UNIVERSITY OF AGDER

Biometric Fish Classification of Nordic Species Using Convolutional Neural Network with Squeeze-and-Excitation

Christian M. D. Trinh and Erlend Olsvik

SUPERVISORS
Ph.D. Morten Goodwin
Ph.D. Lei Jiao

Master's Thesis
University of Agder, June 2018
Faculty of Engineering and Science
Department of ICT

UiA
University of Agder
Master's thesis

Faculty of Engineering and Science
Department of ICT

Abstract

Squeeze-and-Excitation (SE) is a technique within convolutional neural networks (CNN) that can be applied to existing CNNs by applying fully-connected layers between convolutional layers and merging the outputs. SE was the winning architecture of the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2017. In this thesis, we propose a CNN using the SE architecture for classifying images of fish. Previous work in the field relies on applying filters to the images to separate the fish from the background or sharpen the images by removing background noise. The images from the dataset are extracted from underwater cameras and contain noise, which is why classifying these images is challenging. Different from conventional schemes, this approach is divided into two classification problems. The first approach is to classify fish from the Fish4Knowledge dataset without using image augmentation, and the second is to classify fish from a new dataset consisting of Nordic species. We name the first approach pre-training, and the second post-training. The weights from pre-training are applied to post-training.
Our solution achieves the state-of-the-art accuracy of 99.27% accuracy on the pre-training. The accuracy on the post-training is lower with an accuracy of 83.68%. Experiments on the post-training with image augmentation yields an accuracy of 87.74%, indicating that the solution is viable with a larger dataset.

Keywords: Classification, CNN, Squeeze-and-Excitation

Acknowledgements

This thesis has been submitted as a partial fulfillment of the Master's Degree in Information and Communication Technology at the University of Agder.
For this thesis to be possible, several people have been involved. First of all, we would like to thank Associate Professor Morten Goodwin at the University of Agder for being our supervisor and aiding us at any hour of the day. His expertise in the field of deep learning and report writing has been of great help. We would also like to thank our co-supervisor, Associate Professor Lei Jiao for his aid in the report writing and his invaluable feedback on our work.
Furthermore, we would like to thank Assistant Professor Kristian Muri Knausgård at the University of Agder, for reaching out to IMR and helping us shape our problem statement. Without his help and genuine interest in artificial intelligence, this thesis would not be possible.
A big thanks to Centre for Artificial Intelligence at the University of Agder and the people behind it for giving us access to resources so that we could test our solution and perform all of our experiments.
Our biggest gratitude goes out the people at IMR for devoting countless hours of their spare time to manually create the dataset of Nordic fish species for us, as well as providing us with a deeper understanding of marine research. The people involved from IMR were Alf Ring Kleiven, Torkel Larsen, Kim Halvorsen and Tonje K. Sørdalen. Without the help of these individuals, we would not have a dataset of Nordic fish to use.
We would also like to thank our friend and fellow student Martin Holen for his continuous help when we were facing issues.
Lastly, we would like to thank our families for their support throughout this thesis, and for shaping us to become the hard workers that we have become.

Contents

Abstract iii

Acknowledgements v

Glossary ix

List of Figures xii

List of Tables xiii

1 Introduction 1
- 1.1 Challenges . . . 2
- 1.2 Solution . . . 2
- 1.3 Goals . . . 3
  - 1.3.1 Research Questions . . . 3
  - 1.3.2 Hypotheses . . . 3
- 1.4 Contributions . . . 4
- 1.5 Background . . . 4
- 1.6 Report Outline . . . 5

2 Theoretical Background 7
- 2.1 Convolutional Neural Networks . . . 7
  - 2.1.1 Activation Functions . . . 11
  - 2.1.2 Pooling . . . 12
  - 2.1.3 Dropout . . . 13
  - 2.1.4 Classification Functions . . . 14
  - 2.1.5 Loss Functions . . . 15
- 2.2 State-of-the-Art . . . 16
  - 2.2.1 Convolutional Neural Networks . . . 18

3 Approach and Implementation 25

3.1 Datasets . . . 26
3.1.1 Fish4Knowledge Dataset . . . 26
3.1.2 Nordic Fish Species Dataset . . . 27
3.2 Pre-processing . . . 29
3.3 CNN-SENet Architecture . . . 30
3.3.1 Dropout . . . 31
3.3.2 Batch Normalization . . . 32
3.3.3 Learning Rate . . . 33
3.4 CNN-SENet vs. DeepFish . . . 35

4 Experiments and Numerical Results 37
4.1 Configurations . . . 37
4.1.1 Pre-training . . . 38
4.1.2 Post-training . . . 38
4.1.3 Post-training with Image Augmentation . . . 39
4.2 Results . . . 40
4.2.1 Pre-training . . . 40
4.2.2 Post-training . . . 44
4.2.3 Post-training with Image Augmentation . . . 50

5 Conclusion and Further Work 55
5.1 Conclusion . . . 55
5.2 Future Work . . . 56

References 63

Appendices 65
A CNN-SENet Source-code . . . 65
A.1 cnn_senet.py . . . 65
A.2 se.py . . . 67

Glossary

API Application Programming Interface. 25

C NMR C Nuclear Magnetic Resonance. 16

CNN Convolutional Neural Network. xi, 1–3, 5, 7, 8, 10, 11, 13–19, 22, 25, 29, 30, 40–42, 48, 55, 56

CNN-SENet The name of our solution. viii, xi–xiii, 25, 26, 28–35, 37, 40–44, 47–50, 52, 53, 55, 56

Fast R-CNN Fast Region-based Convolutional Network. 17

ILSVRC ImageNet Large Scale Visual Recognition Competition. 4, 16, 19–21, 23

IMR Institute of Marine Research. 1, 2, 4, 5, 27, 56

PNN Probabilistic Neural Network. 16

ReLU Rectified Linear Unit. xi, 11, 12, 19, 21, 30, 31

SE Squeeze-and-Excitation. 2, 21, 22, 25, 30, 35, 41, 42, 47, 48, 53, 55, 56

SENet Squeeze-and-Excitation network. 21, 22

SGD Stochastic Gradient Descent. 22

SVM Support Vector Machine. 14–17

List of Figures

2.1 Architecture of the LeNet-5 network [27]. 8
2.2 Convolving the 5×5 filter on the input image, creating one feature map f_0 [2]. Each time the filter is applied to the receptive field yields one neuron in the feature map. 9
2.3 Example of zero-padding [9]. 10
2.4 Figure showing the entire process of a CNN [8]. The smaller box in the input image is the receptive field. 10
2.5 The sigmoid activation function and its graphs [2]. 11
2.6 The ReLU activation function, its graph and its derivative graph [2]. 12
2.7 Example of max-pooling with stride = 2 and size = 2 [2]. . . 13
2.8 Illustration of dropout [41]. On the left: neural net without dropout. On the right: neural net with dropout. 14
2.9 The DeepFish architecture [35]. 17
2.10 The process of detecting a fish with CatchMeter [45]. 18
2.11 The AlexNet architecture [26]. 19
2.12 The design principle of ResNet [11]. 20
2.13 Inception-ResNet-v2 module A [42]. 21
2.14 A Squeeze-and-Excitation block [13]. 22

3.1 File structure of the datasets. 26
3.2 Distribution of the Fish4Knowledge dataset [14]. 27
3.3 Distribution of the second dataset. 28
3.4 Four images of wrasse female and sneakers. 28
3.5 The architecture of CNN-SENet. 30
3.6 10% Dropout. 31
3.7 50% Dropout. 31
3.8 80% Dropout. 31
3.9 0.5 dropout without batch normalization 32
3.10 0.5 Dropout with batch normalization 32
3.11 Learning rate of 0.1 with a decay of 0.001. 33

3.12 Learning rate of 0.01 with a decay of 0.0001. 33
3.13 Learning rate of 0.001 without decay. 33

4.1 Inception-V3. 40
4.2 ResNet-50. 40
4.3 Inception-Resnet-V2. 41
4.4 CNN-SENet. 41
4.5 CNN-SENet Without Squeeze-and-Excitation. 41
4.6 Confusion matrix for pre-training with CNN-SENet. 43
4.7 Boxplot showing the testing accuracies over ten runs with the different networks. 45
4.8 Inception-V3. Average training, validation and testing accuracies over 10 runs. 45
4.9 ResNet-50. Average training, validation and testing accuracies over 10 runs. 46
4.10 Inception-ResNet-V2. Average training, validation and testing accuracies over 10 runs. 46
4.11 CNN-SENet. Average training, validation and testing accuracies over 10 runs. 47
4.12 CNN-SENet without Squeeze-and-Excitation. Average training, validation and testing accuracies over 10 runs. 47
4.13 Confusion matrix for post-training with CNN-SENet. 49
4.14 Boxplot showing the testing accuracies when using image augmentation over ten runs. 50
4.15 Inception-V3 with augmentation. Average training, validation and testing accuracies over 10 runs. 51
4.16 ResNet-50 with augmentation. Average training, validation and testing accuracies over 10 runs. 51
4.17 Inception-ResNet-V2 with augmentation. Average training, validation and testing accuracies over 10 runs. 52
4.18 CNN-SENet with augmentation. Average training, validation and testing accuracies over 10 runs. 52
4.19 CNN-SENet without Squeeze-and-Excitation with augmentation. Average training, validation and testing accuracies over 10 runs. 53

List of Tables

3.1 Differences between CNN-SENet and DeepFish. 35

4.1 Testing accuracy and time per epoch on pre-training. 40
4.2 Average testing accuracy over 10 runs and time per epoch on post-training. 44
4.3 Average testing accuracy over 10 runs and time per epoch on post-training with image augmentation. 50

Chapter 1

Introduction

The main fields of research in this thesis are convolutional neural networks (CNNs) and image classification, specifically classification of fish species. CNN is a popular area of research within the field of artificial intelligence and image classification. It can be used to detect and classify objects by simply supplying a computer program with enough data and set up this network to train itself in order to solve the given classification problem. This area of research has many potential commercial usages, and includes traffic sign recognition and object detection in autonomous transport. It can also be used in the existing surveillance system the Norwegian Institute of Marine Research (IMR) [18] uses in some of their research projects [10][40] to classify the fish species, removing the need to manually classify these and potentially increase the recognition accuracy.

The goal of this thesis is divided into two classification problems, where the first approach is to classify fish from an existing dataset called Fish4Knowledge [14], which consists of 23 fish species, and is further explained in Section 3.1.1. The first approach will throughout the thesis be referred to as pre-training, because the results of the first approach will be used in the second. The second approach is to classify the species from a dataset consisting of Nordic fish species with the weights from pre-training, hereby referred to as post-training.

1.1 Challenges

Some previous solutions have been proposed to the pre-training using CNNs [35][23]. The biggest challenge with these images is the noise caused by underwater cameras. The previous solutions rely heavily on pre-processing of the training images to remove this noise, which can negatively impact the classification accuracy when deployed to a live scenario. The dataset used in both articles has a very limited amount of images for some of the species. CNNs need large amounts of training data to perform well and to not suffer from a concept called overfitting, which will be further explained in Chapter 2.

There are no publicly available datasets for the post-training, which means that IMR had to extract individual fish from the video feeds for the solution to be applicable to their surveillance system. Manually looking through video feeds is a time-consuming process for researchers in the fish industry, and the process can be improved significantly by applying a CNN. The dataset given by IMR will be reviewed in Subsection 3.1.2, but the challenges with this dataset are few images to train the CNN on, and high variation in the data for each species.

1.2 Solution

To solve the biometric fish classification problem, a CNN with the Squeeze-and-Excitation (SE) architecture [13] is developed and pre-trained on the Fish4Knowledge dataset. The resulting weights from the pre-training are then applied to the new dataset for the Nordic fish species received from IMR. Our solution differs from previous solutions by not using filters to manipulate the dataset. Instead, state-of-the-art technologies within CNNs have been tested and applied to our solution. The specific goal that this project aims to achieve is defined in Section 1.3.

1.3 Goals

The main goal of this thesis is to explore whether a CNN using the SE architecture can be used to classify different Nordic fish species. The approach taken is to pre-train the network on the Fish4Knowledge dataset and apply those weights to continue training on the new dataset containing the Nordic fish species. Specific research questions to be answered are defined in the next subsection, and its following subsection will introduce hypotheses to get a clearer understanding of what this thesis aims to achieve.

1.3.1 Research Questions

The research questions defined below are the specific problems we aim to solve throughout this thesis. The experiments and discussion will revolve around whether the solution achieves these goals. Two research questions are defined.

1. Is it possible to achieve state-of-the-art accuracy on the Fish4Knowledge dataset without data augmentation?
2. Can the convolutional neural network from the pre-training and its weights be used to classify Nordic fish species, achieving the same accuracy as during pre-training?

1.3.2 Hypotheses

The hypotheses provide details to support the research questions. Three hypotheses are defined.

1. A convolutional neural network can be trained to reach the state-of-the-art accuracy for pre-training of 98.64% without data augmentation or image denoising.
2. An accuracy of 98.64% or better can be achieved on the post-training using the same convolutional neural network and weights from the pre-training.

3. The implementation of the Squeeze-and-Excitation architecture yields a better accuracy than other state-of-the-art networks.

1.4 Contributions

This thesis provides the following contributions:

- We apply a convolutional neural network using the winning technology of the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) 2017 competition, Squeeze-and-Excitation [17], to Nordic fish recognition.
- Our solution achieves the state-of-the-art on the Fish4Knowledge dataset with a testing accuracy of 99.27%.
- The proposed solution achieves an accuracy on the Nordic species dataset of 83.68%.

1.5 Background

Norway is one of Europe's most important fishery nations and the world's second-largest seafood exporter, where 95% of its harvesting and production is exported [1]. One of the reasons is due to Norway's long coastline and fjords, which provide excellent conditions for harvesting from the sea and develop a sustainable maritime industry. Furthermore, Norway has one of the best management systems for fisheries and agriculture combined with its influence from nature and knowledge in these areas. In 2017, Norway exported fish for 94.5 million Norwegian Kroner, which is an increase of 3% from 2016, to over 140 markets all over the world.

IMR continuously researches and monitors fish species and their behavior [19]. According to Alf Ring Kleiven, a researcher at IMR, monitoring of fish is a manual process, and looking through a video feed can take between one to three hours.

IMR is currently enrolled in a project called "Aktiv Forvaltning av Marine Ressurser – Lokalt Tilpasset Forvaltning", which aims to study the effect of

human impact on eco-systems, specifically in conservation zones [10]. The project mainly records video of cod, but other species like coalfish, pollack and common ling have been recorded during the project.

Another project IMR is currently working on, is "Leppefisk Biologi, Bestand og Bestandsstruktur" [40]. The main goal of the project is to map the population of wrasse in different areas to better understand how to maintain a sustainable catch rate of wrasse. Another goal of the project is to map diseases and infection rate of wrasse. The project also includes a sub-project which looks at the mating choices among corkwing. Videos of wrasse and corkwing are recorded for this project, and images captured from this project and [10] are used for the post-training. The data received from IMR are explained in more detail in Section 3.1.2.

1.6 Report Outline

The approaches have now been defined. Chapter 2 provides the theoretical background on CNNs necessary to understand the solution to the approaches implemented in Chapter 3, and the state-of-the-art within CNNs and fish classification. Chapter 4 contains the experiments used to verify the solution against the goals defined in this chapter. Chapter 5 reviews the results achieved with regards to the approaches and provides an answer to the research questions and hypotheses. Potential further work to the solution is also discussed.

Chapter 2

Theoretical Background

This chapter provides history and technical background information within the field of CNNs and image classification, as well as state-of-the-art within CNNs and fish classification. The background leading to the state-of-the-art within CNNs is reviewed in Section 2.1. Section 2.2 provides the state-of-the-art within fish classification, and some of the more recent CNNs together with popular techniques used in these CNNs are reviewed.

2.1 Convolutional Neural Networks

The first convolutional neural network as known today was the LeNet-5 proposed by LeCun, et.al. [27] in 1998, and was used to recognize documents. The solution was based on several classification problems solved by LeCun and others in 1989 and 1992 [31][30][29]. The articles proposed solutions for classifying single digits, strings of digits and zip codes, respectively. Those solutions did not use a CNN, but important features of CNNs like backpropagation and softmax were used.

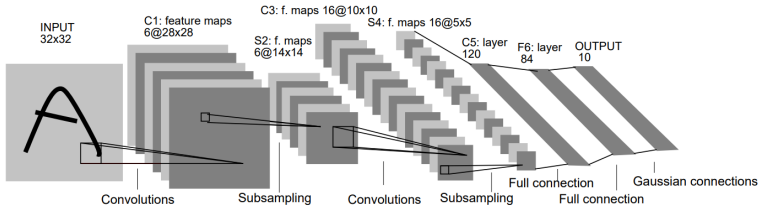

Figure 2.1: Architecture of the LeNet-5 network [27].

As opposed to fully-connected neural networks that connect every neuron in the previous layer with every neuron in the next layer, CNNs only consider the closest neurons in a given range. This is achieved by extracting the pixel information from images in specified block sizes, say 5×5, which greatly reduces the number of connections to each neuron. By grouping the neurons in each hidden layer in a block and use the same weights for each block, the number of connections can be even further reduced. Grouping the weights together is called weight-sharing [2]. We can use Aghdam and Heravi's example from "Guide to Convolutional Neural Networks" to show the reduction in connections when applying weight sharing to a 16×16 pixels grayscale image [2, pp. 86-88]. The number of inputs is, in this case, 16×16 = 1024 neurons, and the neural network has a hidden layer with 7200 neurons. This gives a total of 1024×7200 = 7372800 connections. The pixels in the image are grouped together in a 5×5 grid, and the neurons in the hidden layer are grouped into 50 12×12 blocks. When applying weight-sharing to the blocks in the hidden layer, we get a total of (5×5)×50 = 1250 connections, which is a 99.98% reduction of connections.

The techniques for reducing the number of connections is how CNNs work. The 5×5 blocks the input image in the above example are grouped into, are called filters [2]. The regions of the input image these filters are applied to are called receptive fields. The process of applying a filter to a receptive field is called convolving. The output of convolving a filter is called a feature map. The process is illustrated in Figure 2.2. It is possible to specify the number of units a filter can slide over an image during convolving. This is called striding and can be specified in both x- and y-direction of the matrix. By default, a stride of one is used. This causes some overlap between strides, but allows for more detailed feature maps. Sometimes when striding over an image, the border pixels may be left out as result of selection of filter-size. The striding does not add up and potential important features may

be excluded. This can be resolved by the use of zero-padding, where zeros will be added around the image shown in Figure 2.3.

It can be seen from the LeNet-5 example in Figure 2.1 that the output of the first convolving process consists of six feature maps of size 28×28. The amount of feature maps is manually defined. The matrix in the first convolutional layer will therefore be 6-dimensional. The input can also have multiple dimensions, the most common is 3-dimensional. This is due to the fact that most images use the RGB-scale, which has one value each for red, green and blue. The number 3 in the dimension size in Figure 2.3 indicates that it is an RGB image.

The output size of the first convolutional layer is 28×28 because a 5×5 filter has been used, which can be applied a total of 28×28 times over the input image. In the S_2 subsampling layer, a 2×2 filter has been applied with the sigmoid activation function, leading to six 14×14 feature maps.

Filters and activation functions can be applied as many times as desired, and there is no single answer to how many layers should be used.

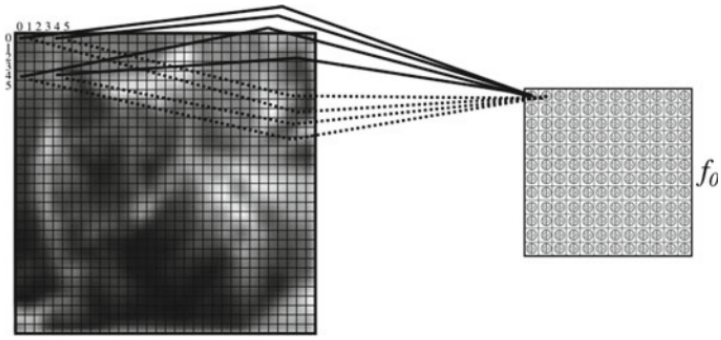

Figure 2.2: Convolving the 5×5 filter on the input image, creating one feature map f_0 [2]. Each time the filter is applied to the receptive field yields one neuron in the feature map.

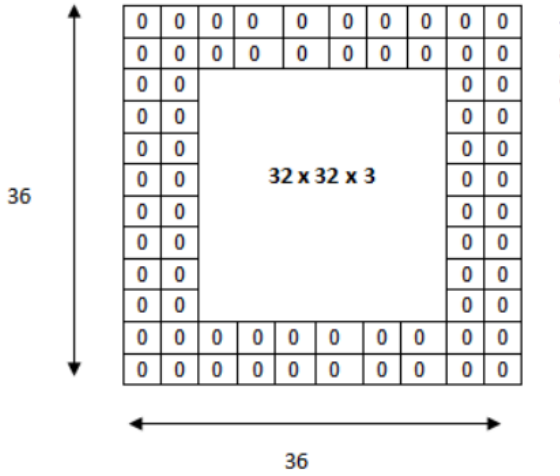

Figure 2.3: Example of zero-padding [9].

After convolving and activation functions and pooling have been applied through the specified amount of layers, the feature maps are merged into fully-connected layers to produce the final output. Most often, more than one fully-connected layer is used. LeNet-5 has two fully-connected layers, where the output of the final layer is the different classes the input can be classified into. In the case of LeNet-5, the final output is 10 because the classification problem solved is single digit recognition. Figure 2.4 shows more clearly the use of fully-connected layers. The matrix is flattened to a 1-dimensional matrix between the last convolutional layer and the first fully-connected layer, as can be seen by the depth of the matrix vanishing. This represents the feature maps being merged.

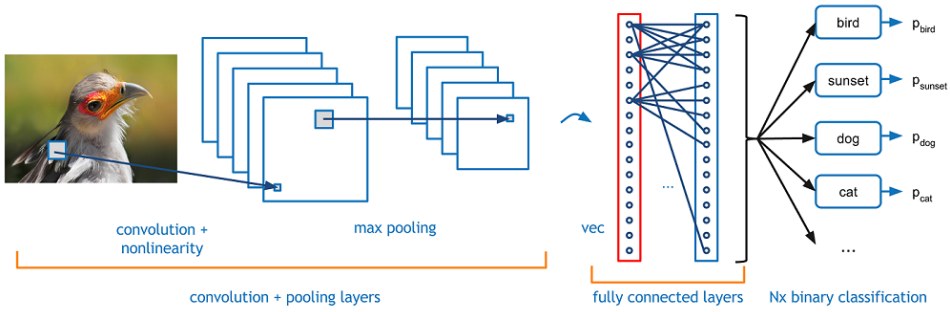

Figure 2.4: Figure showing the entire process of a CNN [8]. The smaller box in the input image is the receptive field.

2.1.1 Activation Functions

Activation functions are used to determine if a neuron should be activated and the output of each neuron in a layer in a neural network [2]. The activation functions should be nonlinear and continuously differentiable. That means that the network can learn nonlinearities in the images and that the function has a derivative for all values in the range of neuron values. The values are often in range 0 to 1, but can also be 0 to 255 if using RGB directly.

Sigmoid used to be the most common choice of activation function. It is differentiable, nonlinear and has a smooth curve as shown in the figure below.

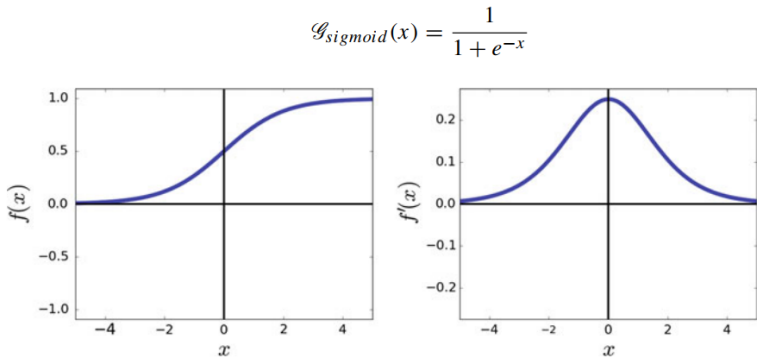

Figure 2.5: The sigmoid activation function and its graphs [2].

However, the sigmoid function suffers from a problem called vanishing gradient. If the neuron's value is not close to the graph's origin, the gradient becomes very small. When this gradient is multiplied with its children, the gradient becomes vanishingly small, which eventually stops the learning process for that neuron.

An often used activation function in CNNs is Rectified Linear Unit (ReLU). Figure 2.6 shows the ReLU activation function. It is a very simple function, which makes all negative neurons equal to zero. ReLU does not suffer from the vanishing gradient problem, because the derivative for all positive numbers is always 1. The function does not approximate the identity function near origin, but will always produce a strong gradient around origin. If a

neuron's weighted value is always below 0, the function will always return 0 for that neuron, causing something called dead neurons. This will stop training for that particular neuron, and potentially cause a slightly lower accuracy for the classification problem.

$$\mathscr{G}_{relu}(x) = \max(0, x)$$

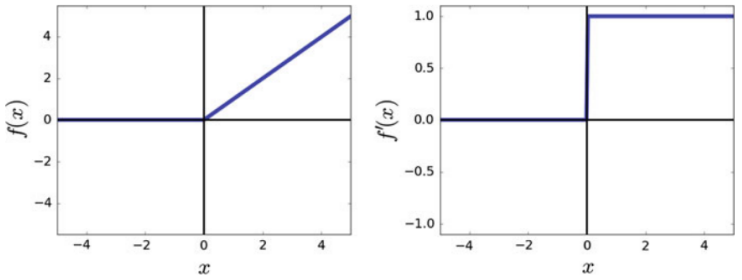

Figure 2.6: The ReLU activation function, its graph and its derivative graph [2].

2.1.2 Pooling

In between the convolving processes after applying ReLU, a technique called pooling can be performed if desired as illustrated in Figure 2.4. After generating n-subsamples from convolving, pooling can be used for what is called downsampling for further feature extraction. Sometimes the dimensionality of the feature maps can become very large. If an image is 190×190 and 50 filters of size 7×7 are applied [2], the output after convolving will result in 50×184×184 = 1,692,800 neurons. This number is clearly very high and becomes even larger. One of the main purposes of pooling is to reduce the dimensions of the generated feature maps. Pooling also uses the same concept as filters and striding as well. To illustrate, a vector $v1$ = [1,3,7,2,9,10,6,12,13,20] and stride = 2 is used as example. This would output v1 = [1,7,9,6,13], which is now half the size.

A well-known version of pooling is max pooling. Figure 2.7 shows an N×N box of size 2 sliding over the feature map with a stride value of 2. Max pooling takes the largest value within each $N \times N$ region for every stride, and this value corresponds to a pixel of the newly generated feature map. In Figure 2.7, the max value of the region is 142. An alternative to max

pooling is average pooling, which works the same way but instead calculates the average of the values within each $N \times N$ region. However, max pooling is the most preferred one due to better results [38]. Another key thing to remember is that pooling is used on every feature map individually and that the dimensions of the last output vector are fairly lower.

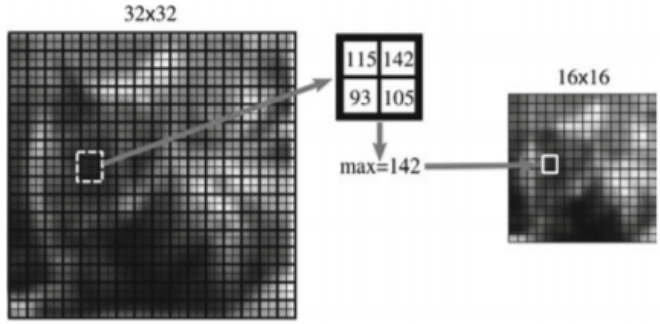

Figure 2.7: Example of max-pooling with stride = 2 and size = 2 [2].

2.1.3 Dropout

Dropout is a technique often used in CNNs to prevent overfitting by randomly dropping some units from the network temporarily during training [41]. As can be seen in Figure 2.8, all incoming and outgoing connections are also removed with the neuron. Overfitting is a problem most common when training with small datasets, because the network trains on potential noise or patterns in the dataset that may not exist in the test data. Another advantage of dropout is that it simulates multiple neural networks because different neurons are dropped in each iteration. In CNNs, dropout is only applied in the fully-connected layers, to prevent loss of features in the feature maps.

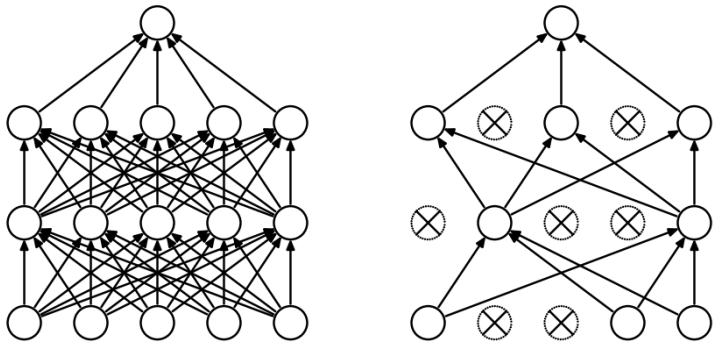

Figure 2.8: Illustration of dropout [41]. On the left: neural net without dropout. On the right: neural net with dropout.

2.1.4 Classification Functions

There are several classification functions that can be used, but the most common in CNNs and multiclass classification problems is softmax. Sigmoid can be used as a classification function in binary classification problems, but the problems in this thesis are multiclass, and sigmoid as a classification function will therefore not be explained in further detail.

The softmax function is used in the last layer of a CNN and merges the output of the previous layer into a vector consisting of an equal amount of values as the number of classes in the classification problem [5]. All values are between 0 and 1, and the sum of all values is equal to 1. The values represent probabilities for each class, where the result of the classification will be the class with the highest probability.

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \; for \; j = 1, \ldots, K \tag{2.1}$$

The z in the softmax function above is the input vector the probabilities will be calculated from, and K is the number of output classes.

Another well-known classification function is Support Vector Machine (SVM) [6], which can behave as a linear- or nonlinear classifier. Generally, SVM is trying to compute a hyperplane where the intention is to maximize the

margin between the vectors of the two classes. These vectors that are closest to the hyperplane, are called support vectors and determine the margin. In cases when the data points are non-linear and inseparable, SVM provides Kernel Tricks [21] which maps the data points into a high dimensional space so that a linear classification can be performed.

SVM is originally designed for binary classification problems. It is possible to extend SVM to multiclass classification problems by either combining multiple binary classifiers or combine all data in one optimization formulation [12]. This is generally more computationally expensive than considering binary classification problems, as the number of variables increases to the number of output classes.

2.1.5 Loss Functions

The purpose of loss functions is to calculate the difference between predicted labels in the final output and the actual label corresponding to the input image [46]. There are several options for choosing loss function, but as stated by [22], loss functions are highly underrepresented in research. The majority of solutions use logarithmic loss or cross-entropy loss. Other options for loss functions are hinge loss, Tanimoto loss, Cauchy-Schwarz Divergence and expectation loss. Other alternatives exist, but they focus more on linear models.

$$C = -\frac{1}{n} \sum_{x} [y \ln a + (1 - y) \ln(1 - a)] \tag{2.2}$$

The cross-entropy loss function is shown in Equation (2.2) [34]. n is the number of images in the batches, and the sum goes over all training inputs. y is the desired output. The cross-entropy loss function operates with positive values, and results close to 0 indicate a result close to the desired output.

In the next section, the current state of research within CNNs will be explained with focus on image classification of fish.

2.2 State-of-the-Art

Image classification is becoming an increasingly important part of research, industries and also commercial solutions. A well-known benchmark test for large scale image classification is the ILSVRC competition [37]. The ILSVRC uses 1.2 million images from 1000 categories from the ImageNet database [7]. CNNs are a newer type of neural networks, and became a popular algorithm for solving image classification after AlexNet won ILSVRC 2012 [16]. Since then, countless numbers of CNNs have been created, and ILSVRC is a common benchmark test for these networks. CNNs have been applied to automatically perform many tasks, such as document recognition [39] for recognizing numbers and letters, face detection [33][44] and object recognition [36].

The area of interest in this thesis is fish related to the Norwegian fish industry, as it is a very large industry and automatic classification in this market has not yet been fully researched. Classification of fish in marine research is still performed manually. A study performed by Aursand, Mabon and Martin [3], uses a combination of fatty acids, different isotope ratios and molar fractions of isotopomeric deuterium clusters in wild and farmed salmons' muscle tissue and fish oil. The numbers are used to classify between wild and farmed salmon and whether it was a Norwegian, Scottish or Atlantic fish. The experiment achieves 100% accuracy when combining all the factors, but does not work in a live scenario, because muscle tissue needs to be extracted.

Another more recent study [4] uses a C nuclear magnetic resonance (C NMR) machine in combination with both a probabilistic neural network (PNN) and an SVM. The C NMR machine is used to determine molecular structures and individual compounds, and examine kinetics of specific reactions using magnetism [32]. The experiments achieve accuracies of 98.5% with PNN and 100% with SVM when recognizing species. The fish used originate from Norway, Scotland, Canada, Iceland, Ireland, the Faroe Islands and Tasmania, and the classification of origins achieves 82.2% accuracy with PNN and 99.3% with SVM. This experiment use muscle tissue from the fish as well, which does not conform well with a live fish ecosystem.

There has been an increase in solutions within artificial intelligence to classify different types of fish [28][35][23]. All of these use the dataset from

the Fish4Knowledge project [14], consisting of 27370 images of exotic fish categorized into 23 species.

A CNN called Fast R-CNN is one of the earlier models tested on the Fish4Knowledge dataset [28]. The experiments performed in the article use a subset of Fish4Knowledge, limiting the number of classes to those with more than 600 images. Object detection is used to extract only the fish from the images. The approach taken starts by pre-training an AlexNet [26], which is illustrated in Figure 2.11, on the ImageNet database. Then the AlexNet is modified to train on the subset of Fish4Knowledge and apply the weights to Fast R-CNN. The Fast R-CNN takes the pre-trained weights and region proposals made by AlexNet as input, and achieves a mean average precision of 81.4%.

A solution proposed by Leilei and Liang [23], pre-trains a CNN similar to AlexNet with five convolutional layers and three fully-connected layers, using 1000 images from 1000 categories from the ImageNet dataset, and applies the weights to the Fish4Knowledge dataset, using only 50 images per category and 10 categories from the Fish4Knowledge dataset during training. The images from the Fish4Knowledge dataset are pre-processed using image de-noising. The accuracy achieved on the 1420 test images is 85.08% using very small amounts of data.

The highest achieved accuracy on this dataset is 98.64% using a CNN with an SVM classifier function after applying filters to the original images to extract the shape of the fish and remove the background [35]. This network is called DeepFish, and is shown in Figure 2.9. DeepFish consists of three standard convolutional layers and three fully-connected layers, and does not utilize any of the state-of-the-art networks described in Section 2.2.1.

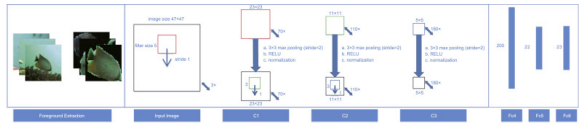

Figure 2.9: The DeepFish architecture [35].

A commercial product somewhat similar to what this thesis aims to solve does exist. The system is called CatchMeter [45] and consists of a conveyor belt with a camera that photographs the fish using a light box to detect and classify fish, as well as provide a length estimate. The fish are fed through

the conveyor belt and photographed, and the fish are recognized by utilizing a threshold for detecting the outline of fish in the images. If one pixel is outside the threshold, a 10×10 region is further analyzed. If 40% of these pixels are outside the threshold, the process is repeated two centimeters further along the image. This process is shown in Figure 2.10. If that process passes, a fish is assumed to be present and the rest of the image is scanned for an outline of the fish. If no fish is found, a new image is scanned for the end of the fish and merged together if a fish is found. The classification accuracy is 98.8%. The length measurements deviate with around 1-2 mm. The accuracy achieved is higher than what [35] achieved, but does not utilize artificial intelligence. The fish are photographed in a relatively structured environment, as opposed to the images in the Fish4Knowledge dataset.

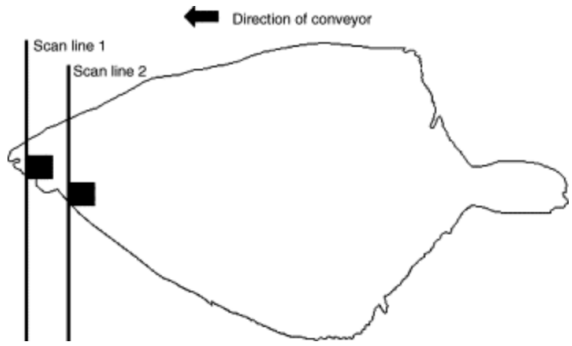

Figure 2.10: The process of detecting a fish with CatchMeter [45].

The following subsection explains some algorithms leading to the state-of-the-art within CNNs today, including Squeeze-and-Excitation. A brief review of common techniques and algorithm principles included in state-of-the-art CNNs is included towards the end of this chapter.

2.2.1 Convolutional Neural Networks

The next paragraphs review the state-of-the-art within CNNs and some of the CNNs leading to the state-of-the-art. The last paragraphs explain some common techniques used in state-of-the-art CNNs.

AlexNet

One of the more well-known CNNs is AlexNet [26]. AlexNet outperformed the winning network from ILSVRC 2010 [15] and was the winner of ILSVRC 2012 [16].

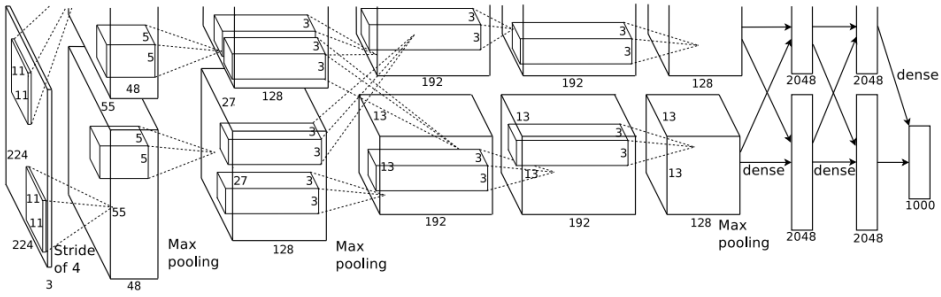

Figure 2.11: The AlexNet architecture [26].

When comparing AlexNet in Figure 2.11 with LeNet-5 from Figure 2.1, it is clear that AlexNet is far more complex than LeNet-5. The ReLU activation function was not used in LeNet-5, as well as dropout, and the ImageNet classification problem has 1000 categories compared to the 10 categories LeNet-5 was designed to solve.

GoogLeNet

Other more recent CNNs are VGGNet and GoogLeNet. They both competed in ILSVRC 2014 where GoogLeNet was the winner [46]. GoogLeNet introduced inception modules, which run several convolving processes in parallel and merges the results in the next layer [43]. This leads to faster computation time and better extraction of higher-level features. Inception-V4 is GoogLeNet's latest version, as of May 2018.

ResNet

ResNet, proposed by He, et.al. [11], was the winner of ILSVRC 2015. It uses a technique called deep residual learning, where the desired mapping of a layer is a combination of the identity mapping and weighted mappings as shown in Figure 2.12. The network is designed to prevent the vanishing gradient problem and degradation of the network, which is when a network's accuracy gets lower after it has converged in a deep neural network. ResNet uses batch normalization without dropout. The argument for this is that the original paper on batch normalization [20] states that batch normalization regularizes the model and removes the need for dropout.

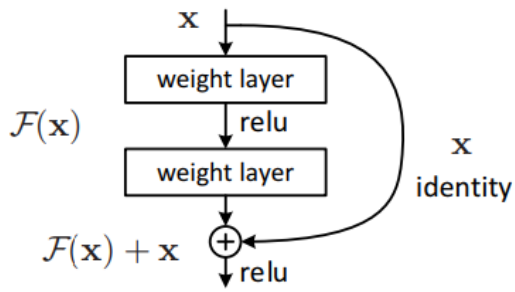

Figure 2.12: The design principle of ResNet [11].

Inception-ResNet is an extension of Google's Inception net and ResNet, and achieved great results when testing on the ILSVRC 2012 dataset [42]. Two solutions were proposed, where Inception-ResNet-v2 had the best performance. Figure 2.13 below shows one of the Inception-ResNet-v2 modules.

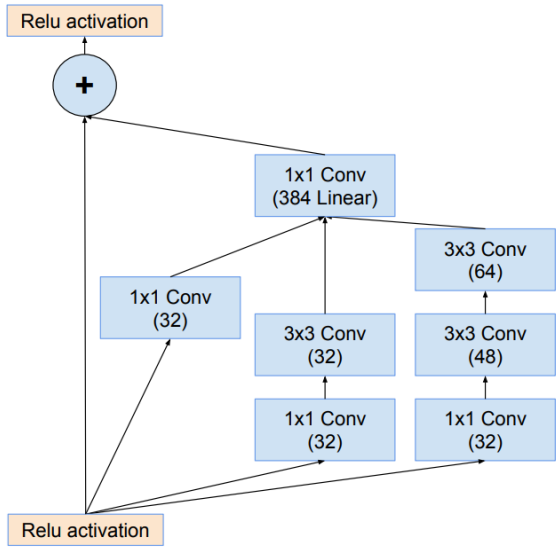

Figure 2.13: Inception-ResNet-v2 module A [42].

Squeeze-and-Excitation

Inception-ResNet was a common foundation for many of the contributions to the ILSVRC 2017 competition [17]. The winner of that competition was one of the Squeeze-and-Excitation networks (SENets) [13] and can be considered state-of-the-art within image classification, at least on large datasets with many output classes. Several versions of SENet were developed and tested by applying Squeeze-and-Excitation on several existing networks, including Inception-Resnet-v2, ResNet-50, ResNet-152 and ResNeXt-50. The squeeze operation flattens the input and applies global average pooling on each channel in the input to prevent the network from becoming channel-dependent and unable to exploit contextual information outside of the receptive fields. The output of the squeeze operation is then fed to a fully-connected layer, followed by ReLU activation and another fully-connected layer. Finally, a sigmoid activation function is applied. The squeeze and excitation operations combined are called SE blocks. The SE blocks are combined with the identity branch to form the final output of the layer, as shown in Figure 2.14. SE can be applied to any number of layers and can be applied to any existing neural network.

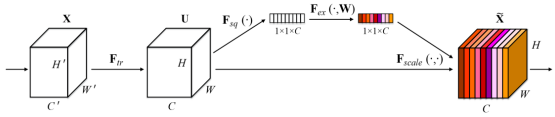

Figure 2.14: A Squeeze-and-Excitation block [13].

The next paragraphs take a further look at the most common techniques and parameters to use in CNNs, including different activation functions popularly used in successful networks.

Activation Functions

Pooling is normal to include in CNNs. GoogLeNet uses max pooling between most of its layers as well as average pooling in the last layer [43]. ResNet also uses average pooling in the last layer. The classifier function softmax can be seen as an activation function and is commonly used to return the probabilities in the last fully-connected layer. Both ResNet and GoogLeNet use softmax as the classifier function [11][43]. SENet uses global average pooling in the SE blocks.

Dropout

Dropout is still commonly used in CNNs, including GoogLeNet, which uses a dropout rate of 40%. Dropout is most important to use when dealing with small datasets to prevent overfitting. ResNet utilizes a more modern alternative to dropout called batch normalization [20]. It normalizes mini-batches of SGD using two extra trainable parameters to prevent the activation functions from working in the linear plane. This process is implemented in the architecture itself. The main goal of using batch normalization is to prevent what is called covariate shift, where the nonlinearities saturate, and to drastically increase the training speed.

The background information necessary to understand the solution has now been explained. The proposed solution to the problem is presented in the next chapter. The solution differs from the ones described in Section 2.2 by using a CNN without any use of muscle tissue or machines that have a

physical impact on the fish. No filters are applied to the images used in the solution. The proposed convolutional neural network implements the winning network from ILSVRC 2017, Squeeze-and-Excitation. Additionally, the solution is tested on the new dataset of Nordic fish species. The results are discussed throughout the next chapters.

Chapter 3

Approach and Implementation

The proposed solution to the problem is presented in this chapter. The solution uses a CNN with SE blocks described in Section 2.2. The CNN is inspired by the DeepFish architecture [35]. However, the hyperparameters and the depth of the network have been changed. SE was added to the solution because the network can be implemented in any existing network, and was the winner of the ImageNet classification competition in 2017 [17]. As opposed to some of the deeper networks like Inception-ResNet, the proposed solution is more shallow and therefore has a faster execution time, which is useful if the solution is to be implemented in a live environment. The proposed solution with the SE extension will throughout the rest of this thesis be referred to as CNN-SENet.

All tests leading to the solution is performed on the Fish4Knowledge dataset, before applying the model with pre-trained weights from this dataset to the dataset consisting of Nordic fish species. The approach when training on the Fish4Knowledge dataset is referred to as pre-training for the remainder of this thesis, and the approach when training on the Nordic species dataset with the weights from pre-training is referred to as post-training. The solution is implemented in Keras, which is a high-level API for building neural networks and runs on top of Tensorflow [24].

This chapter starts with an introduction to the datasets used for testing,

followed by the pre-processing done with the datasets. Then CNN-SENet is explained in detail, as well as tests leading to the final solution. The chapter is concluded with a comparison between CNN-SENet and DeepFish.

3.1 Datasets

There are two datasets used in the solution. The first is the Fish4Knowledge dataset [14], which is used in the pre-training process. The dataset used for post-training is the Nordic fish species dataset. A common feature for these two, is that each fish species has their own folder with the corresponding images, and all these folders are under a root folder as illustrated in Figure 3.1.

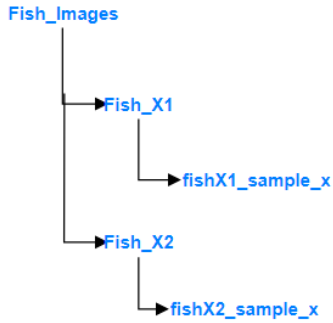

Figure 3.1: File structure of the datasets.

3.1.1 Fish4Knowledge Dataset

The Fish4Knowledge dataset consists of 23 different exotic species. From Figure 3.2, the species and their corresponding distribution are illustrated. The difference between some of the species in terms of quantity, is relatively big and therefore provides an imbalanced collection of data. An example is between *species 1* with their 12112 samples, and *species 23*, holding only 25 samples. In addition, each image usually consists of one individual fish, but there are occasions where the image is crowded with other individuals. Some images per species have similar backgrounds and environment, but

there are also some variations.

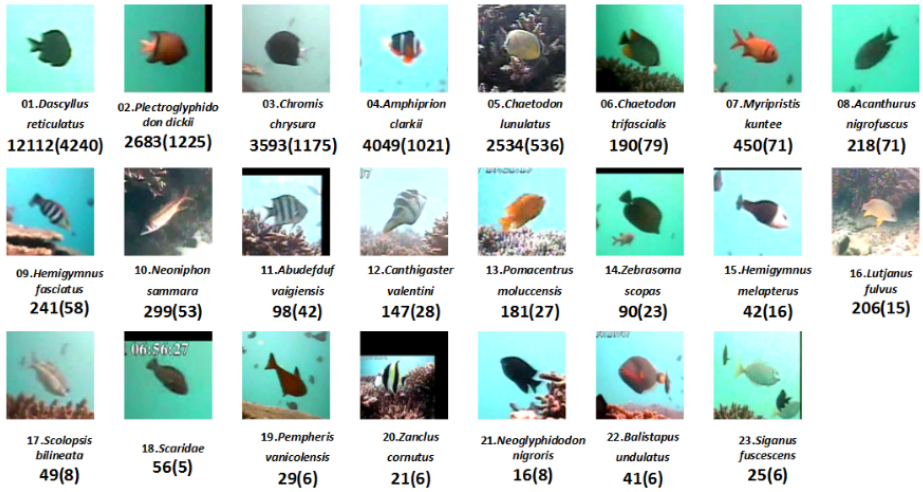

Figure 3.2: Distribution of the Fish4Knowledge dataset [14].

3.1.2 Nordic Fish Species Dataset

The dataset used for post-training is distributed from the Institute of Marine Research (IMR) [18] in Flødevigen (Norway) and consists of 4 typical Nordic fish species with a total of 1019 samples. As mentioned in Chapter 1, the images have been extracted mainly from the underwater video cameras used in the projects [10][40]. Furthermore, the data was extracted this year, which means no one has tested this dataset yet. The distribution is not significantly imbalanced as in the dataset from Fish4Knowledge, but the number of samples for each species is less. Noise in the background is more dominant in this dataset, as Figure 3.4 shows. Some species have images with a similar background, other species have images with a variety of backgrounds. As a result of the low amount of images per species, the ones with the most sample data have been chosen to include in the experiments. The species included are Pollack, Corkwing wrasse male, Corkwing wrasse female combined with a morph of males that mimics females (sneakers), and Coalfish, which is shown below in Figure 3.3.

Figure 3.3: Distribution of the second dataset.

Figure 3.4: Four images of wrasse female and sneakers.

The information necessary to understand the datasets have now been explained, and the next section describes how the datasets are pre-processed before the architecture of CNN-SENet is explained.

3.2 Pre-processing

One of the research questions is whether the state-of-the-art accuracy on the Fish4Knowledge can be achieved without data augmentation. No filters have been applied when training CNN-SENet, as opposed to the previous state-of-the-art solutions. The images are re-sized to 200×200 pixels, because all images need to have the same dimensions for the CNN to work, and some of the CNNs tested in Chapter 4 require the images to be a minimum of 200×200 pixels in order for all filters to be applied to the input images. The images are also re-scaled from RGB values between 0 and 255, to values between 0 and 1 to prevent too much variation in output between the CNN layers.

3.3 CNN-SENet Architecture

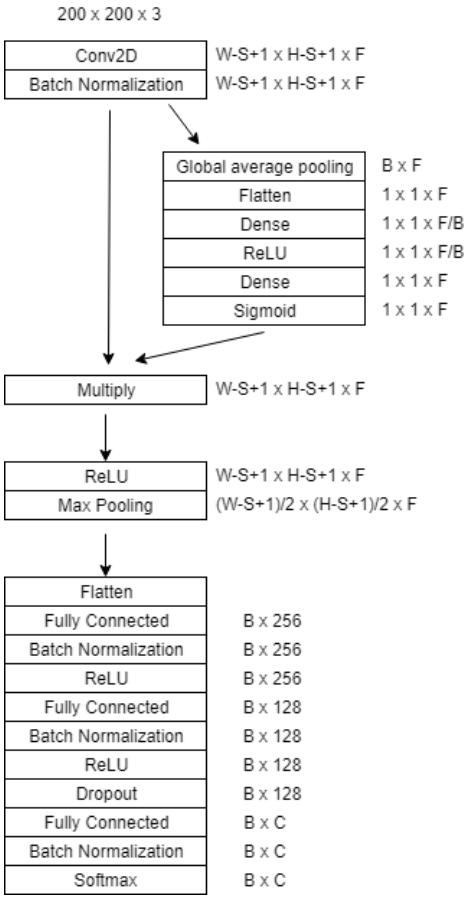

Figure 3.5: The architecture of CNN-SENet.

The architecture of the CNN used to solve the problem is based on the SE architecture described in Section 2.2.1. The architecture of CNN-SENet is shown in Figure 3.5, where W and H are input width and height, respectively. F is the number of filters, B is batch size and S is filter size. The default batch size is 16. The C in the final layers is the number of output classes corresponding to the amount of fish species to be classified. The input to the first convolutional layer is 200×200×3, because the image size is 200×200 pixels and the 3 channels represent the RGB color scheme. The first CNN layer creates 32 filters with 5×5 receptive fields, giving an output dimension of 196×196×32. The output of the first layer is batch normalized before a SE block is applied to the output. The output of the batch normalization and the SE block is merged together by multiplication. ReLU is applied to the multiplied output, which has the same output dimensions as the batch normalized output. Max pooling is applied to the multiplied output with a filter of size 2×2, giving an output of 98×98×32, because the max pooling operation only outputs half of the input width and height. The network consists of five convolutional layers with a SE block between each layer. The first one is described above. The two next layers use 64 filters with 3×3 receptive fields. The fourth layer outputs 128 filters of size 2×2, and the last convolutional layer outputs 256 filters of size 2×2. The stride on all layers is 1.

The network has three fully-connected layers. The output of the final convolutional layer is flattened and fed to a fully-connected layer consisting of 256 neurons. The outputs of all three fully-connected layers are batch normalized. ReLU activation is applied to the first two fully-connected layers after batch normalization. The second fully-connected layer has 128 neurons, and the last fully-connected layer is the final output layer, which contains as many neurons as there are classes in the datasets. The softmax classifier function is used to get the probability distribution for each class per input image. A 50% dropout is applied before the final softmax layer.

The loss function used is categorical cross-entropy with the Adam optimizer [25].

3.3.1 Dropout

A dropout of 50% is applied between the two last fully-connected layers of CNN-SENet to create variation in the dataset during training and to prevent overfitting. Dropout values of 0.1, 0.5 and 0.8 have been tested, and figures 3.6, 3.7 and 3.8 illustrate the training and validation accuracies per epoch from the pre-training with these dropout values, respectively. While 10% and 50% dropout have very similar graphs, 80% dropout converges at a lower training accuracy than the other two. The validation accuracy with 80% dropout remains high, but is still slightly lower than 10% and 50%. The higher the dropout, the more information is lost during training because forward- and backpropagation is done only on the remaining neurons after dropout is applied [41].

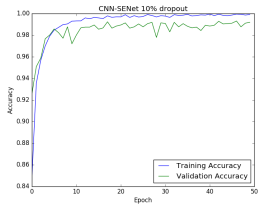

Figure 3.6: 10% Dropout.

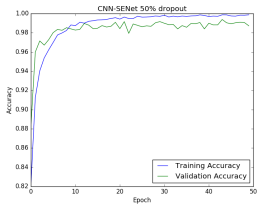

Figure 3.7: 50% Dropout.

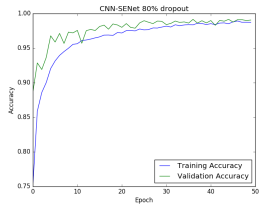

Figure 3.8: 80% Dropout.

Dropout values of 10% and 50% give an identical accuracy on the testing set

of 99.27%. The losses on the testing set for both 10% and 50% are almost identical, with 50% dropout having a loss of 0.03567 and 10% dropout a loss of 0.03846. 80% dropout gives a testing accuracy of 99.03%. These factors led to 50% dropout being the one implemented in the final solution.

Applying dropout after each fully connected layer similar to what DeepFish [35] does has also been tested, but results in a slightly lower testing accuracy than using one dropout activation in CNN-SENet.

3.3.2 Batch Normalization

Batch normalization has been tested and results in a slightly better performance than not applying it. It is said to increase the performance due to the normalization of each previous output layer. Batch normalization is applied in addition to the 50% dropout, although both techniques regularize the model. However, batch normalization aims to reduce the covariate shift and increase performance, so the two techniques work well together. The figures 3.9 and 3.10 below illustrate the pre-training process without and with batch normalization, respectively. The training accuracy is very similar in both cases, but slightly more stable when using batch normalization, and the validation accuracy is higher and more stable. This is due to the means and variances of the layer inputs being normalized, causing less variation in the data [20].

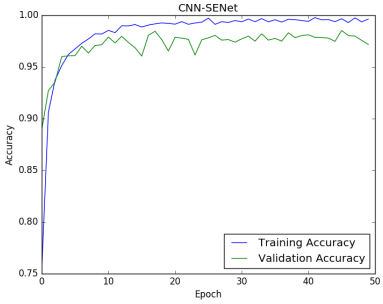

Figure 3.9: 0.5 dropout without batch normalization

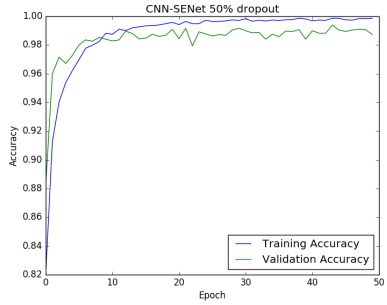

Figure 3.10: 0.5 Dropout with batch normalization

The accuracy on the testing set without batch normalization is 98.35%, while the accuracy with batch normalization is 99.27%.

3.3.3 Learning Rate

The learning rates when using the Adam optimizer can be tuned to further optimize the network. Adam has four tunable parameters: learning rate, learning rate decay and two exponential decay rates for moment estimates. The experiments performed have only focused on the learning rate and decay. The default values for the four parameters in the original paper on Adam proposed by Kingma and Ba [25] uses a learning rate of 0.001, decay rates of 0.9 and 0.999 and no learning rate decay. The experiments in the figures below use the default exponential decay values. Figure 3.11 shows CNN-SENet with a learning rate of 0.1 with a decay of 0.001, and the experiment from Figure 3.12 uses a learning rate of 0.01 and a decay of 0.0001. The decay was added in these experiments because the learning rates are 100 and 10 times higher than the default learning rate, respectively, which can cause very large variations in the weights during training. The training and validation accuracy from Figure 3.13 comes from training with the default values proposed in [25]. Even though a low learning rate is used, the training stabilizes after about 30 epochs with a training accuracy close to 100%.

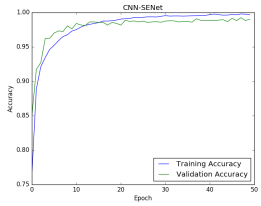

Figure 3.11: Learning rate of 0.1 with a decay of 0.001.

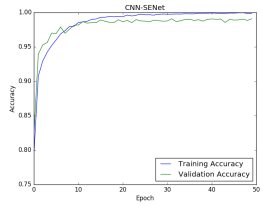

Figure 3.12: Learning rate of 0.01 with a decay of 0.0001.

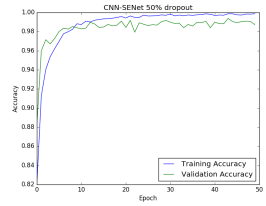

Figure 3.13: Learning rate of 0.001 without decay.

When comparing the training and validation accuracies in the figures above, there are no significant differences in the results, even though the experiment on the left has a learning rate 100 times larger than the one on the right. The highest learning rate seems to have the slowest learning curve. This may be because the learning rate is too high and the adjustment of the weights become too large to approximate the optimal weights in the beginning. After about 40 epochs, the learning rate has decayed enough for the training process to converge. With a learning rate of 0.001, as seen in Figure 3.12, the graph for the training accuracy converges earlier and has a higher y-

value. When evaluating the two models on the testing set, 0.1 learning rate gets an accuracy of 99.15%, while 0.001 learning rate achieves 98.98% accuracy. By using the default values seen in Figure 3.13, the accuracy on the testing set is 99.27%. These values are the ones that have been used when testing different dropout values and batch normalization. The model trains faster and has a higher validation accuracy than the other two, which makes it the final model used in the pre-training process.

The fine-tuning of the learning rate concludes the architecture of CNN-SENet, and the next section discusses the differences between CNN-SENet and the previous state-of-the-art solution on the Fish4Knowledge dataset, DeepFish.

3.4 CNN-SENet vs. DeepFish

The architecture of CNN-SENet has now been discussed. CNN-SENet is based on DeepFish, and the table below illustrates the significant differences between CNN-SENet and DeepFish.

	CNN-SENet	**DeepFish**
Image Size	200×200	47×47
Testing Samples	4126	3098
Network Architecture	Basic with SE blocks	Basic
Classifier	Softmax	SVM
Convolutional Layers	5	3

Table 3.1: Differences between CNN-SENet and DeepFish.

The biggest difference between CNN-SENet and DeepFish is the network architecture. CNN-SENet has five convolutional layers with more filters and SE blocks, while DeepFish only uses three convolutional layers. The image size used during testing in this thesis is larger, which can lead to more details being captured. However, the details might be stretched slightly with larger image sizes. DeepFish trains their solution over 65000 iterations, while CNN-SENet trains for 59800 iterations. As seen in Figure 3.13, the network has already stabilized after the 50 epochs CNN-SENet is trained for, which means that there should not be any gains in accuracy by training for 65000 iterations instead.

Even though DeepFish trains longer and uses an image size more similar to the actual dataset, the CNN-SENet achieves a better testing result.

The approach and implementation of CNN-SENet have been discussed in this chapter. The next chapter shows experiments on the pre-training and post-training approaches with some of the other state-of-the-art CNN architectures. The results with these networks are compared to the results of CNN-SENet.

Chapter 4

Experiments and Numerical Results

This chapter will compare and discuss CNN-SENet and its results against other state-of-the-art solutions presented in Section 2.2. The networks chosen for comparison are Inception-V3, ResNet-50 and Inception-ResNet-V2, as well as CNN-SENet without the Squeeze-and-Excitation blocks.

Section 4.1 describes the configurations made prior to the experiments on the two approaches, pre-training and post-training, which has been defined in Chapter 3. A third experiment using image augmentation on post-training is added, because the post-training dataset is small and image augmentation adds size and variation to the dataset. The different augmentations added are explained in the next section.

Finally, the numerical results along with graphs showing the accuracies are discussed for all three experiments in Section 4.2.

4.1 Configurations

There are some similarities in the configurations of the experiments mentioned above. The datasets in both approaches are divided into 70% training data, 15% validation data and 15% testing data. All networks train for

50 epochs and use an image size of 200×200 pixels, except for Inception-ResNet-V2, which uses an image size of 299×299 pixels. The final fully-connected layer is altered from 23 to 4 neurons in all CNNs between pre-training and post-training to account for the different number of species in the two datasets.

The subsections below describe configurations specifically for the given experiments.

4.1.1 Pre-training

The specific configurations for the pre-training experiments are described in this subsection.

- 19149 images for training, 4126 images for testing.
- One run of 50 epochs.
- Batch size of 16
- Evaluated on the testing set using weights from the epoch with the highest validation accuracy. The weights are therefore not necessarily from the 50^{th} epoch.

4.1.2 Post-training

The specific configurations for the post-training experiments are described in this subsection.

- 712 images for training, 155 images for testing.
- The weights from pre-training are loaded before each run on post-training.
- Ten runs of 50 epochs.
- Batch size of 8
- Evaluated with the weights from the 50^{th} epoch.

On post-training, the model and weights from pre-training are loaded. The reason for using the weights from pre-training, is that the network is able to learn common features of fish before starting the training process, making it easier for the network to learn features on the small dataset used in post-training. To adapt the loaded model to post-training, the last dense layer with 23 neurons is changed to a dense layer with four neurons to get the correct number of output classes.

The images from the Fish4Knowledge dataset have less variation per species than the Nordic species dataset, in addition to having more samples in training, testing and validation. These factors make it viable to draw a conclusion on one run on the pre-training, while the post-training results are based on the average of ten runs.

4.1.3 Post-training with Image Augmentation

The configurations and dataset in the experiment on post-training with image augmentation are the same as the one used in the post-training experiments. The images are rotated randomly from 0 to 40 degrees, vertically and horizontally shifted a random fraction of the image size between 0 and 0.2, shearing transformations are randomly applied, a zooming fraction of 0.2 is randomly applied, and half of the images are flipped horizontally. This is done to create more variation in the dataset, so that the networks can learn features regardless of the position, size or orientation of the features in the images.

4.2 Results

This section will present and discuss the results and experiments obtained from both pre-training and post-training, as well as post-training with image augmentation.

4.2.1 Pre-training

In this subsection, the results for the pre-training approach are discussed. Table 4.1 shows the accuracies achieved when evaluating the networks on the testing subset of the dataset, as well as the time it takes for one training epoch to finish for each network.

Network	Testing Accuracy	Time One Epoch
Inception-V3	99.18%	923 s
ResNet-50	98.86%	646 s
Inception-ResNet-V2	98.59%	2221 s
CNN-SENet	99.27%	197 s
CNN-SENet without Squeeze-and-Excitation	99.15%	159 s

Table 4.1: Testing accuracy and time per epoch on pre-training.

The graphs below show the training and validation accuracy per epoch for the CNNs tested.

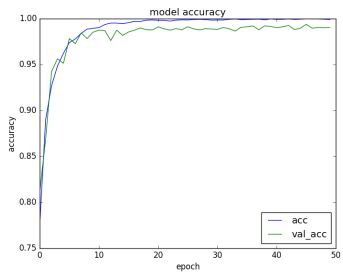

Figure 4.1: Inception-V3.

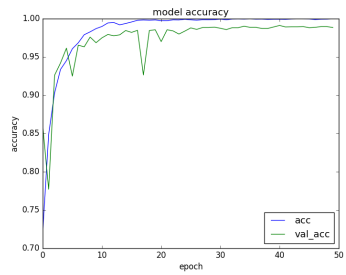

Figure 4.2: ResNet-50.

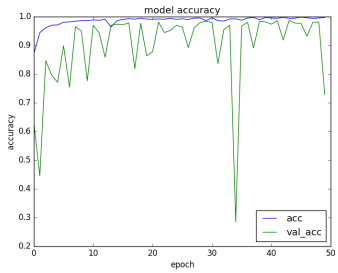

Figure 4.3: Inception-Resnet-V2.

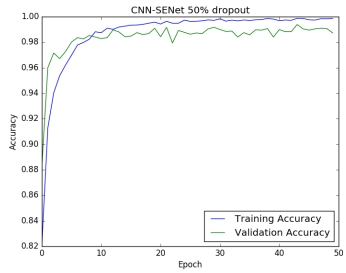

Figure 4.4: CNN-SENet.

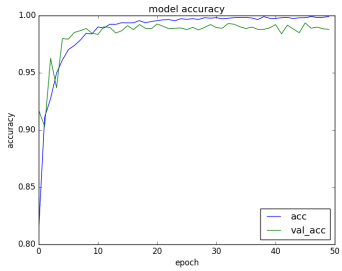

Figure 4.5: CNN-SENet Without Squeeze-and-Excitation.

Table 4.1 shows that most of the CNNs achieve a higher accuracy than the previous state-of-the-art on the testing data. However, the execution time for each epoch varies greatly between the networks. One can see that the bigger and deeper the network is, the longer it takes per epoch. CNN-SENet is approximately ten times faster in terms of execution time than Inception-ResNet-V2, and about three times faster than ResNet-50. When removing the SE blocks from the proposed CNN, the execution time is lower due to the removal of several fully-connected layers. However, when enabling SE blocks, the testing accuracy is higher.

Inception-ResNet-V2 achieves the lowest testing accuracy of the CNNs tested. One of the reasons for this can be due to the image size being 299×299, as opposed to 200×200 pixels in the other networks. The images in the pre-training dataset vary in size but are much smaller than 299×299. The images become more stretched when the image size is this big. Inception-ResNet-V2 has so many layers and so many reductions in dimensions, that

the network causes a negative dimension size when using 200×200 images, which is why 299×299 pixels are chosen. Another reason for Inception-ResNet-V2's lower accuracy might be the depth of the network. Inception-ResNet-V2 was built for the ImageNet database described in Section 2.2.1, which consists of 1000 categories to classify. Both Inception-V3 and ResNet-50 were created for ImageNet, but Inception-ResNet-V2 combines both the Inception and ResNet architecture, leading to a deeper network. The significant difference in time per epoch when comparing Inception-ResNet-V2 to the other solutions is also a result of the increased image size and the depth of the network. The total amount of pixels per image is more than doubled, 40000 pixels in a 200×200 pixel image compared to 89401 with an image size of 299×299.

The graphs in the figures 4.1 and 4.2 show that Inception-V3 and ResNet-50 have a more stable validation accuracy throughout the training process than the other networks. ResNet-50 fluctuates a little until epoch 20, before it stays stable throughout the rest. Inception-ResNet-V2 is highly unstable throughout the whole training process, which is not the case for any of the other networks. This indicates that even though Inception-ResNet-V2 is a newer network than both Inception-V3 and ResNet-50, it does not perform as well as its predecessors on the pre-training dataset. CNN-SENet with and without SE fluctuate to some extent in the validation accuracy, as seen in graphs 4.4 and 4.5, but remains high and fairly stable.

CNN-SENet achieves 99.27% testing accuracy with only five convolutional layers and three fully-connected layers, with SE blocks between the convolutional layers. To be able to implement a CNN in a live video feed, it is important that the network is fast. CNN-SENet is the second fastest among the networks tested, only beaten in execution time by CNN-SENet without SE blocks. The addition of the SE blocks leads to an increase of 0.12% compared to not including them.

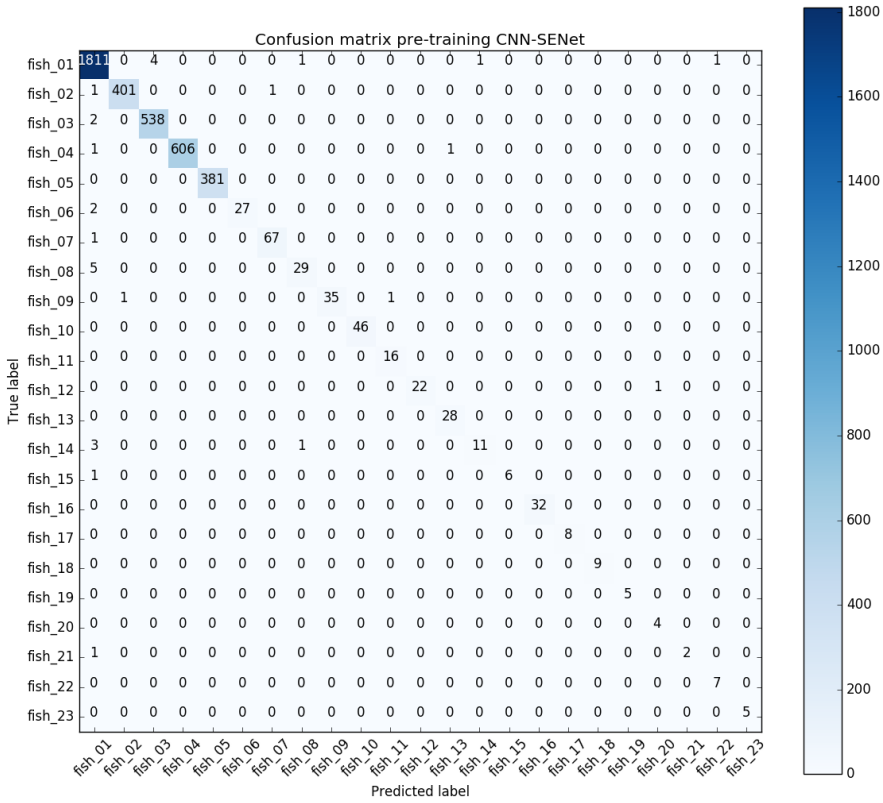

Figure 4.6: Confusion matrix for pre-training with CNN-SENet.

Figure 4.6 shows a confusion matrix of the predictions from pre-training with CNN-SENet. The y-axis represents the actual species, and the x-axis is the predicted species. CNN-SENet mostly predicts the correct species, with some random wrong classifications. Fish 21 stands out with only three images in the testing set, where one of these is classified as fish 1, giving an accuracy on fish 21 of 66.67%. This is due to the uneven distribution of the dataset, meaning that the network may overfit on fish 1 because there are way more images of that species than the others. Most wrong predictions are predicted as fish 1 because of this. Some species like fish 1 and fish 3 are similar, which may be the cause of some predictions on fish 1 being classified as fish 3. Another factor for wrong classifications could be noise in the images.

4.2.2 Post-training

This subsection discusses the results from training with the weights from pre-training on the post-training dataset.

Network	Average Testing Accuracy	Time One Epoch
Inception-V3	85.42%	33 s
ResNet-50	82.39%	47 s
Inception-ResNet-V2	78.84%	91 s
CNN-SENet	83.68%	9 s
CNN-SENet without Squeeze-and-Excitation	82.32%	7 s

Table 4.2: Average testing accuracy over 10 runs and time per epoch on post-training.

Figures 4.8 to 4.12 below contain two graphs per network tested. The graph on the left of each figure is the average training and validation accuracy per epoch during training. The graph on the right is the testing accuracy achieved on each of the ten runs. Figure 4.7 summarizes the right-hand side graphs for all networks as a boxplot. The first column in Table 4.2 is the average of these values.

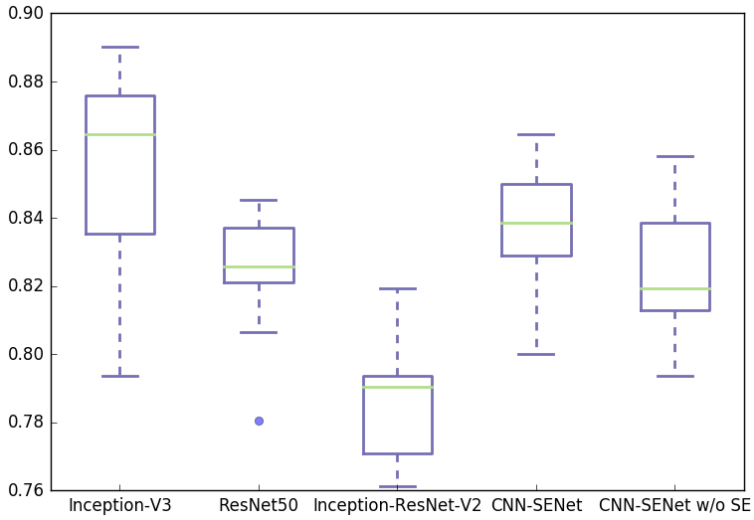

Figure 4.7: Boxplot showing the testing accuracies over ten runs with the different networks.

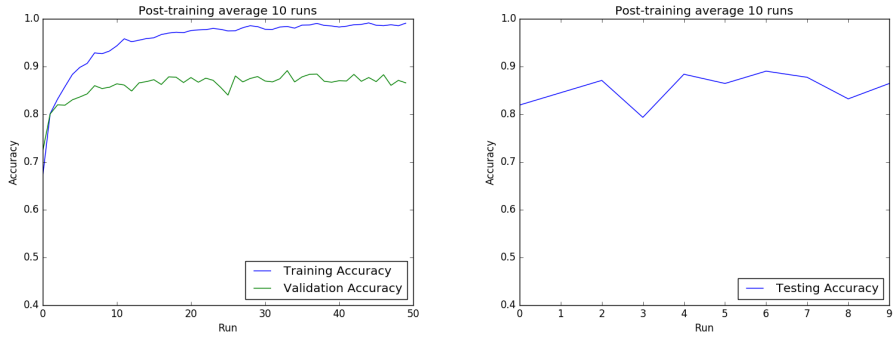

Figure 4.8: Inception-V3. Average training, validation and testing accuracies over 10 runs.

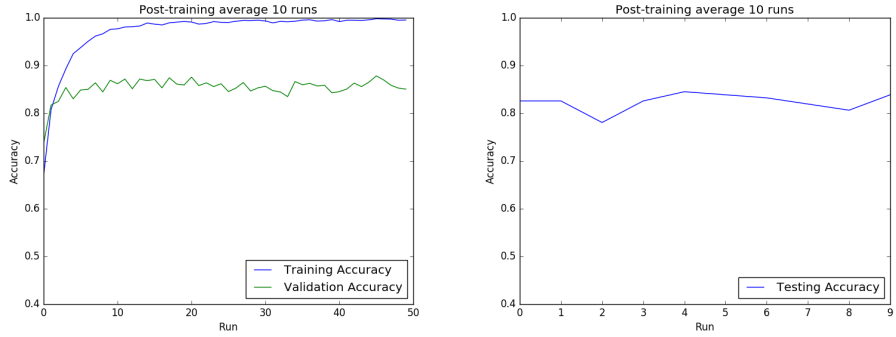

Figure 4.9: ResNet-50. Average training, validation and testing accuracies over 10 runs.

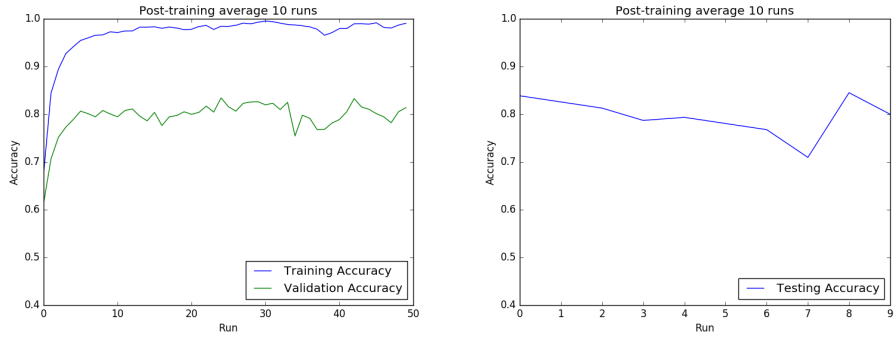

Figure 4.10: Inception-ResNet-V2. Average training, validation and testing accuracies over 10 runs.

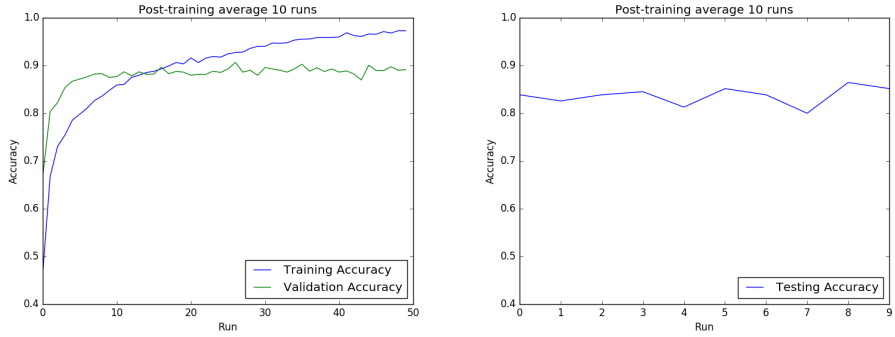

Figure 4.11: CNN-SENet. Average training, validation and testing accuracies over 10 runs.

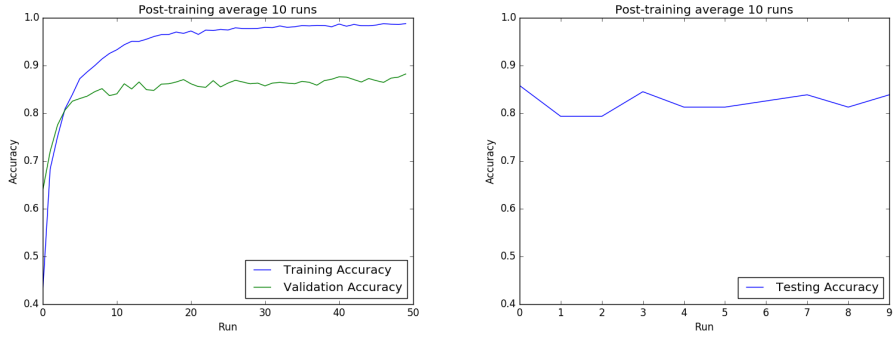

Figure 4.12: CNN-SENet without Squeeze-and-Excitation. Average training, validation and testing accuracies over 10 runs.

The testing accuracies on post-training are generally lower than on pre-training, as can be seen when comparing the results from Table 4.2 and Table 4.1. Inception-ResNet-V2 still has the lowest testing accuracy, but trains and evaluates at a more stable rate than during pre-training. This shows evidence that it is not the size of the dataset that is the leading factor in Inception-Resnet-V2 performing worse than the others during pre-training. The training accuracy of Inception-ResNet-V2 is actually higher than CNN-SENet with and without SE blocks. The deviation in training and validation accuracy measured on Inception-ResNet-V2 could be a sign of overfitting. However, Inception-ResNet-V2 uses a dropout of 80%, which should counter overfitting, while ResNet-50 only uses batch normalization without any dropout. A more likely reason that all networks have a lower

validation and testing accuracy, is the large variation in the images per species, as seen in Section 3.1.2.

CNN-SENet performs better than most of the CNNs tested, only beaten by Inception-V3. When comparing the boxplot of CNN-SENet and Inception-V3 from Figure 4.7, it can be observed that although Inception-V3 has a higher average testing accuracy, the variation in achieved testing accuracy is greater. The lowest achieved accuracy is slightly lower than CNN-SENet's lowest accuracy, but the median shown by the green line is higher for Inception-V3.

The boxplot of ResNet-50 shows low variations in testing accuracy, except for one outlier indicated by the blue dot in the figure. ResNet-50 trains fast and stabilizes at a high training accuracy, but does not manage to achieve a similar validation accuracy, which is the general observation made on all networks from post-training.

Figure 4.11 shows that CNN-SENet has the slowest training progress during post-training, and has still not converged after 50 epochs. The validation accuracy seems to have converged, which means that more epochs would likely not improve the testing accuracy. An interesting observation when looking at the training and validation graph of CNN-SENet, is that the validation accuracy starts higher than the training accuracy. This is probably caused by some similar features being taught during the pre-training process. The training increases much slower than during pre-training, but it increases at a stable rate. CNN-SENet still has the second-best performance when it comes to testing accuracy, and the boxplot indicates more stable results than Inception-V3. The boxplot for CNN-SENet is evenly distributed around the median, while Inception-V3's accuracies are skewed downwards. The greatest advantage with CNN-SENet over Inception-V3 is the time used per epoch. CNN-SENet trains about 3.6 times faster than Inception-V3 on post-training, making it advantageous to use in a live scenario.

When removing the SE blocks, the boxplot is slightly skewed upwards, but generally lower than with the SE blocks, and with a lower median. The training is faster without the SE blocks, but does not fully converge after 50 epochs. The validation accuracy increases somewhat towards the end of training, meaning that the CNN could probably benefit from training further.

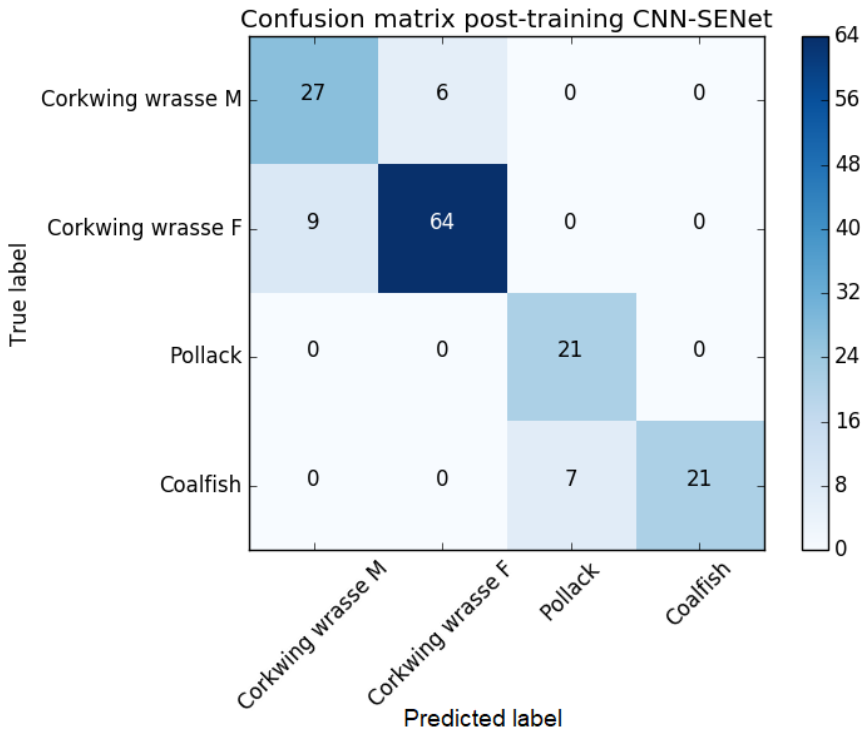

Figure 4.13: Confusion matrix for post-training with CNN-SENet.

Figure 4.13 above shows the confusion matrix for CNN-SENet for post-training. Although the Nordic species dataset has great variations in the data per species, there are similarities between species. Corkwing wrasse female and male has some similarities, which is indicated by the wrong classifications from the confusion matrix. No images of corkwing wrasse are classified as pollack or coalfish, because there are clear differences between those species. Pollack and coalfish also share similarities, and the background and color spectrum in the images for these species are very similar. Many of the images of coalfish have noisy backgrounds and several individuals in the images. Some images are taken at an angle where the fins do not show, making coalfish look very similar to pollack. These may be some of the reasons why some images of coalfish are classified as pollack. Figure 3.3 shows these similarities more clearly.

4.2.3 Post-training with Image Augmentation

The results for post-training with the addition of image augmentation are discussed below. The experiments are the same as in post-training, except the time per epoch is not taken into account in Table 4.3, because there is only a slight difference in the time used per epoch when compared to the post-training results without image augmentation.

Network	Testing Accuracy
Inception-V3	88.45%
ResNet-50	90.20%
Inception-ResNet-V2	82.39%
CNN-SENet	87.74%
CNN-SENet without Squeeze-and-Excitation	83.55%

Table 4.3: Average testing accuracy over 10 runs and time per epoch on post-training with image augmentation.

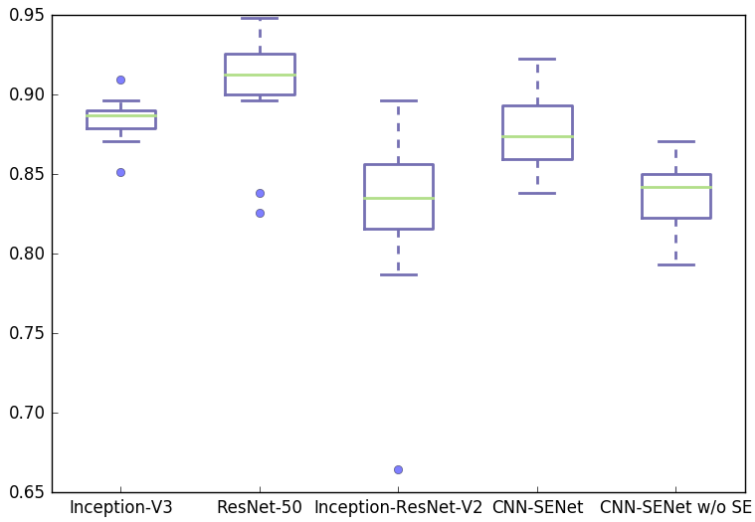

Figure 4.14: Boxplot showing the testing accuracies when using image augmentation over ten runs.

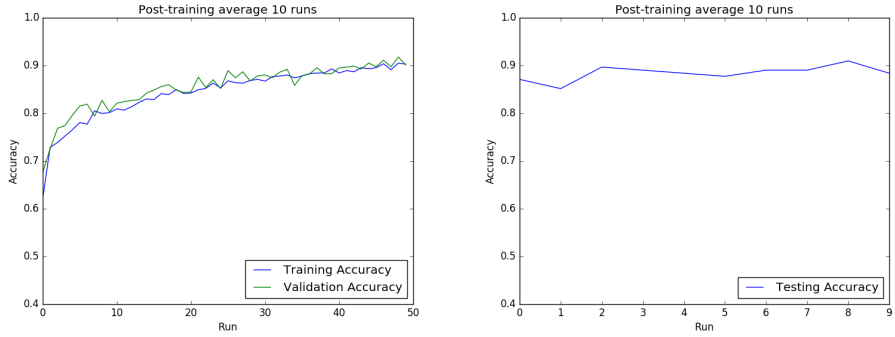

Figure 4.15: Inception-V3 with augmentation. Average training, validation and testing accuracies over 10 runs.

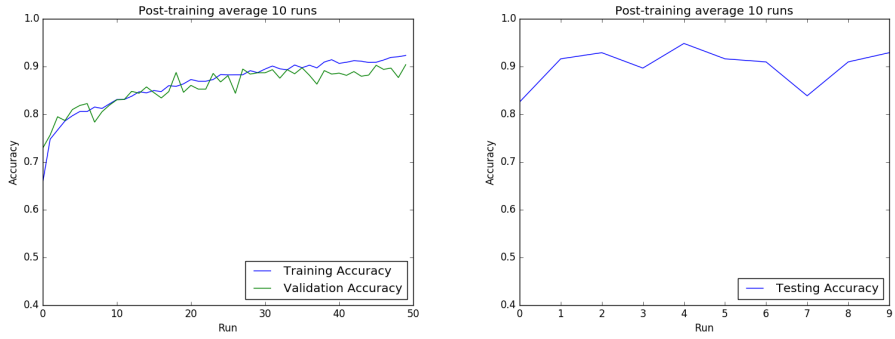

Figure 4.16: ResNet-50 with augmentation. Average training, validation and testing accuracies over 10 runs.

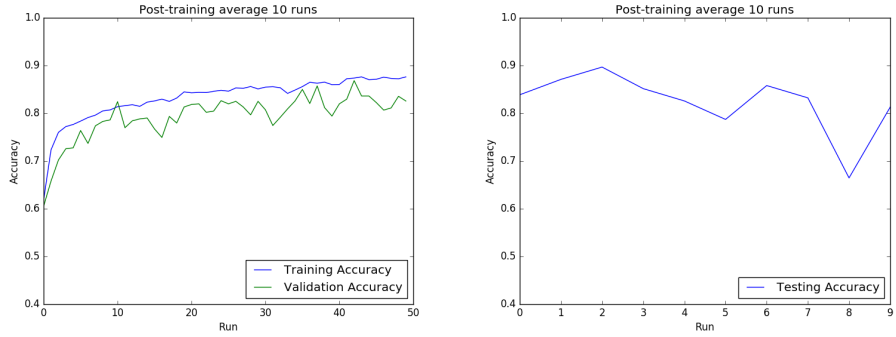

Figure 4.17: Inception-ResNet-V2 with augmentation. Average training, validation and testing accuracies over 10 runs.

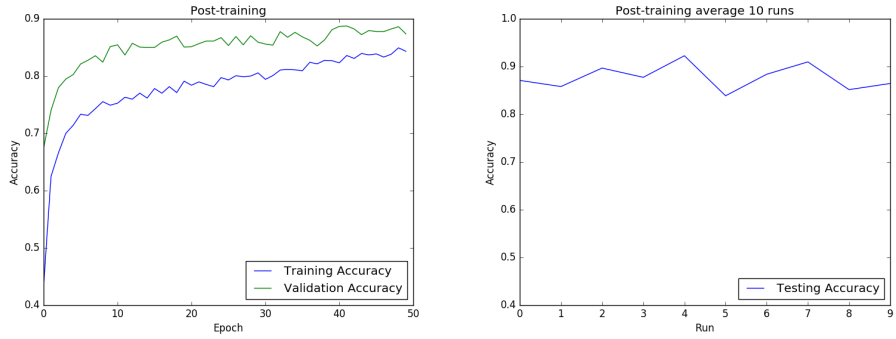

Figure 4.18: CNN-SENet with augmentation. Average training, validation and testing accuracies over 10 runs.

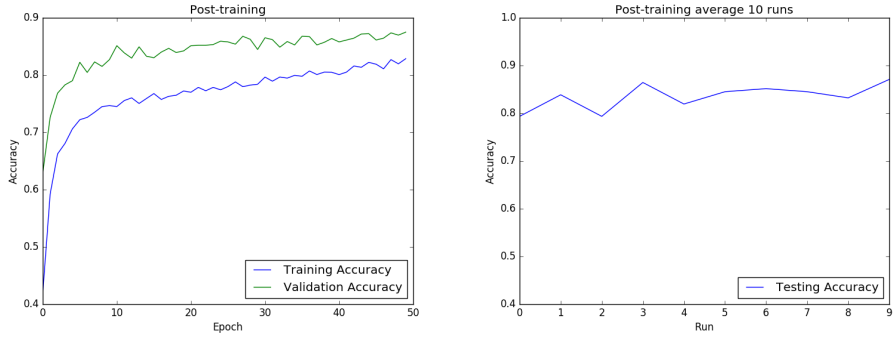

Figure 4.19: CNN-SENet without Squeeze-and-Excitation with augmentation. Average training, validation and testing accuracies over 10 runs.

Table 4.3 illustrates that data augmentation does have a positive effect on the performance, and all networks achieve a higher accuracy. The difference between CNN-SENet and CNN-SENet without SE in terms of testing accuracy is higher now. CNN-SENet SE achieves a 4.06% higher accuracy, while CNN-SENet without SE only increases by 1.23%. Additionally, ResNet-50 now outperforms the other networks with a testing accuracy of 90.20%. Inception-V3 still performs better than CNN-SENet, and Inception-ResNet-V2 still has the lowest average testing accuracy.

The boxplot from Figure 4.14 shows that there is less variation in the accuracies from the different runs for all networks. Inception-V3, ResNet-50, and Inception-ResNet-V2 all have outliers, which indicate abnormalities. The difference in Inception-V3's lowest and highest outlier is at about 6%, meaning Inception-V3 generally has a low spread in accuracy. ResNet-50 has some outliers below 85% with its highest achieved accuracy of 95%, while Inception-ResNet-V2 has one abnormality that largely contributes to a lower total average accuracy than the other networks.

The results from the experiments have now been discussed, and the next chapter will conclude this thesis with a summary of the results and provide an answer to the research questions and hypotheses defined in Chapter 1.

Chapter 5

Conclusion and Further Work

This chapter summarizes the solution and results with regards to the research questions and hypotheses defined in Chapter 1. The thesis will be concluded with potential future work and improvements to the solution, targeted at biometric classification of fish.

5.1 Conclusion

In this thesis, we propose a CNN with the implementation of the SE architecture. The network is specifically tuned and trained for biometric classification of fish. The results show that CNN-SENet achieves the state-of-the-art accuracy of 99.27% on the Fish4Knowledge dataset without any form for data augmentation or image de-noising, beating DeepFish's accuracy of 98.64% [35]. This shows that the SE architecture can be considered state-of-the-art in other problems than ImageNet. With a more evenly distributed dataset, this accuracy could increase, indicated by most wrong classifications in Figure 4.6 being classified as fish 1, which is the species with the most images.

For post-training, CNN-SENet achieves an accuracy of 83.68%. On this dataset, CNN-SENet is not the winning network, with Inception-V3 achiev-

ing an average accuracy of 85.42%, showing evidence that SE is not the best suited to all problems. The best performing CNN on one single run is Inception-V3 with an accuracy of 89.03%. Due to the small size of the dataset and high variation in data, CNN-SENet performs worse on the Nordic species dataset than on Fish4Knowledge. When applying image augmentation on the post-training, we see that the general accuracy increases, showing evidence that it is the limited dataset size that causes the low performance on post-training. With the addition of more images and image augmentation, the accuracy could potentially increase to the same level as pre-training. CNN-SENet is not the highest performing network when applying image augmentation, with ResNet-50 achieving the highest average accuracy of 90.20%.

For both approaches, CNN-SENet with SE blocks has a higher accuracy than without the SE blocks, indicating that SE has a positive effect on accuracy.

5.2 Future Work

The results achieved show that CNN-SENet performs well, and could potentially be used in marine research provided more data for training. This section lists out potential future work for CNN-SENet and other areas within marine research CNN-SENet could be applied to.

- Getting more images of Nordic fish species for CNN-SENet to train on.
- Perform further experiments on the CNN-SENet architecture and parameters to fine-tune the network further.

The list below indicates the areas where IMR sees the potential of deep learning and our solution, as stated by Alf Ring Kleiven.

- Otolith-reading of fish. This is work that takes years for IMR because they have to know the fish's age when making population assessments and advice on quota with regards to management. This is an important area within marine research, but not an area our solution is

directly applicable to. Our solution could be altered to work for this particular problem.

- Individual recognition of fish that the human being's eyes cannot distinguish between. Our solution could be extended to replace the process of recognizing individual fish instead of manually marking individuals.
- Species identification from fish eggs and other marine animals. Today, this is performed manually for identifying species from egg tests.
- Species identification of animal- and plant plankton. This is mostly done manually and our solution could support the work and perform the task more effectively.

References

[1] Norges sjømatråd. https://sjomatnasjonen.seafood.no/. Accessed: 2018-01-31.

[2] Hamed Habibi Aghdam and Elnaz Jahani Heravi. *Guide to Convolutional Neural Networks*. Springer, 2017.

[3] M Aursand, F Mabon, and GJ Martin. Characterization of farmed and wild salmon (salmo salar) by a combined use of compositional and isotopic analyses. *Journal of the American Oil Chemists' Society*, 77(6):659–666, 2000.

[4] Marit Aursand, Inger B Standal, Angelika Praël, Lesley McEvoy, Joe Irvine, and David E Axelson. 13c nmr pattern recognition techniques for the classification of atlantic salmon (salmo salar l.) according to their wild, farmed, and geographical origin. *Journal of agricultural and food chemistry*, 57(9):3444–3451, 2009.

[5] Christopher M Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[6] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[7] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

[8] Adit Deshpande. A beginner's guide to understanding convolutional neural networks. https://adeshpande3.github.io/A-Beginner'

s-Guide-To-Understanding-Convolutional-Neural-Networks/, July 2016. [Online; accessed 10.05.2018].

[9] Adit Deshpande. A beginner's guide to understanding convolutional neural networks part 2. https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks-Part-2/, July 2016. [Online; accessed 10.05.2018].

[10] Sigurd Heiberg Espeland, Alf Ring Kleiven, Even Moland, and Jan Atle Knutsen. Aktiv forvaltning av marine ressurser – lokalt tilpasset forvaltning. Årsrapport 2015. 2015.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[12] Chih-Wei Hsu and Chih-Jen Lin. A comparison of methods for multiclass support vector machines. *IEEE transactions on Neural Networks*, 13(2):415–425, 2002.

[13] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. *arXiv preprint arXiv:1709.01507*, 2017.

[14] Phoenix X. Huang, Bastiaan B. Boom, and Robert B. Fisher. Fish recognition ground-truth data. http://groups.inf.ed.ac.uk/f4k/GROUNDTRUTH/RECOG/, 2013. [Online; accessed 30.01.2018].

[15] ImageNet. Imagenet large scale visual recognition challenge 2010 (ilsvrc2010). http://www.image-net.org/challenges/LSVRC/2010/, 2010. [Online; accessed 22.01.2018].

[16] ImageNet. Imagenet large scale visual recognition challenge 2012 (ilsvrc2012). http://www.image-net.org/challenges/LSVRC/2012/, 2012. [Online; accessed 22.01.2018].

[17] ImageNet. Imagenet large scale visual recognition challenge 2017 (ilsvrc2017). http://image-net.org/challenges/LSVRC/2017/results, 2017. [Online; accessed 14.02.2018].

[18] IMR. Institute of marine research. https://www.hi.no/en, n.a. [Online; accessed 09.03.2018].

[19] IMR. Projects. `https://www.hi.no/forskning/prosjekter/en`, n.a. [Online; accessed 22.03.2018].

[20] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456, 2015.

[21] Vikramaditya Jakkula. Tutorial on support vector machine (svm). *School of EECS, Washington State University*, 37, 2006.

[22] Katarzyna Janocha and Wojciech Marian Czarnecki. On loss functions for deep neural networks in classification. *arXiv preprint arXiv:1702.05659*, 2017.

[23] Leilei Jin and Hong Liang. Deep learning for underwater image recognition in small sample size situations. In *OCEANS 2017-Aberdeen*, pages 1–4. IEEE, 2017.

[24] Keras. Keras: The python deep learning library. `https://keras.io/`, n.a. [Online; accessed 20.02.2018].

[25] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[26] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

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

[28] Xiu Li, Min Shang, Hongwei Qin, and Liansheng Chen. Fast accurate fish detection and recognition of underwater images with fast r-cnn. In *OCEANS'15 MTS/IEEE Washington*, pages 1–5. IEEE, 2015.

[29] Ofer Matan, Henry S. Baird, Jane Bromley, Christopher J. C. Burges, John S. Denker, Lawrence D. Jackel, Yann Le Cun, Edwin P. D. Pednault, William D Satterfield, Charles E. Stenard, et al. Reading handwritten digits: A zip code recognition system. *Computer*, 25(7):59–63, 1992.

[30] Ofer Matan, Christopher JC Burges, Yann LeCun, and John S Denker. Multi-digit recognition using a space displacement neural network. In

Advances in neural information processing systems, pages 488–495, 1992.

[31] Ofer Matan, RK Kiang, CE Stenard, B Boser, JS Denker, D Henderson, RE Howard, W Hubbard, LD Jackel, and Yann Le Cun. Handwritten character recognition using neural network architectures. In *Proceedings of the 4th USPS advanced technology conference*, pages 1003–1011, 1990.

[32] Mark A Nanny, Roger A Minear, and Jerry A Leenheer. *Nuclear magnetic resonance spectroscopy in environmental chemistry*. Oxford University Press, 1997.

[33] Fabian Nasse, Christian Thurau, and Gernot A Fink. Face detection using gpu-based convolutional neural networks. In *International Conference on Computer Analysis of Images and Patterns*, pages 83–90. Springer, 2009.

[34] Michael Nilsen. *Neural Networks and Deep Learning*. Determination Press, 2015.

[35] Hongwei Qin, Xiu Li, Jian Liang, Yigang Peng, and Changshui Zhang. Deepfish: Accurate underwater live fish recognition with a deep architecture. *Neurocomputing*, 187:49–58, 2016.

[36] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 91–99. Curran Associates, Inc., 2015.

[37] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[38] Dominik Scherer, Andreas Müller, and Sven Behnke. Evaluation of pooling operations in convolutional architectures for object recognition. In Konstantinos Diamantaras, Wlodek Duch, and Lazaros S. Iliadis, editors, *Artificial Neural Networks – ICANN 2010*, pages 92–101, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[39] Patrice Y Simard, David Steinkraus, John C Platt, et al. Best practices for convolutional neural networks applied to visual document analysis. In *ICDAR*, volume 3, pages 958–962, 2003.

[40] Anne Berit Skiftesvik et al. Bestander og fangstkvalitet av leppefisk. `https://www.hi.no/filarkiv/2014/02/hi-rapp_3-2014_leppefisk_til_web.pdf/nb-no`, 2014. [Online; accessed 02.04.2018].

[41] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of machine learning research*, 15(1):1929–1958, 2014.

[42] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI*, volume 4, page 12, 2017.

[43] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich, et al. Going deeper with convolutions. Cvpr, 2015.

[44] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. Deepface: Closing the gap to human-level performance in face verification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1701–1708, 2014.

[45] DJ White, C Svellingen, and NJC Strachan. Automated measurement of species and length of fish by computer vision. *Fisheries Research*, 80(2-3):203–210, 2006.

[46] Zizhao Zhang, Fuyong Xing, Hai Su, Xiaoshuang Shi, and Lin Yang. Recent advances in the applications of convolutional neural networks to medical image contour detection. *arXiv preprint arXiv:1708.07281*, 2017.

Appendices

A CNN-SENet Source-code

A.1 cnn_senet.py

```
import h5py, keras
from keras import backend as K
from keras.layers import Conv2D, MaxPooling2D, Activation, Dropout,
Flatten, Dense, BatchNormalization, Input
from keras.models import Model

from se import squeeze_excite_block

channel_axis = 1 if K.image_data_format() == "channels_first" else -1

def se_net(input_shape, preorpost):

  x = Conv2D(32, (5,5))(input_shape)
  x = BatchNormalization(axis=channel_axis)(x)

  x = squeeze_excite_block(x)

  x = Activation('relu')(x)
  x = MaxPooling2D(pool_size=(2,2))(x)

  x = Conv2D(64, (3,3))(x)
  x = BatchNormalization(axis=channel_axis)(x)
```

```
x = squeeze_excite_block(x)

x = Activation('relu')(x)
x = MaxPooling2D(pool_size=(2,2))(x)

x = Conv2D(64, (3,3))(x)
x = BatchNormalization(axis=channel_axis)(x)

x = squeeze_excite_block(x)

x = Activation('relu')(x)
x = MaxPooling2D(pool_size=(2,2))(x)

x = Conv2D(128, (2,2))(x)
x = BatchNormalization(axis=channel_axis)(x)

x = squeeze_excite_block(x)

x = Activation('relu')(x)
x = MaxPooling2D(pool_size=(2,2))(x)

x = Conv2D(256, (2,2))(x)
x = BatchNormalization(axis=channel_axis)(x)

x = squeeze_excite_block(x)

x = Activation('relu')(x)
x = MaxPooling2D(pool_size=(2,2))(x)

x = Flatten()(x)
x = Dense(256)(x)
x = BatchNormalization(axis=channel_axis)(x)
x = Activation('relu')(x)

x = Dense(128)(x)
x = BatchNormalization(axis=channel_axis)(x)
x = Activation('relu')(x)
x = Dropout(0.5)(x)

if preorpost == "pre":
```

```
    x = Dense(23)(x)
  elif preorpost == "post":
    x = Dense(4)(x)
  else:
    print("Invalid input")
  x = BatchNormalization(axis=channel_axis)(x)
  x = Activation('softmax')(x)

  model = Model(input_shape, x)
  return model
```

A.2 se.py

```
from keras.layers import GlobalAveragePooling2D, Reshape, Dense,
multiply, Permute
from keras import backend as K

#https://github.com/titu1994/keras-squeeze-excite-network

def squeeze_excite_block(input, ratio=16):
    ''' Create a squeeze-excite block
    Args:
        input: input tensor
        filters: number of output filters
        k: width factor
    Returns: a keras tensor
    '''
    init = input
    channel_axis = 1 if K.image_data_format() == "channels_first"
    else -1
    filters = init._keras_shape[channel_axis]
    se_shape = (1, 1, filters)

    se = GlobalAveragePooling2D()(init)
    se = Reshape(se_shape)(se)
    se = Dense(filters // ratio, activation='relu',
    kernel_initializer='he_normal', use_bias=False)(se)
    se = Dense(filters, activation='sigmoid',
```

```
kernel_initializer='he_normal', use_bias=False)(se)

if K.image_data_format() == 'channels_first':
    se = Permute((3, 1, 2))(se)

x = multiply([init, se])
return x
```

UiA University of Agder
Master's thesis
Faculty of Engineering and Science
Department of ICT