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 LaTeX template. 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 [2, 1]. 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. It then uses this information to extract a detailed trailing edge and its 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 [1] dataset consisting of about 400 identified individuals. We also show that in combination with Hotspotter [3], a general pattern based matching algorithm, 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 inference time.

CHAPTER 1
Introduction

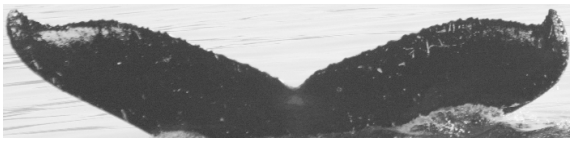
1.1 Humpback Whales

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 [5]. 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 endeavors. 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 1.1). The internal texture is a more obvious choice for identification, as it is distinctive 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), and in some cases when the fluke is backlit, the constrast of the internal texture is affected. Additionally, the work of Blackmer et al. [2] 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 is vulnerable to being obscured by out of plane rotations (see Figure 1.3), and as such are optimally photographed when the fluke is "flat" to the photographer. This also generally means that crowd-sourcing fluke pictures for their trailing edge can be difficult, although if their quality is measurable then it might be possible to extract a quality picture from a "burst" series of pictures.

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 [1] 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.

1.2 Current Identification Methods

Computer-assisted photo-identification of humpback whale flukes has been attempted since the early 90s [6]. While early efforts mostly relied on a manual description of the fluke that would then be matched against other stored descriptions [6, 7], later efforts have involved matching flukes based on automated analysis of both the internal texture and trailing edge [8, 9, 10].

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 I^3S contour system [10], the user must input start and end points on the query trailing edge, after which its contour is extracted. This trailing edge is

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

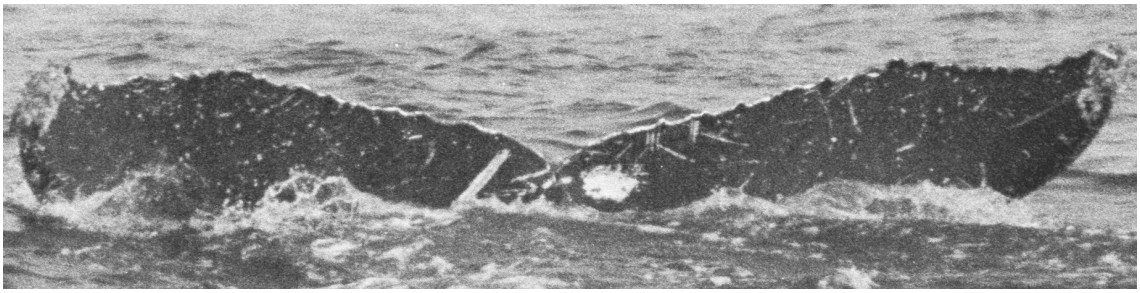
then resized and aligned so that it can be compared with absolute difference against the database trailing edges. It also compares a set of possible shifts, rotations, and scales of the query trailing edge to account for these differences. At the time of writing no published results on this system applied to humpback whales could be found.

Automatically identifying humpback whales by their entire trailing edge contour is done experimentally in Hughes et al. [8], using a technique 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 for a much smaller dataset of humpback flukes than the one evaluated here. A more detailed comparison to this method is beyond the scope of this thesis, but a possible avenue for future work.

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 [11, 12, 7] — with varying levels of manual effort. In Whitehead's work [7], 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. [11] uses a semi-automatic extraction of sperm whale trailing edge, and then applies wavelet transformations which are

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.

cross-correlated to determine a similarity measurement.

An investigation of the above methods for sperm whale identification is carried out in the work of Beekmans et al. [12], 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 method only achieves about 65% top-1 accuracy.

1.2.2 Based on General Fluke Appearance

The primary paradigm 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. [6] and the Flukematcher [9] of Kniest et al[1].

In Mizroch et al., information about the fluke is manually catalogued and

[1]Flukematcher also uses information about the trailing edge, but is focused on the internal fluke texture

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 [9], control points are manually annotated which allow the program to automatically find pigmentation patterns in the fluke and align accordingly. Optionally, distinctive fluke patterns can also be selected and annotated by the user. A variety of heuristic features are then extracted, which are matched using a variety of similarity measures. This method has achieved 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 (see Figure 1.4), and for each point on this contour curvature is measured at multiple scales. Once these curvatures are extracted for each fluke photograph, the algorithm computes the distance between a query fluke and possible identities by aligning its contour curvature against contour curvatures in the database and computing distance with dynamic time warping. This distance is used to give a ranking over possible matching images and subsequently their identities. For identities with multiple images, we use the smallest distance. The query fluke is then given the identity of the database fluke with the lowest distance. This method is based on both traditional edge detection and curvature techniques, modern machine learning, and classical fast sequence comparison.

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

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

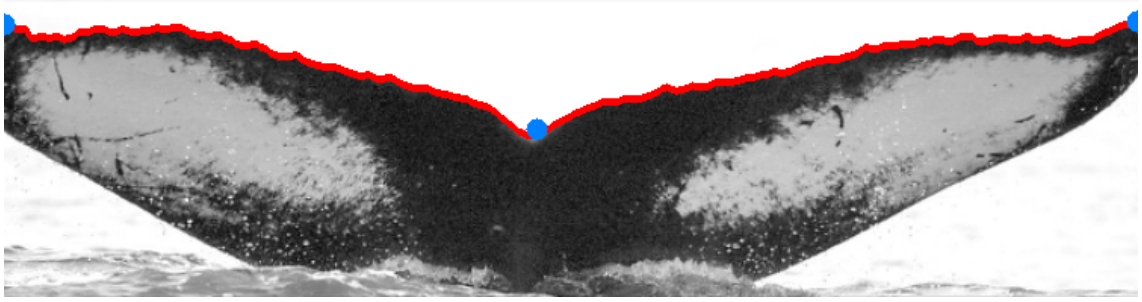


Figure 1.4: Example Keypoint and Trailing Edge Annotation. This image shows a typical set of fluke keypoints and trailing edge extracted by our algorithm.

method that is a powerful matching algorithm for several species [3]. Hotspotter extracts keypoints with SIFT descriptors [13] 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 Hotspotter's efficacy when applied to humpback whale flukes, and the results are presented in Chapter 4.

1.4 The Dataset

The main dataset used and evaluated in this work is a subset of the dataset collected by the SPLASH project [1]. It consists of about 1400 identified photographs spread over about 860 identified individuals. The identities of these flukes were determined by experts. 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. All images shown in this thesis come from this dataset. This dataset additionally provides fluke keypoint annotations for each image.

Additionally, an external dataset of unidentified (but similarly 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 described in Chapter 3, which also includes a brief description of some alternate methods that did not work as well. Chapter 4 presents the results, showing overall algorithm performance and examining the effect of a number of configuration changes. Chapter 4 also describes 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.

We 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. The primary contributions of this work are both the individual algorithms for extracting fluke keypoints and trailing edges, and more importantly the combination of all the components into an effective overall algorithm.

CHAPTER 2
Background

In this chapter, we provide background information on the algorithms on which our trailing edge identifier is based. We describe some of the applications and variants of deep convolutional networks which were used for 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 short 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 been used to substantially advance the state of the art results in several challenging computer vision tasks, including general image classification [14, 15], image segmentation [16, 17] and human face identification [18, 19].

The essential idea of a convolutional network is that it we can use the gradient of an error signal to learn series of convolution kernels separated by nonlinear activation functions. These networks are a type of neural network, and are used in image data as their convolutional structure can better capture spatial relationships. Convolutional networks were introduced nearly 40 years ago by Fukushima in [20]. The current incarnation of these networks can be traced back to the seminal work of LeCun et al. [21]. Modern convolutional networks follow a common framework of using Rectified Linear Units (ReLUs) as activation functions, and Dropout [22] layers after fully connected layers for regularization. Additionally, the convolutional kernels used are commonly small square kernels with "same" padding[3] alternated with 2× downsampling layers (specifically max pooling layers) [24, 25, 14].

The convolutional networks used in this thesis follow the above framework,

[3]Essentially zero-padding such that the output image has the same shape as the input image. For more information on this see [23]

and also use batch normalization [26] at every layer. Every network in this thesis uses orthogonal initialization [27] at all layers to help ensure gradient flow.

2.1.1 Facial Keypoint Prediction

Facial keypoints are coordinates on an image of a (human) face that locate the nose, eyes, mouth, and other salient features. They are commonly used in facial identification pipelines [28], as well as in motion capture [29] and expression recognition [30]. We draw on recent work for using convolutional networks for facial keypoint prediction [31, 32]. 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 work.

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 arbitrarily sized predictions are required (e.g. for semantic segmentation) from 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 useful for segmentation tasks [33, 34, 35], as this process allows for (downsampled) predictions that adhere spatially to the input image. By then upsampling and combining different stages of prediction, the authors in [16] 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 a long standing problem. A primary method for getting coherent contours from image information is the active contour (or snakes) method [36, 37]. 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 continuous function to represent the contour. 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 [38]. 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 dynamic programming 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 significant amount of work has been done on using curvature information for detection [39] and classification [40, 41]. 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 normal to the gradient at each point in an image [40]. The curvature of a contour

is then expressed by using the points on that contour. 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 more quickly.

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 [43]. 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 linear scan of each database fluke to identify a query fluke a bit inefficient.

There are several variants on DTW that give faster speeds [44, 45], however we only use the Sakoe-Chiba bound [43], which constrains the neighborhood in which points can be matched. This bound 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 [46], 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 identification algorithm pipeline. We also briefly discuss some alternative approaches that we found to be less successful than our chosen algorithms.

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 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. The nature of the problem (and the Flukebook dataset) makes these assumptions reasonable. It would also be possible to add a detection and orientation prediction 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.

This network cannot use arbitrarily sized images, so the first step of the keypoint extraction pipeline is to resize the image to 128×128 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 re-scaled back up to the original image size by multiplying with the original image height and width. An example

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.

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 kernels (starting with 8 kernels). This is somewhat similar to VGG-16 [24], 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 and its corresponding points are resized to a fixed width while maintaining the aspect ratio, and the resultant size is recorded as $\vec{s}$. This is done to somewhat normalize the scale of objects in the image, so that large differences in original image size don't produce large differences in training loss magnitude. Each image is then further re-scaled to the network size (128×128)[4], and then the corresponding targets are re-scaled to the range $[0 - 1]$. The loss function that we use for each fluke keypoint 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 $\vec{s}$. Thus, we have the scaled Euclidean loss function SE

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

Where $\vec{t}$ and $\vec{p}$ are the target and predicted coordinates respectively [5]. We then average this loss function over the keypoints that the network outputs.

The networks are trained for 1000 epochs with α set to 2e$-$2 using the Adam [47] 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 15 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 subsequently matching accuracy) was not severely affected by switching between ground truth points and estimated points as our experimental results will show.

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

[4] While this is oftentimes a 10$\times$ downsample for the image, we found this size input to give the best accuracy for memory tradeoff

[5] $\odot$ denotes element-wise 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.

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 [48] to try and handle these 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

We find that as trailing edges are often darker than their backgrounds — and that the fluke is usually oriented such that the trailing edge is on top — the trailing edge often corresponds to a highly negative image gradient. Thus, the base algorithm for extracting the trailing edge is constructed to find a path from the left

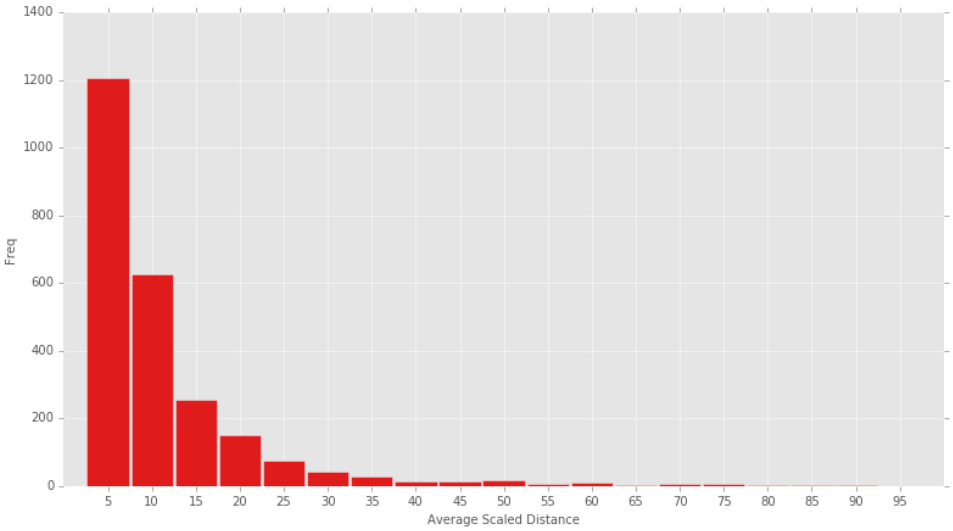


Figure 3.3: Histogram of Keypoint Distances. This is a histogram of the average distance from predicted keypoints to annotated keypoints on the testing set for the keypoint extraction network. The vast majority of keypoints are predicted within 10 pixels of the true keypoints.

to right tip that minimizes the vertical gradient information of the image (denoted as I_y). We compute I_y using a vertically oriented 5×5 Sobel kernel [49]. I_y is then normalized with min-max scaling, giving N_y as

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

Given N_y, we extract the trailing edge using a dynamic programming algorithm that finds the minimal gradient path between the left and right tips of the fluke (points denoted s and e respectively) given a neighborhood constraint. This algorithm builds a cost matrix C by choosing for each pixel a "neighbor" from the previous column with minimal cost, and then adding the pixel's cost. It also maintains a backtrace matrix B that then maintains the chosen neighbor, and once the cost matrix is filled we construct the path by working backwards from e. This provides an optimal path while also limiting the length of the trailing edge to $|s_x - e_x|$,

which simplifies some of the later steps in our method. C is constructed with the following recursive update rule by scanning each pixel (i, j) in N_y starting from the column $s_x + 1$ and ending at the column e_x:

$$C(x,y) = \begin{cases} 0 & x < 0 \\ \infty & y < 0 \text{ or } y > h \\ \min_{y-\frac{n}{2} \leq y_c \leq y+\frac{n}{2}} C(x-1, y_c) + N_y(x,y) & \text{else} \end{cases} \tag{3.3}$$

This algorithm runs in $O(whn)$ time, where w and h are the width and height of the image respectively, and n specifies the neighborhood constraint. We default n to 3, meaning that each pixel considers its immediate "neighbors" 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).

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

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

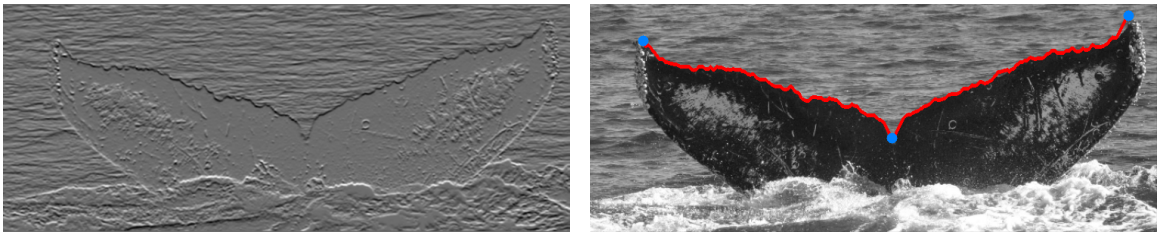
$$TE^i = (TE_x^{i-1} - 1, (TE^{i-1})_y + B(TE_x^{i-1}, TE_y^{i-1})) \tag{3.5}$$

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



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.

cost. We initially did the same "forcing" for the point denoting the bottom of the notch, although we find that this lowers matching accuracy. 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 perhaps make better choices in trailing edge extraction. An example of this at work is shown in Figure 4.8. 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 suited to a fully convolutional network performing a binary classification.

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.

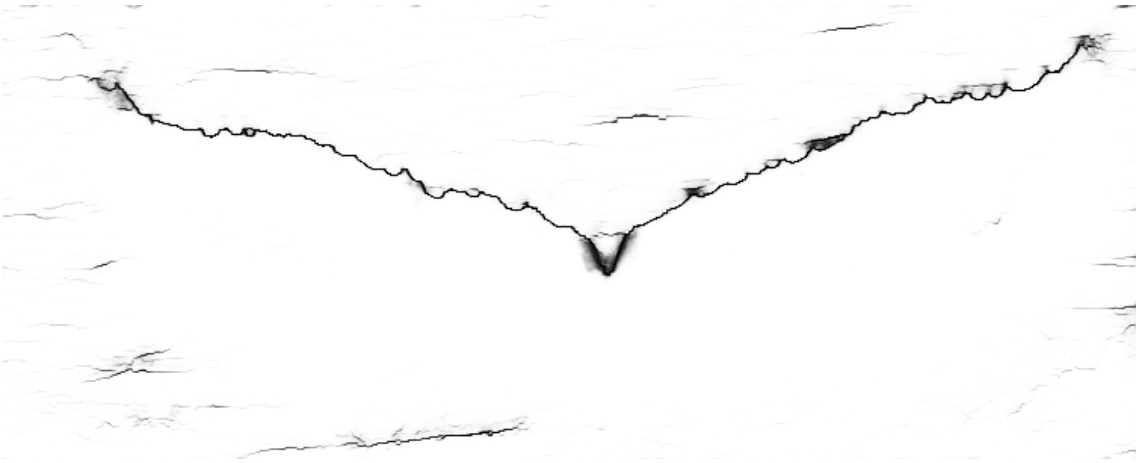
The dataset used to train these networks is sampled from trailing edges extracted using the basic method from section 3.1.2, followed by manual adjustments to fix common issues encountered. These manual adjustments often included manually adding "control points" along the edge, which often fixed major 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.

Given these manually verified 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). Then, each pixel in this patch is labeled as either background or trailing edge based on the annotated trailing edge in the original image. The images that this dataset were extracted from were generally 960 pixels in width (with the trailing edges being only slightly smaller), so 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

a Original Image



b Trailing Edge Score

Figure 3.5: Example Trailing Edge Score. This trailing edge score was produced by the Residual architecture. A darker pixel means that it is more likely to be part of the trailing edge.

convolutional network architecture is its receptive field[6]. 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 massively increasing the depth of the network, we must downsample, which makes the output less fine-grained.

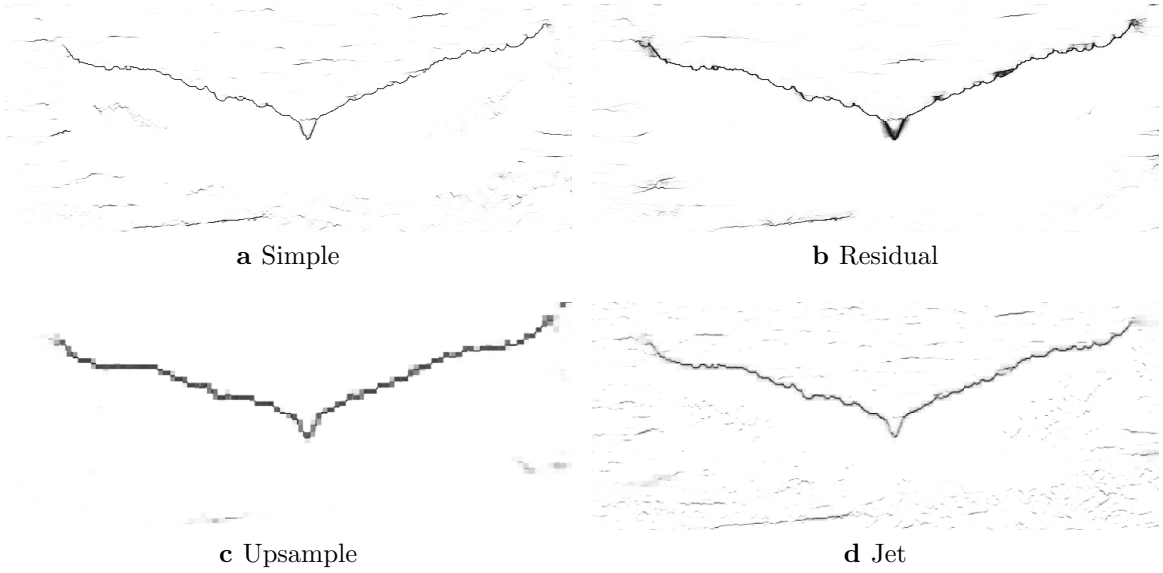
[6]The receptive field of a convolutional network is, at a high level, the region of input that affects the output at a given layer

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) produce 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, which doubles the receptive field. 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 $8\times$ 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 [16].The goal of this is to increase the receptive field of the network's output layer, although the predictions it makes are very "blocky" (see Figure 3.6c).

Jet Following the deep-jet architecture from [16], 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. Ideally this allows the network to take better advantage of both a deep representation of its input and a lower level spatial layout that allows it to make finer predictions. This architecture gives far more fine-grained predictions than the Upsample network while taking advantage of the increased receptive field size, without increasing the number of parameters greatly.

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. We can still increase the receptive field by stacking small convolutions, but the rate of increase is lower, meaning that more layers are needed. We could



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. A darker pixel is predicted to be part of the trailing edge by the network.

simply 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. [50], which introduce residual connections to address this problem. These connections are simply shortcut connections from one layer to another, which 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. We also find that it performs far better than an equivalent network without shortcut connections.

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

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 network 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 proceeds as described in the previous section.

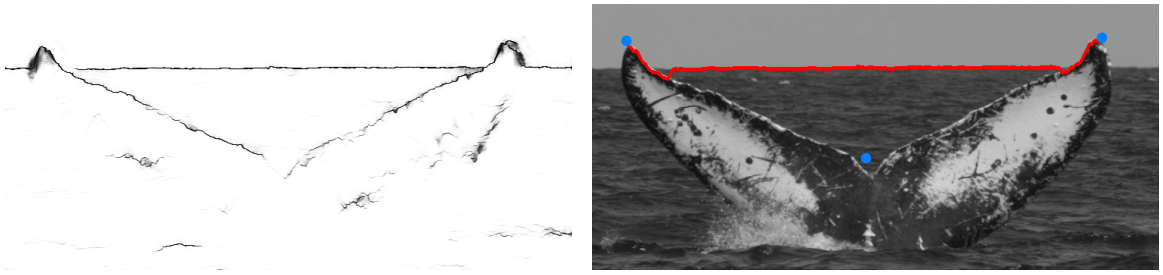
Another variant on combining T_p and N_y that we tried was to dilate the trailing edge predictions and then forbid the trailing edge from going outside of those predictions. An inherent difficulty with this approach is that in order for it to work, we need to have a guarantee from the trailing edge scorer that it will not produce major gaps in its predictions. If there are such gaps, then the trailing edge will not be extractable with this method. 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 and $l2$ regularization using a decay of 1e$-$4. We used the Adam optimizer [47] (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. This provides a much better trailing edge extraction even if it can reduce precision.

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



a Trailing Edge Score **b** Extracted Trailing Edge

Figure 3.7: Trailing Edge Scoring Failure. Unfortunately even the Residual architecture is still imperfect, and can lead to catastrophic trailing edge failures like this one. However this is a rare case.

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, by random application of 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 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. It is difficult to measure how successful this was without further annotated data, however in terms of overall matching accuracy we observed a small improvement using a network trained with this augmentation scheme. 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 precision and recall of each network, as well as the intersection over union (IoU) score commonly used for evaluating segmentation methods [16].

The poor performance of the Upsample architecture is also reflected in the match accuracy of trailing edges extracted from its scores (see Figure 4.6). With the exception of the Upsample architecture, the performance differences of the other architectures are insignificant, and we find that they all perform comparably.

Unfortunately it does not appear that these models learned to reliably find

Table 3.1: Evaluating Trailing Edge Scoring Architectures. 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.

	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

only the trailing edge, as can be seen in 3.7 However we observe that unless a false trailing edge prediction allows for a continuous path from the left to right tip, it is rare for this failed prediction to cause the trailing edge extraction algorithm to go severely "off-course".

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 a query fluke image by ranking possible identities from the set of database flukes images according to their distance. When multiple images are associated with an individual in the database, we use the one with minimal distance to the query image. The distance function we use is dynamic time warping over sequences of block curvature measurements, using Euclidean distance as a local distance function between individual elements of each curvature.

This provides a sequential matching algorithm, and its complexity is proportional to the size of the database. With larger humpback datasets this will become a problem, and we believe an avenue for future work is to evaluate the multitude of dynamic time warping optimizations in the literature [51, 44, 45].

3.2.1 Curvature Measurement

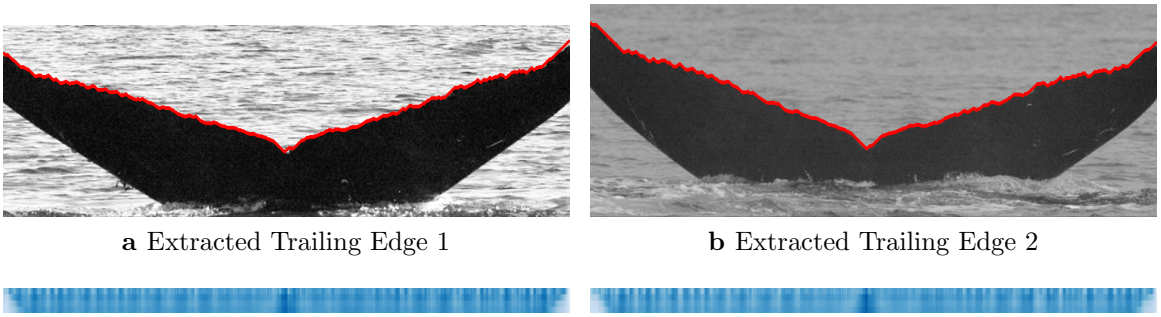
For matching, we first extract the curvature from the trailing edge. Given the trailing edge as a sequence of coordinates into the original image, we construct a zero-image I_0 of shape similar to the original image. For each pixel in the trailing edge all pixels below it in its column, we set their values in I_0 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 implied assumption 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 [52] 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 size $m \times m$ pixels centered on each point, and measure the percentage of that square that is within the filled in trailing edge. The value m is the scale at which we measure curvature, and is computed as a percentage of the trailing edge length. We compute multiple scales M of curvature for each trailing edge. This curvature measurement is carried out by computing $BC_m(x, y)$ for each (x, y) coordinate in the trailing edge.

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

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 $M = [2\%, 4\%, 6\%, 8\%]$ (as percentages of the trailing edge length)



a Extracted Trailing Edge 1 **b** Extracted Trailing Edge 2

c Trailing Edge Curvature 1 **d** Trailing Edge Curvature 2

Figure 3.8: Trailing Edge Curvature. The top images are of the same individual, and the bottom visualizations show the corresponding curvatures for their extracted trailing edges. Each row in the curvature visualization is a curvature scale, increasing from top to bottom. A darker blue implies a "valley" in the trailing edge (i.e. a higher curvature value), whereas lighter blue implies a "peak" (resp. a lower curvature value).

work well for our purposes. We then treat this curvature measurement as a $|M| \times l$ matrix BC, where l is the length of the trailing edge. Two example curvatures (with their corresponding trailing edges) are given in Figure 3.8. 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

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

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

Where $\vec{c}$ is a vector of the curvatures at different scales for a point. Initially there was also a weighting term that would weight each curvature scale differently, however we found that weighting each curvature equally was the best option. We then take the value of $C(l_1 - 1, l_2 - 1)$ as the distance between the two curvatures[7].

Additionally, we impose the Sakoe-Chiba [43] locality constraint T (set as a percentage of l_1) so that for each element i in BC_1, we only consider the range over elements j of BC_2 $j \in [min(i - T, 0), max(i + T, l_2)]$. This essentially provides a bound over the possible matches between points on the trailing edge, preventing (for example) the third element of BC_1 from matching with the second to 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. With this bound, this algorithm's complexity essentially goes from $O(l_1 l_2)$ to $O(l_1 T)$. For all of our experiments T is set to 10% unless specified otherwise, which appears to minimize the time taken for each comparison while preserving the overall accuracy of the algorithm (see Figure 4.11).

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 [53].

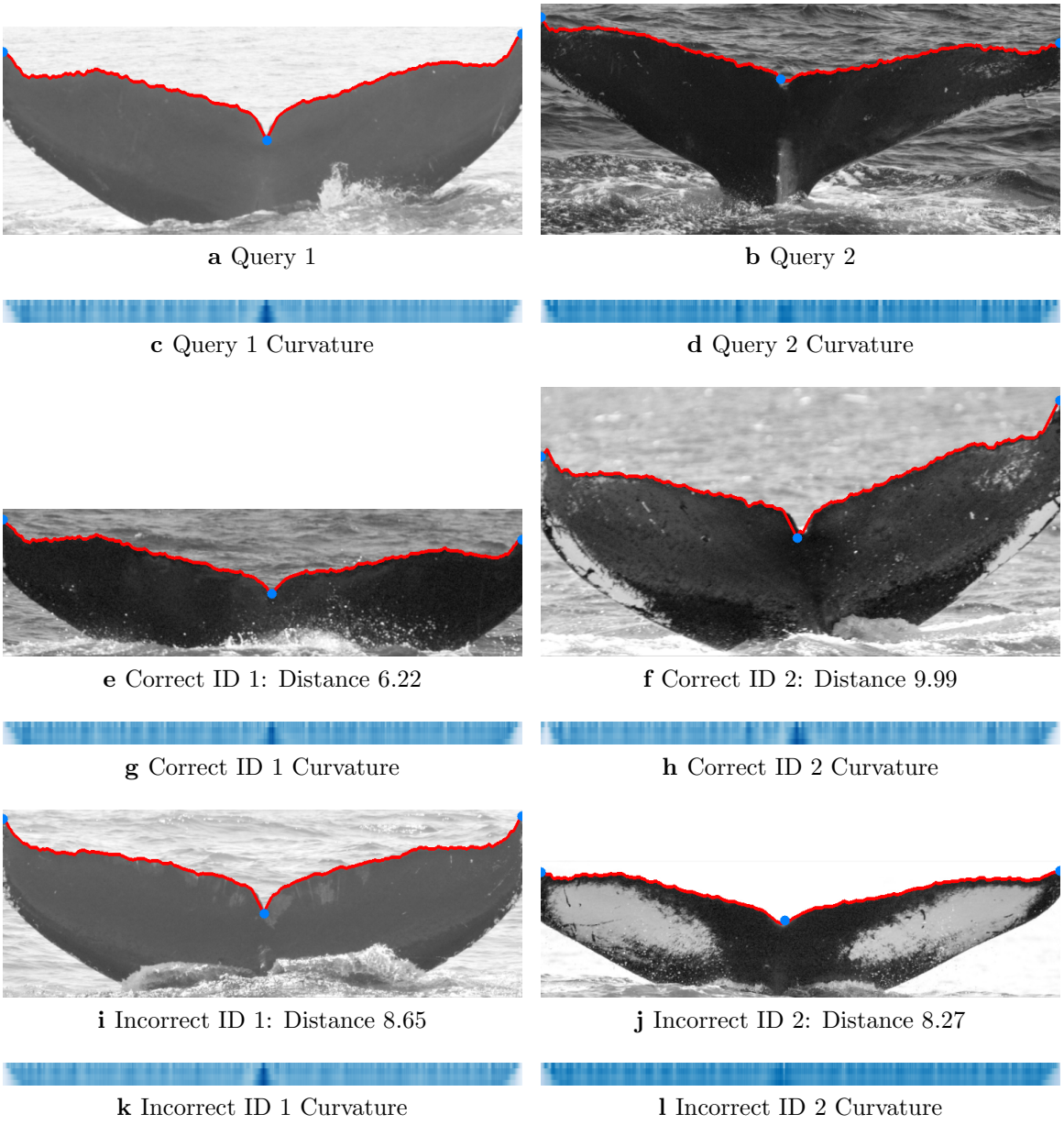
3.3 Alternative Approaches

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 that doing so when comparing curvature was often unnecessary (due to the invariances to rotation and translation), 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

[7]For the same reasons that $C(0, 0) = 0$ — this enforces that the end of each trailing edge aligns



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

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 [54]. 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 similar to the approach taken in Leafsnap [41]. We found that for our task even high resolution histograms did not provide enough detail to match the trailing edges properly.

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 k-dimensional vector, much like the work done for face recognition in Schroff et al. [19] and Parkhi et al. [55]. Most of the previous literature on this technique is applied to larger datasets such as Labeled Faces in the Wild [56], 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), but found that even a severely overfit convolutional network could only achieve less than half the top-1 accuracy on its training set that our algorithm is able to achieve. However, it was able to determine triplets rather accurately. We tried both triplet loss (a modified version of the one detailed in [19]) and contrastive loss [57] 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. We go on to discuss the performance of our method when used in combination with Hotspotter.

4.1 Main method

The main method we settled on achieves an 80% top-1 accuracy on the Flukebook dataset — meaning that for 80% of the query images, the correct identity is ranked first. The figures that we show in this section give the accuracy up to top-5 cumulatively. In general we find that relative accuracies between configurations 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.

- For every image, we predict keypoints by resizing a copy of the image to 128×128 and running it through the keypoint predictor.
- Every image is then cropped between the predicted left and right tips, and resized to the same width (750 pixels) while maintaining aspect ratio.
- We do not use the bottom of the notch as a control point for trailing edge extraction.
- We set the number of neighbors n used in trailing edge extraction to 3.
- We use the Residual architecture for scoring the trailing edge.
- The trailing edge score is averaged with the normalized gradient (i.e. $\beta = 0.5$).
- We use $M = [2\%, 4\%, 6\%, 8\%]$ for our curvature scales.

Table 4.1: Itemized Running Time. This table provides the average time taken for each operation that constitutes our algorithm on a single image. All computations were done on the subset of Flukebook that we used for evaluation (942 images).

Stage	Keypoint Pred.	TE Scoring	TE Extraction	Block Curv.	Pairwise DTW
Time (ms)	8.8	39.2	11.6	2.4	1.2

4.2 Implementation

All experiments in this thesis were carried out on a custom-built machine with a 6-core Intel i7 clocked at 3.5GHz, 64GB of RAM, and two nVidia GeForce GTX Titan X GPUs (each with 12GB of RAM). We make heavy use of numpy [58], Theano[8] [59, 60], and Lasagne [61] for implementing our convolutional networks. While most of the implementation is done in Python, we use C++ with Eigen [62] for some of the core algorithms, namely trailing edge extraction, block curvature computation, and dynamic time warping.

4.3 Running Time

4.3.1 Extracting Trailing Edges

Table 4.1 shows the times for each of these four stages. There are four major stages in extracting a trailing edge that take a significant amount of time. The first step is to predicting the fluke keypoints, the speed of which is largely dependent on the GPU used to run the convolutional network[9]. In order to predict the keypoints faster we can "batch" the images before sending them to the GPU, which reduces communication overhead and better takes advantage of the GPU's massive parallelization capabilities. The next step that takes a considerable amount of time is the trailing edge scoring. This particular step is dependent on the size of the image, and is generally slow as the network is applied multiple times throughout the image. Since each image is of a slightly different size we do not batch them, although this could be done with careful implementation. For this analysis we only look at the Residual architecture, which is the slowest architecture evaluated. Trailing edge ex-

[8]This allows for seamless switching between GPU and CPU computation

[9]A CPU could also be used but we do not test this here, and it would be significantly slower

traction and block curvature computation take considerably less time, making the total time to go from image to curvature sequence 62ms per image.

Notably these computations were performed on our machine which has a considerably powerful GPU for its day. As such, we would not expect the convolutional networks (particularly the trailing edge scorer) to perform similarly on a less powerful GPU. These steps are of course be cached and reused for every image, so our matching algorithm's speed is independent of these times.

4.3.2 Matching Trailing Edges

We use a sequential search with no lower bounds to find a match for a given query in this work. As such, for a given query our algorithm scales linearly in the size of the dataset that the query is to be matched against. Given that the time for a single pairwise comparison between two computed curvatures (with our selected crop size and Sakoe-Chiba bound) is 1.2ms (Table 4.1), with our database of 942 images it takes just over one second to find a matching fluke. Since we are doing a one-to-one comparison, our identification algorithm is embarrassingly parallel, providing an easy target for speed improvements.

4.4 Configuration Options

4.4.1 Variability in Matching Score

We find that a major issue with our method is the lack of strong correlation between the distance between two trailing edges and whether or not they belong to the same individual. We can see this in Figure 4.1, which shows that the scores[10] between query trailing edges and their corresponding ground-truth trailing edge overlaps heavily with the distance between these queries and unrelated trailing edges. The result of this is that small changes in the query and database trailing edges can (in some cases) cause the ranking of a ground-truth trailing edge to change significantly. While this effect is rare given all of the queries, we find that it is the cause of many discrepancies in matching accuracy between different configurations. With this in mind, we discuss the difference in matching accuracy only when it is

[10]The score is computed as $e^{\frac{-\text{distance}}{50}}$ for implementation reasons

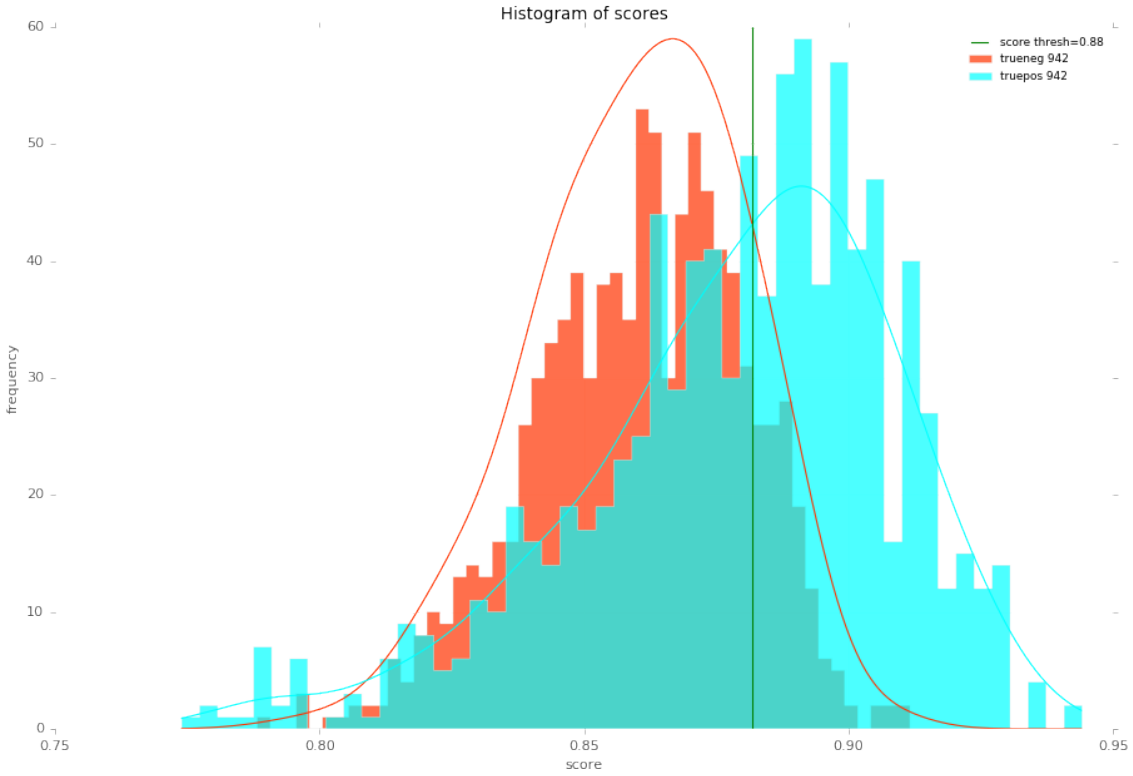


Figure 4.1: Score Separability Histogram. The blue bars in this figure represent true matches, and the red bars represent false matches. The line at score = 0.88 represents the optimal threshold at which to accept a match, although we can see it is not perfect.

a result of significant discrepancies — meaning that the score between query and ground-truth trailing edge changes by more than $\epsilon = 0.02$ for a significant percentage of the match failures when two configuration options are compared. We chose this value based on the score separation in Figure 4.1.

4.4.2 Effectiveness of Keypoint Extractor

In order to test the effectiveness of the trained fluke keypoint predictor, we give a comparison of matching accuracy against manual annotations of fluke keypoints[11] in Figure 4.2. 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, despite being on average 5 to 10 pixels off. Additionally, we found that only a small percentage of the match failures caused by automatic keypoint prediction were a

[11] Provided with the Flukebook dataset

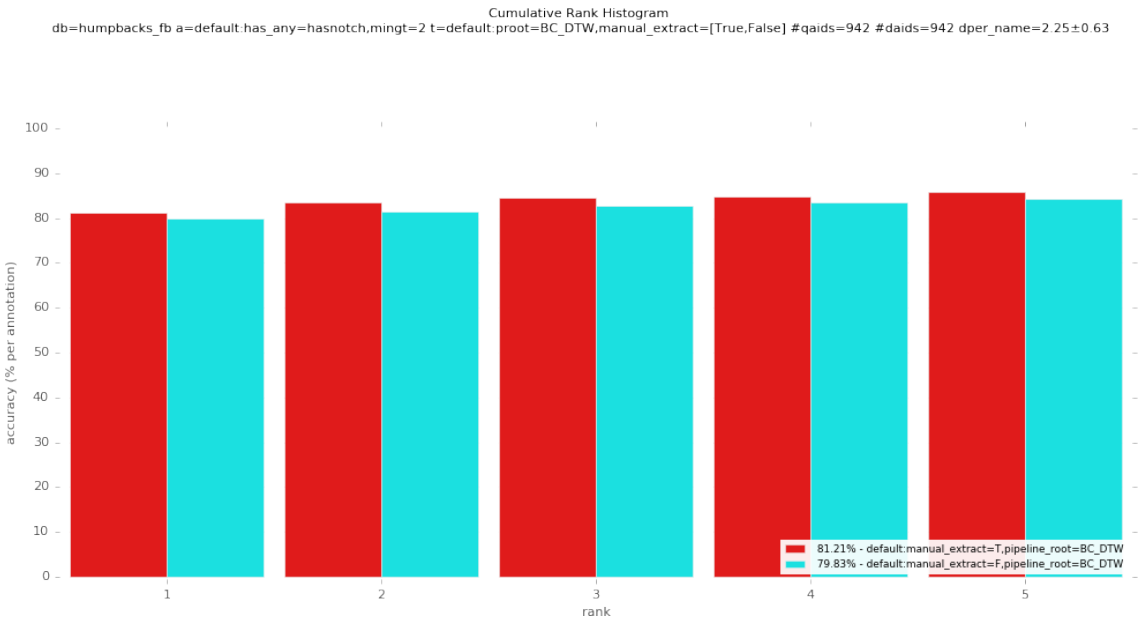


Figure 4.2: Varying Manual Extraction. There is a small difference in matching accuracy between using the manually annotated points (red) provided for this dataset versus the keypoint extractor's predicted points (cyan). The bottom of the notch keypoint is not used in these evaluations.

result of differences in score above ϵ.

4.4.3 Cropping Width

With dynamic time warping, we theoretically can match trailing edges of arbitrary lengths — however the distances can be distorted by large differences in actual trailing edge length. For this reason we fix the length of the trailing edges. Since we are only interested in the width of an image (because of the way the trailing edge extraction algorithm works), 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.

In this way, we standardize the trailing edge length so that differences in image size do not affect detection accuracy. One major caveat with this process is of course

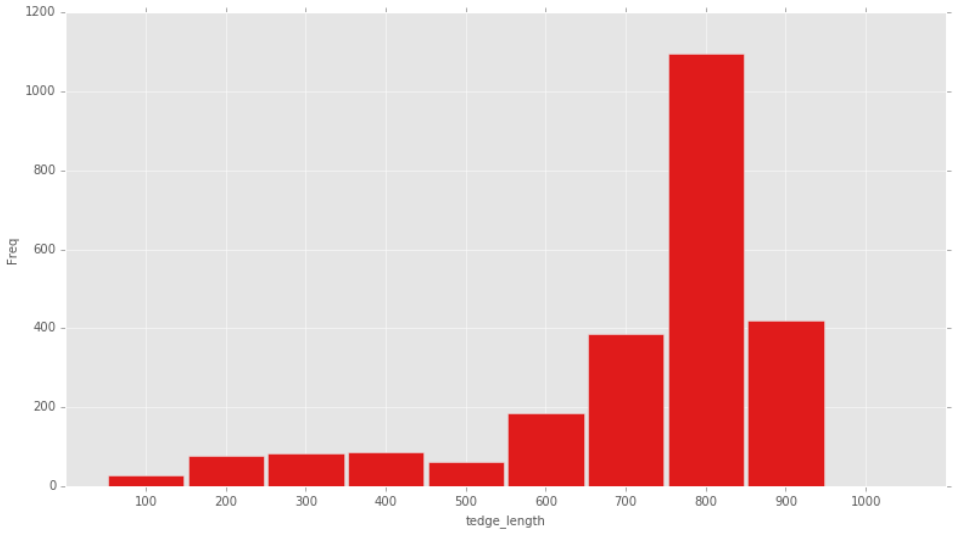


Figure 4.3: Distribution of Unresized Trailing Edge Lengths. This shows a significant distribution of trailing edges centered around a width of 800 pixels.

that using the keypoint extractor's predictions can cause catastrophic failures in this process (e.g. if the left and right points are nowhere near a fluke), however in practice we found that this is not an issue.

Ideally, we would choose a w that minimizes the interpolation artifacts that result from making big changes in image size. Figure 4.3 shows the histogram of post-crop widths (i.e trailing edge lengths) from unresized images in the Flukebook dataset, showing a large concentration of mass around 800 pixels. Subsequently, Figure 4.4 shows that $w = 750$ performs well, although $w = 1000$ is on par. We select the former for efficiency's sake, as smaller trailing edges vastly improves the speed of the matching algorithm. This provides evidence for the hypothesis that the less the image has to be resized, the better the trailing edge.

4.4.4 Trailing Edge Scorer Architecture

The various trailing edge scorer architectures and their results on the task they were trained for is detailed in the previous chapter (see Table 3.1). In Figure 4.5

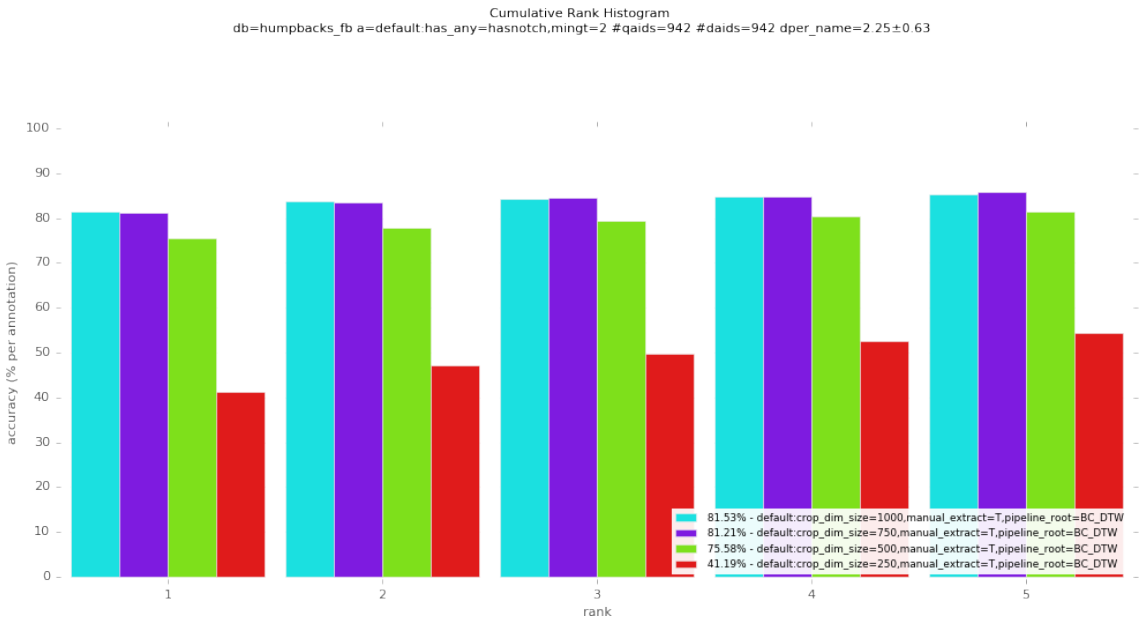


Figure 4.4: Varying w. Note that we use the manually annotated points in this analysis to control for any issues with keypoint extraction.

we present the actual matching accuracies that each one produced with the mixing parameter $\beta = 0.5$. 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, however this performance boost appears to consist mostly of insignificant changes in query to ground truth distance.

A better evaluation of each trailing edge scoring architecture is given in Figure 4.6 where we only use the trailing edge scorer outputs to extract the trailing edge (i.e. $\beta = 1.0$). Unsurprisingly, since the Upsample network gives blocky trailing edges (see Figure 3.6c), it performs very poorly. One of the benefits we observed with the Upsample network is that it does not make the same amount of spurious trailing edge predictions that the other networks make. However we can see that this is not as helpful as simply producing more fine-grained predictions.

One caveat with using the Residual network is that, with 64 layers, it consumes a lot of GPU memory to make a prediction. This is currently an implementation issue, however the machine that we use for these experiments has the capacity to run the Residual network. However, given that each network performs comparably, the Simple network is a good choice in general.

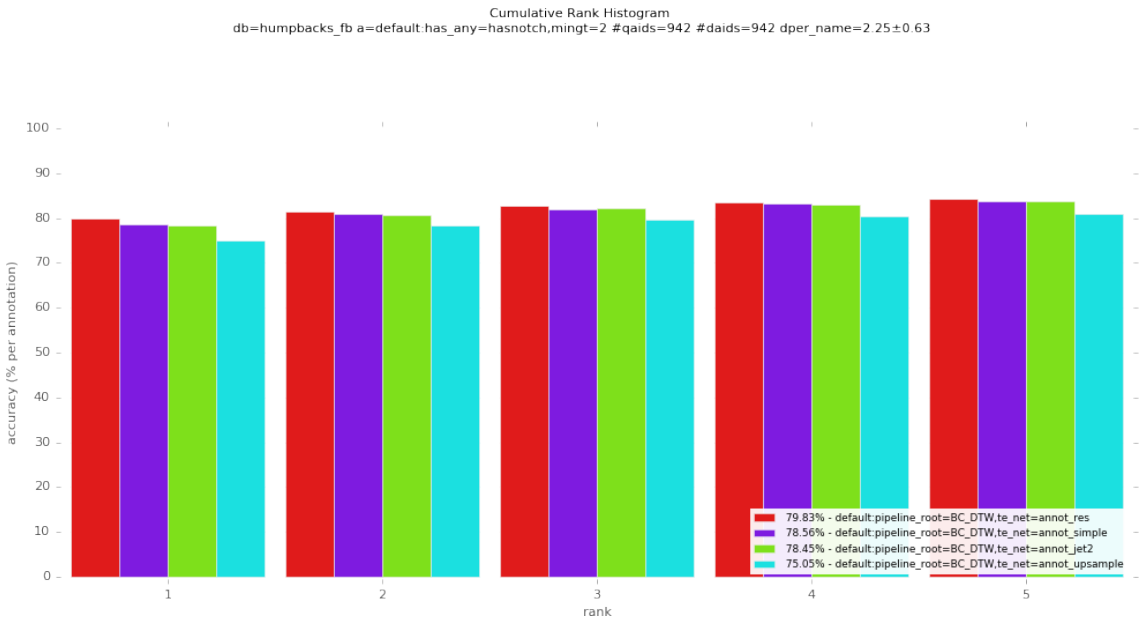


Figure 4.5: Trailing Edge Scorer Architectures. The highest performing trailing edge scorer (Residual) is shown in red, followed by Simple, Jet, and Upsample (in descending order of accuracy).

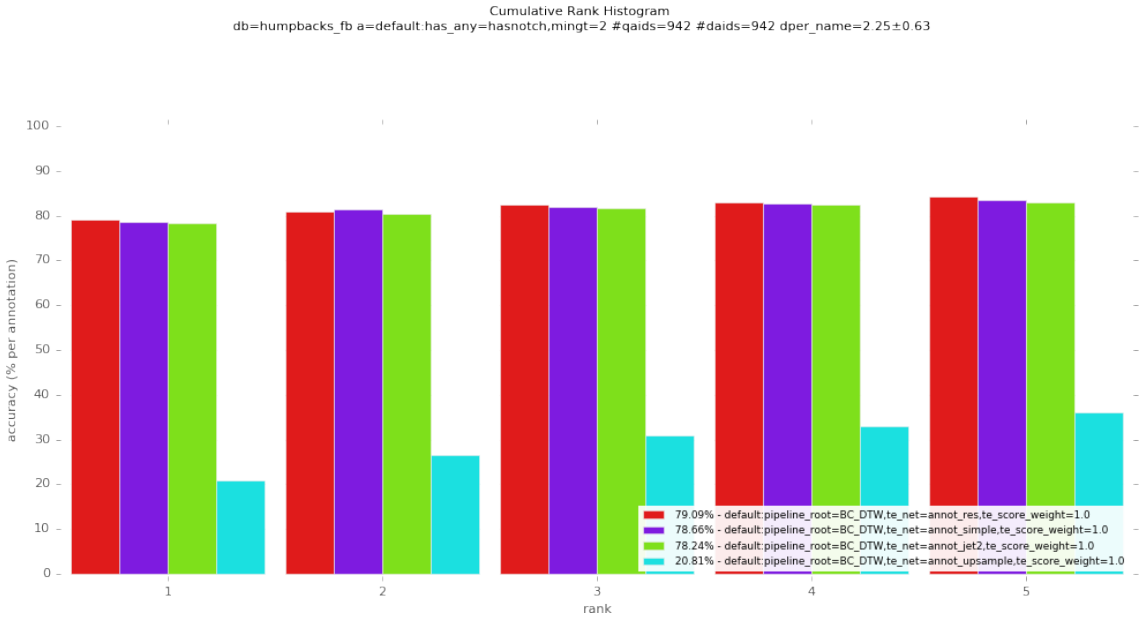


Figure 4.6: Trailing Edge Scorer Architectures at $\beta = 1$. Upsample (cyan) performs significantly worse than the other networks, which all perform comparably.

4.4.5 Using a Trailing Edge Scorer

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, though the differences

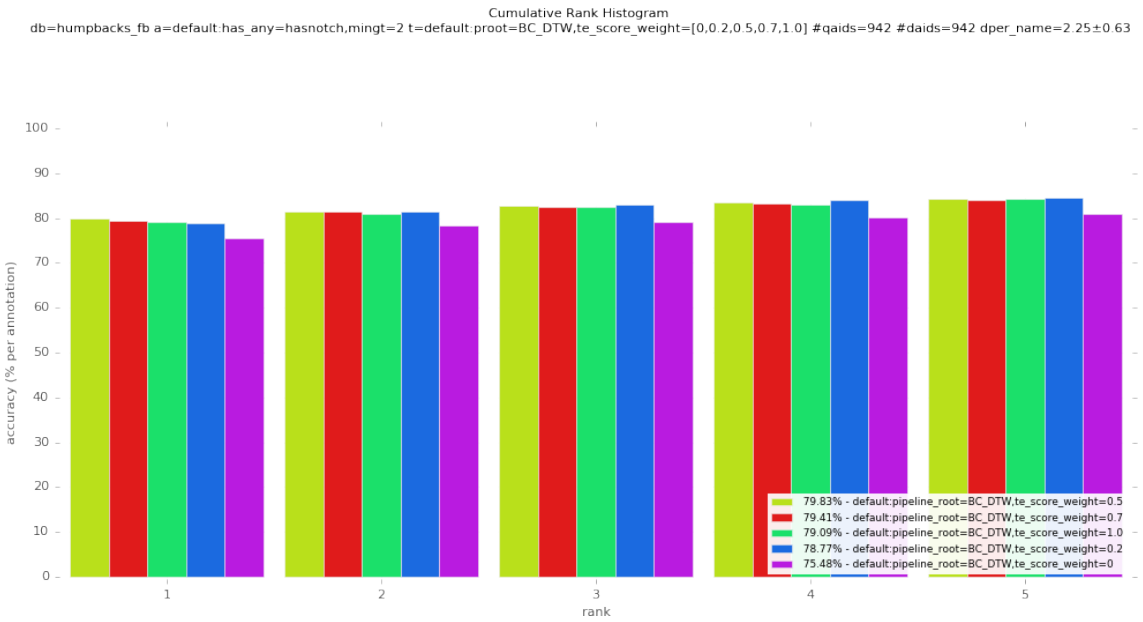


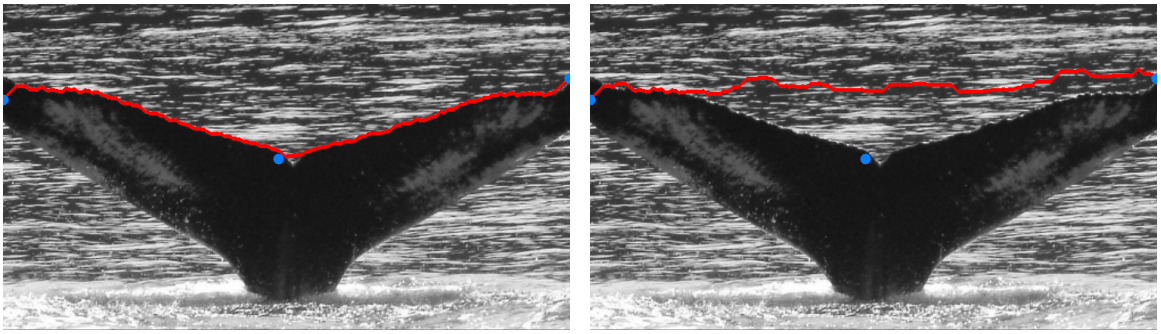
Figure 4.7: Varying β. Setting $\beta = 0.5$ (yellow) provides only marginally better results over any other non-zero value of β, but is significantly better than $\beta = 0$ (purple).

are largely insignificant. However, not using the trailing edge scorer (i.e. $\beta = 0$) performs significantly worse than using one at all (even with a low weight). We show an example case where trailing edge scoring helps in Figure 4.8.

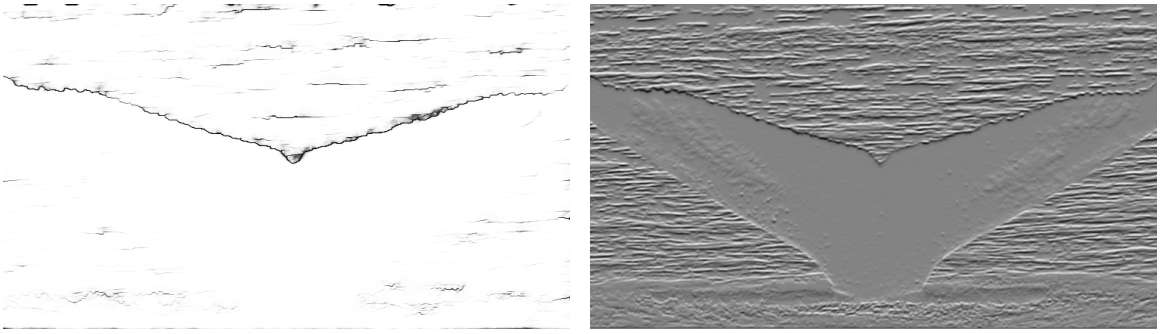
4.4.6 Setting n

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 4.9 that limiting the number of neighbors to the immediate neighborhood (i.e. $n = 3$) produces a significant boost over a larger neighborhood. We hypothesize that the trailing edges extracted with $n = 3$ are necessarily less detailed, and as a result can be more invariant to slight changes in pose of the fluke. While this effect is only present in a small number of cases, the difference in accuracy here is significant, with 78% of the improvements resulting in



a $\beta = 0.5$ **b** $\beta = 0$



c T_p **d** I_y

Figure 4.8: Example Use of Trailing Edge Scorer. In (a), we have the trailing edge extracted with the Residual scorer. Compare to (b), which did not use any scorer at all, resulting in a match failure.

a score difference over ϵ.

4.4.7 Using the Notch Keypoint

Initially, we believed that using the notch as a control point (i.e. forcing the trailing edge to go through the notch) would give better trailing edges, and therefore an increase in accuracy. As it turns out, this is not the case — using the notch as a control point gives somewhat worse accuracy. Surprisingly, this decrease in matching accuracy holds for both manually annotated and automatically extracted keypoints. However, most of these discrepancies did not result in a score difference above ϵ. This could be because of the small difference in trailing edge between those with the notch as a part of the trailing edge and those without, which leads to matching failure but not a large difference in score.

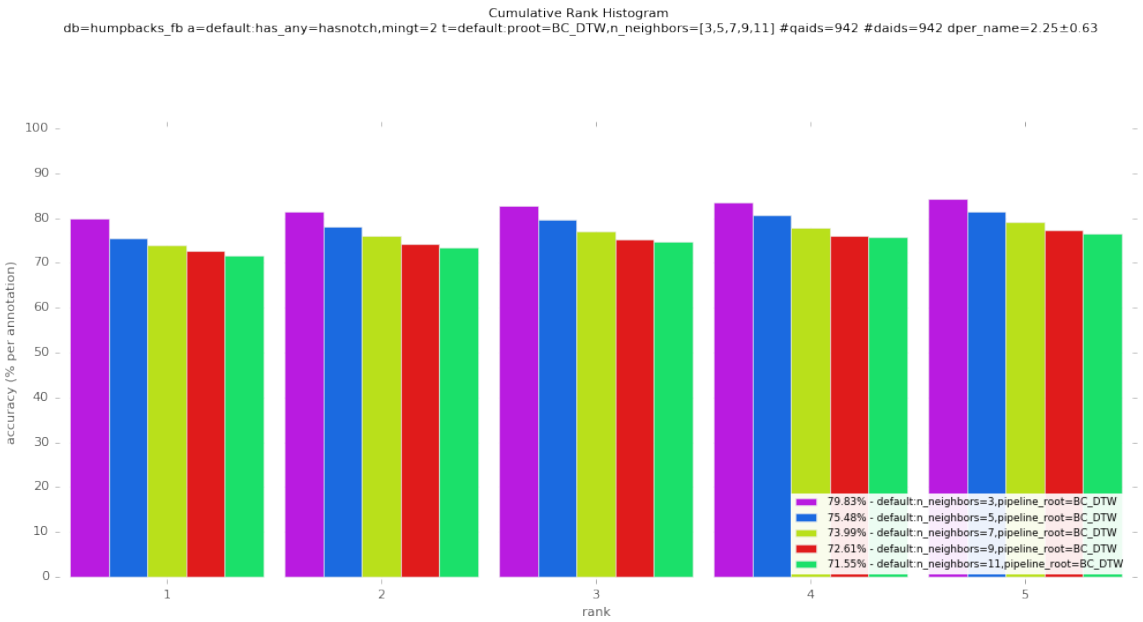
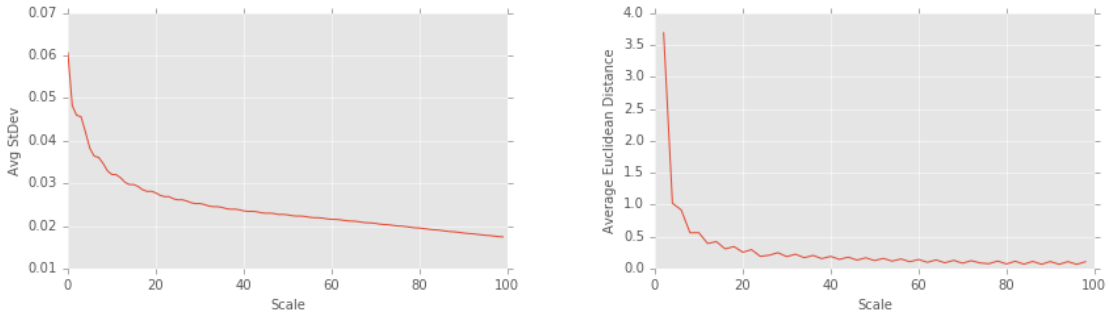


Figure 4.9: Varying n. This shows that the optimal neighborhood constraint is $n = 3$ (purple), despite qualitatively producing worse-looking trailing edges. After $n = 5$ (blue), the trailing edges can become very noisy affecting match accuracy.

4.4.8 Curvature Scales

Computing the curvature is one of the least parameterized parts of the process. Despite this, figuring out what the optimal scales (and how many) are is expensive to do experimentally. Instead of exhaustively exploring these options, we use heuristics to determine which curvature scales to explore. Each scale measures the curvature of some percentage of the trailing edge around each point on the trailing edge. Intuitively, we want to capture the curvature at a scale that provides the most variation between trailing edges. Therefore, in order to determine which scales to measure, we look at how much the actual curvature changes between successive scales, as well as the average variance in curvature among trailing edges at each scale.

We find that (Figure 4.10a) as block curvature scale is increased, the diversity at any given point in the curvature goes down drastically (as expected). Therefore, we stick to the lower end of the scale, keeping the curvature scales measured below 10%. We can also see in Figure 4.10b that successive curvature scales show bigger differences at lower scales than at higher scales, which reinforces using curvatures



a Curvature Diversity

b Inter-scale difference

Figure 4.10: 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 curvatures measured at successive scales.

below 10%, but also encourages bigger jumps between scales to maximize diversity while minimizing computation time. Based on the above, we evaluated scales that run from 1% to 10%, with varying levels of resolution, and found little to no significant effect on matching accuracy compared to using the default set of scales.

4.4.9 Sakoe-Chiba bound

In Figure 4.11, we show the effects of varying the window (i.e. the Sakoe-Chiba bound T) in the dynamic time warping distance computation. At around 10% of the query trailing edge length we maintain only slightly worse accuracy compared to the full window (i.e. 50%), but below this accuracy is severely affected. We find that overall there is a $4\times$ slow-down in wall-clock time on our testing machine when going from a window size of 10% to one of 50%. Thus, we use this value for the window size so as to minimize computation time while maintaining the total accuracy. Additionally, while it is possible for gross mismatches to occur from there being no window boundary, we can see from Figure 4.11 that this does not pose a problem.

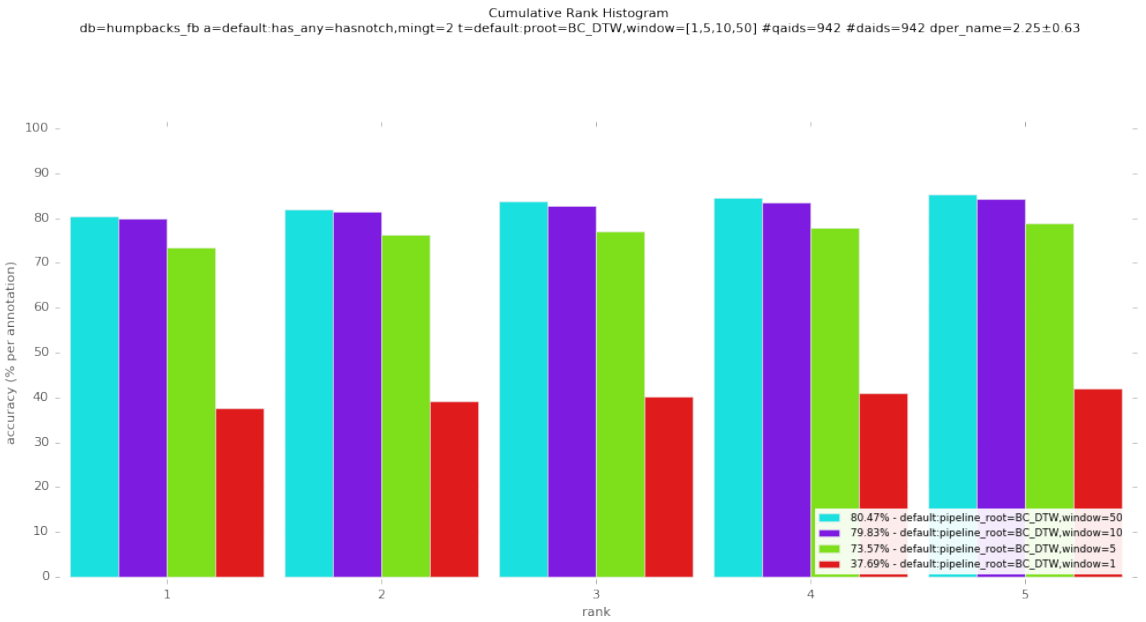
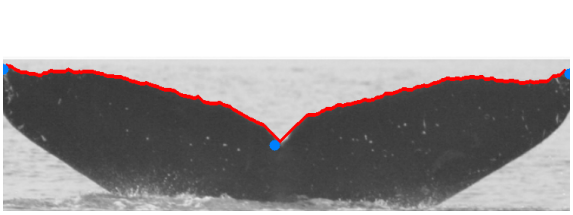
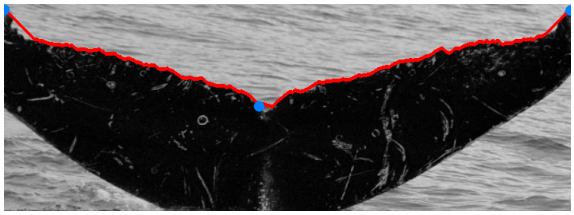


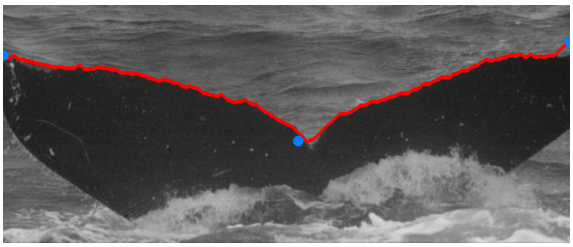
Figure 4.11: Varying Sakoe-Chiba Bound. We achieve good results with the Sakoe-Chiba bound set to 10% (yellow), although we can get slightly better results with it set to 50% (green) at the expense of computation time.



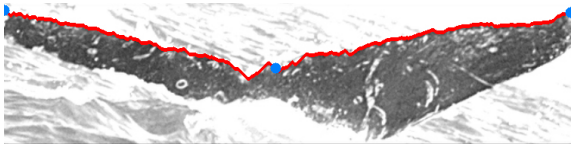
a Query Image 1



b Correct Match 1



c Query Image 2



d Correct Match 2

Figure 4.12: Example Disagreements Between Hotspotter and our method. On the left side, (a) was matched correctly to (c) by our method, whereas Hotspotter could not find any matches for (a). On the right hand side however, Hotspotter gave (d) as the top match for (b) despite a large variance in pose and lighting, while our method failed to rank (d) in the top 5 matches for (b).

Table 4.2: Comparison with Hotspotter. This table provides a comparison between our method and Hotspotter in terms of top-1 ranking accuracy on the Flukebook dataset.

Method	Hotspotter	Our Method	Hotspotter ∪ Our Method
Top-1 Accuracy	77.81%	79.83%	93.52%

4.5 In Combination with Hotspotter

Comparatively, our method gets a slightly better top-1 accuracy than Hotspotter (see Table 4.2). However, our method targets a specific part of the fluke whereas Hotspotter recognizes general patterns, and thus can make use of the internal texture of the fluke. Therefore, by combining our method with Hotspotter — if we were able to automatically pick out which algorithm was right for a given ranking — we would expect to see a significant increase in accuracy. We find that in this ideal scenario, we can achieve a 93% top-1 accuracy on the Flukebook dataset. However, automatically deciding which algorithm to use for matching is non-trivial, and we are still exploring it.

See Figure 4.12 for an example where Hotspotter correctly finds a match where our method fails, and vice-versa.

4.5.1 Characterization of Cases which each Method can Handle

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. We were unable to find a single heuristic that correlates with our

method failing to find a match. Particularly we find that smoothness of the trailing edge, characterized by the standard deviation of the trailing edge slope, does not correlate to match failure. This implies that smooth trailing edges are about as easy to match for our method as rough, distinctive trailing edges. However, since the dataset we use consists primarily of individuals with only two images, there is necessarily a lot of variation in "matchability" of given pairs of images.

CHAPTER 5
Discussion

5.1 Issues with the Proposed Method

The primary issue that we encountered is that whether or not a given pair of trailing edges belongs to the same individual based solely on the distance between them is difficult (see Figure 4.1). This hinders finding a threshold distance such that below it we can consider a pair of trailing edges to match (i.e. a verification task). 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's rank significantly. This affects the usability of our method, as ideally a human operator verifying a match would only have to look at the top-k flukes for some small value k. However, we find that in many failure cases the correct match is far down the list, and there is no reasonable value k (i.e. that a human operator would want to find a match in) that all matches are within (see Figure 5.1).

While this handling of failure cases is sub-optimal, Figure 5.2 shows that by thresholding the difference in scores between the first and second ranked flukes for a given query, we can determine with high confidence whether or not a given query is matched correctly. However, we cannot easily ascertain a failure case from this, as there are many success cases that overlap with score differences that characterize failure cases.

There are several other issues with our method, 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 without needing 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. That said, 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

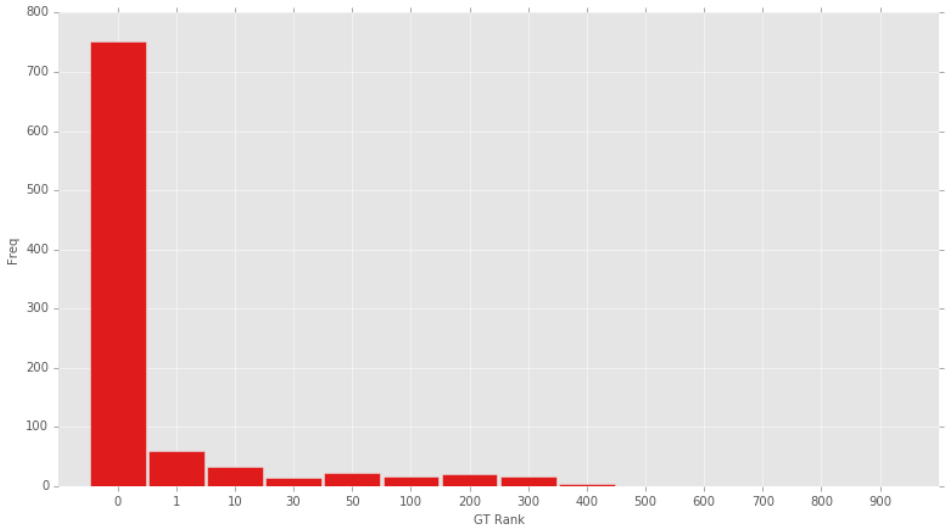
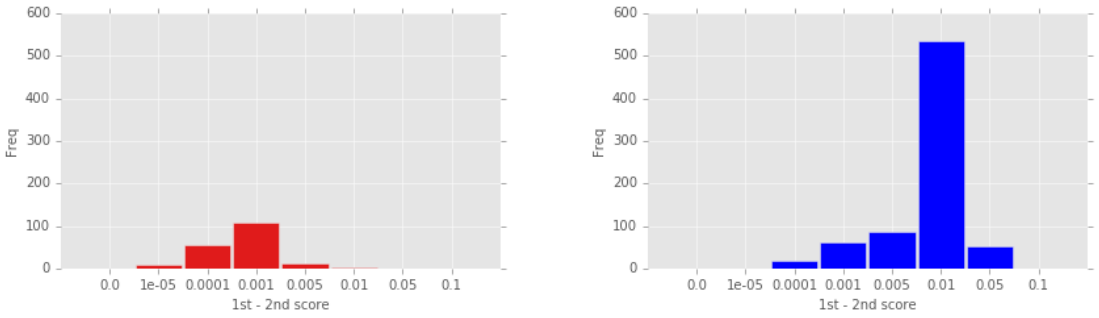


Figure 5.1: Histogram of Ground-Truth Ranks. Note that the histogram ranges are uneven to better show the lower end of the range. In order to have all matches found within the top-k matches we would have to set $k = 414$.

to out-of-plane rotations (as seen in Figures 3.2 and 1.3) — so these assumptions that we make help get a good trailing edge. However, imposing this requirement complicates and restricts the actual photography of these flukes by requiring the photographer to be in a very specific position with respect to the subject fluke.

We noted that for the trailing edge extractor, it would often use the gradient information to define a trailing edge rather than some deeper semantic image properties (see Figure 3.7). Part of the reason 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 (where the gradient signal correlates with the trailing edge), making it hard to train the network to handle hard cases. We did use data augmentation to help rectify this bias, but it was not completely effective. 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.



a Failure Cases **b** Success Cases

Figure 5.2: Histogram of Score Differences between 1st and 2nd Rank. Note that the histogram ranges are uneven to better show the higher end of the range.

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 either Hotspotter or our method for any given query and resultant ranking over possible matches. This would result in a more general method that could be robust to obscured trailing edges and some out-of-plane rotations, while also handling untextured flukes.

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 obviate the need for such a system and make it difficult to train a detection and orientation model.

Extracting the trailing edge is currently done with an unsophisticated and restricted algorithm, and improving it is a viable avenue for future work. There are several drawbacks, but its inability to handle larger “slopes” in the trailing edge

without introducing noise is one of the main issues. Additionally, it is currently non-trivial to efficiently determine the second or third best trailing edge given the completed cost matrix. This would give us the ability to evaluate candidate trailing edges and choose the best one for matching. On top of that, the trailing edge scoring networks could be better trained with more annotated trailing edges, the creation of which requires significant manual effort.

It is worth noting however that the more images that are annotated (and can subsequently be used for training), the better our method gets. This means that crowd-sourced annotations could improve our method, both in fluke keypoint extraction and trailing edge scoring.

There is a lot to be done with the matching algorithm itself, namely in ensuring that it is more tolerant to insignificant 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 and Hughes et al. [8] operate. Another avenue of exploration is speeding up our method, either by pruning completely unlikely candidates for a query or by developing an indexing algorithm for an efficient (i.e. sub-linear) search of possible matches. The latter is a difficult task as our distance function is non-metric. This was beyond the scope of this work, however it would certainly be helpful as the size of datasets on which this method is evaluated increases.

We also believe that the embedding network approach that was briefly described in section 3.3.3 showed promise and is worth re-evaluating on a larger dataset.

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] J. Calambokidis *et al.*, *SPLASH: Structure of populations, levels of abundance and status of humpback whales in the North Pacific.* Olympia, WA: Cascadia Research, 2008.

[2] 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*, vol. 16, no. 2, pp. 338–354, Apr. 2000.

[3] 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.* IEEE, pp. 230–237.

[4] C. Baker *et al.*, "Abundant mitochondrial DNA variation and world-wide population structure in humpback whales," *Proceedings of the National Academy of Sciences*, vol. 90, no. 17, pp. 8239–8243, Sep. 1993.

[5] T. Branch, "Humpback whale abundance south of 60 S from three complete circumpolar sets of surveys," *Journal of Cetacean Research and Management*, vol. 3, no. Special 3, pp. 53–69, 2011.

[6] S. A. Mizroch, J. A. Beard, and M. Lynde, "Computer assisted photo-identification of humpback whales," *Report of the International Whaling Commission*, vol. 12, no. Special 12, pp. 63–70, May. 1990.

[7] H. Whitehead, "Computer assisted individual identification of sperm whale flukes," *Report of the International Whaling Commission*, vol. 12, no. Special 12, pp. 71–77, May. 1990.

[8] B. Hughes and T. Burghardt, "Automated identification of individual great white sharks from unrestricted fin imagery," in *Proceedings of the 2015 British Machine Vision Conference*, M. W. J. Xianghua Xie and G. K. L. Tam, Eds. Swansea, UK: BMVA Press, September 2015, pp. 92.1–92.14.

[9] E. Kniest, D. Burns, and P. Harrison, "Fluke Matcher: A computer-aided matching system for humpback whale (Megaptera novaeangliae) flukes," *Marine Mammal Science*, vol. 26, no. 3, pp. 744–756, Jul. 2010.

[10] d. J. Hartog and R. Reijns. (2013) I3S Contour MANUAL. [Online]. Available: http://www.reijns.com/i3s/download/I3SC.pdf [Accessed: 4 Apr. 2016].

[11] 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*, vol. 2, no. 3, pp. 173–176, Dec. 2000.

[12] 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*, vol. 31, no. 2, pp. 243–247, Sep. 2005.

[13] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, Nov. 2004.

[14] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Lake Tahoe, NV: Curran Associates, Inc., Dec. 2012, pp. 1097–1105.

[15] C. Szegedy *et al.*, "Going deeper with convolutions," in *Proceedings of the 2015 IEEE Conference on Computer Vision and Pattern Recognition*, Boston, MA, Jun. 2015, pp. 1–9.

[16] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the 2015 IEEE Conference on Computer Vision and Pattern Recognition*, Boston, MA, Jun. 2015, pp. 3431–3440.

[17] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Semantic image segmentation with deep convolutional nets and fully connected crfs," *CoRR*, vol. abs/1412.7062, Dec. 2014.

[18] H. Fan, Z. Cao, Y. Jiang, Q. Yin, and C. Doudou, "Learning deep face representation," *CoRR*, vol. abs/1403.2802, Mar. 2014.

[19] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," in *Proceedings of the 2015 IEEE Conference on Computer Vision and Pattern Recognition*, Boston, MA, Jun. 2015, pp. 815–823.

[20] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological Cybernetics*, vol. 36, no. 4, pp. 193–202, Apr. 1980.

[21] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[22] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *CoRR*, vol. abs/1207.0580, 2012.

[23] V. Dumoulin and F. Visin, "A guide to convolution arithmetic for deep learning," *CoRR*, vol. abs/1603.07285, pp. 1–28, Mar. 2016.

[24] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, Sep. 2014.

[25] P. Sermanet *et al.*, "Overfeat: Integrated recognition, localization and detection using convolutional networks," *CoRR*, vol. abs/1312.6229, Dec. 2013.

[26] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *CoRR*, vol. abs/1502.03167, Feb. 2015.

[27] A. M. Saxe, J. L. McClelland, and S. Ganguli, "Exact solutions to the nonlinear dynamics of learning in deep linear neural networks," *CoRR*, vol. abs/1312.6120, Dec. 2013.

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

[29] Y. Akagi, R. Furukawa, R. Sagawa, K. Ogawara, and H. Kawasaki, "A Facial Tracking and Transfer Method with a Key Point Refinement," in *ACM SIGGRAPH 2013 Posters*, SIGGRAPH '13. Anaheim, CA: ACM, 2013, pp. 79–79.

[30] 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*, vol. 27, no. 11, pp. 1021–1036, Nov. 2011.

[31] Y. Sun, X. Wang, and X. Tang, "Deep convolutional network cascade for facial point detection," in *Proceedings of the 2013 IEEE Conference on Computer Vision and Pattern Recognition*, Portland, OR, Jun. 2013, pp. 3476–3483.

[32] D. Nouri. (2014, Dec.) Using convolutional neural nets to detect facial keypoints tutorial. [Online]. Available: http://danielnouri.org/notes/2014/12/17/using-convolutional-neural-nets-to-detect-facial-keypoints-tutorial/ [Accessed: 4 Apr. 2016].

[33] F. Ning *et al.*, "Toward automatic phenotyping of developing embryos from videos," *IEEE Transactions on Image Processing*, vol. 14, no. 9, pp. 1360–1371, Sep. 2005.

[34] 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 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Lake Tahoe, NV: Curran Associates, Inc., Dec. 2012, pp. 2843–2851.

[35] B. Hariharan, P. Arbeláez, R. Girshick, and J. Malik, "Simultaneous detection and segmentation," in *Proceedings of the 2014 European Conference on Computer Vision*. Zurich, CH: Springer, Sep. 2014, pp. 297–312.

[36] 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 *Proceedings of the Second International Conference on Computer Vision*. Tampa, FL: IEEE, Dec. 1988, pp. 95–99.

[37] M. Kass, A. Witkin, and D. Terzopoulos, "Snakes: Active contour models," *International Journal of Computer Vision*, vol. 1, no. 4, pp. 321–331, Jan. 1988.

[38] S. Avidan and A. Shamir, "Seam carving for content-aware image resizing," *ACM Transactions on Graphics*, vol. 26, no. 3, Jul. 2007.

[39] A. Monroy, A. Eigenstetter, and B. Ommer, "Beyond straight lines – object detection using curvature," in *Proceedings of the 18th IEEE International Conference on Image Processing*. Brussels, BE: IEEE, Sep. 2011, pp. 3561–3564.

[40] P. Fischer and T. Brox, "Image Descriptors Based on Curvature Histograms," in *Proceedings of the 36th German Conference on Pattern Recognition*, X. Jiang, J. Hornegger, and R. Koch, Eds. Münster, DE: Springer International Publishing, Sep. 2014, pp. 239–249.

[41] N. Kumar *et al.*, "Leafsnap: A computer vision system for automatic plant species identification," in *Proceedings of the 2012 European Conference on Computer Vision*. Firenze, IT: Springer, Oct. 2012, pp. 502–516.

[42] H. Pottmann, J. Wallner, Q.-X. Huang, and Y.-L. Yang, "Integral invariants for robust geometry processing," *Computer Aided Geometric Design*, vol. 26, no. 1, pp. 37 – 60, Jan. 2009.

[43] H. Sakoe and S. Chiba, "Dynamic programming algorithm optimization for spoken word recognition," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 26, no. 1, pp. 43–49, Feb. 1978.

[44] S. Salvador and P. Chan, "Toward Accurate Dynamic Time Warping in Linear Time and Space," *Intelligent Data Analysis*, vol. 11, no. 5, pp. 561–580, Oct. 2007.

[45] D. Lemire, "Faster retrieval with a two-pass dynamic-time-warping lower bound," *Pattern recognition*, vol. 42, no. 9, pp. 2169–2180, Sep. 2009.

[46] M. E. Munich and P. Perona, "Continuous dynamic time warping for translation-invariant curve alignment with applications to signature verification," in *The Proceedings of the Seventh IEEE International Conference on Computer Vision, 1999.*, vol. 1, Corfu, GR, Sep. 1999, pp. 108–115.

[47] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, Dec. 2014.

[48] M. Jaderberg, K. Simonyan, A. Zisserman *et al.*, "Spatial transformer networks," in *Advances in Neural Information Processing Systems 28*, Montreal, CAN, Dec. 2015, pp. 2008–2016.

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

[50] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, Dec. 2015.

[51] T. Rakthanmanon *et al.*, "Searching and mining trillions of time series subsequences under dynamic time warping," in *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. Beijing, CN: ACM, Aug. 2012, pp. 262–270.

[52] F. C. Crow, "Summed-area tables for texture mapping," *ACM SIGGRAPH Computer Graphics*, vol. 18, no. 3, pp. 207–212, Jul. 1984.

[53] M. Müller, *Information retrieval for music and motion*, M. Müller, Ed. Berlin, DE: Springer, 2007, vol. 2.

[54] Y. Qiao and M. Yasuhara, "Affine invariant dynamic time warping and its application to online rotated handwriting recognition," in *Proceedings of the 18th International Conference on Pattern Recognition*, vol. 2. Hong Kong, HK: IEEE, Aug. 2006, pp. 905–908.

[55] O. M. Parkhi, A. Vedaldi, and A. Zisserman, "Deep face recognition," in *Proceedings of the 2015 British Machine Vision Conference (BMVC)*, M. W. J. Xianghua Xie and G. K. L. Tam, Eds. BMVA Press, Sep. 2015, pp. 41.1–41.12.

[56] 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, Amherst, MA, Tech. Rep., 2007.

[57] R. Hadsell, S. Chopra, and Y. LeCun, "Dimensionality reduction by learning an invariant mapping," in *Proceedings of the 2006 IEEE Conference on Computer Vision and Pattern Recognition*, vol. 2. New York, NY: IEEE, Jun. 2006, pp. 1735–1742.

[58] S. van der Walt, S. Colbert, and G. Varoquaux, "The NumPy array: A structure for efficient numerical computation," *Comput. in Sci. Eng.*, vol. 13, no. 2, pp. 22–30, Mar. 2011.

[59] J. Bergstra *et al.*, "Theano: a CPU and GPU math expression compiler," in *Proc. Python for Scientific Comput. Conf.*, SciPy '10, vol. 4, Austin, TX, Jun. 2010, pp. 3–10.

[60] F. Bastien *et al.*, "Theano: new features and speed improvements," *CoRR*, vol. abs/1211.5590, pp. 1–10, Nov. 2012.

[61] S. Dieleman *et al.* (2015, Aug.) Lasagne: First release. [Online]. Available: http://github.com/Lasagne/Lasagne [Accessed: 1 Nov. 2015].

[62] G. Guennebaud, B. Jacob *et al.* (2010) Eigen v3. [Online]. Available: http://eigen.tuxfamily.org [Accessed: 4 Apr. 2016].