\mathcal{D}, \mathbf{d}_i, K)\} \tag{3.10}$$

3.2.3.2 Annotation scoring

An annotation score is a measure of similarity between two annotations. An annotation score between a query annotation and a database annotation is defined as the sum of the feature correspondence scores matching to the features from that database annotation. Let $\mathcal{Y}_j$ be the database annotation containing feature j. Let $\mathcal{M}_\mathcal{Y} \equiv \{(i,j) \in \mathcal{M} \mid \mathcal{Y}_j = \mathcal{Y}\}$ denote all the correspondences to a particular database annotation. The annotation score between the query annotation $\mathcal{X}$ and database annotation $\mathcal{Y}$ is:

$$\mathrm{K}_{\texttt{annot}}(\mathcal{X}, \mathcal{Y}) \equiv \sum_{(i,j) \in \mathcal{M}_\mathcal{Y}} s_{i,j} \tag{3.11}$$

3.2.3.3 Name scoring (1) — annotation-based

The annotation-based name scoring mechanism aggregates annotation scores into name scores by simply taking the maximum annotation score over all annotations belonging to a name. In our experiments we refer to this version of name scoring as `amech`. Let $\mathcal{N}$ be the set of database annotations with the same name. The annotation-based name score between a query annotation and a database name is:

$$\mathrm{K}_{\texttt{amech}}(\mathcal{X}, \mathcal{N}) \equiv \max_{\mathcal{Y} \in \mathcal{N}} \left(\mathrm{K}_{\texttt{annot}}\left(\mathcal{X}, \mathcal{Y}\right)\right) \tag{3.12}$$

3.2.3.4 Name scoring (2) — feature-based

The annotation-based name scoring mechanism accounts for the fact that animals will be seen multiple times, but it does not take advantage of complementary information available when a name has multiple annotations. The following aggregation mechanism combines scores on a feature level to correct for this. It allows each query feature at a specific location to vote for a given name at most once. Thus, when a query feature (or multiple query features at the same location) correspond(s) to database features from multiple views of the same animal, only the best correspondence for that feature will

contribute to the score. In our experiments we refer to this version of name scoring as `fmech`.

The first step of computing a name score for a specific name is grouping the feature correspondences. Two feature correspondences are in the same group if the query features have the same location and the database features belong to the same name. The next step is to choose the highest scoring correspondence within each group. The sum of the chosen scores is the score for a name. This procedure is illustrated in Figure 3.8.

Formally, consider two feature correspondences $m{=}(i,j)$ and $m'{=}(i',j')$. Let $\mathbf{x}_i$ and $\mathbf{x}_{i'}$ be the xy-location of the query feature in each correspondence. Let $\mathcal{N}_j$ and $\mathcal{N}_{j'}$ be the name of the database annotations containing the matched features. The group that contains feature correspondence m is defined as:

$$\mathcal{M}_m^G \equiv \{m' \in \mathcal{M} \mid ((\mathbf{x}_i = \mathbf{x}_{i'}) \textbf{ and } (\mathcal{N}_j = \mathcal{N}_{j'}))\} \tag{3.13}$$

The correspondence with the highest score in each connected component is flagged as chosen. Ties are broken arbitrarily.

$$\text{chosen}(m) \equiv \begin{cases} 1 & \text{if } \left(s_m > s_{m'} \quad \forall m' \in \mathcal{M}_m^G\right) \textbf{ or } |\mathcal{M}_m^G| = 1 \\ 0 & \text{otherwise} \end{cases} \tag{3.14}$$

Let $\mathcal{M}_\mathcal{N} \equiv \{(i,j) \in \mathcal{M} \mid \mathcal{N}_j = \mathcal{N}\}$ denote all the correspondences to a particular name. The feature-based name score of a name is:

$$\text{K}_{\texttt{fmech}}(\mathcal{X}, \mathcal{N}) \equiv \sum_{m \in \mathcal{M}_\mathcal{N}} \text{chosen}(m)\ s_m \tag{3.15}$$

Figure 3.8: **Feature-based name scoring** The query annotation is at the top left. Each query feature matches at most one feature in the exemplars for a name. Each line denotes a feature correspondence colored by its matching score. In the top right of each database annotation is its annotation score. Feature scores from multiple views are combined into a name score shown on top.

3.3 SPATIAL VERIFICATION

The basic ranking algorithm treats each annotation as an orderless set of feature descriptors (with a small exception in name scoring, which has used a small amount of spatial information). This means that many of the initial feature correspondences will be spatially inconsistent. Spatial verification removes these spatially inconsistent feature correspondences. Determining which features are inconsistent is done by first estimating an affine transform between the two annotations. Then a projective transform is estimated using the inliers to the affine transform. Finally, any correspondences that do not agree with the projective transform transformation are removed [32], [114]. This process is illustrated in Figure 3.10. We have reviewed related work in spatial verification in Section 2.4.1.

3.3.1 Shortlist selection

Standard methods for spatial verification are defined on the feature correspondences between two annotations (images). Normally, a shortlist of the top ranked annotations is passed onto spatial verification. However, in our application we rank names, which may have multiple annotations. In our baseline approach we simply apply spatial verification to the top $N_{\texttt{nameSL}} = 40$ names and the top $N_{\texttt{annotSL}} = 3$ annotations within those names.

3.3.2 Affine hypothesis estimation

Here, we will compute an affine transformation that will remove a majority of the spatially inconsistent feature correspondences. Instead of using random sampling of the feature correspondences as in the original RANSAC algorithm [115], we estimate affine hypotheses using a deterministic method similar to [32], [246]. Given a set of matching features between annotations $\mathcal{X}_1$ and $\mathcal{X}_2$, the shape, scale, orientation, and position of each pair of matching keypoints are used to estimate a hypothesis affine transformation. Each hypothesis transformation warps keypoints from annotation $\mathcal{X}_1$ into the space of $\mathcal{X}_2$. Inliers are estimated by using the error in position, scale, and orientation between each warped keypoint and its correspondence. Each inlier is weighted by its feature correspondence score. The final affine transform is chosen to be the one that produces the maximum weight set of inliers.

3.3.2.1 Enumeration of affine hypotheses

Let $\mathcal{M}_{\mathcal{X}_2}$ be the set of all correspondences between features from query annotation $\mathcal{X}_1$ to database annotation $\mathcal{X}_2$. For each feature correspondence $(i, j) \in \mathcal{M}_{\mathcal{X}_2}$, we construct a hypothesis transformation, $\mathbf{A}_{i,j}$ using the matrices $\mathbf{B}_i$ and $\mathbf{B}_j^{-1}$, which where defined in Equations (3.1) and (3.2). The first transformation $\mathbf{B}_i$, warps points from $\mathcal{X}_1$-space into a normalized reference frame. Then the second transformation, $\mathbf{B}_j^{-1}$, warps points in the normalized reference frame into $\mathcal{X}_2$-space. Formally, the hypothesis transformation is defined as $\mathbf{A}_{i,j} \equiv \mathbf{B}_j^{-1}\mathbf{B}_i$, and the set of hypothesis transformations is:

$$\mathcal{A} \equiv \{\mathbf{A}_{i,j} \mid (i, j) \in \mathcal{M}_{\mathcal{X}_2}\} \tag{3.16}$$

3.3.2.2 Measuring the affine transformation error

The transformation $\mathbf{A}_{i,j}$ perfectly aligns the corresponding i^{th} query feature with the j^{th} database feature in the space of the database annotation ($\mathcal{X}_2$). If the correspondence is indeed correct, then we can expect that other corresponding features will be well aligned by the transformation. The next step is to determine how close the other transformed features from the query annotation ($\mathcal{X}_1$) are to their corresponding features in database annotation ($\mathcal{X}_2$). This can be measured using the error in distance, scale, and orientation for every correspondence. The following procedure is repeated for each hypothesis transform $\mathbf{A}_{i,j} \in \mathcal{A}$. Note that the following description is in the context of the correspondence between the i^{th} query feature and the j^{th} database feature, and the i, j suffix is omitted for clarity. In this context, the suffixes k and ℓ will be used to index into features correspondences.

Let $\mathcal{B}_k = \left\{\mathbf{B}_k^{-1} \mid (k,\ell) \in \mathcal{M}_{\mathcal{X}_2}\right\}$ be the set of keypoint matrices in the query annotation with correspondences to database annotation $\mathcal{X}_2$. Given a hypothesis transform $\mathbf{A}$, each query keypoint in the set of matches $\mathbf{B}_k^{-1} \in \mathcal{B}_k$, is warped into $\mathcal{X}_2$-space:

$$\mathbf{B}_k^{-1\prime} = \mathbf{A}\mathbf{B}_k^{-1} \tag{3.17}$$

The warped position $\mathbf{x}_k{}'$, scale $\sigma_k{}'$, and orientation $\theta_k{}'$, can be extracted from $\mathbf{B}_k^{-1\prime}$ using Equation (3.4). The warped query keypoint properties in $\mathcal{X}_2$-space and can now be directly compared to the keypoint properties of their database correspondences. The absolute distance in position, scale, and orientation between the k^{th} query feature and the ℓ^{th} database feature with respect to hypothesis transformation $\mathbf{A}$ is measured as follows:

$$\begin{aligned}
\Delta\mathbf{x}_{k,\ell} &\equiv ||\mathbf{x}_k{}' - \mathbf{x}_\ell|| \\
\Delta\sigma_{k,\ell} &\equiv \max(\frac{\sigma_k{}'}{\sigma_\ell}, \frac{\sigma_\ell}{\sigma_k{}'}) \\
\Delta\theta_{k,\ell} &\equiv \min(|\theta_k{}' - \theta_\ell| \mod 2\pi, \quad 2\pi - |\theta_k{}' - \theta_\ell| \mod 2\pi)
\end{aligned} \tag{3.18}$$

3.3.2.3 Selecting affine inliers

Any keypoint correspondence $(k,\ell) \in \mathcal{M}_{\mathcal{X}_2}$ is considered an inlier with respect to $\mathbf{A}$ if its absolute differences in position, scale, and orientation are all within a spatial

distance threshold $t_{\mathbf{x}}$, scale threshold t_σ, and orientation threshold t_θ. This is expressed using the function $\mathrm{isinlier}$, which determines if match is an inlier:

$$\mathrm{isinlier}((k,\ell),\mathbf{A}) \equiv \Delta\mathbf{x}_{k,\ell} < t_{\mathbf{x}} \textbf{ and } \Delta\sigma_{k,\ell} < t_\sigma \textbf{ and } \Delta\theta_{k,\ell} < t_\theta \tag{3.19}$$

The set of inlier matches for a hypothesis transformation $\mathbf{A}$ can then be written as:

$$\mathcal{M}_{\mathbf{A}} \equiv \{m \in \mathcal{M}_{\mathcal{X}_2} \mid \mathrm{isinlier}(m,\mathbf{A})\} \tag{3.20}$$

The best affine hypothesis transformation, $\hat{\mathbf{A}}$, maximizes the weighted sum of inlier scores.

$$\hat{\mathbf{A}} \equiv \underset{\mathbf{A}\in\mathcal{A}}{\mathrm{argmax}} \sum_{(k,\ell)\in\mathcal{M}_{\mathbf{A}}} s_{k,\ell} \tag{3.21}$$

3.3.3 Homography refinement

Feature correspondences that are inliers to the best hypothesis affine transformation, $\hat{\mathbf{A}}$, are used in the least squares refinement step. This step is only executed if there are at least 4 inliers to $\hat{\mathbf{A}}$, otherwise all correspondences between features in query annotation $\mathcal{X}_1$ to features in database annotation $\mathcal{X}_2$ are removed. The refinement step estimates a projective transform from $\mathcal{X}_1$ to $\mathcal{X}_2$. To avoid numerical errors the xy-locations of the correspondence are normalized to have a mean of 0 and a standard deviation of 1 prior to estimation. A more comprehensive explanation of estimating projective transformations using point correspondences can be found in [121], 311–320.

Unlike in the affine hypothesis estimation where many transformations are generated, only one homography transformation is computed. Given a set of inliers to the affine hypothesis transform $\mathcal{M}_{\hat{\mathbf{A}}}$, the least square estimation of a projective homography transform is:

$$\hat{\mathbf{H}} \equiv \underset{\mathbf{H}}{\mathrm{argmin}} \sum_{(i,j)\in\mathcal{M}_{\hat{\mathbf{A}}}} ||\mathbf{H}\mathbf{x}_i - \mathbf{x}_j||^2 \tag{3.22}$$

Similar to affine error estimation, we will identify the subset of inlier features correspondences $\mathcal{M}_{\hat{\mathbf{H}}} \subseteq \mathcal{M}_{\mathcal{X}_2}$. A correspondence is an inlier if the query feature is transformed to within a certain spatial distance threshold $t_{\mathbf{x}}$, orientation threshold t_θ, and scale threshold t_σ of its corresponding database feature. For convenience, let $\mu(\cdot)$ and $\mu^{-1}(\cdot)$

transform points into and out of homogeneous form (recall homogeneous form augments an xy-point with a z coordinate). For each feature correspondence $(k, \ell) \in \mathcal{M}_{\mathcal{X}_2}$, the query feature position, $\mathbf{x}_k$, is warped from $\mathcal{X}_1$-space into $\mathcal{X}_2$-space.

$$\mathbf{x}_k{}' = \mu^{-1}\left(\hat{\mathbf{H}}\,\mu\left(\mathbf{x}_k\right)\right) \tag{3.23}$$

However, because projective transformations are not guaranteed to preserve the structure of the affine keypoints, warped scales and orientations cannot be estimated using Equation (3.4). Therefore, we estimate values that will serve a similar purpose relative to a reference point.

3.3.3.1 Estimation of warped shape parameters

Because we cannot warp the shape of an affine keypoint using a projective transformation, we instead estimate the warped scale and orientation for the k^{th} query feature using a reference point. Given a single feature correspondence $(k, \ell) \in \mathcal{M}_{\mathcal{X}_2}$, we associate a reference point $\mathbf{r}_k$ with the query location $\mathbf{x}_k$, scale σ_k and orientation θ_k. The reference point is defined to be σ_k distance away from the feature center at an angle of θ_k radians in $\mathcal{X}_1$-space.

$$\mathbf{r}_k = \mathbf{x}_k + \sigma_1 \begin{bmatrix} \sin\theta_k \\ -\cos\theta_\ell \end{bmatrix} \tag{3.24}$$

To estimate the warped scale and orientation, first the reference point is warped from $\mathcal{X}_1$-space into $\mathcal{X}_2$-space.

$$\mathbf{r}_k{}' = \mu^{-1}\left(\hat{\mathbf{H}}\,\mu\left(\mathbf{r}_k\right)\right) \tag{3.25}$$

Then we compute the residual vector $\tilde{\mathbf{r}}$ between the warped point and the warped reference point:

$$\tilde{\mathbf{r}} = \begin{bmatrix} \tilde{r}_x \\ \tilde{r}_y \end{bmatrix} = \mathbf{x}_k{}' - \mathbf{r}_k{}'. \tag{3.26}$$

The warped scale and orientation are estimated using the length and angle of the residual vector. Recall, the warped location can be computed exactly. In summary, the warped

location, scale, and orientation of the k^{th} query feature is:

$$\begin{aligned} \mathbf{x}_k{}' &\equiv \mu^{-1}\left(\hat{\mathbf{H}}\,\mu\left(\mathbf{r}_k\right)\right) \\ \sigma_k{}' &\equiv ||\tilde{\mathbf{r}}|| \\ \theta_k{}' &\equiv \arctan2\left(\tilde{r}_y, \tilde{r}_x\right) \end{aligned} \tag{3.27}$$

3.3.3.2 Homography inliers

The rest of homography inlier estimation is no different from affine inlier estimation. Equation (3.18) is used to compute the errors $(\Delta\mathbf{x}_{k,\ell}, \Delta\sigma_{k,\ell}, \Delta\theta_{k,\ell})$ between the warped query location, scale, and orientation, $(\mathbf{x}_k{}', \sigma_k{}', \theta_k{}')$, and the corresponding database location, scale, and orientation, $(\mathbf{x}_\ell, \sigma_\ell, \theta_\ell)$. The final set of homograph inliers is given as:

$$\mathcal{M}_{\hat{\mathbf{H}}} \equiv \left\{ m \in \mathcal{M}_{\mathcal{X}_2} \mid \text{isinlier}(m, \hat{\mathbf{H}}) \right\} \tag{3.28}$$

Spatial verification results in a reduced set of inlier feature correspondences from the query annotation to the database annotations. The name scoring mechanism from Section 3.2.3 is then applied to these inlier feature correspondences. This final per-name score is the output of the identification algorithm and used to form a ranked list that is presented to a user for review.

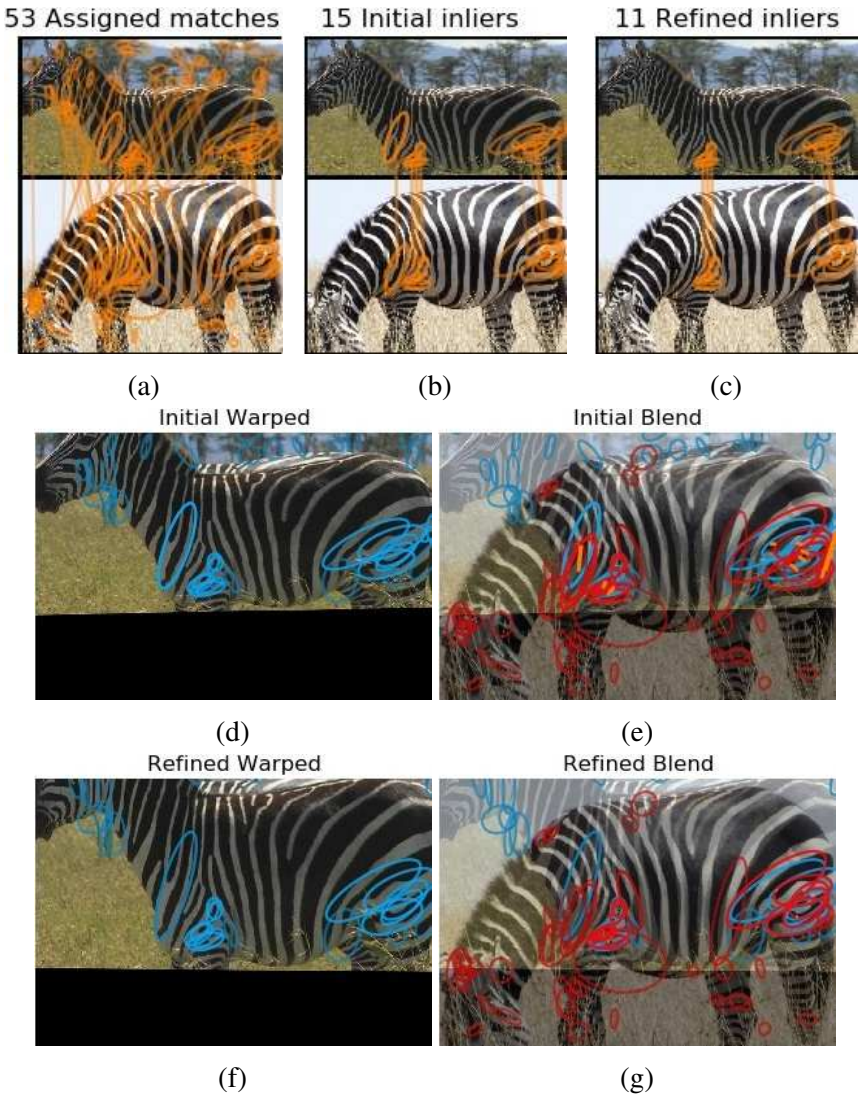


Figure 3.10: **Spatial verification** This shows an example of spatial verification process. The three images on the top show (3.9a) the original matches, (3.9b) the best set of inliers from affine hypothesis generation, and (3.9c) the final set of homography inliers. The images on the bottom show (3.9d and 3.9f) the matching images warped and superimposed by both the best affine (3.9e) and estimated homography transformation (3.9g).

3.4 EXEMPLAR SELECTION

To scale one-vs-many matching to larger databases and to allow the LNBNN mechanism to find appropriate normalizers we restrict the number of examples of each individual in the database to a set of exemplars.

Exemplars that represent a wide range of viewpoints and poses are automatically chosen by using a modified version of the technique presented in [247]. The idea is to treat exemplar selection as a maximum weight set cover problem. For each individual, the input is a set of annotations. A similarity score is computed between pairs of annotations. To compute covering sets we first choose a threshold, each annotation is assigned a covering set as itself and the other annotations it matches with a similarity score above that threshold. The maximum number of exemplars is restricted by setting a maximum weight. Searching for the optimal set cover is NP-hard, therefore we use the greedy $(1 - \frac{1}{e})$-approximation algorithm [248]. The algorithm is run for several iterations in order to find a good threshold that minimizes the difference between the weight of the set cover and the maximum weight limit. The similarity score between annotations can be computed using the LNBNN scores, but a better choice is the of the algorithm we will later describe in Chapter 4 to produce the probability that a pair of annotation correctly matches.

3.5 EXPERIMENTS

This section presents an experimental evaluation of the identification algorithm using annotated images of plains zebras, Grévy's zebras, Masai giraffes, and humpback whales. The input to each experiment is (1) a dataset, (2) a subset of query and database annotations from the database, (3) a pipeline configuration. The datasets are described in Section 3.5.1. The subsets of query and database annotations are carefully chosen to measure the accuracy of the algorithm under different conditions and to control for time, quality, and viewpoint. The pipeline configuration is a set of parameters — *e.g.* the level of feature invariance, the number of the nearest neighbors, and the name scoring mechanism — given to the identification algorithm. We will vary these pipeline parameters in order to measure their effect on the accuracy of the ranking algorithm.

For each query annotation, the identification algorithm returns a ranked list of

names with a score for each name. The accuracy of identification is measured using the cumulative match characteristic [249] which can be understood as the probability that a query correctly finds a match at a specified rank under the assumption that a correct match exists in the database. We are primarily concerned with only the first point in this curve — the fraction of queries with a correct result at rank 1 — because often a user of the system will only review the first result of a query.

The outline of this section is as follows. First, Section 3.5.1 introduces and describes each dataset. Our first experiment in Section 3.5.2 establishes the accuracy of our ranking algorithm on several datasets using a default pipeline configuration. We then compare our approach to an alternative SMK approach in Section 3.5.3. The next subsections perform in depth experiments on the parameter settings of our algorithm. Section 3.5.4 tests the effect of the foregroundness weight on identification accuracy. Section 3.5.5 investigates the effect of the level of feature invariance and viewpoint. Section 3.5.6 compares the annotation-based and the feature-based name scoring mechanism. Section 3.5.7 varies the K parameter (the number of the nearest neighbors used in establishing feature correspondences) and investigates the relationship between K and database size in terms of both the number of annotations and the number of exemplars per name. Section 3.5.8 discusses the failure cases of our ranking algorithm. Finally, Section 3.5.9 summarizes this section.

3.5.1 Datasets

All the images in the datasets used in these experiments were taken by photographers in the field. Each dataset is labeled with ground truth in the form of annotations with name labels. Annotations (bounding boxes) have been drawn to localize animals within the image. A unique name label has been assigned to all annotations with the same identity. Some of this ground truth labeling was generated independently. However, large portions of the datasets were labeled with assistance from the ranking algorithm. While this may introduce some bias in the results, there was no alternative because the amount of time needed to independently and completely label a large dataset is prohibitive and error prone.

There are two important things to note before we describe each dataset. First, in

order to control for challenging factors in the images such as quality and viewpoint some experiments sample subsets of the datasets we describe here. Second, labeling errors exist in some datasets.

Table 3.1: **Database statistics** These statistics indicate the number of individuals and annotations in each dataset. An encounter is a group of annotations from the same individual taken at the same place and time. Resighted names contain multiple encounters.

	Names (singleton)	Names (resighted)	Encounters per name (resighted)	Annots per encounter	Annots
Plains zebras	633	653	2.9 ± 1.3	2.6 ± 2.7	6474
Grévy's zebras	378	348	4.7 ± 2.3	1.1 ± 0.4	2285
Masai giraffes	102	46	2.8 ± 1.2	2.7 ± 3.1	625
Humpbacks	440	403	2.2 ± 0.6	1.0 ± 0.1	1345

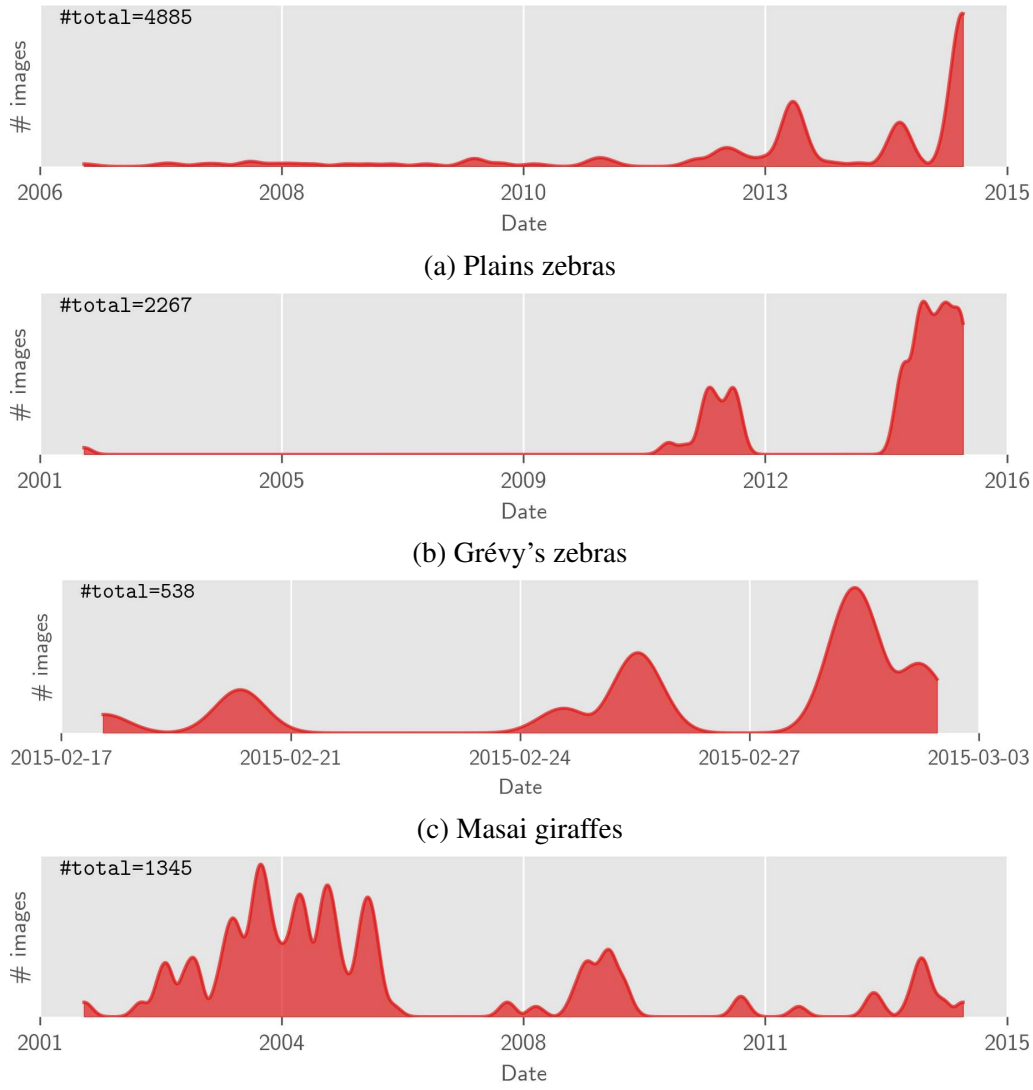
Table 3.2: **Annotations per quality** The "None" column indicates the number of annotations without a quality label.

	Excellent	Good	Ok	Poor	None
Plains zebras	407	1490	2936	1056	585
Grévy's zebras	3	552	385	136	1209
Masai giraffes	32	207	324	62	0
Humpbacks	0	0	0	0	1345

Table 3.3: **Annotations per viewpoint** Columns names are abbreviated using the first letters of front, left, back, and right. The "None" column indicates the number of annotations without a viewpoint label.

	BL	L	FL	F	FR	R	BR	B	None
Plains zebras	458	5603	413	0	0	0	0	0	0
Grévy's zebras	0	0	0	1	112	1138	235	1	798
Masai giraffes	59	388	72	7	4	70	12	13	0
Humpbacks	0	0	0	0	0	0	0	0	1345

The number of names, annotations, and their distribution within each database are summarized in the following tables. In these tables we distinguish between *singleton* and *resighted* names. Singleton names are individuals sighted only once, *i.e*. contain only



(a) Plains zebras

(b) Grévy's zebras

(c) Masai giraffes

(d) Humpbacks

Figure 3.12: **Distribution of image timestamps** The y-axis is plotted on a square-root scale to emphasize times when only a few images were taken. For plains zebras and Grévy's zebras images were collected over many years. For Masai giraffes all data was collected immediately before and during the GZC.

a single encounter. Resighted names contain more than one encounter. We make this distinction because resighted names have correct matches across a significant time delta. Note, that singleton names may still have more than one annotation, but those annotations are all from the same encounter. We have pre-filtered each database to remove annotations that are unidentified, are missing timestamps, or are labeled as "junk" quality.

Table 3.1 summarizes the number of annotations and individuals in each database as

well as the number of times (distinct encounters) each individual was sighted. Table 3.2 summarizes the quality labels of the annotations. Table 3.3 summarizes the viewpoint labels of the annotations. Distributions of when images in each dataset were taken are illustrated in Figure 3.12. The name and a short description of each dataset is given in the following list.

- **Plains zebras**. Our plains zebras dataset is an aggregate of several smaller datasets. There is variation in how the data was collected and preprocessed. Some images are cropped to the flank of the animal, while others are cropped to encompass the entire body. The constituent datasets were collected in Kenya at several locations including Nairobi National Park, Sweetwaters, and Ol Pejeta. More than 90% of the ground truth generated for this dataset was assisted using the matching algorithm. This dataset contains many imaging challenges including occlusion, viewpoint, pose, quality, and time variation. There are some annotations in this dataset without quality or viewpoint labelings and some images contain undetected animals. This data was collected between 2006 and 2015, but the majority of the data was collected in 2012–2015.
- **Grévy's zebras**. This is another aggregated dataset. The original ground truth for this dataset was generated independently of the ranking algorithm, however the ranking algorithm revealed several ground truth errors that have since been corrected. The Grévy's dataset was collected in Mpala, Kenya. Most of the annotations in this database have been cropped to the animal's flank. This dataset contains a moderate degree of pose and viewpoint variation and occlusion. This data was collected between 2003 and 2012, but the majority was collected in 2011 and 2012.
- **Masai giraffes**. These images of Masai giraffes were all taken in Nairobi National Park during the GZC between February 20, 2015 and March 2, 2015. All ground truth was established using the ranking algorithm followed by manual verification. This dataset contains a high degree of pose and viewpoint variation, and occlusion. Because of their long necks, it is difficult to ensure that only a single giraffe appears in each annotation. This results in many *photobombs* — pairs of annotations where a background animal in one annotation matches the foreground animal in the other — when matching.

- **Humpbacks**. The humpback dataset was collected by FlukeBook [15] over nearly 15 years. Images were contributed by both marine and citizen scientists. The original ground truth was established using both manual and automated methods that are disjoint from these techniques considered here; however our software was used to correct mistakes. The annotations in this dataset have not been manually reviewed. Some are cropped to the fluke while others encompass the entire image. Quality and viewpoint labels do not exist for this dataset.

3.5.2 Baseline experiment

This first experiment determines the accuracy of the identification algorithm using the baseline pipeline configuration. The baseline pipeline configuration uses affine invariant features oriented using the gravity vector, $K=4$ as the number of feature correspondences assigned to each query feature, and feature-based name scoring (`fmech`). In this test we control for several biases that may be introduced by carefully selecting a subset of our datasets. We only use annotations that (1) are known (*i.e.* have been assigned a name), (2) are comparable to the species primary viewpoint (*e.g.* left, front-left, and back-left for plains zebras), (3) have not been assigned a quality of "junk". Furthermore, to account for the fact that some names contain more annotations than others, we constrain our data selection such that there is only one correct exemplar in the database for each query annotation.

Of these annotations, we group them into encounters. For each encounter we sample one annotation with the highest quality. Names with only one encounter are added to the database as distractors. For the other names, we randomly sample two encounters — regardless of quality — one for the database and one to use as a query. This defines a set of query and database annotations that are separated by time, testing the ability of our system to match animals across gaps of time using only a single image per individual. The CMC curves for this baseline test are illustrated in Figure 3.14.

The results of this baseline experiment demonstrates that our algorithm is able to reliably find matching annotations in a database with many other images. The accuracy is over 60% for all species considered. Subsequent experiments will restrict our focus to Grévy's and plains zebras in order to investigate detailed parameter choices of the ranking

algorithm and alternative ranking algorithms.

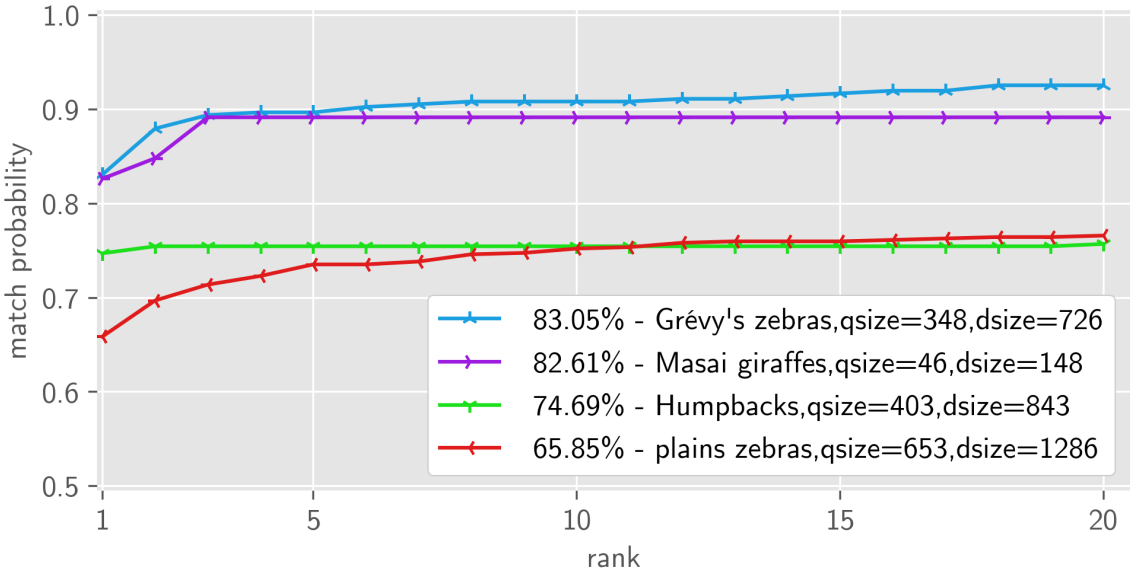


Figure 3.14: **Baseline experiment** The baseline experiment is a high-level indicator of the ranking accuracy of each species. We measure ranking accuracy using a single query and database annotation — selected from different encounters — per individual. The number of query annotations (`qsize`) and database annotations (`dsize`) are given for each species in the legend.

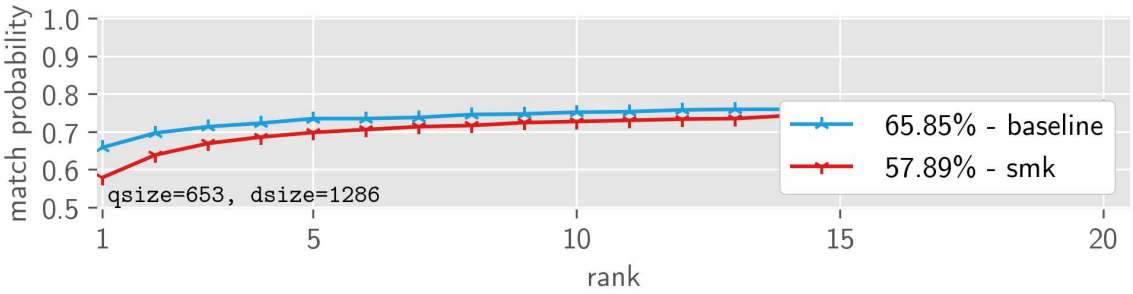
3.5.3 SMK as an alternative

Before we investigate the parameter choices of the LNBNN ranking algorithm, we briefly evaluate the performance of an alternative ranking algorithm, namely VLAD-flavored SMK. The SMK algorithm is a vocabulary-based algorithm that is representative of more traditional approaches to instance recognition problems. In contrast to the raw descriptors used in LNBNN, SMK assigns each descriptor to a visual word and builds a weighted histogram of visual words (accumulated residual vectors in the case of VLAD) to represent each annotation as a sparse fixed length vector. We have experimented with several configurations of VLAD and report our best results here.

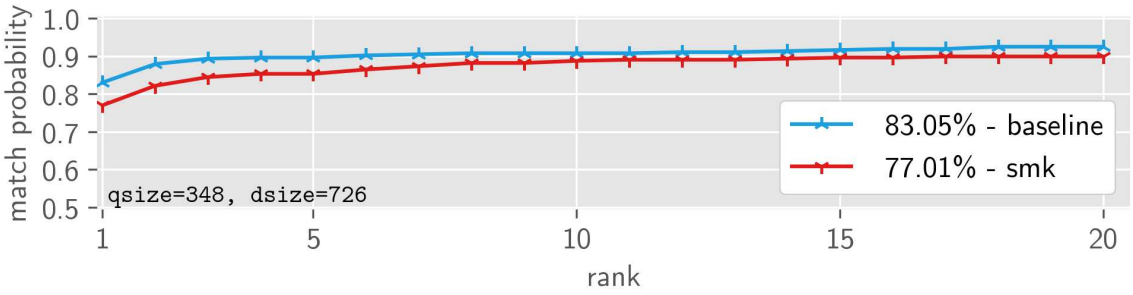
In our SMK implementation, we pre-trained an 8000 word vocabulary using mini-batch k-means on the stacked descriptors from all database annotations. Note that typically the vocabulary is trained using a disjoint external dataset in order to prevent over-fitting. However, we naïvely train using the database annotations to be indexed, understanding that this will inflate the accuracy measurements. Each word in the vocabulary

is weighted with its inverse document frequency. We use the vocabulary to compute an inverted index that maps each visual word to annotations containing that word in the database. Initial feature correspondences for a descriptor are computed using single assignment to a visual word and then creating a correspondence to every feature in that word's inverted index. We use spatial verification to filter spatially invalid correspondences, and re-score the remaining matches.

The results of the SMK experiment are illustrated in Figure 3.16. The query and database annotations are the same in each experiment. Despite the bias in the SMK vocabulary, our measurements show that LNBNN provides more accurate rankings. For plains zebra's there is a difference of 8% in the number of correct matches at rank 1, and for Grévy's zebras the difference is 6%.



(a) Plains zebras



(b) Grévy's zebras

Figure 3.16: **SMK experiment** In this experiment we compare the (VLAD based) SMK algorithm to our LNBNN ranking algorithm. The results demonstrate that LNBNN outperforms the ranking accuracy of SMK. The number of query/-database annotations (`qsize` / `dsize`) are shown in the lower left.

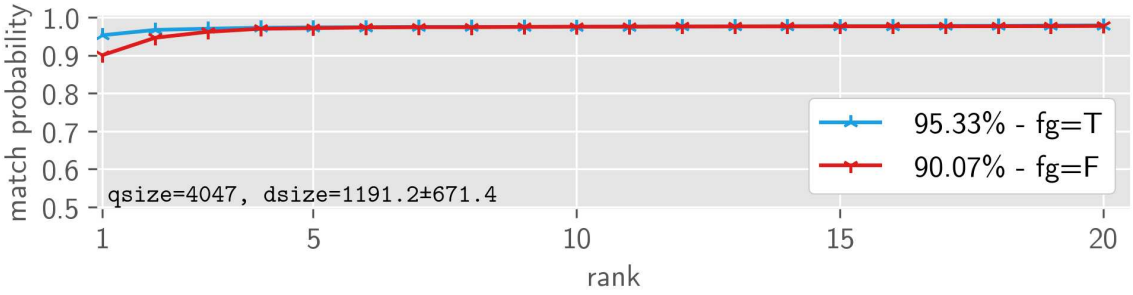
3.5.4 Foregroundness experiment

In this experiment we test the effect of our foregroundness weights — weighting the score of each features correspondence with a foregroundness weight — on identification accuracy. When foregroundness is enabled (`fg=T`), each feature correspondence is weighted using a foregroundness measure learned using a deep convolutional neural network [17]. When disabled (`fg=F`), the weight of every correspondence effectively becomes 1.

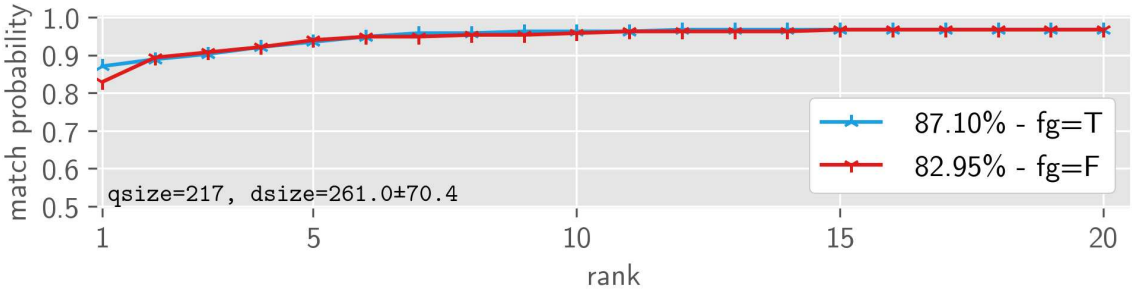
Running this experiment with using the query / database sample as outlined in the baseline experiment does not result in a noticeable difference in scores because the purpose of the foregroundness measure is to down weight matches between scenery objects (*e.g.* trees, grass, bushes) that appear in multiple annotations. The baseline database sample contains only a single image from each encounter and only two encounters per individual. This means that it will be unlikely for an annotation in the query set and another annotation in the database set to have a similar background.

To more clearly illustrate the effect of the foregroundness measure we use a different sampling strategy. We group all encounters by which occurrence they belong to. Annotations within the same occurrence are more likely to share background. We sample query and database annotations from within occurrences to simulate matching annotations within an encounter. We do not limit the number of exemplars in this test to ensure that annotation pairs that share common scenery exist. We perform this test over multiple occurrences and aggregate the results. Therefore, the reported database size will be an average, and the query size is the sum of all unique query annotations.

The accuracy of the foregroundness is illustrated in Figure 3.18. The results show that using foregroundness weights improves the number of correct results at rank 1 by a significant margin for both species. In the higher ranks using the `fg=T` line occasionally dips below the `fg=F` line because sometimes the foregroundness mask covers distinguishing keypoints, but this is neither significant nor common. Therefore, we find it beneficial to always include foregroundness when a trained estimator is available.



(a) Plains zebras



(b) Grévy's zebras

Figure 3.18: **Foregroundness experiment** Applying foregroundness weights to feature correspondences improves the identification accuracy at the top rank by filtering matches in scenery. This experiment was performed by matching annotations within several occurrences. Thus, in this experiment `qsize` is a sum and `dsize` is an average.

3.5.5 Invariance experiment

In this experiment we vary the feature invariance configuration. This influences the location, shape, and orientation of keypoints detected in each annotation, which in turn influences which regions in each annotation are matchable using SIFT descriptors extracted at each keypoint. The best invariance settings will be depend on properties of the data.

In our experiments we test different settings by enabling (denoted as T) or disabling (denoted as F) the parameters affine invariance (AI), and our query-side rotation heuristic (QRH). Initially we also tested rotation invariance, but found that it provided the poorest results for all datasets by a significant margin, likely because the gravity vector assumption is mostly satisfied in all images. Therefore, we exclude rotation invariance from our experiments.

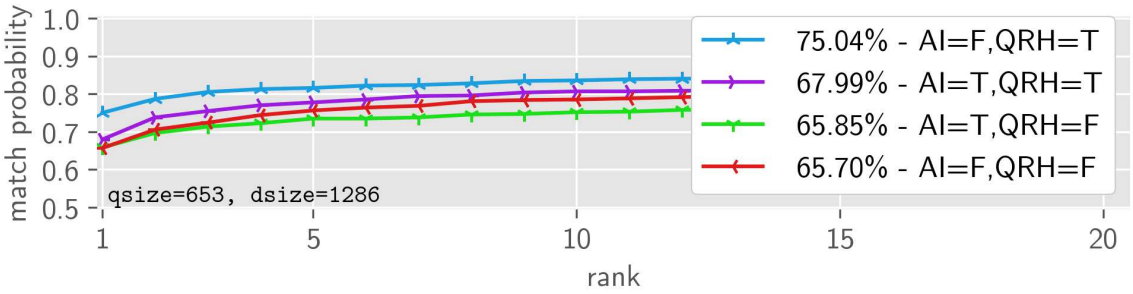
In configurations where `AI=F`, keypoints are circular with a radius defined by the

scale at which it was detected. When `AI=F`, the keypoint shape is adapted into an ellipse to normalize for small viewpoint changes. When `QRH=F`, each keypoint is assigned its orientation as normal, but when `QRH=T`, each keypoint in a query annotation is replaced by three keypoints, one rotated slightly to the left, another slightly to the right, and the last is the original keypoint. The four specific configuration that we test are outlined in the following list:

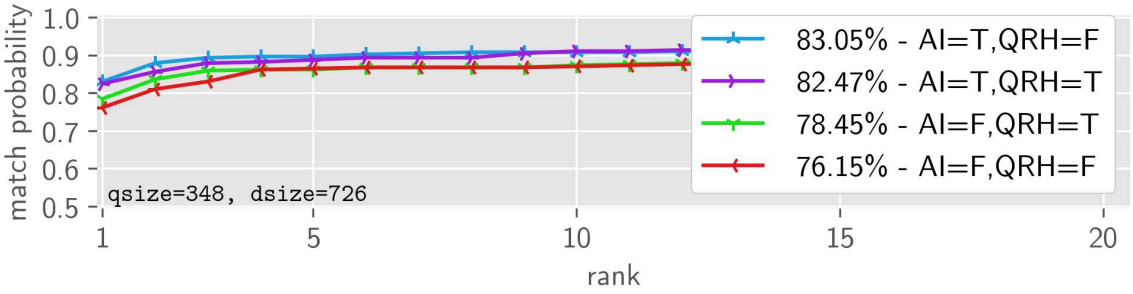
- No Invariance (`AI=F,QRH=F`): This configuration uses circular keypoints and assumes the gravity vector.
- Affine (`AI=T,QRH=F`): This is the baseline setting that assumes the gravity vector and where each feature's shape is skewed from a circle into an ellipse.
- QRH (`AI=F,QRH=T`): This is a novel invariance heuristic where each database feature assumes the gravity vector, but query feature is 3 orientations: the gravity vector and two other orientations at $\pm 15^{\circ}$ from the gravity vector. Ideally, this will allow feature correspondences to be established between features seen from slightly different orientations.
- QRH+Affine (`AI=T,QRH=T`): This is the combination of QRH and Affine.

The example in Figure 3.22 illustrates the difference between Affine and QRH features for plains and Grévy's zebras. The accuracy of the invariance experiment is shown in Figure 3.20. For plains zebras, the QRH scores are significantly better than all other invariance settings. Interestingly, affine invariance results in worse performance when QRH is on, but if the QRH is off then affine invariance improves accuracy. This suggests that the QRH better handles matching the coarse patterns seen on the plains zebras across pose and viewpoint variations than using affine invariance, which can tend to adapt itself around non-distinctive diagonal stripes. Even though affine keypoints provide more precise localization, the area they describe is often smaller than a circular keypoint. It makes sense that affine keypoints would not describe coarse features as well as circular keypoints do, because circular keypoints typically cover a larger area. The results for Grévy's zebras demonstrate similar levels of accuracy for Affine and QRH+Affine. Affine invariance seems to be the most important setting for matching Grévy's zebras. The distinctive details on Grévy's zebras are finer than plains zebras and are well captured by affine keypoints. While the QRH does improve accuracy for Grévy's zebras the density

of the distinctive keypoints means that it is less important because it is more likely two annotations will have at least one distinctive region aligned and in common.

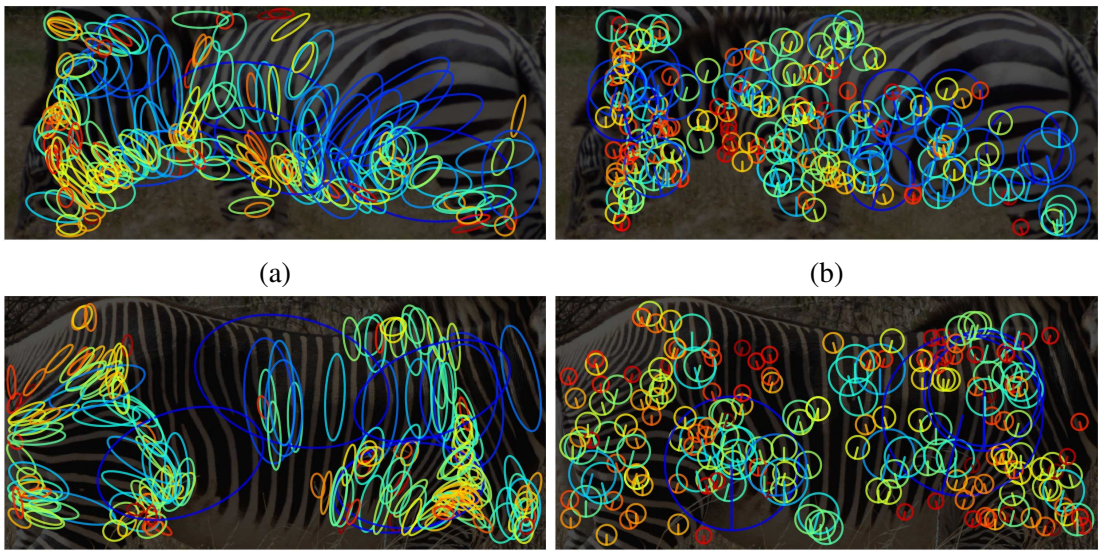


(a) Plains zebras



(b) Grévy's zebras

Figure 3.20: **Feature invariance experiment** This experiment tests the effect of affine invariance (AI) and the query-side rotation heuristic (QRH) on identification accuracy. For plains zebras circular keypoints with the QRH are the most accurate. For Grévy's zebras enabling affine invariance works the best. The number of query/database annotations (`qsize` / `dsize`) are shown in the lower left.



(a) (b)

(c) (d)

Figure 3.22: **Examples of keypoint invariance** Many affine keypoints detected on plains zebras tend to encompass only one or two stripes. The distinctive stripe patterns on Grévy's zebras are well captured by affine keypoints, whereas circular keypoints are more spread out. For visibility this figure shows a random sample of all keypoints on a darkened image. Elliptical keypoints in (3.21a and 3.21c) are colored by eccentricity and circular keypoints in (3.21b and 3.21d) are colored by scale.

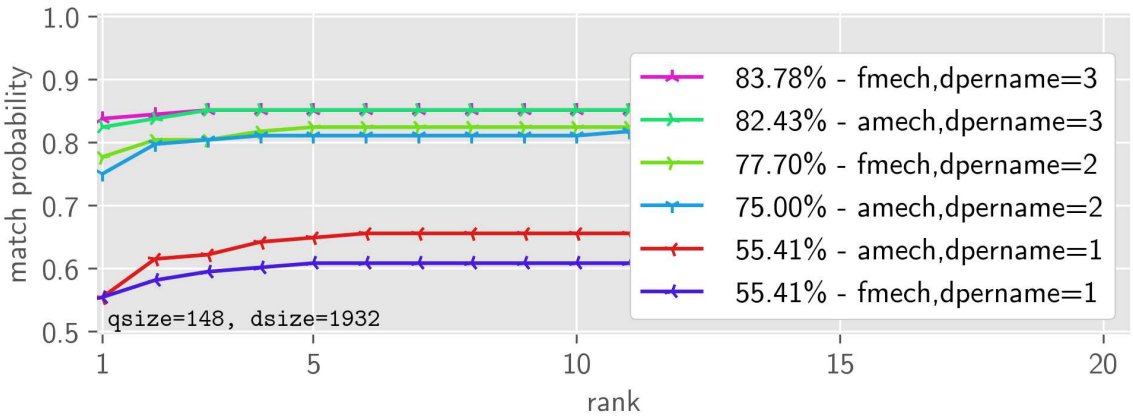
3.5.6 Scoring mechanism experiment

The purpose of the scoring mechanism is to aggregate scores of individual feature correspondences across multiple annotations into a single score for each name — *i.e.* an annotation-to-name similarity, which is analogous to the image-to-class distance used in [48]. We test the identification accuracy of the two name scoring mechanisms that were described earlier in Section 3.2.3: (1) annotation-based name scoring (denoted as `amech`) and (2) feature-based name scoring (denoted as `fmech`).

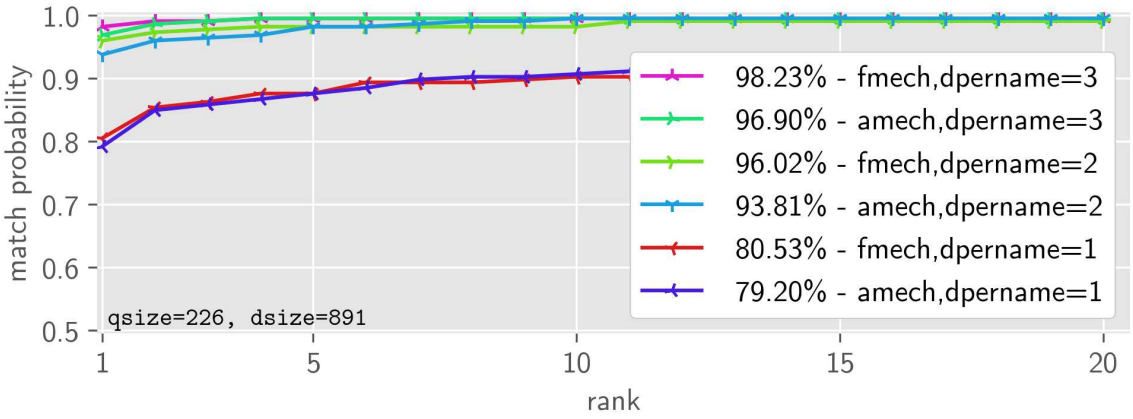
Because the scoring mechanism is meant to take advantage of multiple database annotations, we vary the number of exemplars per database name (`dpername`) between 1, 2, and 3. Varying the number of exemplars will cause each database to contain a different number of annotations. To normalize difference in database size we include additional confuser annotations (annotations that do not match any query) in the smaller databases to maintain a constant database size across experiments. Each exemplar is chosen from a separate encounter.

The accuracy of the scoring mechanism experiment is shown in Figure 3.24. The results of this test does suggest that `fmech` results in slightly more accurate ranking, but the overall difference in accuracy is relatively small (about $1 - 3\%$). Intuitively, the `fmech` scoring should produce better ranks because it can combine scores from multiple correspondences to different correct annotations. Note that when `dpername` $= 1$, the `amech` and `fmech` scores might still be different because multiple correspondences to a name which may be generated when $K > 1$ or when `QRH=T`.

Perhaps the more interesting result of this experiment is the effect of increasing the number of exemplars in the database from 1 to 2. There is a drastic improvement in ranking accuracies in both species. The accuracy of plains zebras increases by 10% and for Grévy's zebras the gain is almost 20%. It makes sense that this should be the case. If there are more examples of an individual in the database, then the probability that the query is similar to at least one of them should increase as long as there is sufficient variation. This suggests that even if a new query new annotation initially fails to rank the correct result, subsequent annotations added to the system of the same individual will be more likely to correctly match a previous annotation. As more annotations of that individual are added, the likelihood that the ranking algorithm will make a connection between all instances of that individual will increase.



(a) Plains zebras



(b) Grévy's zebras

Figure 3.24: **Name scoring experiment** There is a clear separation between identification accuracy when the number of exemplars per name is 1 compared to when it is 3. Feature based name scoring (`fmech`) is slightly more accurate than scoring using the annotation based name scoring (`amech`). The number of query /database annotations (`qsize` / `dsize`) are shown in the lower left. Database size was normalized using confusors.

3.5.7 K experiment

In this experiment we investigate the effect of K (the number of the nearest neighbors used in establishing feature correspondences, which was discussed in Section 3.2.1) on identification accuracy. We vary K between the values $1, 2, 4$, and 6. In all these experiments we set the number of normalizing neighbors to be $K^* = 1$.

Two database factors that may influence the best choice of K are the number of annotations in the database and the number of annotations per name in the database. If there are more correct matches for a query annotation it would be beneficial to allow it to

match more annotations. Likewise, if there are more overall annotations in the database, then it might be beneficial to search deeper into all the database descriptors to find the correct matches. Therefore, in addition to varying K we also vary the number exemplars per name (`dpername`) and the overall number of annotations in the database (`dsize`)

We use a protocol similar to the one used in the scoring mechanism experiment to sample databases. The difference is that we use the extra confuser annotations to vary the total number of annotations in the database. However, controlling for these factors constrains the number of annotations we can use. For Grévy's, we can vary the total database size between 476 and 774. For plains, we have more confuser annotations allowing us to test database sizes of 578 and 1650. We vary the number of exemplars per name between 1 and 2.

The results of this experiment are illustrated in Figures 3.25 and 3.26. Similar to the previous experiment, the number of exemplars per name is the most significant variable impacting accuracy. Furthermore, when there are more exemplars in the database the choice of K starts become less significant. The results also show that accuracy does slightly decrease when the database becomes larger, but magnitude of the decrease is between 1% and 3%. Interestingly, the optimal choice of K is not consistent between species when there is only one exemplar per name. For Grévy's zebras using a lower K results in better results, but for plains zebras there is a significant loss when $K = 1$ and the database size is large. This is likely due to the nature of the distinguishing patterns on the different zebras. When matching the detailed patterns of the Grévy's zebras, it is better to use a low K to reduce noise, but for coarser plains zebras patterns a low K might not find a correct match immediately. Thus, the choice of K is a trade-off between precision and recall that depends on the type of texture patterns that are being matched.

Overall the experiments on the setting of K does not yield definitive choice for this parameter. However, it appears that K only has a small influence on identification accuracy. This section does shows that the number of exemplars per annotation has a significant impact on identification accuracy.

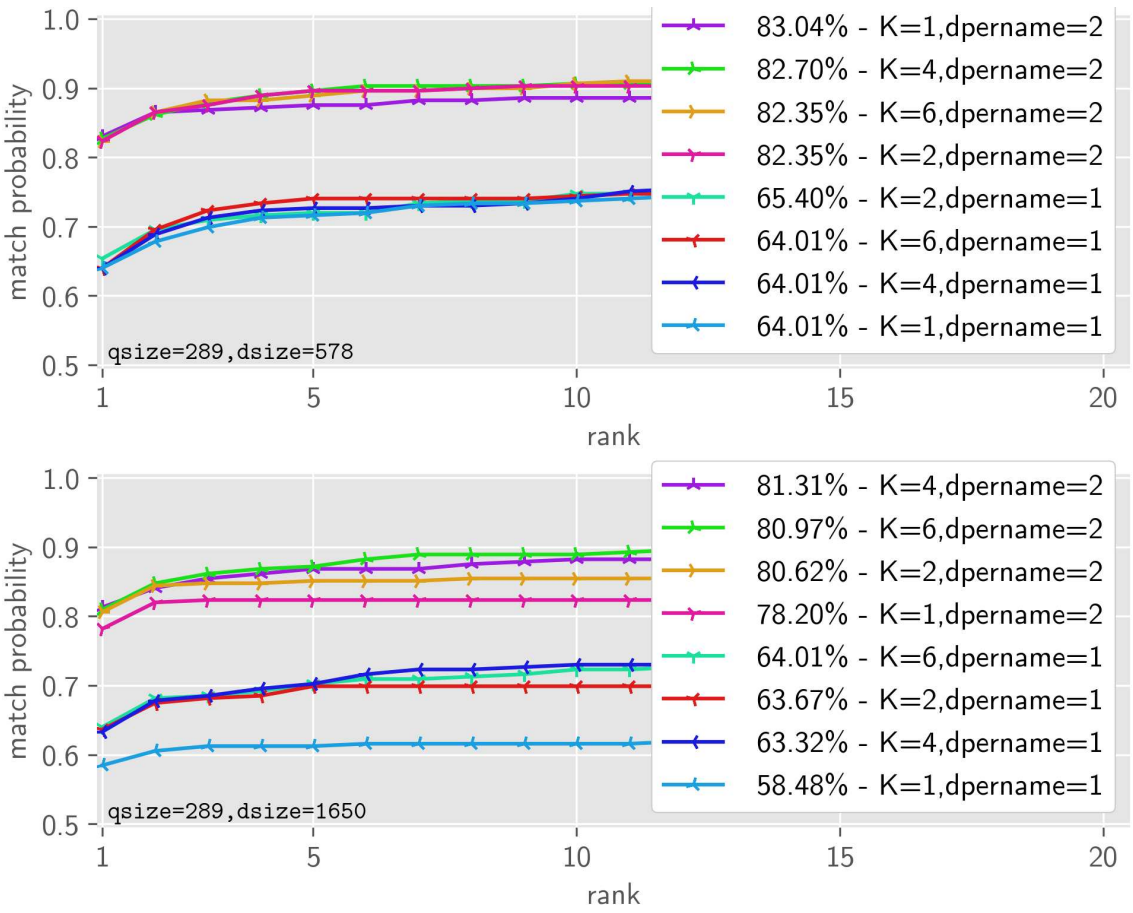


Figure 3.25: **The K experiment for plains zebras** This shows the identification accuracy for plains zebras using different values of K (the number of nearest neighbors assigned to each query feature), different numbers of exemplars (`dpername`), and different database sizes (`dsize`).

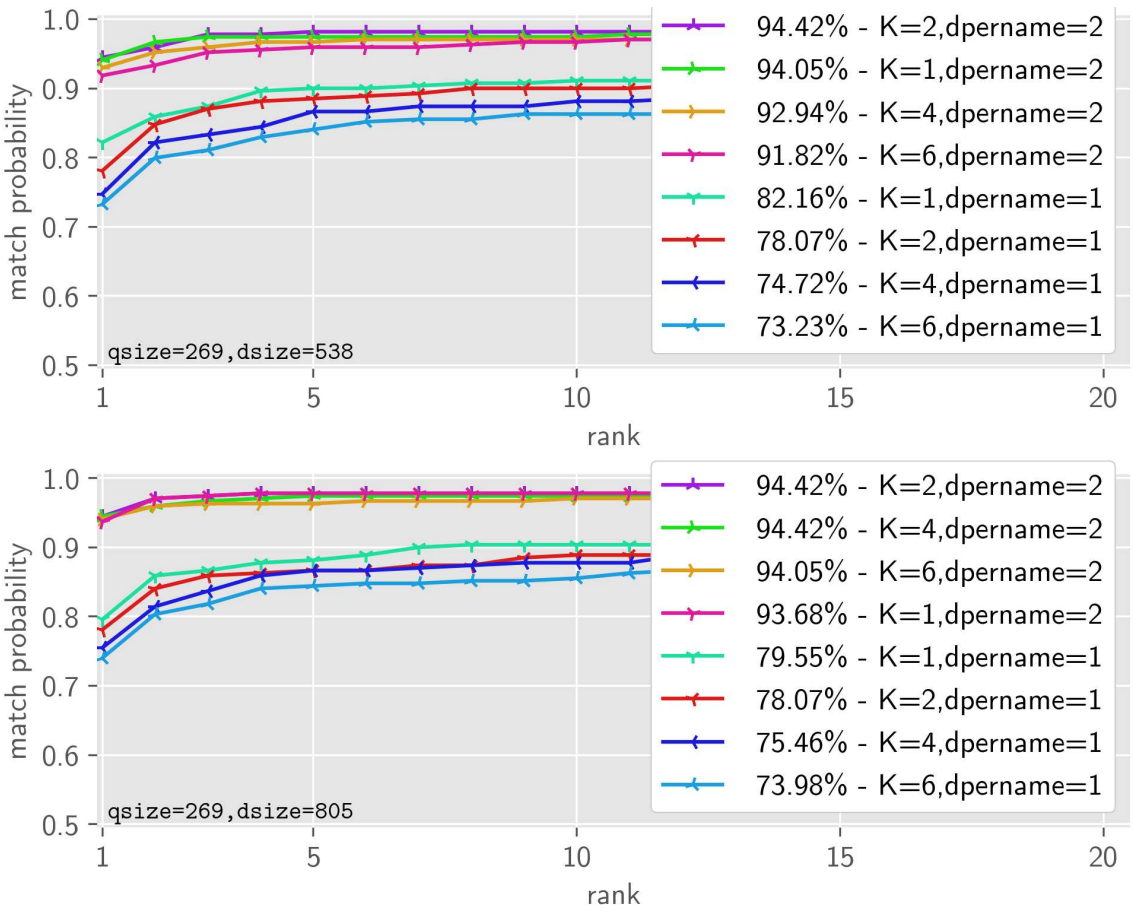


Figure 3.26: **The K experiment for Grévy's zebras** This shows the identification accuracy for Grévy's zebras using different values of K (the number of nearest neighbors assigned to each query feature), different numbers of exemplars (`dpername`), and different database sizes (`dsize`).

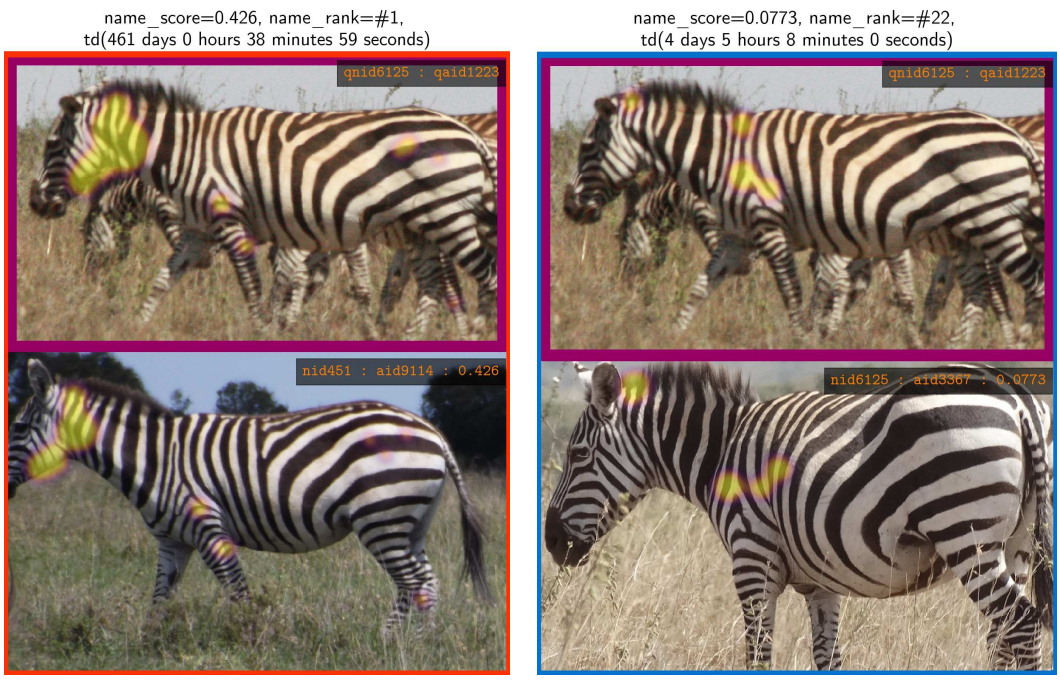
3.5.8 Failure cases

We now investigate the causes of identification failure and consider example failure cases. When investigating the cause of a failure case we consider both (1) the matches between the query annotation and the incorrect name at rank 1 and (2) the matches between the query annotation and the correct name that appears further down the ranked list. We identify the main 3 failure cases as: (1) unaligned annotations, (2) quality factors, and (3) non-primary correspondences. The remainder of this subsection defines, discusses, and provides examples of these failure cases.

3.5.8.1 Alignment

When two annotations are not aligned (ignoring translation and small scale differences), there can be significant differences in appearance that can cause inconsistency in feature localization and description. There are two major causes of alignment error: (1) viewpoint variations which cause out-of-plane rotations and (2) pose variations which can cause local non-rigid non-linear transformations of distinguishing features. These issues cause variations in feature description which renders the approximate nearest neighbor algorithm unable to establish the correct correspondence. Furthermore, non-projective transformations between annotations can cause homography-based spatial verification to discard correctly established correspondences. Failing to establish correspondences and incorrectly discarding them ultimately results in identification failure.

The example in Figure 3.28 illustrates a failure case due to a difference in viewpoint and pose. To address matching across different viewpoints and poses it helps to choose an appropriate level of feature invariance (like affine invariance and the query-side rotation heuristic), but these only work up to a point. However, in practice the animal identification problem is not a one-shot identification challenge. Given multiple annotations of an individual we expect that the matching algorithm will be able to overcome viewpoint and pose differences by matching annotations with intermediate positions.



(a) (b)

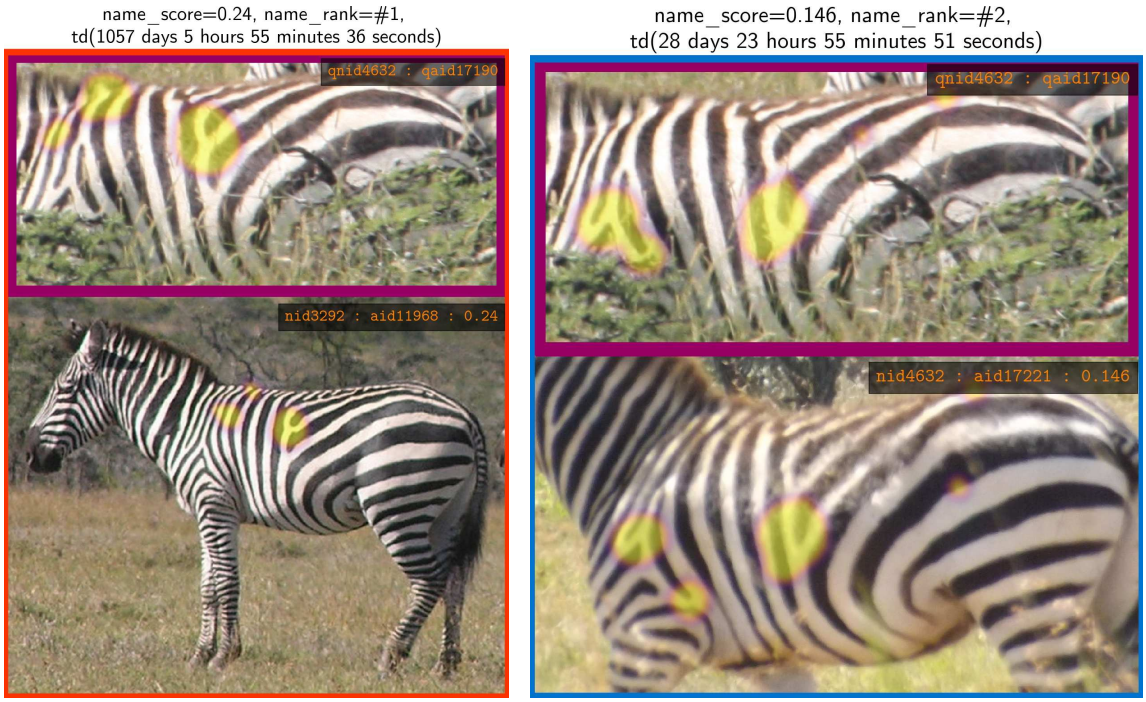
Figure 3.28: **Unaligned failure case** Due to pose and viewpoint variations, the correctly matching pair of annotations (3.27b) is returned at rank 22 while the incorrect pair of annotations (3.27a) is returned at rank 1. In the correct pair, the features on the front leg are not aligned and failed to match.

3.5.8.2 Quality factors

Factors such as low resolution, blurring, poor exposure, lighting (shadows / non-uniform illumination), and occlusion can significantly reduce the density of distinctive features on an annotation. Note that lighting and occlusion (scene quality factors) should be distinguished from the other factors (capture quality factors) because they are related to the scene itself rather than a poor capturing of that scene. Annotations with low capture quality tend to generate fewer, larger, less distinct, and distorted features. Annotations with low scene quality tend to have their distinguishing features distorted or masked by grass, tree branches, shadows, and other animals. This is a significant problem for species with relatively few distinctive features like plains zebras. The example in Figure 3.30 illustrates a failure case due to occlusion, and Figure 3.32 illustrates failure case due to low resolution.

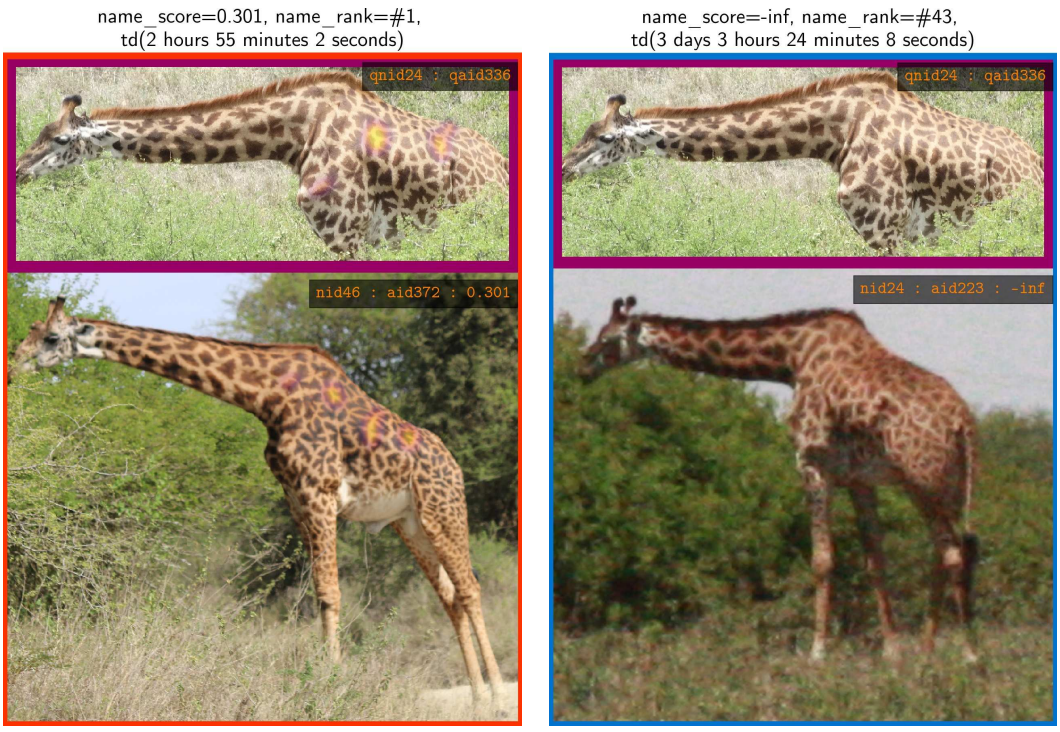
In some cases low quality annotations can be still be matched, but in the worst case all distinctive features are missing and there is no way to visually identify the individual.

Therefore, the best way to handle these annotations is either to ignore them entirely, or to first attempt to match them, but then discard them if they cannot be matched.



(a) (b)

Figure 3.30: **Occlusion failure case** The plants in the query annotation inhibit the creation of feature correspondences, causing the correct pair of annotations (3.29b) to be returned at rank 2. The incorrect pair of annotations (3.29a) at rank 1 are not the same individual even though they share similar features.



(a) (b)

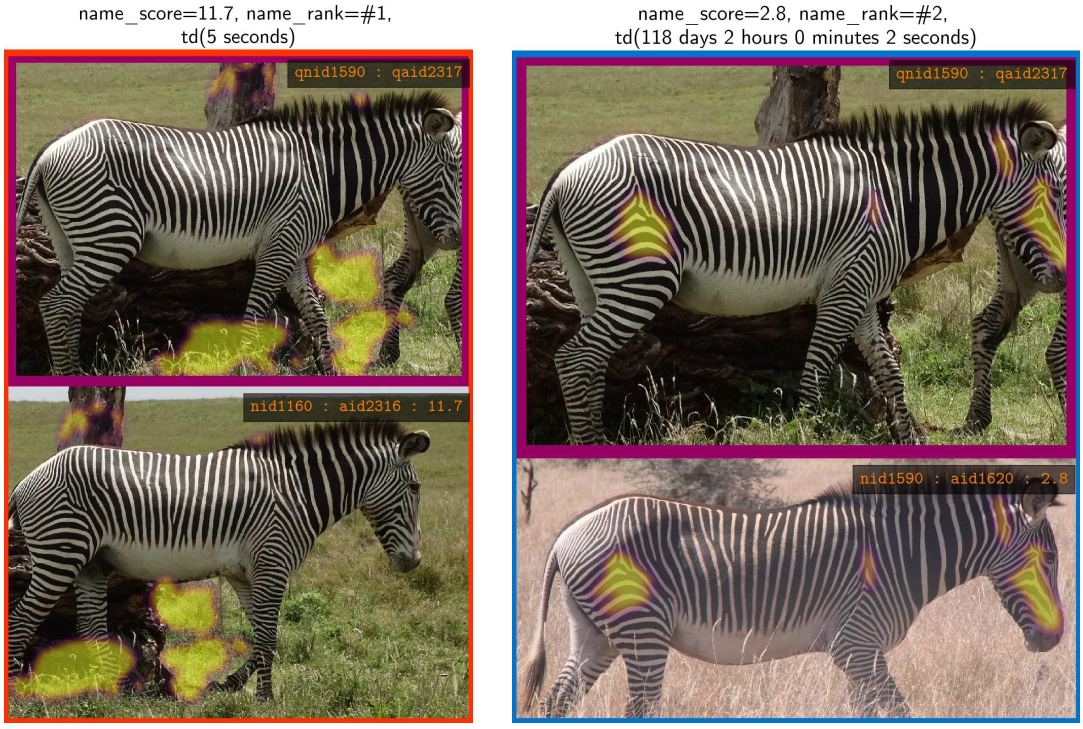
Figure 3.32: **Quality failure case** The low resolution of the database annotation causes the correct pair of annotations (3.31b) to be returned at rank 43. The incorrect pair of annotation (3.31a) did not receive a particularly high score, but it was returned at rank 1 because no feature correspondences were established to the correct match.

3.5.8.3 Non-primary correspondences

Sometimes an annotation bounding box cannot be placed tightly around an animal (this happens often for some species like giraffes), which means that other objects will appear in the background. Similarly, objects that occlude the animal will be in the foreground. Ideally, the primary animal in each annotation would be segmented, but when simply matching raw annotations non-primary correspondences may be formed. This results in the photobomb and scenery-match failure cases.

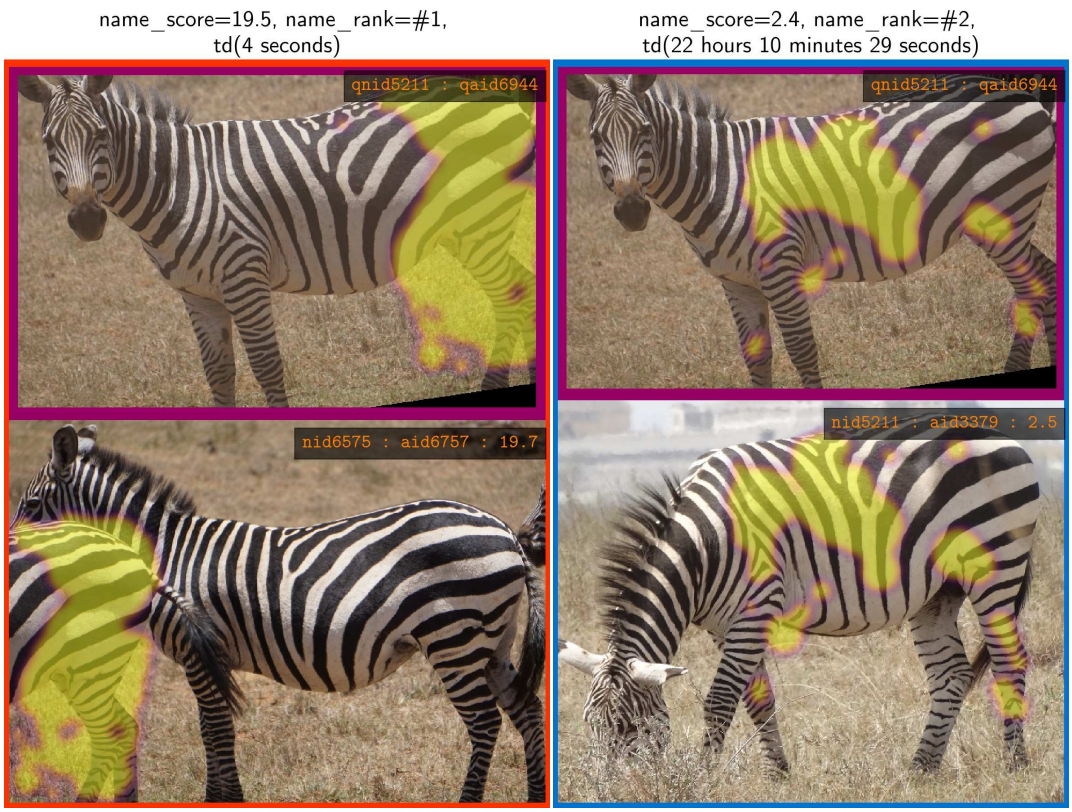
Photobombs are caused by correct correspondences to a non-primary animal (seen either in the foreground or background) in an annotation. Likewise, scenery matches are caused by matches in the background landscape. Both cases are most commonly caused by pairs of annotations with the same occurrence, but photobombs can occur over larger time deltas. The example in Figure 3.36 illustrates a failure case due to a photobombing

background animal, and Figure 3.34 illustrates a scenery match. For plains and Grévy's zebras, most scenery matches are eliminated using the foregroundness measure, but the problem remains in databases without a trained foregroundness estimator. Accounting for photobombs is a more challenging problem because a simple patch-based classifier cannot distinguish a primary feature from a secondary feature without having information about the animal identity. However, there are some patterns that photobomb matches present that we seek to take advantage of later in Section 4.3.



(a) (b)

Figure 3.34: **Scenery failure case** The incorrect pair of annotations (3.33a) was returned at rank 1 because of strong matches in the background scenery. The correct pair (3.33b) was returned at rank 2 and did not produce matches in the front leg due to pose variations. The annotations in the scenery match pair were taken seconds apart in the same location causing their backgrounds to be near duplicates. The foregroundness measure was disabled to produce this example, enabling it addresses nearly all scenery match cases.



(a) (b)

Figure 3.36: **Photobomb failure case** A photobombing animal in the foreground of the database annotation causes LNBNN to return the incorrect result (3.35a) at rank 1. The correct match (3.35b), has a significant number of matches, but there is a difference of 1 day between the pair. On the other hand, the annotations in the photobomb pair were taken within seconds of each other and therefore have much higher visual similarity.

3.5.9 Experimental conclusions

In this section we have evaluated our ranking algorithm on multiple species, compared it to an alternative ranking algorithm, and evaluated detailed parameter choices. Our experiments were performed under restrictive conditions to control for the effect of time, database size, and number of exemplars. Based on the results of these experiments we are able to make several observations and conclusions.

Our experiments with comparing the SMK and LNBNN ranking algorithm demonstrated that LNBNN achieved better ranking accuracy. LNBNN does not quantize descriptor and therefore it is able to distinguish more subtle descriptor details. Because LNBNN does not require an expensive pre-training phase, it makes it ideal to rank the

databases on the scales considered in this thesis. However, we note that SMK is more efficient on larger scales, and it may be necessary to consider when databases become very large.

In most experiments we evaluated our ranking algorithm as if it were addressing a single-shot identification problem. This was to establish the performance of the algorithm when an individual has only been seen once before. However, in practice this will not be the norm. The name scoring and K experiments demonstrated that the ranking accuracy significantly increases with the number of exemplars per database name. We will use this observation to address the challenges of matching through viewpoint and occlusion by taking advantage of multiple images of an individual in Chapter 5.

We also saw that the best choice for feature invariance is data dependent. The invariance experiment demonstrated that affine invariance produces better results for Grévy's zebras, whereas circular keypoints lead to more accurate results for plains zebras. This experiment also showed that the query-side rotation heuristic improves accuracy by adding a small amount of orientation invariance to feature localization. Likewise, the K experiment shows that identification accuracy is not significantly influenced by the choice of K for plains zebras, but for Grévy's zebras the most accurate results were obtained with $K{=}1$. This is likely because the features from plains zebras are less distinguishing than features from Grévy's zebras, hence the correct match of a plains zebra feature is less likely to be its closest neighbor. Both the choice of K and invariance settings should be evaluated on a per-dataset basis and there is likely benefit to performing identification using multiple parameter choices.

3.6 RANK-BASED IDENTIFICATION SUMMARY

In this chapter we have addressed the problem of animal identification using a computer-assisted algorithm based on LNBNN that ranks a labeled database of names by their similarity to a single query annotation. This algorithm begins by extracting local patch-based features from cropped and normalized chips. Features from database annotations are indexed for fast nearest neighbor search using a kd-tree. A mechanism based on LNBNN is used to compute a matching score for each database annotation. A shortlist of top scoring results have their feature correspondences spatially verified and then are

re-scored. We have shown how this algorithm can be applied to individual animal identification and demonstrated that in a majority of cases the correct match is ranked first by our algorithm for plains zebras, Grévy's zebras, Masai giraffes, and humpback whales.

Because we have used the algorithm to curate the ground truth, we do not claim the reported accuracies in our experiments to be quantitatively absolute. However, qualitative evidence for the algorithm's overall success is provided by the facts that (1) we were able to use the algorithm to identify a significant number of individuals from different species and (2) our approach provides more accurate rankings than standard instance recognition techniques. We therefore make the conclusion that the algorithm is effective at identifying medium to high quality images of animals with distinguishing patterns when taken from similar viewpoints.

While we have demonstrated that the ranking algorithm accurately ranks correct matches when they exist, there are several limitations to this approach.

(1) All results must be manually verified, which can be time-consuming.

(2) There is no mechanism for recovering from errors once they occur.

(3) There is no mechanism to determine when identification is complete.

In the following chapters we seek to address these issues. In Chapter 4 we introduce an algorithm to make automatic decisions based on results from this algorithm, and in Chapter 5 we introduce a graph-based framework that determines identification confidence and introduces error recovery mechanisms.

4. PAIRWISE VERIFICATION

In this chapter we consider the problem of verifying if two annotations are from the same animal or from different animals. By addressing this problem we improve upon the ranking algorithm from Chapter 3 — which ranks the names in a database based on similarity to a query — by making semi-automatic decisions about results returned in the ranked lists. The algorithms introduced in this chapter will assign a confidence to results in the ranked list, and any pair above a confidence threshold can be automatically reviewed. We will demonstrate that our decision algorithms can significantly reduce the number of manual interactions required to identify all individuals in an unlabeled set of annotations.

To make semi-automatic decisions up to a specified confidence we develop a *pairwise probabilistic classifier* that predicts a probability distribution over a set of events given two annotations (typically a query annotation and one of its top results in a ranked list). Given only the information in two annotations, there are three possible decisions that can be made. A pair of annotations is either:

(1) incomparable — the annotations are not visually comparable,

(2) positive — the annotations are visually comparable and the same individual, or

(3) negative — the annotations are visually comparable and different individuals.

Two annotations can be incomparable if the annotations show different parts or sides of an animal, or if the distinguishing information on an animal is obscured or occluded. The positive and negative states each require distinguishing information to be present. These mutually exclusive "match-states" are illustrated in Figure 4.2. The multi-label classifier then predicts the probability of each of the three states, with the probabilities necessarily summing to 1.

To construct a pairwise probabilistic classifier we turn towards supervised machine learning. This requires that we: (1) determine a set of labeled annotation pairs for training, (2) construct a fixed-length feature vector to represent a pair of annotations, and (3) choose a probabilistic learning algorithm. The first requirement can be satisfied by carefully selecting representative annotations pairs, and the last requirement is satisfied by many pre-existing algorithms (*e.g.* random forests and neural networks). The sec-

(a) Positive (b) Negative (c) Incomparable

Figure 4.2: **Match-state example** Examples of positive (4.1a), negative (4.1b), and incomparable (4.1c) pairs of annotations. Local feature correspondences are superimposed over the pairs.

ond requirement — constructing an appropriate fixed-length feature vector — is the most challenging. Given enough training data and a technique to align the animals in two annotations, using image data with a Siamese or triplicate network [41], [112] might be appropriate, but without both of these pre-conditions we must turn towards more traditional methods. Recall from Section 3.1 that our annotation representation is an unordered bag-of-features, which cannot be directly fed to most learning algorithms. Therefore, we develop a method for constructing a fixed length *pairwise feature vector* for a pair of annotations. This novel feature vector will take into account local matching information as well as more global information such as GPS and viewpoint. A collection of these features from multiple labeled annotation pairs is used to fit a random forest [250] which implements our pairwise classifier. We choose to use random forest classifiers, in part because they are fast to train, insensitive to feature scaling, robust to overfitting, and naturally output probabilities in a multiclass setting, and in part because they can handle (and potentially take advantage of) missing data — *i.e*. nan values in feature vectors — using the "separate class" method [251].

A final concern investigated in this chapter is the issue of image challenges that may confound the match-state pairwise classifier. Recall from Section 3.5.8, photobombs — pairs of annotations where feature correspondences are caused by a secondary animal — are the most notable cause of such a challenge. Photobombs appear very similar to

positive matches, and this similarity could confuse the match-state classifier. However, because photobombs are inherently a pairwise property between annotations, it should be possible to learn a separate classifier explicitly tasked with the challenge. Therefore, we also learn a photobomb classifier using the same sort of pairwise feature vector and random forest classifier. This supporting classifier will allow us to increase the accuracy of our identification by restricting automatic classification to pairs where the decision is straightforward.

This outline of this chapter is as follows. Section 4.1 details the construction of the feature vector that we use as input to the pairwise classifier. Section 4.2 describes the process of collecting training data and learning the match-state pairwise classifier. Section 4.3 extends this approach to predict secondary attributes (*e.g.* if a pair is a photobomb) beyond just the matching state. Section 4.4 presents a set of experiments that evaluate the pairwise classifier. Section 4.5 summarizes and concludes this chapter.

4.1 CONSTRUCTING THE PAIRWISE FEATURE VECTOR

In order to use the random forest learning algorithm to address the problem of pairwise verification, we must construct a feature vector to representing a pair of annotations. This feature vector must contain enough information to differentiate between the positive, negative, and incomparable classes. In contrast to the unordered bag-of-features used to represent an annotation, the dimensions in this feature vector must be ordered and each dimension should correspond to a specific measurement. In practice this means that the feature vector must be ordered and have a fixed length.

We construct this feature vector to contain both global and local information. Global information is high level metadata about the annotation pair and includes non-visual information. The local information aggregates statistics about feature correspondences between the two annotations. The local and global vectors are constructed separately and then concatenated to form the final pairwise feature vector. The remainder of this section discusses the construction of these vectors.

4.1.1 The global feature vector

The global feature vector contains information that will allow the classifier to take advantage of semantic labels and non-visual attributes of our data to solve the verification problem. Semantic labels such as quality and viewpoint are derived from visual information and can provide abstract knowledge to help the classifier make a decision. Non-visual attributes such as GPS and timestamp can be extracted from EXIF metadata and may help determine facts not discernible from visual data alone. The global feature vector is derived from the following "unary" attributes extracted from each annotation individually:

(1) Timestamp, represented in POSIX format as a float.
(2) GPS latitude and longitude, represented in radians as two floats.
(3) Viewpoint classification label, given as a categorical integer ranging from 1 to 8.
(4) Quality classification label, given as a categorical integer ranging from 1 to 5.

We gather the GPS and timestamp attributes from image EXIF data, and the viewpoint and quality labels are outputs of the deep classifiers previously discussed in Section 1.3.2. The GPS and timestamp attribute inform the classifier of when it is not possible for two annotations to match (*e.g.* when a pair of annotations is close in time but far in space) and when two annotations were taken around the sample place and time. Pairs of annotations taken around the same place and time tend to have a higher similarity and are more likely to contain photobombs and scenery matches. The viewpoint and quality attributes should help the classifier predict when pairs of annotations are not comparable — forcing there to be stronger evidence to form a match, such as strong correspondences on a face turned toward the camera in both a left and right side view. An example illustrating such a case — where two annotations with different viewpoints are a positive match — is illustrated in Figure 4.3.

These four "unary" attributes are gathered for each annotation. Thus, for each attribute we have two measurements, one for each annotation. However, we do not use them directly because the ordering of the annotations in each pair is arbitrary. For each unary attribute, we either ignore it (as in the case of GPS and time) or record the minimum of the two values in one feature dimension and the maximum in another (as is done with viewpoint and quality). This results in 4 unary measurements, 2 for viewpoint and 2 for

Figure 4.3: **A comparable pair with different viewpoints** Even though this pair has different viewpoints, it is positive and comparable because we can establish a distinctive correspondence in the face.

quality. These measurements are appended to the front of the global feature vector.

The remaining dimensions of the global feature vector are constructed by encoding relationships between pairs of unary attributes using distance measurements. In the case of GPS coordinates we use the haversine distance (as detailed in Appendix A). In the case of viewpoint we use a cyclic absolute difference — *i.e*. the distance between viewpoints v_1 and v_2 is $\min(|v_1 - v_2|, 8 - |v_1 - v_2|)$. For quality and time we simply use the absolute difference between their values. This results in 4 pairwise measurements, one for each global attribute. Lastly, we include the "speed" of the pair, which is the GPS-distance divided by the time-delta. Thus, the total number of global measurements is $4+4+1 = 9$.

In the event that an attribute is not provided or not known (*e.g*. the EXIF data is missing), then a measurement cannot be made, so a nan value is recorded instead. To apply random forests learning, these nan values must be handled by either modifying the learning algorithm or replacing them with a number. Ding and Simonoff investigate several methods for handling missing data in [251], and they conclude that the best choice is application dependent. For our application we choose the "separate class" method because their experiments demonstrate that it performs the best when nan values are in both the training and testing data, which is the case for our data.

In addition to being the best fit for our application, the separate class method is

simple. The idea is to replace all nan measurements with either an extremely large or extremely small number. The choice of large or small is made independently at each node in the decision tree, depending on which choice is best. In this way the nan values are essentially treated as a separate category because a test can always be chosen that separates the measured and unmeasured data. This has several effects. In the case that a nan measurement in a feature dimension is informative (*e.g.* if images without timestamps are less likely to match other annotations), the random forest can take advantage of that dimension. However, in the more likely case that the same nan measurement is uninformative, the dimension can still be used, but it is penalized proportional to the fraction of samples where it takes a nan value. This captures the idea that a feature dimension is less likely to be informative if it cannot be measured reliably. However, if that feature dimension is highly informative for samples where it has a numeric value, then a node in a decision tree can still make use of it, and the samples with nan values can be split by nodes deeper in the tree.

4.1.2 The local feature vector

The local feature vector distills two orderless bag-of-features representations into a fixed length vector containing matching information about a pair of annotations. Three basic steps are needed to construct the local feature vector. First we determine feature correspondences between the two annotations. Then for each correspondence we make several measurements (*e.g.* descriptor distance and spatial position). Finally, we aggregate these measurements over all correspondences using summary statistics (*e.g.* mean, sum, std). Later we augment this basic scheme by constructing multiple sets of feature correspondences. Thus, the total length of the feature vector is the number of measurements times the number of summary statistics times the number of ways feature correspondences are established.

4.1.2.1 Establishing feature correspondences

To determine feature correspondences between two annotations, $\mathcal{X}$ and $\mathcal{Y}$, we use what we refer to as a one-vs-one matching algorithm. Each annotation's descriptors are indexed for fast nearest neighbor search [29]. Keypoint correspondences are formed by

searching for the reciprocal nearest neighbors between annotation descriptors [252]. For each feature in each correspondence, the next nearest neighbor is used as a normalizer for Lowe's ratio test [22]. Because matching is symmetric, each feature correspondence is associated with two normalizing neighbors. The feature / normalizer pair with the minimum descriptor distance is used as a normalizing pair. If the ratio of the descriptor distance between a correspondence to the distance between the normalizing pair is above a threshold, the correspondence is regarded as non-distinct and removed. For the simplicity of the description we consider just one ratio threshold for now, but later we will describe this process using multiple thresholds. Spatial verification [32] is applied to further refine the correspondences. This one-vs-one matching algorithm results in a richer set of correspondences between annotations than would be found using the LNBNN ranking algorithm.

4.1.2.2 Local measurements

After the one-vs-one matching stage, several measurements are made for each feature correspondence. Before describing these measurements, it will be useful to set up some notation. Given two annotations $\mathcal{X}$ and $\mathcal{Y}$, consider a feature correspondence between two features i and j with descriptors $\mathbf{d}_i$ and $\mathbf{d}_j$. Let $\mathbf{d}_i^*$ be the normalizer for i, and let $\mathbf{d}_j^*$ be the normalizer for j. Note that while i is from $\mathcal{X}$, its normalizer, $\mathbf{d}_i^*$, is a descriptor from $\mathcal{Y}$. The converse is true for j. Let $c \in \{i, j\}$ indicate which feature / normalizer pair is used in the ratio test. Thus, $c = \underset{c \in i,j}{\operatorname{argmin}} ||\mathbf{d}_c - \mathbf{d}_c^*||$. Given these definitions, the measurements we consider are:

- Foregroundness score: This is the geometric mean of the features' foregroundness measures, $\sqrt{w_i w_j}$. This adds 1 measurement, denoted as `fgweight`, for each correspondence.
- Correspondence distance: This is the Euclidean distance between the corresponding descriptors, $||\mathbf{d}_i - \mathbf{d}_j||/Z$. This serves as a measure of visual similarity between the features. (Recall $Z{=}\sqrt{2}$ for SIFT descriptors). This adds 1 measurement, denoted as `match_dist`, for each correspondence.
- Normalizer distance: This is the distance between a matching descriptor and the normalizing descriptor, $||\mathbf{d}_c - \mathbf{d}_c^*||/Z$. This serves as a measure of visual distinc-

tiveness of the features. We also include a weighted version of this measurement by multiplying it with the foregroundness score. This adds 2 measurements, denoted as `norm_dist` and `weighted_norm_dist`, for each correspondence.

- Ratio score: This is the one minus the ratio of the correspondence distance to the normalizer distance, $1 - ||\mathbf{d}_i - \mathbf{d}_j||/||\mathbf{d}_c - \mathbf{d}_c^*||$. This combines both similarity and distinctiveness. Note that this is one minus the measure used to filter correspondences in the ratio test. We perform this subtraction in order to obtain a score that varies directly with distinctiveness — *i.e*. the ratio score can increase by decreasing the visual difference or increasing the distinctiveness. We also include a weighted version of this measurement by multiplying it with the foregroundness score. This adds 2 measurements, denoted as `ratio_score` and `weighted_ratio_score`, for each correspondence.
- Spatial verification error:

  This is the error in location, scale, and orientation, $(\Delta\mathbf{x}_{i,j}, \Delta\sigma_{i,j}, \Delta\theta_{i,j})$, as measured using Equation (3.18). This adds 3 measurements, denoted as `sver_err_xy`, `sver_err_scale`, and `sver_err_ori`, for each correspondence. These measurements carry information about the alignment between the feature correspondences.
- Keypoint relative locations:

  These are the xy-locations of the keypoints divided by the width and height of the annotation $x_i/w_{\mathcal{X}}, y_i/h_{\mathcal{X}}$ and $x_i/w_{\mathcal{Y}}, y_j/h_{\mathcal{Y}}$. These measurements carry information about the spatial distribution of the feature correspondences. This will be useful for determining if a pair of annotations is a photobomb because often the photobombing animal is not in the center of the annotation. This adds 4 measurements, denoted as `norm_x1`, `norm_y1`, `norm_x2`, and `norm_y2`, for each correspondence. Note that unlike the global quality and viewpoint measures, we do not make an effort to account for the arbitrary ordering of annotations when recording these local features. This is to preserve the association between the spatial dimensions of each annotation. The same is true for the next feature.
- Keypoint scales:

  These are the keypoint scale parameters σ_i and σ_j, as measured using Equation (3.4). The scales indicate the size of each keypoint with respect to its annotation. This

may be useful for disregarding matches between large coarsely described features that do not carry a significant amount of individually distinctive information. This adds 2 measurements, denoted as `scale1` and `scale2`, for each correspondence.

4.1.2.3 Summary statistics

Once these 15 measurements have been made for each keypoint correspondence we summarize them using summary statistics. We consider the sum, median, mean, and standard deviation over all correspondences. We have also considered taking values from individual correspondences based on rankings and percentiles with respect to a particular attribute (*e.g.* ratio score), however we found that these did not improve the performance of our classifiers. In practice the summary statistics work quite well. The resulting measurements are stacked to form the local feature vector. This results in $15 \times 4 = 60$ measurements. A final step we have found useful is to append an extra dimension simply indicating the total number of feature correspondences. So, in total there are 61 summary statistics computed for a set of feature correspondences.

4.1.2.4 Multiple ratio thresholds

As previously noted we establish multiple groups of feature correspondences for different threshold values of the ratio test. We do this because we have observed that some positive annotation pairs had all of their correspondences filtered by the ratio test. However, when we increase the ratio threshold, the overall classification performance decreases. Therefore, we include both small and large thresholds to allow the classifier to have access to both types of information. Including larger threshold values helps ensure that most pairs generate at least a few correspondences, while smaller threshold values capture the information in highly distinctive correspondences. Overall, this softens the impact of the ratio test's binary threshold and adds robustness to viewpoint and pose variations that may cause correspondences to appear slightly less distinctive.

The details of this process are as follows: Once we have assigned feature correspondences using symmetric nearest neighbors, we create a group of correspondences for each ratio value. The members of each group are all correspondences with a ratio score less than that value. Each group is then spatially verified, and the union of the groups is

the final set of correspondences. When measuring spatial verification errors, each keypoint may be associated with multiple values. Therefore, we use the minimum spatial verification error over all values of the ratio threshold. The local feature vector is constructed by applying summary statistics to each of these groups independently and then concatenating the results. Thus, the size of the local feature vector is multiplied by the number of ratio thresholds used.

While using multiple values of the ratio test can further enrich the pairwise local feature vector, there are two trade-offs that must be taken into account. First, spatial verification must be run multiple times, which noticeably increases computation time. Second, the resulting size of the feature vector is larger, which can make learning more difficult due to the curse of dimensionality. Therefore, in our implementation we choose to use only two ratio threshold values of 0.625 and 0.8. Thus, the total number of local measurements is $2 \times 61 = 122$.

4.1.2.5 Additional notes

We have found that, for some species like plains zebras, it is important to use the keypoint orientation heuristic described in Section 3.1 when computing one-vs-one matches. This heuristic causes each keypoint to extract 3 descriptors instead of 1. In this case we should not use the second nearest neighbor as the normalizer for the ratio test, because the augmented keypoints may have similar descriptors. We account for this by using the 3^{rd} nearest neighbor as the normalizer instead.

4.1.3 The final pairwise feature vector

The final pairwise feature vector is constructed by concatenating the local and the global vector. This results in a 131 dimensional vector containing information that a learning algorithm can use to predict if a pair of annotations is positive, negative, or incomparable. Of these dimensions, 9 are from global measurements and 122 are from local measurements. The example in Figure 4.4 illustrates part of a final feature vector.

```
OrderedDict([('global(qual_min)',     3),
             ('global(qual_max)',     nan),
             ('global(qual_delta)',   nan),
             ('global(gps_delta)',    5.79),
             ('len(matches[ratio < .8])',         20),
             ('sum(ratio_score[ratio < .8])', 10.05),
             ('mean(ratio_score[ratio < .8])', 0.50),
             ('std(ratio_score[ratio < .8])',  0.09)])
```

Figure 4.4: **A pairwise feature vector** This is an example of a small pairwise feature vector containing local and global information. The feature vector we use in practice contains 131 dimensions. Note the summary statistics in this example are all computed for correspondences with a ratio value that is less than 0.8.

4.2 LEARNING THE MATCH-STATE CLASSIFIER

Having defined the pairwise feature vector there are two remaining steps to constructing the pairwise classifier. We must: (1) choose a probabilistic learning algorithm, and (2) select a sample of labeled training data. We have previously stated that we will use the random forest [250] as our probabilistic classifier. Therefore, we briefly review the details of random forest learning and prediction. Then we describe the sampling procedure used to generate a dataset of labeled annotation pairs.

4.2.1 The random forest learning algorithm

The random forest learning algorithm [250] is well understood, so we only provide a brief overview. A random forest is constructed by learning a forest of decision trees. Learning begins by initializing each decision tree as a single node. Each root node is assigned a random sample of the training data with replacement, and then a recursive algorithm is used to grow each root node into a decision tree. Each iteration of the recursive algorithm is given a leaf node, and will choose a test to split the training data at the node into two child nodes. The test is constructed by first choosing a random subset of feature dimensions. We then find choose a dimension and threshold to maximize the decrease in class-label entropy. Note that when using the "separate class" method, the algorithm tests placing samples with missing data on both the left and right side of the split. The algorithm is then recursively executed on the right and left node until a leaf is assigned fewer than a minimum number of training examples. To select a test for a node, the number of

candidate features dimensions we choose is the square root of the total number of feature dimensions. Each decision tree predicts a probability distribution over all classes for a testing example by descending the tree, choosing left or right based on the test chosen at the node until it reaches a leaf node. The predicted probabilities are the proportions of training class-labels at that leaf node. The probability prediction of the random forest is the average of the probabilities predicted by all decision trees. We use the random forest implementation provided by Scikit Learn [253]. To choose hyper-parameters, we preform a grid search. We find that it works best to use 256 decision trees, and to stop branch growth once a leaf node contains 5 or fewer training samples. For other parameters we use the implementation defaults.

4.2.2 Sampling a labeled dataset of annotation pairs

Now that we have chosen a learning algorithm, the last remaining step is the selection of training data and generation of labels. Recall that the purpose of our classifier is to output a probability distribution over three labels: positive, negative and incomparable. Given a pair of annotations we need to assign one of these three labels using ground truth data. In recent versions of our system, this ground truth label is stored along with each unordered pair of annotations that has been manually reviewed, but because this is a new feature, only a few pairs have been assigned an explicit three-state label. Therefore, we must make use of the name and viewpoint labels assigned to each annotation by previous versions of the system. This allows us to determine if an annotation pair shows the same or different animals, but it does not allow us to determine if the pair is comparable. To account for this we use heuristics to assign the incomparable label using the viewpoints, and if either annotation is not assigned a viewpoint it is assumed that they are comparable because most images in our datasets are taken from a consistent viewpoint (*i.e.* collection events were designed to reduce incomparability).

Given a pair of annotations, a training label is assigned as follows. First, if an explicit three-state label exists, return it. Otherwise, we must heuristically decide if the pair is comparable based on viewpoint information. If the heuristics determine that a pair is not comparable, then return incomparable. In all other cases return positive if the annotations share a name label and negative if they do not.

In order to select pairs from our ground truth dataset, we sample representative pairs of annotations guided by the principle of selecting examples that exhibit a range of difficulties [109] (*e.g.* hard-negatives and moderate positives). We use the LNBNN ranking algorithm to estimate how easy or difficult it might be to predict the match-state of a pair. Pairs with higher LNBNN scores will be easier to classify for positive examples and will be more difficult for negative examples, and lower scores will have the reverse effect.

Specifically, to sample a dataset for learning, we first rank the database for each query image using the ranking algorithm. We partition the ranked lists into two parts: a list of correct matches and a list of incorrect matches. We select annotations randomly from the top, middle, and bottom of each list. For positive examples we select 4 from the top, 2 from the middle, 2 from the bottom, and 2 randomly. For negative examples we select 3 from the top, 2 from the middle, 1 from the bottom, and 2 randomly. If there are not enough examples to do this, then all are taken. We include all pairs explicitly labeled as incomparable because there are only a few such examples. If this was not the case, then we would include an additional partition for incomparable cases.

4.3 SECONDARY CLASSIFIER TO ADDRESS PHOTOBOMBS

It is useful to augment the primary match-state pairwise classifier with a secondary classifier able to determine if a pair of annotations contains information that might confuse the main classifier. These confusing annotation pairs should not be considered for automatic review. One of the most challenging of these secondary states is one that we refer to as a photobomb. A pair of annotations is a photobomb if a secondary animal in one annotation matches an animal in another annotation (*e.g.* see Figure 4.6). Only the primary animal in each annotation should be used to determine identity, but photobombs provide strong evidence of matching that can confuse a verification algorithm.

(a) (b)

Figure 4.6: **Photobomb example** A secondary animal in an annotation can cause a "photobomb". Notice the primary animal in (4.5a) appears in the background of (4.5b).

During events like the GZC we labeled several annotation pairs as photobombs. Using these labels we will construct a classifier in the same way that we constructed the primary match-state classifier. We start with the same set of training data used to learn the primary classifier. Because we only have a small set of explicitly labeled photobomb pairs, we include all such pairs in the training dataset. Any pair in this set that is explicitly labeled as a photobomb is given that label, otherwise it is labeled as not a photobomb.

4.4 PAIRWISE CLASSIFICATION EXPERIMENTS

We evaluate the pairwise classifiers on two datasets, one of plains zebras and the another of Grévy's zebras with 5720 and 2283 annotations, respectively. The details of these datasets were previously described in Section 3.5.1. To evaluate our pairwise classifier, we choose a sample of annotation pairs from these datasets as detailed in Section 4.2. This results in 47312 pairs for plains zebras and 18010 pairs for Grévy's zebras. The number of pairs per class is detailed in Table 4.1. Note that our datasets only contain a small number of labeled incomparable and photobomb cases. For plains zebras only 53 incomparable cases were explicitly labeled, while the other 300 were generated using heuristics. For Grévy's zebras, there are no incomparable cases because all annotations have a right-side viewpoint. Therefore, the primary focus of our experiments will be separating positive from negative matching states.

After sampling, we have a set of annotation pairs and each is associated with a ground truth matching state label of either positive, negative, or incomparable. Addition-

ally, each pair is also labeled as either a photobomb or not a photobomb. For each pair we construct a pairwise feature vector as described in Section 4.1.

We split this dataset of labeled annotation pairs into multiple disjoint training and testing sets using grouped stratified k-fold cross validation (we use 3 folds). Note that this grouping introduces a slight variation on standard stratified k-fold cross validation. First, we enforce that each sample (a pair of annotations) within the same name or between the same two names must be placed in the same group. Then, the cross validation is constrained such that all samples in a group are either in the training or testing set for each split. In other words, this means that a specific individual cannot appear in both the training and testing set. The same is true for specific pairs of individuals. By grouping our cross-validation folds in this way, we ensure that the classifier cannot exploit individual specific information to improve its predictions on the test set. This increases our confidence that our results will generalize to new individuals.

For each cross validation split, we train the matching state and photobomb-state classifier on the training set and then predict probabilities on each sample in the testing set. Because the cross validation is k-fold and the splits are disjoint, each sample appears in a testing set exactly once. This means that each sample in our dataset is assigned match-state and photobomb-state probabilities exactly once. Because each prediction is made using classifiers trained on disjoint data, the predictions will be unbiased. Thus, each sample in the dataset has a match-state and photobomb-state probability. Therefore, we can use all sample pairs to evaluate the performance of each classifier.

In our experiments we compare these predicted match-state probabilities to the scores generated by LNBNN. In order to do this, we must generate an LNBNN score for each pair in our sample. This is done by first taking the unique set of annotations in the sample of pairs. Then, we use these annotations as a database. We issue each anno-

Table 4.1: **Database statistics for the pairwise experiment** Starting with a database of annotations with name labels, we sample a set of annotation pairs to evaluate our pairwise classifiers with.

	Names	Annots	Positive	Negative	Incomparable	Photobombs
Plains zebras	1202	5720	16583	30376	353	286
Grévy's zebras	771	2283	5002	13008	0	76

tation as a query, taking care to ignore self-matches. Any (undirected) pair in our dataset that appears as a query / database pair in the ranked list is assigned that LNBNN score. If the same pair is in two ranked lists, then the maximum of the two scores is used. Any pair that does not appear in any ranked list (because LNBNN failed to return it) is implicitly given a score of zero. Note that the scores from the LNBNN ranking algorithm can only be used to distinguish positive cases from non-positive pairs. Unlike the match-state classifier, the LNBNN scores cannot be used to distinguish non-positive cases as either negative or incomparable. Later in Chapter 5, it will be vitally important to distinguish these cases, but for now, in order to fairly compare the two algorithms, we only consider positive probabilities from the match-state classifier.

We have now predicted match-state probabilities, photobomb-state probabilities, and one-vs-many LNBNN scores for each pair in our dataset. In the next subsections we will analyze the predictions of each classifier. For both the match-state and photobomb-state classifiers we will measure the raw number of classification errors and successes in a confusion matrix. We will then use standard classification metrics like precision, recall, and the Matthews correlation coefficient to summarize these confusion matrices. We will also inspect the importance of each feature dimension of our pairwise feature vector as measured during random forest learning. For each classifier we will present several examples of failure cases to illustrate where improvements are can be made. Additionally, we will compare the match-state classifier to LNBNN in two ways. First, we will compare the original LNBNN ranking against a re-ranking using the positive probabilities. Second, we will compare the ability of LNBNN and the match-state classifier to predict if a pair is positive or not by looking at the distribution of positive and non-positive scores as well as the ROC curves.

4.4.1 Evaluating the match-state classifier

The primary classifier predicts the matching state (positive, negative, incomparable) of a pair of annotations. Each pair of annotations is assigned a probability for each of these states, and those probabilities sum to one. In this context we classify a pair as the state with the maximum predicted probability. However, in practice we will choose thresholds for automatic classification where the false positive rate is sufficiently low.

From these multiclass classifications we build a confusion matrix for plains and Grévy's zebras. These confusion matrices are shown in Table 4.2. In Table 4.4 we summarize the information in each confusion matrix by computing the precision, recall, and Matthews correlation coefficient (MCC) [254] for each class. The MCC provides a measurement of overall multiclass classification performance not biased by the number of samples contained in each class. An MCC ranges from $+1$, indicating perfect predictions, to -1, indicating pathological inverse predictions, and 0 represents random predictions. The measured MCC of 0.83 for plains zebras and 0.91 for Grévy's zebras, indicates that our match-state classifiers have strong predictive power.

Table 4.2: **Match-state experiment confusion matrix** This is the multiclass match-state confusion for plains and Grévy's zebras. The rows are the real (ground truth) state, and the columns are the predicted states. Each pair is classified as positive, negative, or incomparable depending on which state has the maximum probability.

	Negative	Positive	Incomparable	$\sum$ real
Negative	29154	1217	5	30376
Positive	2622	13954	7	16583
Incomparable	65	14	274	353
$\sum$ predicted	31841	15185	286	

(a) Plains zebras match-state confusion matrix

	Negative	Positive	Incomparable	$\sum$ real
Negative	12777	231	0	13008
Positive	417	4585	0	5002
Incomparable	0	0	0	0
$\sum$ predicted	13194	4816	0	18010

(b) Grévy's zebras match-state confusion matrix

Table 4.4: **Match-state experiment evaluation metrics** The multiclass match-state evaluation metrics for plains and Grévy's zebras are computed from the confusion matrix. These metrics demonstrate that our match-state classifiers have strong predictive power.

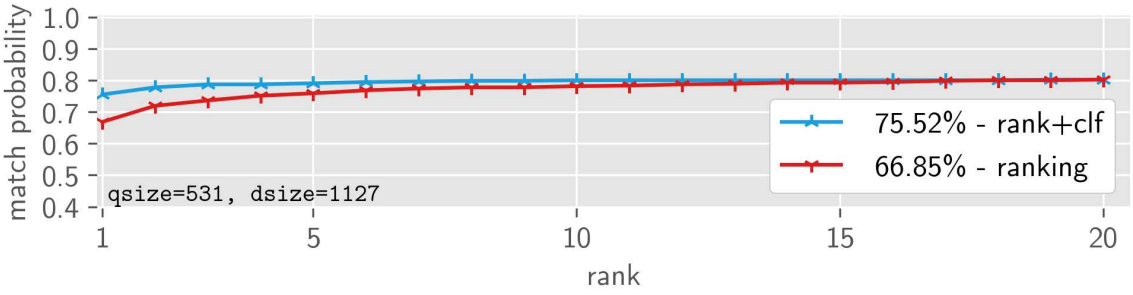
	Precision	Recall	MCC	Support
Negative	0.92	0.96	0.82	30376
Positive	0.92	0.84	0.82	16583
Incomparable	0.96	0.78	0.86	353
ave/sum	0.92	0.92	0.83	47312

(a) Plains zebras match-state metrics

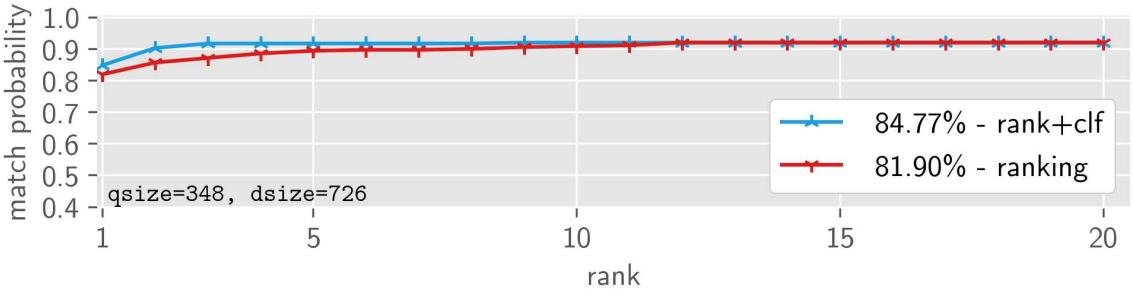
	Precision	Recall	MCC	Support
Negative	0.97	0.98	0.91	13008
Positive	0.95	0.92	0.91	5002
Incomparable	nan	nan	nan	0
ave/sum	0.96	0.96	0.91	18010

(b) Grévy's zebras match-state metrics

In addition to being strong predictors of match state, we show that the positive probabilities from our classifiers can be used to re-rank the ranked list produced by LNBNN. Using the procedure from our experiments in Section 3.5, we issue each annotation in our testing set as a query and obtain a ranked list. Using the fraction of correct results found at each rank, we construct a cumulative match characteristic (CMC) curve [249]. We denote this CMC curve as `ranking`. Then we take each ranked list and compute match-state probabilities for each top ranked pair of query/database annotations. We use the positive probabilities to re-rank the lists, and then we construct another CMC curve corresponding to these new ranks. This re-ranked CMC curve is denoted as `rank+clf`. The results of this experiment — illustrated in Figure 4.8 — clearly demonstrate that the number of correct matches returned at rank 1 is improved by re-ranking with our pairwise classifier.



(a) Plains zebras



(b) Grévy's zebras

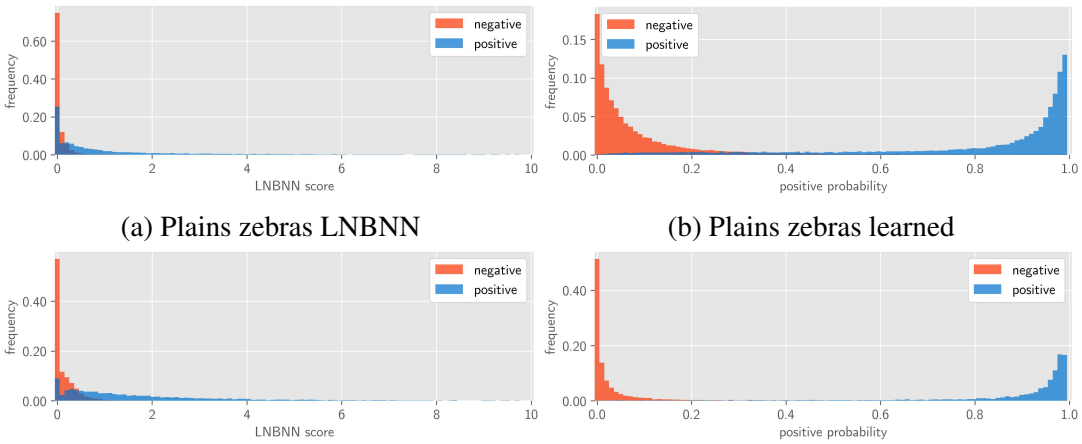
Figure 4.8: **Re-ranking experiment** Re-ranking the top LNBNN results using the positive probabilities from the match-state classifier improves the number of correct matches at rank 1 for both plains and Grévy's zebras.

4.4.1.1 Binary positive classification

Although the accuracy of multiclass predictions is important, the most important task in animal identification is to determine when a pair of annotations is positive and when it is not. Therefore, we design an experiment that tests how well our match-state classifier can distinguish positive pairs from other cases. We will compare our learned classifiers to a baseline method that simply uses the LNBNN scores. We begin this comparison by illustrating histograms of LNBNN scores and positive pairwise probabilities for positive and non-positive cases in Figure 4.10. The advantages of the pairwise scores are immediately noticeable. The pairwise scores provide a superior separation between positive and non-positive cases. Furthermore, the pairwise scores range from zero to one, which makes them easier to interpret than the unbounded LNBNN scores.

We can make a more direct and precise comparison by considering both LNBNN scores and positive pairwise probabilities as binary classifiers. In this context, we can measure the separability of each method using the area under an ROC curve. The ROC curves comparing the LNBNN scores and the learned probabilities are illustrated in Figure 4.12. In all cases the learned AUC is significantly better than the AUC of LNBNN . For plains zebras, the pairwise AUC is 0.97 and the LNBNN AUC is 0.82. For Grévy's zebras the pairwise AUC is 0.99, and the LNBNN AUC is 0.89. These experiments clearly demonstrate that the pairwise classifier outperforms LNBNN in this classification task.

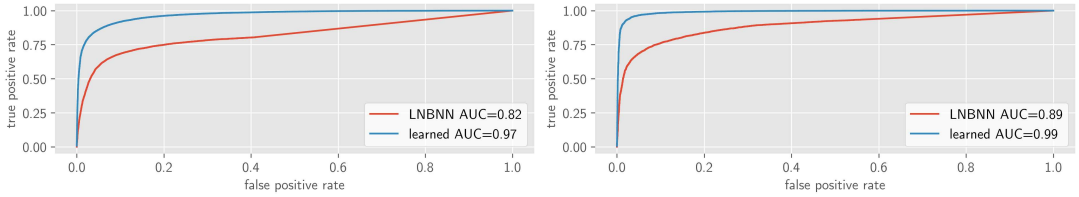
In addition to illustrating the effectiveness of our classifiers when classifications are made for all samples, the ROC curves — which plot the true positive rate and the false positive rate as a function of a threshold — demonstrate that an operating point can be chosen to automatically classify a significant number of positive pairs while making very few mistakes. For plains zebras, a true positive rate of 0.25 results in 4146 true positives and only 38 false positives. For Grévy's, an operating point with a true positive rate of 0.5, results in 2501 true positives and only 28 false positives. Therefore, by carefully selecting an operating point we can bypass a significant number of manual reviews without making a significant number of mistakes when adding new annotations to our dataset.



(a) Plains zebras LNBNN (b) Plains zebras learned

(c) Grévy's zebras LNBNN (d) Grévy's zebras learned

Figure 4.10: **Positive score histogram experiment** This shows positive scores of LNBNN (left) and the pairwise algorithm (right) for pairs of plains (top) and Grévy's (bottom) zebras. The learned probabilities are more separable and more interpretable than LNBNN scores. Note that in this plot, negative refers to annotation pairs with a non-positive match-state label.



(a) Plains zebras (b) Grévy's zebras

Figure 4.12: **Positive match-state ROC experiment** This shows the positive match-state ROC for scores computed by the pairwise classifier and LNBNN. The pairwise classifier significantly improves the separation of positive and non-positive pairs.

4.4.1.2 Feature importance

To better understand which feature dimensions are the most useful for classification, we measure the "mean decrease impurity" (MDI) [255] of each feature dimension. The MDI is a measure of feature importance that is computed during the training phase. As each decision tree is grown, for each node, we record the number of training samples of each class that reach it. This is used to compute the impurity of each node, *i.e.* the entropy of class labels. Each node is weighted using the fraction of total samples that reach it. The weighted impurity decrease of a node is its weighted impurity minus the weighted sum of its children's impurity. The MDI for a single feature dimension in a

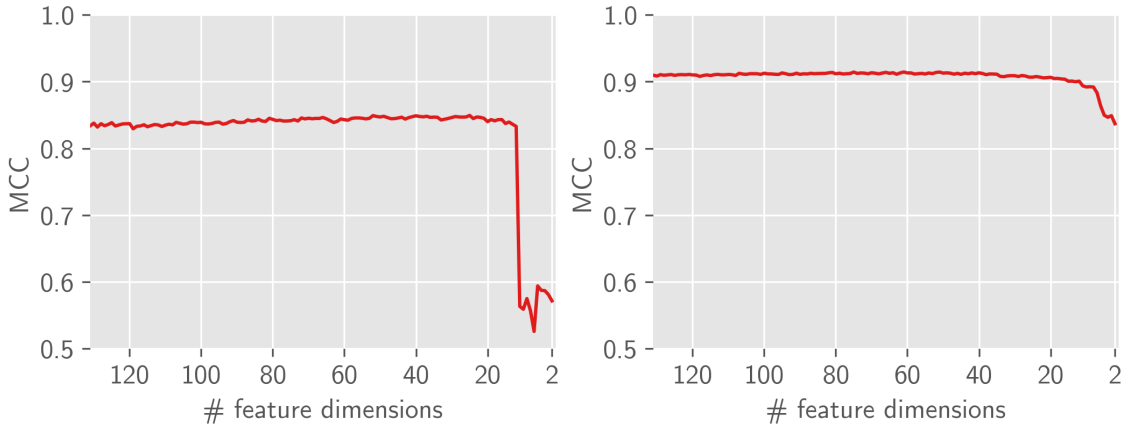
single tree is computed as the weighted impurity decrease of all nodes using that feature. The overall MDI for the forest is obtained by averaging over all trees.

Using the MDI we "prune" the dimensions of our sample feature vectors by removing the least important dimensions. We measure the effect of pruning on classification accuracy using a greedy algorithm. First we learn a random forest on a training set and compute the MCC on a test set. Then we find the feature dimension with the lowest MDI and remove it from the dataset. We repeat these two steps until there is only a single feature dimension remaining.

The impact of pruning on classification accuracy is illustrated in Figure 4.14, where we plot the MCC as a function of the number of feature dimensions remaining. The results of this experiment indicate that there is a small increase in classification accuracy from pruning features dimensions. We find that reducing the number of feature dimensions to 25 increases the MCC by 0.0155 for plains zebras, and using 61 dimensions increases the MCC by 0.0048 for Grévy's. Note that once the number of feature dimensions falls below $\sim$20 the performance starts to degrade and suffers a harsh drop at $\sim$10 dimensions. This suggests that these top features are the most important to making a match-state prediction.

The numeric importance of these top 10 pruned feature dimensions is reported in Table 4.6. For both species we find that the most important features are statistics involving the ratio score. These statistics indicate the distinctiveness and similarity of a pair of annotations. Statistics about the spatial verification error, which signifies when the matches are not well aligned, are also important for both species. For plains zebras, the global viewpoint is important because the dataset contains incomparable examples.

Even though we have shown that a small improvement can be made by pruning to only the most important feature dimensions, we choose to use full 131 dimensions in the remainder of our experiments because the overall performance gain is small.



(a) Plains zebras

(b) Grévy's zebras

Figure 4.14: **Pruning feature dimensions for match classification** This shows the effect of pruning the least important feature dimensions on the MCC of the match-state classifier. We find that a reduced subset of feature dimensions results in a slight increase in classification accuracy over the original 131 features. However, there is a point at which reducing the number of feature dimensions significantly degrades performance.

Table 4.6: **Important features for match-state prediction** These are the top 10 most important feature dimensions for predicting the match-state (positive, negative, incomparable) for a pair of annotations, after removing the least important dimensions.

Dimension	Importance
mean(ratio_score[ratio<0.625])	0.1246
mean(ratio_score[ratio<0.8])	0.1041
std(ratio_score[ratio<0.8])	0.0779
med(ratio_score[ratio<0.625])	0.0712
std(ratio_score[ratio<0.625])	0.0681
med(ratio_score[ratio<0.8])	0.0625
global(delta_gps)	0.0446
global(delta_time)	0.0364
global(delta_view)	0.0344
med(sver_err_xy[ratio<0.625])	0.0330

(a) Plains zebras

Dimension	Importance
mean(ratio_score[ratio<0.8])	0.1222
mean(ratio_score[ratio<0.625])	0.1078
std(ratio_score[ratio<0.625])	0.0947
std(ratio_score[ratio<0.8])	0.0921
med(ratio_score[ratio<0.8])	0.0653
med(ratio_score[ratio<0.625])	0.0544
med(sver_err_xy[ratio<0.8])	0.0369
med(sver_err_xy[ratio<0.625])	0.0304
mean(match_dist[ratio<0.625])	0.0285
mean(sver_err_ori[ratio<0.8])	0.0238

(b) Grévy's zebras

4.4.1.3 Failure cases

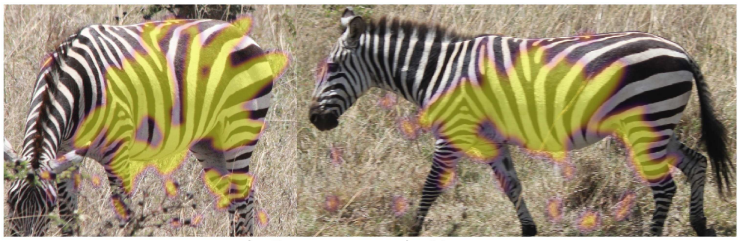
Lastly we investigate several causes of failure. The primary reasons that cause the match-state classifier to fail are similar to those that create difficulty for the ranking algorithm. These were previously discussed in Section 3.5.8. Factors such as viewpoint, quality, and occlusion are inherently challenging because they reduce the similarity between matching descriptors, and increase the disparity between the annotations. This makes it difficult to correctly establish and spatially verify feature correspondences. Photobombing animals and scenery matches also pose a challenge to the match-state classifier, often causing it to produce ambiguous probabilities.

We separate failure cases into three main categories: (1) failure to classify a pair as positive, (2) failure to classify a pair as negative, and (3) failure to classify a pair as incomparable. The examples in Figure 4.16 illustrate the first and most important failure case category. Due to occlusion, quality, and viewpoint, the established correspondences were not distinctive enough for the classifier to confidently predict positive. However, in all but the last case the probabilities are ambiguous, which implies that improvements could be made to fix these failures. In the last case, only a few distinctive matches were made, which suggests that procedure that establishes feature correspondences could be improved.

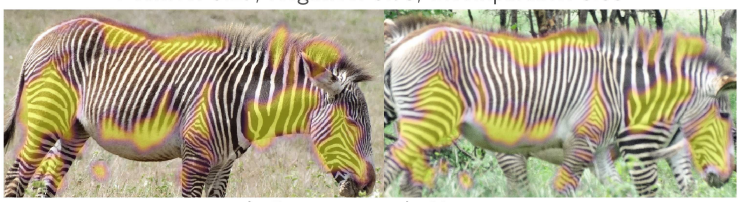
Examples of the second case are illustrated in Figure 4.18. These examples were incorrectly classified as positive, even though they are negative. In two cases this is due to scenery matches and photobombing animals. In the other cases the pairwise classifier matches similar regions of the animal, but it is unable to key in on the strong negative evidence provided by different distinctive patterns on corresponding body parts of the animal. In the future, algorithms powered by convolutional neural networks may be able to take advantage of this strong negative evidence. For the third case, it is not surprising that the examples in Figure 4.20 were not labeled as incomparable because only a small amount of incomparable training data was available. Furthermore, photobombs and scenery matches also seem to cause a problem for incomparable examples. Note that in the majority of all three types of failure, the examples have non-extreme probabilities assigned to each state. This demonstrates that the classifier is not confident in these failed predictions.

Interestingly, inspection of the failure cases revealed several labeling errors in the database. The examples illustrated in Figure 4.22 all show pairs of annotations with non-positive labels that should have been labeled as positive. Note that the positive probabilities from two of these examples are very close to 1.0, indicating that the classifier is confident that these ground truth labels are incorrect. Furthermore, these success cases were found in the context of noisy ground truth labels, showing that our classifier is robust to errors in ground truth labels.

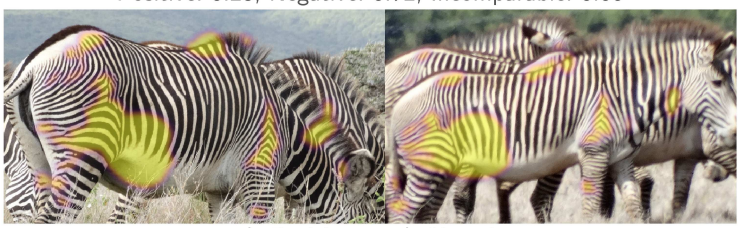
real=Positive, pred=Negative,
Positive: 0.50, Negative: 0.50, Incomparable: 0.00



real=Positive, pred=Negative,
Positive: 0.27, Negative: 0.73, Incomparable: 0.00



real=Positive, pred=Negative,
Positive: 0.28, Negative: 0.72, Incomparable: 0.00

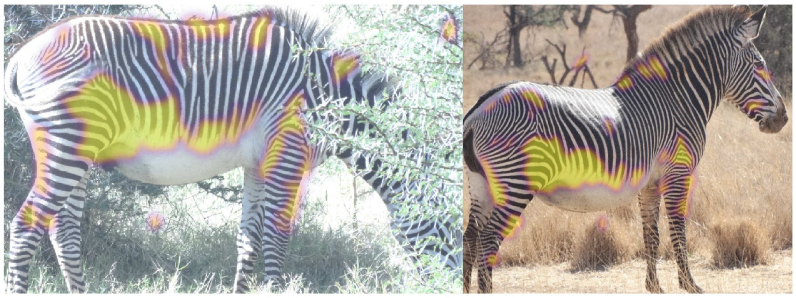


real=Positive, pred=Negative,
Positive: 0.01, Negative: 0.99, Incomparable: 0.00

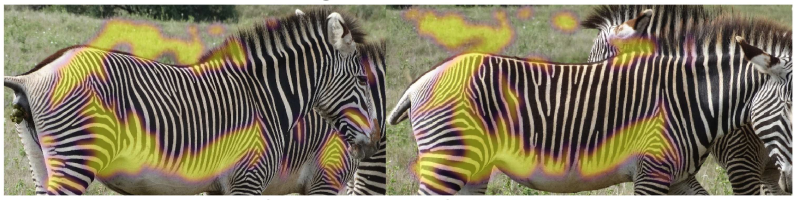
Figure 4.16: **Positive pairwise failure case** These pairs are all positive, but the match-state classifier predicts each as negative. These failures can be attributed to poor image quality, occlusion, and viewpoint variations. Notice that the positive probability is well above zero in all but one case.

real=Negative, pred=Positive,
Positive: 0.67, Negative: 0.32, Incomparable: 0.00

real=Negative, pred=Positive,
Positive: 0.68, Negative: 0.32, Incomparable: 0.00



real=Negative, pred=Positive,
Positive: 0.50, Negative: 0.50, Incomparable: 0.00



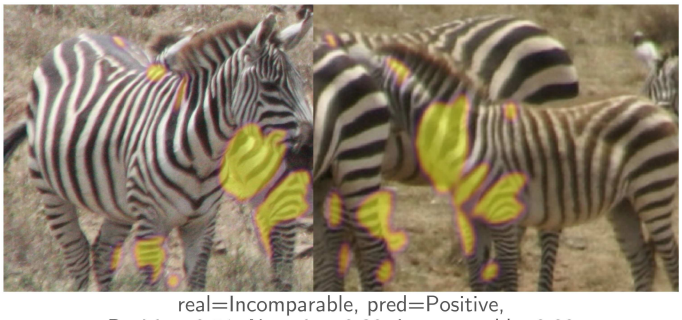
real=Negative, pred=Positive,
Positive: 0.69, Negative: 0.31, Incomparable: 0.00

Figure 4.18: **Negative pairwise failure case** These pairs are negative, but the classifier predicts positive. Notice that the negative probability in each case is not close to zero. While the classifier can recognize that the matches may be weak, it is not able to explicitly recognize that the same region on two animals contains different distinctive patterns. Photobomb and scenery matches also contribute to negative failure cases.

real=Incomparable, pred=Negative,
Positive: 0.29, Negative: 0.45, Incomparable: 0.26

real=Incomparable, pred=Negative,
Positive: 0.07, Negative: 0.49, Incomparable: 0.44

real=Incomparable, pred=Negative,
Positive: 0.05, Negative: 0.52, Incomparable: 0.43

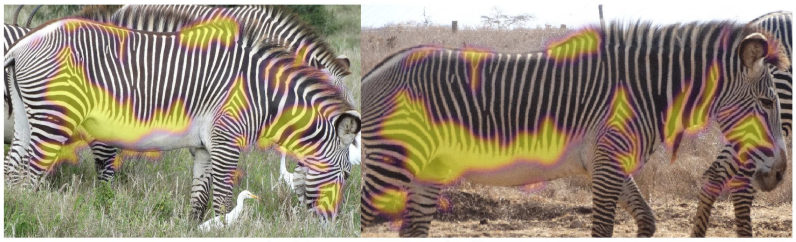


real=Incomparable, pred=Positive,
Positive: 0.51, Negative: 0.39, Incomparable: 0.09

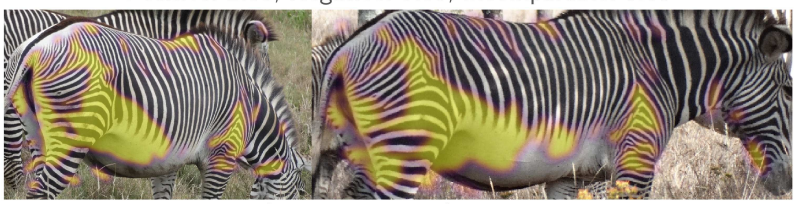
Figure 4.20: **Incomparable pairwise failure case** These pairs are incomparable, but the classifier predicted either positive or negative. In part this is due to a small amount of available incomparable training data. In the top two examples the confidence in the incorrect negative prediction is low. In the bottom two examples, scenery matches and photobombing animals hinder the classifier's ability to predict incomparable.

real=Incomparable, pred=Positive,
Positive: 0.67, Negative: 0.20, Incomparable: 0.13

real=Positive, pred=Negative,
Positive: 0.00, Negative: 1.00, Incomparable: 0.00



real=Negative, pred=Positive,
Positive: 1.00, Negative: 0.00, Incomparable: 0.00

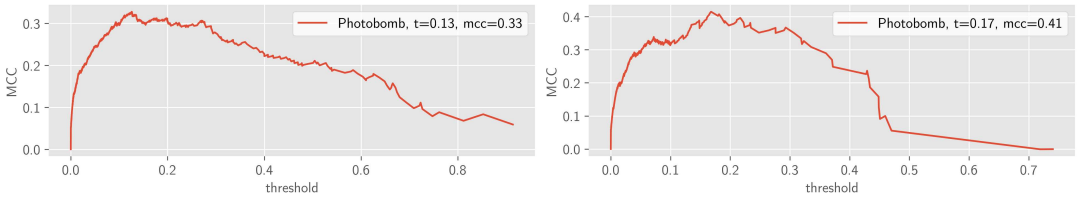


real=Negative, pred=Positive,
Positive: 0.99, Negative: 0.01, Incomparable: 0.00

Figure 4.22: **Errors in the match-state ground truth** Ground truth errors in the database are the reason for several match-state failure cases. In these examples the classifier picks the correct answer even though the ground truth is incorrect. Note that the probability assigned to the true state of each pair is close to 1.0.

4.4.2 Evaluating the photobomb-state classifier

In this subsection we perform a set experiments — similar to those used to evaluate the match-state classifier — to demonstrate the effectiveness of our photobomb-state classifier. Because there are only a few photobomb training examples, the random forest is not able to learn strong probabilities. In the initial design of these experiments, a pair was classified as a photobomb if its probability was greater than 0.5, but we found that this caused the MCC of Grévy's zebras to drop to 0.0. However, because this is a binary classification problem we can choose any threshold as an operating point and classify a pair as a photobomb if its probability is above that threshold and as not a photobomb otherwise. Therefore, for each dataset, we choose a photobomb-state probability threshold to maximize the MCC as illustrated in Figure 4.24. We classify a pair as a photobomb if its probability is above 0.13 for plains zebras and 0.17 for Grévy's zebras. Using these adjusted thresholds, the overall performance measured using the classification confusion matrices in Table 4.8 and evaluation metrics in Table 4.10.



(a) Plains zebras (b) Grévy's zebras

Figure 4.24: **Maximizing the photobomb MCC** Because there are not many labeled photobomb pairs, the probabilities returned by the photobomb-state classifier are low. However, good classification results can be achieved by choosing an operating point that maximizes the MCC. In each plot the legend indicates the threshold corresponding to the maximum MCC.

Due to the small amount of available training data, the performance of the photobomb-state classifier is weaker than the match-state classifier. By choosing the appropriate thresholds we achieve an MCC of 0.34 for plains zebras and 0.40 for Grévy's zebras. These scores indicate that each photobomb-state classifier has weak but significant predictive power. While these MCCs are not overwhelmingly strong, they do demonstrate that each classifier is able to learn from only a few labeled training examples, and it seems likely that the MCCs would significantly improve with more labeled training data. From these measurements we can conclude that the photobomb-state classifier is learning.

Table 4.8: **Photobomb-state adjusted confusion matrix** This shows the confusion matrix after adjusting the probability threshold to maximize the MCC. The columns indicate predicted classes, and the rows indicate real (ground truth) classes. The final column indicates the number of examples of each class.

	Not Photobomb	Photobomb	$\sum$ real
Not Photobomb	46814	212	47026
Photobomb	187	99	286
$\sum$ predicted	47001	311	

(a) Plains zebras photobomb adjusted confusion matrix

	Not Photobomb	Photobomb	$\sum$ real
Not Photobomb	17913	18	17931
Photobomb	55	24	79
$\sum$ predicted	17968	42	18010

(b) Grévy's zebras photobomb adjusted confusion matrix

Table 4.10: **Photobomb-state adjusted evaluation metrics** These evaluation metrics are computed from the confusion matrix after adjusting the probability threshold to maximize the MCC.

	precision	recall	mcc	support
Not Photobomb	1.00	1.00	0.33	47026
Photobomb	0.32	0.35	0.33	286
ave/sum	0.99	0.99	0.33	47312

(a) Plains zebras adjusted photobomb metrics

	Precision	Recall	MCC	Support
Not Photobomb	1.00	1.00	0.41	17931
Photobomb	0.57	0.30	0.41	79
ave/sum	1.00	1.00	0.41	18010

(b) Grévy's zebras adjusted photobomb metrics

Our conclusion that the photobomb-state classifier is learning is supported the top 10 most important features as measured using the MDI. It makes intuitive sense that the important features — illustrated in Table 4.12 — would be the ones selected by the random forest learning algorithm. Because pairs of annotations taken around the same place and time are more likely to be photobombs, each random forest places the most weight on global feature dimensions such as time delta, GPS delta, and speed. The classifiers also makes use of the spatial position of the feature correspondences, which can be indicative of a photobomb (*e.g.* when all the matches are in the top left corner of an annotation). Because the random forest seems to be selecting reasonable features, the main source of weakness is likely due to the amount of training data.

We gain further insight into the photobomb-state classifier by considering several failure cases examples, which are illustrated in Figure 4.26. In each example we observe that the classifier is either confused or it does not have enough information to label a pair as a photobomb. Furthermore, the classifier was able to find several new photobomb pairs that were mislabeled in the database. We illustrate several of these mislabeled cases in Figure 4.28. Notice that many of the probabilities predicted for these cases are well above the thresholds. Predicting high probabilities for these undiscovered photobomb cases indicates that the photobomb-state classifier is stronger than our measurements suggest. These mislabeled ground truth cases also help explain why the predicted probabilities are so low; the learning algorithm is encouraged to incorrectly classify them. Reviewing and relabeling all of these cases would improve results by both removing noise from the training set and increasing the number of labeled examples.

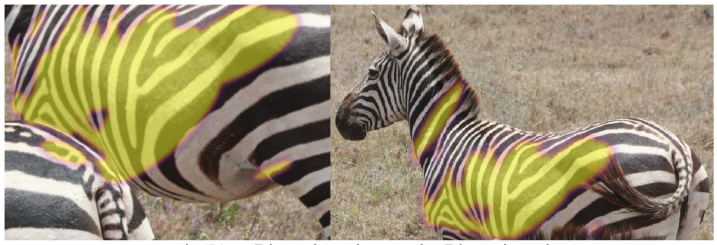
Table 4.12: **Important features for photobomb-state prediction** These are the top 10 most important features for predicting if a pair of annotations has a photobomb. Features like speed and GPS delta are important because photobombs are more common in pairs of annotations taken at the same time and place. Features related to the spatial distribution of the feature correspondences are important because photobombing animals often appear off to one side of an annotation.

Dimension	Importance
global(delta_time)	0.0383
global(delta_gps)	0.0314
std(norm_x1[ratio<0.8])	0.0262
std(norm_x2[ratio<0.625])	0.0243
std(norm_x2[ratio<0.8])	0.0209
std(norm_x1[ratio<0.625])	0.0198
mean(norm_x2[ratio<0.8])	0.0193
mean(norm_x2[ratio<0.625])	0.0191
mean(norm_x1[ratio<0.625])	0.0182
med(norm_x2[ratio<0.625])	0.0176

(a) Plains zebras photobomb importance

Dimension	Importance
global(delta_time)	0.0440
global(speed)	0.0339
global(delta_gps)	0.0331
std(norm_x2[ratio<0.625])	0.0312
std(ratio_score[ratio<0.8])	0.0279
std(norm_x2[ratio<0.8])	0.0278
std(norm_x1[ratio<0.8])	0.0273
std(ratio_score[ratio<0.625])	0.0244
global(delta_qual)	0.0226
global(max_qual)	0.0200

(b) Grévy's zebras photobomb importance



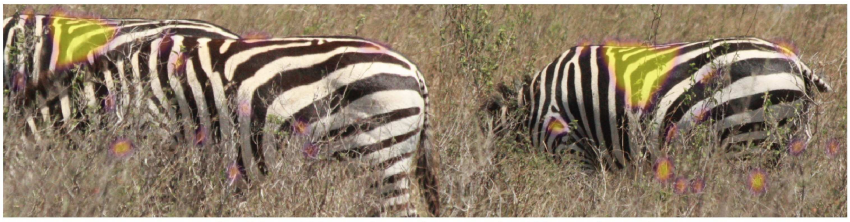
real=Not Photobomb, pred=Photobomb,
Not Photobomb: 0.80, Photobomb: 0.20

real=Photobomb, pred=Not Photobomb,
Not Photobomb: 1.00, Photobomb: 0.00

real=Photobomb, pred=Not Photobomb,
Photobomb: 0.47, Not Photobomb: 0.53

real=Photobomb, pred=Not Photobomb,
Photobomb: 0.06, Not Photobomb: 0.94

Figure 4.26: **Photobomb failure cases** In the top example the classifier incorrectly predicts photobomb due to the alignment of the annotations. In the next case down, the classifier incorrectly predicts photobomb, but no matches were made between the photobombing animals. The last two cases the classifier incorrectly predicts not photobomb, but the confidence of the prediction is low.



real=Not Photobomb, pred=Photobomb,
Not Photobomb: 0.87, Photobomb: 0.13

real=Not Photobomb, pred=Photobomb,
Not Photobomb: 0.19, Photobomb: 0.81

real=Not Photobomb, pred=Photobomb,
Photobomb: 0.74, Not Photobomb: 0.26

real=Not Photobomb, pred=Photobomb,
Photobomb: 0.73, Not Photobomb: 0.27

Figure 4.28: **Errors in the photobomb-state ground truth** Ground truth errors in the database are the reason for several photobomb-state failure cases. It is encouraging that the photobomb-state classifier is able to detect errors in the ground truth even given only a few training examples.

4.4.3 Classifier experiment conclusions

In these experiments we have demonstrated that our pairwise match-state classifier is able to reliably separate positive from negative and incomparable cases. When making automatic decisions, these probabilities have several advantages over the LNBNN scores from Chapter 3. Not only do they have more predictive power, they are interpretable, and they always range between 0 and 1.

We will use these classifiers to make automatic decisions about pairs of annotations where the match-state probability is above a threshold. Our experiments have shown that it is possible to select a threshold where the false positive rate is sufficiently low. Furthermore, the classifier is able to improve the ranking algorithm by re-ranking its results. Lastly, because the classifier predicts probabilities independent of their position in the ranked list, it can be used to determine when a query individual is new — *i.e.* does not have a correct match in the database.

The performance of secondary photobomb classifier is weaker, but this is likely due to a small amount of training data. Even in its weak state, it can be used to prevent automatic review of some photobomb cases by adjusting the classification threshold to be less than 0.5. Because the photobomb classifier can only prevent automatic decisions and does not make them, the cost of including it in our algorithms is small, and by doing so we will increase the amount of labeled training data from which a stronger photobomb classifier can be bootstrapped.

4.5 SUMMARY OF PAIRWISE CLASSIFICATION

In this chapter we have constructed a verification mechanism that can predict the probability that a pair of annotations is positive, negative, or incomparable. We have also constructed a secondary classifier that can predict when — namely in the case of photobombs — a pair of annotations might confuse the primary match-state classifier. This was done by constructing a feature vector that contains matching information about a pair of annotations. We have constructed a representative training set by selecting hard, moderate, and easy training examples. We used the random forest learning algorithm to train our classifiers. Our experiments demonstrate that the match-state classifier is able to strongly separate positive and negative cases. The performance of the photobomb-state

classifier was weaker, but could likely be improved with more training data.

Based on our experiments, it is clear that the ranking algorithm is improved by an automatic verifier, but by themselves ranking and verification are not enough to robustly address animal identification. There is no mechanism for error recovery, nor is there a mechanism for determining when identification is complete. This means the operating point for the automatic review threshold must be set conservatively to avoid any errors, which results in more work for a human reviewer. These issues are addressed in Chapter 5 using a graph-based framework to manage the identification process. This framework will use graph connectivity to further reduce the number of required manual reviews, detect when errors have occurred, recover from the errors, and stop the identification process in a timely manner.

5. IDENTIFICATION USING CONNECTIVITY IN A DECISION GRAPH

In this chapter we frame the problem of animal identification in terms of constructing a *decision graph*. In this graph, each node is an annotation, and each edge represents a decision made between two annotations. Edges determine if two annotations are the same (positive) or different (negative) individuals or if they cannot be compared (incomparable). Using the connectivity of the decision graph, we naturally address the problem of animal identification. Assuming no edges are incorrectly labeled, each connected component of positive edges are all annotations from the same individual animal. We will refer to these as *positive connected components* (PCCs). We call a graph *id-complete* if there is at least one negative edge between all pairs of PCCs. In this case all labeled individuals must be distinct, and we have therefore completed identification. Alternatively, if every pair of PCCs either has a negative edge between them except for pairs of PCCs where all possible edges between them are incomparable, then we know that the data cannot be used to determine if those incomparable PCCs are the same or different. Removing the assumption that all edges are correctly labeled, we call a decision graph *inconsistent* if any PCC contains a negative edge because it implies that an edge has been mislabeled. Therefore, stated abstractly, goal of graph identification is to determine a correct, consistent, and id-complete set of edges in the decision graph.

In the most general case, the decision graph is initialized with an empty set of edges. This captures the challenge posed by events like the GZC from Section 1.3, where we are given a set of annotations without name labels. In essence, each annotation starts by itself as an individual animal, but because there are no negative edges, we cannot be sure that this is the correct name labeling. In order to refine the name labeling, we add edges to the decision graph, gradually moving from a state of zero confidence that the name labelings are correct to a state of high confidence. However, it is important to note that the graph algorithm does not require that we start in an empty state. Given a known set of annotations with name labels and one or more new annotations with unknown name labels, we can add these new annotations to the existing database simply by labeling the

graph with edges that captures our current knowledge. Simply put, this means each known individual is a PCC, there is one negative edge between each pair of PCCs, and the new annotations are added as nodes without any edges. Regardless of the initial state, graph identification proceeds to complete the decision graph. For the remainder of this chapter, without loss of generality, we can assume that the graph starts in an empty state.

To construct the decision graph, we develop a semi-automatic review procedure that combines the ranking and verification algorithms presented in Chapters 3 and 4. The ranking algorithm will be used to suggest candidate edges to be placed in the graph, and the verification algorithm will be used to automatically review as many edges as possible. The key reason for combining these algorithms with a decision graph is to take advantage of its connectivity information. Connectivity not only identifies the individuals, but it can also be used to develop graph measures of *redundancy*, *completeness*, *consistency*, and *convergence*. By combining these graph measures with the ranking and verification algorithms we can prioritize edges for review based on both their pairwise probabilities and their ability to affect the consistency of the graph, which in turn allows us to: (1) increase confidence that the identifications are correct, (2) reduce the number of manual reviews, (3) detect and recover from review errors, and (4) determine when identification is complete.

An important property of the graph identification framework is that it is agnostic to the underlying computer vision procedures, which are abstracted into three components: (1) a ranking algorithm used to search for candidate positive edges, (2) a verification algorithm used to automatically review edges, and (3) a probability algorithm used assign probabilities to edges (note this is typically a by-product of the ranking or verification algorithm). In this thesis we use ranking algorithm from Chapter 3, and the verification algorithm from Chapter 4 to define these components because these are suitable for identifying textured species. However, while graph identification benefits from accurate computer vision subroutines, it can stand alone without them. This means that existing identification algorithms that only define a subset of these procedures (*e.g.* contour-based rank-only identification of humpback whales and bottlenose dolphins) could be seamlessly incorporated into our framework and realize the benefits of graph identification (*e.g.* a reduced number of manual reviews and error recovery mechanisms). Furthermore,

because pairwise decisions are gathered and maintained by this framework, verification algorithms can be retrained and improved, moving closer to a fully-automatic algorithm.

The first section (Section 5.1) of this chapter formalizes the decision graph and summarizes the priority-based review procedure used to construct it. This provides an overview of each component of the processes. Each of these components is then discussed in further detail in Sections 5.2 to 5.6. Section 5.7 experimentally demonstrates the ability of the graph identification algorithm to reduce the number of manual reviews and recover from decision errors. Section 5.8 concludes and summarizes the chapter.

5.1 THE DECISION GRAPH

The graph identification algorithm is a review procedure formalized around the notion of a *decision graph* $G = (V, E)$ whose nodes are annotations and whose edges are suggested by a ranking algorithm (LNBNN in our case) and decided upon by a combination of the probabilities output by a verification algorithm and by manual review. The edge set $E = E_p \cup E_n \cup E_i$ is composed of three disjoint sets. Throughout this chapter we will refer to the set that an edge belongs to as the label of that edge. Each edge in E_p is *positive*, meaning that it connects two annotations determined to be from the same individual. Each edge in E_n is *negative*, meaning that it connects annotations determined to be from different individuals. Finally, each edge in E_i is *incomparable*, meaning that it connects two annotations where it has been determined that there is not enough information to tell if they are from the same individual (*e.g.* when one annotation shows the left side of an animal and another other shows the right side). An example of a decision graph with all three edge types is illustrated in Figure 5.1. The goal of graph identification is to construct these edges.

The most important task is to determine the positive edges E_p. This is because each connected component in the subgraph $G_p = (V, E_p)$ corresponds to a unique individual. Producing an accurate set of these *positive connected components* (PCCs) addresses the larger problem of animal identification. However, an algorithm that only determines positive edges is not enough. This is because the algorithm may have failed to find all positive edges, resulting in two unconnected PCCs that should be *merged* into one. To ensure that this is not the case we must turn towards negative edges.

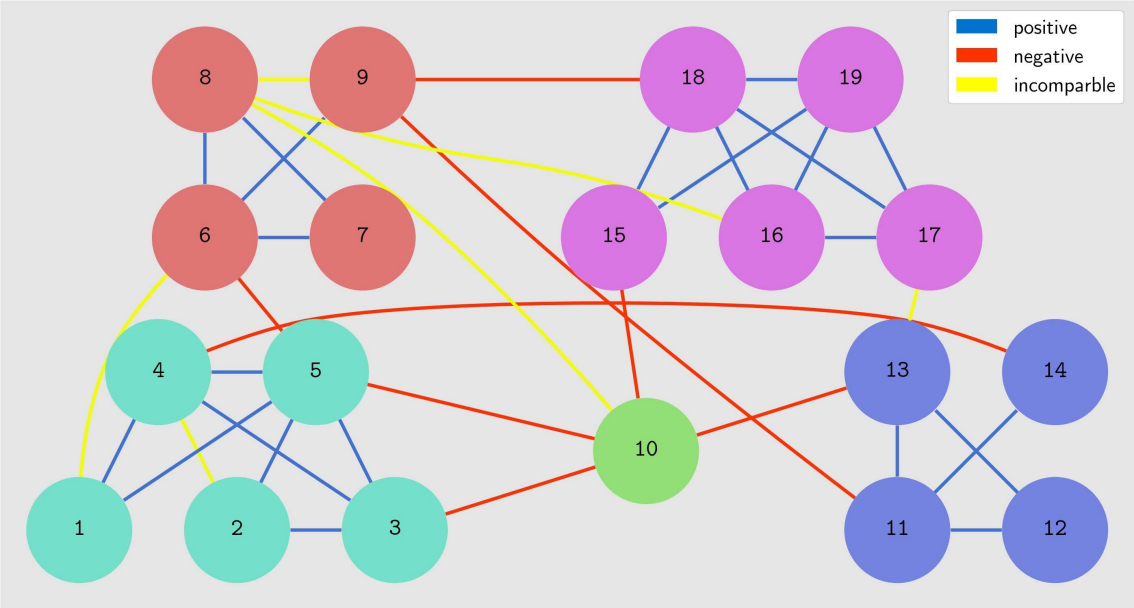


Figure 5.1: **A synthetic decision graph** This is a consistent synthetic decision graph with positive, negative, and incomparable edges. The color of each node represents the positive connected component (PCC) it belongs to.

We can gain confidence that all positive edges have been found by using negative edges E_n, which provide direct evidence that two annotations are different individuals. A correctly labeled negative edge between two PCCs means that no other unreviewed edge between those PCCs can be positive. Another important case is when a negative edge is contained within a PCC. When this happens, the PCC is *inconsistent*, and it implies that the PCC contains at least one mislabeled edge. Whenever an inconsistency is detected, we resolve it using the algorithm that we will define in Section 5.5.

Lastly, incomparable edges, E_i, simply signify that a positive or negative decision cannot be made. Whenever an edge is not labeled as positive, it is critically important to the construction of the decision graph that this non-positive edge is distinguished as either negative or incomparable. Negative edges restrict what new edges can be added because they carry information about the completeness and consistency of the graph. In contrast, incomparable edges do not. Incomparable edges can exist internally in a PCC without causing inconsistencies or between two PCCS without precluding them from being matched at a later point. In the case where all edges between two PCCs are incomparable and those PCCs are complete, then we know the current data is not enough

to determine if those PCCs are the same or different. In our datasets most images were taken to reduce the number of incomparable PCCs in order to simplify the sight-resight analysis, where all PCCs must be comparable to each other, but incomparable cases do exist. Thus, in our context incomparable edges play a minor but necessary role.

Using the connectivity of these edges, we can reduce the number of manual reviews needed. We now make several observations assuming that each edge is reviewed correctly. To reduce the number of potential reviews, notice, that once a group of nodes is connected by (a tree of) positive edges, all those nodes in that PCC can be inferred to belong to the same individual, and it is not necessary to consider any other edge internal to the PCC for review. Likewise, once a negative edge has been placed between two PCCs, all edges between those PCCs can be ignored. By ignoring these redundant edges we can reduce the number of reviews. Furthermore, we can construct a *deterministic termination criterion*; if a negative edge is placed between every pair of PCCs, then all individuals must have been discovered and identification has converged. We call such a graph id-complete because when all vertices in a PCC are collapsed into one, the resulting meta-graph is complete.

Unfortunately, there are two issues with these observations. First, they depend on the condition that each edge was correctly reviewed. In fact, we know that both the verification algorithm and human reviewers are sometimes wrong. We therefore will introduce redundancy into our graph that allows the algorithm to detect and correct errors, trading off the level of redundancy with the sophistication of the errors that may be caught. A small amount of redundancy is desirable because:

- PCCs with redundant edges are less likely to contain errors.
- Redundancy can potentially introduce inconsistency into a PCC, which signifies that an error has occurred.

Therefore, we will define a redundancy criterion in Section 5.2 which ignores edges within and between PCCs, but only after they meet a minimum level of redundancy.

Additionally, the deterministic termination criterion would require that each pair of PCCs has a redundant set of negative edges between them before the algorithm stops. However, the number of edges that need review grows quadratically. Therefore, unless the automatic algorithm is perfect or the dataset is small, the number of negative reviews

needed to converge will be too large for a human to handle. We address this concern in Section 5.6 using a probabilistic termination criterion.

5.1.1 The review algorithm

The review algorithm that produces the edges of a decision graph is outlined in Algorithm 1. Akin to a segmentation algorithm [256] that starts with an over-segmentation of an image, the identification graph starts with an empty set of edges, $G = (V, \{\})$, so in essence each annotation starts by itself as an individual animal, but because there are no negative edges we cannot be confident that any pair of annotations should indeed be given different labels. Therefore, the algorithm proceeds to prioritize and add positive edges that merge multiple annotations into the same PCC, negative edges that indicate that two annotations are different individuals, and incomparable edges that prevent two annotations from being labeled as positive or negative. Throughout the main algorithm, the graph is maintained in a *consistent* state, which means that each PCC has no internal negative edges. Note, that while we describe the algorithm starting from an empty graph, without loss of generality, the edges in the graph can be initialized to reflect a previously known labeling. Thus, the algorithm can address the case where nothing is known, new images are being added to an existing dataset, or multiple datasets are being combined.

Algorithm 1 Overview of the graph identification review procedure

While the graph has not converged:
- (1) Generate and prioritize candidate edges
- (2) Insert candidate edges into a priority queue
- (3) While the priority queue is not empty:
  - (3.1) Pop an edge from the priority queue
  - (3.2) Make a decision and add the edge to the graph
  - (3.3) If the edge causes an inconsistency drop into inconsistency recovery mode
  - (3.4) Update the priority queue based on the new edge
  - (3.5) If candidate edges require refresh, break

The first step of the algorithm is to generate candidate edges and predict probability measures (positive, negative, or incomparable) for each candidate edge. In the next step each edge is entered into a priority queue with a priority based first on its ability to be automatically reviewed and then on its positive probability. Next, the algorithm enters a loop where the next candidate edge is selected, a decision is made about this edge — either

automatically (as much as possible) or by the user — and it is added to the graph. The algorithm proceeds toward convergence by removing candidate edges from the priority queue, either directly from the top of the queue or indirectly by eliminating candidate edges that are no longer needed. A candidate edge is no longer needed when there are sufficient redundancies in the edge set within or between its PCCs. A pair of PCCs is *complete* when there are enough negative edges between them.

Each new edge addition could trigger two important events: (1) a *merge* — addition of a positive edge between different PCCs combines them into one PCC, and (2) an *inconsistency* — addition of either a negative edge within a PCC or a positive edge between PCCs that already have a negative edge between them creates an inconsistent PCC. Handling a merge is largely a matter of bookkeeping and can be done efficiently using a data structure that can dynamically maintain connected components [257]. Finding an inconsistency, however, drops the user into inconsistency recovery mode which alternates between hypothesizing one or more edges to fix and manually verifying these edges with the user until consistency is restored.

Finally, the outer loop of the overall algorithm allows the ranking algorithm to generate additional candidate edges — this allows the ranking algorithm to take advantage of more subtle matches as the PCCs begin to form. The priority queue will gradually be emptied as each PCC obtains a sufficiently redundant set of positive edges and enough negative edges to be complete.

Details of each step in the review algorithm are described in the following sections. First we describe the redundancy criterion in Section 5.2. Then we will define candidate edge generation in Section 5.3 and decision-making in Section 5.4. Inconsistency recovery is covered in Section 5.5. Finally, we describe the refresh and termination criteria in Section 5.6.

5.2 POSITIVE AND NEGATIVE REDUNDANCY

In this section we define criteria that (1) increases our confidence that PCCs are correct by enforcing a minimum level of redundancy and (2) prevents edges that exceed this redundancy from being reviewed. At a minimum each PCC must be a tree of positive edges, but when errors can occur, it's difficult to be confident that all nodes in the PCC

are really annotations from the same individual. By adding a redundant edge we either increase the confidence that other edges are correct or detect an inconsistency which can be resolved with the algorithm in Section 5.5. However, the gains in confidence from adding each additional edge are diminishing. Therefore, it is desirable to achieve a minimum level of redundancy, but once this has been achieved we should prevent additional redundant edges from being reviewed. We formalize this minimum level of redundancy in two forms. The first is for positive edges within PCCs and the second is for negative edges between PCCs.

(1) positive-redundancy — A PCC is k-positive-redundant if its positive subgraph is k-edge-connected (contains no cut-sets involving fewer than k positive edges [258]), or if the PCC has k or fewer nodes and the union of positive and incomparable edges is a complete graph.

(2) negative-redundancy — A pair of PCCs C and D is k-negative-redundant if there are k negative edges between C and D, or if there are $|C| \cdot |D|$ negative or incomparable edges between them.

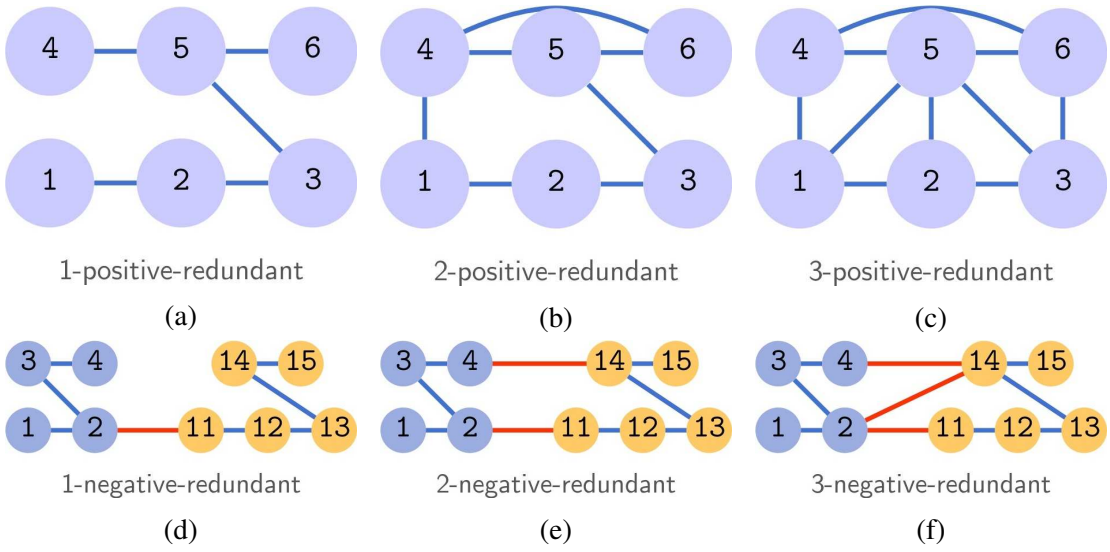


Figure 5.3: **Examples of k-redundant PCCs** This shows examples of positive (top) and negative (bottom) redundancy. The positive edges are colored blue and the negative edges are colored red. Choosing the level of redundancy is a trade-off between the number of required reviews and the confidence that the reviews are correct.

The example in Figure 5.3 illustrates different levels of redundancy. To understand

these criteria better, consider what it means for a PCC that has been determined to be k-positive-redundant to have an undiscovered error. The error means that the PCC really should be split into (at least) two separate PCCs. Suppose these PCCs correspond to animals C and D. If the combined PCC is k-positive-redundant then are k separate undiscovered mistakes connecting C and D, and there must also be no negative edges connecting C and D. This may be plausible if C were identical twins, but these tend not to occur for species where the distinguishing markings (*e.g.* hip and shoulder of zebras) are mostly random. In other words, an error only becomes undiscoverable if a reviewer makes the same mistake with different annotations from the same individual k times. Note that k can be different for positive and negative redundancy, but in our current implementation we use $k = 2$ for both positive and negative redundancy.

5.2.1 Checking redundancy

We now describe how to check if a consistent PCC with n nodes and m edges is k-positive-redundant. An PCC that contains a negative edge is inconsistent and is never considered as positive-redundant. In the case where $n \leq k$, the PCC is positive redundant if all possible edges are either positive or incomparable. This can be determined in $O(m)$ by checking if the sum of the number of positive and incomparable edges between the nodes in the PCC is equal to $\binom{n}{2}$. Otherwise, in the more interesting case, when $n > k$, a PCC is positive-redundant iff it is k-edge-connected — *i.e.* it is impossible to disconnect the nodes in the PCC by removing any set of $k - 1$ edges. In practice, we are primarily concerned with the case when $k = 2$. This special case of 2-edge-connectivity is also called bridge-connectivity, and can be tested for in $O(n + m)$ [258], [259]. The basic idea is to trace cycles encountered during a depth-first-search of the graph and mark all edges that are part of a cycle. Any unmarked edge is not part of any cycle, and is called a bridge. Removing any bridge edge would disconnect the PCC. Thus, if no bridge exists then the PCC is 2-edge-connected.

In the general case when $k > 2$, edge-connectivity can be determined in $O(mn)$ amortized time [260]. This involves first computing a small (not necessarily the smallest) dominating set. A small dominating set can be computed using a greedy algorithm that starts with an empty set and iteratively adds an arbitrary node that is not in or adjacent to

the current set until all nodes are in or adjacent to that set. Then an arbitrary node in the dominating set is chosen. The local-edge-connectivity is computed between the chosen node and every other node in the dominating set. The local-edge-connectivity between two nodes is simply the maximum flow between those nodes, if all edge capacities are equal to 1. The edge-connectivity of the entire PCC is the minimum of (1) the minimum degree of the PCC and (2) the minimum computed local-edge-connectivity.

We now describe how to check if two PCCs C and D with sizes n_1 and n_2 are k-negative-redundant. This can be done in $O(n_1 n_2)$ time using adjacency lists and set intersections to check if the number of negative edges between the nodes in C and D is greater than k. For each node in C we simply check if any node in D is in the adjacency list (stored as a set) of that node. The number of times this is true is the number of negative edges between C and D.

Using this redundancy criterion we are able to find and remove edges from the priority queue that are no longer needed. When a positive edge is added within a single PCC, we check for positive-redundancy. If this passes, all remaining internal edges for that PCC may be removed from the priority queue. When a negative edge is added between a pair of PCCs, we run the negative-redundancy check on the pair, and if this passes, all remaining edges between the PCCs may be removed from the priority queue. When a positive edge is added between a pair of PCCs, the two PCCs are merged into a single new PCC C', and the above negative-redundancy check must be run between C' and all other PCCs having a negative edge connecting to C'. It can be shown that if the graph is in a consistent state, that these are the only updates required.

As a final note, consider the case where a PCC is composed of two positive k-edge-connected subgraphs joined by a single positive edge. While the entire PCC is not positive redundant, much of it is. In this case, we do not need to review any edge within any k-edge-connected component of the PCC. We can dynamically remove these from the priority queue by checking when a new edge popped off of the priority queue is within an existing PCC. We can check if the local-edge-connectivity [260] — *i.e.* the maximum flow — between the edge's endpoints is at least k. If the local-edge-connectivity between those nodes is at least k, then that edge is part of a k-edge-connected component and can be ignored.

5.2.2 Redundancy augmentation

In addition to determining if existing edges are redundant, it would be useful determine a small set of edges that would make an existing PCC positive-redundant or two PCCs negative-redundant. Reviewing these edges would help expose any undiscovered errors in the graph.

To find edges that would complete positive-redundancy for a PCC, we compute a k-positive-augmentation. This is equivalent to the problem of finding a minimum k-edge-augmentation. When $k = 2$ the problem is called bridge augmentation and can be computed $O(m)$ [258] as long as all edges in the complement of the PCC can be used in the augmentation. However, if the PCC contains incomparable edges, then these edges cannot be used. We can address this by using a weighted variant of the problem, setting the weights of the incomparable edges to ∞, and searching for a minimum cost augmentation. However, the weighted variant of this problem is NP-hard, even for $k = 2$, but can be approximated within a factor of 2 [261]. We make use of this algorithm later in Section 5.3

To find a set of edges that would complete negative-redundancy for a pair of PCCs, we compute a k-negative-augmentation. Computing this set of augmenting edges is trivial. Initialize an empty augmenting set. Iterate through all combinations of nodes between the two PCCs. For each combination of nodes, if there is no existing edge between those nodes in the graph, add it to the augmenting set. Once k edges have been added to the augmenting set or there are no more node combinations, stop and return the augmenting set. Note, that we do not use this algorithm in practice because we use a probabilistic termination criteria instead of enforcing that all pairs of PCCs are negative-redundant.

5.3 CANDIDATE EDGE GENERATION AND PRIORITIES

In this section we describe the first step of the algorithm where candidate edges are generated and then prioritized for review. There many ways that candidate edges can be generated. Different sets and orderings of candidate edges will impact different properties of the graph at different rates. Therefore, we choose candidate edges to depend on what properties of the graph we want to manipulate. In the context of identifying individual animals, the most obvious manipulation is to reduce the number of PCCs in the graph by

adding positive edges between existing PCCs, thus merging them into one. A less obvious property that we can manipulate is the fraction of PCCs that are positive redundant. By increasing this fraction to 1, we discover mistakes that have been made, which can be fixed using the algorithm in Section 5.5. This ultimately decreases the likelihood that any PCC contains a mislabeled positive edge. In each iteration outer loop of our main algorithm, we alternate between first generating candidate edges to merge PCCs, and then generating candidate edges to ensure that all PCCs are positive redundant.

To generate candidate edges that may merge existing PCCs we use the LNBNN ranking algorithm from Chapter 3. We issue each annotation as a query to the ranking algorithm, and form edges from the top results of the ranked lists. We then assign a priority to each new candidate edge. We use the pairwise algorithm from Chapter 4 to estimate the positive, negative, and incomparable probabilities of each edge. Any edge whose maximum positive, negative, or incomparable probability is above the threshold for automatic decision-making is ranked according to this probability. All other edges are ordered by their positive probability. This ensures automatic decision-making is first, followed by an ordering of the edges needed for manual review that are most likely to be positive and therefore add the most information to the graph. It is desirable to add positive decisions to the graph first because (1) they are the most important edges with respect to determining the animal identities, and (2) larger PCCs increase the number of edges that can be skipped using the redundancy criterion.

The first iteration of the outer loop the algorithm generates edges using the ranking algorithm. Review of these edges continues until the priority queue is empty, or we determine that the candidate edges should be refreshed using the algorithm we will describe in Section 5.6. This ends the current inner loop, and as long as the algorithm has not converged, it proceeds to the next iteration of the outer loop. In this next iteration, we generate candidate edges to ensure that all PCCs are positive redundant.

To generate candidate edges that will make PCCs positive redundant, we use the positive-augmentation algorithm from Section 5.2.2 to generate edges. These edges are assigned priorities in the same way, however in this iteration the refresh criterion is disabled. This enforces that if each edges is reviewed as positive, then all PCCs will be positive-redundant at the end of this inner loop. However, sometimes an edge will not

be reviewed as positive. If it is reviewed as negative, then it will introduce an inconsistency and be handled by the algorithm in Section 5.5. If it is reviewed as incomparable, then the PCC will only be positive-redundant if all edges between the nodes in the PCC have been reviewed. Because we want to ensure that all PCCs are positive-redundant, we delay alternating back to the ranking algorithm. Instead, we simply regenerate candidate edges using the positive-augmentation algorithm in the next iteration of the outer loop until all PCCs are positive-redundant. Therefore, at the end of this process all PCCs are positive-redundant by construction.

After we have ensured positive-redundancy, the outer loop alternates back to generating candidate edges using the ranking algorithm. In this way our algorithm alternates between two modes, first searching for merges, and then ensuring redundancy. This continues until the termination criterion that we will describe in Section 5.6 is satisfied. At the end of this process it is guaranteed that all PCCs are consistent and positive-redundant.

5.4 MAKING DECISIONS

Now that we have generated and prioritized a set of candidate edges we come to the core of the inner loop — decision-making. Because of the surrounding structure of the graph framework, this step is quite simple. Given a popped edge from the priority queue, we check if any of the positive, negative, and incomparable state probabilities produced by the pairwise algorithm are above their automatic decision threshold (set externally as a hyperparameter). If the edge cannot be automatically reviewed we issue a request for user feedback. Once we have obtained feedback for an edge — either automatically or manually — the edge is added to the appropriate edge set. After the new edge is added, we update candidate edge priorities discussed in Section 5.2. If the new edge causes an inconsistency, then we drop into inconsistency mode, which we will discuss in Section 5.5.

For each decision we record a user-id to identify the reviewer or algorithm making the decision. We also follow the approach of [262] and store a user-specified categorical confidence value of unspecified, guessing, not-sure, pretty-sure, and absolutely-sure (with associated integer values 0, 1, 2, 3, and 4). The user-id allows us to differentiate between edges that were automatically reviewed from those that were manually reviewed. While

this is not directly used in this algorithm description, it enables a variety of possible post-processing techniques, *e.g.* manually reviewing automatically reviewed edges between PCCs containing only two annotations. However, the user confidence contributes to the edges weights in the error detection and recovery algorithm from Section 5.5.

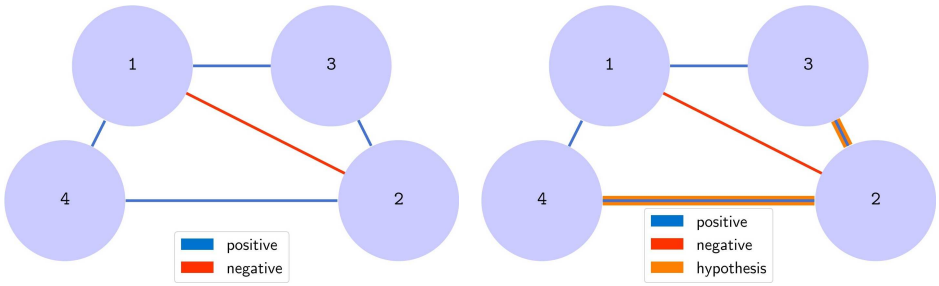
5.5 RECOVERING FROM INCONSISTENCIES

In Section 5.2 we described a redundancy criterion that exposes errors by introducing inconsistencies. In this section we describe an algorithm for fixing these errors and recovering from these inconsistencies.

Whenever a decision is made that either adds a negative edge within a PCC or adds a positive edge between two PCCs with at least one negative edge between them, the graph becomes inconsistent. In both cases the result is a single PCC C with internal negative edges. The goal of inconsistency recovery mode is to change the labels of edges in order to make the subgraph formed by the nodes of C and all of their edges consistent. An inconsistency implies that a mistake was made, but does not necessarily determine which edge has the wrong label. Therefore, we develop an algorithm to hypothesize the edge(s) most likely to contain the mistake(s) using a minimum cut. If the hypothesis is correct, and all the labels on these edges were changed, then we show that the PCC would either become consistent or be split into multiple consistent PCCs. Because the hypothesis might not be correct, we present these edges to a user for manual review, and if the user agrees with the hypothesis, the algorithm completes. Otherwise, the new information received by the user it taken into account, and the hypothesis is recomputed. An example of an inconsistent PCC with hypothesized edges is illustrated in Figure 5.5.

5.5.1 Hypothesis generation

The procedure alternates between steps of generating "mistake hypothesis" edges, and presenting these to the user for review. The "hypothesis generation algorithm" returns a set of negative edges or a set of positive edges, which if re-labeled as positive or negative respectively would cause C to become consistent. For simplicity, we focus first the case where C contains exactly one negative edge. It will not be hard to extend to the general case.



(a) (b)

Figure 5.5: **An inconsistent PCC** Any PCC containing at least one negative edge is inconsistent. Subfigure (5.4a) shows an inconsistent PCC, and (5.4b) shows the same PCC where the edges hypothesized to be errors are highlighted.

First consider a cut of C that disconnects the endpoints of the negative edge. If the labels on these edges were changed from positive to negative or incomparable, then the inconsistent PCC would be split into multiple consistent PCCs. Alternatively, if the label on the negative edge was changed to positive or incomparable, the PCC would no longer be inconsistent. Therefore, the algorithm starts by creating a minimum s-t-cut using the subgraph of C containing only positive edges. The endpoints of the single negative edge are the terminal nodes s and t. Unlike in our preceding discussion where the edges only have positive/negative/incomparable labels the edges will now have weights that reflect a measure of confidence in their current label. In the case where the negative edge is correct, this will encourage the minimum cut to return the positive edges that are most likely to be mislabeled.

The weight of an edge in this cut problem is the sum of three values:

(1) its positive probability previously computed using the pairwise classifier,

(2) the number of consecutive times that edge was manually reviewed with its current label, and

(3) an integer ranging from 0 to 4 indicating the confidence of the most recent review (see Section 5.4).

Because we are using a minimum cut, the algorithm will find the lowest confidence cut set of these edges. The higher each of these values is, the more likely that the edge is correctly labeled. The positive probability offers a baseline estimate of this confidence. Using the number of manual reviews means that if a user disagrees with a hypothesis, the weight on that edge will be increased and the algorithm will be encouraged to select a

different edge. The confidence performs a similar role, speeding up the pace at which the algorithm tries different edges.

Using this weighting scheme we find the minimum s-t-cut, which returns a cut set of positive edges. We now need to decide if it is more likely that the positive cut-edges should be relabeled, or the negative edge should be relabeled. We compare the total weight of cut positive edges with the weight of the negative edge (weighted using the same scheme). If the positive weight is smaller, the algorithm suggests that the cut positive edges should be relabeled as negative. Otherwise, it suggests that the negative edge should become positive.

5.5.2 Generalization

So far, we have only considered the case when only one negative edge exists in a PCC. However, in practice it is possible for multiple negative edges to exist within a PCC. In this case we can slightly modify the inconsistency recovery algorithm. The modification is simple. Instead of using a minimum s-t-cut, we use a multicut to disconnect the endpoints in all pairs of negative edges. Multicut is NP-hard, but a simple approximation algorithm is to perform a minimum s-t-cut for each set of terminal nodes, and then take the union of the resulting cut [263], 203–208. When considering which edges to return we consider all negative edges together, comparing the sum of their weights to the sum of the positive edge weights.

5.5.3 Hypothesis review

The user iterates through each hypothesis edge and chooses (1) to agree with the hypothesis and change the label of the edge, or (2) to disagree with the hypothesis and keep the edge label. In the case where the user agrees that a positive label should be changed, it does not matter if the label is changed to negative or incomparable, it will still remove the positive connection. A similar argument is true when the user agrees to change a negative label to either positive or incomparable; either way, the negative edge between the nodes in the PCC is removed. As long as the user agrees with the hypothesis and the hypothesis set is non-empty, the iteration continues. If there are no more hypothesis edges, then either all inconsistencies were removed or all PCCs were split into

multiple consistent PCCs. In the case where the reviewer disagrees with the algorithm, the new review is recorded and we generate a new set of hypothesis errors. Because reviewing the edge increases its weight, the algorithm will be forced to look elsewhere for a cut. This alternation between hypothesis generation and user review repeats until the reviewer agrees with all hypothesis edges, which means that all inconsistencies have been eliminated

Fixing inconsistencies can result in splitting C into multiple PCCs. This may invalidate implicit reviews inferred from redundancy either within or incident to this subgraph. Therefore, we recompute positive-redundancy within each new PCC, ignoring edges where the criterion is satisfied and re-prioritizing unreviewed edges where it is no longer valid. A similar process happens for negative-redundancy between each pair of new PCCs as well as between each new PCC and all other PCCs previously negative-redundant with C.

5.5.4 Implementation details

While it might be conceptually simpler to think of inconsistency recovery as a separate mode, in practice it is actually integrated as part of the algorithm's inner loop. The algorithm dynamically maintains a list of all PCCs that contains errors. We disable redundancy checks for any PCCs that contain errors. Hypothesis edges are computed whenever a new inconsistency is created, and these edges are entered into the queue with their normal priority plus 10 to ensure they are reviewed first. It can be shown that as long as the user agrees with the current hypothesis edges computed so far, recomputing the new hypothesis edges will always be the same as the remaining hypothesis edges. This implementation is an important first step for generalizing the graph algorithm into a distributed setting with multiple reviewers.

5.6 REFRESH AND TERMINATION CRITERIA

In this section we discuss the criteria we use for both determining when to refresh candidate edges and when to stop the algorithm altogether. We first consider the need for a refresh criterion. As the review algorithm proceeds, we should only continue to manually review edges as long as the algorithm is consistently generating edges that —

once reviewed — change the PCCs. This happens whenever a review merges two PCCs into one or splits one PCC into two. Because splits only happen during inconsistency recovery, we are primarily concerned with searching for new positive edges. However, at some point the candidate edges may no longer contain positive results, but undiscovered positive matches may still exist. This is because LNBNN, working initially with each annotation having a separate label, can miss more subtle but correct matches, especially when there are several annotations for an individual with multiple viewpoints. As the labeling improves, so does the reliability of LNBNN. Recall that the verification algorithm is only run after candidate edges are generated. If the edges required to complete the underlying real PCCs are missing from the candidate set, then nothing can be done. We therefore must develop a "refresh criterion" to determine when to stop and replace the current priority queue with new LNBNN matches. This will be done by predicting if none of the remaining edges would change the PCCs, and then recomputing candidate edges when this happens. However, before we describe this process, we consider the problem of termination, which will turn out to have a similar solution.

Similar to the refresh criterion, we must be able to determine when the algorithm should stop. To ensure that the identification is perfect, the algorithm would need to use all $\binom{|V|}{2}$ edges as candidates and then terminate using the deterministic convergence criterion explained in the introduction of this chapter. Recall that the deterministic convergence criterion only stops the algorithm once each PCC is positive redundant and each pair of PCCs is negative-redundant. This essentially results in a brute-force search, that requires $O(|V|^2)$ reviews, and is only feasible if (1) the number of annotations is very small, or (2) all edges can be automatically reviewed. However, even if all edges can be automatically reviewed the quadratic computation required to run the verification algorithm on all pairs of annotations might be too computationally expensive for very large databases. Therefore, in practical circumstances, we turn towards probabilistic methods to determine when to stop. Like, the refresh criterion, this can be determined — in part — by predicting if new reviews will change the PCCs.

5.6.1 Convergence as a Poisson process

Both the review and termination criteria can be addressed by considering the question: "Will there be a label-changing review anytime soon?". A *label-changing review* is one that changes the name labeling of the annotations, *i.e.* it is either a positive edge that merges two PCCs or a non-positive edge that splits one. Correct and label-changing reviews improve the accuracy of the identification by increasing the similarity — in terms of the name labeling — between the predicted PCCs and the real underlying PCCs. However, producing a label-changing review has a cost: the number of manual reviews since the last label-changing review. During the inner loop of the algorithm, when edges popped from the priority queue no longer consistently result in label-changing reviews, the marginal gains in identification accuracy that could be made from continuing are outweighed by the cost of manual review. In this circumstance it is best to break out of the inner loop. Note that if any label-changing reviews were made during that loop, we should refresh candidate edges and start a new loop because a refresh could result in new high priority label-changing edges. On the other hand, if no label-changing reviews were made in the loop, then refreshing will have no benefit, and the algorithm should terminate.

Thus, the task is to construct a criterion that determines when edges on the top of the priority queue are no longer label-changing. In this way we directly address the refresh criterion and indirectly address termination criterion. This is direct in the case of the refresh criterion because when the name labeling is more accurate, the LNBNN ranking will improve. When we directly measure that the next reviews in the current priority queue are unlikely to be label-changing, and the name labeling of the decision graph has changed, then it is more likely that we would review a label-changing edge on the top of a new set of candidate edges sorted by priority.

This task indirectly addresses the termination criterion because instead of stopping once the probability that identification is complete is high, the algorithm simply stops when the cost in terms of manual labor is too high. However, if we were to directly address the termination criterion, we would have to estimate the probability that undiscovered merge and split cases exist. This probability depends on the effectiveness of the ranking algorithm. Even if our estimate of this probability was perfect, once the ranking algorithm starting producing label-changing reviews at a rate no better than random edge generation,

it would take an enormous amount of manual effort to push this probability passed a desired threshold. Thus, instead of providing guarantees about identification accuracy our termination criterion achieves a trade-off between the number of manual reviews and the cost of identification.

Based on these observations we estimate the probability that "there will be a label-changing review soon". We define "soon" using a patience parameter a, defined as the maximum number of consecutive reviews that a manual reviewer is willing to do between label-changing reviews. Let $L{=}1$ be the event that a review is label-changing and $L{=}0$ otherwise. Because reviews are ordered, denote if the i^{th} review is label-changing as L_i, and denote the index of the next review as n. Let $C = \bigvee_{i=n}^{n+a} L_i$ be a binary random variable that takes the value $C = 1$ in the event that any of the next a reviews will be label-changing. Thus, the aforementioned question can be addressed by measuring the probability of the event $C = 1$. We can periodically check if $\mathrm{P}(C = 1)$ is less than a threshold, and if so, we stop the current loop and either refresh or terminate.

We model the event $C{=}1$ as a Poisson processes, but for this to be appropriate, L_i must follow a uniform distribution. This would be true if the edges were reviewed in a random order. However, the priority queue orders edges more likely to be label-changing first, causing L_i to follow a right skewed long tail distribution and violate Poisson assumptions. Even so, the use of a Poisson model can be justified by considering a sliding window along the distribution of L_i. Recall that we only need to make predictions about the next a reviews in the future, thus we are only concerned with a small window to the right on the distribution. Assuming the long-tailed distribution is monotonic decreasing, we can use a small window in the past to estimate an upper bound on probability of $C{=}1$ in the future. As the window moves to the right, the interval on the distribution becomes increasingly approximately uniform and the tightness of the bound improves and eventually becomes tight. This is because the order of the remaining reviews becomes random once the prioritization algorithm cannot distinguish positive from negative cases. In this case the Poisson model becomes appropriate. Thus, the use of a Poisson model with a sliding window allows us to approximate an upper bound on $\mathrm{P}(C{=}1)$, and the smaller $\mathrm{P}(C{=}1)$ is, the more accurate our estimate will be.

5.6.2 Details of Poisson convergence

Having justified its use, we model C as a Poisson process, which is determined by a rate parameter μ and an interval length parameter a. We can measure μ as the fraction of recently observed manual reviews that were label-changing using an exponentially weighted moving window. We initialize $\mu_0 = 1$ to denote that it is likely that the first review will be label-changing and because it will ensure our estimated probability is an upper bound. Then, after each new review we update the parameter as $\mu_{i+1} \leftarrow \ell_i\alpha + (1 - \alpha)\mu_i$, where ℓ_i=1 if the i^{th} review was label-changing and 0 otherwise. The exponential decay $\alpha = 2/(s + 1)$ is determined by a span parameter s, which roughly represents the number of previous reviews that are significant. Using this model, the desired probability that any of the next a reviews will be label-changing is $\mathrm{P}(C{=}1) = 1 - \exp(-\mu a)$. The example in Figure 5.6 illustrates the behavior of the criterion using a synthetic dataset. In this example we use a window span of $s = 20$, a patience of $a = 20$, and a threshold of $\tau = 0.135$. [1]

[1] A better choice would be to set these parameters such that $\tau = 1 - \exp\left(-a\,(s-1)^{a}\,(s+1)^{-a}\right)$ is satisfied. This will guarantee convergence after a maximum of a consecutive non-label-changing reviews. Unfortunately, this was discovered after the completion of this thesis, so the parameters in our experiments do not satisfy this property. Suggested values for future work are $s = 20$, $a = 72$, and $\tau = 0.052$.

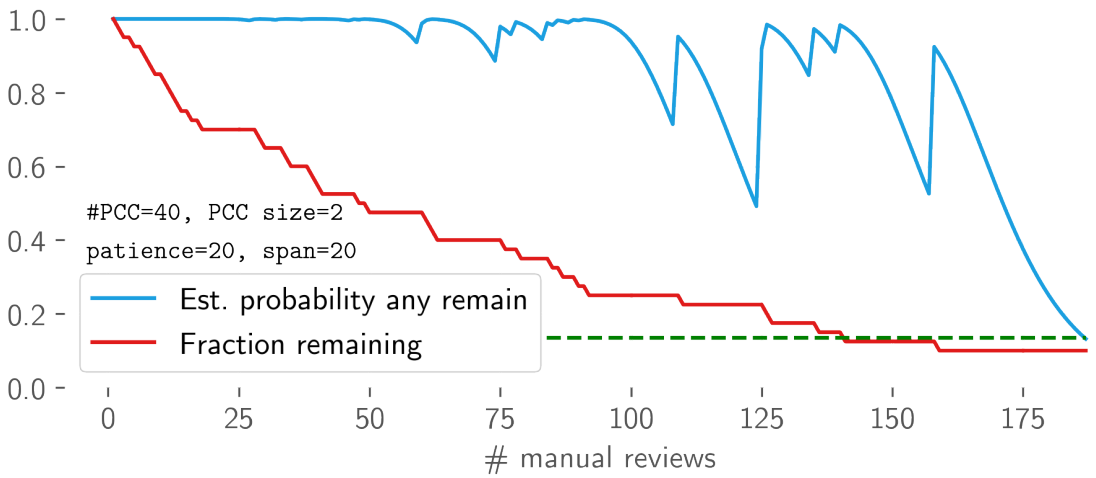


Figure 5.6: **The convergence criteria on a synthetic dataset** The convergence criteria is applied to a synthetic dataset with 40 names and 2 annotations per name. The red line indicates the fraction of label-changing reviews that remain undiscovered. The blue line is the probability that at least one of the next 20 reviews will be label-changing. Notice that the blue line dips when the red line flattens. The process terminates once this probability drops below a threshold, which is denoted by the green dotted line.

5.7 EXPERIMENTS

In this section we design an end-to-end experiment where we start with a set of annotations without any name labels, and then we proceed to construct all the names. This simulates identification events like the GZC and provides insight into how the algorithms behave in practice. Because our algorithms are semi-automatic, we simulate a noisy user response using ground truth data. Given a pair of annotations, the simulated user returns the ground truth classification 99% of the time, making errors 1% of the time uniformly at random. These assumptions may be simple and too inaccurate to model an expert reviewer, but it will serve to demonstrate our graph algorithm's ability to recover from errors. We will measure the simulation's accuracy and error as a function of the number of manual reviews. Using these measures we will compare our graph algorithm to alternative techniques to demonstrate that it produces accurate identifications with fewer errors using significantly fewer manual reviews.

For this experiment, we will use the plains and Grévy's zebras datasets previously described in Section 3.5.1. Because some of our algorithms will require training, we split each dataset into a training set and a testing set each containing half of the names.

The training set will be used to learn the pairwise classifiers and determine thresholds for automatic classification. The details of training and testing sets are summarized in Table 5.1.

Table 5.1: **Database statistics for graph identification experiments** Each database is split into a training and a testing set. We report the number of names, the number of annotations, and the average number (as the mean and standard deviation) of annotations per name for each set. Additionally, we report how many edges (pairs of annotations) were used to train the classifiers.

	Names	Annots	Annots size	Training edges
Training	564	2752	4.88 ± 3.74	24328
Testing	563	2968	5.27 ± 4.14	-

(a) Plains zebras

	Names	Annots	Annots size	Training edges
Training	363	1164	3.21 ± 3.03	9360
Testing	363	1119	3.08 ± 2.82	-

(b) Grévy's zebras

The experiment will compare the three different methods of determining the identities of annotations based on the algorithms defined in this thesis. These algorithms are: (1) `ranking` — the ranking algorithm from Chapter 3, (2) `rank+clf` — the same ranking algorithm but augmented with the automatic classification algorithm from Chapter 4, and finally (3) `graph` — the graph identification algorithm introduced in this chapter. In the case of `graph`, the procedure from Section 5.1.1 can be directly applied. However, in order to compare `graph` to `ranking` and `rank+clf` we must define baseline methods to determine the ordering of the reviews and when to stop. Details of these identification procedures are given in the next subsection.

5.7.1 Identification procedures

We design the procedure for `ranking` to be similar to the approach described in Section 1.3 that was used in the GZC. This algorithm does not require a pre-training phase and thus only the testing set is used. Given the unlabeled annotations, we index the database, issue each annotation as a query, and collect the top 5 results from each ranked list. The resulting pairs of query and database annotations are stacked and sorted

by LNBNN score. The user reviews each pair in the list sequentially. As was done in the GZC, all pairs are all manually considered and verified, and all inconsistencies are ignored. In the GZC the choice to re-run the ranking algorithm was made manually, making it difficult to reproduce. Therefore, we simplify the experiment by choosing to run the ranking algorithm once. Consistency checks are not applied because developing these is the point of the graph algorithm.

The procedure for `rank+clf` is similar to the one used for `ranking`. The main difference is that we use the automatic classification algorithm to predict pairwise probabilities for each pair of top ranked query and database annotations. If the probabilities of a class are above a threshold, then the pair is automatically reviewed. This has the effect of reducing the total number of manual reviews. The rest of the procedure is unchanged. The pairwise classifier is trained on the training set, and the algorithm is evaluated on the test set. We choose automatic thresholds by finding the thresholds that result in a specified false positive rate on a validation dataset. Because `rank+clf` has no mechanisms for error recovery, we choose conservative automatic thresholds by specifying an acceptable false positive rate of 0.001. This results in positive, negative, and incomparable thresholds of 0.976, 0.991, and 0.5 respectively for plains zebras, and 0.997, 0.998, and 1.0 for Grévy's.

Finally, we quickly recap the procedure for `graph` which was defined in Section 5.1.1. The algorithm begins by using LNBNN to search for candidate edges, which are then assigned probabilities and inserted into a priority queue based on these probabilities. As candidate edges are removed from the queue they are either automatically or manually classified based on probability thresholds. Connectivity information is used to enforce a minimum level of redundancy, to prevent extraneous redundancy, and to ensure consistency. The convergence criterion determines when candidate edges should be refreshed and when the algorithm should terminate. We use the same pairwise classifier from `rank+clf`, but due to the graph algorithm's error recovery mechanisms we can choose more aggressive thresholds. However, we have found that the algorithm is sensitive to this parameter, therefore we only slightly increase the acceptable false positive rate from 0.001 to 0.0014. This results in positive, negative, and incomparable thresholds of 0.969, 0.986, and 0.5 respectively for plains zebras, and 0.989, 0.992, and 1.0 for Grévy's.

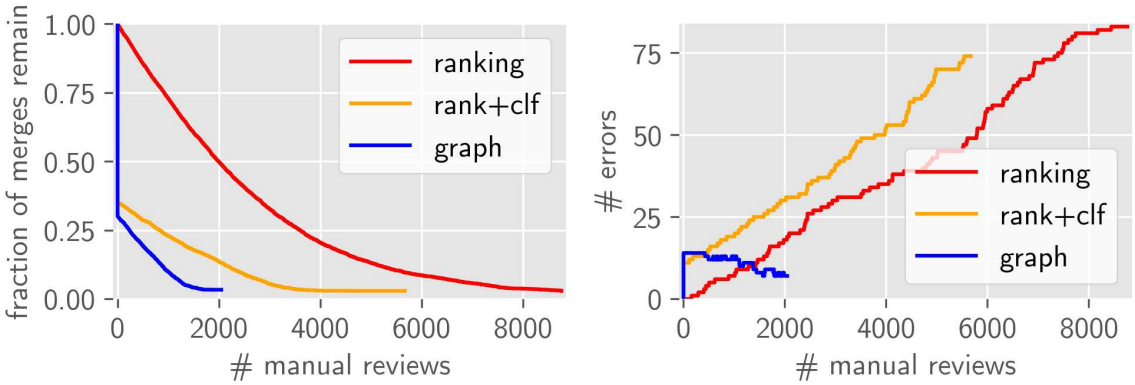
For the convergence criterion we use a patience of $a = 20$, a window span of $s = 20$, and a termination threshold of $\tau = 0.135$.

5.7.2 Results

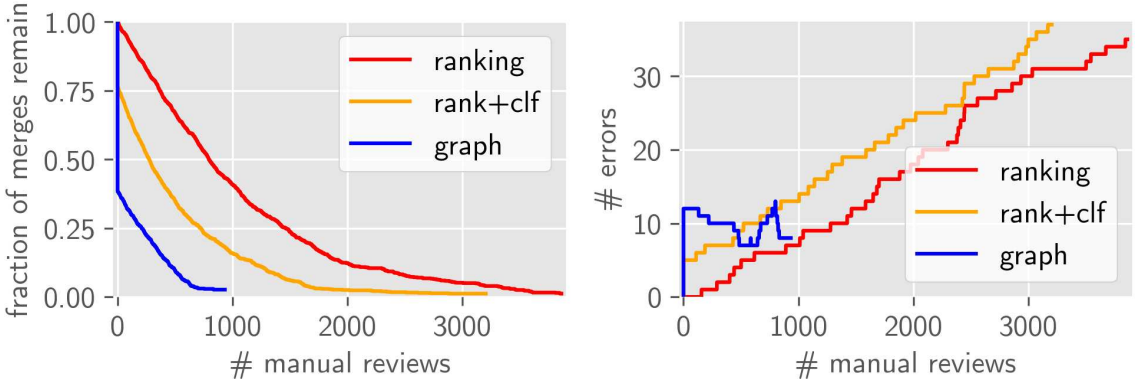
We run the simulation for all combinations of datasets and algorithms. During the simulation, after each review decision is made we record two measurements pertaining to accuracy and error. The accuracy measurement is the number of merges remaining — *i.e.* the number of PCCs that must be merged — before all individuals have been identified. This is the number of edges in a spanning forest of the ground truth positive subgraph minus the same measurement but applied to the subgraph of all correctly predicted positive edges. The error measurement is the total number of edges with a predicted label that differs from the ground truth match-state. Additionally, for `graph`, after each review decision we record the probability of convergence estimated by the convergence criterion. These measurements are plotted against the number of manual reviews. Figure 5.8 shows these accuracy and error plots, and Figure 5.10 shows the convergence criterion.

The results illustrated in Figure 5.8 demonstrate that `graph` achieves the fewest manual reviews with the fewest errors while still correctly identifying almost all individuals. We first focus on the left part of this figure. The number of remaining merges slowly decreases for `ranking`, which is the only algorithm that requires manual review of each pair. For `graph` and `rank+clf` the initial decrease appears instantaneous due to the automatic classification algorithm, which does not cost manual reviews. Notice, once manual reviews begin the slope of `graph` is steeper than `rank+clf` due to the redundancy mechanisms, which removes extraneous reviews. Furthermore, while `rank+clf` and `ranking` continue until their candidate edges are exhausted, `graph` uses the convergence criterion to terminate shortly after the curve flattens. It is noteworthy that `graph` completes identification using less than 25% of the manual reviews needed by `ranking`, which models the way that we counted individuals in the GZC.

We now turn our attention to the right of Figure 5.8. The `ranking` and `rank+clf` algorithms do not have mechanisms for error recovery, and thus their error steadily increases over time. However, `graph` is able to recover from many of these errors and achieve a low error rate despite starting with more errors due to an aggressive auto-



(a) Plains zebras



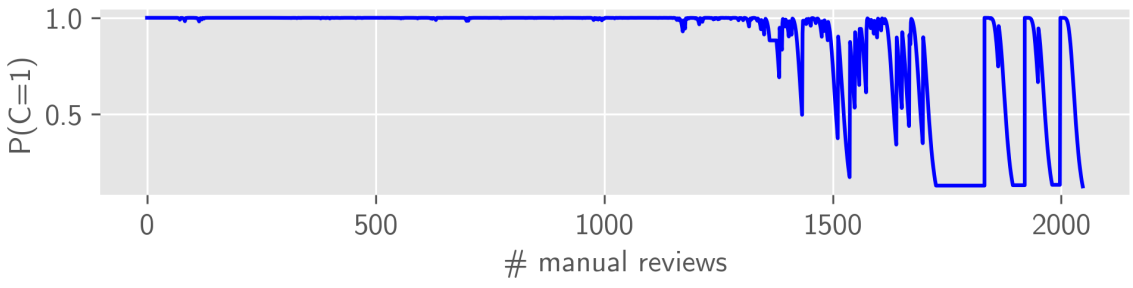
(b) Grévy's zebras

Figure 5.8: **Simulation experiment** The user simulation experiment compares the three identification algorithms defined in this thesis. The plot on the left indicates the identification accuracy using the number of remaining merges and the plot on the right counts the number of errors made (lower is better in both cases). The best results are clearly achieved by `graph`.

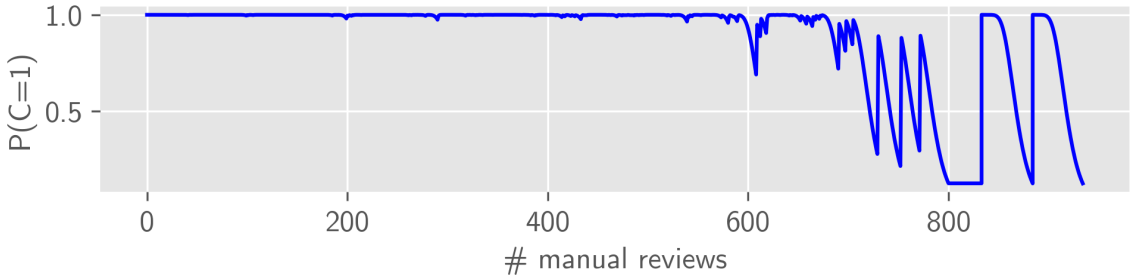
classification threshold.

5.7.3 Error cases

To further analyze the predictions of the graph the simulation, we compare the node groupings of the predicted PCCs to the real ground truth PCCs. For convenience, we will still refer to a group of nodes as a PCC. In this analysis we categorize groups of predicted PCCs and their corresponding real PCCs as one of three types: correct, split, or merge. Consider an example where we are given a graph with 8 nodes and the real PCCs are $\{\{a,b,c\},\{d\},\{e,f\},\{g\}\}$, and we predict the PCCs $\{\{a,b\},\{c\},\{d,e,f\},\{g\}\}$. Using this example we define the three group types:



(a) Plains zebras



(b) Grévy's zebras

Figure 5.10: **Measured refresh and termination probabilities** The probability that the next reviews will be label-changing ($\mathrm{P}(C{=}1)$) is high while new merges are discovered. Once the probability falls under the threshold, positive redundancy is enforced (the flat areas) on existing PCCs, and then candidate edges are recomputed. After an iteration with no label-changing reviews, the process terminates.

(1) A "correct" group is a predicted PCCs that contains all nodes in a real PCC. In the example the PCC $\{g\}$ is a real group.

(2) A "split" error group consists of multiple real PCCs that are labeled as belonging to the same predicted PCC. Each split group contains at least one edge incorrectly labeled as positive. In the example, the real PCCs $\{\{d\}, \{e, f\}\}$ are a split group because in the predicted graph $\{d, e, f\}$ is a PCC.

(3) A "merge" error group consists of multiple predicted PCCs that must be merged into a single real PCC. Each pair of predicted PCCs in a merge group is missing a positive edge. This can happen if an edge is incorrectly labeled, if the ranking algorithm never generates this edge as a candidate, or the termination criterion stops the algorithm before the edge can be reviewed. In the example, the predicted PCCs $\{\{a, b\}, \{c\}\}$ are a merge group because in the real graph $\{a, b, c\}$ is a PCC.

As defined, the split and merge error groups do not cover all cases. We call the previously

described cases "pure" split/merge error groups because there is a mapping between sets of real/predicted PCCs to exactly one predicted/real PCC. However, consider the real PCCs $\{\{x, y\}, \{z\}\}$ and predicted PCCs $\{\{x\}, \{y, z\}\}$. This case cannot be cleanly categorized as a pure split or a merge group, but it contains elements of both. Thus, we call this a "hybrid" case. A slight modification to the above groups can incorporate hybrid cases. First, we modify the definition of a split group. For a predicted PCC containing annotations in more than one real PCC, the split group consists of the subsets of each of these real PCCs that intersect with the predicted PCC. In essence this means we treat $\{\{y\}, \{z\}\}$ as a split group, even though $\{y\}$ is only a subset of the real PCC $\{x, y\}$. Second, for merges, we simply change predicted PCCs to be the predicted PCCs after they have been split. This lets us treat $\{\{x\}, \{y\}\}$ as a merge group.

To analyze statistics of these groups, we gather the set of real PCCs and the set of predicted PCCs for each simulation. Then, we group them into correct, split, and merge groups. For each group type we count the number of predicted PCCs and the number of real PCCs they correspond to. We also measure the average number of annotations in each PCC. These number are reported in Table 5.3.

From this table we observe that `graph` predicts the most correct PCCs and fewest splits in all cases. This is not surprising due to its error recovery mechanism. However, we also see that `graph` also has the most merge cases. The reason for this is that the other algorithms do not have termination criteria. They continued to review pairs, even long after the rankings were no longer consistently label-changing, and the simulated reviewer essentially began to brute-force search the database. In the thousands of extra iterations, the other algorithms managed to find $\sim$20 extra merge cases that were not discovered by `graph`.

We continue our analysis focusing only on the results of `graph`. We measure the average number of PCCs in each error group. Additionally, we consider the smallest and largest PCC in each error group, and we measure the average number of annotations in each. These numbers are given in Table 5.5.

Here, we notice that almost all merge cases for `graph` are the result of failing to match a single annotation to a larger group of annotations. Similarly, most split cases involve splitting just one singleton annotation off of a larger PCC. To gain further insight

into what is causing these errors we will visualize individual cases.

Table 5.3: **Simulation error sizes** This table analyzes the simulation errors. We compare statistics of the predicted PCCs with statistics of the real ground truth PCCs. In each category "Pred PCCs" is the number of predicted PCCs and "Pred PCC size" is the average number of annotations in those PCCs (measured as mean and standard deviation). The "Real PCCs" and "Real PCC size" columns are similarly defined.

		Pred PCCs	Pred PCC size	Real PCCs	Real PCC size
ranking	correct	-	-	459	5.3 ± 4.3
	split	22	12.4 ± 6.9	48	5.7 ± 3.8
	merge	130	2.4 ± 2.7	60	5.2 ± 3.5
rank+clf	correct	-	-	442	5.1 ± 4.1
	split	35	12.5 ± 5.5	71	6.1 ± 4.7
	merge	130	2.3 ± 2.5	60	5.0 ± 3.2
graph	correct	-	-	491	5.3 ± 4.3
	split	2	6.5 ± 1.5	4	3.2 ± 1.5
	merge	150	2.5 ± 2.6	70	5.3 ± 3.3

(a) Plains zebras

		Pred PCCs	Pred PCC size	Real PCCs	Real PCC size
ranking	correct	-	-	322	2.8 ± 2.6
	split	17	9.6 ± 4.9	35	4.7 ± 3.5
	merge	18	2.7 ± 2.4	9	5.4 ± 2.4
rank+clf	correct	-	-	319	2.9 ± 2.7
	split	17	9.5 ± 5.5	38	4.3 ± 3.1
	merge	18	2.8 ± 2.4	9	5.7 ± 2.1
graph	correct	-	-	334	3.0 ± 2.8
	split	5	2.6 ± 1.2	10	1.3 ± 0.9
	merge	40	3.0 ± 2.6	20	5.9 ± 2.5

(b) Grévy's zebras

Table 5.5: **Simulation error group sizes** This table analyzes the error groups in the graph algorithm. A merge error group is a set of predicted PCCs that are incorrectly disconnected, and a split error group is a set of real PCCs that are incorrectly connected. For each case we report the number of error groups, the average number of PCCs in each group, and the average size of the smallest and largest PCC in each group.

	Error groups	Group size	Small PCC size	Large PCC size
split	2	2.0 ± 0.0	2.0 ± 1.0	4.5 ± 0.5
merge	70	2.1 ± 0.5	1.2 ± 0.5	3.8 ± 3.3

(a) Plains zebras

	Error groups	Group size	Small PCC size	Large PCC size
split	5	2.0 ± 0.0	1.0 ± 0.0	1.6 ± 1.2
merge	20	2.0 ± 0.0	1.0 ± 0.0	4.9 ± 2.5

(b) Grévy's zebras

We illustrate several individual error cases from `graph` in Figures 5.11 to 5.16. On the top of each figure we will show the subgraph corresponding to an error group. For a split case this will be the single PCC that must be broken apart, and for a merge case this will be multiple PCCs that should be connected. For merge cases, if no edge connecting the PCCs was ever added to the priority queue as a candidate, then we will insert a dashed edge between two arbitrary annotations. We will highlight the edges with labels that differ from their ground truth. On the bottom we show the annotation pair corresponding to a selected highlighted edge.

Upon inspection, we discover that the split cases in Figures 5.11 and 5.12 are caused by ground truth errors. In each of these cases the automatic verification algorithm made the PCCs positive redundant, thus the simulated reviewer — which is driven by the ground truth — was unable to split the PCC. In fact, we discovered that all split cases measured for `graph` are due to ground truth errors. This means that `graph` did not predict any PCCs that were split cases.

When inspecting merge cases, we also found several caused by ground truth errors, but most were due to challenging image conditions such as viewpoint, occlusion, and pose. These factors either prevented an edge from being generated as a candidate or caused the classifier to produce a low positive probability. Two merge cases for plains

zebras are illustrated in Figures 5.13 and 5.14, and two for Grévy's zebras are illustrated in Figures 5.15 and 5.16.

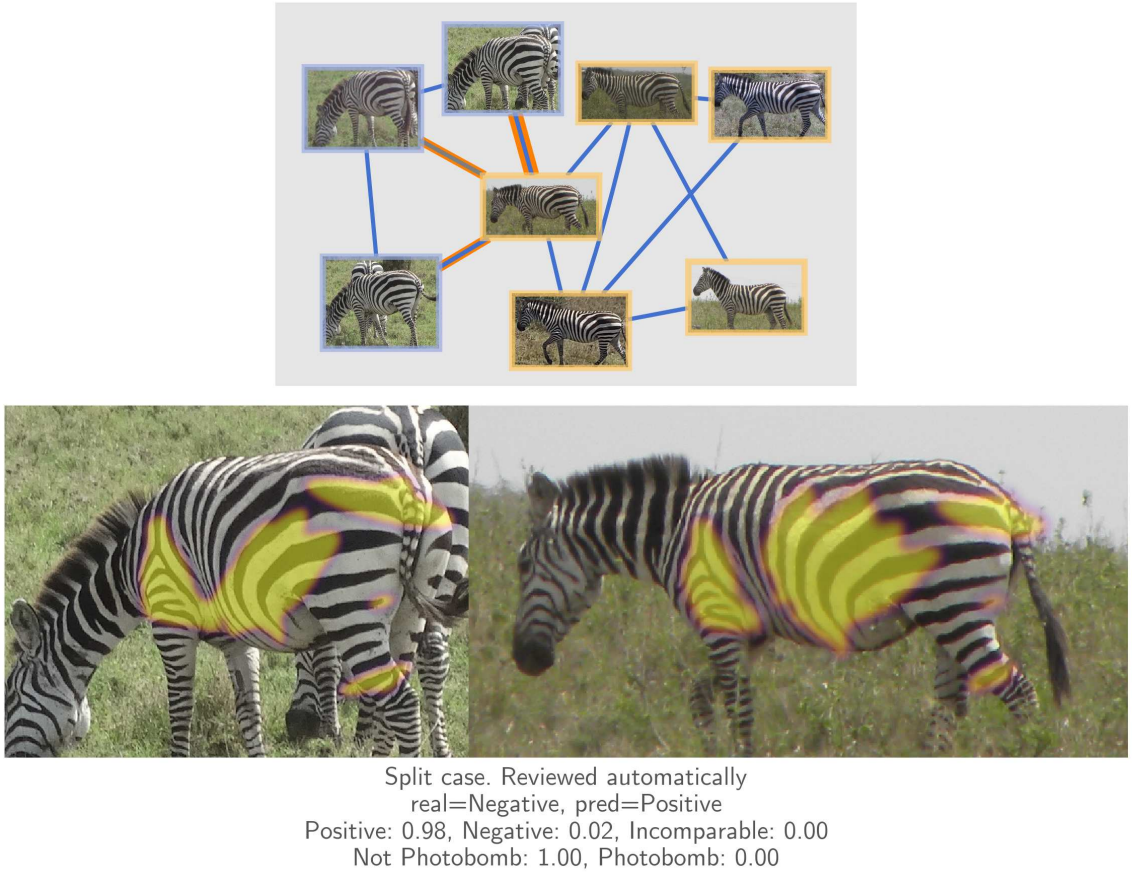


Figure 5.11: **Plains split case due to ground truth error** This was incorrectly reported as a split case. The automatic verification algorithm correctly predicted that this pair is positive.

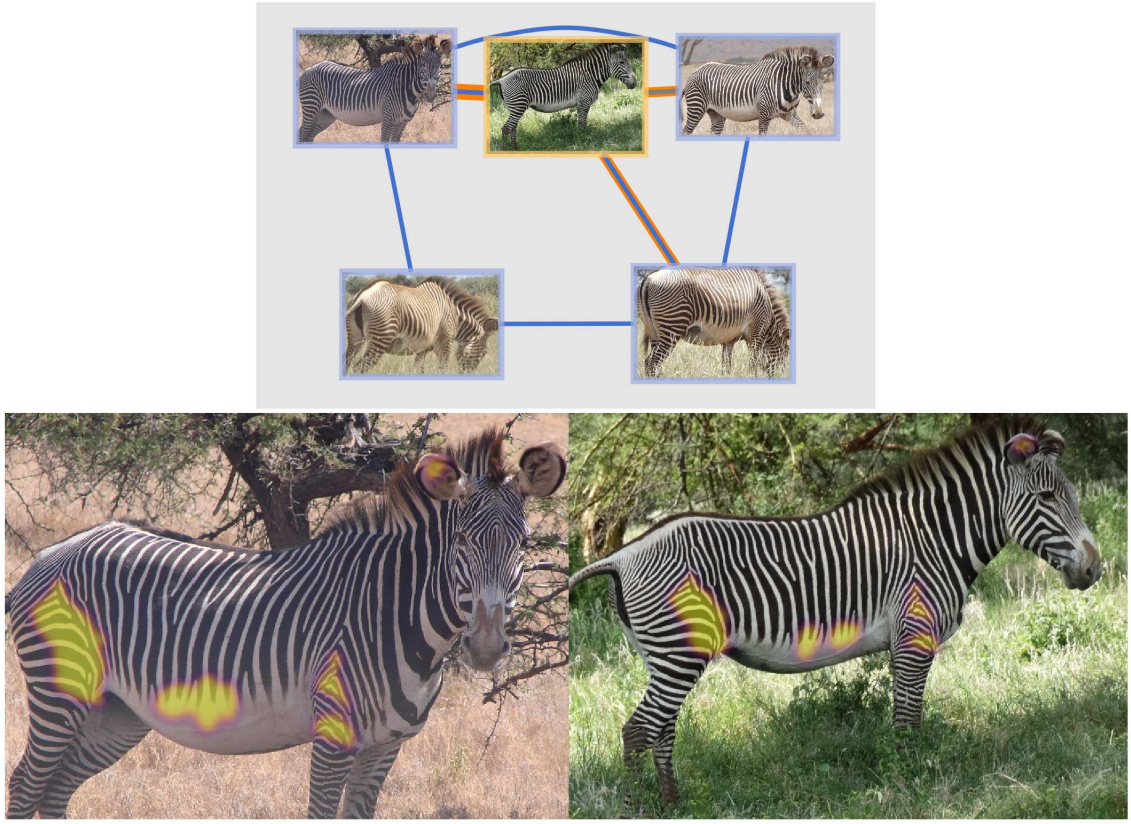
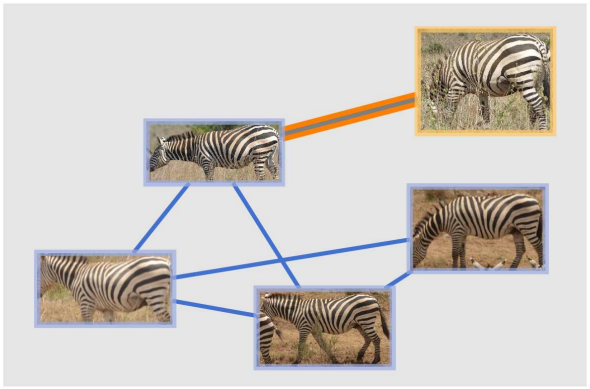
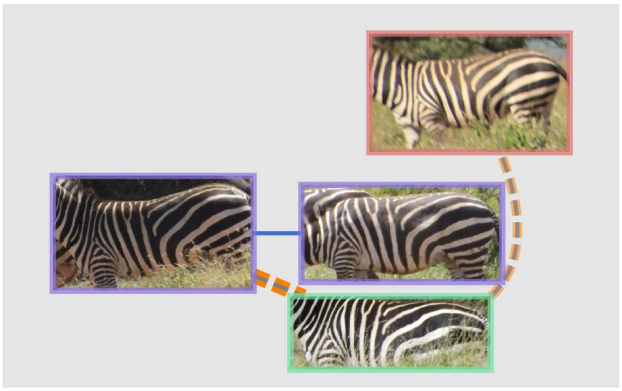


Figure 5.12: **Grévy's split case due to ground truth error** This was incorrectly reported as a split case. The automatic verification algorithm correctly predicted that these annotations should be in the same PCC.



Merge case. Edge was a candidate, but not reviewed
real=Positive, pred=Unreviewed
Positive: 0.29, Negative: 0.71, Incomparable: 0.00
Not Photobomb: 1.00, Photobomb: 0.00

Figure 5.13: **Plains merge case due to low probability** In this case the ranking algorithm generated an edge that could have connected the PCCs. Because the of the low positive probability due to viewpoint and occlusion, the termination criteria stopped the algorithm before this edge was reviewed.



Merge case. Edge did not appear in candidate set
real=Positive, pred=None
Positive: 0.77, Negative: 0.22, Incomparable: 0.00
Not Photobomb: 1.00, Photobomb: 0.00

Figure 5.14: **Plains merge case due to ranking failure** In this case there are three PCCs that should have been merged into one. Occlusion and pose variation prevented the ranking algorithm from generating candidate edges. However, the positive probability of the selected pair is 0.77, which means that the pair would have likely been reviewed if the ranking algorithm had found it.

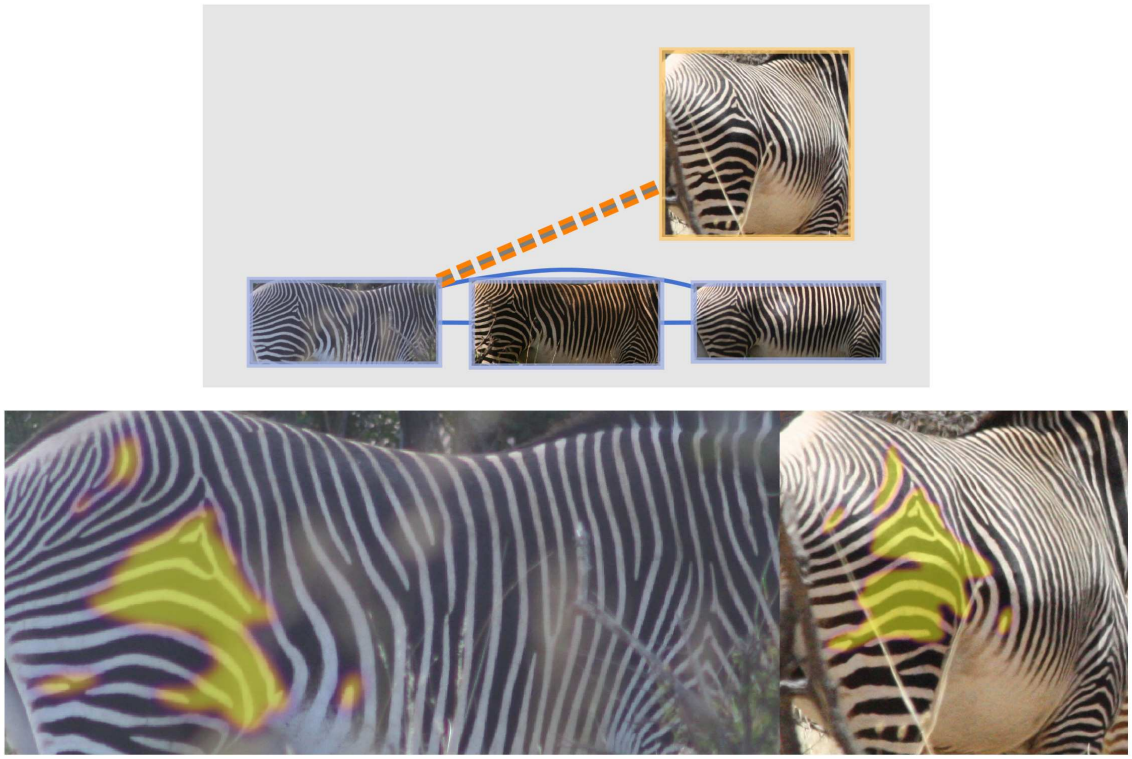


Figure 5.15: **Grévy's merge case due to ranking failure** The ranking algorithm did not generate any candidate edge that could have merged these PCCs due to viewpoint variations. Even, if the ranking algorithm had selected this edge, the positive probability on the edge would not have given it a high enough priority to be reviewed.

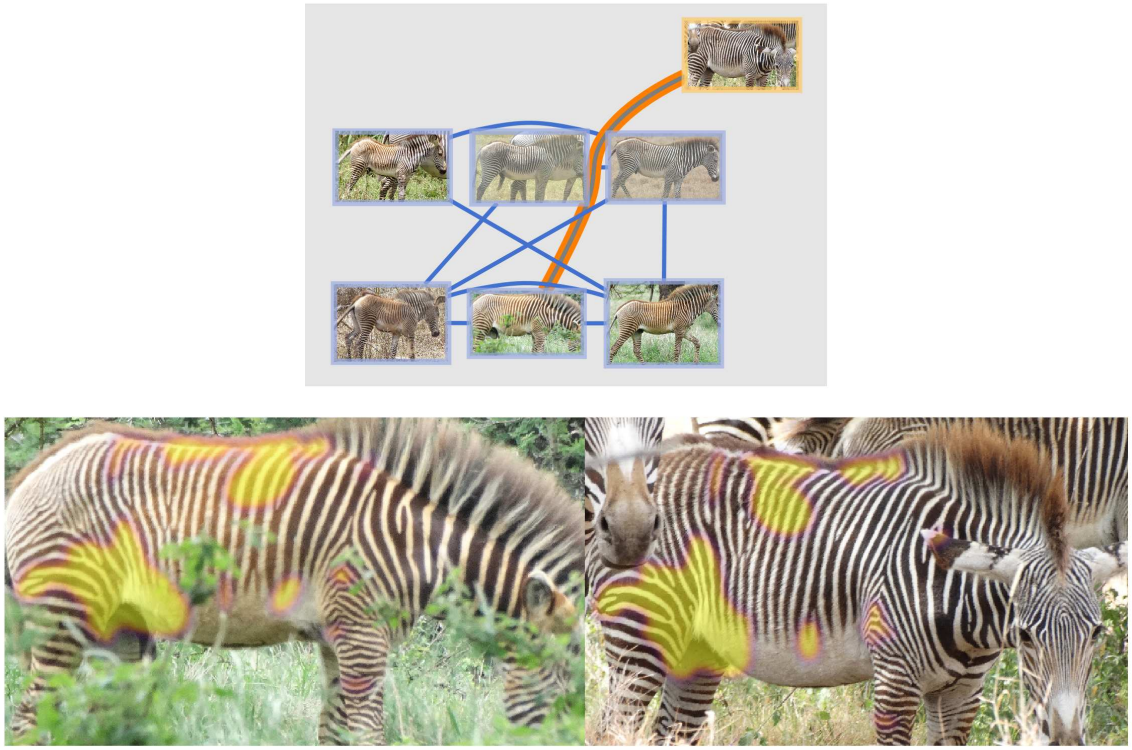


Figure 5.16: **Grévy's merge case due to low probability** The ranking algorithm selected this edge a candidate, but due to occlusion from scenery and other animals the predicted positive probability is low. Therefore, the termination criteria stopped the algorithm before this edge was reviewed.

5.8 SUMMARY OF GRAPH IDENTIFICATION

In this chapter we have described the final component in our approach to animal identification. The graph-based framework takes advantage of the algorithms previously developed in Chapters 3 and 4 and uses them in a principled way to address the identification problem. The ranking algorithm quickly identifies candidate edges that are likely to be label-changing, and the pairwise classifier automatically verifies pairs of annotations, reducing the amount of required manual interaction. While these algorithms can be applied to address identification directly, our experiments have demonstrated that there is a considerable advantage to placing them in the context of the graph identification framework. Specifically, we have simulated an identification event similar to the GZC using our ranking and graph algorithms. We have shown that using the graph framework can reduce the number of manual reviews needed to complete identification by a factor of 4. Furthermore, the graph framework reduces the number of errors and can recover from them when they occur. The only false positives (split cases) produced using the graph framework were the result of errors in the ground truth.

Our graph framework uses the connectivity of positive, negative, and incomparable edges in order to quickly converge on a final identification. Edge connectivity is used to enforce that the graph contains a small exact amount of redundancy, which reduces the number of manual decisions that must be made while providing the means to detect inconsistencies. When inconsistencies are detected, edge-connectivity is again used to quickly find and fix errors in edge labeling. Our framework also includes a probabilistic convergence criterion based on a Poisson process. This criterion stops identification once the ranking and verification algorithm are no longer able to find label-changing edges. This means that the algorithm will stop the identification process before it devolves into a brute force search. The point at which this happens will depend on the power of the ranking and verification algorithms.

This brings us to our last point. The graph framework is general. It is not restricted to animal identification. It could be applied to any instance recognition problem where one annotation corresponds to one individual object. It does not depend on our specific ranking and verification algorithms and could easily incorporate other algorithms. It is even possible to use it without any algorithms. Even though this does result in a brute

force search, the redundancy criterion still reduces the number of reviews, and error recovery still ensures that the graph is always consistent. This case is not just theoretical. To extend to new species and domains it is important to be able to construct a small labeled database from which learned ranking and verification algorithms can be bootstrapped. As more pairwise training data is gathered and maintained using this framework, more sophisticated pairwise classifiers trained using deep learning could be applied, perhaps removing the need for manual interaction and resulting in a fully automatic identification algorithm.

6. CONCLUSION

In this thesis we have addressed the problem of identifying individual animals from images. We have demonstrated that our approach is effective for identifying plains zebras, Grévy's zebras, Masai giraffes, and humpback whales. Our approach consists of three main components: (1) the ranking algorithm from Chapter 3 that uses a bounding box annotation around an animal to search a labeled database of annotations for likely matches, (2) the classification algorithm from Chapter 4 that probabilistically verifies if a pair of annotations is positive, negative, or incomparable, and (3) the graph framework from Chapter 5 that harnesses the previous algorithms in a principled way to dynamically determine the identity of all animals in a dataset. Each of these algorithms was designed to build on the previous one(s), improving the overall accuracy and efficiency of the counting process. In Section 5.7 we demonstrated that this was indeed the case.

By combining these algorithms we have made several meaningful contributions to the problem of animal identification. In Section 1.3 we discussed the Great Zebra Count (GZC), where the ranking algorithm was used in combination with the effort of citizen scientists to provide an estimate of the number of plains zebras and Masai giraffes in Nairobi National Park. In Section 3.5 we investigated several parameters and factors that can impact the performance of the ranking algorithm. We discovered that having multiple photos of each individual significantly improves the accuracy of the ranking algorithm and we designed a novel name scoring mechanism with this in mind. In Section 4.4 we demonstrated that a classification algorithm can be used to improve the separation of positive results from negative and incomparable results in a ranked list. In Section 5.7 we simulated the GZC and demonstrated that our improvements to the ranking algorithm — made by the classification and graph algorithm — enable us to perform identification using less than 25% of the number of manual reviews required by the original event.

6.1 DISCUSSION

The research that resulted in this thesis began in 2010 and was completed in 2017. During that time, many significant developments were made in the fields of computer

vision and machine learning, most notably the explosion of deep learning [264]. While some steps in our approach (*e.g.* the foregroundness weights) do make use of deep convolutional neural networks (DCNNs), most do not. In some sense this is an advantage because the algorithms can be applied to different species without any need for pre-training, but this also means they do not obtain the level of accuracy shown to be achievable by these networks. Yet, the contributions of this thesis are still relevant and complementary to DCNNs. This is trivially true in the case of the ranking and classification algorithms, in part due to the aforementioned reasons. However, the contribution of the graph algorithm is relevant, even in the era of deep learning.

The graph identification algorithm models the abstract constraints of the identification problem and provides a framework that can efficiently harness the power of any ranking or verification algorithm, whether it be deep or shallow. The framework dynamically manages the relationships between annotations. In most cases this means deciding if two annotations are the same (positive) or different (negative), but this also means handling cases like when the annotations are incomparable or when there is some other interesting connection between two annotations like scenery matches and photobombs. As new relationships are added, errors are discovered and corrected, and the identifications are updated.

The framework also provides a means of prioritizing which edges need to be reviewed based on (1) the underlying computer vision algorithms, (2) the edge-augmentation needed to ensure minimum redundancy, and (3) the minimum cut needed to correct an error and split an inconsistent individual. Edge prioritization works in conjunction with a convergence criteria that determines when identification has been completed. A signal is emitted whenever manual interaction is needed, and the algorithm continues after the user returns with a response. The only time a user interacts with the algorithm after it begins is to label an edge as positive, negative, or incomparable. All other decisions are made internally. The algorithm stops once there is a high probability that the vast majority of identifications have been made correctly and consistently. This means that the graph algorithm requires little expertise to use and can be thought of as an "identification wizard" that simply guides the user through a set of simple questions. This design allows the graph algorithm to be run on a web server, where requests for manual interactions can

be sent to remote users and quickly done in a web browser.

6.2 CONTRIBUTIONS

A summary of the contributions made in this thesis are as follows:

(1) The ranking algorithm:

(1.1) We have adapted LNBNN [47] to the problem of individual animal identification. We have performed experiments that demonstrate the effect of several parameters at multiple database sizes. We have shown that tripling the number of annotations in a database can reduce the ranking accuracy at rank 1 by 2%.

(1.2) We have evaluated the effect of various levels of feature invariance in our experiments. We have introduced a heuristic that augments the orientation of query keypoints to account for pose variations. For plains zebras, this can improve the ranking accuracy at rank 1 by 7%.

(1.3) We have accounted for the influence of background features using a learned a foregroundness measure to weight the LNBNN scores of feature correspondences. We have empirically shown that this procedure can increase the ranking accuracy at rank 1 by 5%.

(1.4) We have demonstrated the impact of image redundancy and the importance of collecting more than one annotation in each encounter. Our experiments show that multiple exemplars per name can significantly increase the ranking accuracy at rank 1 by 20%.

(1.5) We have developed a name scoring mechanism to take advantage of information in database names with multiple exemplars. We have shown that this can increase the ranking accuracy at rank 1 by 1% when there are multiple exemplars per name.

(2) The pairwise classification algorithm:

(2.1) We have developed a novel feature vector that represents local and global matching information between two annotations. Our experiments have shown that both the local and global feature dimensions are important for predicting if two annotations match.

(2.2) We have used this feature vector to learn a random forest that can predict the

probability that two annotations are either positive, negative, or incomparable. We have shown that this learned pairwise classifier is a strong predictor of match-state by measuring an MCC of 0.83 for plains zebras and 0.91 for Grévy's zebras.

(2.3) We have compared the learned probabilities to LNBNN scores and demonstrated that re-ranking using the positive probabilities can improve the ranking accuracy at rank 1 by 9% for plains zebras and 2% for Grévy's zebras. Additionally, the probabilities significantly improve the separation of positive and non-positive annotation pairs. For both species, an ROC AUC of less than 0.9 is improved to an AUC greater than 0.97.

(3) The graph identification algorithm:

(3.1) We have demonstrated that combining the graph algorithm with existing ranking and verification algorithms improves the accuracy and efficiency of semi-automatic animal identification. We have designed the framework to be agnostic to the specific ranking and verification algorithms so future DCNN-based algorithms can be swapped in.

(3.2) We have proposed a measure of redundancy based on edge-connectivity used to increase accuracy and reduce the number of reviews needed.

(3.3) We have developed an algorithm for fixing errors whenever inconsistencies in the graph are been discovered.

(3.4) We have developed a probabilistic termination criteria that determines when to stop identification.

6.3 FUTURE WORK

We have shown that our ranking and match-state classification algorithms are both accurate and work well for identifying animals. However, the clearest direction for future research is to replace these algorithms with ones based on DCNNs. To replace the ranking algorithm, we believe that the approach in [52] is a good starting point. We had briefly investigated replacing the pairwise classifier using the techniques in [41], but the results were poor because we did not have as much training data or an alignment procedure. Research into the geometric matching technique described in [242] may help address

both of these issues.

There are also improvements that can be made to the graph algorithm. First it would be useful to parallelize the algorithm so reviews could be distributed across multiple users. This can be obtained by popping multiple edges from the queue at a time, but this could add extraneous redundancy if one edge in the popped set would have been filtered by another. Second, the current prioritization of edges is based completely on the output of the pairwise classifier. In the best case, the ordering would first construct each PCC as a chain, and then only 1 redundant review would be needed. In the worst case, this order would connect one annotation of an individual to all others causing a star shaped PCC. Then to make the PCC 2-positive-redundant, it would take $n - 2$ reviews, where n is the number of annotations in the PCC. Determining the best order in which to review edges depending on the specified level of redundancy is an interesting question, which is perhaps made more challenging if considered in a distributed setting.

BIBLIOGRAPHY

[1] C. Krebs, *Ecological Methodology*. Menlo Park, California, USA: Benjamin Cummings, 1999, vol. 620.

[2] G. C. White and K. P. Burnham, "Program MARK: survival estimation from populations of marked animals," *Bird Study*, vol. 46, pp. S120–S139, Jan 1999.

[3] C. J. Schwarz and A. N. Arnason, "Jolly-seber models in MARK," in *Program MARK - A Gentle Introduction*, 17th ed., E. G. Cooch and G. C. White, Eds., Fort Collins, CO, USA, 2006. [Online]. Available: http://www.phidot.org/software/mark/docs/book/mark_book.zip Accessed on: Jul, 30, 2015.

[4] G. A. F. Seber, *The Estimation of Animal Abundance*, 1st ed. London, UK: Charles Griffin, 1982.

[5] G. M. Jolly, "Explicit estimates from capture-recapture data with both death and immigration-stochastic model," *Biometrika*, vol. 52, no. 1, pp. 225–247, Jun 1965.

[6] G. A. F. Seber, "A note on the multiple-recapture census," *Biometrika*, vol. 52, no. 1, pp. 249–259, Jun 1965.

[7] R. M. Cormack, "Estimates of survival from the sighting of marked animals," *Biometrika*, vol. 51, no. 3, pp. 429–438, Dec 1964.

[8] A. Chao, "Estimating the population size for capture-recapture data with unequal catchability," *Biometrics*, vol. 43, no. 4, pp. 783–791, Dec 1987.

[9] J. D. N. Kenneth. H. Pollock, "Statistical inference for capture-recapture experiments," *Wildlife Monographs*, vol. 107, no. 2, pp. 3–97, Jan 1990.

[10] S. R. Sundaresan, I. R. Fischhoff, J. Dushoff, and D. I. Rubenstein, "Network metrics reveal differences in social organization between two fission-fusion species, grevys zebra and onager," *Oecologia*, vol. 151, no. 1, pp. 140–149, Jan 2007.

[11] D. I. Rubenstein, "Ecology, Social Behavior, and Conservation in Zebras," in *Advances in the Study of Behavior*, M. Naguib, J. Podos, L. W. Simmons, L. Barrett, S. D. Healy, and M. Zuk, Eds. London, UK: Elsevier, 2010, vol. 42, pp. 231–258.

[12] ——, "Behavioral ecology and conservation policy: On balancing science, applications, and advocacy," in *Behavioral Ecology and Conservation Policy*, 1st ed., ser. Behavioral ecology and conservation biology, T. Caro, Ed. New York, NY, USA: Oxford University Press, 1998, pp. 527–556.

[13] Z. Jablons, "Identifying humpback whale flukes by sequence matching of trailing edge curvature," Master's thesis, Department of Computer Science, Rensselaer Polytechnic Institute, Troy, NY, 2016.

[14] D. I. Rubenstein, C. V. Stewart, T. Y. Berger-Wolf, J. Parham, J. Crall, C. Machogu, P. Kahumbu, and N. Maingi, "The great zebra and giraffe count: The power and rewards of citizen science," Kenya Wildlife Service, Nairobi, Kenya, Technical Report, 2015.

[15] J. Levenson, S. Gero, J. Van Oast, and J. Holmberg. (2015) Flukebook: a cloud-based photo-identification analysis tools for marine mammal research. [Online]. Available: http://www.flukebook.org Accessed on: Jan, 21, 2017.

[16] J. Wieczorek, D. Bloom, R. Guralnick, S. Blum, M. Döring, R. Giovanni, T. Robertson, and D. Vieglais, "Darwin core: An evolving community-developed biodiversity data standard," *PLOS ONE*, vol. 7, no. 1, pp. 1–8, Jan 2012. Accessed on: Jan, 15, 2016. [Online]. Available: https://doi.org/10.1371/journal.pone.0029715

[17] J. R. Parham, "Photographic censusing of zebra and giraffe in the nairobi national park," Master's thesis, Department of Computer Science, Rensselaer Polytechnic Institute, Troy, NY, 2015.

[18] J. Parham and C. Stewart, "Detecting plains and grevy's zebras in th real world," in *Winter Appl. of Comput. Vision Workshops*, 2016, pp. 1–9.

[19] K. Mikolajczyk, T. Tuytelaars, C. Schmid, A. Zisserman, J. Matas, F. Schaffalitzky, T. Kadir, and L. V. Gool, "A comparison of affine region detectors," *Int. J. of Comput. Vision*, vol. 65, no. 1, pp. 43–72, Oct 2005.

[20] T. Tuytelaars and K. Mikolajczyk, "Local invariant feature detectors: A survey," *Foundations and Trends in Comput. Graph. and Vision*, vol. 3, no. 3, pp. 177–280, Jan 2007.

[21] M. Perdoch, O. Chum, and J. Matas, "Efficient representation of local geometry for large scale object retrieval," in *Comput. Vision and Pattern Recognition*, 2009, pp. 9–16.

[22] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *Int. J. of Comput. Vision*, vol. 60, no. 2, pp. 91–110, Nov 2004.

[23] K. Mikolajczyk and C. Schmid, "A performance evaluation of local descriptors," *IEEE Trans. on Pattern Anal. and Mach. Intell.*, vol. 27, no. 10, pp. 1615–1630, Oct 2005.

[24] K. Simonyan, A. Vedaldi, and A. Zisserman, "Learning local feature descriptors using convex optimisation," *IEEE Trans. on Pattern Anal. and Mach. Intell.*, vol. 12, pp. 25–70, Jan 2014.

[25] S. Winder, G. Hua, and M. Brown, "Picking the best DAISY," in *Comput. Vision and Pattern Recognition*, 2009, pp. 178–185.

[26] S. Zagoruyko and N. Komodakis, "Learning to compare image patches via convolutional neural networks," in *Comput. Vision and Pattern Recognition*, 2015, pp. 4353–4361.

[27] X. Han, T. Leung, Y. Jia, R. Sukthankar, and A. C. Berg, "MatchNet: Unifying feature and metric learning for patch-based matching," in *Comput. Vision and Pattern Recognition*, 2015, pp. 3279–3286.

[28] C. Silpa-Anan and R. Hartley, "Optimised KD-trees for fast image descriptor matching," in *Comput. Vision and Pattern Recognition*, 2008, pp. 1–8.

[29] M. Muja and D. G. Lowe, "Fast approximate nearest neighbors with automatic algorithm configuration." in *Int. Conf. on Comput. Vision Theory and Appl.*, 2009, pp. 331–340.

[30] J. Sivic and A. Zisserman, "Efficient visual search of videos cast as text retrieval," *IEEE Trans. on Pattern Anal. and Mach. Intell.*, vol. 31, no. 4, pp. 591–606, May 2009.

[31] D. Nister and H. Stewenius, "Scalable recognition with a vocabulary tree," in *Comput. Vision and Pattern Recognition*, vol. 2, 2006, pp. 2161–2168.

[32] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman, "Object retrieval with large vocabularies and fast spatial matching," in *Comput. Vision and Pattern Recognition*, 2007, pp. 1–8.

[33] H. Jégou, M. Douze, and C. Schmid, "Hamming embedding and weak geometric consistency for large scale image search," in *Eur. Conf. on Comput. Vision*, 2008, pp. 304–317.

[34] L. Bo and C. Sminchisescu, "Efficient match kernel between sets of features for visual recognition," in *Advances in Neural Inform. Process. Syst.*, 2009, pp. 135–143.

[35] H. Jégou, F. Perronnin, M. Douze, and C. Schmid, "Aggregating local image descriptors into compact codes," *IEEE Trans. on Pattern Anal. and Mach. Intell.*, vol. 34, no. 9, pp. 1704–1716, Sep 2012.

[36] G. Tolias, Y. Avrithis, and H. Jégou, "To aggregate or not to aggregate: Selective match kernels for image search," in *Int. Conf. on Comput. Vision*, 2013, pp. 1401–1408.

[37] S. Chopra, R. Hadsell, and Y. LeCun, "Learning a similarity metric discriminatively, with application to face verification," in *Comput. Vision and Pattern Recognition*, vol. 1, 2005, pp. 539–546.

[38] G. B. Huang, M. Ramesh, T. Berg, and E. Learned-Miller, "Labeled faces in the wild: A database for studying face recognition in unconstrained environments," University of Massachusetts, Amherst, MA, Technical Report 07-49, 2007.

[39] T. Berg and P. N. Belhumeur, "Tom-vs-pete classifiers and identity-preserving alignment for face verification." in *Brit. Mach. Vision Conf.*, vol. 1, 2012, p. 5.

[40] D. Chen, X. Cao, F. Wen, and J. Sun, "Blessing of dimensionality: High-dimensional feature and its efficient compression for face verification," in *Comput. Vision and Pattern Recognition*, 2013, pp. 3025–3032.

[41] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, "DeepFace: Closing the gap to human-level performance in face verification," in *Comput. Vision and Pattern Recognition*, 2014, pp. 1701–1708.

[42] O. Parkhi, A. Vedaldi, A. Zisserman, and C. Jawahar, "Cats and dogs," in *Comput. Vision and Pattern Recognition*, 2012, pp. 3498–3505.

[43] T. Berg and P. N. Belhumeur, "POOF: Part-based one-vs-one features for fine-grained categorization, face verification, and attribute estimation," in *Comput. Vision and Pattern Recognition*, 2013, pp. 955–962.

[44] E. Gavves, B. Fernando, C. G. M. Snoek, A. W. M. Smeulders, and T. Tuytelaars, "Local alignments for fine-grained categorization," *Int. J. of Comput. Vision*, vol. 111, no. 2, pp. 191–212, Jul 2014.

[45] S. Lazebnik, C. Schmid, and J. Ponce, "Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories," in *Comput. Vision and Pattern Recognition*, vol. 2, 2006, pp. 2169–2178.

[46] J. Zhang, M. Marszaek, S. Lazebnik, and C. Schmid, "Local features and kernels for classification of texture and object categories: A comprehensive study," *Int. J. of Comput. Vision*, vol. 73, no. 2, pp. 213–238, Sep 2006.

[47] S. McCann and D. G. Lowe, "Local naive bayes nearest neighbor for image classification," in *Comput. Vision and Pattern Recognition*, 2012, pp. 3650–3656.

[48] O. Boiman, E. Shechtman, and M. Irani, "In defense of nearest-neighbor based image classification," in *Comput. Vision and Pattern Recognition*, 2008, pp. 1–8.

[49] J. Sánchez, F. Perronnin, T. Mensink, and J. Verbeek, "Compressed fisher vectors for large-scale image classification," INRIA, Rocquencourt, France, Technical Report RR-8209, 2013.

[50] A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Inform. Process. Syst.*, 2012, pp. 1106–1114.

[51] A. S. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson, "CNN features off-the-shelf: an astounding baseline for recognition," in *Comput. Vision and Pattern Recognition*, 2014, pp. 806–813.

[52] R. Arandjelovi, P. Gronat, A. Torii, T. Pajdla, and J. Sivic, "NetVLAD: CNN architecture for weakly supervised place recognition," in *Comput. Vision and Pattern Recognition*, 2016, pp. 5297–5307.

[53] C. Harris and M. Stephens, "A combined corner and edge detector." in *Alvey Vision Conf.*, vol. 15, 1988, p. 50.

[54] K. Mikolajczyk and C. Schmid, "Indexing based on scale invariant interest points," in *Int. Conf. on Comput. Vision*, vol. 1, 2001, pp. 525–531.

[55] S. M. Smith and J. M. Brady, "SUSANa new approach to low level image processing," *Int. J. of Comput. Vision*, vol. 23, no. 1, pp. 45–78, May 1997.

[56] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," in *Eur. Conf. on Comput. Vision*, 2006, pp. 430–443.

[57] P. R. Beaudet, "Rotationally invariant image operators," in *Int. Joint Conf. on Pattern Recognition*, vol. 579, 1978, pp. 579–583.

[58] T. Lindeberg and J. Gaarding, "Shape-adapted smoothing in estimation of 3-d depth cues from affine distortions of local 2-d brightness structure," in *Eur. Conf. on Comput. Vision*, 1994, pp. 389–400.

[59] P. Gaussier and J.-P. Cocquerez, “Neural networks for complex scene recognition: simulation of a visual system with several cortical areas,” in *Int. Joint Conf. on Neural Networks*, vol. 3, 1992, pp. 233–259.

[60] J. Matas, O. Chum, M. Urban, and T. Pajdla, “Robust wide-baseline stereo from maximally stable extremal regions,” *Image and Vision Computing*, vol. 22, no. 10, pp. 761–767, Feb 2004.

[61] S. Buoncompagni, D. Maio, D. Maltoni, and S. Papi, “Saliency-based keypoint selection for fast object detection and matching,” *Pattern Recognition Lett.*, vol. 62, pp. 32–40, Sep 2015.

[62] X. Ren and J. Malik, “Learning a classification model for segmentation,” in *Comput. Vision and Pattern Recognition*, 2003, pp. 10–17.

[63] G. Mori, X. Ren, A. Efros, J. Malik, and others, “Recovering human body configurations: Combining segmentation and recognition,” in *Comput. Vision and Pattern Recognition*, vol. 2, 2004, pp. II–326.

[64] C. Liu, J. Yuen, A. Torralba, J. Sivic, and W. T. Freeman, “SIFT flow: Dense correspondence across different scenes,” in *Eur. Conf. on Comput. Vision*, 2008, pp. 28–42.

[65] J. Revaud, P. Weinzaepfel, Z. Harchaoui, and C. Schmid, “Deep convolutional matching,” *Computing Res. Repository*, vol. abs/1506.07656v1, pp. 1–19, Jun 2015. Accessed on: Jul, 22, 2015. [Online]. Available: https://arxiv.org/abs/1506.07656v1

[66] A. Iscen, G. Tolias, P.-H. Gosselin, and H. Jégou, “A comparison of dense region detectors for image search and fine-grained classification,” *IEEE Trans. on Image Process.*, vol. 24, no. 8, pp. 2369–2381, Aug 2015.

[67] M. Dantone, J. Gall, G. Fanelli, and L. Van Gool, “Real-time facial feature detection using conditional regression forests,” in *Comput. Vision and Pattern Recognition*, 2012, pp. 2578–2585.

[68] T. Lindeberg, *Scale-Space Theory in Computer Vision*. New York, NY, USA: Springer-Verlag, 1993.

[69] ——, "Feature detection with automatic scale selection," *Int. J. of Comput. Vision*, vol. 30, no. 2, pp. 79–116, Nov 1998.

[70] Y. Ke and R. Sukthankar, "PCA-SIFT: a more distinctive representation for local image descriptors," in *Comput. Vision and Pattern Recognition*, vol. 2, 2004, pp. II–506–II–513.

[71] I. Jolliffe, *Principal Component Analysis*, 1st ed. New York, NY, USA: Springer-Verlag, 1986.

[72] T. Lindeberg and J. Gaarding, "Shape-adapted smoothing in estimation of 3-d shape cues from affine distortions of local 2-d brightness structure," *Image and Vision Computing*, vol. 15, no. 6, pp. 415–434, Jun 1997.

[73] A. Baumberg, "Reliable feature matching across widely separated views," in *Comput. Vision and Pattern Recognition*, vol. 1, 2000, pp. 774–781.

[74] D. G. Lowe, "Object recognition from local scale-invariant features," in *Int. Conf. on Comput. Vision*, vol. 2, 1999, pp. 1150–1157.

[75] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "Brief: Binary robust independent elementary features," in *Eur. Conf. on Comput. Vision*, 2010, pp. 778–792.

[76] H. Bay, T. Tuytelaars, and L. V. Gool, "SURF: Speeded up robust features," in *Eur. Conf. on Comput. Vision*, 2006, pp. 404–417.

[77] S. Leutenegger, M. Chli, and R. Siegwart, "BRISK: Binary robust invariant scalable keypoints," in *Int. Conf. on Comput. Vision*, 2011, pp. 2548–2555.

[78] A. Alahi, R. Ortiz, and P. Vandergheynst, "Freak: Fast retina keypoint," in *Comput. Vision and Pattern Recognition*, 2012, pp. 510–517.

[79] H. Jégou and A. Zisserman, "Triangulation embedding and democratic aggregation for image search," in *Comput. Vision and Pattern Recognition*, 2014, pp. 3310–3317.

[80] K. Simonyan, A. Vedaldi, and A. Zisserman, "Descriptor learning using convex optimisation," in *Eur. Conf. on Comput. Vision*, 2012, pp. 243–256.

[81] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE Trans. on Pattern Anal. and Mach. Intell.*, vol. 35, no. 8, pp. 1798–1828, Aug 2013.

[82] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, and others, "Imagenet large scale visual recognition challenge," *Int. J. of Comput. Vision*, pp. 1–42, Apr 2015.

[83] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Comput. Vision and Pattern Recognition*, vol. 1, 2005, pp. 886–893.

[84] A. Oliva and A. Torralba, "Modeling the shape of the scene: A holistic representation of the spatial envelope," *Int. J. of Comput. Vision*, vol. 42, no. 3, pp. 145–175, May 2001.

[85] M. Douze, H. Jégou, H. Sandhawalia, L. Amsaleg, and C. Schmid, "Evaluation of GIST descriptors for web-scale image search," in *Int. Conf. on Image and Video Retrieval*, 2009, pp. 19:1–19:8.

[86] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, "Object detection with discriminatively trained part-based models," *IEEE Trans. on Pattern Anal. and Mach. Intell.*, vol. 32, no. 9, pp. 1627–1645, Sep 2010.

[87] K. van de Sande, T. Gevers, and C. Snoek, "Evaluating color descriptors for object and scene recognition," *IEEE Trans. on Pattern Anal. and Mach. Intell.*, vol. 32, no. 9, pp. 1582–1596, Sep 2010.

[88] T. Hassner, V. Mayzels, and L. Zelnik-Manor, "On sifts and their scales," in *Comput. Vision and Pattern Recognition*, 2012, pp. 1522–1528.

[89] E. Tola, V. Lepetit, and P. Fua, "A fast local descriptor for dense matching," in *Comput. Vision and Pattern Recognition*, 2008, pp. 1–8.

[90] T. Ojala, M. Pietikäinen, and D. Harwood, "A comparative study of texture measures with classification based on featured distributions," *Pattern Recognition*, vol. 29, no. 1, pp. 51–59, Jan 1996.

[91] B. Zhang, Y. Gao, S. Zhao, and J. Liu, "Local derivative pattern versus local binary pattern: Face recognition with high-order local pattern descriptor," *IEEE Trans. on Image Process.*, vol. 19, no. 2, pp. 533–544, Feb 2010.

[92] M. Heikkilä, M. Pietikäinen, and C. Schmid, "Description of interest regions with local binary patterns," *Pattern Recognition*, vol. 42, no. 3, pp. 425–436, Mar 2009.

[93] M. Brown, G. Hua, and S. Winder, "Discriminative learning of local image descriptors," *IEEE Trans. on Pattern Anal. and Mach. Intell.*, vol. 33, no. 1, pp. 43–57, Jan 2011.

[94] G. Shakhnarovich, I. Piotr, and D. Trevor, *Nearest-neighbor Methods in Learning and Vision: Theory and Practice*. Cambridge, Massachusetts, USA: The MIT Press, 2006.

[95] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni, "Locality-sensitive hashing scheme based on p-stable distributions," in *Symp. on Computational Geometry*, 2004, pp. 253–262.

[96] B. Kulis and K. Grauman, "Kernelized locality-sensitive hashing," *IEEE Trans. on Pattern Anal. and Mach. Intell.*, vol. 34, no. 6, pp. 1092–1104, Jun 2012.

[97] Y. Weiss, A. Torralba, and R. Fergus, "Spectral hashing," in *Advances in Neural Inform. Process. Syst.*, 2009, pp. 1753–1760.

[98] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Commun. ACM*, vol. 18, no. 9, pp. 509–517, Sep 1975.

[99] J. H. Friedman, J. L. Bentley, and R. A. Finkel, "An algorithm for finding best matches in logarithmic expected time," *ACM Trans. on Math. Software*, vol. 3, no. 3, pp. 209–226, Sep 1977.

[100] R. F. Sproull, "Refinements to nearest-neighbor searching ink-dimensional trees," *Algorithmica*, vol. 6, no. 1, pp. 579–589, Jun 1991.

[101] J. S. Beis and D. G. Lowe, "Shape indexing using approximate nearest-neighbour search in high-dimensional spaces," in *Comput. Vision and Pattern Recognition*, 1997, pp. 1000–1006.

[102] S. Lloyd, "Least squares quantization in PCM," *IEEE Trans. on Inform. Theory*, vol. 28, no. 2, pp. 129–137, Mar 1982.

[103] M. S. Charikar, "Similarity estimation techniques from rounding algorithms," in *ACM symposium on Theory of computing*, 2002, pp. 380–388.

[104] B. Kulis, P. Jain, and K. Grauman, "Fast similarity search for learned metrics," *IEEE Trans. on Pattern Anal. and Mach. Intell.*, vol. 31, no. 12, pp. 2143–2157, Dec 2009.

[105] R. Tao, E. Gavves, C. Snoek, and A. Smeulders, "Locality in generic instance search from one example," in *Comput. Vision and Pattern Recognition*, 2013, pp. 2091–2098.

[106] J. A. Nelder and R. Mead, "A simplex method for function minimization," *The Comput. J.*, vol. 7, no. 4, pp. 308–313, Jan 1965.

[107] H. Jégou, M. Douze, and C. Schmid, "Product quantization for nearest neighbor search," *IEEE Trans. on Pattern Anal. and Mach. Intell.*, vol. 33, no. 1, pp. 117–128, Jan 2011.

[108] T. Ge, K. He, Q. Ke, and J. Sun, "Optimized product quantization for approximate nearest neighbor search," in *Comput. Vision and Pattern Recognition*, 2013, pp. 2946–2953.

[109] H. Shi, Y. Yang, X. Zhu, S. Liao, Z. Lei, W. Zheng, and S. Z. Li, "Embedding deep metric for person re-identification: A study against large variations," in *Eur. Conf. on Comput. Vision*, 2016, pp. 732–748.

[110] S. Karanam, Y. Li, and R. J. Radke, "Person re-identification with discriminatively trained viewpoint invariant dictionaries," in *Int. Conf. on Comput. Vision*, 2015, pp. 4516–4524.

[111] Z. Wu, Y. Li, and R. J. Radke, "Viewpoint invariant human re-identification in camera networks using pose priors and subject-discriminative features," *IEEE Trans. on Pattern Anal. and Mach. Intell.*, vol. 37, no. 5, pp. 1095–1108, May 2015.

[112] F. Schroff, D. Kalenichenko, and J. Philbin, "FaceNet: A unified embedding for face recognition and clustering," in *Comput. Vision and Pattern Recognition*, 2015, pp. 815–823.

[113] G. Tolias, Y. Avrithis, and H. Jégou, "Image search with selective match kernels: aggregation across single and multiple images," *Int. J. of Comput. Vision*, vol. 116, no. 3, pp. 247–261, Mar 2015.

[114] M. A. Fischler and R. C. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Commun. ACM*, vol. 24, no. 6, pp. 381–395, Jun 1981.

[115] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*. New York, NY, USA: Cambridge University Press, 2003.

[116] O. Chum, J. Matas, and J. Kittler, "Locally optimized RANSAC," in *German Conf. on Pattern Recognition*, 2003, pp. 236–243.

[117] R. Raguram, O. Chum, M. Pollefeys, J. Matas, and J. Frahm, "USAC: A universal framework for random sample consensus," *IEEE Trans. on Pattern Anal. and Mach. Intell.*, vol. 35, no. 8, pp. 2022–2038, Aug 2013.

[118] J. Sivic and A. Zisserman, "Video google: a text retrieval approach to object matching in videos," in *Int. Conf. on Comput. Vision*, 2003, pp. 1470–1477.

[119] O. Chum, A. Mikulík, M. Perdoch, and J. Matas, "Total recall II: Query expansion revisited," in *Comput. Vision and Pattern Recognition*, 2011, pp. 889–896.

[120] R. Arandjelovic and A. Zisserman, "Three things everyone should know to improve object retrieval," in *Comput. Vision and Pattern Recognition*, 2012, pp. 2911–2918.

[121] R. Szeliski, *Computer vision: algorithms and applications*. London, UK: Springer-Verlag, 2010.

[122] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman, "Lost in quantization: Improving particular object retrieval in large scale image databases," in *Comput. Vision and Pattern Recognition*, 2008, pp. 1–8.

[123] A. Mikulík, M. Perdoch, O. Chum, and J. Matas, "Learning a fine vocabulary," in *Eur. Conf. on Comput. Vision*, 2010, pp. 1–14.

[124] J. Philbin, M. Isard, J. Sivic, and A. Zisserman, "Descriptor learning for efficient retrieval," in *Eur. Conf. on Comput. Vision*, 2010, pp. 677–691.

[125] H. Jégou, M. Douze, and C. Schmid, "On the burstiness of visual elements," in *Comput. Vision and Pattern Recognition*, 2009, pp. 1169–1176.

[126] O. Chum, J. Philbin, J. Sivic, M. Isard, and A. Zisserman, "Total recall: Automatic query expansion with a generative feature model for object retrieval," in *Int. Conf. on Comput. Vision*, 2007, pp. 1–8.

[127] G. Tolias and H. Jégou, "Visual query expansion with or without geometry: Refining local descriptors by feature aggregation," *Pattern Recognition*, vol. 47, no. 10, pp. 3466–3476, Oct 2014.

[128] P. Turcot and D. G. Lowe, "Better matching with fewer features: The selection of useful features in large database recognition problems," in *Int. Conf. on Comput. Vision*, 2009, pp. 2109–2116.

[129] O. Chum, J. Philbin, and A. Zisserman, "Near duplicate image detection: min-hash and tf-idf weighting," in *Brit. Mach. Vision Conf.*, 2008, pp. 50.1–50.10.

[130] S. Romberg and R. Lienhart, "Bundle min-hashing for logo recognition," in *ACM Int. Conf. on Multimedia Retrieval*, 2013, p. 113.

[131] J. Wang, S. Kumar, and S.-F. Chang, "Semi-supervised hashing for large-scale search," *IEEE Trans. on Pattern Anal. and Mach. Intell.*, vol. 34, no. 12, pp. 2393–2406, Dec 2012.

[132] Y. Zhang, Z. Jia, and T. Chen, "Image retrieval with geometry-preserving visual phrases," in *Comput. Vision and Pattern Recognition*, 2011, pp. 809–816.

[133] O. Chum, M. Perdoch, and J. Matas, "Geometric min-hashing: Finding a (thick) needle in a haystack," in *Comput. Vision and Pattern Recognition*, 2009, pp. 17–24.

[134] O. Chum and J. Matas, "Large-scale discovery of spatially related images," *IEEE Trans. on Pattern Anal. and Mach. Intell.*, vol. 32, no. 2, pp. 371–377, Feb 2010.

[135] H. Jégou, M. Douze, and C. Schmid, "Improving bag-of-features for large scale image search," *Int. J. of Comput. Vision*, vol. 87, no. 3, pp. 316–336, Aug 2010.

[136] F. Perronnin, Y. Liu, J. Sánchez, and H. Poirier, "Large-scale image retrieval with compressed fisher vectors," in *Comput. Vision and Pattern Recognition*, 2010, pp. 3384–3391.

[137] H. Jégou, M. Douze, C. Schmid, and P. Perez, "Aggregating local descriptors into a compact image representation," in *Comput. Vision and Pattern Recognition*, 2010, pp. 3304–3311.

[138] F. Perronnin and C. Dance, "Fisher kernels on visual vocabularies for image categorization," in *Comput. Vision and Pattern Recognition*, 2007, pp. 1–8.

[139] R. Cinbis, J. Verbeek, and C. Schmid, "Image categorization using fisher kernels of non-iid image models," in *Comput. Vision and Pattern Recognition*, 2012, pp. 2184–2191.

[140] C. Sun and R. Nevatia, "Large-scale web video event classification by use of fisher vectors," in *Winter Appl. of Comput. Vision*, 2013, pp. 15–22.

[141] J. Sánchez, F. Perronnin, T. Mensink, and J. Verbeek, "Image classification with the fisher vector: Theory and practice," *Int. J. of Comput. Vision*, vol. 105, no. 3, pp. 222–245, Jan 2013.

[142] M. Juneja, A. Vedaldi, C. Jawahar, and A. Zisserman, "Blocks that shout: Distinctive parts for scene classification," in *Comput. Vision and Pattern Recognition*, 2013, pp. 923–930.

[143] M. Douze, A. Ramisa, and C. Schmid, "Combining attributes and fisher vectors for efficient image retrieval," in *Comput. Vision and Pattern Recognition*, 2011, pp. 745–752.

[144] B. Ma, Y. Su, and F. Jurie, "Local descriptors encoded by fisher vectors for person re-identification," in *Eur. Conf. on Comput. Vision*, 2012, pp. 413–422.

[145] N. Murray and F. Perronnin, "Generalized max pooling," in *Comput. Vision and Pattern Recognition*, 2014, pp. 2473–2480.

[146] P.-H. Gosselin, N. Murray, H. Jégou, and F. Perronnin, "Revisiting the fisher vector for fine-grained classification," *Pattern Recognition Lett.*, vol. 49, pp. 92–98, Jan 2014.

[147] H. Jégou and O. Chum, "Negative evidences and co-occurences in image retrieval: The benefit of PCA and whitening," in *Eur. Conf. on Comput. Vision*, 2012, pp. 774–787.

[148] J. Delhumeau, P.-H. Gosselin, H. Jégou, and P. Pérez, "Revisiting the VLAD image representation," in *ACM Int. Conf. on Multimedia*, 2013, pp. 653–656.

[149] R. Arandjelovic and A. Zisserman, "All about VLAD," in *Comput. Vision and Pattern Recognition*, 2013, pp. 1578–1585.

[150] A. Torii, J. Sivic, and T. Pajdla, "Visual localization by linear combination of image descriptors," in *Int. Conf. on Comput. Vision*, 2011, pp. 102–109.

[151] P. Viola and M. J. Jones, "Robust real-time face detection," *Int. J. of Comput. Vision*, vol. 57, no. 2, pp. 137–154, May 2004.

[152] F. Radenovi, G. Tolias, and O. Chum, "CNN image retrieval learns from BoW: Unsupervised fine-tuning with hard examples," in *Eur. Conf. on Comput. Vision*, 2016, pp. 3–20.

[153] W. Zhao, R. Chellappa, P. J. Phillips, and A. Rosenfeld, "Face recognition: A literature survey," *ACM Computing Surveys*, vol. 35, no. 4, pp. 399–458, Dec 2003.

[154] C. Liu and H. Wechsler, "Gabor feature based classification using the enhanced fisher linear discriminant model for face recognition," *IEEE Trans. on Image Process.*, vol. 11, no. 4, pp. 467–476, Apr 2002.

[155] B. Zhang, S. Shan, X. Chen, and W. Gao, "Histogram of gabor phase patterns (HGPP): A novel object representation approach for face recognition," *IEEE Trans. on Image Process.*, vol. 16, no. 1, pp. 57–68, Jan 2007.

[156] L. Shen and L. Bai, "A review on gabor wavelets for face recognition," *Pattern Anal. and Appl.*, vol. 9, no. 2, pp. 273–292, Aug 2006.

[157] T. Ahonen, A. Hadid, and M. Pietikainen, "Face description with local binary patterns: Application to face recognition," *IEEE Trans. on Pattern Anal. and Mach. Intell.*, vol. 28, no. 12, pp. 2037–2041, Dec 2006.

[158] P. N. Belhumeur, D. W. Jacobs, D. J. Kriegman, and N. Kumar, "Localizing parts of faces using a consensus of exemplars," in *Comput. Vision and Pattern Recognition*, 2011, pp. 545–552.

[159] M. Turk and A. Pentland, "Eigenfaces for recognition," *J. of Cognitive Neuroscience*, vol. 3, no. 1, pp. 71–86, Jan 1991.

[160] P. Belhumeur, J. Hespanha, and D. Kriegman, "Eigenfaces vs. fisherfaces: recognition using class specific linear projection," *IEEE Trans. on Pattern Anal. and Mach. Intell.*, vol. 19, no. 7, pp. 711–720, Jul 1997.

[161] S. Lawrence, C. Giles, A. C. Tsoi, and A. Back, "Face recognition: a convolutional neural-network approach," *IEEE Trans. on Neural Networks*, vol. 8, no. 1, pp. 98–113, Jan 1997.

[162] J. Wright, A. Yang, A. Ganesh, S. Sastry, and Y. Ma, "Robust face recognition via sparse representation," *IEEE Trans. on Pattern Anal. and Mach. Intell.*, vol. 31, no. 2, pp. 210–227, Feb 2009.

[163] Z. Jiang, Z. Lin, and L. Davis, "Label consistent k-SVD: Learning a discriminative dictionary for recognition," *IEEE Trans. on Pattern Anal. and Mach. Intell.*, vol. 35, no. 11, pp. 2651–2664, Nov 2013.

[164] I. Craw and P. Cameron, "Face recognition by computer," in *Brit. Mach. Vision Conf.*, 1992, pp. 498–507.

[165] C. Liu and H. Wechsler, "Robust coding schemes for indexing and retrieval from large face databases," *IEEE Trans. on Image Process.*, vol. 9, no. 1, pp. 132–137, Jan 2000.

[166] J. Lu, K. Plataniotis, and A. Venetsanopoulos, "Face recognition using LDA-based algorithms," *IEEE Trans. on Neural Networks*, vol. 14, no. 1, pp. 195–200, Jan 2003.

[167] P. J. Phillips, "Support vector machines applied to face recognition," in *Advances in Neural Inform. Process. Syst.*, vol. 285, 1999, pp. 803–809.

[168] L. W. N. Levy, "The SVM-minus similarity score for video face recognition," in *Comput. Vision and Pattern Recognition*, 2013, pp. 529–534.

[169] M. Aharon, M. Elad, and A. Bruckstein, "K-SVD: An algorithm for designing overcomplete dictionaries for sparse representation," *IEEE Trans. on Signal Process.*, vol. 54, no. 11, pp. 4311–4322, Nov 2006.

[170] D. Zhang, M. Yang, and X. Feng, "Sparse representation or collaborative representation: Which helps face recognition?" in *Int. Conf. on Comput. Vision*, 2011, pp. 471–478.

[171] M. Hirzer, P. Roth, M. Köstinger, and H. Bischof, "Relaxed pairwise learned metric for person re-identification," in *Eur. Conf. on Comput. Vision*, 2012, pp. 780–793.

[172] S. Cao and N. Snavely, "Learning to match images in large-scale collections," in *Eur. Conf. on Comput. Vision*, 2012, pp. 259–270.

[173] O. Chum and J. Matas, "Fast computation of min-hash signatures for image collections," in *Comput. Vision and Pattern Recognition*, 2012, pp. 3077–3084.

[174] Y. Gong, L. Wang, R. Guo, and S. Lazebnik, "Multi-scale orderless pooling of deep convolutional activation features," in *Eur. Conf. on Comput. Vision*, vol. 8695, 2014, pp. 392–407.

[175] M. Everingham, L. V. Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes challenge," *Int. J. of Comput. Vision*, vol. 88, no. 2, pp. 303–338, Jun 2010.

[176] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes challenge: A retrospective," *Int. J. of Comput. Vision*, vol. 111, no. 1, pp. 98–136, Jan 2015.

[177] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet - a large-scale hierarchical image database," in *Comput. Vision and Pattern Recognition*, 2009, pp. 248–255.

[178] L. Fei-Fei, R. Fergus, and P. Perona, "One-shot learning of object categories," *IEEE Trans. on Pattern Anal. and Mach. Intell.*, vol. 28, no. 4, pp. 594–611, Apr 2006.

[179] G. Griffin, A. Holub, and P. Perona. (2007) Caltech-256 object category dataset. [Online]. Available: http://resolver.caltech.edu/CaltechAUTHORS:CNS-TR-2007-001 Accessed on: Apr, 7, 2015.

[180] G. Csurka, C. Dance, L. Fan, J. Willamowski, and C. Bray, "Visual categorization with bags of keypoints," in *Eur. Conf. on Comput. Vision*, vol. 1, 2004, pp. 1–2.

[181] J. Yang, K. Yu, Y. Gong, and T. Huang, "Linear spatial pyramid matching using sparse coding for image classification," in *Comput. Vision and Pattern Recognition*, 2009, pp. 1794–1801.

[182] M. Varma and D. Ray, "Learning the discriminative power-invariance trade-off," in *Int. Conf. on Comput. Vision*, 2007, pp. 1–8.

[183] A. Vedaldi and A. Zisserman, "Efficient additive kernels via explicit feature maps," *IEEE Trans. on Pattern Anal. and Mach. Intell.*, vol. 34, no. 3, pp. 480–492, Jan 2012.

[184] K. Grauman and T. Darrell, "The pyramid match kernel: discriminative classification with sets of image features," in *Int. Conf. on Comput. Vision*, vol. 2, 2005, pp. 1458–1465.

[185] Y.-L. Boureau, J. Ponce, and Y. LeCun, "A theoretical analysis of feature pooling in visual recognition," in *Int. Conf. on Mach. Learning*, 2010, pp. 111–118.

[186] Y.-L. Boureau, F. Bach, Y. LeCun, and J. Ponce, "Learning mid-level features for recognition," in *Comput. Vision and Pattern Recognition*, 2010, pp. 2559–2566.

[187] L. Liu, L. Wang, and X. Liu, "In defense of soft-assignment coding," in *Int. Conf. on Comput. Vision*, 2011, pp. 2486–2493.

[188] J. Wang, J. Yang, K. Yu, F. Lv, T. Huang, and Y. Gong, "Locality-constrained linear coding for image classification," in *Comput. Vision and Pattern Recognition*, 2010, pp. 3360–3367.

[189] A. Vedaldi, V. Gulshan, M. Varma, and A. Zisserman, "Multiple kernels for object detection," in *Int. Conf. on Comput. Vision*, 2009, pp. 606–613.

[190] S. Maji, A. Berg, and J. Malik, "Classification using intersection kernel support vector machines is efficient," in *Comput. Vision and Pattern Recognition*, 2008, pp. 1–8.

[191] F. Perronnin, J. Sánchez, and Y. Liu, "Large-scale image categorization with explicit data embedding," in *Comput. Vision and Pattern Recognition*, 2010, pp. 2297–2304.

[192] F. Jurie and B. Triggs, "Creating efficient codebooks for visual recognition," in *Int. Conf. on Comput. Vision*, vol. 1, 2005, pp. 604–610.

[193] J. Yang, K. Yu, and T. Huang, "Supervised translation-invariant sparse coding," in *Comput. Vision and Pattern Recognition*, 2010, pp. 3517–3524.

[194] ——, "Efficient highly over-complete sparse coding using a mixture model," in *Eur. Conf. on Comput. Vision*, 2010, pp. 113–126.

[195] F. Perronnin, J. Sánchez, and T. Mensink, "Improving the fisher kernel for large-scale image classification," in *Eur. Conf. on Comput. Vision*, 2010, pp. 143–156.

[196] M. Marszaek, C. Schmid, H. Harzallah, and J. V. d. Weijer, "Learning object representations for visual object class recognition," in *Int. Conf. on Comput. Vision*, 2007, pp. 1–21.

[197] M. Varma and A. Zisserman, "Unifying statistical texture classification frameworks," *Image and Vision Computing*, vol. 22, no. 14, pp. 1175–1183, Dec 2004.

[198] T. Berg and P. N. Belhumeur, "How do you tell a blackbird from a crow?" in *Int. Conf. on Comput. Vision*, 2013, pp. 9–16.

[199] C. Rother, V. Kolmogorov, and A. Blake, "Grabcut: Interactive foreground extraction using iterated graph cuts," *ACM Trans. on Graph.*, vol. 23, pp. 309–314, Aug 2004.

[200] Catherine Wah, Grant Van Horn, Steve Branson, Subhransu Maji, Pietro Perona, and Serge Belongie, "Similarity comparisons for interactive fine-grained categorization," in *Comput. Vision and Pattern Recognition*, 2014, pp. 859–866.

[201] S. Branson, G. Van Horn, S. Belongie, and P. Perona, "Bird species categorization using pose normalized deep convolutional nets," *Computing Res. Repository*, vol. abs/1406.2952, pp. 1–14, Jun 2014. Accessed on: Apr, 28, 2015. [Online]. Available: https://arxiv.org/abs/1406.2952

[202] Y. Zhang, X.-s. Wei, J. Wu, J. Cai, J. Lu, V.-A. Nguyen, and M. N. Do, "Weakly supervised fine-grained image categorization," *Computing Res. Repository*, vol.

abs/1504.04943, pp. 1–9, Apr 2015. Accessed on: Apr, 28, 2015. [Online]. Available: https://arxiv.org/abs/1504.04943

[203] T. Xiao, Y. Xu, K. Yang, J. Zhang, Y. Peng, and Z. Zhang, "The application of two-level attention models in deep convolutional neural network for fine-grained image classification," in *Comput. Vision and Pattern Recognition*, 2015, pp. 842–85.

[204] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov 1998.

[205] K. Fukushima, "Neocognitron: A hierarchical neural network capable of visual pattern recognition," *IEEE Trans. on Neural Networks*, vol. 1, no. 2, pp. 119–130, Sep 1988.

[206] V. N. Vapnik, *Statistical Learning Theory*. Hoboken, NJ, USA: Wiley-Interscience, 1998.

[207] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Int. Conf. on Learning Representations*, 2015, pp. 1–14.

[208] K. Chatfield, K. Simonyan, and A. Zisserman, "Efficient on-the-fly category retrieval using ConvNets and GPUs," in *Asian Conf. on Comput. Vision*, 2014, pp. 129–145.

[209] K. Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman, "Return of the devil in the details: Delving deep into convolutional nets," *Computing Res. Repository*, vol. abs/1405.3531, pp. 1–11, May 2014. Accessed on: Jul, 20, 2015. [Online]. Available: https://arxiv.org/abs/1405.3531

[210] M. Oquab, L. Bottou, I. Laptev, and J. Sivic, "Learning and transferring mid-level image representations using convolutional neural networks," in *Comput. Vision and Pattern Recognition*, 2014, pp. 1717–1724.

[211] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Comput. Vision and Pattern Recognition*, 2015, pp. 1–9.

[212] J. L. Long, N. Zhang, and T. Darrell, "Do convnets learn correspondence?" in *Advances in Neural Inform. Process. Syst.*, 2014, pp. 1601–1609.

[213] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," in *Eur. Conf. on Comput. Vision*, 2014, pp. 346–361.

[214] T. Dean, M. A. Ruzon, M. Segal, J. Shlens, S. Vijayanarasimhan, and J. Yagnik, "Fast, accurate detection of 100,000 object classes on a single machine," in *Comput. Vision and Pattern Recognition*, 2013, pp. 1814–1821.

[215] A. S. Razavian, J. Sullivan, A. Maki, and S. Carlsson, "A baseline for visual instance retrieval with deep convolutional networks," *Computing Res. Repository*, vol. abs/1412.6574v3, pp. 1–8, Apr 2015. Accessed on: May, 11, 2015. [Online]. Available: https://arxiv.org/abs/1412.6574v3

[216] S. Liu and L. Hongtao, "Learning better encoding for approximate nearest neighbor search with dictionary annealing," *Computing Res. Repository*, vol. abs/1507.01442, pp. 1–10, Jul 2015. Accessed on: Jul, 10, 2015. [Online]. Available: https://arxiv.org/abs/1507.01442

[217] D. Held, S. Thrun, and S. Savarese, "Deep learning for single-view instance recognition," *Computing Res. Repository*, vol. abs/1507.08286, pp. 1–16, Jul 2015. Accessed on: Aug, 3, 2015. [Online]. Available: https://arxiv.org/abs/1507.08286

[218] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell, "DeCAF: A deep convolutional activation feature for generic visual recognition," in *Int. Conf. on Mach. Learning*, 2014, pp. 647–655.

[219] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for

accurate object detection and semantic segmentation," in *Comput. Vision and Pattern Recognition*, 2014, pp. 580–587.

[220] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, "OverFeat: Integrated recognition, localization and detection using convolutional networks," *Computing Res. Repository*, vol. abs/1312.6229, pp. 1–16, Dec 2013. Accessed on: Jul, 20, 2015. [Online]. Available: https://arxiv.org/abs/1312.6229

[221] Li Wan, David Eigen, and Rob Fergus, "End-to-end integration of a convolutional network, deformable parts model and non-maximum suppression," in *Comput. Vision and Pattern Recognition*, 2015, pp. 851–859.

[222] G. Huang, H. Lee, and E. Learned-Miller, "Learning hierarchical representations for face verification with convolutional deep belief networks," in *Comput. Vision and Pattern Recognition*, 2012, pp. 2518–2525.

[223] Y. Sun, X. Wang, and X. Tang, "Deep convolutional network cascade for facial point detection," in *Comput. Vision and Pattern Recognition*, 2013, pp. 3476–3483.

[224] C. Osendorfer, J. Bayer, S. Urban, and P. Van Der Smagt, "Convolutional neural networks learn compact local image descriptors," in *Int. Conf. on Neural Inform. Process.*, 2013, pp. 624–630.

[225] J. Y.-H. Ng, F. Yang, and L. S. Davis, "Exploiting local features from deep networks for image retrieval," in *Comput. Vision and Pattern Recognition*, 2015, pp. 53–61.

[226] D. Ciresan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," in *Comput. Vision and Pattern Recognition*, 2012, pp. 3642–3649.

[227] D. C. Cirean, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber, "High-performance neural networks for visual object classification," Dalle Molle Institute for Artificial Intelligence, Galleria 2, 6928 Manno, Switzerland, Technical Report IDSIA-01-11, 2011.

[228] P. Y. Simard, D. Steinkraus, and J. C. Platt, "Best practices for convolutional neural networks applied to visual document analysis," in *Int. Conf. on Document Anal. and Recognition*, vol. 3, 2003, pp. 958–962.

[229] G. E. Dahl, T. N. Sainath, and G. E. Hinton, "Improving deep neural networks for LVCSR using rectified linear units and dropout," in *Int. Conf. on Acoust., Speech and Signal Process.*, 2013, pp. 8609–8613.

[230] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. of Mach. Learning Res.*, vol. 15, no. 1, pp. 1929–1958, Jun 2014.

[231] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *Computing Res. Repository*, vol. abs/1207.0580, pp. 1–18, Jul 2012. Accessed on: Aug, 3, 2015. [Online]. Available: https://arxiv.org/abs/1207.0580

[232] N. Vinod and G. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Int. Conf. on Mach. Learning*, 2010, pp. 807–814.

[233] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *Int. Conf. on Mach. Learning*, vol. 30, 2013, pp. 1–6.

[234] D. Rumelhart, G. Hinton, and R. Williams, "Learning representations by back-propagating errors," *Nature*, pp. 323–533, Oct 1986.

[235] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in *Int. Conf. on Mach. Learning*, 2013, pp. 1139–1147.

[236] T. Serre, L. Wolf, S. Bileschi, M. Riesenhuber, and T. Poggio, "Robust object recognition with cortex-like mechanisms," *IEEE Trans. on Pattern Anal. and Mach. Intell.*, vol. 29, no. 3, pp. 411–426, Mar 2007.

[237] I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio, "Maxout networks," in *Int. Conf. on Mach. Learning*, vol. 28, 2013, pp. 1319–1327.

[238] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *Eur. Conf. on Comput. Vision*, 2014, pp. 818–833.

[239] D. Gabor, "Theory of communication. part 1: The analysis of information," *J. of the Institution of Elect. Engineers*, vol. 93, no. 26, pp. 429–441, Nov 1946.

[240] A. Mahendran and A. Vedaldi, "Understanding deep image representations by inverting them," in *Comput. Vision and Pattern Recognition*, 2015, pp. 5188–5196.

[241] Y. LeCun and F. Huang, "Loss functions for discriminative training of energy-based models," in *Int. Conf. on Artificial Intell. and Statist.*, 2005, pp. 1–8.

[242] I. Rocco, R. Arandjelovi, and J. Sivic, "Convolutional neural network architecture for geometric matching," in *Comput. Vision and Pattern Recognition*, 2017, pp. 1–15.

[243] C. Lanczos, *Applied Analysis*. North Chelmsford, MA, USA: Courier Corporation, 1988.

[244] S. M. Pizer, E. P. Amburn, J. D. Austin, R. Cromartie, A. Geselowitz, T. Greer, B. ter Haar Romeny, J. B. Zimmerman, and K. Zuiderveld, "Adaptive histogram equalization and its variations," *Graphical Models and Image Process.*, vol. 39, no. 3, pp. 355–368, Sep 1987.

[245] R. Arandjelovic and A. Zisserman, "Dislocation: Scalable descriptor distinctiveness for location recognition," in *Asian Conf. on Comput. Vision*, 2014, pp. 188–204.

[246] O. Chum and J. Matas, "Homography estimation from correspondences of local elliptical features," in *Int. Conf. on Pattern Recognition*, 2012, pp. 3236–3239.

[247] A. Oddone, "A mobile application for the image based ecological information system," Master's thesis, Department of Computer Science, University of Illinois at Chicago, Chicago, IL, 2016.

[248] G. Michael and D. S. Johnson, *A Guide to the Theory of NP-Completeness*. New York, NY, USA: WH Freemann, 1979.

[249] B. DeCann and A. Ross, "Relating roc and cmc curves via the biometric menagerie," in *Biometrics: Theory, Appl. and Syst.*, 2013, pp. 1–8.

[250] L. Breiman, "Random forests," *Mach. Learning*, vol. 45, no. 1, pp. 5–32, Oct 2001.

[251] Y. Ding and J. S. Simonoff, "An investigation of missing data methods for classification trees applied to binary response data," *J. of Mach. Learning Res.*, vol. 11, pp. 131–170, Jan 2010.

[252] D. Qin, S. Gammeter, L. Bossard, T. Quack, and L. Van Gool, "Hello neighbor: Accurate object retrieval with k-reciprocal nearest neighbors," in *Comput. Vision and Pattern Recognition*, 2011, pp. 777–784.

[253] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, and others, "Scikit-learn: Machine learning in python," *J. of Mach. Learning Res.*, vol. 12, pp. 2825–2830, Oct 2011.

[254] D. M. Powers, "Evaluation: from precision, recall and f-measure to ROC, informedness, markedness and correlation," *J. of Mach. Learning Technologies*, vol. 2, no. 1, pp. 37–63, Feb 2011.

[255] G. Louppe, "Understanding random forests: From theory to practice," Ph.D. dissertation, Department of Electrical Engineering and Computer Science, University of Liège, Liège, Belgium, 2014.

[256] B. Fulkerson, A. Vedaldi, and S. Soatto, "Class segmentation and object localization with superpixel neighborhoods," in *Int. Conf. on Comput. Vision*, 2009, pp. 670–677.

[257] Jacob Holm, Kristian de Lichtenberg, and Mikkel Thorup, "Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity," *J. of the ACM*, vol. 48, no. 4, pp. 723–760, Jul 2001.

[258] K. P. Eswaran and R. E. Tarjan, "Augmentation problems," *SIAM J. on Computing*, vol. 5, no. 4, pp. 653–665, Oct 1976.

[259] T. Wang, Y. Zhang, F. Y. L. Chin, H.-F. Ting, Y. H. Tsin, and S.-H. Poon, "A simple algorithm for finding all k-edge-connected components," *PLOS ONE*, vol. 10, no. 9, pp. 1–10, Sep 2015. Accessed on: Mar, 2, 2017. [Online]. Available: https://doi.org/10.1371/journal.pone.0136264

[260] A.-H. Esfahanian. (2017) Connectivity algorithms. [Online]. Available: http://www.cse.msu.edu/~cse835/Papers/Graph_connectivity_revised.pdf Accessed on: Jan, 21, 2017.

[261] S. Khuller and R. Thurimella, "Approximation algorithms for graph augmentation," *J. of Algorithms*, vol. 14, no. 2, pp. 214–225, Mar 1993.

[262] S. Branson, C. Wah, F. Schroff, B. Babenko, P. Welinder, P. Perona, and S. Belongie, "Visual recognition with humans in the loop," in *Eur. Conf. on Comput. Vision*, 2010, pp. 438–451.

[263] V. V. Vazirani, *Approximation Algorithms*. Berlin, DE: Springer-Verlag, 2013.

[264] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015.

[265] Eric Jones, Travis Travis Oliphant, Pearu Peterson, and others. (2001) SciPy: Open source scientific tools for python. [Online]. Available: http://www.scipy.org/ Accessed on: Jan, 21, 2017.

APPENDIX A
OCCURRENCES

In the identification workflow we separate groups of images into *occurrences*. The Darwin Core defines an occurrence as a collection of evidence that shows an organism exists within specific location and span of time [16]. For our purposes this amounts to a cluster of images localized in space and time. We outline an occurrence grouping algorithm performs agglomerative clustering on the GPS coordinates and time specified in the image metadata. We first describe the space-time measure of distance between images and then describe the clustering algorithm.

A.1 SPACE-TIME IMAGE DISTANCE

To compute occurrences we define a space-time feature $\mathbf{g}_i$ for each image i, and a pairwise distance, $\Delta(\mathbf{g}_i, \mathbf{g}_j)$, between these features. This feature will a two-dimensional feature tuple, $\mathbf{g}_i = (t_i, \mathbf{p}_i)$, where the first component is the POSIX timestamp t_i, and the second component is a GPS coordinate $\mathbf{p}_i = [\varphi_i, \lambda_i]^T$, where the angles of latitude and longitude are measured in radians. To compute this distance between two images $\mathbf{g}_i$ and $\mathbf{g}_j$ we first compute the distance in each component of the feature tuple. The difference in time is the absolute value of the timedelta, $\Delta_t(\mathbf{g}_i, \mathbf{g}_j) = |t_i - t_j|$, which is in seconds.

Next, the distance in space is computed by approximating the Earth as a sphere. In general, the distance between two points on a sphere with radius r is a function of inverse haversines, and is expressed as:

$$d(\mathbf{p}_i, \mathbf{p}_j, r) = 2r \operatorname{asin}\left(\sqrt{\operatorname{hav}(\varphi_i - \varphi_j) + \operatorname{hav}(\lambda_i - \lambda_j) + \cos(\varphi_i)\cos(\varphi_j)}\right) \quad \text{(A.1)}$$

In the previous equation, $\operatorname{hav}(\theta) = \sin^2\left(\frac{\theta}{2}\right)$ is the half vertical sine function. Thus, we arrive at the spatial distance between two images by estimating the radius of the earth to be $r = 6367$ kilometers.

$$\Delta_s(\mathbf{g}_i, \mathbf{g}_j) = d(\mathbf{p}_i, \mathbf{p}_j, 6367). \quad \text{(A.2)}$$

This results in distance in seconds and a distance in kilometers, which are in incompatible

units. To combine these distances we convert kilometers to seconds by heuristically estimating the walking speed, S, of an animal (for zebras we use $S = 2 \times 10^{-3}$ kilometers per second). This allows us to cancel kilometers from the expression and express GPS distance as a unit of time: $\frac{\Delta_s(\mathbf{g}_i, \mathbf{g}_j)}{S}$. This distance can be interpreted as the total amount of time it would take an animal to move between two points. The total distance between two images is the sum of these components.

$$\Delta(\mathbf{g}_i, \mathbf{g}_j) = \Delta_t(\mathbf{g}_i, \mathbf{g}_j) + \frac{\Delta_s(\mathbf{p}_i, \mathbf{p}_j)}{S} \tag{A.3}$$

Notice that if there is no difference in GPS location, then this measure becomes to a distance in time.

A.2 CLUSTERING PROCEDURE

Having defined pairwise a distance between two images, we use the agglomerative clustering procedures implemented in SciPy [265] to group the images. There are two inputs to the agglomerative clustering algorithm: (1) The matrix of pairwise distance between images, and (2) the minimum distance threshold between two images. The matrix of distances is computed using Equation (A.3), and we set the distance threshold to 30 minutes. Any pair of images that is within this threshold connected via a linkage matrix. Connected components in this matrix form the final clusters that we use as occurrences. To improve the efficiency of the algorithm, we separate it into disjoint parts by sorting the images by timestamp and splitting the data wherever the difference in consecutive timestamps exceeds the threshold. Images that are missing either timestamp or GPS location are grouped by what data they do have and clustered separately.