IDENTIFYING HUMPBACK WHALE FLUKES BY SEQUENCE MATCHING OF TRAILING EDGE CURVATURE

By

Zachary Jablons

A Thesis Submitted to the Graduate

Faculty of Rensselaer Polytechnic Institute

in Partial Fulfillment of the

Requirements for the Degree of

MASTER OF SCIENCE

Major Subject: COMPUTER SCIENCE

Examining Committee:

_____

Dr. Charles Stewart, Thesis Adviser

_____

Dr. Barbara Cutler, Member

_____

Dr. Bülent Yener, Member

Rensselaer Polytechnic Institute
Troy, New York

April 2016
(For Graduation May 2016)

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGMENTS

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

ABSTRACT

Photographic identification of humpback whale (Megaptera novaeangliae) flukes (i.e. their tail) is an important task in marine ecology, and is used in tracking migration patterns and estimating populations [5] [7]. In this thesis, we lay out a method that automates the photo-identification of humpback flukes, using the "trailing edge" of the fluke. Using this method, we achieve a fairly high top-1 ranking accuracy on a large subset of the SPLASH [7] dataset consisting of about 400 identified individuals. We also show that in combination with a general appearance based matching algorithm, Hotspotter [9], 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 much manual effort at test-time.

CHAPTER 1
Introduction

1.1 Humpback Whales

Since the international ban on commercial hunting of Humpback whales in 1966, humpback whales have grown from a population of only 5000 [3] to over 50000 [6]. As the population grows, it becomes more and more important to be able to automatically identify individual whales in order to accurately monitor their population growth and follow their migration patterns, among other ecological conservation endeavours. One of the most reliable methods for photo-identifying humpback whales is by taking pictures of their flukes as they dive after breaching the surface of the water.

1.1.1 Distinguishing Individual Flukes

The primary distinguishing features of these flukes are — for the purposes of this work — separated into two main areas; the "trailing edge" of the fluke, and the internal texture. There are pros and cons to identification with either feature. The internal texture is a more obvious choice for identification, as it is disinctive even from a distance and blurred, whereas trailing edges require high quality photographs. However, there are humpback flukes who have indistinct (e.g. all-black) internal textures, which make matching based on texture impossible. An example of this is shown in Figure 1.2.

Additionally, the work of Blackmer et al. [5] finds that the trailing edge changes less with age than the internal texture of the fluke, which means that it can (potentially) be a more reliable identifier over time.

That said, the requirements for getting a good photograph of the trailing edge can be impractical, as the trailing edge can be obscured by out of plane rotations (an example is shown in Figure 1.3).

Figure 1.1: Example Fluke. Example image of a humpback whale fluke from the SPLASH [7] dataset [1]. This image has a clearly visible trailing edge and internal fluke pattern.

1.2 Current Identification Methods

Computer-assisted photo-identification of humpback whale flukes has been attempted since the early 90s [30]. While early efforts mostly relied on a manual description of the fluke that would then be matched (against other stored descriptions), later efforts have involved matching flukes based on automated analysis of both the internal texture and trailing edge.

Existing computer-assisted photo-identification methods can be broadly separated into three categories:

- Manually annotated – a human must manually annotate or catalogue features on each fluke image, which are then automatically matched
- Semi-automated – a human must guide an algorithm (e.g. by setting control points or highlighting interesting regions) that then automatically identifies the individual
- Fully-automated – the algorithm can identify individuals from raw images

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

1.2.1 Based on Trailing Edge

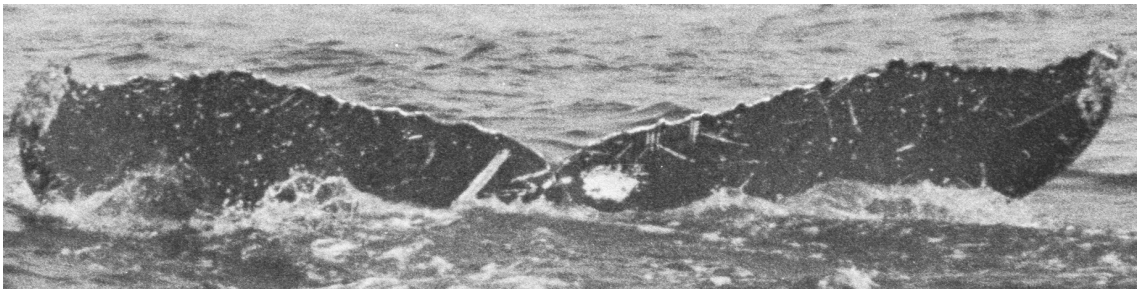
In the I3S contour system [15], the user must input start and end points on the query trailing edge, after which its contour is extracted. This trailing edge is then resized and aligned so that it can be compared with absolute difference against the database trailing edges. It also compares a set of possible shifts, rotations, and scales of the query trailing edge to account for these differences. At the time of writing no published results on this system applied to humpback whales could be found.

Automatically identifying humpback whales by their entire trailing edge contour is done experimentally in Hughes et al. [20], using a technique that is originally designed for great white sharks. This technique segments the trailing edge into a set of possible contours and matches them combinatorially using Difference of Gaussians. The authors achieve a comparable accuracy to our method, however for a much smaller dataset of humpback flukes than the one evaluated here.

While trailing edge matching has seen limited use in Humpback whale identification, it is a much more common technique in sperm whale (P. macrocephalus) identification [19], [4] [48], with varying levels of manual effort. We detail these methods below as the fluke trailing edge matching paradigm is similar across species.

In Whitehead's work [48], 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, with a distance threshold. This requires significant manual effort, and achieves a low accuracy [4].

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.

To contrast, the method proposed by Huele et al. [19] uses a semi-automatic extraction of sperm whale trailing edge, and then applies wavelet transformations which are cross-correlated to determine a similarity measurement.

1.2.2 Based on general Fluke appearance

The primary method for computer-assisted photo-identification of humpback whale flukes is to use the internal fluke pattern, as seen in the work of Mizroch et al. [30] and Flukematcher [24].

In [30], information about the fluke is manually catalogued and used to match individual whales. The fields that are catalogued contain information primarily about the overall coloration patterns of the fluke, as well as the shape of the central notch. The matching algorithm simply ranks flukes by looking at how similar annotated patterns are, and requires significant manual effort to identify individuals.

In Flukematcher [24], control points (including the ones we use) 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 by the user. A variety of heuristic features are then extracted, which are matched using a variety of similarity measures. This is an effective method, achieving 82% top-1 accuracy on similar data to ours, but requires significant manual effort.

1.3 Method Outline

This method is, as stated before, a fully automated algorithm at test time, however in order to train the models some manual annotation is required. The steps are roughly as follows, taking raw images as input.

- Find the left and right tips of the fluke
- Extract the trailing edge contour betweeen these points
- Compute the curvature of the trailing edge contour at multiple scales
- Determine a ranking of possible identities using a distance computed by dynamic time warping

We also combine this with a ranking from Hotspotter [9], a generalized appearance based identification method that has been successful for several different flukes.

1.4 The Dataset

The main dataset that is used and evaluated in this work is a subset of the dataset collected by the SPLASH project [7]. It consists of about 1400 identified photographs spread over about 860 identified individuals. Of these, only 433 individuals have more than one image associated with them, giving 942 images that can be used in a one-to-one comparison.

We also use an external dataset of unidentified (but annotated) humpback flukes for training models.

1.5 Thesis Outline

The rest of this thesis will cover the background literature on many of the algorithms used in this work as well as the primary method itself. We also detail some of the more significant alternatives that we evaluated, and how changing the various parameters of the main method affects identification accuracy. We conclude with a discussion on the failings of the primary method, as well as ways to improve it and generalize it to identifying trailing edges in other animals.

CHAPTER 2
Background

In this chapter, we provide a series of sections detailing background information on the algorithms on top of which our trailing identifier was developed.

2.1 Convolutional Networks

In recent years, convolutional neural networks have provided state of the art results in several challenging computer vision tasks, including general image classification [25], [46], image segmentation [29], [8] and individual identification (specifically for human faces) [11], [41].

The essential idea of a convolutional network is that it we can use the gradient of an error signal to learn hierarchies of convolution kernels separated by nonlinear activation functions. These networks are considered to be a form of neural network where the convolutions provide a meaningful prior when applied to data with spatial relationships (e.g. image data). Convolutional networks have been around for a long time [13], but the current incarnation of these networks can be traced back to the seminal work of LeCun et al. [27]. More modern convolutional networks follow a common framework of using Rectified Linear Units (ReLUs) as activation functions, and Dropout [17] layers after fully connected layers for regularization. Additionally, the convolutional kernels used are commonly small square kernels with "same" padding alternated with 2× downsampling layers (specifically max pooling layers) [43], [42], [25].

The convolutional networks used in this thesis follow the above framework, and also use batch normalization [21] at every layer.

2.1.1 Facial Keypoint Prediction

Facial keypoints are essentially coordinates on an image of a (human) face that detail the locations of nose, eyes, mouth, etc. They are commonly used in facial identification pipelines [47], as well as in motion capture [1] and expression

recognition [**?**].

There has been recent work in using convolutional networks for facial keypoint prediction [45], [35]. 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.

In this work, we similarly predict fluke keypoints which are predicted using essentially the same paradigm as the above works.

2.1.2 Fully Convolutional Networks

Classically, convolutional networks reduce an image to a single (spatially invariant) vector, which is then used for classification (or embedding, regression, etc.) To do this, these networks usually have fully connected (or dense) layers towards the end. This ensures that the receptive field of the network covers the entire image, which is practical for many applications where a scalar or fixed size prediction is required. However, when dealing with arbitrarily sized images, it is useful to use networks that are "fully convolutional", in which case the entire network consists of convolution kernels. Convolutional networks that reduce to dense layers can be cast as fully convolutional networks by replacing the dense layers with 1×1 kernels, using the dense units as channels. This technique is especially applicable in segmentation tasks [34], as this process allows for (downsampled) predictions spatially distributed throughout the image. By then upsampling and combining different stages of prediction, the authors in [29] produce high quality image segmentations, a technique that we replicate.

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 Seam Carving

Seam carving is a technique that tries to resize images without warping or distorting the objects shown in the image [2]. This technique uses a dynamic programming algorithm to find minimal salience paths through an image, where salience is often defined as the gradient. The motivation for this is that these minimal saliency

paths are not important to the image, so they can be removed to reduce its size.

While this method is not directly used in this work, the underlying algorithm for trailing edge extraction is essentially a single iteration of the seam carving algorithm, using gradient information.

2.3 Curvature Measures

Contour curvature measures are commonly used to characterize the overall shape of an contour. A lot of work has been done on using curvature information for detection [31], classification [12], and species identification [26]. This curvature information can be broadly broken down into either integral or differential curvature, and is usually computed at multiple scales.

2.3.1 Differential Curvature

Differential curvature can generally be seen as measuring the angle of the tangent normal of the gradient at each point in an image [12]. For our purposes, we can then take only those points that lie on the contour and use their curvature. While doing this directly can be fast to compute, it tends to be noise sensitive and we found that integral curvature (below) works better for our purposes.

2.3.2 Integral Curvature

Integral curvature works (conceptually) by sliding a circle of some radius r along the contour [37], and measuring how much of the circle is 'inside' the contour. This measurement is usually taken at multiple scales, and has the appealing property of being invariant to rotation and translation (of the entire contour). In this work, we approximate the circular curvature with a square of size r, which appears to perform just as well but can be computed much faster.

2.4 Dynamic Time Warping

In deciding a sequence comparator, one criterion that is often important is ensuring that small shifts in the sequence do not balloon into large differences. Dynamic Time Warping (DTW) is a sequence comparison method that, roughly,

finds the optimal matching between all sets of points in the two given sequences that minimizes the overall distance (for some defined distance function) between the matched points, while keeping the locality of the points intact. This allows for shifts and some warps in the two sequences to be compensated for, and results in a nonlinear mapping of one sequence onto another. The algorithmic complexity of dynamic time warping can be limiting in large datasets, as it is quadratic in both space and time – making a one-to-one comparison a bit daunting.

There are several variants on DTW that give faster speeds [40] [28], however we only use the Sakoe-Chiba bound [39], which both constrains the neighborhood in which points can be matched and gives a complexity of $O(nw)$, where w is a user set parameter.

Sequences of curvature measures have been used with DTW for signature verification [33], however this combination has not been used for matching trailing edges to our knowledge.

CHAPTER 3
Methods

In this chapter, we detail the finalized algorithm pipeline, as well as some alternative approaches that we found to have limited successs.

3.1 Trailing Edge Extraction

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

One major assumption we make when extracting these trailing edges is that the humpback whale fluke is aligned such that its major axis is horizontal. Additionally, we generally assume that all parts of the fluke are present, however theoretically it would be possible to match a partial fluke – although unlikely.

While these assumptions do not make for an incredibly robust system, the nature of the problem (and the dataset that we had at hand) makes these assumptions reasonable. It would also be possible to add a detection step beforehand to ensure this assumption, however we do not explore this in this work.

3.1.1 Fluke keypoint prediction

One major issue with automating the above trailing edge extraction algorithm is that it requires manual annotation in the selection of the starting and ending points (i.e., the fluke keypoints), as well as any control points (specifically the bottom of the notch). To work around this, we propose a convolutional network that predicts these tip points as part of the identification pipeline.

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

The convolutional network does not need full-sized images, so the first step

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.

of the keypoint extraction pipeline is to resize the image to 256×256 pixels. This size choice is somewhat arbitrary, but we find that it provides the best performance without using an unnecessary amount of memory. The network then predicts the points as values between 0 and 1, for both the width and height of the image. These predictions are then then rescaled back up to the original image size. An example prediction is shown in Figure 3.1.

3.1.1.1 Network Design

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

We theorize that this may be because shared units between each of the three predictions leads to stronger correlations between them, reducing overall prediction flexibility.

3.1.1.2 Training Details

Generating the training data for this is straightforward given a set of annotations with the associated points to learn. The dataset that we created for this purpose contains approximately 2100 training images, 700 validation images, and 900 test images.

First, each image is resized to a fixed width while maintaining the aspect ratio. This is done to somewhat normalize the relative scale of objects in each image on the assumption that they are constrained to contain the fluke. Each image is rescaled to the network size, and then the corresponding targets are rescaled to the range $[0-1]$. The size of the original image is recorded as well. While it would be possible to treat this as a simple multi-variate regression and use RMSE loss, we achieved better results by averaging the Euclidean distance between the predicted points and true points. We also include a scalar scaling factor α, which scales each point by a proportion of the original image size. Thus, we have the scaled Euclidean loss SE

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

Where

- $\vec{t}$ and $\vec{p}$ are the (x, y) ground truth and predicted values respectively
- $\vec{s}$ is the original image width and height

The networks are trained for 1000 epochs with α set to 2e$-$2 using the Adam [23] optimizer (with recommended settings) and $l2$ regularization on the trainable parameters with a decay of 1e$-$4. All of these hyper parameters were tuned using the validation set, although the possible parameter space was not fully explored due to time constraints.

3.1.1.3 Evaluation

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

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

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. Qualitative inspection of these images shows that they are either of flukes which are not the singular or major object in the image, or horizontally oriented of flukes that are not horizontally rotated. An example of this is shown in Figure 3.1.

We attempted to use a spatial transformer network [22] 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

3.1.2 Basic Trailing Edge Extraction Algorithm

The base algorithm that is used for extracting the trailing edge uses the vertical gradient information of the image (denoted as I_y). We extract I_y using a vertically oriented 5×5 Sobel kernel [44].

We then normalize I_y with min-max scaling, giving N_y as

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

With N_y, we then need a starting point and ending point for the algorithm, denoted s and e respectively, with their x and y coordinates being denoted with subscripts.

For our purposes, we use the left and right tips of the fluke as our start and end points respectively. Additionally, we experiment with using the bottom of the notch to aid the trailing edge extractor. We explain how these points are determined in the next section.

Given a point p, we then set its corresponding column in N_y to ∞, and set the point itself set to 0.

$$N_y(p_x, \cdot) = \infty \tag{3.3}$$

$$N_y(p_x, p_y) = 0 \tag{3.4}$$

This forces the path to "go through" these points. We do this for the start (i.e. left tip) and end points, as well as the bottom of the notch.

The minimal path is then found by scanning the columns of N_y from left to right, starting and ending at s and e respectively. For each pixel in a column we set its cost with the following update rule:

$$C(x,y) = \begin{cases} 0 & x < 0 \\ \infty & y < 0 \text{ or } y > h \\ \min_{y-n \le y_c \le y+n}(C(x-1, y_c) + N_y(x,y)) & \text{else} \end{cases} \tag{3.5}$$

Where n is a neighborhood constraint and h is the height of N_y. We default n to 1, meaning that each pixel considers 3 'neighbors' in 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 chosen in equation 3.5. Once the end column is reached, we work backwards from e to construct the path, adding the coordinate corresponding to $B(x, y)$ at each step.

We can also extend this algorithm by adding extra 'control points' which the path is forced through, using the same methodology as forcing the path through the start and end points. Commonly, we use the bottom of the notch as a control point, although this affects accuracy negatively if it too far from the trailing edge.

While this algorithm has no understanding of humpback whale flukes, a lot of images that are constrained around the fluke with oceanic backgrounds (which is a large majority of the dataset at hand) provide high quality trailing edges when put through this algorithm. However this is not a robust algorithm for finding trailing edges, something that we will fix later on.

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 a lot of cases, but is not a robust method.

If we had a score of each pixel's "trailing edginess" in an image, the trailing edge extractor could make use of this information to make better choices in trailing edge extraction. To do this, we need a prediction of whether or not each pixel belongs to the trailing edge of a fluke, a task that is best suited to a fully convolutional network. In these networks, all convolutions (aside from max pooling layers) are "same" convolutions, which have square, odd filters (usually 3×3) and 1-padding. These "same" convolutions produce a spatial output shape that is the same as the input shape, obviating the need for any interpolation.

The four major variants on trailing edge scoring networks that we evaluated are detailed below. All of these networks function on the same paradigm of taking an arbitrarily sized image and producing an image of the same size but with a class score for each pixel.

The dataset was sourced from trailing edges extracted using the basic method detailed above, however with manual adjustments to fix many of the more common

issues we encountered with this algorithm.

To generate the training set, we extract 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). 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

There are several network architectures that were tried, however here we will report only the major variants. One major consideration that has to be made when selecting a fully convolutional network architecture is its receptive field. 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 a network, we must downsample, which makes the output less fine-grained.

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

Upsample To deal with the small receptive fields, convolutional networks typically downsample at various stages. This downsampling usually takes the form of max pooling (instead of e.g. convolutional kernels with a stride greater than one). The upsample architecture is analogous to the FCN-32s architecture in [29], although we do not go down to a 1x1 spatial extent before upsampling. This produces blocky and nearly unusable trailing edge scores, however they tend to be more connected than the disparate (but fine-grained) predictions made by the simple network.

Jet Following the deep-jet architecture from [29], we combine softmax predictions from various stages throughout the network with predictions from "farther down" the network, cascading until we hit predictions from the last convolutional layer before any downsampling. This gives more fine-grained predictions than the upsample network while giving somewhat better predictions than the simple network.

Residual While the receptive field of the Simple network is small, it can produce very fine trailing edges, which we found to be necessary for good trailing edge extraction. However with such a small receptive field it can be more prone to making mistakes. It is possible to create a very deep network of 'same' convolutions, however training very deep networks like this can run into problems very quickly. Recently, there has been work on making very deep networks feasible by adding residual connections[16], showing that these very deep networks can achieve high accuracy in a challenging benchmark. We replicate this architecture by stacking 64 3×3 'same' convolution kernels, and adding a residual connection every other layer. This performs better than the Simple network, however we do not see a marked improvement in trailing edge matching as a result.

3.1.3.2 Using the trailing edge scores

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

$$S_{te} = (1 - \beta) * N_y + \beta * (1 - T_p) \tag{3.6}$$

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

vious section.

For most of the evaluation process, we simply set $\beta = 0.5$.

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. However one of the main issues with this is that if there are any breaks in the prediction (which can happen even with the Upsample architecture), the trailing edge cannot be extracted, leading to broken trailing edges.

3.1.3.3 Training Details

All networks were trained for 100 epochs (or until convergence) with a batch size of 32, with $l2$ regularization using a decay of 1e$-$4. We used the Adam optimizer[23] (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.

Due to the nature of the training data, we were concerned that the networks would simply learn to replicate the function that generated the trailing edges, despite human corrections. In order to mitigate this issue, we included some non-spatial data augmentation, namely random Gaussian blur and randomly inverting the pixel intensities. Unfortunately this had limited success.

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

3.1.3.4 Evaluation

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

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

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

Initially, we thought that precision would be the most important metric for evaluating networks (which is reflected in the class weighting). However, we found that even a low precision classifier could give good trailing edges, and that the important measure was how detailed these trailing edges were.

This is likely because small segments of predicted trailing edge would not be chosen by the trailing edge extractor if they are too disconnected from the actual trailing edge.

3.2 Trailing Edge Matching

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

3.2.1 Curvature Measurement

In order to do this, we must first extract the curvature from the trailing edge. Given the trailing edge as a sequence of coordinates into the original image, we construct a zero-image I_0 of shape similar to the original image. Each pixel corre-

sponding to and below the trailing edge in I_0 is set to 1. 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 [10] ST from I_0 as follows.

$$ST(x, y) = \sum_{i=0}^{i=y} \sum_{j=0}^{j=x} I_0 \tag{3.7}$$

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

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

$$\begin{aligned} b(i) &= i - \frac{s}{2} \\ e(i) &= i + \frac{s}{2} \\ BC_s(x, y) &= \frac{(ST(b(x), b(y)) + ST(e(x), e(y))) - (ST(b(x), e(y)) + ST(e(x), b(y)))}{s^2} \end{aligned} \tag{3.8}$$

The numerator in equation (3.8) gives the total area within the square that is below the trailing edge, which we then normalize by dividing by the square's area.

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

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 $l_1 \times l_2$. We initialize this cost matrix

by setting the first column and row to ∞, and then $C(0,0) = 0$, intuitively forcing the optimal path to match align the beginning of BC_1 with BC_2. Then, for each cell (i, j) in the cost matrix starting with $C(1, 1)$, we use the following update rule

$$D_{\vec{s_w}}(c_1, c_2) = \|\vec{s_w} \odot (\vec{c_1} - \vec{c_2})\|_2 \tag{3.9}$$

$$C(i,j) = D_{s_w}(BC_1(i,\cdot), BC_2(j,\cdot)) + min(C(i-1,j), C(i,j-1), C(i-1,j-1)) \tag{3.10}$$

Where $\vec{s_w}$ is a vector giving the weight for each curvature scale, and $\vec{c}$ is a vector of the curvatures at different scales for a point.

Additionally, we impose the Sakoe-Chiba [39] locality constraint T so that for each element i in BC_1, we only consider the range over elements j in BC_2 of $j \in [min(i - T, 0), max(i + T, l_2)]$.

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

It's worth noting that while this distance measure is not a metric distance (i.e. it doesn't satisfy the triangle inequality), it is a symmetric distance as (3.9) is symmetric [32].

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, scale was taken care of separately), and using the Euclidean distance between points on the aligned trailing edges (i.e. in place of curvature for (3.9)) did not give good results.

There were two approaches to alignment that we evaluated, although neither achieved top-1 accuracies above 20%.

3.3.1.1 Keypoint Alignment

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

One major issue with aligning these iamges 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

Taking after the AI-DTW approach laid out in [38], we ran an iterative alignment process that used 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 was impractical.

3.3.2 Histogram Matching

One early curvature comparison method that we evaluated was to use histograms to match instead of a sequence-based method. This is a common approach for comparing curvatures [26], however we found that for our purposes even high resolution histograms did not provide enough detail to match the trailing edges properly, although this could potentially be explored further.

3.3.3 Embedding via Convolutional Networks

We also made an attempt at training convolutional networks to directly embed the images of flukes into a n dimensional vector, much like [41] and [36]. However, most of the previous literature on this technique is applied to larger datasets such as LFW[18], which is significantly larger than the dataset that we had available. A major factor in this is that these larger datasets often have five to ten images per identity (if not more), whereas most of the identities in our dataset had one or two images associated.

Regardless, we attempted the embedding approach (from raw images), however even a severely overfit convolutional network only achieved half the top-1 accuracy on its training set that the main method is able to achieve. We tried both triplet loss (a modified version of the one detailed in [41]) and contrastive loss [14] 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 this method achieves on the Flukebook dataset. The main results for the optimal method are given briefly, and then we discuss how different variations on the method affect accuracy.

4.1 Main method

4.1.1 Characterization of Success cases

4.1.2 Characterization of Failure cases

4.2 Variations

4.2.1 Keypoint Extraction

4.2.1.1 Keypoint Extractor

In this section, we detail the training results of the fluke keypoint extractor, and note issues with it. We also show the variation in accuracy between keypoint extractors run on different sized inputs.

4.2.1.2 Manual versus Automatic

One interesting note is to see how the final keypoint extractor compares with the manual annotations provided for the dataset. In Figure (TODO), we can see that when using manual annotations, we get a slight boost in accuracy using the notch as a control point. However, when the keypoint extractor network takes over, the inaccuracies in the notch prediction produce worse trailing edge artifacts than inaccuracies in the left / right tips — so it's better to ignore it.

It must be stated that some of these manual annotations made it into the keypoint extractor's training set, however the vast majority of its training set comes from an external dataset. Additionally, we can see from its training results that the keypoint extractor generalizes well regardless.

4.2.1.3 Varying training image sizes

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

As seen in Figure (TOD), resizing all images to 256×256 achieves the highest accuracy.

4.2.1.4 STN

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

4.2.2 Image Preprocessing

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

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

4.2.2.1 Cropping and Image Width

While, with dynamic time warping, we theoretically can match sequences of similar or different lengths, the distances are distorted by large differences in actual trailing edge length. Since we are only interested in the width of an image (assuming

that the trailing edge is roughly horizontal in the image), we can get every trailing edge to have exactly some fixed length w by the following process

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

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

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

Trailing Edge Length Determining what w should be is not super obvious. Ideally, one would simply look at the mode or average post-crop width, and try to keep w around there.

We can see in Figure (TODO vary_crop_size.png) that the optimal post-crop w for this dataset (of the few we evaluated) is 750 pixels. Figure (TODO te_size_hist_fb.png) shows the histogram of post-crop widthsi showing that 750 is a good width, which is what we use as a default.

Intuitively, this implies that the less the image has to be resized, the better the trailing edge.

4.2.3 Trailing Edge Extraction

There are a lot of variants on trailing edge scoring networks that were attempted, often with mixed results. One major result that we found was that, when using the averaging method to combine the trailing edge scores with N_y, having a robust trailing edge prediction wasn't as important as having a detailed trailing edge.

4.2.3.1 Trailing Edge Scorer variations

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

Overall, we were unable to get a significant improvement from any trailing edge scorer besides the 'Simple' all convolutional scorer. We theorize that this is because it could correct some mistakes made by the gradient only method (although it made many of the same mistakes itself) while providing detailed trailing edges.

More importantly, it appears that blocky trailing edge scores (such as those produced by any of the networks that are required to upsample their predictions) result in blocky trailing edges, which result in lower quality matches.

4.2.3.2 Combining N_y and T_y

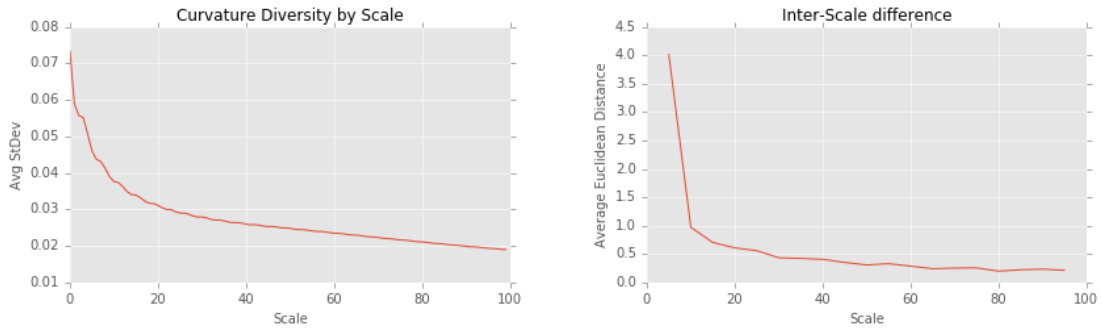
We only evaluate the 'average' method here, and show that the mixing parameter β has a significant effect on matching accuracy.

In Table (TODO: generate cartesian product of score weights and scorers), 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.

4.2.3.3 Number of neighbors in the extraction

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

Ultimately, we can see in Figure (TODO: vary_neighbors.png) that limiting the number of neighbors to the immediate neighborhood (i.e. $n = 3$) produces a significant boost over a larger neighborhood. While the trailing edges extracted with $n = 3$ can be somewhat less detailed, they are less likely to go completely off course. We hypothesize that despite the trailing edges being less detailed, the



a **b**

Figure 4.1: Curvature Diversity. Left panel (a) shows the average standard deviation of the (fixed length) curvature at different scales. Right panel (b) shows the average Euclidean distance between successive scales of curvature

extraction artifacts (e.g. significant sections of the trailing edge where the bottom or top neighbor is consistently chosen) are produced in a way that is unique to the trailing edge.

4.2.4 Curvature Extraction

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

4.2.4.1 Different scales

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

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

We find that (as shown in the left panel of Figure 4.1) as block curvature scale

is increased, the diversity at any given point in the curvature (if it is of a fixed size) goes down drastically. As a result, we stick to the lower end of the scale, keeping the curvature scales measured below 15%.

We can also see on the right side of Figure 4.1 that successive curvature scales show bigger differences at lower scales than at higher scales, which reinforces this need, but that it's advantageous to make bigger jumps between scales to maximize diversity while minimizing computation time.

Based on the above, we evaluate scales that run from 1% to 15%, with varying levels of resolution (TODO: set this up and run it).

4.2.4.2 Number of scales

Increasing the number of scales measured could potentially give more detail to work with, however at some point successive scales will not produce a meaningful difference from other scales, an effect that will become more notable the larger the scales get. Additionally, beyond some percentage of the trailing edge the curvature that is extracted becomes very uniform, and thus useless for matching.

4.2.5 Dynamic Time Warp Matching

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

4.2.5.1 Weighting the different scales

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

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

Figure (TODO: vary_weight_import.png) shows that despite this effort, it appears that equally weighting each curvature scale provides the best performance.

4.2.5.2 Window size

In Figure (TODO: vary_window_size.png), we can see that if we decrease the window (i.e. the Sakoe-Chiba bound) size, at around 10% of the query trailing edge length we maintain the same accuracy as the full window (i.e. 100$), but below this accuracy is severely affected. Thus, we use this value for the window size so as to minimize computation time while maintaining the total accuracy.

Additionally, while it's possible for gross mismatches to occur from there being no window boundary, we can see from Figure (TODO same as above) that this does pose a problem for this dataset.

4.2.5.3 Aggregating over multiple trailing edges per identity

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

We find that the minimum distance version provides significantly better accuracy than the average distance. While this is a bit surprising given that most of the individuals in the dataset only have two images associated, there are still a significant amount of individuals for whom there are more than that. That said, the minimal distance version also is in line with the LNBNN decision criterion used by Hotspotter [9], and as such as theoretical backing.

4.3 In Combination with Hotspotter

4.3.1 Failure cases

4.3.2 Characterization of when to use which method

CHAPTER 5
Discussion

5.1 Issues with current method

5.2 Future work

5.3 Conclusion

REFERENCES

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

APPENDIX