IDENTIFYING HUMPBACK WHALE FLUKES BY SEQUENCE MATCHING OF TRAILING EDGE CURVATURE

By

Zachary Jablons

A Thesis Submitted to the Graduate

Faculty of Rensselaer Polytechnic Institute

in Partial Fulfillment of the

Requirements for the Degree of

MASTER OF SCIENCE

Major Subject: COMPUTER SCIENCE

Examining Committee:

_____

Dr. Charles Stewart, Thesis Adviser

_____

Dr. Barbara Cutler, Member

_____

Dr. Bülent Yener, Member

Rensselaer Polytechnic Institute
Troy, New York

April 2016
(For Graduation May 2016)

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGMENTS

This project could not have been completed so quickly and thoroughly without the invaluable help from the rest of the IBEIS team. I would like to thank Jon Crall for helping to write the experimentation framework which has saved a lot of time and effort. I would also like to thank Jason Parham and Hendrik Weidemann for the great discussions and advice, as well as the handy LaTeXtemplate. Additionally, without the development and annotation efforts of Andrew Batbouta, a lot of models could not have been so successfully trained. Importantly, I would like to thank the Wildbook team (Jason Holmberg, Jon van Oast) for providing the dataset and corresponding annotations with which a lot of models were trained and experiments run. Of course, this work could never have been undertaken without the help, advice, and direction of my advisor, Charles Stewart.

ABSTRACT

Photographic identification of humpback whale (Megaptera novaeangliae) flukes (i.e. their tail) is an important task in marine ecology, and is used in tracking migration patterns and estimating populations [7] [9]. In this thesis, we lay out a method that automates the photo-identification of humpback flukes, using the "trailing edge" of the fluke. The method uses convolutional networks to identify keypoints on the fluke and possible trailing edge locations. This information is then used to extract a detailed trailing edge and it curvature, which is then matched to other trailing edges in the database via dynamic time warping. Using this method, we achieve nearly 80% top-1 ranking accuracy on a large subset of the SPLASH [9] dataset consisting of about 400 identified individuals. We also show that in combination with a general appearance based matching algorithm, Hotspotter [12], we can achieve 93% accuracy.

To our knowledge, this is the first method that can achieve this level of accuracy on humpback fluke identification without extensive manual effort at test-time.

CHAPTER 1
Introduction

1.1 Humpback Whales

Since the international ban on commercial hunting of Humpback whales in 1966, humpback whales have grown from a population of only 5000 [4] to over 50000 [8]. As the population grows, it becomes more and more important to be able to automatically identify individual whales in order to accurately monitor their population growth and follow their migration patterns, among other ecological conservation endeavours. One of the most reliable methods for photo-identifying humpback whales is by taking pictures of their flukes as they dive after breaching the surface of the water.

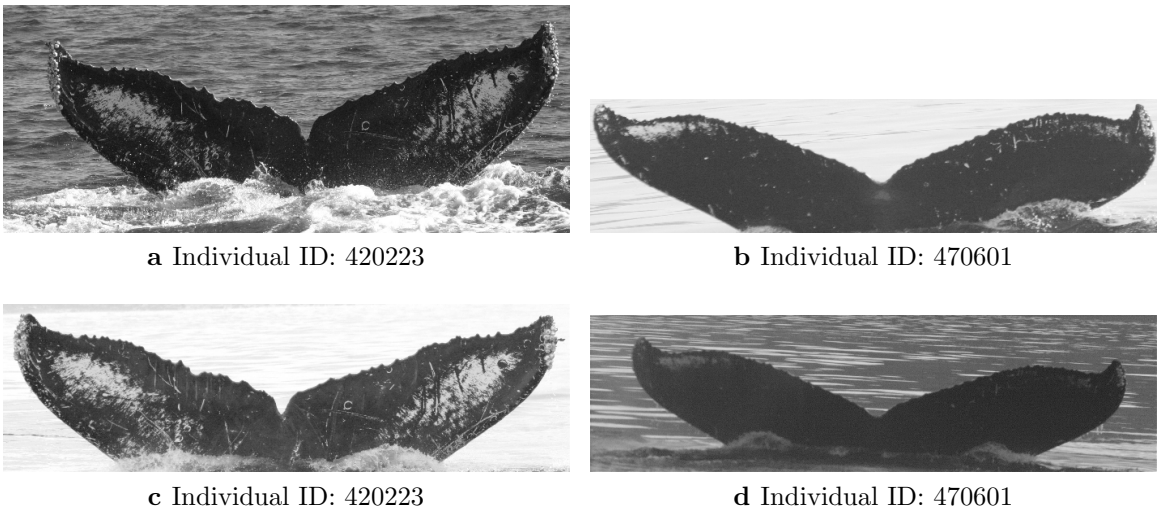
1.1.1 Distinguishing Individual Flukes

The primary distinguishing features of these flukes are — for the purposes of this work — separated into two main areas; the "trailing edge" of the fluke, and the internal texture (see Figure **??**[1]). The internal texture is a more obvious choice for identification, as it is disinctive even from a distance even when the image is blurred. Unfortunately, some humpback flukes have indistinct (e.g. all-black) internal textures, which make matching based on texture impossible (see Figure 1.2). Additionally, the work of Blackmer et al. [7] finds that the trailing edge changes less with age than the internal texture of the fluke, which means that it can (potentially) be a more reliable identifier over time. That said, the requirements for getting a good photograph of the trailing edge can be impractical, as the trailing edge can be obscured by out of plane rotations (see Figure 1.3).

1.2 Current Identification Methods

Computer-assisted photo-identification of humpback whale flukes has been attempted since the early 90s [35]. While early efforts mostly relied on a manual

[1] All images of humpbacks flukes in this thesis come from this dataset



a Individual ID: 420223 **b** Individual ID: 470601

c Individual ID: 420223 **d** Individual ID: 470601

Figure 1.1: Example Flukes. Example images of humpback whale flukes from the SPLASH [9] dataset. These flukes both have distinctive internal textures (more so on the left). However, the trailing edge on the left is far more distinctive than the trailing edge on the right.

description of the fluke that would then be matched (against other stored descriptions) [35], [54], later efforts have involved matching flukes based on automated analysis of both the internal texture and trailing edge [24], [29], [19].

Existing computer-assisted photo-identification methods can be broadly separated into three categories. There are manual methods, in which humans both identify and describe distinguishing features, semi-automated methods, in which humans identify distinguishing features that are then described and matched automatically, and automated methods — which can match based solely on raw images with no human involvement.

1.2.1 Based on Trailing Edge

In the I3S contour system [19], the user must input start and end points on the query trailing edge, after which its contour is extracted. This trailing edge is then resized and aligned so that it can be compared with absolute difference against the database trailing edges. It also compares a set of possible shifts, rotations, and scales of the query trailing edge to account for these differences. At the time of writing no published results on this system applied to humpback whales could be

Figure 1.2: Uniform Internal Texture. This image of a humpback fluke shows no clear internal texture, but a distinctive trailing edge.

found.

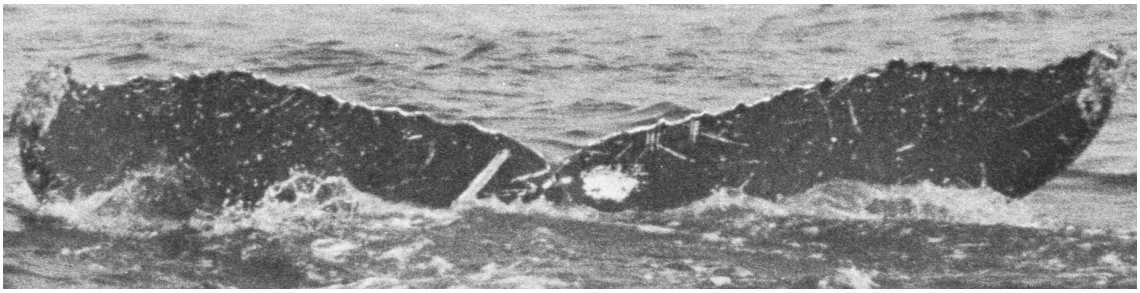
Automatically identifying humpback whales by their entire trailing edge contour is done experimentally in Hughes et al. [24], using a technique that is originally designed for great white sharks. This technique segments the trailing edge into a set of possible contours and matches them combinatorially using Difference of Gaussians. The authors achieve a comparable accuracy to our method, however for a much smaller dataset of humpback flukes than the one evaluated here.

While trailing edge matching has seen limited use in humpback whale identification, it is a much more common technique in sperm whale (P. macrocephalus) identification [23], [5] [54], with varying levels of manual effort. In Whitehead's work [54], points of interest on the trailing edge are entered and catalogued manually along with their positions. In order to match these trailing edges, all of the points are compared against points on annotated trailing edges in the database, using a distance threshold to ensure locality. This is a manual method, requiring extensive annotation for matching.

The method proposed by Huele et al. [23] uses a semi-automatic extraction of sperm whale trailing edge, and then applies wavelet transformations which are cross-correlated to determine a similarity measurement.

An investigation of the above methods for sperm whale identification is carried out in [5], showing that combining these two methods yields an 80% top-1 accuracy (meaning that the correct identity is ranked at the top of the potential matches 82% of the time) on a slightly smaller dataset than ours. However on their own each

a



b

Figure 1.3: Change in Trailing Edge. The above images show that out of plane rotations of the fluke can obscure it or otherwise make it hard to match. These images are both of the same individual, however in the top image the fluke is rotated slightly towards the camera.

method only achieves about 65% top-1 accuracy.

1.2.2 Based on general Fluke appearance

The primary method for computer-assisted photo-identification of humpback whale flukes is to use the internal fluke pattern, as seen in the work of Mizroch et al. [35] and Flukematcher [29].

In [35], information about the fluke is manually catalogued and used to match individual whales. The fields that are catalogued contain information primarily about the overall coloration patterns of the fluke, as well as the shape of the central notch. The matching algorithm generates potential fluke matches by looking at how similar the annotated patterns are. This requires significant manual effort to identify individuals.

In Flukematcher [29], control points are manually annotated which allow the program to automatically find pigmentation patterns in the fluke and align accord-

ingly. Optionally distinctive fluke patterns can also be selected by the user. A variety of heuristic features are then extracted, which are matched using a variety of similarity measures. This method has acheived a 82% top-1 accuracy on a smaller dataset than ours. However, it requires significant manual effort on the order of five minutes per fluke photograph.

1.3 Method Outline

In this thesis, we develop and present an efficient fully-automated[2] algorithm that identifies humpback flukes based on their trailing edges. For each fluke, the algorithm first determines the left and right tip points of the fluke (as well as the bottom of the central notch). The trailing edge contour is then extracted between the left and right tip points, and for each point on this contour curvature is measured at multiple scales. Once these curvatures are extracted for each fluke photograph, the algorithm identifies a query curvature by computing a distance from it to each database curvature with dynamic time warping. The query fluke is then identified with the identity of the database fluke with the lowest distance.

We also show we can greatly improve the accuracy of this method by combining it with matches found by Hotspotter, a generalized pattern based identification method that is a powerful matching algorithm for several species [12]. Hotspotter identifies multiple salient SIFT keypoints in an image that are then spatially verified and matched with other keypoints in the database to produce a ranking over possible identities. Despite the general nature of this method, we find that it does not identify keypoints on the trailing edge and thus struggles with flukes that have no significant internal texture. This work is the first to our knowledge that details Hotspotters efficacy when applied to humpback whale flukes, and the results are presented in Chapter 4.

1.4 The Dataset

The main dataset that is used and evaluated in this work is a subset of the dataset collected by the SPLASH project [9]. It consists of about 1400 identified

[2]With the caveat that manual annotation is needed to train parts of the algorithm

photographs spread over about 860 identified individuals. Of these, only 433 individuals have more than one image associated with them, giving 942 images that can be used in a one-to-one comparison. We refer to this dataset as the Flukebook dataset, which is the team from which it originates.

Additionally, an external dataset of unidentified (but annotated) humpback flukes is used for training individual components of the method.

1.5 Thesis Outline

The rest of this thesis starts by giving a background on the algorithms that our method is based on in Chapter 2. Afterwards, the main method is detailed in Chapter 3, along with a description of some alternate methods that failed to work as well. Chapter 4 goes into the results, failure cases, and success cases of the primary algorithm as well as various parameter changes and how they affect the results. We also describe the effectiveness of this identification method in combination with Hotspotter. This thesis concludes with a discussion on the failings of the primary method, as well as ways to improve it and generalize it to extracting and matching edge characteristics in other animals.

The primary contributions of this work are the individual components of the main method for extracting trailing edges and fluke keypoints, as well as the combination of all the components into a coherent identification pipeline. We also contribute an evaluation of dynamic time warping on curvature measures as a method for matching humpback flukes, as well as an evaluation of Hotspotter for this task.

CHAPTER 2
Background

In this chapter, we provide a series of sections detailing background information on the algorithms on top of which our trailing identifier was developed. We describe some of the applications and variants of deep convolutional networks on top of which we build our fluke keypoint and trailing edge extractors. We briefly introduce the concept of seam carving as it relates to our trailing edge extraction algorithm, and provide a small overview of contour curvature measures. Additionally, we describe briefly the concept of dynamic time warping for sequence matching.

2.1 Convolutional Networks

In recent years, convolutional neural networks have provided state of the art results in several challenging computer vision tasks, including general image classification [30], [52], image segmentation [34], [10] and individual identification (specifically for human faces) [14], [47].

The essential idea of a convolutional network is that it we can use the gradient of an error signal to learn hierarchies of convolution kernels separated by nonlinear activation functions. These networks are considered to be a form of neural network where the convolutions provide a meaningful prior when applied to data with spatial relationships (e.g. image data). Convolutional were introduced nearly 40 years ago by Fukushima in [16]. The current incarnation of these networks can be traced back to the seminal work of LeCun et al. [32]. Modern convolutional networks follow a common framework of using Rectified Linear Units (ReLUs) as activation functions, and Dropout [21] layers after fully connected layers for regularization. Additionally, the convolutional kernels used are commonly small square kernels with "same" padding alternated with $2\times$ downsampling layers (specifically max pooling layers) [49], [48], [30].

The convolutional networks used in this thesis follow the above framework, and also use batch normalization [25] at every layer. We also use orthogonal initialization

[46] at all layers to help ensure gradient flow.

2.1.1 Facial Keypoint Prediction

Facial keypoints are essentially coordinates on an image of a (human) face that detail the locations of nose, eyes, mouth, etc. They are commonly used in facial identification pipelines [53], as well as in motion capture [1] and expression recognition [6]. There has been recent work in using convolutional networks for facial keypoint prediction [51], [40]. The essential idea behind these networks is that they predict points (rather than classifications) in the form of (x, y) coordinates, and are trained with a regression loss function. In this work, we adjust these methods to predict fluke keypoints marking the left and right tips of the trailing edge using essentially the same paradigm as the above works.

2.1.2 Fully Convolutional Networks

Classically, convolutional networks reduce an image to a single (spatially invariant) vector, which is then used for classification (or embedding, regression, etc.) To do this, these networks usually have fully connected (or dense) layers towards the end. This ensures that the receptive field of the network covers the entire image, which is practical for many applications where a scalar or fixed size prediction is required. However, when dealing with arbitrarily sized images, it is useful to use networks that are "fully convolutional", in which case the entire network consists of convolution kernels. Convolutional networks that reduce to dense layers can be cast as fully convolutional networks by replacing the dense layers with 1×1 kernels, using the dense units as channels. This technique is especially applicable in segmentation tasks (e.g. [39] [11] [18]), as this process allows for (downsampled) predictions spatially distributed throughout the image. By then upsampling and combining different stages of prediction, the authors in [34] produce high quality image segmentations, a technique that we replicate for classifying pixels as being part of the trailing edge. In this work, fully convolutional networks are used for predicting the 'trailing edginess' of an image, which allows us to refine the trailing edge contour extraction.

2.2 Contour Extraction

2.2.1 Active Contour

Automatically extracting contours from edges in images is an old technique, and one of the primary methods for getting coherent contours from image information is the active contour (or snakes) method [2] [27]. This technique explicitly models the contour as a function that minimizes a combination of curvature, smoothness, and image edge-ness constraints, weighted appropriately. Often these contours are fairly smooth, as the constraints imposed require a parameterized function. Additionally, this is an iterative process that is subject to local minima, meaning that it can produce different results based on the initialization in the image. For these reasons, we did not investigate using active contours for trailing edge extraction.

2.2.2 Seam Carving

Seam carving is a technique that tries to resize images without warping or distorting the objects shown in the image [3].

This technique uses a dynamic programming algorithm to find minimal salience paths through an image, where salience is often defined as the gradient. The motivation for this is that these minimal saliency paths are not important to the image, so they can be removed to reduce its size. While this method is not directly used in this work, the underlying algorithm for trailing edge extraction is essentially a single iteration of the seam carving algorithm, using gradient information.

2.3 Curvature Measures

Contour curvature measures are commonly used to characterize the overall shape of an contour. A lot of work has been done on using curvature information for detection [36], classification [15] [31]. This curvature information can be broadly broken down into either integral or differential curvature, and is usually computed at multiple scales.

Differential curvature can generally be seen as measuring the angle of the tangent normal of the gradient at each point in an image [15]. For our purposes, we can then take only those points that lie on the contour and use their curvature.

While doing this directly can be fast to compute, it tends to be noise sensitive and we found that integral curvature (below) works better for our purposes.

Integral curvature works (conceptually) by sliding a circle of some radius r along the contour [42], and measuring how much of the circle is 'inside' the contour. This measurement is usually taken at multiple scales, and has the appealing property of being invariant to rotation and translation (of the entire contour). In this work, we approximate the circular curvature with a square of size r, which appears to perform just as well but can be computed much faster.

2.4 Dynamic Time Warping

In deciding a sequence comparator, one criterion that is often important is ensuring that small shifts in the sequence do not balloon into large differences. Dynamic Time Warping (DTW) is a sequence comparison method that, roughly, finds the optimal matching between all sets of points in the two given sequences that minimizes the overall distance (for some defined distance function) between the matched points, while keeping the locality of the points intact [44]. This allows for shifts and some warps in the two sequences to be compensated for, and results in a nonlinear mapping of one sequence onto another. The algorithmic complexity of dynamic time warping can be limiting in large datasets, as it is quadratic in both space and time – making a one-to-one comparison a bit daunting.

There are several variants on DTW that give faster speeds [45] [33], however we only use the Sakoe-Chiba bound [44], which both constrains the neighborhood in which points can be matched and gives a complexity of $O(nT)$, where T is a user set parameter constraining the neighborhood.

Sequences of curvature measures have been used with DTW for signature verification [38], however this combination has not been used for matching trailing edges to our knowledge.

CHAPTER 3
Methods

In this chapter, we describe in detail the humpback fluke trailing edge algorithm pipeline. We also briefly discuss some alternative approaches that we found to have limited successs.

3.1 Trailing Edge Extraction

Extracting good, high quality trailing edges images is one of the primary challenges when matching Humpback whales by their trailing edge. In this section, we describe the steps that go into automating the extraction of high quality trailing edges.

One major assumption we make when extracting these trailing edges is that the humpback whale fluke is aligned such that its major axis is horizontal. Additionally, we generally assume that all parts of the fluke are present. While these assumptions do not make for an incredibly robust system, the nature of the problem (and the dataset that we had at hand) makes these assumptions reasonable. It would also be possible to add a detection step beforehand to ensure this assumption, however we do not explore this in this work.

3.1.1 Fluke keypoint prediction

When extracting trailing edges, one of the first steps we take is to identify the starting and ending points of the trailing edge, as well as the bottom of the central notch. To do this, we train a convolutional network to predict these three points.

The convolutional network does not need full-sized images, so the first step of the keypoint extraction pipeline is to resize the image to 256×256 pixels. This size choice is somewhat arbitrary, but we find that it provides strong performance without using an unnecessary amount of memory. The network predicts the points as values between 0 and 1, essentially giving coordinates into the image as percentages of its height and width. These predictions are then then rescaled back up to the

Figure 3.1: Example Keypoint Prediction. Example image showing the left tip, bottom of the notch, and right tip located by the keypoint extractor convolutional network.

original image size by multiplying with the original image height and width. An example prediction is shown in Figure 3.1.

3.1.1.1 Network Design

The overall design of the network follows the pattern of alternating small (3×3) convolutional filters with 2×2 max pooling layers, at each step doubling the number of channels (starting with 8 channels). This is somewhat similar to VGG-16 [49], although with half the trainable layers. After a $32\times$ downsample has been achieved, we attach a decision layer which consists of a dense layer followed by three separate dense layers with separate predictions layers after (one for each point being predicted). While this is not a common approach in keypoint prediction, we found that it gave better performance than having the points predicted as a single vector. We theorize that this may be because shared units between each of the three predictions leads to stronger correlations between them, reducing overall prediction flexibility.

3.1.1.2 Training Details

Generating the training data for this is straightforward given a set of annotations with the associated points to learn. The dataset that we created for this purpose contains approximately 2100 training images, 700 validation images, and

900 test images.

First, each image is resized to a fixed width while maintaining the aspect ratio. This is done to somewhat normalize the relative scale of objects in each image on the assumption that they are constrained to contain the fluke. Each image is rescaled to the network size, and then the corresponding targets are rescaled to the range $[0-1]$. The size of the original image is recorded as well. The loss function that we use for each point is the Euclidean distance between the target and predicted point. We also include a scalar scaling factor α, which scales each point by a proportion of the original image size $\vec{s}$. Thus, we have the scaled Euclidean loss SE

$$SE(\vec{t}, \vec{p}, \vec{s}, \alpha) = \|(\alpha * \vec{s}) \odot (\vec{t} - \vec{p})\| \tag{3.1}$$

Where $\vec{t}$ and $\vec{p}$ are the target and predicted coordinates respectively [3]. We then average this loss function over each point that the network predicts.

The networks are trained for 1000 epochs with α set to 2e$-$2 using the Adam [28] optimizer (with recommended settings) and $l2$ regularization on the trainable parameters with a decay of 1e$-$4. All of these hyper parameters were tuned using the validation set, although the possible parameter space was not fully explored due to time constraints.

3.1.1.3 Evaluation

On average, the best network achieved a 10 pixel distance error on the validation and testing sets (in the original image scale). While this may seem like a lot, the trailing edge extraction (and subsequent matching accuracy) was not severely affected when only using the start and end point predictions.

We find that, for the vast majority of images, the network achieves a low pixel distance error, while there are a few that have a much higher error (see Figure 3.3). Qualitative inspection of these images shows that they are either of flukes which are not the singular or major object in the image, or flukes that are significantly rotated out-of-plane relative to the camera. An example of this is shown in Figure 3.2.

We attempted to use a spatial transformer network [26] to try and handle these

[3] $\odot$ denotes elementwise multiplication

Figure 3.2: Example Keypoint Failure. Example image showing a keypoint extraction failure case from its testing set. Note the difference in pose of the fluke from the success case shown in Figure 3.1. This is an example of a fluke image that violates our assumptions.

cases, but we were unable to get it to perform as well as the standard convolutional network, nor produce sensible transformations. Since these failure cases represent a small amount of the dataset, it is both difficult to train a network to handle them, and simultaneously not a large issue. Additionally, we find that keypoint extraction failures are only a small percentage of the matching failure cases that we encounter.

3.1.2 Basic Trailing Edge Extraction Algorithm

The base algorithm for extracting the trailing edge uses the vertical gradient information of the image (denoted as I_y). We extract I_y using a vertically oriented 5×5 Sobel kernel [50].

Then we normalize I_y with min-max scaling, giving N_y as

$$N_y = \frac{I_y - \min(I_y)}{\max(I_y) - \min(I_y)} \tag{3.2}$$

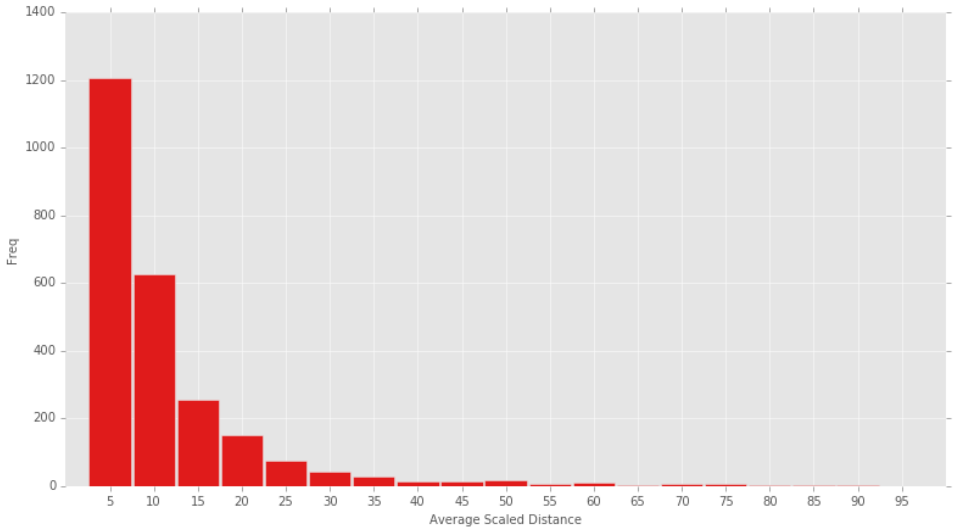


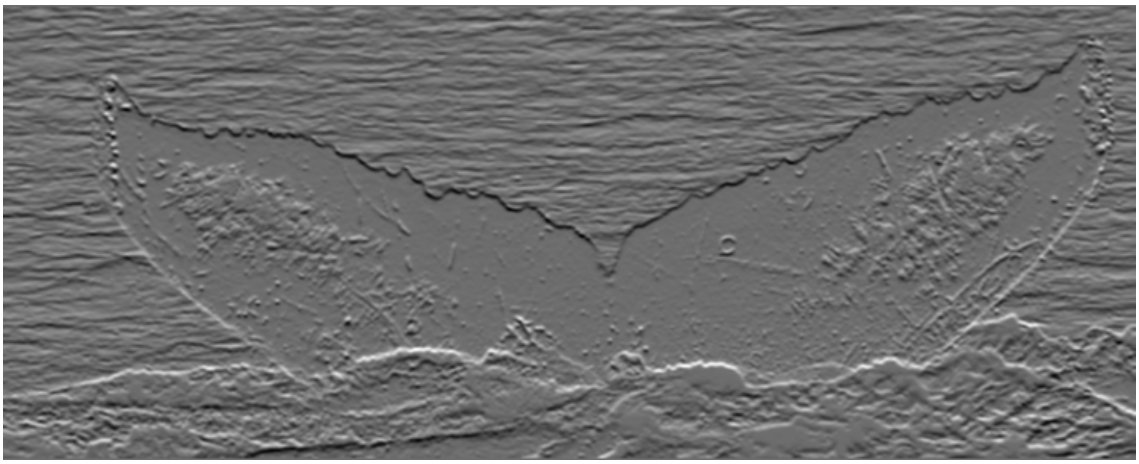
Figure 3.3: Histogram of Keypoint Distances. This is a histogram of the average distance from predicted keypoints to annotated keypoints on the testing set for the keypoint extraction network. The vast majority of keypoints are predicted within 10 pixels of the true keypoints.

Given N_y, we extract a trailing edge starting at the left tip of the fluke (a point denoted s) and ending at the right tip (a point denoted e). To do this, we scan each pixel (i, j) in N_y starting from the column $s_x + 1$ and ending at the column e_x, updating a cost matrix C cost with the following recursive update rule:

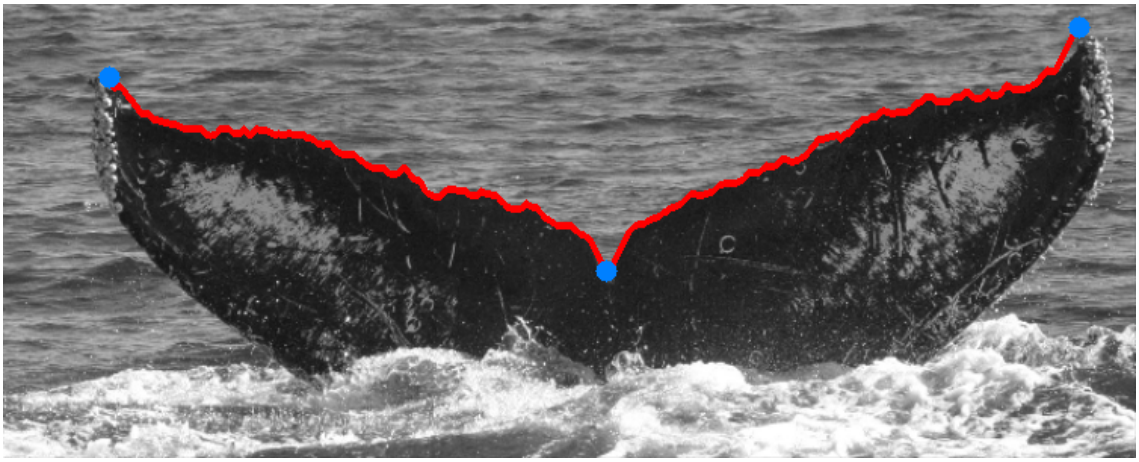
$$C(x, y) = \begin{cases} 0 & x < 0 \\ \infty & y < 0 \text{ or } y > h \\ \min_{y-n \leq y_c \leq y+n} C(x-1, y_c) + N_y(x, y) & \text{else} \end{cases} \tag{3.3}$$

Where n is a neighborhood constraint and h is the height of N_y. We default n to 1, meaning that each pixel considers 3 "neighbors" y_c from the previous column.

As C is filled out, we also keep a backtrace matrix B, which keeps track of the index of the minimal candidate neighbor chosen in equation (3.3).



a I_y



b Extracted Trailing Edge

Figure 3.4: Example Trailing Edge Extraction. Example of the baseline trailing edge extraction with $n = 2$. Note that the gradient image has a significant black area where the trailing edge is, making this an easy case.

$$B(x, y) = \text{argmin}_{-n \leq n_c \leq n} C(x - 1, y + n_c) \tag{3.4}$$

Once the end column is reached, we scan the columns in reverse order from e to construct the path by adding the chosen neighbor from B at each step. More formally, we start the trailing edge sequence TE^0 at (e_x, e_y), and add elements to TE as follows

$$TE^i = (TE_x^{i-1} - 1, (TE^{i-1})_y + B(TE_x^{i-1}, TE_y^{i-1})) \tag{3.5}$$

In order to enforce that the path begins at s, we apply the following process before running the above.

$$N_y(s_x, \cdot) = \infty \tag{3.6}$$

$$N_y(s_x, s_y) = 0 \tag{3.7}$$

This forces the path to start at s, as otherwise it would take on infinite cost. We optionally do the same for the point denoting the bottom of the notch, although we find that this can affect matching accuracy negatively if it is too far from the trailing edge.

An example of the base algorithm's output is given in Figure 3.4. While this algorithm has no understanding of humpback whale flukes, it generally finds high quality trailing edges in images that are constrained around the fluke with oceanic backgrounds (which is a large majority of the dataset at hand). In the next section, we outline our approach for trying to make this more robust.

3.1.3 Trailing Edge Scoring

As mentioned in the beginning of this section, using only the gradient information for extracting the trailing edge works in many cases, but is not a robust method.

If we had a score of each pixel's "trailing edginess" in an image, the trailing edge extractor could make use of this information to make better choices in trailing edge extraction. To do this, we need a prediction of whether or not each pixel belongs to the trailing edge of a fluke — a task that is best suited to a fully convolutional network.

In the fully convolutional networks that we use, we learn "same" convolutional kernels at each layer so as to maintain spatial shape from one layer to the next, except at explicit downsampling layers. In order to construct a "same" kernel, we use square kernels of size $k \times k$ such that k is odd, and apply them with a stride of 1 in each direction. Then, in order to ensure that the size of the output is the same as the size of the input, we add a zero-padding of size $\lfloor \frac{k}{2} \rfloor$ pixels to each side (filling

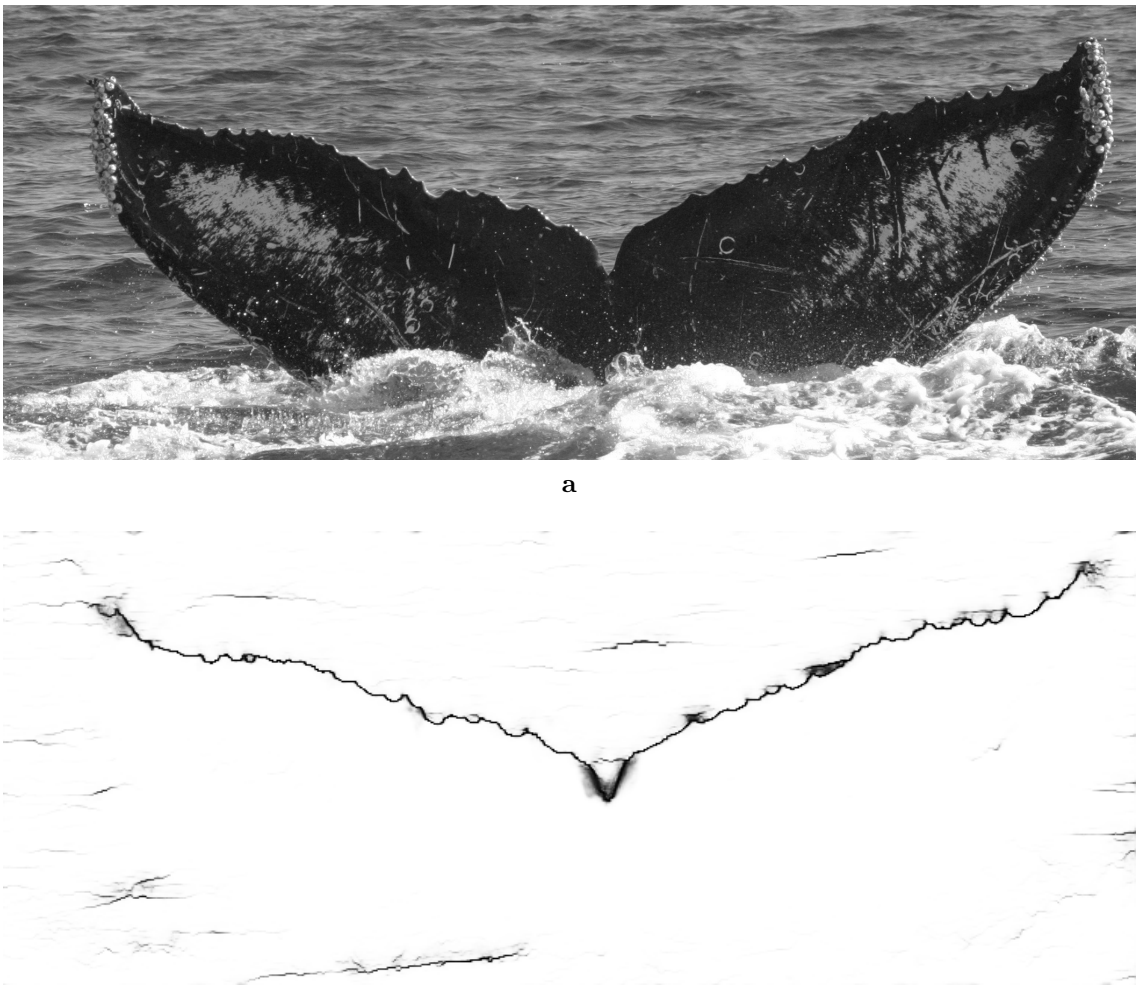
in the corners). In the networks used for trailing edge scoring, all convolutions (aside from the downsampling, i.e. max pooling layers) are "same" convolutions, The four major variants on trailing edge scoring networks that we evaluated are detailed below. All of these networks function on the same paradigm of taking an arbitrarily sized image and producing an image of the same size but with a class score for each pixel.

The dataset was sourced from trailing edges extracted using the basic method detailed above, however with manual adjustments to fix many of the more common issues we encountered with this algorithm. These manual adjustments often included manually adding "control points", which forced the trailing edge through a specific point and often fixed major failures. Additionally small adjustments were made that would not be found by the gradient finding algorithm. Due to the extensive manual effort required to annotate and correct these images, only about 500 images were annotated, many of which required little to no manual adjustments. With these manually corrected trailing edges, we generated the training set for trailing edge scoring by extracting a 128×128 patch at 128 pixel intervals along the trailing edge (giving a positive patch), and a corresponding patch (with no trailing edge pixels) randomly sampled from the left over space above and below the trailing edge (giving a negative patch). As the images that this dataset were extracted from were generally 960 pixels in width (with the trailing edges being only slightly smaller), on average about 15 patches were extracted from each image. These patches are then randomly split into training, testing, and validation sets each with 3700, 1200, and 1600 patches respectively.

3.1.3.1 Trailing Edge Scoring Architectures

Several network architectures were tried, although here we only report the major variants. One major consideration that has to be made when selecting a fully convolutional network architecture is its receptive field[4]. If the receptive field is too small, it may not have enough information to accurately determine if a given pixel is part of the trailing edge. However, in order to increase the receptive field without

[4]The receptive field of a convolutional network is, at a high level, the region of input that affects the output at a given layer



a

b Trailing Edge Scores

Figure 3.5: Example Trailing Edge Score. Bottom image is the Residual scorer's classification of the top image. Trailing edge is class is colored black.

massively increasing the depth of a network, we must downsample, which makes the output less fine-grained.

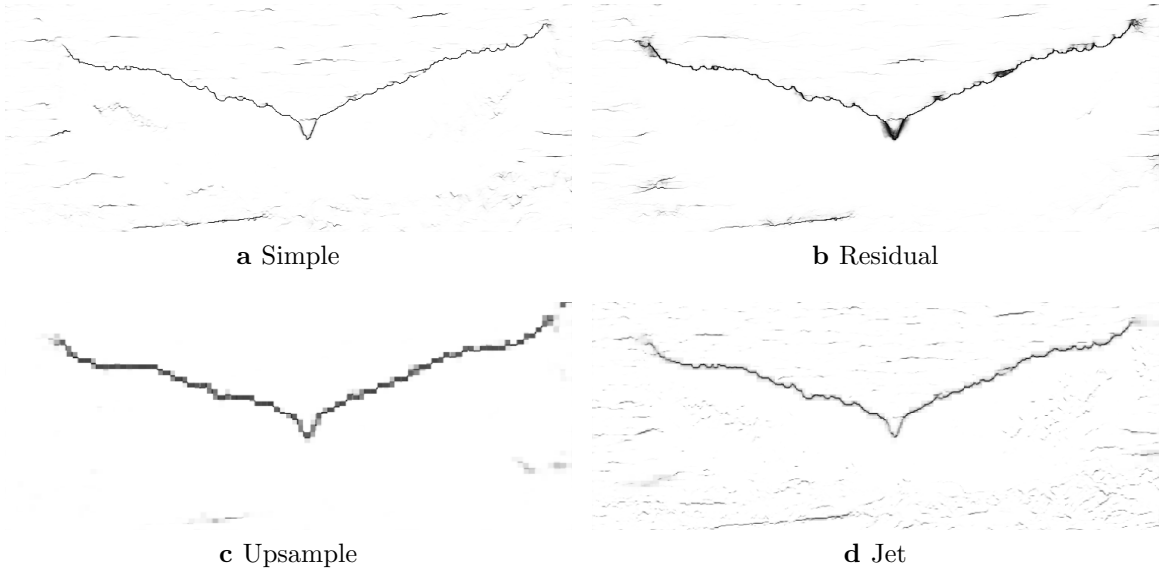
Simple This network is simply a stack of 6 "same" convolutional layers (of decreasing spatial extent), with no downsampling regions. This has a small receptive field, but can produce detailed predictions. Due to the small receptive field however, at convergence it gives low precision predictions, and seems to (in many cases) produces many of the same mistakes that normal gradient based trailing edges do.

Upsample To deal with the small receptive fields, convolutional networks typically downsample at various stages. This downsampling usually takes the form of max pooling (instead of e.g. convolutional kernels with a stride greater than one). The Upsample network downsamples the input 8× through alternating "same" convolution layers and max pooling layers, makes a prediction, and then simply upsamples its output (with bilinear interpolation) The Upsample architecture is analogous to the FCN-32s architecture in [34].The goal of this is to increase the receptive field of the network's output layer, although the predictions it makes are very "blocky".

Jet Following the deep-jet architecture from [34], we modify the Upsample network to combine prediction layers at intermediate levels of downsampling. The essential idea behind this is to take the prediction output from the last downsampling layer, and combine it with predictions made on the output of the layers before the downsample. This is repeated in a cascade, ending with a combination of the merged predictions and a prediction made on the last convolutional layer before the first downsampling layer. This architecture can give more fine-grained predictions than the Upsample network while taking advantage of the increased receptive field size.

Residual While the receptive field of the Simple network is small, it can produce very fine trailing edges, which we found to be necessary for good trailing edge extraction. However with such a small receptive field it can be more prone to making mistakes. It is possible to create a very deep network of "same" convolutions, however training very deep networks like this can run into problems very quickly, as found in the work of He et al. [20], which introduce residual connections to address this problem. These connections are soimply shortcut connections from one layer to another, a surprisingly simple method that allows the network to learn the "residual" of its input rather than the whole transformation. We create a residual network architecture by stacking 64 3×3 "same" convolution kernels, and adding a residual connection every other layer. This is our best performing network.

An example of all of the above networks' outputs can be seen in Figure 3.6.



a Simple **b** Residual

c Upsample **d** Jet

Figure 3.6: Trailing Edge Scores. These are the trailing edge scores given by each of the networks described in section 3.1.3.1 on the image used in Figure 3.5.

3.1.3.2 Using the trailing edge scores

Once we have a "trailing edginess" score for each pixel in an image, we need to combine this information with the normalized gradient N_y in a way that causes the trailing edge extraction algorithm to follow those pixels that the scoring map marks as trailing edge. More formally, the trailing edge scoring map gives us an image $T_p \in [0-1]^{w \times h}$ (where w and h are the width and height of the image) which denotes the network's predicted probability of each pixel being part of the trailing edge. The most simple and obvious way to combine this information with N_y from before is to combine them with a mixing parameter β.

$$S_{te} = (1 - \beta) * N_y + \beta * (1 - T_p) \tag{3.8}$$

We use $1 - T_p$ in this case because we are minimizing the path through S_{te}. Once done, the trailing edge extraction algorithm procedes as described in the previous section.

For most of the evaluation process, we simply set $\beta = 0.5$, and we find that this gives us the best results.

Another variant on combining T_p and N_y that we tried was to dilate the

trailing edge predictions and then forbid the trailing edge from going outside of those predictions. An inherent difficulty with this approach is that in order for it to work, we need to have a guarantee from the trailing edge scorer that it will not produce major gaps in its predictions. If there are such gaps, then the trailing edge will not be extractable. It is difficult if not impossible to guarantee that this will not happen, although with more data the risk can be mitigated. In our experience, these breaks were common enough that this approach was abandoned.

3.1.3.3 Training Details

All networks were trained for 100 epochs (or until convergence) with a batch size of 32, with $l2$ regularization using a decay of 1e$-$4. We used the Adam optimizer [28] (with the recommended settings) for calculating weight updates.

One detail that turned out to be important is the class imbalance. The trailing edge pixels (necessarily) make up a small percentage of the total image, meaning that these networks could get a fairly high accuracy (and thus low loss) simply by predicting only background pixels. In order to prevent this, we only sample a negative patch once for every positive patch (as detailed above), and additionally we weight the loss for the trailing edge pixels $10\times$ higher than the loss for the background pixels.

As noted earlier, many of the manually annotated trailing edges did not require extra input. This can be an issue as it biases the network towards marking pixels as trailing edge when they would be marked as such based on just the image gradient. In order to mitigate this issue, we included some data augmentation, namely random Gaussian blur and randomly inverting the pixel intensities with probability of $P = 0.3$. The inverting of pixel intensities as data augmentation is meant to try and influence the network to handle cases where the trailing edge is white against a somewhat darker background, producing an inverted gradient compared to when the trailing edge is dark. Unfortunately this had limited success.

All of these hyper parameters were tuned using the validation set, although the possible parameter space was not fully explored due to time constraints.

3.1.3.4 Evaluation

Due to the overwhelming class imbalance, the model accuracy is rather meaningless (e.g. the accuracy of an all-background prediction is 99%). Instead, we report the intersection-over-union (IoU) score between the two pixel label classes (i.e. background and trailing edge), which gives a much better idea of model performance. We also report the precision and recall of the model.

	Training			Validation			Testing		
Architecture	Pr.	Re.	IoU	Pr.	Re.	IoU	Pr.	Re.	IoU
Simple	0.59	0.95	0.57	0.59	0.94	0.57	0.60	0.95	0.59
Upsample	0.17	0.88	0.17	0.17	0.86	0.17	0.17	0.87	0.17
Jet	0.62	0.89	0.57	0.62	0.88	0.57	0.63	0.89	0.58
Residual	0.57	0.93	0.54	0.57	0.92	0.54	0.58	0.93	0.56

Table 3.1: Table showing the precision, recall, and IoU of each of the evaluated trailing edge scorers on each section of the trailing edge dataset. For the purposes of this analysis, we use the `argmax` over the classes to determine a positive (i.e. trailing edge) or negative pixel.

Initially, we thought that precision would be the most important metric for evaluating networks (which is reflected in the class weighting). However, we found that even a low precision classifier could give good trailing edges, and that the important measure was how detailed these trailing edges were. This is likely because small segments of predicted trailing edge would not be chosen by the trailing edge extractor if they are too disconnected from the actual trailing edge.

3.2 Trailing Edge Matching

Given extracted trailing edges, we now define a method for doing a one-to-one comparison between a given query and database trailing edge. The simplest way to do this is to define a (potentially non-metric) distance function between any two trailing edges. Once this distance function is defined, we can identify an individual by its trailing edge (referred to as the query trailing edge) by looking at the identity of the closest trailing edge in the database. As a distance function we use dynamic time warping over block curvature measurements, using a weighted Euclidean distance as a local distance function between curvatures.

This is essentially a ranking function over a database of identities, where each identity is assigned the rank of the closest image. There is a slight detail in how the multiple images per identity is handled, which we will describe briefly in a later section.

3.2.1 Curvature Measurement

In order to do this, we first extract the curvature from the trailing edge. Given the trailing edge as a sequence of coordinates into the original image, we construct a zero-image I_0 of shape similar to the original image. Each pixel corresponding to and below the trailing edge in I_0 is set to 1. This essentially means that everything "inside" the fluke is set to 1, and everything outside the fluke is set to 0 — with the assumption of course that everything below the trailing edge is part of the fluke. Because the trailing edge extractor produces only one coordinate per column, we can do this safely, however this algorithm could be easily adapted to this not being the case. Once this is done, we calculate a summed area table [13] ST from I_0 as follows.

$$ST(x, y) = \sum_{i=0}^{i=y} \sum_{j=0}^{j=x} I_0 \tag{3.9}$$

Conceptually, the next step is to slide a square of shape $s \times s$ centered on each point, and measure the percentage of that square that is within the filled in trailing edge. These values s are the different scales at which we meassure curvature, and are computed as a percentage of the trailing edge length.

To do this, we compute $BC_s(x, y)$ for each (x, y) coordinate in the trailing edge.

$$\begin{aligned} b(i) &= i - \frac{s}{2} \\ e(i) &= i + \frac{s}{2} \\ BC_s(x, y) &= \frac{(ST(b(x), b(y)) + ST(e(x), e(y))) - (ST(b(x), e(y)) + ST(e(x), b(y)))}{s^2} \end{aligned} \tag{3.10}$$

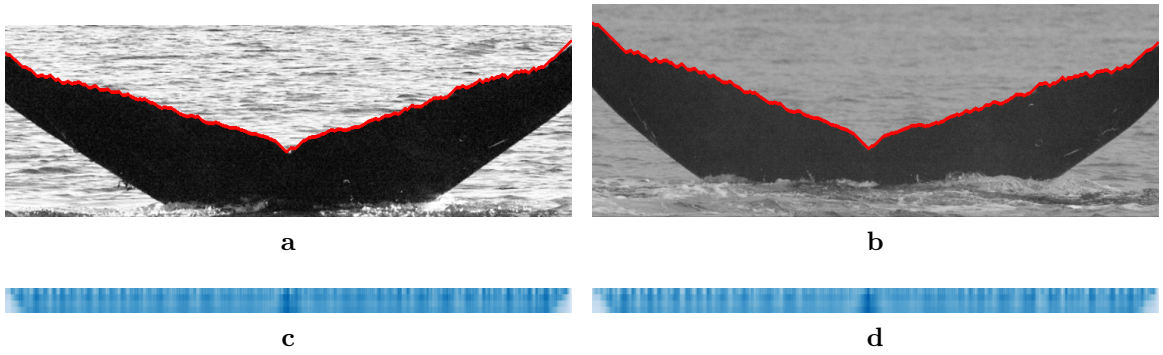


Figure 3.7: Trailing Edge Curvature. The top images are of the same individual, and the bottom images visualize the corresponding curvatures for the trailing edges that were extracted. Each row in the visualization is a curvature scale, increasing from top to bottom. Note that darker blue implies a "valley" in the trailing edge, whereas lighter blue implies a "peak".

The numerator in equation (3.10) gives the total area within the square that is below the trailing edge, which we then normalize by dividing by the square's area.

The set of scales to choose presents a large parameter space, however we have found that the scales $S = [2\%, 4\%, 6\%, 8\%]$ (as percentages of the trailing edge width) work well for our purposes. We then treat this curvature measurement as a $|S| \times l$ matrix BC, where l is the length of the trailing edge.

Two example curvatures (with their corresponding trailing edges) are given in Figure 3.7. In this case, we can see that at a high level, these curvature patterns look very similar.

3.2.2 Sequence Matching

Given two sequences of curvatures BC_1 and BC_2 (referred to as query and database curvature respectively) we match them using dynamic time warping as follows. First, we create a cost matrix C of size $l_1 \times l_2$ (i.e. the length of the first and second trailing edge respectively). We initialize this cost matrix by setting the first column and row to ∞, and then $C(0,0) = 0$, intuitively forcing the optimal path to match align the beginning of BC_1 with BC_2. Then, for each cell (i, j) in the cost matrix starting with $C(1,1)$, we use the following update rule

$$D_{\vec{s_w}}(c_1, c_2) = \|\vec{s_w} \odot (\vec{c_1} - \vec{c_2})\|_2 \tag{3.11}$$

$$C(i,j) = D_{s_w}(BC_1(i,\cdot), BC_2(j,\cdot)) + min(C(i-1,j), C(i,j-1), C(i-1,j-1)) \tag{3.12}$$

Where $\vec{s_w}$ is a vector giving the weight for each curvature scale, and $\vec{c}$ is a vector of the curvatures at different scales for a point. We then take the value of $C(l_1 - 1, l_2 - 1)$ as the distance between the two curvatures.

Additionally, we impose the Sakoe-Chiba [44] locality constraint T so that for each element i in BC_1, we only consider the range over elements j in BC_2 of $j \in [min(i - T, 0), max(i + T, l_2)]$. This essentially provides a bound over the possible matches between points on the trailing edge, preventing (for example) the first element of BC_1 from matching with the last element of BC_2. If we do not believe that these matches would be reasonable, then this bound not only prevents these (likely) erroneous matches, but also greatly speeds up the computation of the distance.

We set T as a percentage of l_1. For most of these experiments T is set to 10%, which appears to minimize the time taken for each comparison while preserving the overall accuracy of the algorithm.

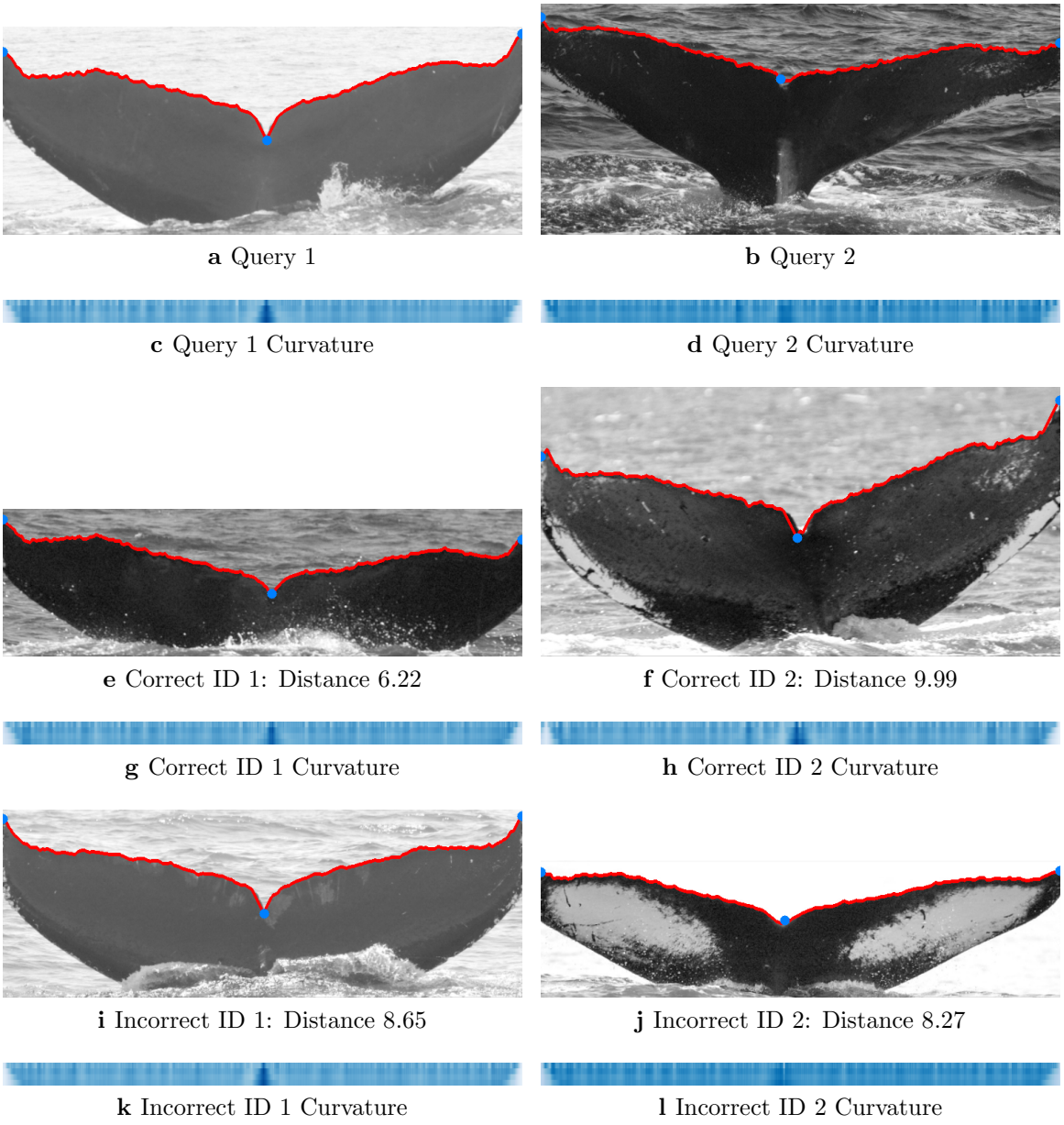
It's worth noting that while this distance measure is not a metric distance (i.e. it doesn't satisfy the triangle inequality), it is a symmetric distance as the local distance function (3.11) is symmetric [37].

3.3 Alternative Approaches

In this section, we list and briefly describe alternative approaches that were tried, although they did not prove accurate enough to make it into the final system.

3.3.1 Aligning Trailing Edges

One obvious pre-processing step that would make sense when comparing trailing edges is to make sure that they are aligned in image space. However, we found



a Query 1 **b** Query 2

c Query 1 Curvature **d** Query 2 Curvature

e Correct ID 1: Distance 6.22 **f** Correct ID 2: Distance 9.99

g Correct ID 1 Curvature **h** Correct ID 2 Curvature

i Incorrect ID 1: Distance 8.65 **j** Incorrect ID 2: Distance 8.27

k Incorrect ID 1 Curvature **l** Incorrect ID 2 Curvature

Figure 3.8: Example Matches. The left side shows a success case, and the right side shows a failure case.

that doing so when comparing curvature was often unnecessary (due to the invariances to rotation and translation, scale was taken care of separately), and using the Euclidean distance between points on the aligned trailing edges (i.e. in place of curvature for (3.11)) did not give good results.

There were two approaches to alignment that we evaluated, although neither

achieved top-1 accuracies above 20%.

3.3.1.1 Keypoint Alignment

As noted in the section on fluke keypoints, there are three points to predict — left, notch and right. Originally the intention for recording (and predicting) all three, as opposed to just left and right, was to have three corresponding points with which to estimate an affine transformation from database image onto query image. This would be done prior to any computation of trailing edge or curvature.

One major issue with aligning these images however is that if a non-affine transformation was required, the trailing edge itself would be warped in such a way that made matching difficult.

3.3.1.2 Dynamic Time Warping Alignment

We also attempted to align the trailing edges based on matches generated by the dynamic time warping, in similar fashion to the AI-DTW approach laid out in [43]. This approach used an iterative alignment process using the correspondences found by DTW (using either curvature distance or Euclidean distance as criteria). Essentially this method would find alignments using DTW, and then use these alignments to estimate an affine transformation of the database image onto the query image — and then repeat until convergence.

However, we found that this process oftentimes wouldn't converge, and when it did the alignments provided were of worse quality than those found by aligning the three fluke keypoints. Additionally, the extra time taken to carry out this process proved untenable.

3.3.2 Histogram Matching

One early curvature comparison method that we evaluated was to use histograms to match instead of a sequence-based method. This is a common approach for comparing curvatures [31]. We found that for our task even high resolution histograms did not provide enough detail to match the trailing edges properly, although this could potentially be explored further.

3.3.3 Embedding via Convolutional Networks

We also made an attempt at training convolutional networks to directly embed the images of flukes into a n dimensional vector, much like the work done for face recognition in Schroff et al. [47] and Parkhi et al. [41]. However, most of the previous literature on this technique is applied to larger datasets such as LFW [22], which is significantly larger than the dataset that we had available. A major factor in this is that these larger datasets often have five to ten images per identity (if not more), whereas most of the identities in our dataset had one or two images associated.

Regardless, we attempted the embedding approach (from raw images), however even a severely overfit convolutional network only achieved half the top-1 accuracy on its training set that the main method is able to achieve. We tried both triplet loss (a modified version of the one detailed in [47]) and contrastive loss [17] to no avail.

We believe that the small amount of images per identity is the main factor for the failure of these methods, and that a larger dataset would be necessary to properly train them.

CHAPTER 4
Results

In this chapter we present the results that our primary method achieves on the Flukebook dataset. The main results for the optimal method are given briefly, and then we discuss how different variations on the method affect accuracy.

4.1 Main method

The main method we settled on achieves an 80% top-1 accuracy on the dataset that we evaluated. The figures that we show in this section give the accuracy up to top-5 cumulatively. In general we find that there are no surprises up to top-5, and that relative accuracies do not change significantly as we increase the rank at which we allow a match.

The optimal configuration that we used for this method is given below.

- Crop size: 750px in width
- We do not use the notch for trailing edge extraction
- We set n to 1
- We use the Residual architecture for scoring trailing edge
- β is set to 0.5
- We use [2%, 4%, 6%, 8%] for our curvature scales
- We weight all curvature scales equally

4.1.1 Characterization of Success cases

Success and failure cases are difficult to characterize due to the nature of our dataset. (TODO put in something real for this)

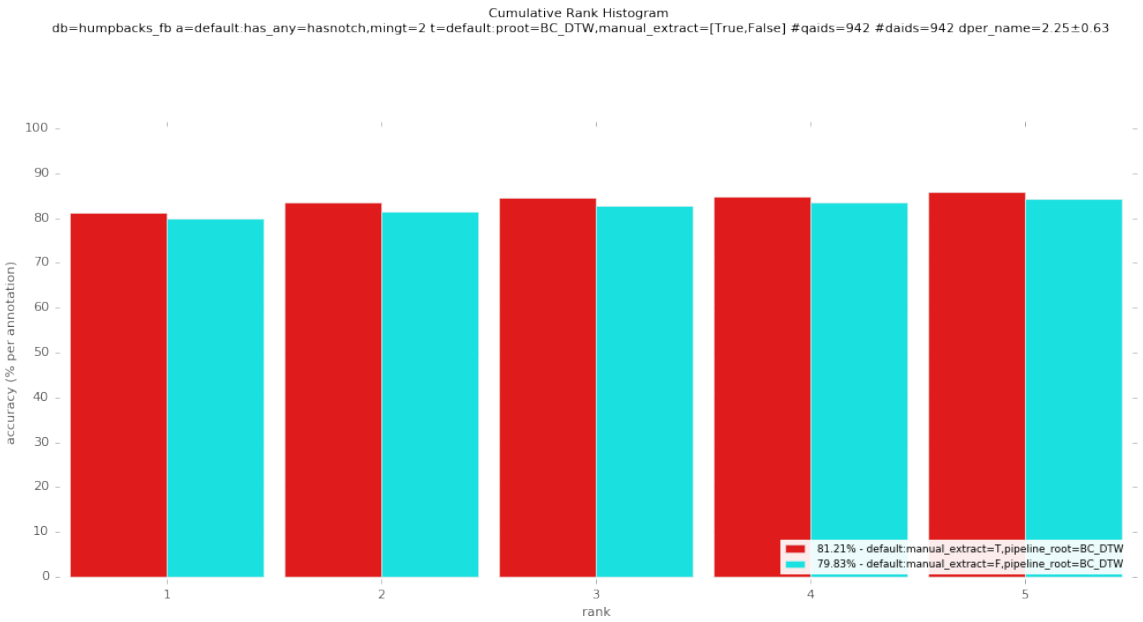


Figure 4.1: Varying Manual Extraction. We can see here that the difference in using the manually annotated points (purple) provided for this dataset versus the keypoint extractor's predicted points (cyan) produces a small drop in top-1 matching accuracy.

4.1.2 Characterization of Failure cases

Success and failure cases are difficult to characterize due to the nature of our dataset. (TODO put in something real for this)

4.2 Variations

4.2.1 Keypoint Extraction

4.2.1.1 Manual versus Automatic

One interesting note is to see how the final keypoint extractor compares with the manual annotations provided for the dataset. We can see in Figure **??** that the manual extraction is only slightly better than the keypoint network extraction. It is worth noting that none of the images in the keypoint network's training, validation, or even testing set came from the images evaluated here, so we believe that the keypoint extractor is very close to perfect for this task. While this does not mean that the keypoints predicted are perfect, it does imply that they are "good enough" to extract a matchable trailing edge.

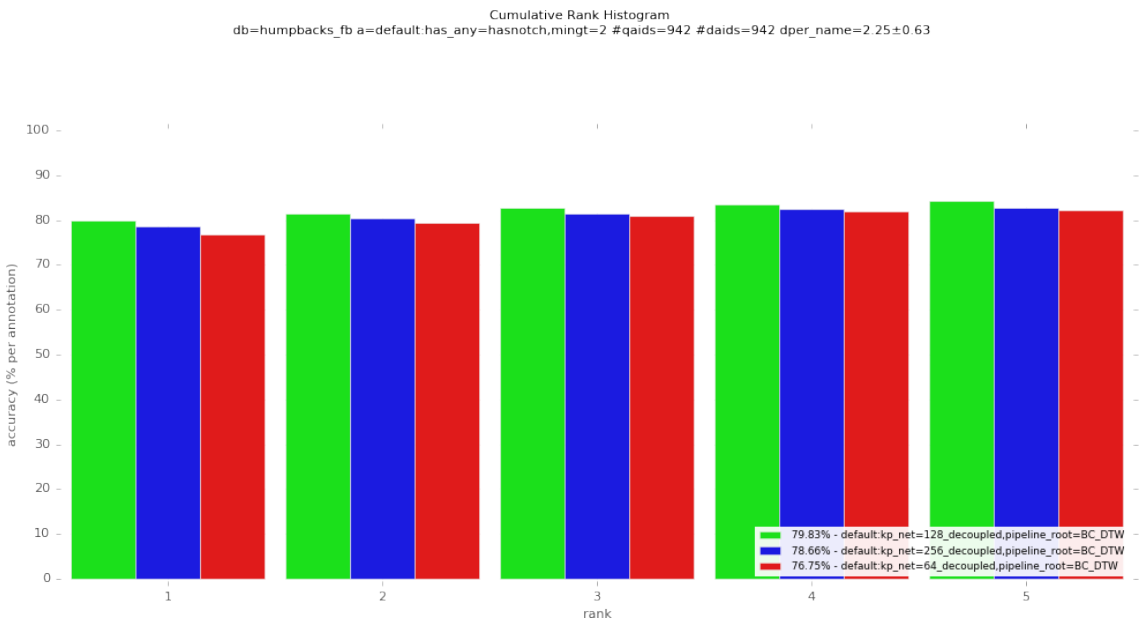


Figure 4.2: Varying Keypoint Image Size. While the 128×128 size network is better than the 256×256 network (green and blue respectively), we believe the difference between them to be insignificant. The 64×64 network (red) is clearly inferior however.

However, using the notch as the control point actually does reduce the accuracy more significantly for the predicted points than it does for the manually extracted points, which further implies that its predictions are not perfect.

4.2.1.2 Varying training image sizes

Intuitively, the bigger the image the better the prediction can be, but at the expense of requiring more parameters to handle the input. The primary difference between the networks that were trained to handle different size inputs is that, in order to ensure that all inputs to the final dense layers have the same spatial size (2×2) across the different networks, an extra convolutional and pooling layer is added.

The performance of networks trained and run on different sized images is given in Figure 4.2.

We trained each of these networks on separate splits of data, and due to the inherent stochasticity of the training it is difficult to ascertain whether or not the image size is a cause of performance issues.

4.2.1.3 STN

We also briefly experimented with an Spatial Transformer Network. This was largely motivated by the tendency of the keypoint extractor to do a terrible job of predicting fluke keypoints on flukes that did not 'fill' the image horizontally. Unfortunately, we could not get the STN to converge at a better accuracy than the standard keypoint extractor, even if we held its parameters fixed for a few training epochs. Usually, the STN would produce nonsensical transformations of the image.

4.2.2 Image Preprocessing

In this part we also detail some simple image preprocessing steps that we took (given the fluke keypoints) that greatly influenced the efficacy of the method. Ultimately we just ended up using a combination of cropping and resizing the image so as to normalize the length of the trailing edge, as detailed below.

We also note that we originally tried histogram equalization as part of the preprocessing pipeline, althoough it produced significantly worse trailing edges (and subsequently matching accuracy).

4.2.2.1 Cropping and Image Width

While, with dynamic time warping, we theoretically can match sequences of similar or different lengths, the distances are distorted by large differences in actual trailing edge length. Since we are only interested in the width of an image (assuming that the trailing edge is roughly horizontal in the image), we can get every trailing edge to have exactly some fixed length w by the following process

- Crop the image horizontally between the left and right columns found by the keypoint extraction process (or manually determined).
- Resize the cropped image to some fixed width w while preserving the aspect ratio, using Lanczos interpolation.

In this way, we standardize the trailing edge length so that image scale does not affect detection accuracy too much.

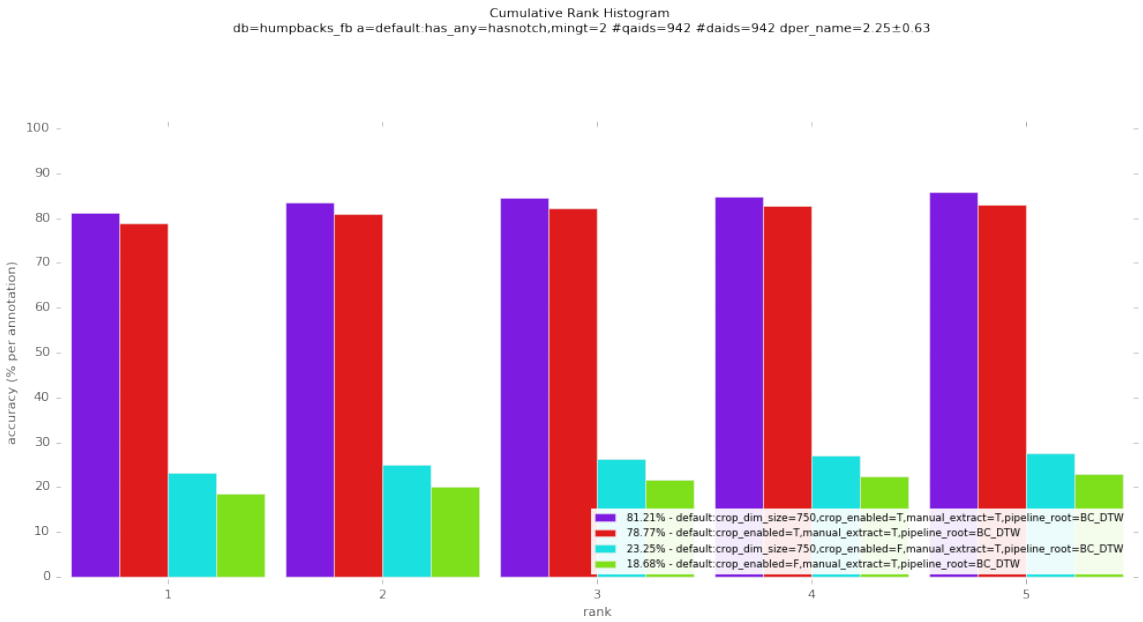


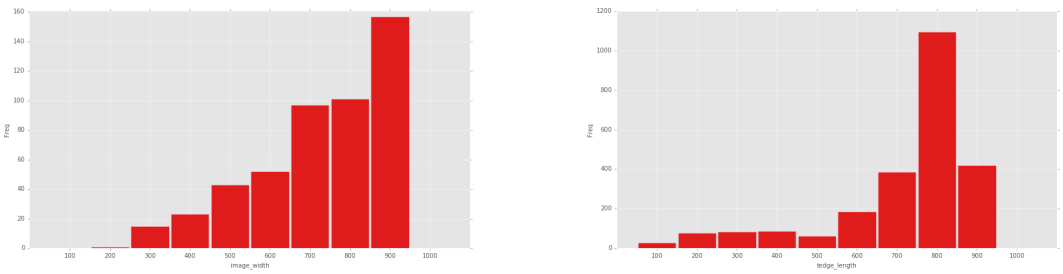
Figure 4.3: Varying Crop Strategy. Cropping images around the trailing edge and then resizing them proves to be very important, not doing so gives a very low accuracy. We can see in Figure **??** that there is a wide distribution of image sizes, which can hamper the effectiveness of DTW.

One major caveat with this process is of course that using the keypoint extractor's predictions can cause catastrophic failures in this process (e.g. the left and right points are nowhere near a fluke), however in practice we found that it works well enough – far better than the alternative.

We can see in Figure 4.3 that cropping and resizing is absolutely vital to the performance of our algorithm.

Trailing Edge Length Determining what w should be is not super obvious. Ideally, one would simply look at the mode or average post-crop width, and try to keep w around there, since intuitively we want to introduce as few interpolation artifacts from resizing as possible.

We can see in Figure 4.5 that the optimal post-crop w for this dataset (of the few we evaluated) is 750 pixels, althout 1000 pixels is on par. We select the former for efficiency's sake, as smaller trailing edges vastly improves the speed of the matching algorithm. Figure **??** shows the histogram of post-crop widths (i.e trailing edge lengths), showing a large concentration of mass around 800 pixels This



a **b**

Figure 4.4: Distribution of Image Widths. The image width distribution (left) is fairly wide, with most of the mass centered between 600 and 800.

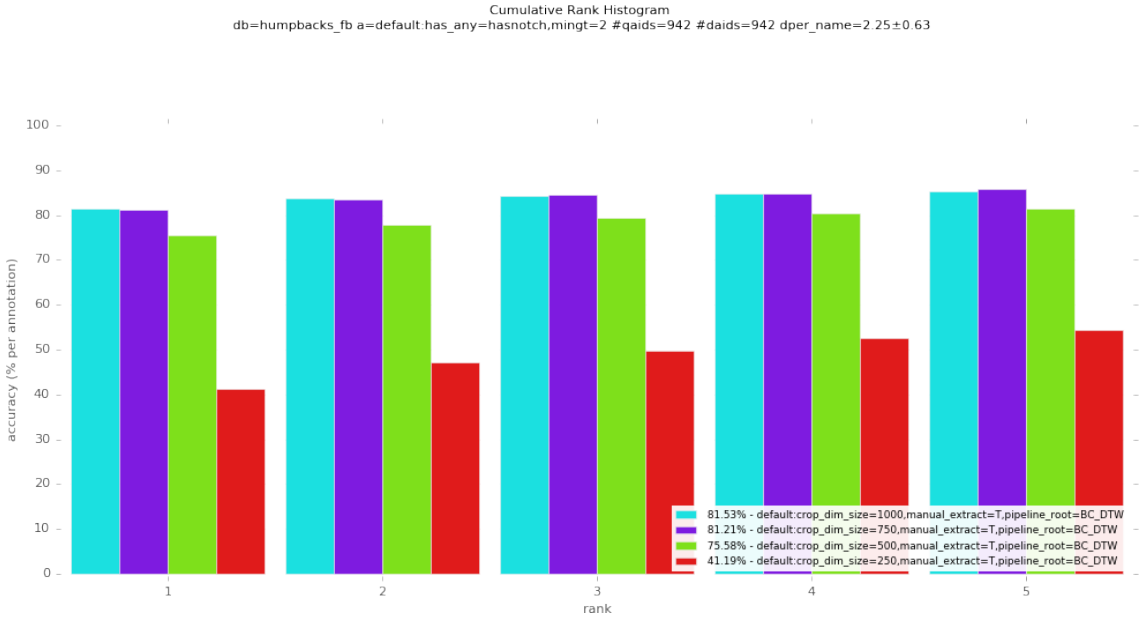


Figure 4.5: Varying Crop Size. Since we crop around the start and end points of the trailing edge, this crop size effectively controls the length of the extracted trailing edge. Note that we use the manually annotated points in this analysis to control for any issues with keypoint extraction.

confirms that the less the image has to be resized, the better the trailing edge.

4.2.3 Trailing Edge Extraction

One major result that we found was that, when using the averaging method to combine the trailing edge scores with N_y, having a robust trailing edge prediction

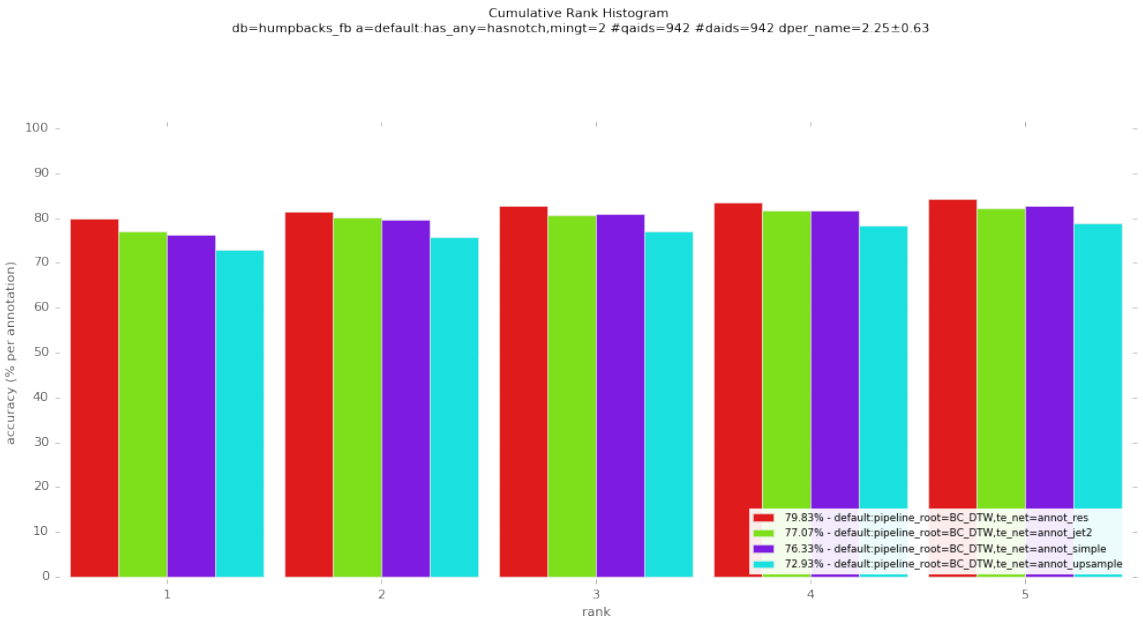


Figure 4.6: Trailing Edge Scorer Architectures. The highest performing trailing edge scorer (Residual) is shown in red, followed by Simple, Jet, and Upsample (in descending order of accuracy).

wasn't as important as having a detailed trailing edge.

4.2.3.1 Trailing Edge Scorer variations

The various trailing edge scorer architectures and their results on the task they were trained for is detailed in the previous chapter. In Figure 4.6 we present the actual matching accuracies that each one produced.

We can see that the detailed and higher quality trailing edges produced by the Residual network give a decent performance boost over the other networks. The only other network that beats not using the trailing edge scorer at all (i.e. $\beta = 0$, see the purple bar in Figure 4.7) is the Simple network.

We hypothesize that the Jet and Upsample architectures do poorly due to their inability to produce fine-grained trailing edges.

One caveat with using the Residual network is that, with 64 layers, it consumes a lot of GPU memory.

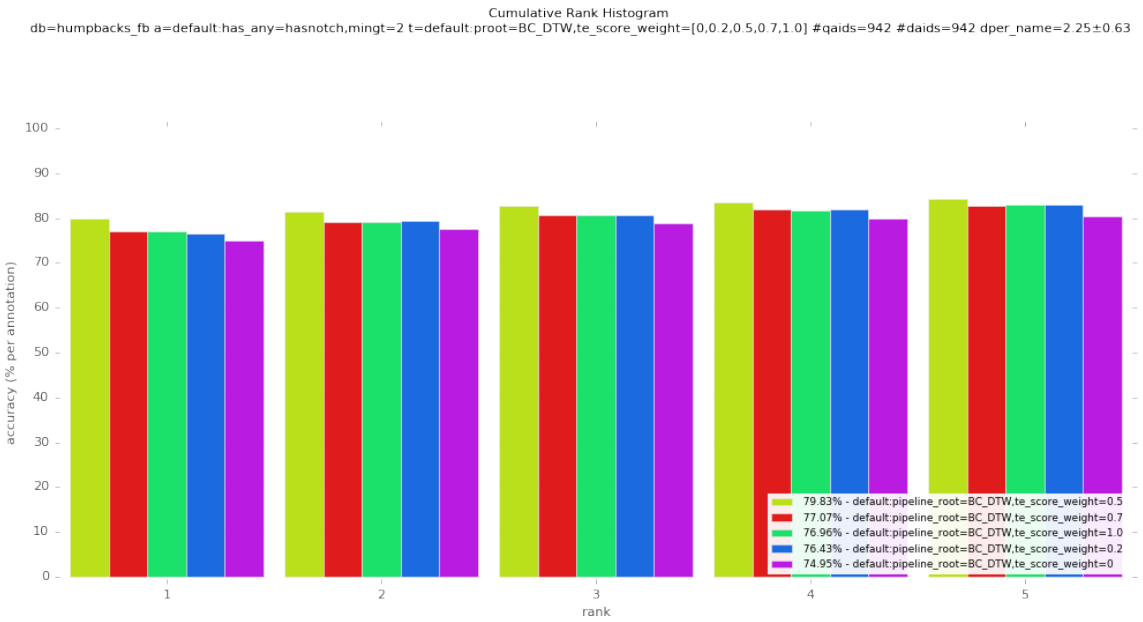


Figure 4.7: Varying β. It's clear from this that the optimal β is 0.5, although it is interesting to note that using only the network's trailing edge scores provides better accuracy than not using it at all.

4.2.3.2 Combining N_y and T_y

We only evaluate the 'average' method here, and show that the mixing parameter β has a significant effect on matching accuracy.

In Figure 4.7, we can see that simply a pixel-wise average of N_y and $(1 - T_y)$ (i.e. $\beta = 0.5$) produces the best results for the Residual network.

4.2.3.3 Number of neighbors in the extraction

The number of neighbors n effectively limits the slope of the trailing edge. We limit it to an odd number for convenience. On the one hand, a lower n can cause the trailing edge to be limited in vertical breadth, but does prevent it from going way off course. Despite this, with trailing edge scoring in place, it might be beneficial to increase n so as to avoid parts of the trailing edge that continually "max out" the number of neighbors.

Ultimately, we can see in Figure 4.8 that limiting the number of neighbors to the immediate neighborhood (i.e. $n = 3$) produces a significant boost over a larger neighborhood. While the trailing edges extracted with $n = 3$ can be somewhat

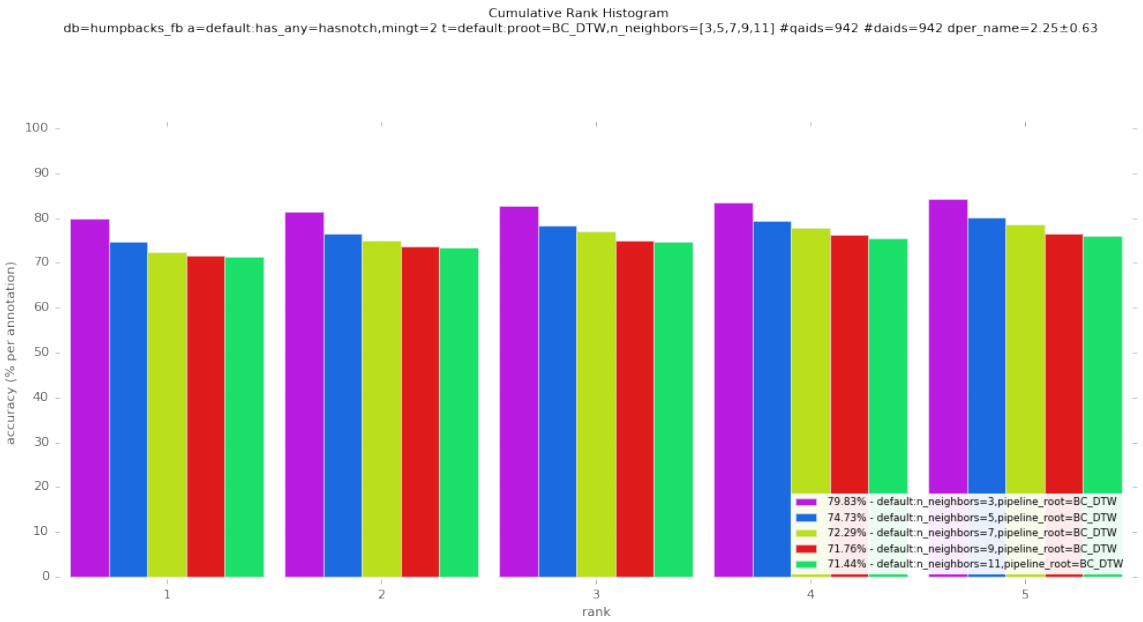


Figure 4.8: Varying n. This shows that the optimal neighborhood constraint is $n = 1$, despite qualitatively producing worse-looking trailing edges.

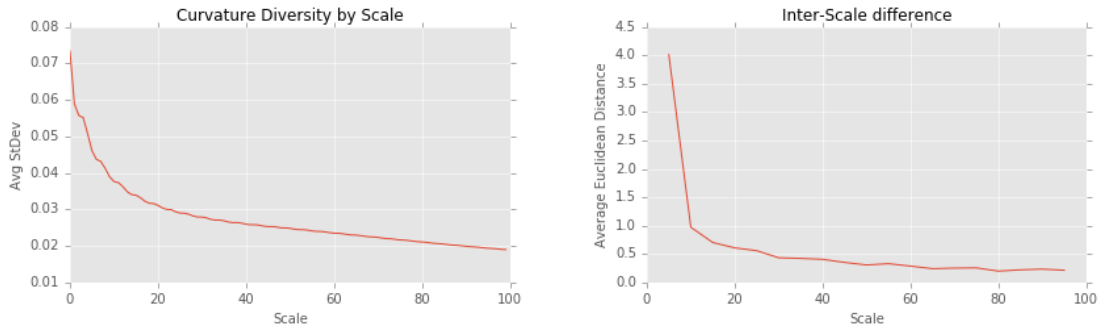
less detailed, they are less likely to go completely off course. Additionally, despite the trailing edges being less detailed, the extraction artifacts (i.e. the parts of the trailing edge that "max out") are produced in a way that is unique to the trailing edge, which may explain the accuracy boost.

4.2.4 Curvature Extraction

Curvature extraction is one of the least parameterized part of the process, however figuring out what the optimal scales to extract are and how many is non-trivial. One major point is that increasing the cardinality of S increases the time it takes to evaluate the dynamic time warping, and as such we try to keep it at a small value (i.e. $|S| = 4$).

4.2.4.1 Different scales

We can look at each scale as measuring the curvature of some percentage of the trailing edge around the given point. Intuitively, since the meaningful differences in trailing edge can be fairly small, it makes sense to measure a small curvature scale. Additionally, this gives the most diversity in the trailing edge, as larger curvatures



a **b**

Figure 4.9: Curvature Diversity. Left panel (a) shows the average standard deviation of the (fixed length) curvature at different scales. Right panel (b) shows the average Euclidean distance between successive scales of curvature

will change less and less from one point to the next.

However, larger curvatures are also less sensitive to noise, and thus small trailing edge extraction failures.

We find that (as shown in the left panel of Figure 4.9) as block curvature scale is increased, the diversity at any given point in the curvature (if it is of a fixed size) goes down drastically. As a result, we stick to the lower end of the scale, keeping the curvature scales measured below 15%.

We can also see on the right side of Figure 4.9 that successive curvature scales show bigger differences at lower scales than at higher scales, which reinforces this need, but that it's advantageous to make bigger jumps between scales to maximize diversity while minimizing computation time.

Based on the above, we evaluate scales that run from 1% to 10%, with varying levels of resolution. Overall we found that this does not have a major effect on accuracy, but that the scales [2%, 4%, 6%, 8%] provides marginally better accuracy than anything else that we tried. This is shown in Figure **??**

4.2.5 Dynamic Time Warp Matching

The main variants shown in this section are the different Sakoe-Chiba bound windows and the scale weighting term in the curvature distance function.

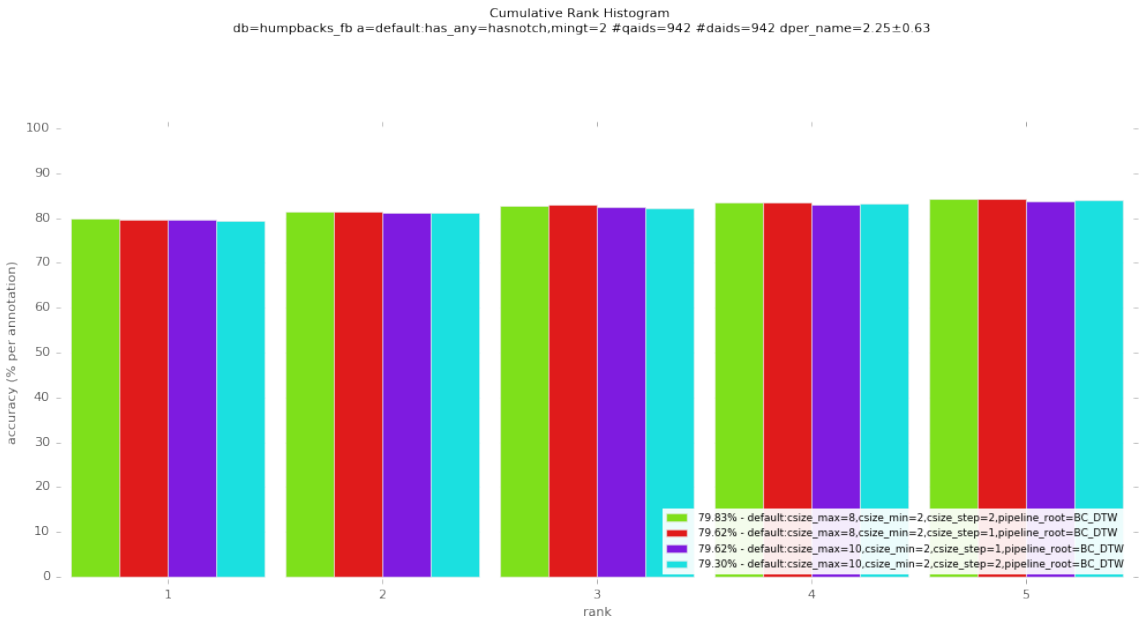


Figure 4.10: Varying Curvature Scales. The scales evaluated here are parameterized with a start, end, and step size. We always start at 2%, and vary the step size between 1 & 2 and the end between 8% & 10%.

4.2.5.1 Weighting the different scales

There are many ways to produce the weights s_w for curvatures, but intuitively there should be a monotonic relationship between curvature scale and importance.

We could specify this relationship as a ratio W of importance between successive pairs of curvatures (increasing in size). We parameterize that importance by giving each scale the weight $s_w = [W^i \forall i \in [0, |S|]$. In order to maintain this ratio without blowing up the distances, we also normalize so that $\sum s_w = 1$.

Figure 4.11 shows that despite this effort, it appears that equally weighting each curvature scale provides the best performance. Interestingly weighting larger scales lower (i.e. $W < 1$, which we evaluate with $W = 0.5$ (purple bar) in Figure 4.11) provides worse performance.

4.2.5.2 Window size

In Figure 4.12, we can see that if we decrease the window (i.e. the Sakoe-Chiba bound) size, at around 10% of the query trailing edge length we maintain the same accuracy as the full window (i.e. 100%), but below this accuracy is severely

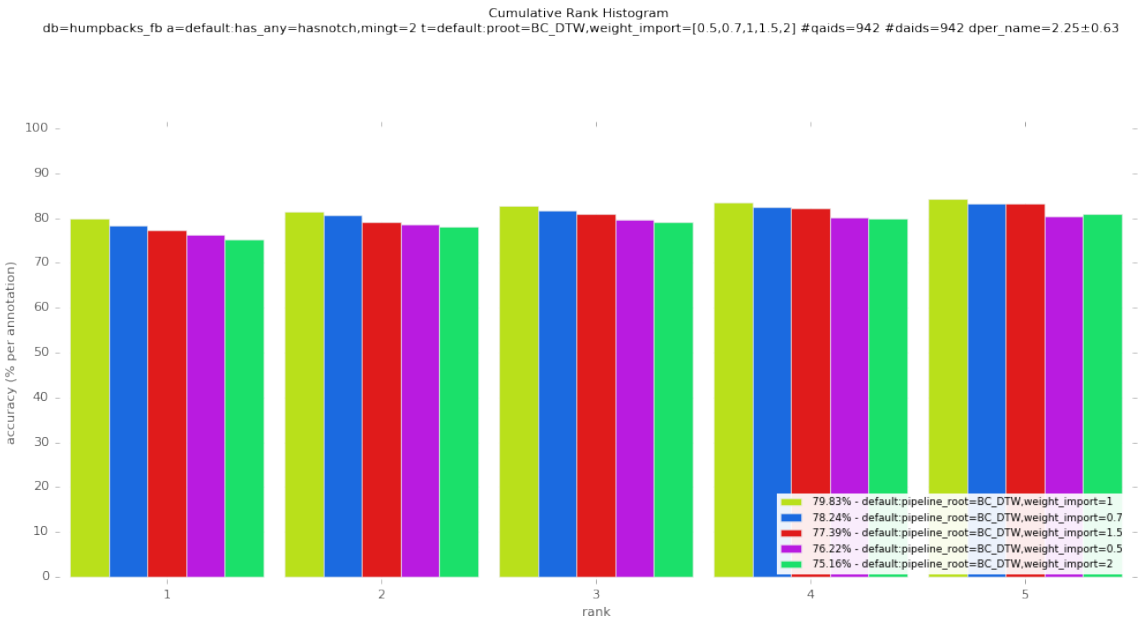


Figure 4.11: Varying s_w. The yellow bar shows $W = 1$, i.e. all curvatures weighted equally.

affected. We find that overall there is a 4× slow-down in wall-clock time on our testing machine when going from a window size of 10% to one of 100%. Thus, we use this value for the window size so as to minimize computation time while maintaining the total accuracy.

Additionally, while it's possible for gross mismatches to occur from there being no window boundary, we can see from Figure 4.12 that this does not pose a problem in our case.

4.2.5.3 Aggregating over multiple trailing edges per identity

Determining the identity of a given query image given distances to other images in the database is not entirely simple when there are multiple database images for a given individual. Essentially we need to transform these distances from query image to database image into distances from query to known individual. To do so we evaluate two options given a group of distances for an individual — either the average distance or the minimum distance.

We find that the average decision does slightly better, as shown in Figure 4.13.

This is unsurprising given that most of the individuals in the dataset only have

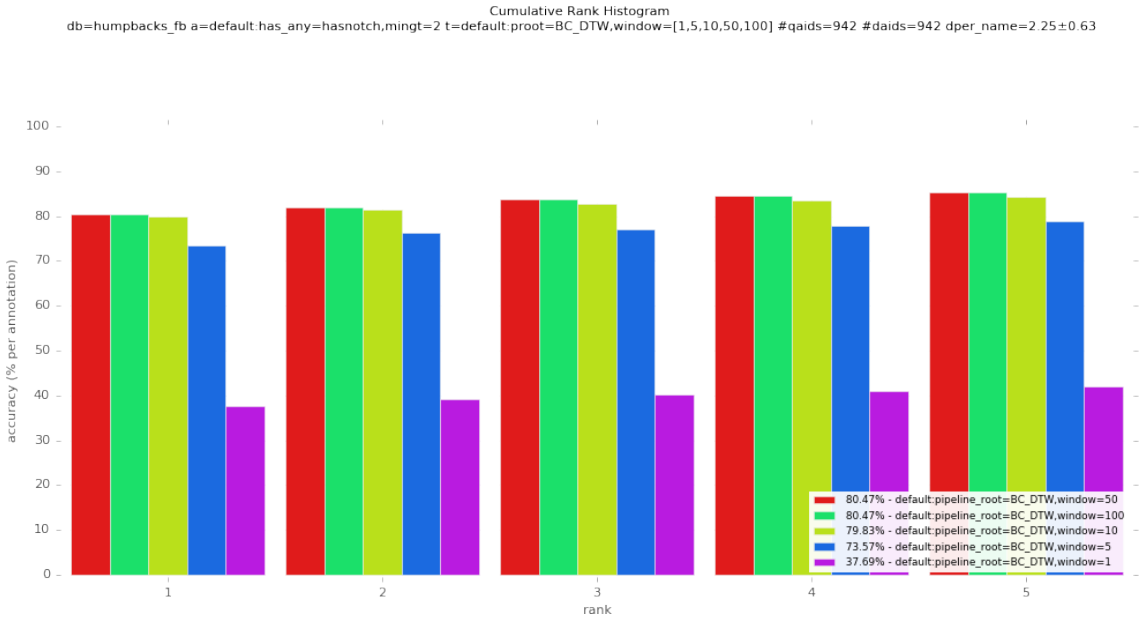


Figure 4.12: Varying Sakoe-Chiba bound

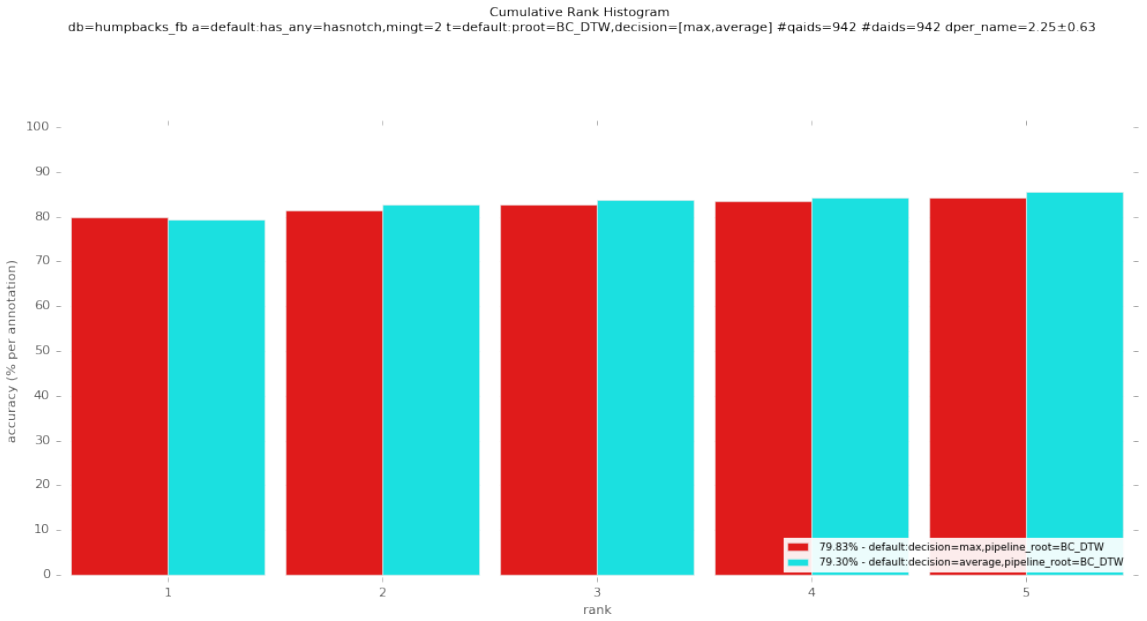


Figure 4.13: Varying Decision Criterion

two images associated, meaning that the average is equivalent to the minimum. That said, the minimal distance criterion is effectively the single-feature case of LNBNN used in Hotspotter [12] and the work of Hughes et al. [24].

4.3 In Combination with Hotspotter

By combining our method with Hotspotter — if we were able to automatically pick out which algorithm was right for a given ranking — we can achieve 93% top-1 accuracy on the Flukebook dataset. This implies that these methods are complementary to each other, and should be used in combination. However, being able to automatically decide which method to use is non-trivial.

(TODO: this)

4.3.1 Failure cases

(TODO: this)

4.3.2 Characterization of when to use which method

From an intuitive standpoint, it appears that Hotspotter cannot find effective keypoints from trailing edges, which hampers its ability to handle flukes which do not have an apparent pattern.

We hypothesize that this is because the trailing edge — while a distinctive feature — inevitably shares a region with an oceanic background. Since this oceanic background changes from image to image, it cannot verify salient keypoints as matches. As a result, flukes which have little to no internal texture are nearly impossible for Hotspotter to match.

On the other hand, when the trailing edge is unclear or significantly distorted in the image, our method struggles to find an appropriate match. In these cases, Hotspotter can provide a good match, although if the fluke is additionally untextured we have an issue.

(TODO: Figures showing this)

We were unable to find a single heuristic that correlates with our method failing to find a match. Interestingly, we find that smoothness of the trailing edge, characterized by the standard deviation of the trailing edge slope, does not correlate to match failure, implying that smooth trailing edges are about as easy to match as rough, distinctive trailing edges.

CHAPTER 5
Discussion

5.1 Issues with the Proposed Method

The primary issue that we encountered is that the distance between a given pair of trailing edges is very unstable in terms of determining if they match or not (see Figure **??**)). A side effect of this is that small changes in the trailing edge can lead to matching failures, and in the worst case "push down" the correct match significantly. This "push down" effect severely hampers the effectiveness of our method, as in the use case that we target a human operator verifying a match would ideally only have to look at the top-k flukes for some small value k. However, we find that in many cases the correct match is far down the list, and there is no reasonably value k that all matches are within.

There are several other issues, primarily having to do with the stability and generalization capability of the convolutional networks used for fluke keypoint predictions and trailing edge scores. For the former network, ideally the keypoints could be predicted regardless of our assumption on fluke position and orientation, although it is clear to us that this does not happen. We believe that this is primarily due to the lack of training data that defies these assumptions, although it is also possible that it is beyond the capacity of the models we trained.

Having the fluke horizontally oriented and flat to the camera does not only help the keypoint extraction, but also avoids an obscured trailing edge due to out-of-plane rotations (as seen in Figures 3.2 and 1.3). However imposing this requirement significantly complicates and restricts the actual photography of these flukes.

The network that we trained for scoring trailing edges was ultimately somewhat disappointing. For the most part, these networks were very good at predicting a trailing edge where a strong gradient was present, however this does not significantly improve on the base trailing edge extraction algorithm. Part of the problem for this is similar in nature to the problem for training the fluke keypoint network, i.e. that a large majority of the dataset represent "easy" cases, making it hard to

train the network to handle hard cases. We did use data augmentation to help rectify this bias, but found that it had limited effectiveness. It is possible that more sophisticated data augmentation could help, but primarily we think that spending more time annotating and creating a trailing edge scoring dataset would be ideal.

5.2 Future work

There is a lot of work that can be done based on this method. The immediate focus is to achieve the theoretical 93% accuracy by accurately choosing to use Hotspotter or our method for any given query and result. This would result in a more general method that could be robust to obscured trailing edges and some out-of-plane rotations.

One part of this identification pipeline that we mostly ignored is a detection and orientation step. While orientation is not necessarily important for the curvature measure (as it is rotation invariant), it is important for the trailing edge extraction. Additionally, being able to detect and crop the fluke automatically from an image would give a much more robust and flexible system. We did not explore this as most of the images in the Flukebook dataset came pre-cropped around the fluke, as well as rotated such that the major axis of the trailing edge was horizontal. These conditions both obviates the need for such a system and make it difficult to train a detection model.

Extracting the trailing edge is currently done with an unsophisticated and restricted algorithm, which can be improved. There are several ways to improve this algorithm, but the main drawbacks are its lack of rotation invariance and the inability to easily draw the second or third best trailing edge. On top of that, the trailing edge scoring networks could be better evaluated with more annotated trailing edges, the creation of which is non-trivial.

There is a lot to be done with the matching algorithm itself, namely in ensuring that it is more tolerant to significant deformations in the trailing edge. One major paradigm that we do not explore in this work is that of extracting and matching multiple features per trailing edge, much in the same way that Hotspotter operates. Hughes et al. [24] take this approach for a more general contour matching system. It

would be possible to do something similar but making use of the curvature measures.

5.3 Conclusion

In this thesis we have presented a novel, fully-automated method for photo-identifying humpback flukes that achieves a high top-1 ranking accuracy on a relatively large dataset. This method extracts fine grained trailing edge contours from images of flukes and identifies individuals based on sequence properties of the contour curvature.

REFERENCES

[1] Y. Akagi, R. Furukawa, R. Sagawa, K. Ogawara, and H. Kawasaki. A facial motion tracking and transfer method based on a key point detection. 2013.

[2] A. A. Amini, S. Tehrani, and T. E. Weymouth. Using dynamic programming for minimizing the energy of active contours in the presence of hard constraints. In *Computer Vision., Second International Conference on*, pages 95–99. IEEE, 1988.

[3] S. Avidan and A. Shamir. Seam carving for content-aware image resizing. *ACM Trans. Graph.*, 26(3), July 2007.

[4] C. Baker, A. Perry, J. Bannister, M. Weinrich, R. B. Abernethy, J. Calambokidis, J. Lien, R. Lambertsen, J. U. Ramirez, and O. Vasquez. Abundant mitochondrial dna variation and world-wide population structure in humpback whales. *Proceedings of the National Academy of Sciences*, 90(17):8239–8243, 1993.

[5] B. Beekmans, H. Whitehead, R. Huele, L. Steiner, and A. G. Steenbeek. Comparison of two computer-assisted photo-identification methods applied to sperm whales (physeter macrocephalus). *Aquatic Mammals*, 31(2):243, 2005.

[6] S. Berretti, B. B. Amor, M. Daoudi, and A. Del Bimbo. 3d facial expression recognition using sift descriptors of automatically detected keypoints. *The Visual Computer*, 27(11):1021–1036, 2011.

[7] A. L. Blackmer, S. K. Anderson, and M. T. Weinrich. Temporal variability in features used to photo-identify humpback whales (megaptera novaeangliae). *Marine Mammal Science*, 16(2):338–354, 2000.

[8] T. Branch. Humpback whale abundance south of 60 s from three complete circumpolar sets of surveys. *Journal of Cetacean Research and Management (special issue)*, 3:53–69, 2011.

[9] J. Calambokidis, E. A. Falcone, T. J. Quinn, A. M. Burdin, P. Clapham, J. Ford, C. Gabriele, R. LeDuc, D. Mattila, L. Rojas-Bracho, et al. *SPLASH: Structure of populations, levels of abundance and status of humpback whales in the North Pacific*. Cascadia Research, 2008.

[10] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Semantic image segmentation with deep convolutional nets and fully connected crfs. *arXiv preprint arXiv:1412.7062*, 2014.

[11] D. Ciresan, A. Giusti, L. M. Gambardella, and J. Schmidhuber. Deep neural networks segment neuronal membranes in electron microscopy images. In *Advances in neural information processing systems*, pages 2843–2851, 2012.

[12] J. P. Crall, C. V. Stewart, T. Y. Berger-Wolf, D. I. Rubenstein, and S. R. Sundaresan. HotSpotter - patterned species instance recognition. In *Applications of Computer Vision (WACV), 2013 IEEE Workshop on*, pages 230–237. IEEE.

[13] F. C. Crow. Summed-area tables for texture mapping. *ACM SIGGRAPH computer graphics*, 18(3):207–212, 1984.

[14] H. Fan, Z. Cao, Y. Jiang, Q. Yin, and C. Doudou. Learning deep face representation. *arXiv preprint arXiv:1403.2802*, 2014.

[15] P. Fischer and T. Brox. Image descriptors based on curvature histograms. In *Pattern Recognition*, pages 239–249. Springer, 2014.

[16] K. Fukushima. Neural network model for a mechanism of pattern recognition unaffected by shift in position- neocognitron. *ELECTRON. & COMMUN. JAPAN*, 62(10):11–18, 1979.

[17] R. Hadsell, S. Chopra, and Y. LeCun. Dimensionality reduction by learning an invariant mapping. In *Computer vision and pattern recognition, 2006 IEEE computer society conference on*, volume 2, pages 1735–1742. IEEE, 2006.

[18] B. Hariharan, P. Arbeláez, R. Girshick, and J. Malik. Simultaneous detection and segmentation. In *Computer vision–ECCV 2014*, pages 297–312. Springer, 2014.

[19] d. J. Hartog and R. Reijns. I3S Contour MANUAL, 2013.

[20] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

[21] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[22] 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. Technical report, Technical Report 07-49, University of Massachusetts, Amherst, 2007.

[23] R. Huele, H. U. De Haes, J. Ciano, and J. Gordon. Finding similar trailing edges in large collections of photographs of sperm whales. *Journal of Cetacean Research and Management*, 2(3):173–176, 2000.

[24] B. Hughes and T. Burghardt. Automated identification of individual great white sharks from unrestricted fin imagery. In M. W. J. Xianghua Xie and G. K. L. Tam, editors, *Proceedings of the British Machine Vision Conference (BMVC)*, pages 92.1–92.14. BMVA Press, September 2015.

[25] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[26] M. Jaderberg, K. Simonyan, A. Zisserman, et al. Spatial transformer networks. In *Advances in Neural Information Processing Systems*, pages 2008–2016, 2015.

[27] M. Kass, A. Witkin, and D. Terzopoulos. Snakes: Active contour models. *International journal of computer vision*, 1(4):321–331, 1988.

[28] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[29] E. Kniest, D. Burns, and P. Harrison. Fluke matcher: A computer-aided matching system for humpback whale (megaptera novaeangliae) flukes. *Marine Mammal Science*, 26(3):744–756, 2010.

[30] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[31] N. Kumar, P. N. Belhumeur, A. Biswas, D. W. Jacobs, W. J. Kress, I. C. Lopez, and J. V. Soares. Leafsnap: A computer vision system for automatic plant species identification. In *Computer Vision–ECCV 2012*, pages 502–516. Springer, 2012.

[32] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[33] D. Lemire. Faster retrieval with a two-pass dynamic-time-warping lower bound. *Pattern recognition*, 42(9):2169–2180, 2009.

[34] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3431–3440, 2015.

[35] S. A. Mizroch, J. A. Beard, and M. Lynde. Computer assisted photo-identification of humpback whales. *Report of the International Whaling Commission*, 12:63–70, 1990.

[36] A. Monroy, A. Eigenstetter, and B. Ommer. Beyond straight linesobject detection using curvature. In *Image Processing (ICIP), 2011 18th IEEE International Conference on*, pages 3561–3564. IEEE, 2011.

[37] M. Müller. *Information retrieval for music and motion*, volume 2. Springer, 2007.

[38] M. E. Munich and P. Perona. Continuous dynamic time warping for translation-invariant curve alignment with applications to signature verification. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, volume 1, pages 108–115. IEEE, 1999.

[39] F. Ning, D. Delhomme, Y. LeCun, F. Piano, L. Bottou, and P. E. Barbano. Toward automatic phenotyping of developing embryos from videos. *Image Processing, IEEE Transactions on*, 14(9):1360–1371, 2005.

[40] D. Nouri. Using convolutional neural nets to detect facial keypoints tutorial, 2014.

[41] O. M. Parkhi, A. Vedaldi, and A. Zisserman. Deep face recognition. *Proceedings of the British Machine Vision*, 1(3):6, 2015.

[42] H. Pottmann, J. Wallner, Q.-x. Huang, and Y.-l. Yang. Integral invariants for robust geometry processing. 2007.

[43] Y. Qiao and M. Yasuhara. Affine invariant dynamic time warping and its application to online rotated handwriting recognition. In *Pattern Recognition, 2006. ICPR 2006. 18th International Conference on*, volume 2, pages 905–908. IEEE, 2006.

[44] H. Sakoe and S. Chiba. Dynamic programming algorithm optimization for spoken word recognition. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 26(1):43–49, 1978.

[45] S. Salvador and P. Chan. Fastdtw: Toward accurate dynamic time. *Warping in Linear Time and Space*, 2007.

[46] A. M. Saxe, J. L. McClelland, and S. Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*, 2013.

[47] F. Schroff, D. Kalenichenko, and J. Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 815–823, 2015.

[48] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*, 2013.

[49] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[50] I. Sobel and G. Feldman. A 3x3 Isotropic Gradient Operator for Image Processing. Never published but presented at a talk at the Stanford Artificial Project, 1968.

[51] Y. Sun, X. Wang, and X. Tang. Deep convolutional network cascade for facial point detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3476–3483, 2013.

[52] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.

[53] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf. Deepface: Closing the gap to human-level performance in face verification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1701–1708, 2014.

[54] H. Whitehead. Computer assisted individual identification of sperm whale flukes. *Reports of the International Whaling Commission*, 12:71–77, 1990.

APPENDIX