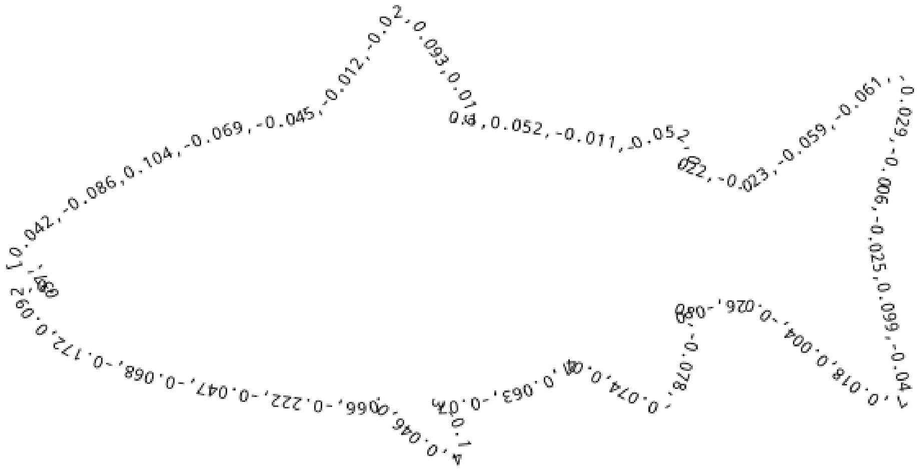
Espen Meidell
Edvard Schreiner Sjøblom

FishNet: A Unified Embedding for Salmon Recognition

Master's thesis in Artificial Intelligence
Supervisor: Kerstin Bach
June 2019

Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science



Espen Meidell
Edvard Schreiner Sjøblom

FishNet: A Unified Embedding for Salmon Recognition

Master's thesis in Artificial Intelligence
Supervisor: Kerstin Bach
June 2019

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU
Norwegian University of
Science and Technology

Abstract

The current methods of individual salmon tagging and tracking rely on physical interaction with the fish. This process is inefficient, and can cause physical harm and stress for the salmon. The use of deep learning techniques has shown great advances in the field of human face recognition. In this thesis we describe a system, FishNet, that applies one of these techniques to the field of salmon recognition.

The system learns a mapping from images of salmon heads to a compact vector embedding, where similarities between embeddings correspond to similarities between salmon. Convolutional neural networks are used to directly optimize the embeddings using tiplets of images. The system does not make any assumptions as to which parts of the head that are useful for distinguishing them from each other.

The experiments show that modern face recognition techniques work well on salmon, and that good performance can be achieved with relatively small neural network models. With a false positive rate of 1%, FishNet achieves a true positive rate of **96%**.

This thesis presents the first use of end-to-end deep learning techniques in the field of salmon recognition. The images used are from cameras deployed in salmon farming pens, an environment with many variable factors influencing the image quality.

Sammendrag

Dagens metoder for markering og sporing av oppdrettslaks baserer seg på fysisk kontakt med fisken. Denne prosessen er både ineffektiv, stressende og potensielt skadelig for laksen. Bruken av dyp læring har de siste årene gjort store fremskritt innenfor ansiktsgjenkjennelse hos mennesker. I denne oppgaven beskriver vi FishNet, et system som bruker dyp læring til individgjenkjenning av laks.

Systemet lærer en funksjon fra bilder av laksehoder til en kompakt vektor, der likhet mellom vektorer tilsvarer likheter mellom laksen. Konvolusjonsnett brukes til å direkte optimalisere disse vektorene ved hjelp av tripler av bilder. Systemet gjør ingen antagelser om hvilke deler av laksens hode som brukes til å skille dem fra hverandre.

Eksperimentene viser at teknikkene som brukes i moderne ansiktsgjenkjenningsprogrammer fungerer godt på laks, og at man kan få god ytelse med relativt små nevrale nett. Med en falsk positiv rate på 1% har FishNet en sann positiv rate på **96%**.

Denne oppgaven beskriver det første systemet som bruker ende til ende dyp læring for individgjenkjenning hos laks. Bildene brukt i oppgaven er fra kameraer i oppdrettsmærer, et miljø med mange varierende faktorer som påvirker bildekvaliteten.

Preface

This thesis was written during the fall of 2018 and spring of 2019 at the Norwegian University of Science and Technology (NTNU), Faculty of Information Technology and Electrical Engineering, Department of Computer Science. The thesis was accomplished in cooperation with Sealab AS.

We would like to thank Sealab AS for an exciting challenge, and for generously providing us with office space during our thesis. We would also like to thank our supervisors for their great help and motivation.

Supervisor: Kerstin Bach
Co-Supervisors: Håkon Måløy & Bjørn Magnus Mathisen

Espen Meidell, Edvard Schreiner Sjøblom
Trondheim, June 7, 2019

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This introduction begins by presenting the motivation behind this thesis. Next, we present the goals and research questions, and introduce the research methodology applied in answering the research questions. Finally, we present our contributions before we finish off with a brief structural overview of the remaining thesis.

1.1 Motivation

The Atlantic salmon (Salmo Salar) farming industry in Norway has experienced a massive growth in the past four decades. The industry has gone from producing 4.300 tonnes of salmon in 1980, to almost 1.240.000 tonnes in 2017 (SSB [2018]). Figure 1.1 shows the development of produced salmon in Norway. In 2017, the total economical results from salmon production was calculated to be over 61 billion Norwegian kroner (NOK) (SSB [2018]). This makes salmon farming one of the most profitable industries in Norway, and it is considered as one of the most important industries in a post oil Norway (Richardsen et al. [2018]). A small improvement in efficiency of this industry would result in a huge economical profit.

At the same time there are many environmental and animal welfare related challenges in the salmon farming industry. Activities such as delousing and feeding are known to greatly impact wildlife in the waters around the farming facilities. In order to reach a state of sustainable production of salmon, more insight in the salmon life is necessary.

Being able to track salmon at an individual level would enable tracking a single individual throughout their lifespan, from smolt to finished product enabling linking salmon fillets to the life-story of the individual. The producers could also optimize their production facilities and processes at a much more granular level. Possibilities that open up are monitoring individual weight development, treating salmon only when the need applies, delousing only the individuals that suffer from

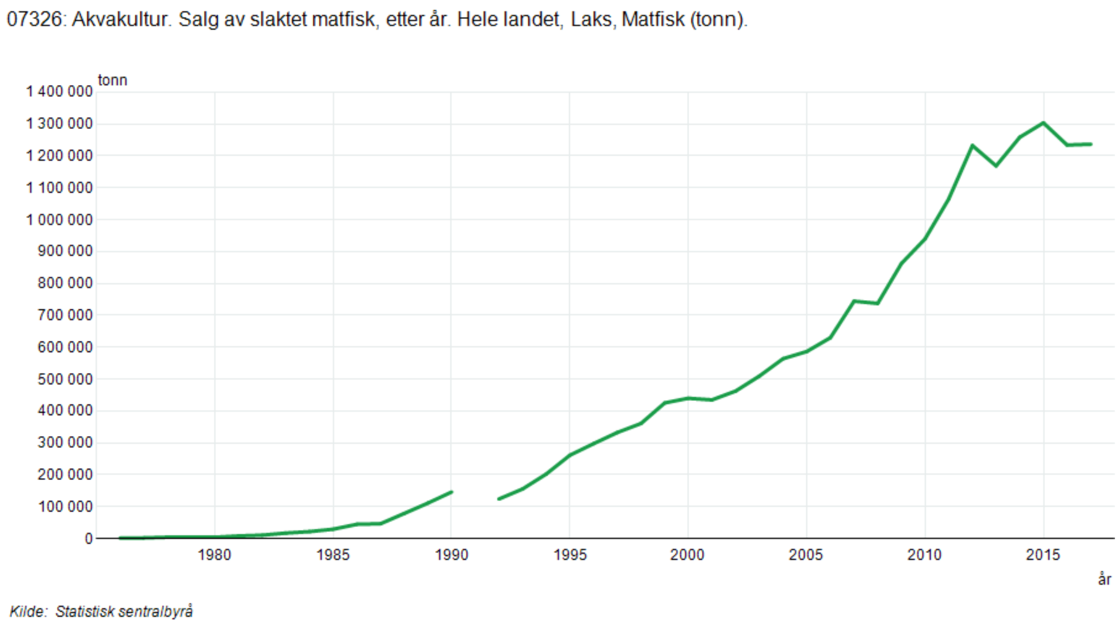


Figure 1.1: The sale of slaughtered salmon in tonnes from 1976 to 2017. The data from 1991 is missing[1].

lice, and thereby preventing unnecessary harm to healthy salmon. Being able to separate the individuals would also allow monitoring the health of a salmon cage at a more accurate level, for instance when performing a lice count of a salmon cage.

With regards to insight in the salmons life, individual tracking would open the doors for many interesting research areas that require monitoring of individuals over time such as feeding behaviour, behaviour of lice, social hierarchy and overall survival.

Tagging individual fish has been performed through a variety of methods, but the techniques have almost exclusively relied on physical engagement with the salmon. The techniques include surgical implantation of tags and external mutilation, such as fin-clipping, freeze brands, tattoos, visible implant tags, and external tag identifiers attached by metal wire, plastic, or string (Merz et al. [2012]). This is a problem both from an animal welfare and product quality perspective. Bacterial growth and unpleasant sensory properties has shown to increase more quickly in salmon experiencing stress in their lifetime prior to being slaughtered. This results in reduced shelf life of the finished product (Hansen AÅ [2012]). A computer vision

[1]Figure generated using SSBs chart generator at `https://www.ssb.no/statbank/table/07326/chartViewLine/`

method for uniquely identifying individuals would elegantly solve this problem by minimizing the impacts from invasive techniques.

In the last few years deep learning approaches such as DeepFace (Taigman et al. [2014]) and FaceNet (Schroff et al. [2015]) have shown great advances in the field of face verification on humans. The systems are able to verify the identity of people in images as well as humans are. They have also shown to be robust with regards to changing lighting conditions and other variations influencing how a person looks.

As of this writing, Sealab AS has over 700.000.000 images taken inside salmon cages, that in combination with known deep learning approaches of face verification could produce good results in the domain of individually identifying salmon.

In this thesis we want to investigate if the methods that have proven to be so successful in human facial recognition can perform well on salmon.

1.2 Goals and Research Questions

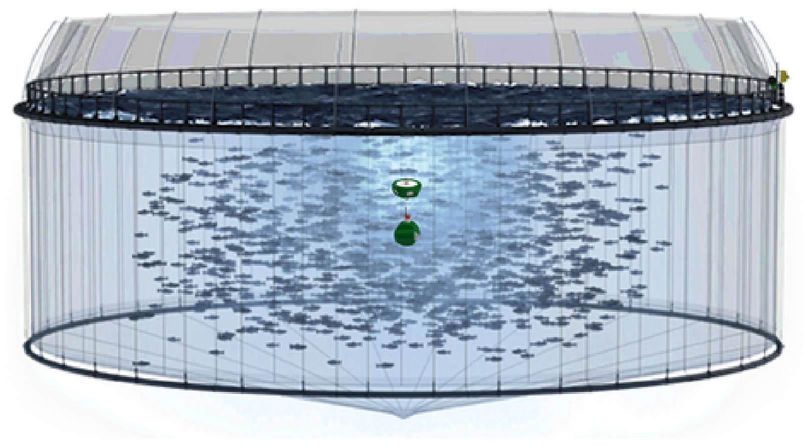


Figure 1.2: The Sealab AS set-up. A high-definition camera is attached to a wire. 70cm above the camera, a dimmable artificial light source with maximum effect of 40 000 lumen is attached. The wire is attached to a winch at the surface that enables the camera rig to move in the 3D-space inside the salmon cage. Illustration by Oscar Marković, co-founder of Sealab AS.

What separates individual salmon recognition from human face recognition is that images of salmon are typically taken of the side of the salmon. The recognition of a salmon is therefore based on one side. Some characteristics of salmon are not symmetrical, so each side of the salmon is unique. To create a model that identifies a salmon from either side would require a camera on each side of the salmon and then train both sides to identify the same salmon. Data from Sealab is based on a single camera, so one side of the salmon is recorded at a time. See Figure 1.2 for an illustration of the set-up. Given this is the data we have available, our goal is of the side-recognition nature. The goal of this thesis is to investigate the feasibility of using deep learning to uniquely identify individual *sides* of salmon from real video footage in the salmon cages.

To aid the direction of the research we have identified the following research questions:

Research question 1 *How well do state-of-the-art face verification methods using end-to-end deep learning perform on salmon in cages?*

Research question 2 *How suitable are different deep learning architectures and what results do they yield?*

Using end-to-end deep learning, we do not have to make assumption as to what it is that uniquely separates each individual salmon, but the melanophore pattern on the side of each salmon head is assumed to serve as a fingerprint for the salmon. Figure 1.3 shows the melanophore pattern on the side of a salmon head. Existing research shows that patterns on the head region of salmon can be used for long-term individual recognition (Merz et al. [2012], Stien et al. [2017]). Due to the crowded conditions in the sea pens, it makes sense to use images of the head region than the entire salmon. Getting a picture of the entire salmon is hard, as parts of the body are often covered by other salmon. Therefore, we base the recognition solely on the head region, instead of the salmon as a whole.

When designing deep neural network models, there are many architectural choices that can be made influencing the models representational capacity and how well it is able to learn. These choices have major influence on the overall performance of the system. This leads to research question 2.

1.3 Research Method

The thesis consists of two phases. The first phase consisted of a structured literature review into the field of face recognition and salmon recognition. The literature review is followed by an applied research phase where we implement a system for salmon recognition. In our applied research, we want to find out how deep neural networks perform on identification of salmon. This technique-driven approach will produce a system where the complexity, underlying models and theories means

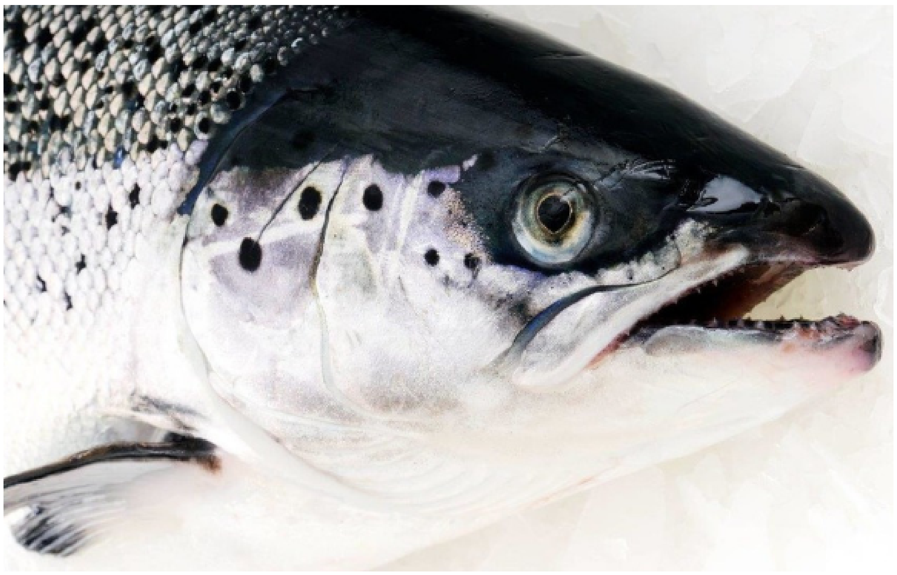


Figure 1.3: Head region of an Atlantic salmon. The dark spots on the side of the salmon head is the melanophore pattern. The pattern is unique for each individual. Figure reprinted from Eilertsen [2017].

that experimental implementation is necessary to evaluate our model and theory behind it. Our methodology is based on the framework for IT research proposed by March and Smith (March and Smith [1995]). The model is based on four natural research activities: build, evaluate, theorize and justify. This experimental approach suits us in achieving a result, as well as seeking to understanding the reason for the results achieved.

1.4 Contributions

The main contribution of this work is the investigation into how well state-of-the-art deep learning methods work in learning embeddings of salmon heads in euclidean space, where the distances between the embeddings can be used to measure similarity. The data used is from relatively uncontrolled conditions in sea cages. Systems that operate in these conditions will have to tolerate a large variety of factors such as weather and water conditions that influence image quality. This is important for creating a robust system that works in the real world. By using deep learning models to learn the representation of salmon, we remove the need for a human to decide on what features are important for the task at hand.

If the techniques that have proven themselves for human face recognition also work well on salmon, it may open up interesting possibilities in other domains where pattern recognition and matching is useful.

1.5 Thesis Structure

Chapter 2 provides the reader with an understanding of the necessary theory used in this master thesis. Section 2.3 presents some of the related work that has influenced our approach. This includes face verification and other attempts at uniquely identifying salmons. Section 2.4 presents our literature review protocol.

Chapter 3 presents how we have processed and structured our training, validation, and test data. We also present the FishNet approach with different architectures.

Chapter 4 describes how we evaluated the models presented in Chapter 3. We also present the results from the evaluations.

In Chapter 5 we discuss and compare the results we achieved. We also discuss the impact and limitations of our results, and what contributions are made. Finally we discuss potential future work.

Chapter 2

Background Theory

In this chapter we present the background and theory necessary for the reader to understand the rest of the thesis. We will start by presenting a clustering method that we utilized in the creation of our dataset. We also explain two methods for dimensionality reduction used to examine our results. Further, we will cover deep learning and its use for computer vision in section 2.2. We then present existing research into individual salmon tracking, and how deep learning can be used for face verification in section 2.3. Finally we will present the structured literature review protocol in section 2.4.

2.1 Clustering and Dimensionality Reduction

Clustering is a technique that groups similar data points together in separate clusters. Clustering algorithms utilizes a *distance function* that defines the distance between two data-points. The data points with a small distance between them are marked to belong in the same cluster. There are several techniques and methods on clustering. Some use a parameter to define how many clusters the data should be divided in, others use a threshold value to restrict what distances are regarded as being in the same cluster. This section will present one clustering technique, DBSCAN, used in the creation of our dataset.

Dimensionality reduction is the process of creating a new, lower-dimensional representation of a dataset while maintaining as much information as possible about the original data. This section will also present two dimensionality reduction techniques that we used to visualize our results; Principal Component Analysis (PCA) and t-Distributed Stochastic Neighbour Embedding (t-SNE).

2.1.1 Density-Based Spatial Clustering of Applications with Noise

Density-Based Spatial Clustering of Applications with Noise (DBSCAN) is an unsupervised machine learning technique for data clustering. The algorithm groups data points that are close together in clusters, and does not require the number of clusters as input (Ester et al. [1996]). DBSCAN has two hyperparameters:

eps is the neighbourhood radius. If point b is within a radius of *eps* from point a, then a and b are neighbours.

minPts is the number of neighbours a point must have to be considered a *core point*.

Data points are divided into three categories:

Core points are points that have *minPts* neighbours within a radius of *eps*.

Edge points are points that are within *eps* distance of a core point, but is not a core point by itself.

Noise are points without neighbours.

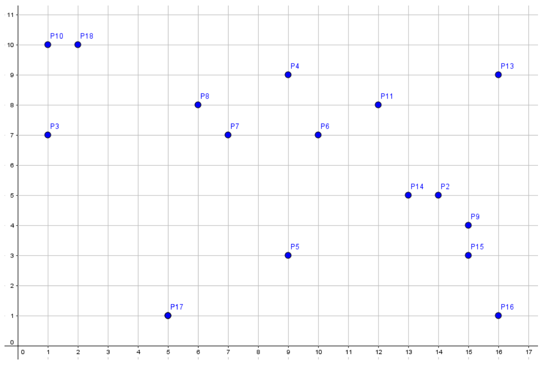
Each cluster formed by DBSCAN consists of one or more *core points*, as well as all points that are *reachable* from a *core point* in the cluster. Figure 2.1 shows the DBSCAN algorithm step-by-step on an arbitrary example.

2.1.2 PCA

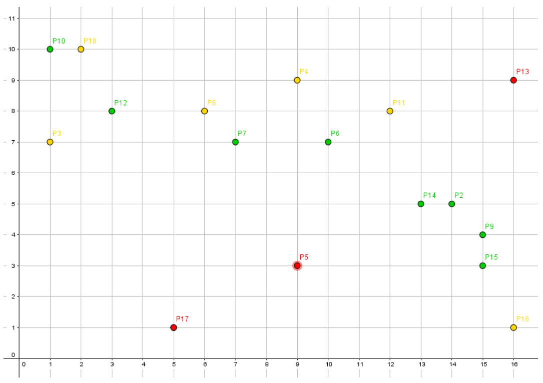
Principal Component Analysis (PCA) computes uncorrelated linear combinations of the original, possibly correlated features. These new variables are called principal components. When computing the components, most variance is stored in the first component, second most variance in the second component and so on (Hotelling [1933]). This means that we can select the first two or three principal components and visualize them, representing as much of the original dataset as possible.

2.1.3 t-SNE

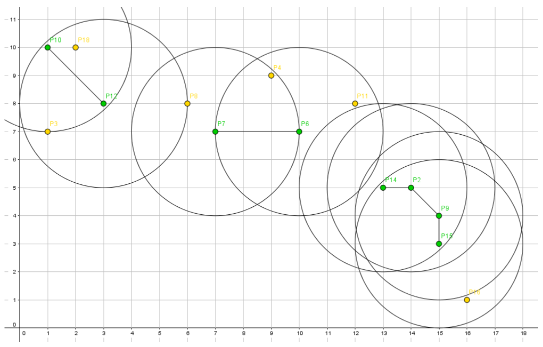
t-Distributed Stochastic Neighbour Embedding (t-SNE) is a dimensionality reduction technique that is commonly used for visualization of high-dimensional data (Maaten and Hinton [2008]). Useful features of t-SNE, is that it is capable of



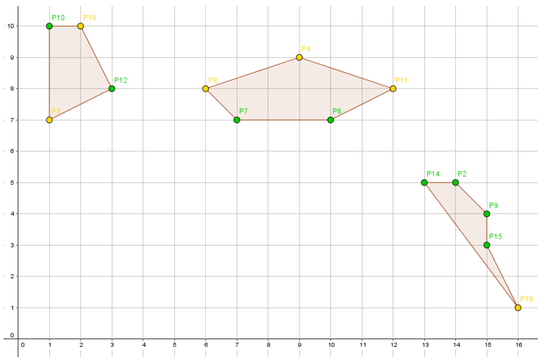
(a) Points plotted in 2D euclidean space.



(b) The points are classified into core (green), border (yellow) and noise (red).



(c) Noise points are removed and edges are placed between core points that are within *eps* of each other.



(d) Connected points are made into clusters. Border points are added to the cluster of the closest associated core point.

Figure 2.1: An illustration of the DBSCAN algorithm with $eps = 3$ and $minPts = 4$.

retaining the local structure of data as well as visualizing global features such as clusters.

t-SNE represents distances between points as conditional probabilities. This is done for the original high-dimensional data and for the new low-dimensional representation (usually two or three dimensions). To measure the distance from point x, a probability distribution is centered at x, and the density is measured at all other points. For the high-dimensional data t-SNE uses a Gaussian distribution, and for the low-dimensional representation it uses a Student-t distribution.

Once the distances are represented as probability distributions, gradient descent can be used to minimize the Kullback-Leibler divergence. This is done by adjusting the values of the randomly initiated low-dimensional representation. The result is a low-dimensional representation that often visualizes important global features of the high-dimensional original data.

2.2 Deep Learning

In the field of artificial intelligence, a true challenge proved to be solving tasks that are easy for humans to perform, but hard to describe formally. Problems that cannot be described by a list of formal, mathematical rules such as recognizing spoken words or faces in images. Deep learning is an approach to provide solutions to these problems. The approach relies on the computer to learn from experience and learn real-world, complex concepts by dividing them into simpler ones. This approach removes the need for humans to formally specify all the knowledge needed by the computer. The learning is instead based on gathered knowledge from experience (Goodfellow et al. [2016] chapter 1).

2.2.1 Neural Networks

Neural networks are a class of machine learning models whose goal is to approximate some function f^* (Goodfellow et al. [2016] chapter 6). Neural networks consist of a graph of nodes, where the connections between nodes have learnable parameters $\boldsymbol{\theta}$. These parameters are called the weights $\boldsymbol{W}$ and biases $\boldsymbol{b}$ of the network. A neural network usually consists of an *input layer*, an *output layer*, and one or more *hidden layers*. The network defines the following mapping: $\hat{\boldsymbol{y}} = f(\boldsymbol{x}; \boldsymbol{\theta})$ providing an output $\hat{\boldsymbol{y}}$ for input $\boldsymbol{x}$, parameterized by $\boldsymbol{\theta}$. Figure 2.2 illustrates a simple neural network with two hidden layers. The network takes a 3-dimensional vector as input and produces a single numerical prediction.

The output $\boldsymbol{o}$ of a single node is computed as $\boldsymbol{o} = g(\boldsymbol{W}^\top \boldsymbol{x} + \boldsymbol{b})$ where g is the activation function. To enable the neural network to approximate any function, non-linear **activation functions** are added to each node in the network.

Training a Neural Network

Neural networks are usually trained using gradient based optimization of a loss function, such as gradient descent (Goodfellow et al. [2016] chapter 4). The neural network is used to calculate the result for some input data, for which the loss is calculated. Once the loss is known it is used to compute a gradient of the loss. This is done recursively backwards throughout the network using the chain rule of calculus. This algorithm is called the backpropagation algorithm. The gradient of the loss can then be used to update the parameters of the model (Goodfellow et al. [2016] chapter 6).

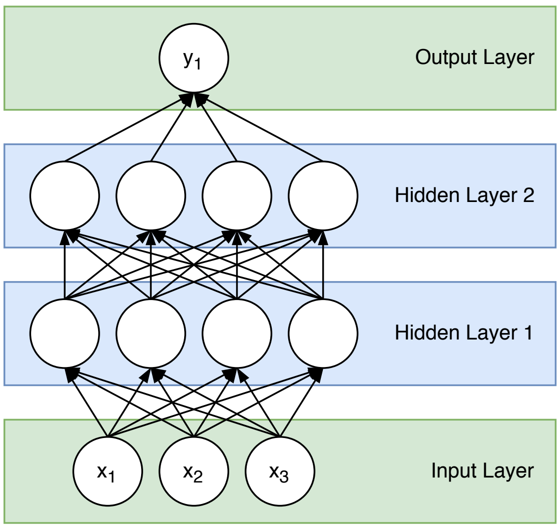


Figure 2.2: Illustration of a simple feed forward, fully connected neural network with two hidden layers. The network takes a 3-dimensional vector as input (bottom layer) and produces a single numerical prediction (top layer). The network is fully-connected, meaning each node connects to all the nodes in the next layer.

Due to the size of the training data required for training neural networks, it is impractical to use the entire training set to compute the gradient before updating the weights. To solve this issue, it is common to split the data in to smaller **mini-batches**, and compute the gradient of each mini-batch of size m, where m is significantly smaller than the size of the entire training set (Goodfellow et al. [2016] chapter 8). Gradient descent with mini-batches is called **stochastic gradient descent** (SDG) because the random sampling of each mini-batch introduces noise into the training process. When the training algorithm has seen each training sample once, it has finished one **epoch**. The network typically goes through several epochs of training to converge on a good set of parameters, but the number of epochs depends heavily on the amount of available training data.

SDG may be slow, and it is common to add **momentum** to the algorithm. Momentum influences how much the parameters are updated during training, and is calculated using a moving average of the gradients. Momentum helps us ”smooth” out some of the noise introduced by the random sampling of mini-batches, and it also helps the training algorithm navigate terrain where the gradient is much larger in one dimension than in another by preventing excessive oscillation.

The Loss Function

The loss function of a neural network measures how well the model performs on the given input. If the model makes wrong predictions, the loss function should return a large number. If it is correct, it should return a small number. The loss function defines a many dimensional loss landscape. By tuning the parameters $\boldsymbol{\theta}$ of the model, we can traverse the loss landscape and find the best parameters for our data. In other words, finding the combination of parameters that minimizes the loss of the model. This is done using an optimization algorithm.

There are two major classes of loss functions:

Regression losses where the model attempts to predict a continuous value.

Categorical losses where the model attempts to predict the output from a finite set of options.

Common loss functions include mean squared error (MSE) and cross-entropy loss. MSE calculates the mean of the squared difference between the predicted value $\hat{\boldsymbol{y}}$ and the ground truth $\boldsymbol{y}$ of m training samples (see equation 2.1). Equation 2.2 shows how the cross entropy is computed for a binary classification problem.

$$J(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m}\left(f(\boldsymbol{x}_i;\boldsymbol{\theta}) - \boldsymbol{y}_i\right)^2 \tag{2.1}$$

$$J(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m}\boldsymbol{y}_i \cdot \log\left(f\left(\boldsymbol{x}_i;\boldsymbol{\theta}\right)\right) + \left(1-\boldsymbol{y}_i\right)\cdot \log\left(1 - f\left(\boldsymbol{x}_i;\boldsymbol{\theta}\right)\right) \tag{2.2}$$

Since the loss function creates the loss landscape that the networks seeks to minimize, the choice of loss function has a vital role in how hard it is to train the network.

Activation Functions

The most used activation functions for deep learning applications are the rectified linear unit (ReLU), the logistic sigmoid, and the hyperbolic tangent. In practice, the most used activation function for hidden layers is ReLU.

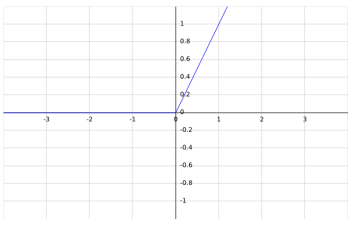
The **ReLU** function is defined as $g(x) = ReLU(x) = max(0, x)$. The ReLU function is identical to a linear unit, except that the output is 0 if the input is negative. This ensures large and consistent derivatives when the node is activated (Goodfellow et al. [2016] chapter 6). One problem with ReLU is that the node cannot learn if the output is zero (because the derivative is zero). This can also result in a node ”getting stuck” and never activating again. One variation on the ReLU function to combat this issue is the **leaky ReLU** function, which ensures there is a small derivative even if x is negative. Equation 2.3 defines the leaky ReLU function.

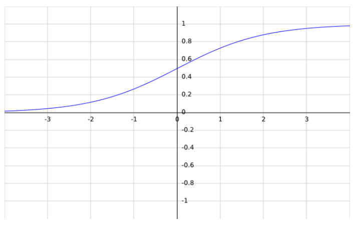
$$g(x) = \begin{cases} 0.01x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases} \tag{2.3}$$

The **sigmoid** function is defined as $g(x) = \sigma(x) = \frac{1}{1+e^{-x}}$. The function takes a number and squashes it into the range between 0 and 1. This can be especially useful in output nodes where the output represents a probability between 0 and 1. The sigmoid function is rarely used for hidden layers as its output saturates for very high and very low input values, while it remains very sensitive to its input when the input is close to zero. The saturation causes very small gradients and therefore very slow training.

The **tanh** function is similar to the sigmoid function, but it squashes the input into the range between -1 and 1. It is defined as $g(x) = \tanh(x)$. The tanh function usually performs better than the sigmoid function when training a neural network (Goodfellow et al. [2016] chapter 6).

See figure 2.3 for an illustration of the graphs the different activation functions produce.


(a) Rectified linear unit function


(b) Logistic sigmoid function.

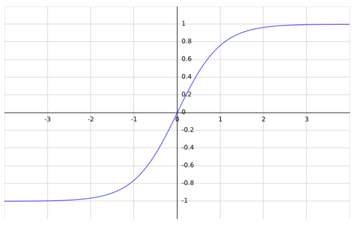

(c) Hyperbolic tangent function

Figure 2.3: The graphs of the most commonly used activation functions.

Dataset splits

Training a neural network requires the network to have a notion of how far away it was from the correct output in order to train the parameters to approximate this output. The most common way to train a neural network is with *supervised learning*. Supervised learning uses labeled examples to learn features from the input data.

Dataset splitting divides the data into three different sets; training set, validation set and test set. The separation percentage can vary, but a common separation is 80/10/10. 80% of the data as the training set, 10% as validation set and the remaining 10% as testing data.

Training set The training set is used to train the network. This partition is the largest, as we want the training to be based on as much data as possible. In stochastic gradient descent, this is the data that is split into mini-batches and trained on.

Validation set The validation set is used to measure the model's performance after each epoch. Using a validation set after each epoch is a good indicator on the performance the model would have on the test set. During the use of the validation set, the model is not allowed to update its parameters.

Test set The test set is used for the final test of the model. It is important that the test set is kept hidden from the system prior to testing. It acts as completely new data for the system, and is therefore a good indicator on how the system would perform in the real world.

Over- and underfitting

Overfitting occurs when the network becomes too familiar with a specific set of training data. An overfitted model could learn to map each example in the data to the labeled solution with 100% accuracy, but would perform poorly when seeing new data. The model has not learned general concepts, but rather knows the output for each training example. An overfitted model will have a large gap between the training and testing error.

Underfitting occurs when the model is unable to learn from the training data, and the training error remains high. This can occur if the model does not have the representational capacity to represent the underlying data distribution.

Regularization

The term regularization refers to strategies used to reduce the test error of a model (Goodfellow et al. [2016] chapter 7). This can often come at the expense of the training error, and many regularization techniques constrain the models or the parameters of a model. There is a large amount of different regularization techniques used in deep learning. Here we present some of the most commonly used techniques, and the techniques used in this thesis.

It is common to use **parameter norm penalties**, $\Omega(\boldsymbol{\theta})$, for the weights of a neural network. The most common norm to use is L^2 regularization, where $\Omega(\boldsymbol{\theta}) = \frac{1}{2}||\boldsymbol{w}||_2^2$. L^2 regularization gives a preference to simpler models (where the magnitude of weights are smaller).

Using **dataset augmentation** to improve how the generalization ability of the model is another simple and common approach to use, especially in computer vision tasks. By slightly altering the characteristics of the training data, one can greatly increase the volume of data. If the training data is images, operations such as translations, rotations, scaling and color adjustments are all techniques that are easy to implement.

Dropout is a technique where units in the network are disabled (multiplied with 0) with a certain probability during training (Srivastava et al. [2014]). This effectively simulates training with many slightly different neural networks, and it prevents the models from overfitting. During test time all units in the network are active, which approximates averaging the prediction of the simpler networks.

Batch normalization is another technique that is commonly used in neural networks (Ioffe and Szegedy [2015]). Batch normalization works by normalizing the inputs to a layer using the mean and standard deviation of the mini-batch. According to Santurkar et al. [2018], batch normalization works by smoothing out the loss landscape and making it more stable, thus making training significantly easier. Equation 2.4 shows how the batch normalization is computed (from Ioffe and Szegedy [2015]). The values γ and β are learned during normal training.

$$\begin{aligned}
\text{Input} &: \text{Values of } x \text{ over a mini-batch: } \mathcal{B} = \{x_{1\ldots m}\} \\
\text{Learnable parameters} &: \gamma, \beta \\
\text{Output} &: y_i = \mathrm{BN}_{\gamma,\beta}(x_i) \\
\mu_{\mathcal{B}} &\leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i, \text{ mini-batch mean} \\
\sigma_{\mathcal{B}}^2 &\leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2, \text{ mini-batch variance} \\
\widehat{x}_i &\leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}, \text{ normalize} \\
y_i &\leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i), \text{ scale and shift}
\end{aligned} \tag{2.4}$$

Transfer Learning

Transfer learning (Goodfellow et al. [2016] chapter 15) refers to the approach where features learned in one domain are used as a starting point for learning about another domain. This is especially useful if there is more training data available for the first domain than the second. For machine vision tasks transfer learning can be useful, because low level features such as shapes and edges are easily transferable between domains.

For commonly used deep learning models there are usually pre trained convolutional weights available for download. Using these can significantly increase training speed and improve generalization.

2.2.2 Convolutional Neural Networks

Convolutional neural networks (convnets) are special kinds of neural networks for processing input that has n-dimensional grid-like topology (Goodfellow et al. [2016] chapter 9). Images can be thought of as two dimensional grids of pixels, and convnets have been very successful in various computer vision applications such as image classification. A convolutional neural network usually uses a combination of convolutional layers, pooling layers and fully connected layers.

The Convolution Operation

Convolution, as implemented in machine learning applications, is an operation that takes two parameters:

The input I For example a 2-dimensional tensor representing an image.

The kernel K Typically a smaller 2-dimensional tensor if the input is 2-dimensional.

The output tensor S from the convolution operation (denoted with an asterisk) is computed as follows:

$$S(i, j) = (I * K)(i, j) = \sum_m \sum_n I(i + m, j + n)K(m, n) \tag{2.5}$$

Figure 2.4 illustrates how equation 2.5 is applied to a 2-dimensional tensor.

When moving the kernel over the image it is possible to skip some positions to further reduce the size of the output as well as reducing the computational cost. The size of these skips is called the **stride** of the convolution.

The Convolutional Layer

A convolutional layer usually consists of several learnable kernels (also called filters). The filters are convolved with the input image to produce a stack of tensors (channels) representing the features each kernel looks for. The first layers will usually learn to detect simple features such as edges, and kernels deeper in the network will find more complex features such as shapes.

One big advantage with convolutional layers is that the network has **sparse connectivity**(Goodfellow et al. [2016] chapter 9.2). Sparse connectivity refers to the fact that each output unit is influenced only by a subset of the input units (unlike in fully connected neural networks where every output unit can be influenced by every input unit). The size of this subset is determined by the kernel size. The input units influencing an output unit is called its **receptive field**. Figure 2.5 illustrates the receptive field of an output unit s_3. By having sparse

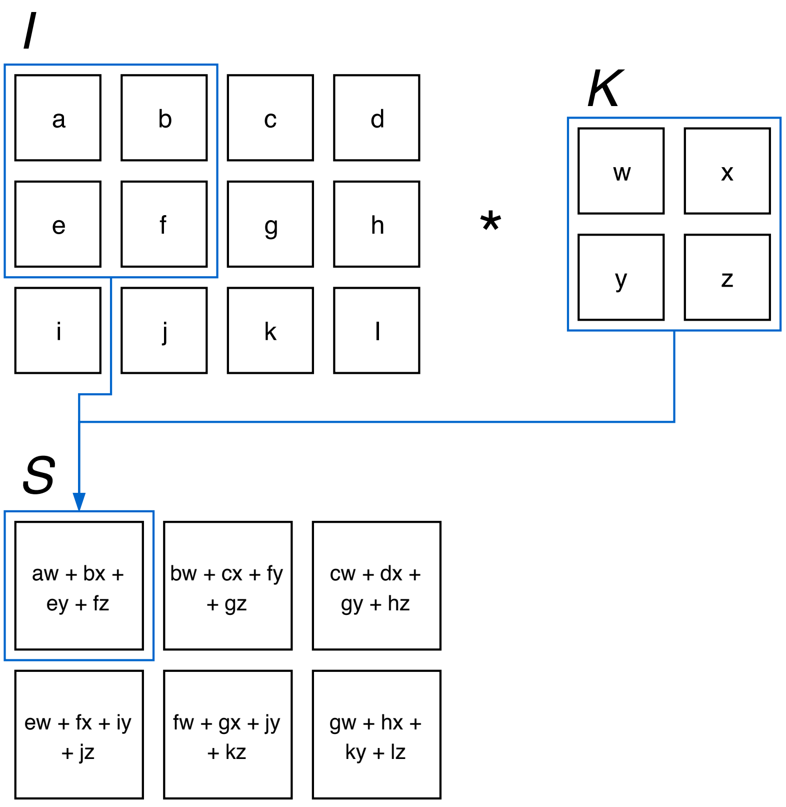


Figure 2.4: Illustration of 2D-convolution (adapted from Goodfellow et al. [2016])

connections in the network we drastically reduce the number of computations needed.

A convolutional network uses **parameter sharing**, which means that we use the same parameters across all locations in the input, rather than learning a distinct set of parameters for all spatial locations (Goodfellow et al. [2016] chapter 9.2). This drastically reduces the number of parameters to learn. Parameter sharing is possible because features detected at one location in the input are most likely useful to detect in other locations as well. Figure 2.6 illustrates the effect the convolution layer can have on an image.

The Pooling Layer

A pooling layer computes a statistical summary of an area in its input layer (Goodfellow et al. [2016] chapter 9.3). The most common pooling method used in deep learning is **max-pooling** which computes the maximum value within a neighbourhood. Pooling layers makes the network robust to small translations in the input,

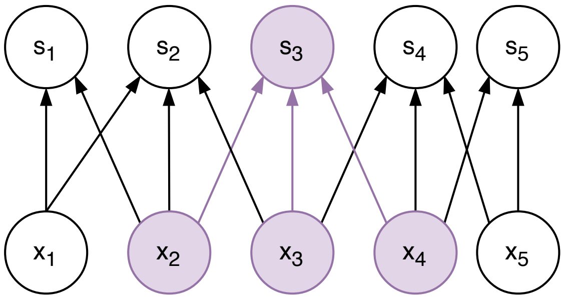
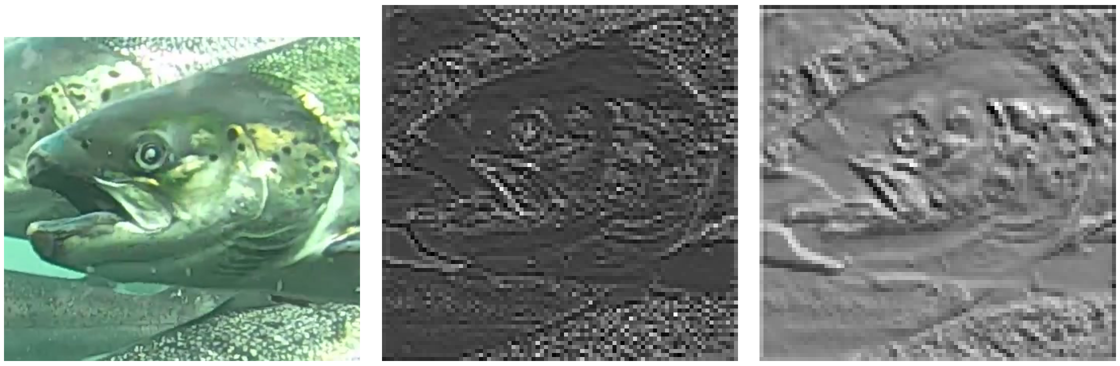


Figure 2.5: The receptive field of output unit s_3 is x_2, x_3, and x_4 (adapted from Goodfellow et al. [2016])



(a) The input to the neural network. (b) The output of one convolutional filter. (c) The output of another convolutional filter.

Figure 2.6: An illustration of two outputs from a convolutional layer in FishNet. We can see that the two filters (b) and (c) represent different features in the input image (a).

and it also reduces the size of the data. Figure 2.7 illustrates max-pooling.

Locally Connected Layer

Unlike convolutional layers where the kernel is identical for all receptive fields in the image the locally connected layer trains a separate kernel for each input patch (Gregor and LeCun [2010]). This increases the number of trainable parameters in the network compared to a convolutional layer, but the computational cost in the inference stage remains the same (Taigman et al. [2014]).

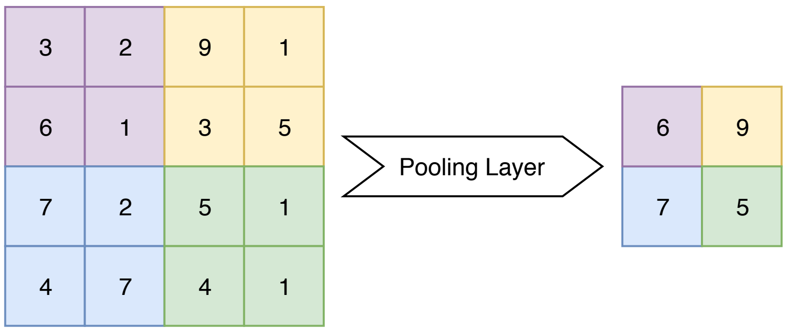


Figure 2.7: Illustration of **max-pooling** with a stride of two. The highest value of each receptive field is kept. Figure adapted from Li et al. [2018].

The Architecture of a Convolutional Network

There is a large number of different convolutional architectures for different applications, but most of them follow a similar pattern. For computer vision tasks the input is typically an image which is then fed into a convolutional layer followed by a pooling layer. Typically reducing the height and width of the image with pooling and increasing the number of channels with convolutions. This is repeated several times, and finally the output is flattened and fed into one or more fully connected layers that act as a classifier. For a description of the architecture used in this master thesis, refer to chapter 3.4.

2.2.3 You Only Look Once version 3 (YOLOv3)

YOLOv3 is a single neural network that uses features from the entire image to predict bounding boxes (see example of YOLOv3 output in Figure 2.8). Each bounding box consists of 5 predictions: x, y, w, h and confidence. x and y defines the coordinates for the center of the bounding box, and w and h are the width and height of the box respectfully. This system divides the image into an SxS grid where each grid cell is responsible for detecting objects that have their center in that cell.

YOLOv3 consists of 24 convolutional layers followed by 2 fully connected layers. The resulting network runs significantly faster than other detection methods with comparable performance.

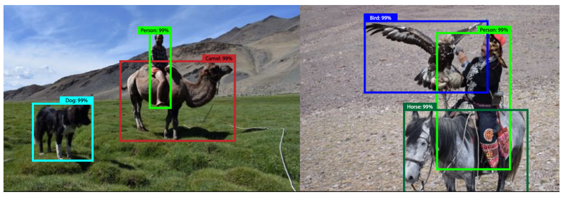


Figure 2.8: Example of YOLOv3 detector. Figure reprinted from Redmon and Farhadi [2018]

2.3 Related Work

With regards to identifying unique salmons without engaging directly with the salmon, the research done is quite limited. Earlier attempts of uniquely identifying salmons have relied on insertion of RF-ID chips or other physical marking systems (Eilertsen [2017]). This is not a good solution as direct contact with the salmon should be avoided as it could potentially hurt the salmon and damage the finished product. It also leaves a physical object that has to be handled later in the process. Physically handling the salmon is also inefficient.

We have chosen a couple of papers that we base the grounds of our research on. Starting off with a pre-study performed by SINTEF on identification of salmons using biometric recognition (Eilertsen [2017]).

2.3.1 SINTEF SalmID

SINTEF did a pre-study on the possibility of recognizing individual salmon based on the assumption that each individual has an unique pattern. They found that there was done little work on this area regarding Atlantic salmon. However, they refer to several papers about using the melanophore pattern of different animals to uniquely identify them.

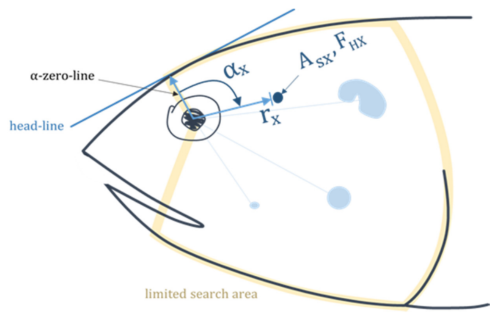
Melanin produces patterns. On humans, these tend to change over time but on salmon they change very little after they become "youths". Research has been done on recognizing several species including cheetahs, sand tiger sharks, and Chinook salmon using their melanophore pattern.

SINTEF plan on using two different methods for recognizing individual salmon:

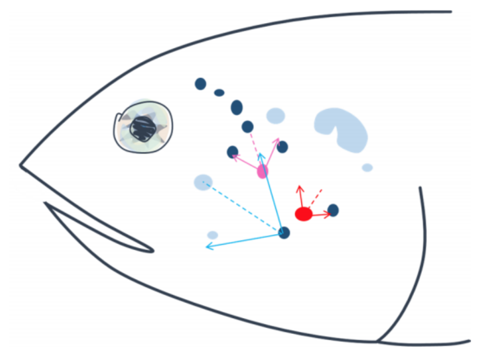
Polar coordinates The principle behind this technique is to find two reference points (the eye and the skull) and map each of the melanophore spots in a coordinate system based on these reference points. Having all the spots mapped with placement and size variables in a database, the database can later be used to look up a specific salmon.

Star constellation This technique takes inspiration from looking at stars and recognizing constellations. The sky in the night displays a sea of dots. When looking for constellations you look for a pattern, for example when looking for the Big Dipper you seek four stars in a row that is connected to a square. This concept can be used on the melanophore spots of a salmon as well. The benefit of using constellations instead of polar coordinates is that you do not need anchor points. This makes the recognition more robust with regards to rotation and movement of the salmon.

The recognition part is based on manually selected features of the salmon rather than learned representations. Figure 2.9 illustrates the two methods for recognizing the spot pattern.



(a) The polar-coordinate recognition method. Eye and head-line are reference points. Each melanophore spot has associated values α_x which is the angle from the reference line from the eye to the headline, and r_x which is the radius from origo (the eye). In addition, each spot has values A_{SX} which is the area of the spot and F_{HX} which is the Heywood Circularity Factor defined as $\mathrm{F_H} = \mathrm{P}/2^* \sqrt(\pi^* \mathrm{A_S})$ where P is the circumference of the spot.



(b) Three constellations found on a salmon head. These constellations are used to uniquely identify a salmon without the need of anchor points.

Figure 2.9: Salmon matching methods in SalmID. Figure reprinted from Eilertsen [2017].

2.3.2 Biometric Recognition and Individual Tracking of Salmon in Large-Scale Sea Cages

In the master thesis *Biometric recognition and individual tracking of salmon in large-scale sea cages*, Ivar Hammerset investigates identifying salmon based on the melanophore patterns on the side of their heads (Hammerset [2018]).

Neural networks are used to discover the location of salmon heads and eyes. After some preprocessing of the image, a simple blob detection algorithm is used to discover the melanophore spots. The locations of the spots and the eye are then translated into a polar representation which is saved in a database with the identity of the salmon. This approach is similar to the SINTEF SalmID polar-coordinate technique. To match the patterns of salmon, a matrix representation is created and the points are compared. More details about the algorithm used can be found in the thesis.

On the test set with images from 333 individuals the algorithm achieved an accuracy of 99.7%. Note that in only 40.4% of the test images the algorithm recognized the individual.

2.3.3 Face Verification Using Deep Learning

When using deep learning to create a face verification system, a common approach is to learn a mapping from images of faces to a n-dimensional embedding. When verifying similarity between two faces, one can check the similarity of the vectors created by the trained neural network (Taigman et al. [2014]). If the vectors are similar enough we can assume the two pictures to be of the same person. Essentially, the neural network learns a mapping of features describing the uniqueness of a person.

DeepFace

The model presented in the DeepFace paper (Taigman et al. [2014]) starts by altering images of faces so that they appear to face the camera (frontalization). The frontalized image is fed into a convolutional layer followed by a max pooling layer and another convolutional layer. According to the authors, these three layers mainly extract low level features and make the network robust to local translations.

The last convolutional layer is followed by three locally connected layers. This is done because the different regions of an aligned image have different characteristics, so the spatial invariance assumption of convolution does not hold. An example of this is that the eyes of a person will always be above the nose.

The final two layers of the network they use are fully connected. These layers are able to identify relations between features in different locations in the feature maps. The first fully connected layer is used as the face representation vector, and the output of the second one is fed into a softmax which produces a class distribution for an input image.

To verify whether two images are of the same person, the authors propose three approaches:

Unsupervised method: The similarity of two images is simply defined as the inner product of the two representation vectors.

Weighted χ^2 distance: For two feature vectors f_1 and f_2, $\chi^2(f_1, f_2) = \sum_i w_i(f_1[i] - f_2[i])^2/(f_1[i] + f_2[i]))$. The weight parameters w are learned using a linear support vector machine.

Siamese network: In the siamese network, the network (except the top layer used for softmax classification) is duplicated. One image is fed into each part of the network and the absolute difference between the feature vectors is computed. A fully connected layer is added and the network is trained to predict a single logistic unit (whether the images are of the same person). Training is only enabled for the new layers, and it is trained using standard cross entropy loss.

All three methods yielded good results compared to the state-of-the-art at the time. The siamese network approach required a lot more training data to avoid overfitting compared to to the other approaches.

FaceNet

The FaceNet paper (Schroff et al. [2015]) describes a system that learns and optimizes a vector representation directly, rather than extracting the representation from a bottleneck layer (like DeepFace). FaceNet learns a 128-dimensional feature vector (embedding) that represents a face. Unlike the DeepFace approach there is no 2D or 3D alignment done on the images.

The authors use **triplet loss** to train the network. The network is presented with three images:

x_i^a The anchor image.

x_i^p The positive image. This image is of the same person as the anchor image, but not the same image.

x_i^n The negative image. This image is of any other person.

When training we want to ensure that the anchor image is closer to the positive images of that person than it is to images of person. See figure 2.10 for an illustration. Equation 2.6 shows how the loss is computed for a mini-batch.

$$L = \sum_{i}^{m} \left[\|f(x_i^a) - f(x_i^p)\|_2^2 - \|f(x_i^a) - f(x_i^n)\|_2^2 + \alpha \right]_+ \tag{2.6}$$

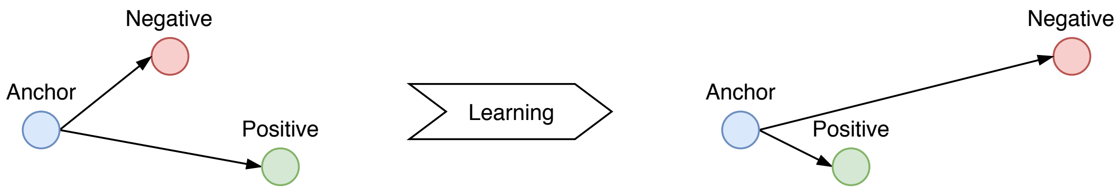


Figure 2.10: Triplet loss minimizes the distance between images of the same person and maximizes the distance to images of other persons (adapted from Schroff et al. [2015])

According to the authors, it is important to select triplets that are hard to ensure that the network converges as quickly as possible. The triplets are chosen from within each mini-batch, and all anchor-positive pairs are used in combination with negative examples. How triplets are selected is discussed further in section 3.3.

The authors describe several different deep neural network architectures, where the major differences between them are the number of trainable parameters. The number of parameters in the networks range from about 4 millions to 140 millions. When evaluating the networks the L_2-distance between two images is compared. If the distance is above a certain threshold they are classified as different. According to the authors they are able to reduce the error reported by the DeepFace paper by a factor of seven. The smaller inception networks perform nearly as good as the very deep networks.

2.4 Literature Review Protocol

To search for relevant literature, we have used a combination of structured literature review (SLR) and the snowballing approach.

The **structured literature review** consists of gathering data, then filter this data through a series of well-defined steps. The data gathered are in our case available primary studies on relevant subjects to this thesis.

Performing these steps according to a predefined protocol enables other researchers to reproduce the result of the literature review. So in addition to helping us reduce and filter out the most relevant papers through the use of inclusion and quality criteria, the SLR also enhances the reproducibility of the literature review of our thesis.

The snowballing approach refers to the method where relevant articles are discovered using the reference lists and citations of articles already known to be relevant through our SLR (Wohlin [2014]).

The structured literature review consisted of the following steps:

1. **Identifying literature research questions.** The first step was to identify research questions for the literature search. These questions are important for defining the scope of the search. For our research we used the following research questions:

   LRQ1 What is the state-of-the-art in face identification and verification?

   LRQ2 What is the state-of-the-art in salmon tracking and recognition?

   LRQ3 What research exists in the field of salmon pattern recognition?

   LRQ4 What neural network architectures are used in the field of face recognition?

   These literature research questions helped us in identifying the relevant search terms for the next step.

2. **Literature search.** The next step was to find literature and studies relevant to the research questions identified in step 1. This includes defining the search engines to use and which search terms to use. The following search engines were used in the literature search:

   - Google Scholar[1]
   - IEEE Explore[2]
   - Oria[3]
   - SpringerLink[4]
   - SINTEF Publications[5]

[1]https://scholar.google.no/

[2]https://ieeexplore.ieee.org/Xplore/home.jsp

[3]https://bibsys-almaprimo.hosted.exlibrisgroup.com/primo-explore/search?vid=NTNU_UB

[4]https://link.springer.com/

[5]https://www.sintef.no/publikasjoner/

The terms used when searching the search engines were: *Deep learning*, *pattern recognition*, *face verification*, *face recognition*, *face identification*, *neural network face recognition*, *facenet architecture*, *deep learning architecture*, *salmon identification*, *salmon recognition*, *biometric identification of animals*, *melanophore pattern salmon*, *melanohpore pattern unique*, *animal pattern recognition*, *fish recognition.*

These search terms were used in different combinations. The terms regarding salmon was also used in Norwegian as a lot of research regarding the Atlantic salmon are published in Norwegian. This lead to the initial stack of papers and articles we considered.

3. **Selection of primary studies.** This step limits the number of results by eliminating results that are duplicates, too old or clearly not relevant for our research questions.

4. **Quality assessment.** This step uses a list of inclusion and quality criteria to rate the literature that wasn not eliminated in the previous step. By assigning a score for each criteria the literature can be sorted by it's total relevance. This way we can determine which studies to analyze first. Table 2.1 presents the inclusion and quality criteria used.

ID	**Criteria**
IC1	The studys main focus is related to the literature research questions.
IC2	The study is a primary study focusing on empirical results.
QC1	There is a clear statement of the aim of the research.
QC2	The study is put into context of other studies and research.
QC3	The system or algorithmic design decisions are justified.
QC4	The test dataset is reproducible.
QC5	The study algorithm is reproducible.
QC6	The experimental procedure is thoroughly explained and reproducible.
QC7	The performance metrics used in the study are explained and justified.
QC8	The test results are thoroughly analyzed.
QC9	The test evidence supports the findings presented.

Table 2.1: Inclusion (IC) and quality (QC) criteria.

After performing all the steps in our structured literature review, we ended up with a set of articles listed in Table 2.2.

ID	Title	Authors and Year
P01	Facenet: A unified embedding for face recognition and clustering	Schroff et al. [2015]
P02	Deepface: Closing the gap to human-level performance in face verification	Taigman et al. [2014]
P03	Identifikasjon av lakseindivider — Biometri fase 1 (SalmID)	Eilertsen [2017]
P04	Consistent melanophore spot patterns allow long-term individual recognition of Atlantic salmon Salmo salar	Stien et al. [2017]
P05	Onset of Melanophore Patterns in the Head Region of Chinook Salmon: A Natural Marker for the Reidentification of Individual Fish	Merz et al. [2012]
P06	Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning.	Szegedy et al. [2017]
P07	MobileNetV2: Inverted Residuals and Linear Bottlenecks	Sandler et al. [2018]
P08	Very Deep Convolutional Networks for Large-Scale Image Recognition	Simonyan and Zisserman [2014]
P09	Deep residual learning for image recognition	He et al. [2016]

Table 2.2: The resulting set of research from our SLR

2.5 Summary

From the structured literature review we can summarize the state-of-the-art in the fields of salmon recognition and face recognition.

The uniqueness of melanophore spot pattern has been researched on many animals. The animals range from cheetahs to frogs and sand tiger sharks. In addition, research shows that the melanophore pattern on both Chinook and Atlantic salmon are persistent over time. The research indicates that these patterns can be used to identify individuals.

Current research into computer vision based approaches for salmon recognition is exclusively based on using the geometric properties of the melanophore patterns on the side of the head. There has been some use of deep learning, but this has been for preprocessing of the images to improve the spot detection. To our knowledge there has been no attempt to use end-to-end deep learning to solve the task.

Face recognition using deep learning has achieved close to human level performance. The models are increasingly tolerant to challenging conditions such as lighting, shadows, and rotation. Models such as FaceNet do not require complex geometric alignment to achieve very good results. The models learn the discriminating features of the input themselves, eliminating the need for manual feature selection.

Chapter 3

The FishNet Approach

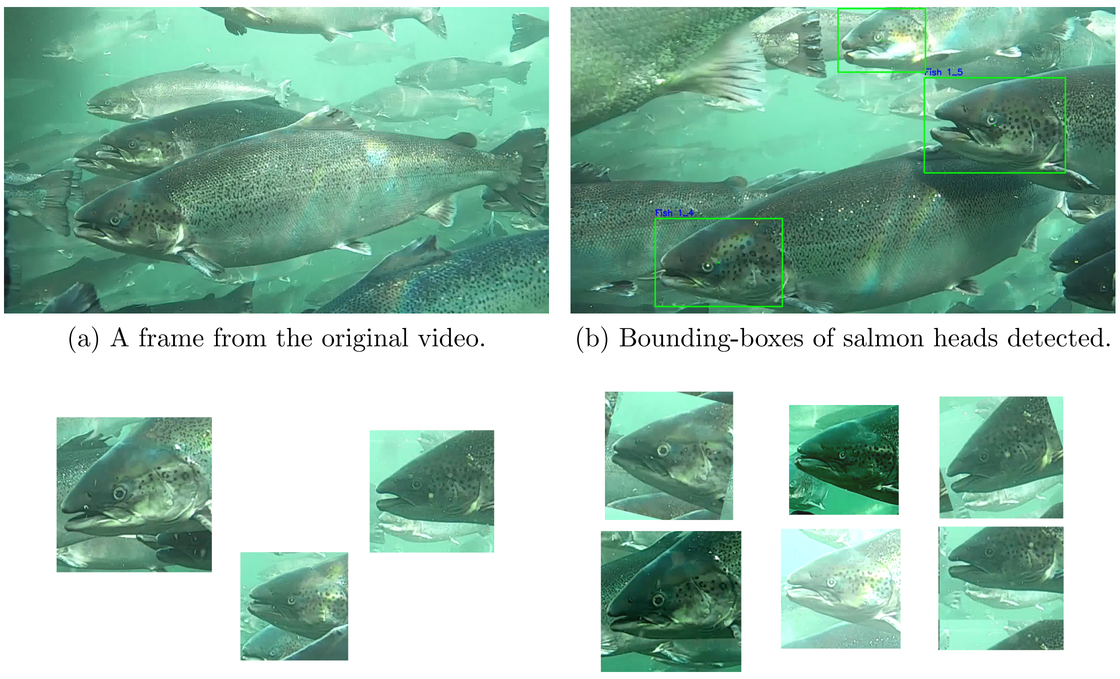
This chapter presents how the training and testing data used in this thesis was created. It further explains the triplet selection algorithm used, before finally the architecture choices are presented and justified.

3.1 The *Labeled Fish NOT in the Wild* (LFNW) Dataset

The data used in this thesis was raw video provided by Sealab AS. The quality, camera angle, light, and distance to the salmon varied in the different video clips. In order to generate the best possible data for the task at hand, video from a steady shot with salmon swimming by, close to the camera was chosen for the creation of our dataset. This section will describe the process of creating our dataset, LFNW. Figure 3.1 illustrates a overview of how a frame from the video is transformed into training data.

To train our models, a dataset where each salmon is labeled with a unique id was required. In order to get a labeled dataset out of a video stream, we started by converting the videos into images. The video was filmed at 30 FPS (frames per second) meaning we had 30 images per second of video. Salmon heads in the images were marked manually with a bounding-box tool. After labeling approximately 500 bounding-boxes. The bounding-boxes were used to train a YOLOv3 (Redmon and Farhadi [2018]) network to recognize salmon heads. A brief explanation of YOLOv3 can be found in section 2.2.3. After training the network for a few hours it was used to create bounding-boxes on every salmon head in all video frames. Figure 3.2 shows the cropped bounding-boxes of two salmon heads detected by the algorithm.

The results were stored as tuples with the following content:



(a) A frame from the original video. (b) Bounding-boxes of salmon heads detected.

(c) Salmon heads extracted from the video. (d) Augmented images.

Figure 3.1: Overview of the dataset creation.

```
(frame, x, y, w, h)
```

The columns are then scaled to the range of $[0, 1]$. DBSCAN was used to cluster the bounding-boxes. The clustering with euclidean distance as the distance function performed poorly in this case. Euclidean distance favored bounding-boxes in the same frame since the distance between the *frame* features yielded a value of 0 in these cases. This is not favorable as two bounding-boxes in the same frame are definitely not the same salmon. To counteract this, the distance function was altered to return an arbitrarily high value when the frame-value of the bounding-boxes were equal. The clustering was improved, but it still mislabeled salmon that swam in the same area of the field of view. This is understandable as the x and y features are similar in these cases. After the realization that the same bounding-box between two frames has a high IOU (intersection over union), the distance function was altered to reward high IOU values to further improve the distance function.

Equation 3.1 describes the distance function used in the clustering algorithm. If two bounding-boxes are in the same frame, the distance is set to an arbitrarily high value. If the bounding-boxes are not in the same frame, the intersection over union is measured to check how closely the bounding-boxes overlap. Then a temporal

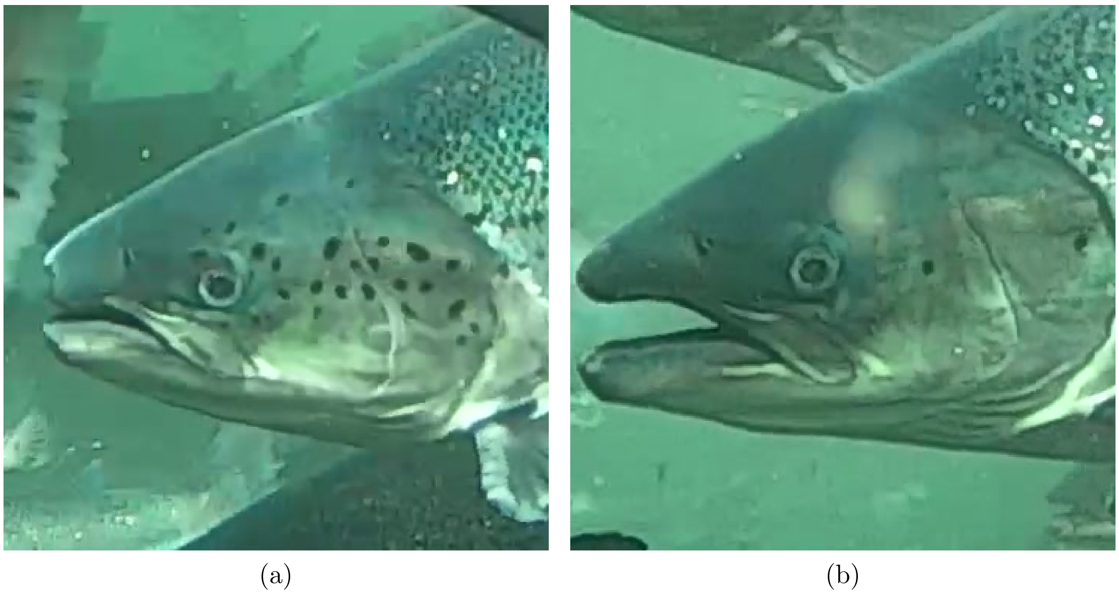


Figure 3.2: Two examples of salmon heads detected by the YOLO network.

distance is added by computing a weighted distance of the frame numbers. This is done to ensure that overlapping bounding-boxes in frames next to each other receive a low distance value.

$$D(b1, b2) = \begin{cases} \infty & : b1_{frame} = b2_{frame} \\ 1 - IOU(b1, b2) + |b1_{frame} - b2_{frame}| \times 0.5 & : otherwise \end{cases}$$

$$IOU(b1, b2) = \frac{\text{Intersection Area}}{\text{Union Area}} \tag{3.1}$$

Figure 3.3 illustrates the bounding-boxes discovered in two video frames, and how the intersection over union is calculated. The blue rectangle is the intersection of the two bounding-boxes, and the red and blue area is the union.

This approach works fairly well except in cases where a salmon disappears behind a different salmon and then reappears again. In those cases it is frequently misidentified as a new salmon. This problem was solved by manually reviewing the labels, and replacing the labels for misidentified salmon.

While the dataset has been manually reviewed, there is a possibility that there are errors in the labeling of salmon. It is quite difficult for humans to distinguish the salmon from each other. This however, should not have major impacts on the training process as the methodology for selecting training data discussed in section

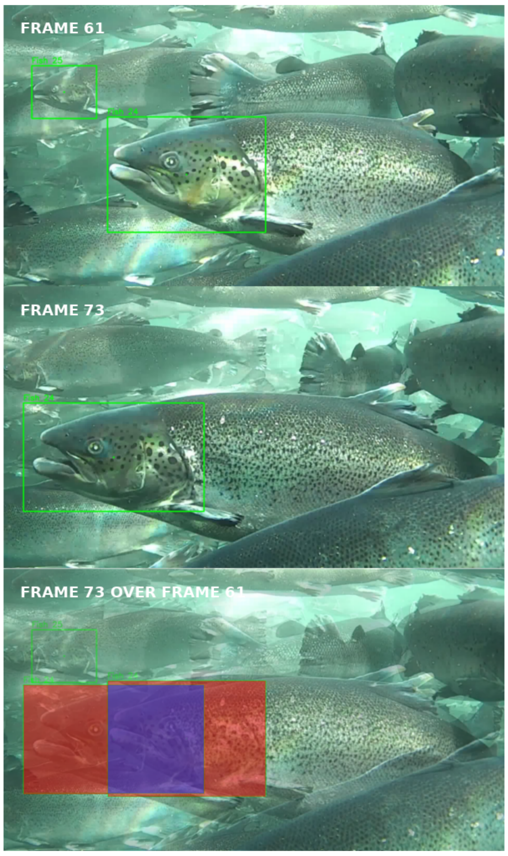


Figure 3.3: Illustration of IOU. The top image is frame 61 in the video, the middle is from frame 73, and the bottom image shows the two images over each other. Despite being 12 frames apart, the IOU is still quite high. The red and blue area is the union between the bounding boxes, and the blue area alone is the intersection.

3.3 is robust to errors in the dataset.

The resulting dataset contained about 15 000 images of 715 different salmon. Figure 3.4 shows a plot of how many images there are of each salmon in the dataset. 5 images is the number of images that is most common. The median is 9 and the mean is 21. Unfortunately the dataset is not publicly available at the time of writing, but hopefully this description will aid in any attempts to create similar datasets.

Data augmentation

To increase the size of the dataset we used data augmentation on the images. The following transformations were applied to the dataset:

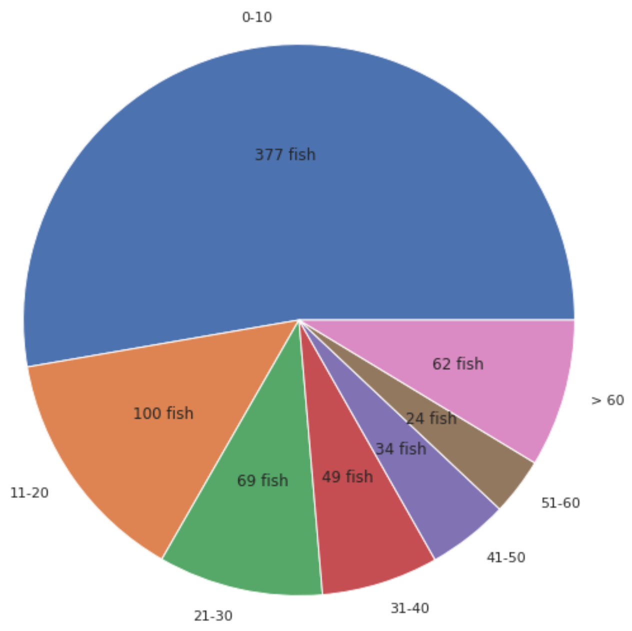


Figure 3.4: Distribution of images per fish. Most (about 53%) of the salmon have between 0 and 10 images of their head. The median is 9 and the mean is 21. For more detailed counts, see appendix A.1.

- Horizontal and vertical shifting of the image.
- Rotation (within a 30° range).
- Channel shift (simulating brighter and darker conditions).

Augmenting the dataset has several advantages other than purely increasing the number of training samples available to us. By shifting the location of the salmon the network can become more tolerant to variations, and avoid overfitting towards the location of the salmon head within the bounding box. Rotating the images makes the network able to detect patterns even if the salmon swim in another angle. We chose 30° as this is a realistic angle range for the salmon to swim. Channel shifting helps us simulate different lighting conditions. See figure 3.5 for examples of channel shift augmentation.

For each image in the dataset we generated five augmentations of each type. This increased our dataset size by a factor of 15. The augmented dataset ended

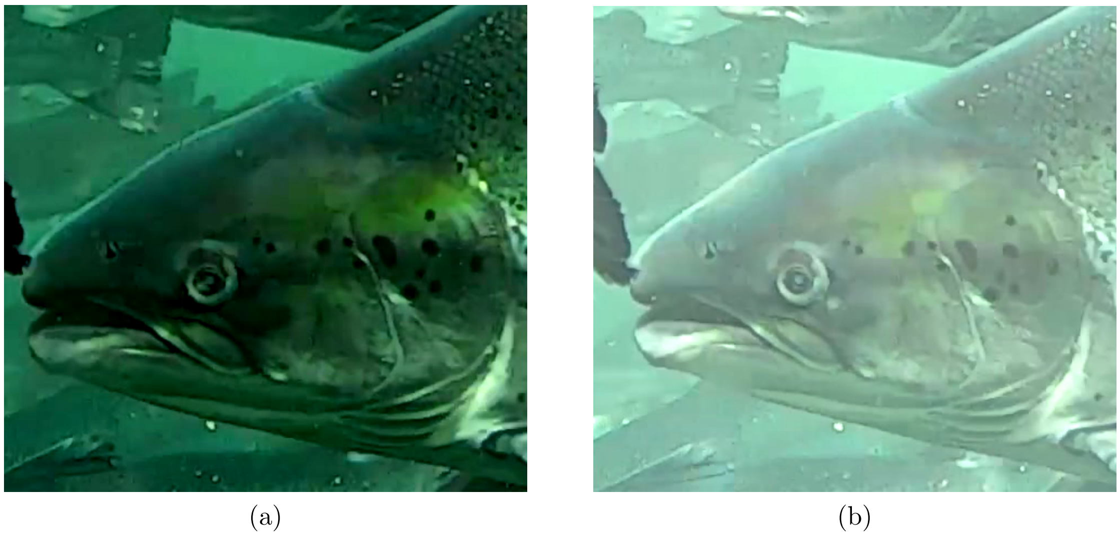


Figure 3.5: Two examples of cropped salmon heads that are augmented with channel shift simulating different lighting conditions.

up with approximately 225 000 images of the 715 different fish. We have decided to call the final dataset the *Labeled Fish NOT in the Wild* (LFNW) dataset.

Dataset Splits

90% of the salmon (about 640 individuals) served as the training dataset. To evaluate the models, the images of the remaining 10% of the salmon were moved to a separate test set. The test set consists of approximately 14000 images.

3.2 Justification for Using the FaceNet Approach to Learn Salmon Embeddings

As mentioned in the FaceNet paper, there are several ways one could learn embeddings to represent the identities of an individual (whether that individual is a salmon or a human being). A common approach is to frame the problem as a classification task where the networks learns to classify each image to the correct class, where each class represents the id of one fish. To extract the embedding for an image we can take the activation from an intermediary layer. Figure 3.6 illustrates this approach. This is the approach used by for example the DeepFace paper (Taigman et al. [2014]).

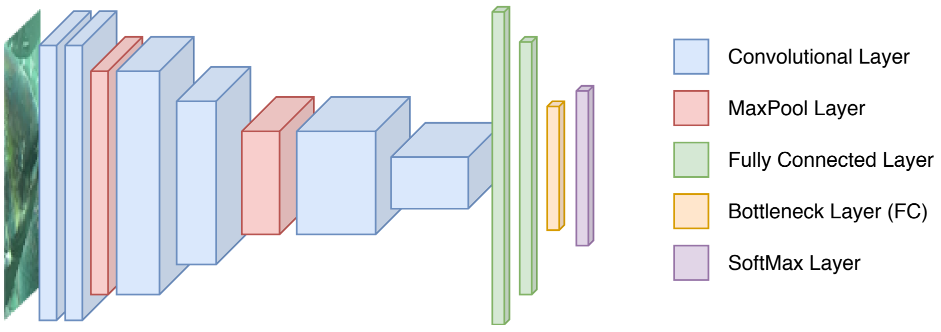


Figure 3.6: One method for learning embeddings. The network is trained for standard classification, and the embedding representing a face is extracted from a fully connected layer towards the end of the network (marked in yellow).

A disadvantage of this approach is that the embeddings are only learned indirectly. One hopes that the bottleneck layer will work well as a representation of the faces. The approach of the FaceNet paper presents us with a way to directly optimize a loss for the task of generating embeddings which identify the face in an image. As it demonstrates such good performance on human faces we believe that it is a reasonable architecture to attempt to use on fish.

3.3 Triplet Selection

As noted in the FaceNet paper (Schroff et al. [2015]), careful triplet selection is important for the training process of the network. The training goal of the algorithm is to ensure that the embeddings of two images (anchor and positive) of the same salmon are closer to each other than any images of other salmon (negatives) by a margin α. For the experiments in this thesis, the value for α was set as 0.2, the same as used in the FaceNet paper.

To ensure effective training, it is important to select triplets that violate this constraint. To do this, the system has to compute the embeddings for images during training, and then select the appropriate training samples. For efficiency purposes, this is done within each batch. First, a random set of salmon images are sampled from the training dataset. Then the images are fed through the network to generate embeddings. Finally, the embeddings are used to select triplets where the difference between the negative and positive embeddings are within α. Algorithm 1 describes this process.

```
Input: embedding vectors
Input: number of fish
Input: number of embeddings per fish
Input: $\alpha$
Data: triplets = []
foreach fish do
    for anchor in embeddings of current fish do
        negative distances = $L_2$-distances from anchor to embeddings of
          other fish
        for positive in remaining embeddings of current fish do
            compute distance between anchor and positive
            negatives = find all negative embeddings where
              $negative\_dist - positive\_dist < \alpha$
            select a random negative from negatives and append (anchor,
              positive, negative) to triplets
        end
    end
end
shuffle triplets
return list of triplets
```

Algorithm 1: Triplet selection

By using this methodology to train the network, we ensure that training is performed on triplets the network can learn from. Using triplets that already satisfy the constraint of α would not contribute to further training, and only slow down the process. Calculating the hardest triplets for the entire dataset every epoch would be computationally very slow. Additionally, if we were to select the hardest triplets every time it could cause poor training. This is because selection of hardest triplets would be dominated by for example mislabeled or low quality images.

3.4 Neural Network Architectures

During our experimentation, we used different neural network architectures to train embeddings. All the networks shared a general architecture of a convolutional neural network where the top layer (classification layer) was replaced by a 128-dimensional dense layer to represent the embedding of the input image. Figure 3.7 shows an illustration of this architecture, which can be used to compute the embedding for one image.

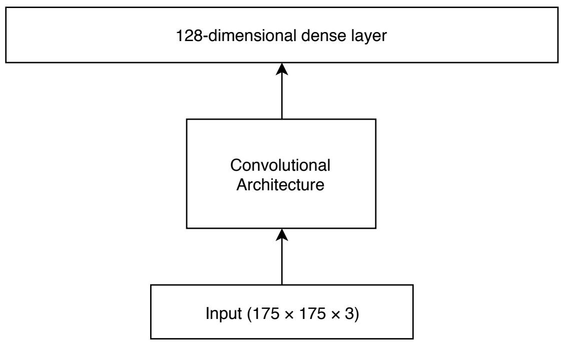


Figure 3.7: Generic architecture for inference

Network Architecture	# Parameters	Pretrained with
FishNet1 (Inception ResNet v2)	55M	ImageNet
FishNet2 (MobileNet v2)	2.4M	ImageNet
FishNet3 (VGG-16)	15M	ImageNet

Table 3.1: The neural network architectures used in the experiments.

To train the network using triplet loss, the network needs to use more than one image at once. To achieve this, the convolutional and embedding parts need to be replicated once for each image. Note that the weights are shared between the instances. The output from the embedding layers is fed into a custom layer that computes the triplet loss, which in turn is used to train the model. Figure 3.8 illustrates the model used for training. Table 3.1 shows the neural network architectures used in the experiments.

All models were initialized with the convolutional weights pretrained on the ImageNet dataset (Deng et al. [2009]). The assumption being that features learned for image classification may be a useful starting point for learning how to distinguish salmon from each other, thereby reducing the amount of training data needed to train the models. This is a form of transfer learning, as discussed in section 2.2.1. Appendix A.2 documents how to create the models using the Keras API in TensorFlow[1].

3.4.1 The Triplet Loss Layer

To compute the loss during the training, a custom triplet loss layer was used. Equation 3.2 defines how the loss L is computed for a minibatch of size m.

$$\begin{aligned} f(x) &= \text{Embedding of image } x \\ L &= \sum_{i}^{m} \left[\| f(x_i^a) - f(x_i^p) \|_2^2 - \| f(x_i^a) - f(x_i^n) \|_2^2 + \alpha \right]_+ \end{aligned} \tag{3.2}$$

This is identical to how the triplet loss is defined in the FaceNet paper. The loss computes the distance between the anchor and the positive, and the anchor and the negative. The goal is to have the positive distance be smaller than the negative distance. The difference between the positive and negative distance are summed. To encourage larger distances the margin α is added to the loss function. To avoid negative loss, the maximum of the loss of the triplet and 0 is taken before summation. What we end up with is a loss function that the model seeks to minimize. See appendix A.2 for how the triplet loss layer is implemented using Keras.

3.4.2 FishNet1 (Inception ResNet v2)

FishNet1 uses the Inception ResNet v2 (Szegedy et al. [2017]) as the convolutional part of the architecture. Inception ResNet v2 has shown great result in computer vision tasks. Inception networks work by introducing an inception module, which essentially is concatenating several different kernel sizes in each layer. This eliminates the need for choosing the right kernel size in each layer, and makes the network wider rather than deeper. This can be advantageous as deep networks are prone to overfitting and vanishing gradients. Figure 3.9 illustrates an example of an inception module.

Inception ResNet v2 also includes residual connections in the inception modules. Residual connections are connections in the network that skip one or more layers to help combat the issue of vanishing gradients (He et al. [2016]). Figure 3.10 illustrates an inception module from FishNet1 with a residual connection added. The Inception ResNet v2 architecture consists of many of these modules.

The decision to use Inception ResNet v2 for the convolutional part of the network was based on the fact that the best performing network described in the

[1]`https://www.tensorflow.org/api_docs/python/tf/keras`

FaceNet paper is based on the inception model. FishNet1 has about 55 million parameters.

3.4.3 FishNet2 (MobileNet v2)

FishNet2 uses the architecture of MobileNet v2 as the convolutional part. MobileNet is an architecture that was created to be able to run on mobile devices. The architecture has shown great performance on many different computer vision tasks while maintaining a relatively small model size (Sandler et al. [2018]). MobileNet also makes use of residual blocks. Unlike FishNet1, the skip connections go from one bottleneck layer with relatively few channels to the next bottleneck layer. The layers in the residual block have more channels than the bottlenecks. This approach reduces the memory requirements of the model.

The MobileNet architecture also makes use of depthwise separable convolutions, which essentially consists of splitting the convolution operation into several smaller operations with a smaller total computational cost. Depthwise separable convolutions consist of two steps. First, the input is convolved with the same number of kernels that it has channels, where each kernel only processes one channel. The result is then convolved with n 1×1 kernels to create the output with n channels. The first step is called the depthwise convolution, and the second step is called the pointwise convolution (Sandler et al. [2018]).

FishNet2 has about 2.4 million trainable parameters.

3.4.4 FishNet3 (VGG-16)

FishNet3 uses a simpler convolutional network than FishNet1 and FishNet2. It is based on VGG-16 (Simonyan and Zisserman [2014]), which consists of 16 layers of convolutional layers and max pooling. Unlike the other architectures it does not make use of inception modules or residual connections. This allows us to get an indication of the effects of adding more complex modules to the feature extraction part of the neural network. FishNet3 has about 15 million trainable parameters.

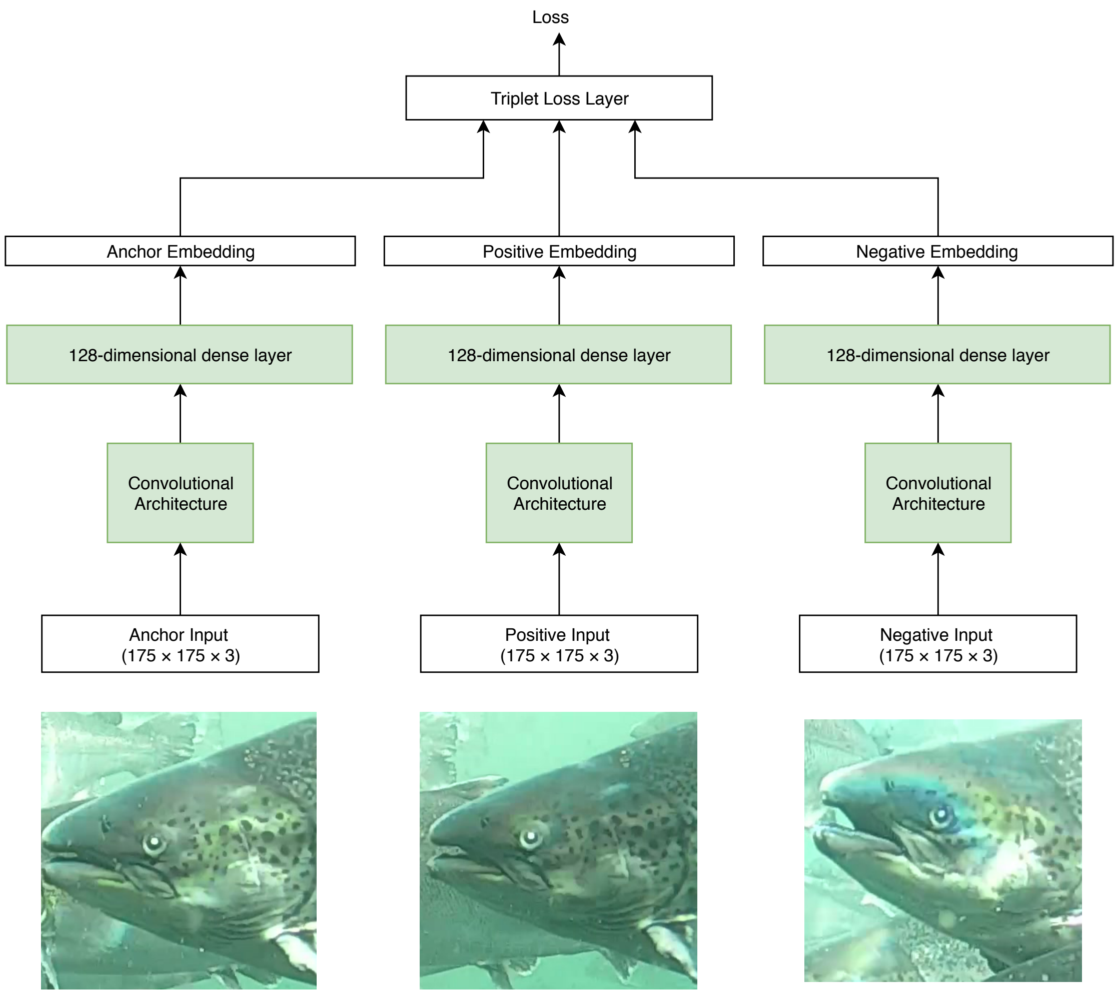

Loss
Triplet Loss Layer
Anchor Embedding
Positive Embedding
Negative Embedding
128-dimensional dense layer
128-dimensional dense layer
128-dimensional dense layer
Convolutional Architecture
Convolutional Architecture
Convolutional Architecture
Anchor Input (175 × 175 × 3)
Positive Input (175 × 175 × 3)
Negative Input (175 × 175 × 3)

Figure 3.8: Generic architecture with triplet loss. Parts of the network with shared weights are colored green.

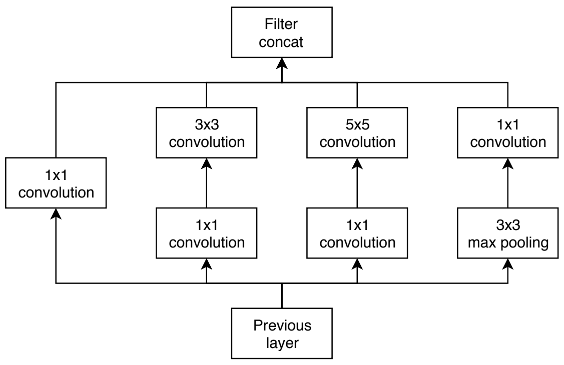


Figure 3.9: An illustration of the inception module, adapted from Szegedy et al. [2015]

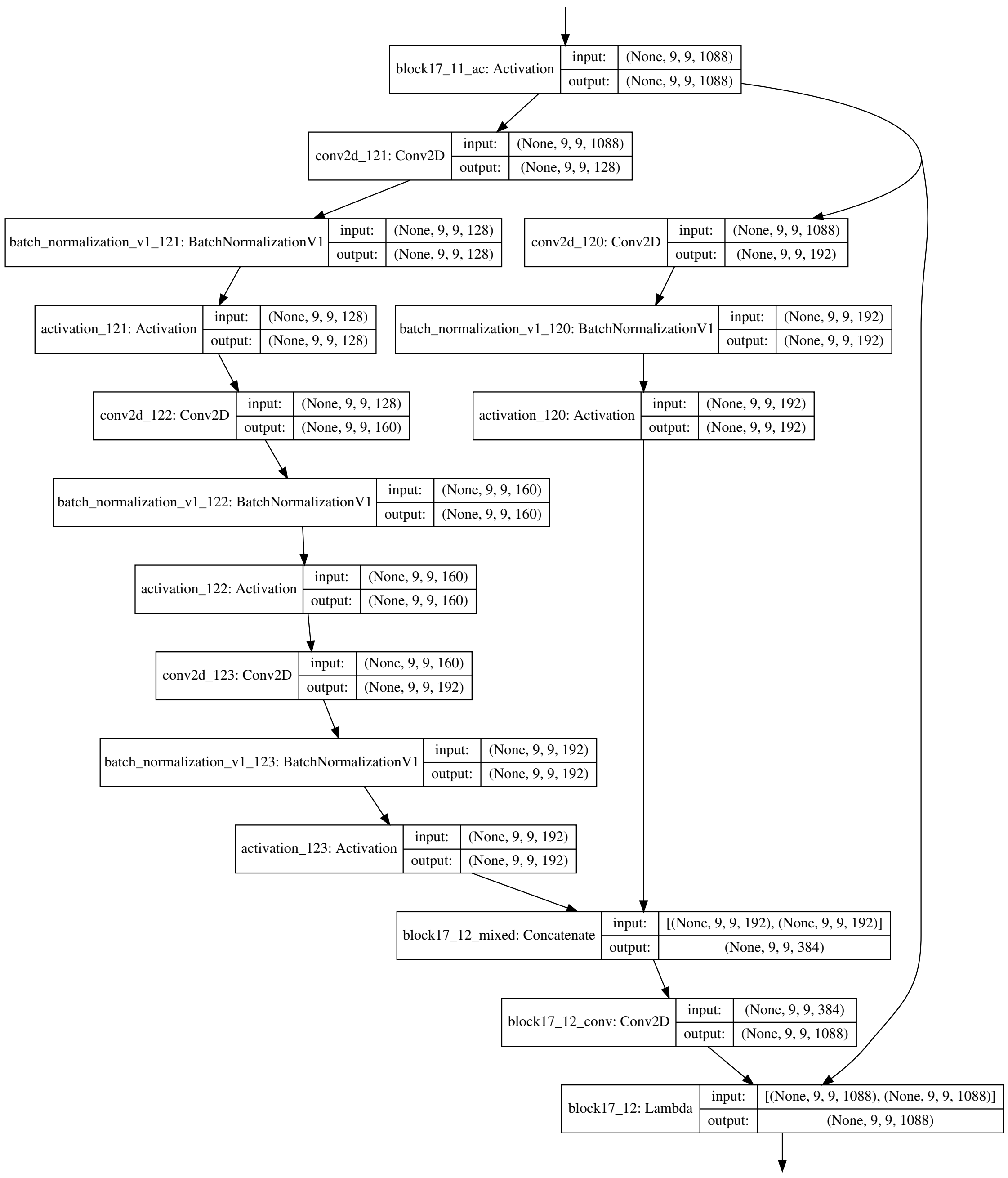

block17_11_ac: Activation
input: (None, 9, 9, 1088)
output: (None, 9, 9, 1088)
conv2d_121: Conv2D
input: (None, 9, 9, 1088)
output: (None, 9, 9, 128)
batch_normalization_v1_121: BatchNormalizationV1
input: (None, 9, 9, 128)
output: (None, 9, 9, 128)
conv2d_120: Conv2D
input: (None, 9, 9, 1088)
output: (None, 9, 9, 192)
activation_121: Activation
input: (None, 9, 9, 128)
output: (None, 9, 9, 128)
batch_normalization_v1_120: BatchNormalizationV1
input: (None, 9, 9, 192)
output: (None, 9, 9, 192)
conv2d_122: Conv2D
input: (None, 9, 9, 128)
output: (None, 9, 9, 160)
activation_120: Activation
input: (None, 9, 9, 192)
output: (None, 9, 9, 192)
batch_normalization_v1_122: BatchNormalizationV1
input: (None, 9, 9, 160)
output: (None, 9, 9, 160)
activation_122: Activation
input: (None, 9, 9, 160)
output: (None, 9, 9, 160)
conv2d_123: Conv2D
input: (None, 9, 9, 160)
output: (None, 9, 9, 192)
batch_normalization_v1_123: BatchNormalizationV1
input: (None, 9, 9, 192)
output: (None, 9, 9, 192)
activation_123: Activation
input: (None, 9, 9, 192)
output: (None, 9, 9, 192)
block17_12_mixed: Concatenate
input: [(None, 9, 9, 192), (None, 9, 9, 192)]
output: (None, 9, 9, 384)
block17_12_conv: Conv2D
input: (None, 9, 9, 384)
output: (None, 9, 9, 1088)
block17_12: Lambda
input: [(None, 9, 9, 1088), (None, 9, 9, 1088)]
output: (None, 9, 9, 1088)

Figure 3.10: An illustration of an inception module in FishNet1. The rightmost connection (from block17_11_ac to block17_12) is the residual connection. The tuples in the input and output fields are in the format *(batch, x, y, channel)*, where *None* symbolizes that the network can take any batch size.

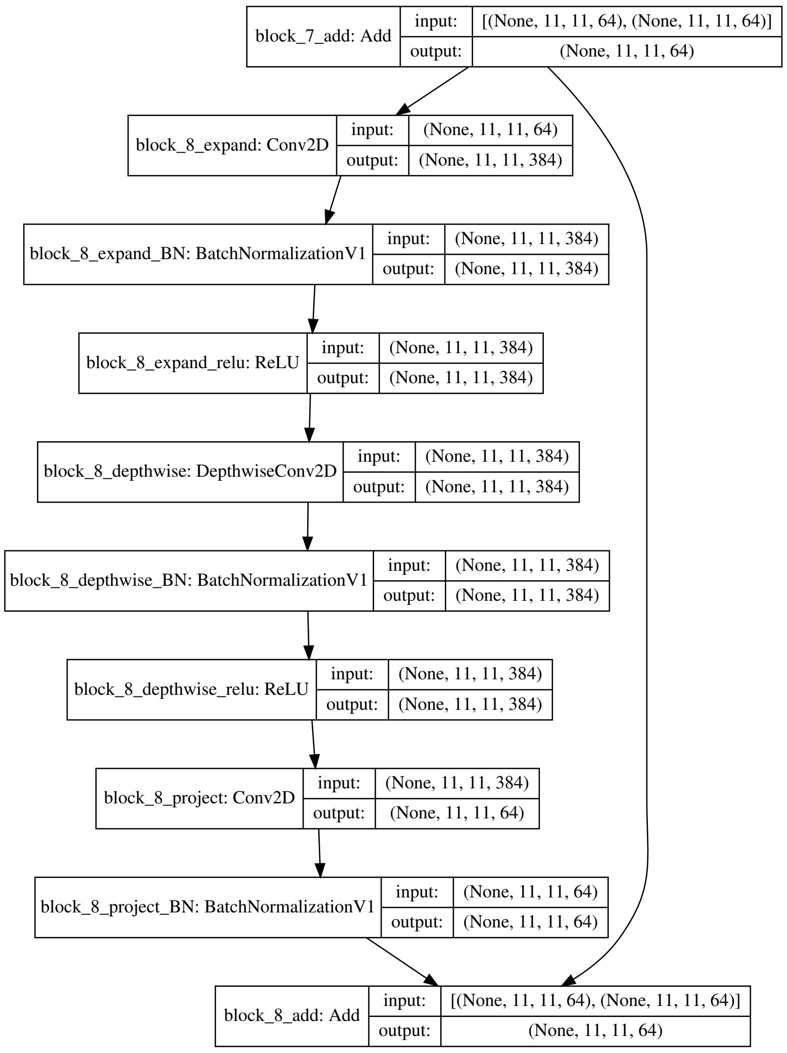

block_7_add: Add
input: [(None, 11, 11, 64), (None, 11, 11, 64)]
output: (None, 11, 11, 64)
block_8_expand: Conv2D
input: (None, 11, 11, 64)
output: (None, 11, 11, 384)
block_8_expand_BN: BatchNormalizationV1
input: (None, 11, 11, 384)
output: (None, 11, 11, 384)
block_8_expand_relu: ReLU
input: (None, 11, 11, 384)
output: (None, 11, 11, 384)
block_8_depthwise: DepthwiseConv2D
input: (None, 11, 11, 384)
output: (None, 11, 11, 384)
block_8_depthwise_BN: BatchNormalizationV1
input: (None, 11, 11, 384)
output: (None, 11, 11, 384)
block_8_depthwise_relu: ReLU
input: (None, 11, 11, 384)
output: (None, 11, 11, 384)
block_8_project: Conv2D
input: (None, 11, 11, 384)
output: (None, 11, 11, 64)
block_8_project_BN: BatchNormalizationV1
input: (None, 11, 11, 64)
output: (None, 11, 11, 64)
block_8_add: Add
input: [(None, 11, 11, 64), (None, 11, 11, 64)]
output: (None, 11, 11, 64)

Figure 3.11: A residual block from the convolutional part of FishNet2. The residual block goes from few channels (block_7_add), to many channels (i.e. block_8_depthwise), to few again (block_8_add). Note how this is different to Figure 3.10 of FishNet1 where the skip connections go from wide to wide layers. The tuples in the input and output fields are in the format *(batch, x, y, channel)*, where *None* symbolizes that the network can take any batch size.

Chapter 4

Experiments and Results

This chapter describes how the FishNet models, described in section 3.4, were trained and evaluated. We also present the results of the evaluations and discuss how the models compare to each other.

4.1 Experimental Plan

All the models were trained for 200 epochs. In each epoch, 10% of the training data was held out and used as validation data. If the validation loss decreased after an epoch, the weights of the network were saved to use for testing. The final testing was performed on a dataset of salmon with disjoint identities to the training data. The evaluation protocol used to test the models is inspired by how the different FaceNet models are compared. The testing protocol is as follows:

1. Compute the embedding for each image in the test set using the weights with best validation loss.
2. Compute the distance between each pair of embeddings, and mark them with the class *same* as true or false.
3. The model can then be evaluated by computing metrics for different similarity threshold values. Section 4.3 presents the different evaluation metrics.
4. Repeat the process for all the neural network architectures.

Appendix A.3 shows the Jupyter Notebook used for model evaluation.

4.2 Experimental Setup

The experiments were run on a computer with the following specifications:

Hardware

- AMD Ryzen threadripper 2920x 12-core processor $\times$ 24
- 2$\times$ GeForce RTX 2080 Ti/PCIe/SSE2
- 128 GB RAM

Software

- Ubuntu 18.04.2 LTS[1]
- NVIDIA Driver Version: 418.56[2]
- CUDA Version: 10.1[3]
- Python 3.6.7[4]
- Tensorflow GPU 1.13.1[5]
- Scikit-Learn 0.20.4[6]

Training

During an epoch of training, 10 images of each salmon was sampled from the training data. Then, the weights from the previous iteration are used to compute the embeddings of the sampled images. The training triplets are then selected using the algorithm described in section 3.3. The image size used was $175 \times 175 \times 3$, and the batch size was 32. To train the networks, gradient descent with backpropagation and the Adam optimizer was used. The Adam optimizer maintains an adaptive learning rate for each parameter in the network. These learning rates are computed using moving averages of the gradient and the squared gradient (Kingma and Ba [2014]). The following parameters were used for learning rate, decay for the moving averages (β_1, β_2), and fuzz factor (ϵ):

Learning Rate: 0.001

β_1**:** 0.9

[1]`https://www.ubuntu.com/`
[2]`https://www.nvidia.com/download/index.aspx`
[3]`https://developer.nvidia.com/cuda-downloads`
[4]`https://www.python.org/`
[5]`https://www.tensorflow.org/`
[6]`https://scikit-learn.org/stable/index.html`

β_2: 0.999

ϵ: 1e-07

4.3 Model Evaluation

As mentioned in section 3.1, the test data consisted of images where the salmon was not present in the training set. We evaluate our models using the salmon image verification task, which is similar to the evaluation method used in the FaceNet paper by Schroff et al. [2015]. That is, given two images of salmon heads, a and b, the L_2 distance $D(a, b)$ is used with a set threshold t to determine if the images are of the same fish. Let P be the pairs of salmon images. We define the set of all true positives (TP) for a threshold t as:

$$TP(t) = \big\{(a, b) \in P \mid a_{id} = b_{id} \land D(a, b) \leq t\big\} \tag{4.1}$$

Similarly, we can calculate the set of false positives (FP) for a given threshold:

$$FP(t) = \big\{(a, b) \in P \mid a_{id} \neq b_{id} \land D(a, b) \leq t\big\} \tag{4.2}$$

This enables us to calculate the true positive rate (TPR) and the false positive rate (FPR) for a model for a given threshold value:

$$\begin{aligned} TPR(t) &= \frac{\mid TP(t) \mid}{\mid \{(a, b) \in P \mid a_{id} = b_{id}\} \mid} \\ FPR(t) &= \frac{\mid FP(t) \mid}{\mid \{(a, b) \in P \mid a_{id} \neq b_{id}\} \mid} \end{aligned} \tag{4.3}$$

The metrics were computed with approximately 14 million pairs of images from the test set. By computing the values for several different threshold values we can plot Receiver Operating Characteristics (ROC) curves for the models and compare the area under the curve (AUC). The area under the ROC curve represents how well the model is capable of seperating different classes, in this case the fish identities. The higher the AUC the better. When evaluating the models, we used the range from 0 to 2.0 (inclusive) with increments of 0.02.

4.4 Experimental Results

Figure 4.1, 4.2a, and 4.2b show the loss curves during the training of the three models presented in section 3.4. One notable observation in the loss curves for FishNet1 is that both the training and validation loss start to fluctuate and increase greatly towards the middle and end of training. This occurs due to the nature of the triplet selection algorithm used during the training phase. The algorithms only uses triplets that fail the triplet constraint test described in section 3.3. This means that if the model learns to separate salmon well, there are fewer triplets available for training as the training progresses. By examining the training logs we can see that this in fact happens. Figure 4.3 shows show many of the sampled triplets the network was able to use for training.

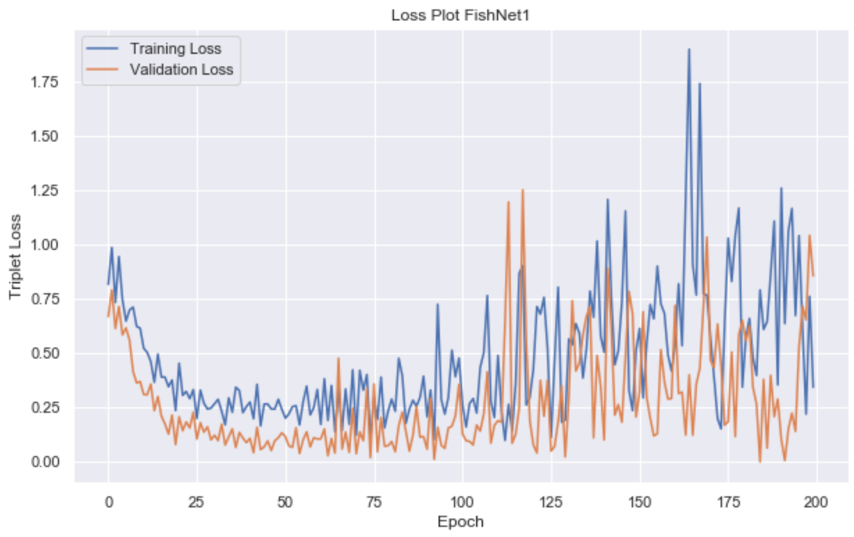
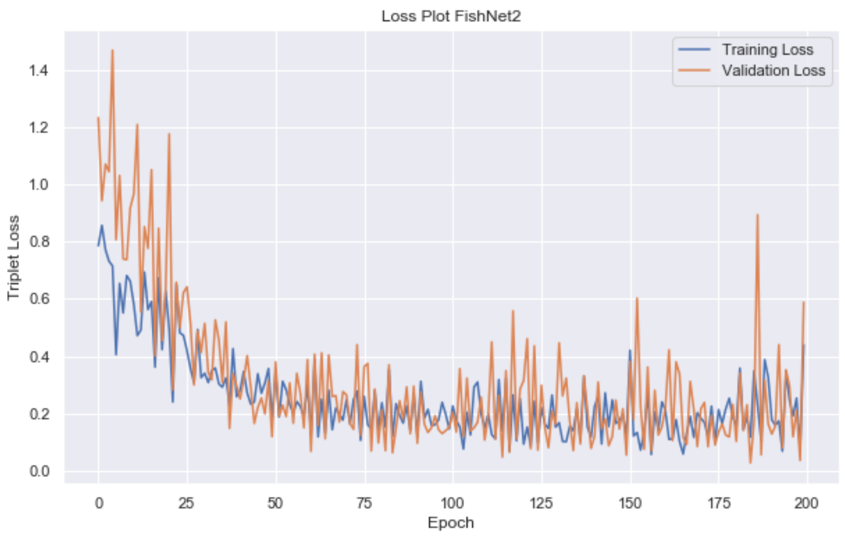
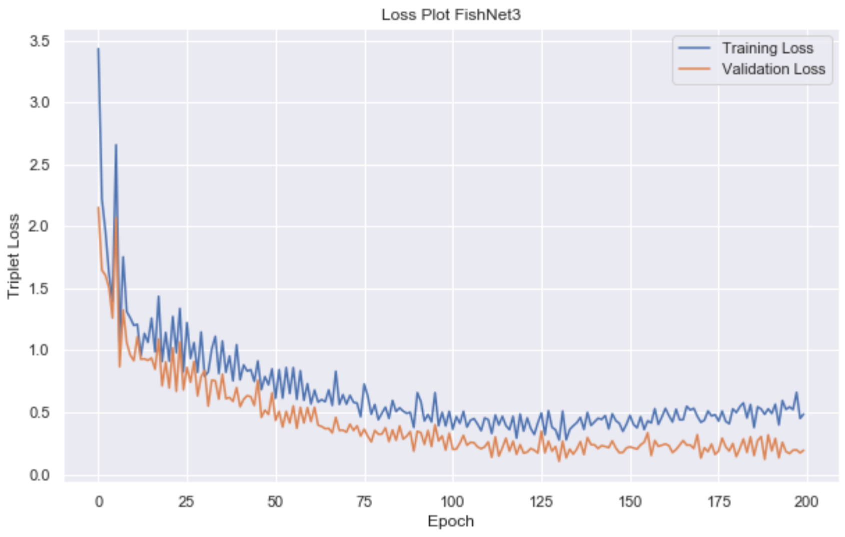


Figure 4.1: The loss curves for FishNet1.



(a) The loss curves for FishNet2.



(b) The loss curves for FishNet3.

Figure 4.2: Loss curves for FishNet2 and FishNet3.

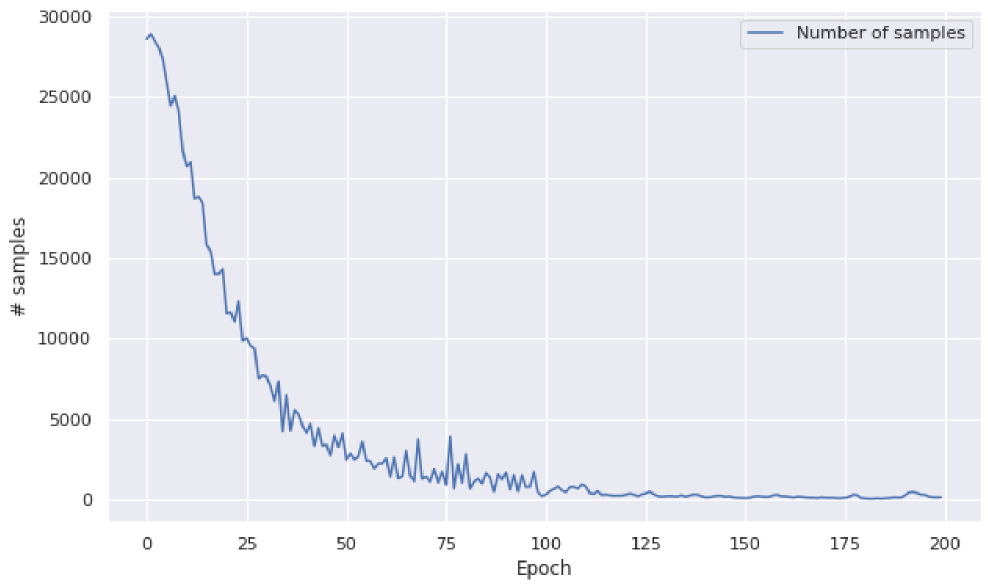


Figure 4.3: Number of triplets available for training each epoch for FishNet1. Towards the end of the training only about 100 samples were available for training.

What might finally happen, is that the model may be stuck with the same training samples. If the model is unable to learn how to separate these, we can end up seeing fluctuating and increasing loss.

The goal of the face verification task is to easily be able to separate the embeddings generated by different identities in the euclidean space. Figure 4.4 illustrates how the embeddings are distributed in the space before and after training. The points in the plots are of 6000 images from 29 different salmon from the test set. The models used are FishNet1 before and after 200 epochs of training. As we can see from both the PCA- and t-SNE-reduced plots the grouping of embeddings from salmon of the same identity is far better after training. This indicates that the model is able to learn some mapping from the images to embeddings.

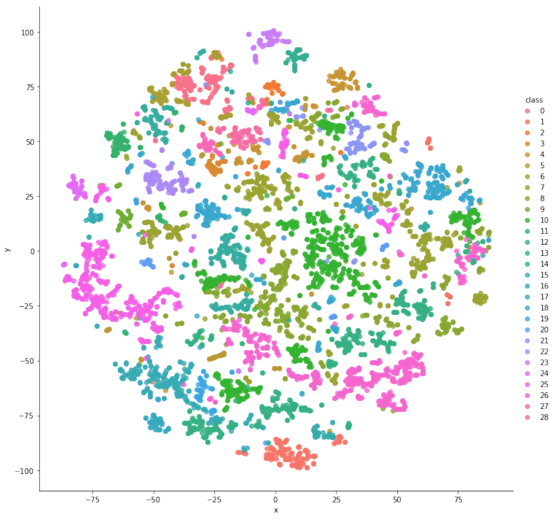
To compute metrics such as true positive rate, false positive rate, accuracy etc., a similarity threshold needs to be set. To compare the models we can examine what the true positive rate (the sensitivity) of the system is at a set false positive rate. We have compared the models where the false positive rate is 0.01, that is, where 1% of the negative samples are misclassified as positive. As we can see in table 4.1 FishNet1 and FishNet2 perform approximately equally with a true positive rate of about 96%. FishNet3 performs significantly worse with a true positive rate of 87%.

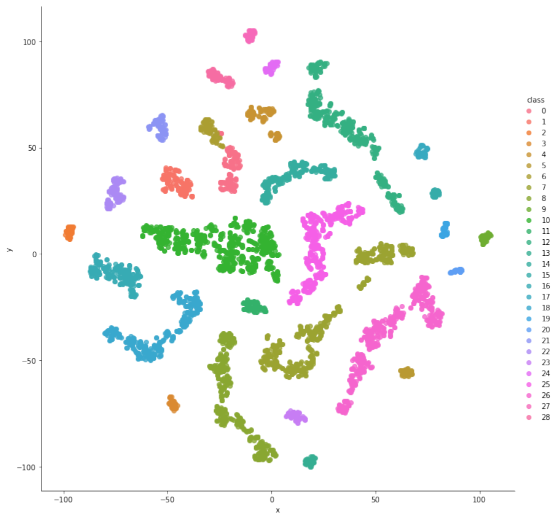
Figure 4.5 shows the ROC curve for the various models tested in this thesis. By

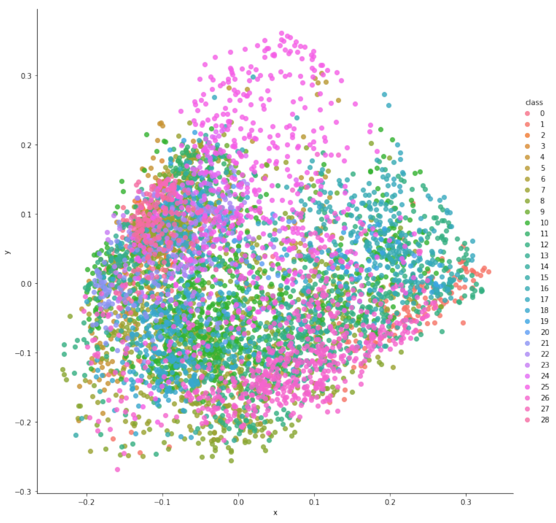
Network Architecture	AUC	TPR @ FPR = 10e-3
FishNet1 (Inception ResNet v2)	0.9977	0.964
FishNet2 (MobileNet v2)	0.9974	0.961
FishNet3 (VGG-16)	0.9919	0.870

Table 4.1: The area under the curve and true positive rate (measured when the false positive rate is 10e-3) of the models.

comparing the area under the curve we can compare the performance of the models across all thresholds. As we can see FishNet1 and FishNet2 perform better than FishNet3, with FishNet1 being the best of the models tested in our experiments. It is interesting to note that the improved results of FishNet1 come at quite a high computational cost compared with FishNet2, a network designed to be able to run on mobile devices.


(a) Before training (t-SNE).


(b) After Training (t-SNE).


(c) Before training (PCA).

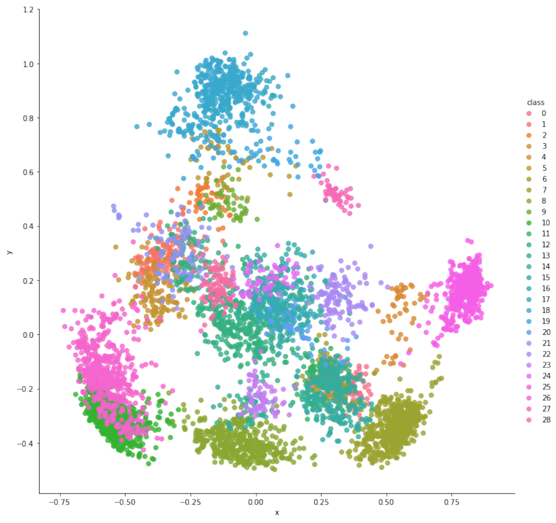

(d) After Training (PCA).

Figure 4.4: The embeddings of 6000 images plotted in 2D-euclidean space. To achieve this, the 128-dimensional embeddings are reduced to two dimensions using t-SNE (first row) and PCA (second row). The color of the points represent the identity of the salmon in the image.

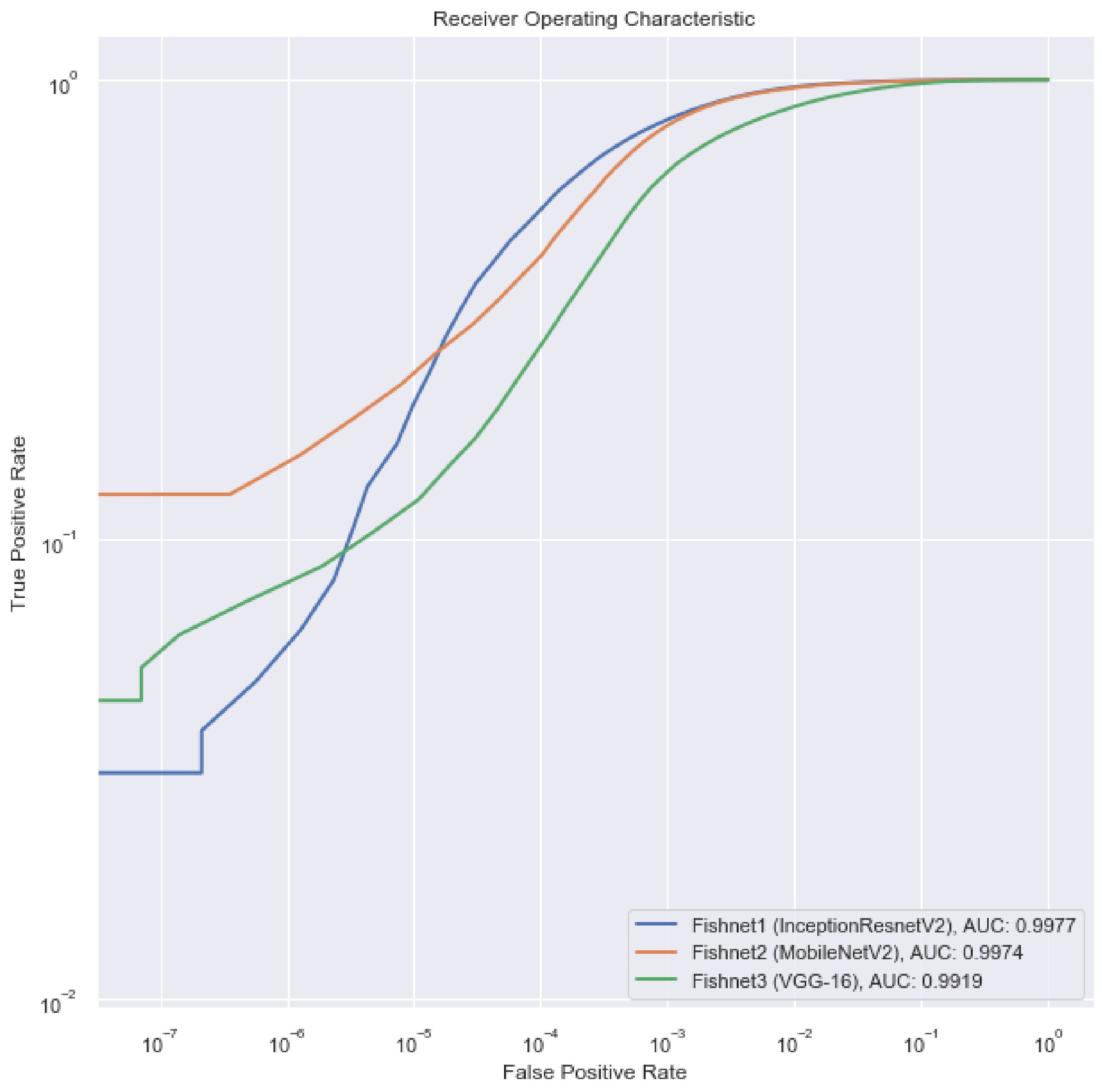


Figure 4.5: The ROC curves of FishNet1, FishNet2, and FishNet3. The true positive rate and false positive rate is computed across similarity thresholds in the range [0.0, 2.0] in increments of 0.2. The model with the largest area under the curve has the best overall performance (FishNet1, with InceptionResnetV2). Note that the axes in the plot are in logarithmic scale.

Chapter 5

Discussion and Conclusion

This chapter presents the conclusions from the work in this thesis. We start off by discussing our results and the limitations we encountered during our work. After the discussion, we present our contributions to the salmon recognition field and their significance. We end the chapter by discussing work that could follow in the future to improve FishNet and LFNW. Finally, we present our thoughts on the steps towards having a stand-alone system able to recognize salmon in sea cages.

5.1 Discussion

In this master thesis we have researched and implemented a state-of-the-art face recognition method on the Atlantic salmon, Salmo Salar. During the thesis we have sought to answer two research questions.

Research question 1 *How well do state-of-the-art face verification methods using end-to-end deep learning perform on salmon in cages?*

To our knowledge, we are the first to implement the recognition of salmon individuals with an end-to-end deep learning approach. In section 4.4, we show our results from the experiments conducted. The best model tested in this thesis has a true positive rate of **96%** at a false positive rate of 1%. The results are promising, and show that convolutional neural network models trained with triplet loss are able to learn embeddings that can be used for salmon identity verification, recognition and clustering. It appears that deep learning is a suitable approach to create a system that automatically identifies salmon without physical interaction.

Figure 5.1 illustrates what distances FishNet1 calculates between three pair of images. From the figure we can see that all distances between the same salmon are considerably shorter than the distances when comparing different salmons. The shortest distance acquired (Egil - Egil comparison) has a distance of 0.287

even though the images have quite different backgrounds and with direct sunlight on the leftmost image. What is more interesting is the fact that Egil has little melanophore spot pattern. We have seen examples of spot-less salmon, and the fact that FishNet has this short distance between them suggests that there are other characteristics that are considered. This is promising with regards to creating a model that can not only recognize salmon with clear distinct melanophore spot patterns, but also the spot-less salmons.

Research question 2 *How suitable are different deep learning architectures and what results do they yield?*

We can observe that the models using residual connections far outperform the simple convolutional architecture. In our work, we have used a representative selection of well known and commonly used convolutional neural network architectures. The work also shows that models created for mobile devices can perform the salmon recognition task with good results. To get a better understanding of the performance of the different FishNet models, we can compare benchmark results of the convolutional parts of the networks. In *Benchmark Analysis of Representative Deep Neural Network Architectures* (Bianco et al. [2018]) the authors compare the models on a Jetson TX1[1], an embedded GPU computer. The results show that MobileNetV2, used in FishNet2, vastly outperforms the other models with an inference time of about 20 ms. Inception Resnet V2 (FishNet1), VGG-16 (FishNet3) have inference times of 199 ms and 152 ms, respectively. The numbers are reported for a batch size of 1.

The results in this thesis should give a good starting point for future exploration into the challenge.

The *Labeled Fish NOT in the Wild* (LFNW) dataset created during our work has some limitations. Ideally, the dataset would contain a lot more data with several bypasses from each salmon in different conditions with respect to water quality, sunlight, distance, rotation etc. We overcame some of the shortcomings by augmenting the images as discussed in section 3.1. Channel shifting the images worked as a simulation for brighter and darker conditions, while rotation and vertical/horizontal shifting increased the overall data at our disposal. It is close to impossible for humans to verify salmon, so acquiring data from different bypasses and labeling this correctly is no easy task.

It would be interesting to be able to evaluate our models on a public standardized dataset that has been used by other researchers. As this does not exist, we had to create our own dataset. This is a time consuming task, which naturally imposes limits on the scale of the dataset. Compared to open source face recognition datasets, LFNW is quite small, and as we noted in section 4.4, especially FishNet1

[1]`https://developer.nvidia.com/embedded/buy/jetson-tx1`

seems to suffer from a lack of challenging training data as training progresses. It is reasonable to assume that a larger and especially more diverse dataset would help the models generalize. Combining video from several salmon cages would help in this regard.

In figure 5.1 we chose images of the fish with many frames apart in order to get images that are not close to identical. We can see that they have quite different backgrounds and pose, but given that both image are from the same passing, the images are quite similar. Doing the same analysis with the same fish, but from different passings of the camera would be valuable to see. It would be nice to have other open source dataset of salmon images, but this does not exist as of this writing.

All the data in the LFNW dataset is from one sea cage. As a result of this we do not know how well a trained model would perform on salmon in a different cage. There is a danger that the models have overfitted to salmon in the cage used for training.

5.2 Contributions and Conclusion

In this thesis we have shown that the state-of-the art methods for face verification in humans show potential in doing the same for salmon. By using end-to-end deep learning we have presented a method which does not require extensive preprocessing of the images, nor relies on handcrafted feature extraction. The experiments are performed on images of salmon in sea cages, rather than strictly controlled laboratory conditions. Other research on the subject, such as the SalmID project (Eilertsen [2017]) focus on gathering data in processing plant, where the salmon are on a conveyor belt. These condition are not necessarily transferable to ocean conditions, because of the large variations of conditions in the sea. Projects such as Hammerset [2018] use images from salmon in sea cages, but rely exclusively on geometric data from the spots on salmon to identify them.

We also see that architectures with residual connections seem to work well as feature extractors for this problem. FishNet2 especially shows that inference can be performed on a device with relatively low memory capacity and computing power. This indicates that salmon recognition should be possible to implement on site without prohibitively high costs. All models are trainable over night on the system described in section 4.2. Relatively fast training times ensures flexibility and customizability for models when deployed in sea pens. The small size of the embeddings ensures that the salmon recognition task can be performed locally without expensive hardware.

The dataset created during this thesis will hopefully make it easier to continue the work on creating a system to recognize salmon. It will also make it possible to compare future models with the ones presented in this thesis. The images are of high quality and are taken in a realistic environment, so we believe it is a good starting point for creating systems to be used in sea cages.

5.3 Future Work

This final section contains our thoughts about the future of LFNW and FishNet, as well as what steps we believe need to be taken to create an on site system for the industry.

5.3.1 Improving the LFNW dataset

There is still a lot of work that can be done to improve and increase the size of the LFNW dataset. Given that our method is data driven, improving the dataset would be a natural starting point for improving the model in the future. Adding images of salmon from different sea cages is an obvious starting point. Images from all the life stages of the salmon would also be useful to make the system more universal. There are also several image augmentation techniques that could be useful to expand the size of the dataset. Introducing noise, blurring, and artificial sun glare are examples of this. The more training data that is available, the better the models should be able to generalize. As we saw in figure 4.3, especially FishNet1 seems to be suffering from a lack of training data.

Acquiring data of salmon from several passes with only the video feed at disposal, could be realized by following these steps:

1. Convert the first few minutes of video (to guarantee that no salmon has passed twice) to a labeled dataset using the steps described in section 3.1.
2. Train the model on this data.
3. Convert the next few minutes to a dataset, and seek to recognize the salmon.
4. Manually compare salmon with suspiciously low distance value between them, and label them accordingly.
5. Train the model on all the data created this far.
6. Repeat step 3 to 5 of the process until the complete video stream is converted to a labeled dataset.

However, this process is prone to human errors. As we have mentioned, verifying the identity of two salmon is not easy, so attempting this could introduce errors to the dataset.

5.3.2 Future of FishNet

It would be interesting to see a class activation heat map of the system. As we are doing end-to-end deep learning, a class activation heat map would give insight as to what part of the salmon the system emphasizes on when learning the different embeddings. This could give further insight into which parts of salmon are useful for distinguishing them from each other.

Investigating other convolutional architectures than the ones presented here would also be interesting. Searching for better hyper parameters for the models we have discussed is also something that could yield even better results.

5.3.3 Towards a Commercialized Salmon Recognition System

There is a lot that has to be done in order to make a viable individual salmon recognition system that can be deployed and used in commercial sea cages.

Recognizing salmon requires images of high quality. Challenges when obtaining those includes water quality, sunlight, focus and camera resolution.

As noted earlier, salmon are not symmetrical when it comes to the melanophore pattern on each side. Side recognition may satisfy the need of recognition in sea cages, but if the systems needs to be able to identify each individual from either side, the initial labeled training examples for each side would have to be acquired somehow.

However, from a machine learning aspect, this kind of system should be possible given the correct conditions mentioned above. By using a preset threshold value for embedding similarity, the system can determine if a salmon has been recorded before. YOLOv3 can create bounding-boxes for salmon heads in real-time, and from having the bounding-box to creating an embedding and compare the embedding with the existing one should not encounter major challenges with state-of-the-art hardware. Combined with modern databases and information retrieval techniques we believe it to be entirely possible to create a salmon recognition system with real-time performance on site.

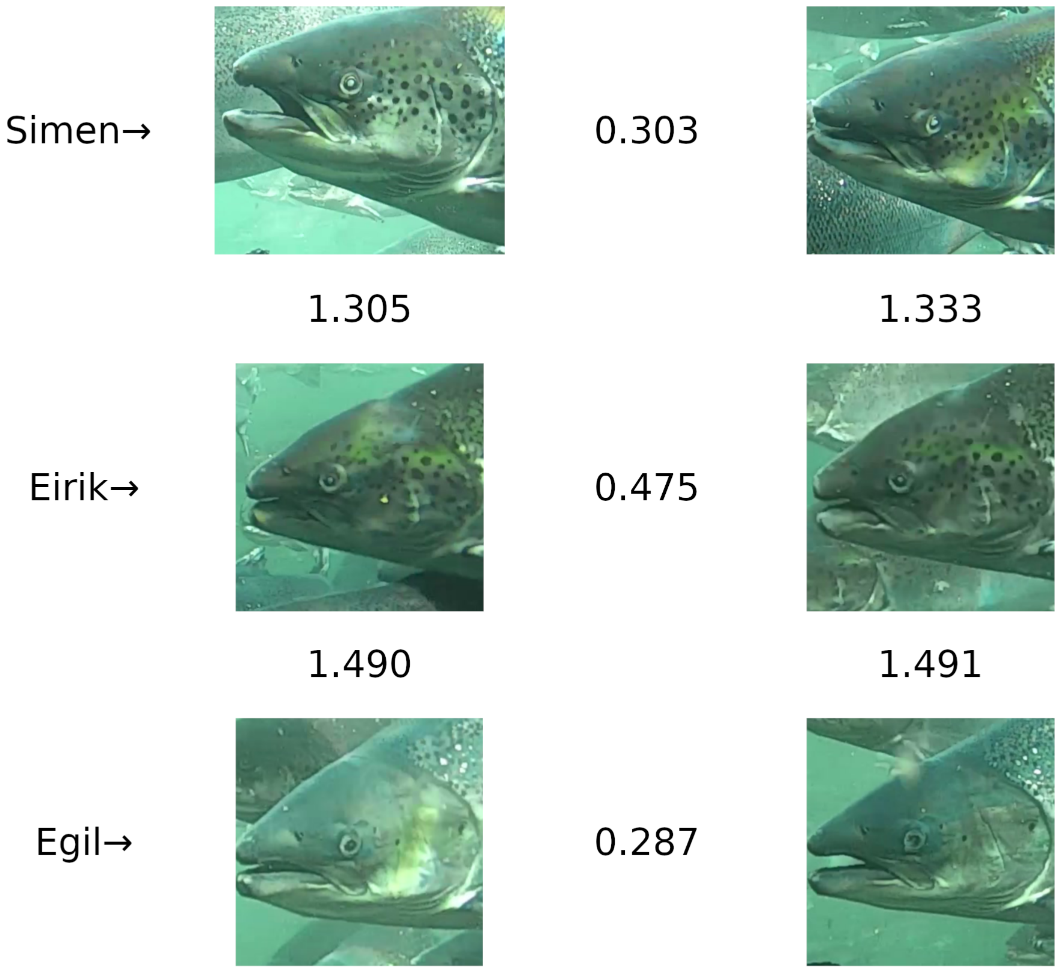


Figure 5.1: An illustration of the distances between six images from salmon with three different identities. Each row contains two images of the same salmon: Simen at the top, Eirik in the middle and Egil at the bottom. The average distance between the same salmon is 0.36 while comparisons between different salmon average at 1.40.

Bibliography

Bianco, S., Cadene, R., Celona, L., and Napoletano, P. (2018). Benchmark analysis of representative deep neural network architectures. *IEEE Access*, 6:64270–64277.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*.

Eilertsen, A. B. (2017). Identifikasjon av lakseindivider — biometri fase 1 (salmid). *Publikasjoner fra CRIStin - SINTEF Ocean*.

Ester, M., Kriegel, H.-P., Sander, J., Xu, X., et al. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press.

Gregor, K. and LeCun, Y. (2010). Emergence of complex-like cells in a temporal product network with local receptive fields. *arXiv preprint arXiv:1006.0448*.

Hammerset, I. (2018). Biometric recognition and individual tracking of salmon in large-scale sea cages.

Hansen AÅ, Rødbotten M, E. T. L. P. R. K. M. T. (2012). The effect of crowding stress on bacterial growth and sensory properties of chilled Atlantic salmon fillets. *Journal of food science*, 77(1):S84–90.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

Hotelling, H. (1933). Analysis of a complex of statistical variables into principal components. *Journal of educational psychology*, 24(6):417.

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

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Li, F.-F., JustinJohnson, and Yeung, S. (2018). Cs231n: Convolutional neural networks for visual recognition, lecture: Convolutional neural networks (cnns / convnets).

Maaten, L. v. d. and Hinton, G. (2008). Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605.

March, S. T. and Smith, G. F. (1995). Design and natural science research on information technology. *Decision support systems*, 15(4):251–266.

Merz, J. E., Skvorc, P., Sogard, S. M., Watry, C., Blankenship, S. M., and Nieuwenhuyse, E. E. V. (2012). Onset of melanophore patterns in the head region of chinook salmon: A natural marker for the reidentification of individual fish. *North American Journal of Fisheries Management*, 32(4):806–816.

Redmon, J. and Farhadi, A. (2018). Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*.

Richardsen, R., Stoud Myhre, M., Bull-Berg, H., and T. Grindvoll, I. L. (2018). Nasjonal betydning av sjømatnæringen. *Publikasjoner fra CRIStin - SINTEF Ocean*.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. (2018). Mobilenetv2: Inverted residuals and linear bottlenecks.

Santurkar, S., Tsipras, D., Ilyas, A., and Madry, A. (2018). How does batch normalization help optimization? In *Advances in Neural Information Processing Systems*, pages 2483–2493.

Schroff, F., Kalenichenko, D., and Philbin, J. (2015). Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 815–823.

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

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.

SSB (2018). Akvakultur - årlig, endelige tall - ssb.

Stien, L. H., Nilsson, J., Bui, S., Fosseidengen, J. E., Kristiansen, T. S., Øverli, Ø., and Folkedal, O. (2017). Consistent melanophore spot patterns allow long-term individual recognition of Atlantic salmon Salmo salar. *Journal of Fish Biology*, 91(6):1699–1712.

Szegedy, C., Ioffe, S., Vanhoucke, V., and Alemi, A. A. (2017). Inception-v4, inception-resnet and the impact of residual connections on learning. In *Thirty-First AAAI Conference on Artificial Intelligence*.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9.

Taigman, Y., Yang, M., Ranzato, M., and Wolf, L. (2014). 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.

Wohlin, C. (2014). Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *Proceedings of the 18th international conference on evaluation and assessment in software engineering*, page 38. ACM.

Appendix A

Appendices

A.1 Dataset

A.2 Architecture Code

models.py   Thu May 23 12:31:30 2019   1

```
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import (
    Dense,
    Dropout,
    Flatten,
    Input,
    Conv2D,
    MaxPooling2D,
    Layer,
    BatchNormalization,
    Lambda,
)
import tensorflow.keras.backend as K
import numpy as np
from tensorflow.keras.applications.inception_resnet_v2 import (
    preprocess_input as resnet_preprocess_input,
)
from tensorflow.keras.applications.vgg16 import preprocess_input as vgg_preprocess_input
from tensorflow.keras.models import load_model
from tensorflow.keras.applications.mobilenet_v2 import (
    preprocess_input as mobile_preprocess_input,
)


class TripletLossLayer(Layer):
    """ Layer to compute triplet loss.

    This layer computes triplet loss as it is defined
    in the FaceNet paper.

    Parameters:
        alpha (float): Threshold value.

    Input shape:
        List of anchor, positive and negative embeddings.

    """

    def __init__(self, alpha, **kwargs):
        self.alpha = alpha
        super(TripletLossLayer, self).__init__(**kwargs)

    def triplet_loss(self, inputs):
        a, p, n = inputs
        p_dist = K.sum(K.square(a - p), axis=-1)
        n_dist = K.sum(K.square(a - n), axis=-1)
        return K.sum(K.maximum(p_dist - n_dist + self.alpha, 0), axis=0)

    def call(self, inputs):
        loss = self.triplet_loss(inputs)
        self.add_loss(loss)
        return loss


normalize = Lambda(lambda x: K.l2_normalize(x, axis=-1), name="normalize")


def create_resnet_model(image_dim, use_pretrained=True):
    """ Create FishNet1 model

    Creates a FishNet model that uses Inception Resnet V2 for the
    convolutional part.

    Parameters:
        image_dim (tuple): Tuple of image shape, i.e. (175, 175, 3).
        use_pretrained (bool):  Default true, should imagenet weights be
                                used?
```

models.py   Thu May 23 12:31:30 2019   2

```
    Returns:
        tuple:  Tuple with (inference_model, triplet_loss_model,
                preprocessing_func)
    """

    my_input = Input(shape=image_dim)
    if use_pretrained:
        weights = "imagenet"
    else:
        weights = None
    pretrained = tf.keras.applications.inception_resnet_v2.InceptionResNetV2(
        include_top=False, weights=weights, input_tensor=my_input, pooling="avg"
    )
    conv_7b_ac = pretrained.get_layer("conv_7b_ac").output
    flattened = Flatten(name="flattened")(conv_7b_ac)
    embedding = Dense(128, name="embedding_layer")(flattened)
    embedding = normalize(embedding)

    # create our model
    model = Model(inputs=[my_input], outputs=embedding)

    input_a = Input(shape=image_dim)
    input_p = Input(shape=image_dim)
    input_n = Input(shape=image_dim)

    embedding_a = model(input_a)
    embedding_p = model(input_p)
    embedding_n = model(input_n)

    triplet_loss_layer = TripletLossLayer(alpha=0.2, name="triplet_loss")(
        [embedding_a, embedding_p, embedding_n]
    )

    triplet_loss_model = Model([input_a, input_p, input_n], triplet_loss_layer)
    return model, triplet_loss_model, resnet_preprocess_input


def create_vgg_model(image_dim, use_pretrained=True):
    """ Create FishNet3 model

    Creates a FishNet model that uses VGG-16 for the
    convolutional part.

    Parameters:
        image_dim (tuple): Tuple of image shape, i.e. (175, 175, 3).
        use_pretrained (bool):  Default true, should imagenet weights be
                                used?

    Returns:
        tuple:  Tuple with (inference_model, triplet_loss_model,
                preprocessing_func)
    """

    my_input = Input(shape=image_dim)
    if use_pretrained:
        weights = "imagenet"
    else:
        weights = None
    pretrained = tf.keras.applications.vgg16.VGG16(
        include_top=False, weights=weights, input_tensor=my_input, pooling="avg"
    )
    x = pretrained.layers[-1].output
    embedding = Dense(128, name="embedding_layer")(x)
    embedding = normalize(embedding)
    model = Model(inputs=[my_input], outputs=embedding)

    input_a = Input(shape=image_dim)
```

models.py   Thu May 23 12:31:30 2019   3

```
    input_p = Input(shape=image_dim)
    input_n = Input(shape=image_dim)

    embedding_a = model(input_a)
    embedding_p = model(input_p)
    embedding_n = model(input_n)

    triplet_loss_layer = TripletLossLayer(alpha=0.2, name="triplet_loss")(
        [embedding_a, embedding_p, embedding_n]
    )

    triplet_loss_model = Model([input_a, input_p, input_n], triplet_loss_layer)

    return model, triplet_loss_model, vgg_preprocess_input


def create_mobile_model(image_dim, use_pretrained=True):
    """ Create FishNet2 model

    Creates a FishNet model that uses MobileNetv2 for the
    convolutional part.

    Parameters:
        image_dim (tuple): Tuple of image shape, i.e. (175, 175, 3).
        use_pretrained (bool):  Default true, should imagenet weights be
                                used?

    Returns:
        tuple:  Tuple with (inference_model, triplet_loss_model,
                preprocessing_func)
    """

    my_input = Input(shape=image_dim)
    if use_pretrained:
        weights = "imagenet"
    else:
        weights = None
    pretrained = tf.keras.applications.mobilenet_v2.MobileNetV2(
        include_top=False, weights=weights, input_tensor=my_input, pooling="avg"
    )
    x = pretrained.layers[-1].output
    embedding = Dense(128, name="embedding_layer")(x)
    embedding = normalize(embedding)
    model = Model(inputs=[my_input], outputs=embedding)

    input_a = Input(shape=image_dim)
    input_p = Input(shape=image_dim)
    input_n = Input(shape=image_dim)

    embedding_a = model(input_a)
    embedding_p = model(input_p)
    embedding_n = model(input_n)

    triplet_loss_layer = TripletLossLayer(alpha=0.2, name="triplet_loss")(
        [embedding_a, embedding_p, embedding_n]
    )

    triplet_loss_model = Model([input_a, input_p, input_n], triplet_loss_layer)

    return model, triplet_loss_model, mobile_preprocess_input
```

A.3 Model Evaluation

Model Evaluation

May 21, 2019

[5]:
```
import pandas as pd
import matplotlib
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from scipy.spatial import distance
from collections import defaultdict
from scipy.spatial.distance import pdist, squareform
from sklearn.metrics import auc
sns.set()
```

0.0.1 Load testdata, create distance matrix and class matrix

[6]:
```
def create_tpr_fpr(path):
    test_df = pd.read_csv(path)
    test_data = []
    for group in test_df.groupby("class"):
        test_data.append(group[1].iloc[0:100])
    df = pd.concat(test_data)

    distances = pdist(df.drop("class", axis=1), metric='euclidean')
    dist_matrix = squareform(distances)

    same_fish = pdist(df, metric=lambda u, v: u[-1] == v[-1])
    same_fish_matrix = squareform(same_fish)
    same_fish_matrix = same_fish_matrix + np.eye(same_fish_matrix.shape[0])

    n_same = np.count_nonzero(same_fish_matrix)
    n_diff = np.size(same_fish_matrix) - np.count_nonzero(same_fish_matrix)

    tprs = []
    fprs = []

    for t in np.arange(0.0,2.02,0.02):
        predict_is_same = dist_matrix <= t
        true_positive_mx = np.logical_and(predict_is_same, same_fish_matrix)
        n_true_positive = np.count_nonzero(true_positive_mx)
        false_positive_mx = np.logical_and(predict_is_same, np.
 ↪logical_not(same_fish_matrix))
        n_false_positive = np.count_nonzero(false_positive_mx)
        tprs.append(n_true_positive / n_same)
        fprs.append(n_false_positive / n_diff)
    return tprs, fprs
```

0.0.2 Compute for different embeddings

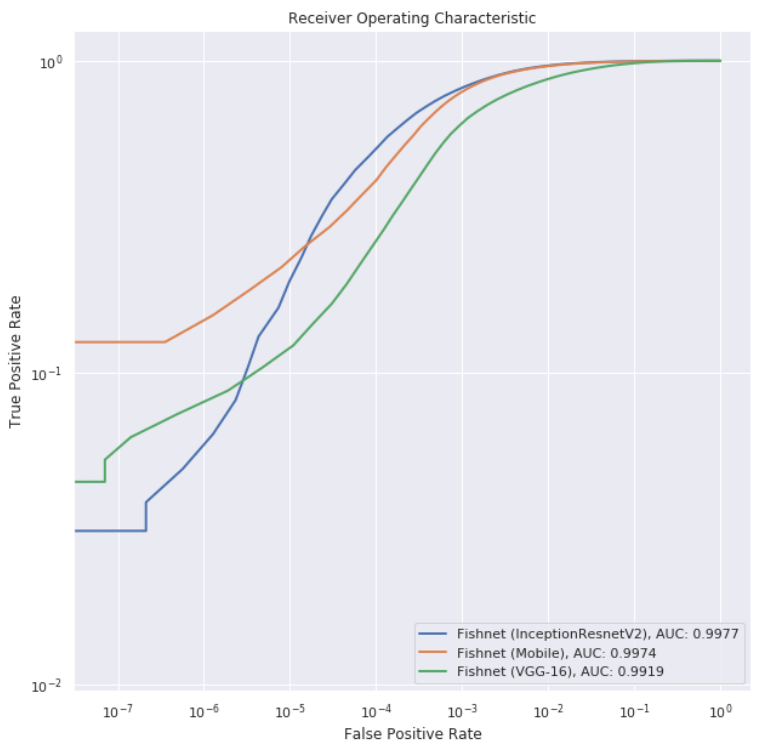
[7]:
```
%%time
trained_resnet_tprs, trained_resnet_fprs = create_tpr_fpr("embeddings/
 ↪nn_resnet_pre_200.csv")
trained_vgg_tprs, trained_vgg_fprs = create_tpr_fpr("embeddings/
 ↪nn_vgg-16_pre_200.csv")
trained_mobile_tprs, trained_mobile_fprs = create_tpr_fpr("embeddings/
 ↪nn_mobile_pre_200.csv")
```

```
CPU times: user 1min 13s, sys: 1.6 s, total: 1min 15s
Wall time: 1min 15s
```

[8]:
```
trained_resnet_auc = np.round(auc(trained_resnet_fprs, trained_resnet_tprs), 4)
trained_vgg_auc = np.round(auc(trained_vgg_fprs, trained_vgg_tprs), 4)
trained_mobile_auc = np.round(auc(trained_mobile_fprs, trained_mobile_tprs), 4)
```

0.0.3 Plot that shit

[9]:
```
plt.figure(figsize=(10, 10))
plt.yscale('log')
plt.xscale('log')
plt.plot(trained_resnet_fprs,
         trained_resnet_tprs,
         lw=2,
         label=f"Fishnet (InceptionResnetV2), AUC: {trained_resnet_auc}")
plt.plot(trained_mobile_fprs,
         trained_mobile_tprs,
         lw=2,
         label=f"Fishnet (Mobile), AUC: {trained_mobile_auc}")
plt.plot(trained_vgg_fprs,
         trained_vgg_tprs,
         lw=2,
         label=f"Fishnet (VGG-16), AUC: {trained_vgg_auc}")
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()
```



0.0.4 TPR @ FPR

```
[10]: trained_resnet_tprs[np.abs(np.array(trained_resnet_fprs)-10e-3).argmin()]
```

```
[10]: 0.9641228851291185
```

```
[11]: trained_vgg_tprs[np.abs(np.array(trained_vgg_fprs)-10e-3).argmin()]
```

```
[11]: 0.869893143365984
```

```
[12]: trained_mobile_tprs[np.abs(np.array(trained_mobile_fprs)-10e-3).argmin()]
```

```
[12]: 0.9610106856634016
```

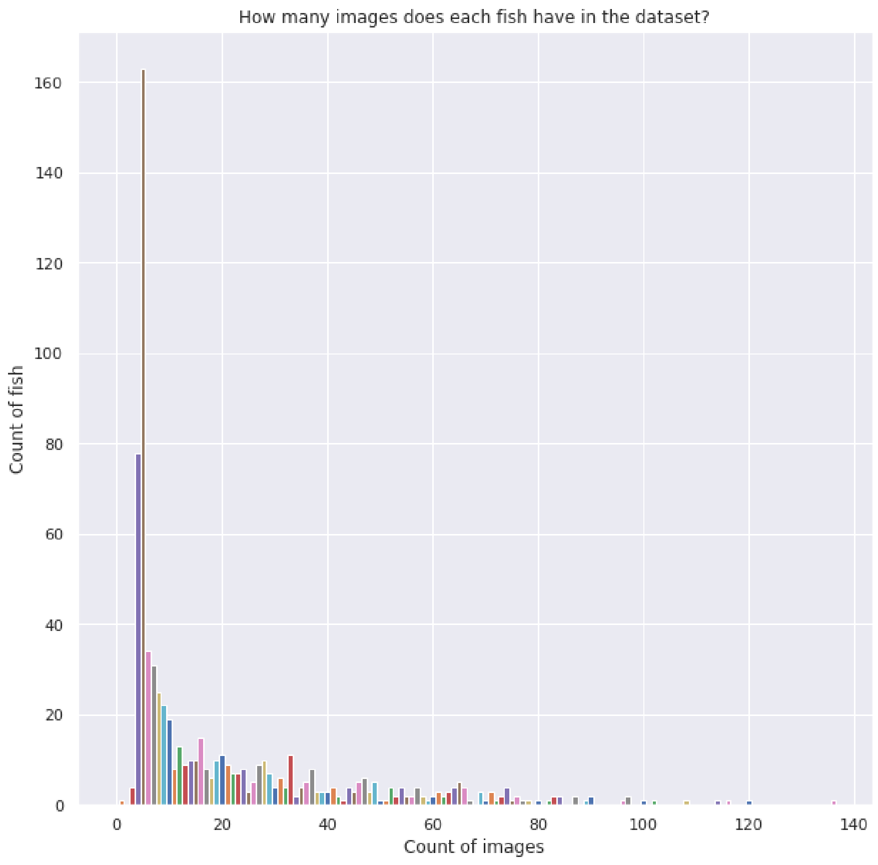


Figure A.1: The number of images per fish in the dataset.