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

Humpback whales (Megaptera novaeangliae) are an important part of our ocean's ecosystems [citation needed], and have historically been at risk for extinction [citation needed]. While they are currently rated as 'Least Concern' [citation needed], tracking their migration patterns is important for helping the (currently small) population grow [citation needed]. In order to discern these migration patterns, conservationists need to be able to track individual humpback whales [citation needed]. One of the easiest (and cheapest) ways to do this is to watch for their tails as they breach the water surface [citation needed], giving a clear view of what is known as a Humpback 'fluke'. These are often patterned and scarred in unique ways, allowing conservationists to identify individuals [citation needed]. However, until recently, most automated identification methods still rely on signficant manual effort to describe and identify the fluke, severely limiting the amount of humpbacks that can be tracked [citation needed?].

This thesis lays out a method that automates the identification of Humpback flukes directly from still images thereof, using the 'trailing edge' of the fluke. Using this method, we achieve a fairly high top-1 ranking accuracy on a large dataset (consisting of about 400 identified individuals). We also show that this method significantly helps the accuracy of a pure appearance based method, Hotspotter [citation needed], giving 89% top-1 accuracy.

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

TODO: Put in citations

CHAPTER 1
Introduction

1.1 Humpback Whales

TODO: Put in something about the importance of identifying humpback whales, with citations and all

1.2 Current Identification Methods

Photo-identification of Humpback whale flukes thas been attempted since the early 90s [23] using computational aid. While early efforts mostly relied on a manual description of the fluke that would then be matched, later efforts have involved matching flukes based on automated analysis of both the patterns on the ventral side of the fluke. It is worth noting however that as shown in [3], the trailing edge changes less with age than surface patterns on the fluke, which means that it can (potentially) be a more reliable identifier over time. That said, trailing edges are hard to photograph well, requiring high resolution imagery and a consistent angle between fluke and photographer.

Each of these annotation methods can be separated into three categories:

- Manually annotated – a human must manually annotate or catalogue features in the fluke
- 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

1.2.1 Based on Trailing Edge

In [23], information about the trailing edge and fluke patterns are manually catalogued and used to match individual whales. This falls under a manual an-

notation approach. In the I3S contour system [12], the user must input start and end points on the contour, which are then extracted and checked manually, giving a semi-automated system. 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 [15], using a technique that is originally designed for Great White Sharks. However the results published on that technique are for a much smaller dataset than the one worked on in this thesis. 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 [14], [2] [37], with varying levels of manual effort.

1.2.2 Based on general Fluke appearance

The primary method for identifying Humpback whale flukes is to use the ventral fluke pattern, as seen in [23], [4], [3], [11], and (in an semi-automated fashion) [17].

1.2.2.1 Hotspotter

Hotspotter [7] is an automated photo-identification algorithm based on SIFT features that has been used in identifying Grevy's Zebras, Plains Zebras, Giraffes, and Elephants [27]. This work is the first to our knowledge of Hotspotter being applied to Humpback whale flukes, and the results are presented in chapter 4.

1.3 The Dataset

TODO: Get information from Jason (and Ted?) about how the dataset(s) I'm using were collected and sourced.

1.4 Method Overview

The method for trailing edge identification put forth in this thesis is fully automated, requiring no human annotation when used (although manual annotation is necessary for training the machine learning models used). On its own, it achieves decent results on a (relatively) large dataset, comparable with the fully automated

method used in [15]. Ultimately we suggest that this method be used in combination with an automated pattern matching method (e.g. Hotspotter) to provide very high accuracy matches. We also explore alternative methods based on more recent advances in deep learning for identification, however it appears that the dataset is too small to properly train these methods.

CHAPTER 2
Background

In this chapter, we provide a series of sections detailing background information on the algorithms from which this method 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 [18], [36], image segmentation [22], [5] and individual identification (specifically for human faces) [9], [32].

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, providing a meaningful prior to neural networks when applied to data with spatial invariances (e.g. image data). Convolutional networks have been around for a long time, but the current incarnation of these networks can be traced back to [20]. More recently however, convolutional networks have grown deeper and are often modeled after the architectural decisions made in [34], [33], and [18]. These decisions are specifically the use of Rectified Linear Units (ReLUs) as activation functions, Dropout [13] after fully connected layers for regularization, and small square kernels with "same" padding alternated with 2x downsampling layers (max pool).

The convolutional networks used in this thesis use the above architectural decisions, and also use batch normalization [16] at every layer.

2.1.1 Facial Keypoint Prediction

Facial keypoints are used in a lot of identification pipelines, as well as in motion capture and expression recognition. There has been recent work in using convolutional networks for facial keypoint prediction [35], [26].

In this work, fluke keypoints are predicted using a very similar technique but

with different underlying convolutional architectures. The essential idea is that the convolutional network predicts points (rather than classifications) in the form of (x, y) coordinates, and is treated as a regression network (i.e. the RMSE loss is used). This will be explained in more detail in the next chapter.

2.1.2 Fully Convolutional Networks

Classically, convolutional networks for classification predict a single class for an image, ignoring the possibility of multiple objects of varying classes being present throughout the image. These classification networks usually have fully connected (or dense) layers towards the end, which forces the size of the network input to be fixed. However, this allows the network to make learned decisions over the entire input without any sort of spatial pooling. When dealing with arbitrarily sized images, it is typical to use networks that are 'fully convolutional', in which case the entire network consists of convolution kernels. This technique can also be used for segmentation, as we can simply replace the dense part of the network with convolutional parts that can make class predictions on every pixel of their input. By upsampling and combining different stages of prediction, the authors in [22] produce high quality image segmentations.

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. This is explained in greater detail in the next chapter.

2.1.3 Embedding Networks

One major difference between the way that individual identification is done with convolutional networks and the standard classification architecture technique is the way that the error to the network is represented. When convolutional networks are used for classification tasks, the standard approach is to use a softmax output layer and the loss function is the cross-entropy loss. While a individual identification task could be expressed as a classification task, it becomes a major issue when a new individual is added to the dataset (which for our task is very common). A better notion for the loss function is to make it teach the network to 'embed' images into some d-dimensional vector (usually constrained to be unit norm), and group images

of the same individual closer than those of different individuals. The most common approach to this is to use the contrastive loss [9] [6], although more recently the triplet loss has risen in popularity [32] [28].

These approaches will be explained in more detail in the next chapter.

2.2 Seam Carving

Seam carving is a technique that uses image saliency information to resize images without warping or distorting the objects shown in the image [1]. This technique uses dynamic programming 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 based on a single iteration of the seam carving algorithm, using gradient information.

2.3 Curvature Measures

Contour curvature allows one to characterize the overall shape of an contour by looking at its edge. A lot of work has been done on using curvature information for detection [24], classification [10], and identification [19]. 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 measures the angle of the tangent normal of the gradient at each point in an image [10]. 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 [29], 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. In this work, we approximate the circular curvature with a square, which appears to perform just as well but can be computed faster. This is further explained in the next chapter.

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 distances. 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 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 [31] [21], however we only use the Sakoe-Chiba bound [30], 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 [25], 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 detail the steps that go into automating the extraction of high quality trailing edges, while trying to minimize human intervention.

3.1.1 Basic Contour Extraction Algorithm

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

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

$$N_y = \frac{I_y - min(I_y)}{max(I_y) - min(I_y)} \tag{3.1}$$

With N_y, we then need a starting point and ending point for the algorithm, denoted s and e respectively. For our purposes, we use the left and right tips of the fluke as our start and end. We then take the columns corresponding to these points and set every pixel in $N_y(s_x, \cdot)$ (i.e. the column s_x) and $N_y(e_x, \cdot)$ to ∞, and then set those points in N_y to 0. This forces the path to start and end at these points (as we are finding the minimal path).

The minimal path is then found by, scanning each column from left to right, and for each pixel in a column we set its cost with the update rule

$$C(x, y) = min^{y+n}_{y_c=(y-n)}(C(x-1, y_c) + N_y(x, y)) \tag{3.2}$$

Where n is a neighborhood constraint which we default to 2, meaning that each pixel considers 5 'neighbors' in the previous column. If y_c is out of bounds, then we have $C(x-1, y_c) = \infty$, and if $x - 1 < 0$, we set $C(x-1, \cdot)$ to 0.

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 3.2. 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. When using manual annotations in evaluating this algorithm, we use the bottom of the notch as a 'control point' as well.

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 (obviously) not a robust algorithm for finding trailing edges, something that we will fix later on.

3.1.2 Fluke keypoint prediction

One major issue with automation and this algorithm is that it requires manual annotation in the selection of the starting and ending points (i.e., the fluke keypoints). To work around this, we propose a convolutional network that predcits these tip points as part of this identification pipeline.

The convolutional network does not need full-sized images, so the first step of the keypoint extraction pipeline is to resize the image to 128×128. This size choice is somewhat arbitrary, but it provides the best performance without using an unnecessary amount of GPU memory. The network then predicts each of three points (left, right, and notch point – the last of which is optional to the trailing edge extraction) in the range $[0 - 1]$ for both x and y. This is then rescaled back up to the original image size. The evaluated network architectures are detailed in Table 3.1.

3.1.2.1 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. This is somewhat similar to VGG-16, although with half the trainable layers. After a 32x downsample has been achieved, there are two dense layers with a small number of units. One major difference in this work is that for predicting three points, we find that decoupling the final dense layer into three separate (smaller) layers (one for each point to predict) gives better performance than predicting them all from one final dense layer. We theorize that this may be because shared biases between each of the three predictions leads to stronger correlations between them, reducing overall flexibility and performance.

3.1.2.2 Training Details

Generating the training data for this is straightforward given a set of annotations with the associated points to learn. 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 (128×128 as above), 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

$$SE(\vec{t},\vec{p},\vec{s},\alpha) = \|(\alpha * \vec{s}) \odot (\vec{t}-\vec{p})\| \tag{3.3}$$

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 100 epochs with α set to 2e$-$2, using Nesterov

momentum with a learning rate of 1e−1 and $l2$ regularization on the 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.

3.1.2.3 Evaluation

On average, the best network achieved a 20 pixel distance error on the validation and testing sets. While this may seem like a lot, the trailing edge extraction (and subsequent matching performance) was not severely affected when only using the start and end point predictions.

Additionally, as can be seen in Figure 3.2, the vast majority of images have a low pixel distance error, while there are a few that have higher error. Qualitative inspection of these images shows that they are of flukes which are not the singular or major object in the image. Ideally we could train the convolutional network to more robustly handle these cases, but since the majority of the training data have the fluke centered and focused, the network heavily biases towards handling these cases.

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. 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 an output shape that is the same as the input shape, obviating the need for any resizing.

There are three major variants on trailing edge scoring that are detailed below, however all three function on the same principle of taking an arbitrarily sized image

and producing a score map of each pixel to how likely it is to be part of a trailing edge.

The dataset was sourced from existing trailing edges extracted using the gradient based algorithm, however with manual adjustments to fix many of the more common issues we encountered. For the training set, we then extract a 128×128 patch at 128 pixel intervals along the trailing edge (a positive patch), and a corresponding negative patch (with no trailing edge pixels) randomly sampled from the left over space above and below the trailing edge. These are then randomly sampled into training, testing, and validation sets.

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

Simple This network is simply a stack of "same" convolutional layers, 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 [22], 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 [22], 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.

3.1.3.2 Using the trailing edge scores

Once we have the map indicating 'trailing edginess', 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 $T_p \in [0-1]^{w \times h}$ 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 some weight β.

$$S_{te} = (1-\beta) * N_y + \beta * (1 - T_p) \tag{3.4}$$

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

3.2 Trailing Edge Matching

Once these trailing edges are extracted, we can use a few simple methods to compare them in search of a trailing edge that is close to the one we are finding a match for. The simplest way to do this is to define a comparator between any two given trailing edges. The best method for this that we have tried is to extract block integral curvature at several scales and then use Euclidean distance as a point-to-point distance measure in a dynamic time warping algorithm. Block integral curvature approximates true integral curvature while allowing for a much faster computation time. This approximation is made better by being taken at multiple scales.

3.2.1 Curvature Measurement

To extract the curvature, we take the trailing edge (which is a set of coordinates into an image) and a zero-image I_0. Each pixel corresponding to and below (in the vertical direction) the trailing edge in I_0 are set to 1. Because the trailing edge extractor produces only one coordinate per column, we can do this safely. Once this is done, we calculate a summed area table [8] ST from I_0 as follows.

$$ST(x, y) = \sum_{i=0}^{i=y} \sum_{j=0}^{j=x} I_0 \tag{3.5}$$

Conceptually, the next step is to slide a square of size s centered on each point, and measure the percentage of that square that is within the filled in trailing edge. To do this, we compute $BC_s(x, y)$ for each (x, y) coordinate in the trailing edge.

$$b_x = x - \frac{s}{2} \tag{3.6}$$

$$b_y = y - \frac{s}{2} \tag{3.7}$$

$$e_x = x + \frac{s}{2} \tag{3.8}$$

$$e_y = y + \frac{s}{2} \tag{3.9}$$

$$BC_s(x, y) = \frac{(ST(b_x, b_y) + ST(e_x, e_y)) - (ST(b_x, e_y) + ST(e_x, b_y))}{s^2} \tag{3.10}$$

The set of scales to choose presents a large parameter space, however we have found that the scales $S = [5, 10, 15, 20]$ work well for our purposes. We then treat this curvature measurement as a $\|S\| \times x$ matrix BC.

3.2.2 Sequence Matching

Given two sequences of curvatures BC_1 and BC_2, we match them using dynamic time warping as follows. First, we create a cost matrix C of size $x_1 \times x_2$ (where x is the number of points in the original trailing edge). We initialize this cost matrix by setting the first column and row to ∞, and then $C(0, 0) = 0$. Then, for each element in the cost matrix starting with $C(1, 1)$, we use the following update rule

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

$$C(i, j) = D(x_1(i), x_2(j)) + min(C(i-1, j), C(i, j-1), C(i-1, j-1)) \tag{3.12}$$

Additionally, we impose the Sakoe-Chiba [30] locality constraint w so that for each element i in BC_1, we only consider the range over elements j in BC_2 of $j \in [min(i-w, 0), max(i+w, x_2)]$.

For most of these experiments w is set to 50, which appears to minimize the time taken for each comparison while preserving the overall accuracy of the algorithm.

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 Embedding via Convolutional Networks

3.3.1.1 Raw Images

3.3.1.2 Trailing Edges

3.3.2 Aligning Trailing Edges

3.3.2.1 Keypoint Alignment

3.3.2.2 Dynamic Time Warping Alignment

3.3.3 Histogram Matching

CHAPTER 4
Results

4.1 Main method

4.1.1 Characterization of Success cases

4.1.2 Characterization of Failure cases

4.1.3 Variations

4.2 In Combination with Hotspotter

4.2.1 Failure cases

CHAPTER 5
Discussion

5.1 Issues with current method

5.2 Future work

5.3 Conclusion

REFERENCES

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

[2] 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.

[3] 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.

[4] C. A. Carlson, C. A. Mayo, and H. Whitehead. Changes in the ventral fluke pattern of the humpback whale (megaptera novaeangliae), and its effect on matching; evaluation of its significance to photo-identification research. *Rep. int. Whal. Commn (special issue)*, 12:105–11, 1990.

[5] 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.

[6] S. Chopra, R. Hadsell, and Y. LeCun. Learning a similarity metric discriminatively, with application to face verification. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 539–546. IEEE, 2005.

[7] 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.

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

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

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

[11] N. Friday, T. D. Smith, P. T. Stevick, and J. Allen. Measurement of photographic quality and individual distinctiveness for the photographic identification of humpback whales, megaptera novaeangliae. *Marine Mammal Science*, 16(2):355–374, 2000.

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

[13] 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.

[14] 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.

[15] 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.

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

[17] 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.

[18] 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.

[19] 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.

[20] 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.

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

[22] 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.

[23] 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.

[24] 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.

[25] 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.

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

[27] J. R. Parham. Photographic censusing of zebra and giraffe in the nairobi national park.

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

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

[30] 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.

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

[32] 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.

[33] 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.

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

[35] 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.

[36] 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.

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

APPENDIX