Vision based facial emotion detection using deep convolutional neural networks

Fredrik Julin
Fjn14001@student.mdh.se

Examiner: Mobyen Uddin Ahmed
Mälardalen University, Västerås, Sweden

Supervisor: Hamidur Rahman
Mälardalen University, Västerås, Sweden

January 16 2019
Abstract

Emotion detection, also known as Facial expression recognition, is the art of mapping an emotion to some sort of input data taken from a human. This is a powerful tool to extract valuable information from individuals which can be used as data for many different purposes, ranging from medical conditions such as depression to customer feedback. To be able to solve the problem of facial expression recognition, smaller subtasks are required and all of them together form the complete system to the problem. Breaking down the bigger task at hand, one can think of these smaller subtasks in the form of a pipeline that implements the necessary steps for classification of some input to then give an output in the form of emotion. In recent time with the rise of the art of computer vision, images are often used as input for these systems and have shown great promise to assist in the task of facial expression recognition as the human face conveys the subjects emotional state and contain more information than other inputs, such as text or audio. Many of the current state-of-the-art systems utilize computer vision in combination with another rising field, namely AI, or more specifically deep learning. These proposed methods for deep learning are in many cases using a special form of neural network called convolutional neural network that specializes in extracting information from images. Then performing classification using the SoftMax function, acting as the last part before the output in the facial expression pipeline. This thesis work has explored these methods of utilizing convolutional neural networks to extract information from images and builds upon it by exploring a set of machine learning algorithms that replace the more commonly used SoftMax function as a classifier, in attempts to further increase not only the accuracy but also optimize the use of computational resources. The work also explores different techniques for the face detection subtask in the pipeline by comparing two approaches. One of these approaches is more frequently used in the state-of-the-art and is said to be more viable for possible real-time applications, namely the Viola-Jones algorithm. The other is a deep learning approach using a state-of-the-art convolutional neural network to perform the detection, in many cases speculated to be too computationally intense to run in real-time. By applying a state-of-the-art inspired new developed convolutional neural network together with the SoftMax classifier, the final performance did not reach state-of-the-art accuracy. However, the machine-learning classifiers used shows promise and bypass the SoftMax function in performance in several cases when given a massively smaller number of samples as training. Furthermore, the results given from implementing and testing a pure deep learning approach, using deep learning algorithms for both the detection and classification stages of the pipeline, shows that deep learning might outperform the classic Viola-Jones algorithm in terms of both detection rate and frames per second.
# Table of Contents

1. Introduction ................................................................................................................. 7  
2. Background .................................................................................................................. 8  
   2.1. Datasets .................................................................................................................. 8  
   2.2. Image preprocessing .............................................................................................. 8  
      2.2.1. Face detection ................................................................................................. 8  
      2.2.2. Feature extraction ......................................................................................... 9  
   2.3. Machine learning ................................................................................................. 9  
      2.3.1. Classification vs regression ........................................................................... 10  
      2.3.2. Train, validation, and test ........................................................................... 10  
      2.3.3. Parameters and hyperparameters ............................................................... 10  
      2.3.4. Underfitting and overfitting ....................................................................... 11  
      2.3.5. ML algorithms ............................................................................................. 11  
   2.4. Deep learning ........................................................................................................ 13  
      2.4.1. Artificial Neural Network ............................................................................ 13  
      2.4.2. Bias ............................................................................................................. 13  
      2.4.3. Batch size .................................................................................................... 14  
      2.4.4. CNN ........................................................................................................... 14  
      2.4.5. Transfer learning ....................................................................................... 18  
3. Related Work .............................................................................................................. 19  
   3.1. ImageNet challenge and state-of-the-art CNN’s .................................................. 19  
   3.2. FER state-of-the-art ......................................................................................... 19  
4. Problem Formulation ................................................................................................. 21  
5. Methods and Materials ............................................................................................... 22  
   5.1. Known limitations ............................................................................................... 22  
   5.2. Data collection ..................................................................................................... 22  
   5.3. Detection and extraction ..................................................................................... 23  
   5.4. CNN’s ............................................................................................................... 24  
   5.5. Machine-learning classification ...................................................................... 26  
      5.5.1. Regression ................................................................................................... 26  
      5.5.2. SVM .......................................................................................................... 27  
      5.5.3. Decision tree and random forest ................................................................. 27  
      5.5.4. KNN ......................................................................................................... 27  
      5.5.5. Naive Bayes ............................................................................................... 28  
   5.6. Real-time classification and detection ................................................................. 28  
6. Results ......................................................................................................................... 29  
   6.1. Face detection ..................................................................................................... 29  
   6.2. CNN ................................................................................................................ 30  
      6.2.1. VGG16 .................................................................................................... 30  
      6.2.2. Final model .............................................................................................. 30  
   6.3. ML classification ............................................................................................... 31  
      6.3.1. Logistic regression ..................................................................................... 31  
      6.3.2. SVM ....................................................................................................... 32
6.3.3. Random forest ensemble.................................................................33
6.3.4. Decision Tree .............................................................................34
6.3.5. KNN .............................................................................................34
6.3.6. Naive Bayes ................................................................................35
6.4. Real-time classification and detection ............................................35

7. Discussion ..........................................................................................38
  7.1. Conclusions ......................................................................................38
  7.1.1. Detection .....................................................................................38
  7.1.2. CNN’s ........................................................................................38
  7.1.3. Classification and ML algorithms ...............................................39
  7.1.4. Summary .....................................................................................39
  7.2. Future work ....................................................................................40

8. References ..........................................................................................41

9. Appendix .............................................................................................45
List of figures

Figure 2.1 – Artificial intelligence subareas ................................................................. 9
Figure 2.2 - Overfitting seen in a graph given from a training session of a CNN ......................................................... 11
Figure 2.3 - A simple overview of an ANN architecture ................................................................. 13
Figure 2.4 - An example pipeline of a CNN using images of faces as input ......................................................... 14
Figure 2.5 - A single kernel(K) applied to the input(I) ......................................................................................................................... 15
Figure 2.6 - Rectified linear function (ReLU) ......................................................................................................................... 16
Figure 2.7 - Max pooling applied to a 4x4 matrix ......................................................................................................................... 17
Figure 5.1 - Non-FER images from AffectNet dataset ........................................................................... 23
Figure 5.2 - Misclassifications from the manually annotated set. From the left, Happy, Sad, Neutral ......................................................... 23
Figure 5.3 - Illustration of the detection and ROI extraction preparing the dataset for classification ......................................................................... 24
Figure 5.4 - 4 Valid augmentation techniques from the left and the last one being a bad example ......................................................... 25
Figure 5.5 - VGG16 architecture with all layers frozen but the last conv layer and swapped the last FC ......................................................................... 25
Figure 5.6 - Final CNN model architecture ........................................................................... 26
Figure 5.7 - Failed detections on AffectNet dataset ........................................................................... 30
Figure 5.8 - Overfitting occurrence of VGG16 trained on AffectNet ........................................................................... 30
Figure 5.9 - Model loss and accuracy from training/validation ........................................................................... 31
Figure 5.10 - CNN CM ......................................................................................................................... 31
Figure 5.11 - Logistic regression solver sag CM ........................................................................... 32
Figure 5.12 - SVM poly kernel CM ......................................................................................................................... 33
Figure 5.13 - Random Forest using 100 N estimators CM ........................................................................... 33
Figure 5.14 - Decision tree CM ......................................................................................................................... 34
Figure 5.15 - KNN(K=24) CM ......................................................................................................................... 35
Figure 6.1 - Failed detections on AffectNet dataset ........................................................................... 35
Figure 6.2 - Real-time classification using DL for detection and classification ........................................................................... 36
Figure 6.3 - Real-time detection results angles ......................................................................................................................... 36
Figure 6.4 - Real-time detection illumination test, from the top 46 lux down to 1 ........................................................................... 37
Figure 6.5 - Down sampling using max pooling operation in a CNN layer ........................................................................... 45
Figure 6.6 - Relation between SoftMax regression and logistic regression ........................................................................... 45
Figure 6.7 - Linear regression between dependent variable weight(y) and independent variable height(x) ........................................................................... 46
Figure 6.8 - Logistic regression with sigmoid function applied between passing an exam and hours studied ........................................................................... 46
Figure 6.9 - KNN using K=3 to determine X ........................................................................... 46
Figure 6.10 - Real-time detections ResNet vs Violá-Jones ........................................................................... 47
Figure 6.11 - FER2013 public test set result CM ........................................................................... 47
Figure 6.12 - FER2013 private test set result CM ........................................................................... 48

List of tables

Table 5.1 - Number of samples in the AffectNet manually annotated dataset for 3 emotion classes ........................................................................... 22
Table 5.2 - Number of samples in the FER2013 test sets for 3 emotion classes ........................................................................... 23
Table 5.3 - Detection algorithms benchmark ........................................................................... 29
Table 5.4 - Test accuracy of CNN on test sets ........................................................................... 30
Table 5.5 - Logistic regression solvers ........................................................................... 32
Table 5.6 - SVM kernels ......................................................................................................................... 32
Table 5.7 - Random forest ......................................................................................................................... 33
Table 5.8 - Decision tree classifier ......................................................................................................................... 34
Table 5.9 - KNN ......................................................................................................................... 34
Table 5.10 - Naive Bayes ......................................................................................................................... 35
Table 5.11 - Real-time classification ......................................................................................................................... 36
Table 5.12 - Detection algorithms illumination benchmark ........................................................................... 37
**List of abbreviations**

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Full Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>AI</td>
<td>Artificial intelligence</td>
</tr>
<tr>
<td>FER</td>
<td>Facial expression recognition</td>
</tr>
<tr>
<td>ROI</td>
<td>Region of interest</td>
</tr>
<tr>
<td>ANN</td>
<td>Artificial neural network</td>
</tr>
<tr>
<td>CNN</td>
<td>Convolutional neural network</td>
</tr>
<tr>
<td>RNN</td>
<td>Recurrent neural network</td>
</tr>
<tr>
<td>DL</td>
<td>Deep learning</td>
</tr>
<tr>
<td>ML</td>
<td>Machine-learning</td>
</tr>
<tr>
<td>SVM</td>
<td>Support vector machine</td>
</tr>
<tr>
<td>KNN</td>
<td>K-nearest neighbor</td>
</tr>
<tr>
<td>ReLU</td>
<td>Rectified linearity unit(function)</td>
</tr>
</tbody>
</table>
1. Introduction

The number of areas utilizing computer vision for image related problems has expanded rapidly in the last decade, now integrated into systems ranging from self-driving cars to the detection of tumors in radiographs. Being able to automate tasks that the human visual system can perform and perhaps even beyond that of the human capabilities is a powerful tool indeed, as it opens possibilities that previously were only speculated in. One could say computer vision is the art of not only making a computer “see” but also imparting human intelligence of processing input. With the rise of AI, the subareas of Machine Learning and Deep Learning, the potential to integrate intelligent models with computer vision systems has become a popular method to handle increasingly complex application areas. One area that has been getting a rising amount of attention for its many potential applications is the art of detecting the emotional state of humans depending on their faces. This task is known as emotion detection or Facial expression recognition (FER) and is what this thesis work will revolve around.

To be able to detect and recognize expressions in a human face is a powerful tool that has potential use in a wide range of applications and industries. FER is the practice of using computers to map certain expressions using images or video footage of the human face. This technology is showing large potential when applied to tasks such as human-abnormal behavior detection, computer interfaces, autonomous driving, intelligent tutoring systems, health management, and other similar assignments [1, 2].

A typical FER system consists of face image acquisition, pre-processing, feature extraction, training, and classification [1, 3, 4]. To be able to get an accurate mapping of an expression to the face, the input data and feature extraction from the region of interest (ROI) must be performed before any form of classification can begin as the recognition task builds upon how well the first two steps are performed [4]. It is especially important for real-time application of FER as it involves detecting the face and performing feature extraction from a constant stream of frames while uncontrolled conditions may apply, for example, but not limited to, uncontrolled movement, different poses, and change in lightning. When these uncontrolled conditions occur, it may prove more challenging to perform the detection of the face(s) in the frames collected from the video footage. This, in turn, will affect how well one can extract the desired ROI if it is possible at all. These challenges are added to what is already considered to be difficult in the FER problem, such as the variety of sizes, and shapes for the human face. Facial detection is the first step of any FER and can be described as two smaller subtasks [5]. The first being the actual detection of the face and the second being to extract the region of interest (ROI) from the detected face by creating a bounding shape around the components that are part of the ROI, in this case, the nose, eyes, eyebrows, and mouth (also known as facial landmarks).

When the features have been extracted, one can start to look at the expression of the face in the images. There is a range of different algorithms and techniques to handle classifying the expression in the images, depending on the features extracted and used for the classification. A promising and well-studied strategy that has been applied and proven to be very accurate is to train a system to recognize expressions by using deep learning [6]. In deep learning, an Artificial Neural Network (ANN) can be applied through mimicking the work-flow of neurons in the human brain and map inputs to outputs by processing the input through a set of interconnected hidden layers containing neurons (or nodes) that activate depending on the data input. In this report, the neurons will be referred to as nodes. When using images as input data, a type of specialized ANN called Convolutional Neural Network (CNN) has been increasingly popular to apply for feature extraction and classification [7-9]. CNN's accept tensors as input into the network and just as with ANNs in general, the CNN contains several hidden layers that perform a range of distinctive calculations and feature extractions on the data.

After the features of the face(s) are extracted from the input image to a CNN, a SoftMax function is frequently used as a classifier. In exploring the possibilities for higher accuracy for the classification process, a set of machine learning classifiers replacing the SoftMax layer are purposed. Therefore, using CNN's as feature extractor, this set of classifiers have been trained and evaluated on a dataset. In this process, different ways to improve the CNN feature extraction for giving optimal features to the machine learning algorithms have been analyzed. Using FER in real-time systems could potentially open doors for many new application areas, e.g. within mental health-related areas. To explore how practical deep learning models can be deployed for such potential systems an algorithm using deep learning for face detection and classification in a real-time system have been implemented and evaluated.
2. Background

In this section, the intuition behind neural networks, machine-learning and important concepts of computer vision and image processing will be described in detail.

2.1. Datasets

When choosing a data collection to work with for classification purposes, there are two major categories that can be considered. One being datasets created in a lab or controlled environment (e.g. CK+) and the other being a set of images that are taken in the “wild”, meaning they are taken from an uncontrolled environment (e.g. FER2013 and AffectNet)[10-12].

2.2. Image preprocessing

To be able to accurately map an emotion to facial expressions in an image, the first step is to find a way to process the data from the image to prepare the data for proper classification in the next stages of the process. Before deep learning (DL) was starting gaining increased popularity in FER, you would extract your own features manually based on an image pre-processing step before feeding these features into your classifier[13, 14]. In this work, the feature extraction is done by the CNN automatically and the only pre-processing steps taken before extraction are face detection and alignment, image resizing and finally cropping.

2.2.1. Face detection

Face detection can be regarded as a special case of object-class detection. In object-class detection, the task is to find the locations and sizes of all objects in an image that belongs to a given class (e.g. car, face, animal etc.). Before being able to inspect faces for expressions and emotion, one must identify where the face resides in an image. Being able to distinguish a face from the rest of the human body, detect a face in an image or in everyday life (real-time) is a trivial task for the visual perception of the human vision. However, the same task is a challenging one for a computer. This is challenging mainly due to two reasons: Firstly, the anatomy of the human face can differ greatly across different subjects, for example, things like size, shape, textures and color. Secondly, in images, variation to lightning, pose and other varying properties can make the detection harder due to vast changes in the pixel values being looked at. These two challenges in combination make it hard to make face detection efficient, especially in real-time where exposure to several different circumstances may occur continuously (head movements, changes in light depending on pose etc). The detection step is not only a crucial part of FER but is also imperative for solving many other computer vision related problems, such as recognition and surveillance.

2.2.1.1. Viola-Jones Algorithm

There are many different approaches to detecting faces, a great breakthrough was accomplished with the viola-jones algorithm making use of “weak-classifiers” [15]. By using simple haar features, it can, after a great deal of training yield very promising results [16]. Plenty of training samples, both positive (images containing faces) and negative (images not containing faces) are required to make the best use of the algorithm. Viola-Jones is still considered to be very robust and accurate and has been used in many studies concerning different types on face detections and classification problems regarding the human face in the past [3, 7, 17].

2.2.1.2. Deep-learning based detection

Viola-Jones was first introduced back in 2001 and was considered a big breakthrough in object detection[15]. In more recent years, detection through DL has demonstrated more promising results as it is not as sensitive to the different conditions that may occur in an image mention earlier if trained properly. This new method enables more robust face detection in varying poses and varying lightning. This improved accuracy under varying conditions comes at the cost of more computational power required for the detection[18]. Both viola-jones and CNN’s are dependent on data to be trained upon to increase accuracy and robustness. However, there is a major difference in how the training occurs. The viola-jones uses AdaBoost which does not measure accuracy but rather only tries to minimize the loss, furthermore, the Haar-like features are predetermined features that no matter with how much training is given, will have trouble detection anything that deviates from full frontal faces with a set range of lightning. Since the CNN is not dependent on a set of predetermined features, it will have the ability to learn the characteristics of a face in positions deviating from full frontal, given that the training data is diverse enough to provide the CNN with these features. Although the CNN has much stronger capabilities when it comes to accurate

---

1 Visual perception is the ability to interpret the surrounding environment using light in the visible spectrum reflected by the objects in the environment. ("Visual perception", n.d)
and robust detection, it comes at the cost of resources, especially in terms of memory. Depending on the device to run the face detection algorithm, this factor may impact the choice between choosing a CNN based approach or viola-jones.

2.2.2. Feature extraction
To be able to classify an emotion to a face in a given image, the first step is to pre-process the image to facilitate the course of extracting the desired features. Depending on what technique is being used to extract features, this process may involve converting images to grayscale format and applying additional filters and/or manipulations if needed (blurring, sharpening, resizing). These steps vary depending on the algorithm used to make the classification as different algorithms might benefit from different kinds of features extracted from the data when making the classification[14, 19, 20]. In the case of using CNN’s for classification, the CNN extract the features it deems important to the class by itself iteratively when passing training data through the network, without any manual pre-processing. Although, it is still possible to extract features first and then let the CNN learn from these manually extracted features to make features of its own[8].

When any potential pre-processing is finished, the second step is to simply input the image into the CNN and as mentioned the features will be extracted automatically. Depending on how many layers chosen for the architecture, alongside what the chosen properties for each layer are, the number of features and how they are interpreted will differ, this process will be covered in greater detail in section 2.3. When the detection and the feature extraction steps are finished, one can start to perform the desired operations on the extracted features. In this thesis work, that means using the features to classify what kind of emotion the expression on the detected face is reflecting (Happy, Sad or Neutral).

2.3. Machine learning
Machine learning (ML) can be described as a subarea of AI. Furthermore, deep learning (DL) can be regarded as a subarea within ML. This order of areas can be observed in figure 2.1. Before explaining these notions, it is important to set out the differences between AI, ML, and DL. AI covers a very broad scope of concepts that specifically refers to the art of using computers to mimic the cognitive functions of the human brain(intelligence). In other words, AI can be described as a system and/or machine that carries out a set of tasks based on one or more algorithms that mimic intelligent behavior. ML, on the other hand, is part of the broad concept of AI but refers more specifically to the ability of these systems to learn for themselves, given an arbitrary set of data. This means that instead of having a set of predefined algorithms that run the same way every iteration, the algorithms are subject to change as they learn more about the data they are given to process. An important takeaway is that using an algorithm to predict an outcome based on some data is not ML. However, using the predicted outcome from the data provided to improve future predictions is ML.

Within ML, there are typically three different ways you can let your algorithms learn about the data you provide. These are supervised, unsupervised and reinforcement learning. This work is limited to working with supervised learning and will not delve into the other methods. Therefore, the reader of this report can assume that all the algorithms regarding ML and DP that are covered in the background section are all based on supervised learning. Supervised machine learning stands for many of the popular algorithms and techniques and algorithms that are used today and can be thought of as guiding the process with input training data(x) that maps to an output value(y). For supervised learning to work as intended, the correct output y corresponding to the input x is essential as it guides the learning process. Two of the most typical uses for supervised machine learning is classification and regression tasks[21].

![Figure 2.1 – Artificial intelligence subareas](https://developer.nvidia.com/deep-learning)
2.3.1. Classification vs regression

ML can be used for classification and regression tasks, but it is important to know the difference between the two when deciding on a model to use. A classification model has the task of mapping some input data to a discrete category or label. A typical example of a classification task would be what is being accomplished in this thesis work, that is, given an image of a face as input, mapping a label to the corresponding emotion to the face. In classification models it is common for the prediction output to be made as a probability of the labels involved, the result can then either be displayed as the probability for all the labels or simply take the highest probability and convert it into a class label as a single output. A regression model outputs a continuous real-valued, e.g. a floating-point value that can then be compared to other values. Most often these values are quantities, e.g. amounts, lengths or sizes[22].

2.3.2. Train, validation, and test

When a ML or DL model has been designed and the dataset with the corresponding input and output has been collected, it must be trained before it can be used to make any useful predictions. The data that has been collected for the model should be split into three smaller sets, each with a different purpose. In general, a good starting point is to split the set into an 80-20 \(^3\) or 70-30 ratio, where the higher percentage of the set will be used for training and the lower is used for validation will be used for validation and testing[23]. The training set is very straightforward and is used just as it implies, to train the model to learn about the features that belong to the classes that are provided in the set iteratively.

During the training session, the model will also perform classification on the validation set. This classification will be performed using the information the model has learned about the data it has been trained on so far in the training set. The reason why the validation set is separate from the training set is so that we can validate the model’s current accuracy on data that the model is not familiar with from the training set that is used to update the learnable parameters. The most important purpose of the validation set is to keep the model from overfitting (this concept is covered in a few sections) to the data in the training set.

Finally, the test set is used to test the model’s final performance and robustness when the training and validation have been completed. When using the test set, the model is not given the corresponding output \(y\) to each input \(x\), but instead must make the predictions based on what it has learned. This step is an absolute necessity for any model that is to be deployed and used in the field, as the entire purpose of a classification model is to be able to classify and label data without having information about the data beforehand.

2.3.3. Parameters and hyperparameters

In ML and DL, the two concepts of parameters and hyperparameters are important to grasp, as they have a significant impact on how well the model will perform. Parameters refer to a configurable variable that is internal to the model which value is determined based on the input data. This form of data is used when the model makes predictions, in DL, an example of parameters are the weights that are iteratively being updated throughout the training process.

Hyperparameters, on the other hand, are responsible for controlling how the parameters should be updated and assigned. One might think of the hyperparameters as being assigned and set before a model starts training or validating a dataset, and these, in turn, affect how the model learns and updates the internal parameters. Some examples on hyperparameters are the learning rate(\(\alpha\)), number of iterations in training, hidden layers (l), hidden units \(n[1], n[2] \ldots n[i]\), and choice of the activation function. These hyperparameters can be hard to estimate depending on the problem at hand and the way to an accurate model is often starting out with some initial values and performing trial and error tweaking of these hyperparameters to see how it affects the model accuracy. All hyperparameters have an impact on model accuracy. The learning rate will be discussed next, which affects how fast or slow the model updates the weights and learns.

2.3.3.1. Learning rate

The learning rate can be described as the path to minimizing the loss between the actual output and the predicted output from the training data. A typical start value to initiate models with usually ranges somewhere in between 0.01 – 0.0001. The learning rate is one of the hyperparameters and has a significant effect on how the weights of the model are updated. The main purpose of this hyperparameter in layman’s terms is to decide how small of a step the model should be updated with towards the minimum loss function value. A learning rate that is in the

---

3 The 80/20 rule, also known as the pareto principle states that in most events, 80% of the outcome is given from 20% of the causes.
upper bound of the range (closer to 0.01) risks overshooting, meaning we take a step that is too large when trying to minimize the loss function and shoot past the minimum value\(^4\). By setting the rate to a value in the lower range (closer to 0.0001) it will take longer to reach the point of minimize loss but reduces the risk of shooting over the minimum. There are also other different strategies to try to optimize the learning rate to avoid a bad local minimum\[^24\].

### 2.3.4. Underfitting and overfitting

When tuning different hyperparameters in the quest for the best accuracy, one might find that the model’s accuracy is good when it comes to classifying on the data that was included in the training set, but bad at classifying data it was not trained on. This is called overfitting and occurs when the model becomes good at classifying what it has already seen but generalizes badly. Overfitting is easy to spot as the results from training the model often points to very high accuracy on the training data, but much worse on the validation. Overfitting is one of the hardest challenges to overcome when using neural networks and machine-learning classifiers\[^25, 26\]. Some common techniques to prevent overfitting are data augmentation, meaning to modify the data within reasonable lengths (not distorting the data beyond recognition). Some augmentation techniques that are considered reasonable are cropping, rotating, flipping and zooming\[^27\]. Other important concepts to reduce overfitting are reducing the complexity of the model and for DL adding dropout\(^5\). Dropout will be explained in more detail further down in the document. In figure 2.2 an example of overfitting is seen in a graph given from training from a CNN on the AffectNet dataset for 100 epochs. One can see that after about 20 epochs the test(validation) accuracy peaks but the training accuracy keeps improving.

![Figure 2.2 - Overfitting seen in a graph given from a training session of a CNN](image)

Underfitting is the exact opposite of overfitting and occurs when a model is unable to classify the data it was trained on and even less data it has not seen before. It is rather self-explanatory that if a model performs badly on data it has been trained on, it will not be able to make meaningful predictions on data it has not seen. Underfitting is easy to spot as the information given from training results will indicate that the accuracy is poor for both training and validation data, as well as having a very high loss. Reducing underfitting is usually easier to deal with than reducing overfitting and the most effective method to reduce it is by increasing the complexity of the model and, if possible, add more features to the input samples. In the case of dealing with images the second mentioned solution may not be possible, so the best solution might be increasing the complexity of the model.

It should be noted that in terms of interest for research, when typing the “underfitting” keyword for the IEEE Xplore digital library, one ends up with 86 hits (as of writing). But when typing the keyword “overfitting” 1682, which does imply that more research is being done in the prevention of overfitting than underfitting.

### 2.3.5. ML algorithms

#### 2.3.5.1. Support vector machine

Support vector machine (SVM) is an ML algorithm mostly used for classification problems in supervised learning. Even if the mathematical process behind how an SVM performs its classification is complex, the intuition is very simple. Each input data is plotted as a point in an \(n\)-dimensional space, where \(n\) corresponds to the number of

---

\(^4\) [https://developers.google.com/machine-learning/crash-course/reducing-loss/learning-rate]

\(^5\) [https://deepnotes.io/dropout]
features in your problem domain. The value of each of the features will hold a specific coordinate in this space. When all the features are plotted, a hyperplane that separates the classes is drawn to perform the classification. The training time and memory complexity increase quickly even when using a simpler form of SVM with linear kernels and the number of samples and number of features in each sample has a significant contribution to this[28].

2.3.5.2. **Linear and logistic regression**
Linear regression is a basic form of regression that simply fits the best straight line between to represent the relationship between a dependent variable y and one independent variable x. One can input more than one independent variable by using multiple linear regression. Logistic regression is quite like linear regression but uses the logit function for classification[29]. In cases where more than two discrete classes, multinomial logistic regression can be used for multi-label classification problems[30]. The multi-label classification is also known as one-vs-all classification, it works by training multiple logistic regression classifiers, one for each K classes in the problem domain. This means that when one train each of the logistic regression classifiers individually, it is trained on one class as true and all the other classes as false, hence the name one-vs-all6.

2.3.5.3. **K-nearest neighbor**
K-nearest neighbor (KNN) is a lazy algorithm that utilizes a non-parametric technique when making its classification. Essentially, this means that the algorithm does not make any presumptions about the data features inputted to the model and that it does not need any training data points to generate the model. This makes KNN a strong classifier when it comes to data that one knows very little or nothing about and gives a fast training session, but a slower testing phase[31].

KNN bases the classification on feature similarity, how closely related one data point is related to another data point. K is the number of nearest neighbors for a certain data point and the number of closest neighbors to this data point votes in what its classification outcome should be. This process is accomplished by getting the points closest to the point P that is to be classified by creating a set S of K points with the shortest distance to P based on the Euclidean distance. The majority class label present in the set S is then set as the class for P. In the most elementary case where K = 1, the algorithm is known as just “nearest neighbor”[31].

2.3.5.4. **Naive Bayes classifier**
Naive Bayes classifiers are a collection of probabilistic ML models that are based on Bayes theorem. They are highly popular to use for text classification (e.g. spam classifiers). The Bayes theorem is defined as follows:

\[
P(A|B) = \frac{P(B|A)P(A)}{P(B)}\]

Using the theorem, given that B has occurred, one can find the probability of A happening7. This is what the term naive refers to, as the assumption is that the predictors or features are independent. Meaning that it does not correlate between say, the mouth and eyes for a happy face but looks at them both separately. There are different types of Bayes classifier that can be applied to continuous or discrete values, as well as binary or multiple class classification problems. The type that is used in this thesis work is the multinomial naive Bayes classifier for multiclass problems. Naive bayes is a very popular algorithm to apply when dealing with text classification problems, such as spam identifiers[32].

2.3.5.5. **Decision trees**
A decision tree in ML is essentially a flowchart represented in a top-down tree to help decide with classification related problems. The bigger idea of using a decision tree is to split the dataset into smaller subsets of data until no further splits can be made and all the leaves lead to a classification of the input[33].

2.3.5.6. **Random forest**
Random forest is an ensemble algorithm used for classification, meaning that it consists of a group of classifiers instead of one. In the previous heading, decision trees were described and as it turns out, random forest is built up randomly created decision trees. Each prediction is made using each decision tree in the ensemble then the final prediction is based on the majority voting method. The majority voting method can be thought of as a political voting system, where each tree represents a person from a political party. When all the votes have been counted from the trees, a decision is made.

6 https://en.wikipedia.org/wiki/Multiclass_classification
7 https://brilliant.org/wiki/bayes-theorem/
The advantages of using an ensemble of decision trees instead of just one is a more robust result as it is coming from \( n \) number of trees instead of one. It reduces overfitting since it takes the average of all prediction, canceling out biases. The ensemble can also be used for regression if needed. The disadvantages are rather straightforward, as you add more trees, you increase the complexity of the algorithm, making it slower in generating predictions. The results are not as self-explanatory as well, because in a single decision tree one can simply print a graph of the tree and follow the flowchart[33].

2.4. Deep learning

DL builds upon the concepts of ML and can be thought of as a subset to ML. The word deep can refer to the depth of a given architecture of a neural network, and while it may be debated of what classifies as deep, in general, all neural networks with an architecture that consists of more than one layer (which would be referred to as “shallow” learning) fit this category. In DL, there are many various types of neural networks that are used to solve different problems. Neural networks are particularly useful for classification or recognition problems. In the following subsections, there will be an in-depth explanation of how these types of networks are constructed and function intuitively.

2.4.1. Artificial Neural Network

Artificial Neural Networks (ANN) refers to the most typical type of DL model that consists of neurons (or nodes) layered together in layers to perform different computational tasks. Even though the networks themselves might be very complex in their nature, the nodes are remarkably simple. The nodes of an ANN are nothing but a floating-point value that is more commonly referred to as weights. These weight values are updated after each iteration through the neural network while the network is learning. The structure of a very simple ANN architecture can be seen in figure 2.3, where one layer takes the input data, passes it through two hidden layers with 4 nodes each and gives two outputs. There is no upper bound to how many layers an ANN might consist of, however, the deeper it is, the larger the solution domain to solve the specific classification problem there is. This is in many cases the desired feature and is traded off for more complexity and longer training time. Precautions should be made when deciding on the architecture for neural networks, as deeper networks might be prone to overfitting[34].

![Figure 2.3 - A simple overview of an ANN architecture](image)

When a model architecture has been decided upon, the model is initialized, and the weights are either loaded from a file or set to arbitrary values. Once the input has been given to the neural network and output has been received, the loss (rate of error) is computed for the output. Having the value of the loss function, the gradient of the chosen loss function is computed in relation to all the weights in the network. Once the value for the gradient of the loss function is acquired, this value can be used to update the model weights. This is process is known as backpropagation. Through each iteration, the values of the weights will be updated to try and move in the direction that minimizes the loss function and takes it closer to its minimum value. Next, the learning rate is used and multiplied with the gradient[34].

2.4.2. Bias

Bias is another learnable parameter, meaning it will change during training. Bias helps the network decide when a certain node activation is meaningfully activated and by how much. The bias is used right before the weighted sum is put through the activation function and is simply added to the sum of the weights. Bias essentially help the model get meaningful information from patterns that otherwise would not pass through the origin[34].
2.4.3. Batch size
The batch size is the number of samples that are to be put through the neural network in a single pass, and we will keep taking samples until our training samples are exhausted and that will end the forward and backward pass of all the samples (epoch). In general, the larger the batch size, the quicker the model will finish through our given epochs during training. This hyperparameter is very dependent on how many samples the given machine can process in parallel through the network. The batch size is another hyperparameter set before training begins[34].

2.4.4. CNN
CNN is a specialized type of neural network that is mainly used in the field of computer vision to solve a variety of different tasks regarding visual imagery and was introduced by Yann Lecun et al. in 1998[35]. One of the biggest reasons why CNN are showing such good performance in these types of problem is its ability to detect and identify underlying patterns that are too complex for the human eye to detect. In a CNN the input image provided is run through several hidden layers that will decompose it into features and perform a set of operations on the data. Each layer in the networks builds upon the previous layer and the output features are then used for classification[8]. There are many possible components that add properties for a CNN depending on the architecture chosen, these include but are not limited to, convolution, pooling, tangent, squashing, activation and normalization.

CNN’s are inspired by the biology of the visual cortex, where local receptive fields represented by groups of neurons respond to a subsample of what your eyes see. Together these fields overlap each other to cover the entire field of view of the human vision. Basically, doing the actual convolution, i.e. breaking up the data received from the eyes into subsamples and processing them individually and these subsamples will reassemble a bigger and bigger picture of what you are seeing the closer you get to the final output. This makes CNN’s outstanding in performance when it comes to image classification. In this thesis work, CNN’s are used for face detection as well as image classification. An example of a CNN used for this purpose can be observed in figure 2.4 where the input are images of faces and the output is one of three classes using the SoftMax function. Another reason for using CNN’s to extract features and then classify them is that it requires a lot less computational power than a standard ANN. If one were to put the images through a standard ANN, one would first have to flatten the image into an order-1-dimensional tensor (or array if you will) where each node would correspond to a pixel value. To make an example of the number of input parameters just for one layer, a 300x300x3 image would have 270 000 parameters mapping to the input layer. With just a single hidden layer with 1000 nodes, this would make the neural network have a total of 270 million learnable parameters. This quickly gets computationally undesirable for larger architectures with more layers. CNN’s, on the other hand does not need to learn the weights corresponding to each pixel value, but instead can learn the weigh mapping to each feature layer, which will be explained in the convolution section in detail.

There are different conventions when it comes to identifying what a layer is in the DL community. In this thesis work, I will be referring to a CNN layer as the combination of a conv layer, its corresponding activation function and potential batch normalization and dropout as one layer, and pool layers as another separate layer. The subsections that follow contain further specific information on each layer’s functionality.

Figure 2.4 - An example pipeline of a CNN using images of faces as input

2.4.4.1. Convolution

A convolutional (Conv) layer takes a tensor as input and outputs another tensor, most often in a different order than the input. An order-3 tensor is simply an image of dimensions \((h \times w \times c)\) where \(c\) is the color dimension of the image. The whole dataset as input to the CNN can be regarded as an order-4 tensor where e.g. \((64 \times 64 \times 3 \times 20)\) indicates that 20 RGB images of 64x64 in height and width will be used as input. Before going into a more detailed explanation of the convolution operation, it is worth noting that convolution regarding neural networks doesn’t always correspond to the operation used in pure mathematics or engineering.

In CNN’s, convolution is simply the art of breaking up the data received from images into subsamples and processing them individually and then using these subsamples as features to identify what is unique to a specific class. Provided an input image, there will exist some feature map that will act as a feature detector. In some research papers, this feature map is referred to as a kernel, this convention will be used to refer to these feature maps in this thesis work as well. An arbitrary kernel of a layer is represented as a tensor that matches the input tensor to the network, in this case, an order-3 tensor. This order-3 tensor will be smaller than the input image \(h \times w\) but keep the same number of channels \(c\) (e.g. 3x3x3 to a 300x300x3 image). All the kernels together (forming one layer) can be represented as an order-4 tensor where the fourth dimension is the number of features the layer will detect. For instance, suppose a 3x3x3x40 tensor, this means we will detect 40 features for this layer. When a tensor is inputted to a convolutional layer, the kernels are applied to the input one by one and “slide” across the input from left to right and top to bottom. This “sliding” is determined by the stride hyperparameter that gives a value of how much the kernel should move in each step. To keep the kernel from stepping out of the image bounds, one specifies the next hyperparameters padding, which can be set manually by adding \(n\) number of rows and columns of a pre-defined number (zero is most common) around the image. This parameter can also be defined as “valid padding” which means no padding at all, or as “same padding” which means choosing a padding value so that the output size has the same \(h \times w\) as the input size. “Same padding” is often used when one wants to avoid losing information in the conv operation, as the output will be the same in terms of height and width. The convolution operation can be observed as a visual representation in figure 2.5 for more clarity, this example can be thought of as a grayscale image of 7x7x1 with a single kernel of 3x3x1x1[36].

![Figure 2.5 - A single kernel(K) applied to the input(I)](image)

To describe the convolution operation for one layer, the following conventions will be used: padding \(p^l\), stride \(S^l\), filter size \(f^l\), number of kernels \(n_c^l\), where the \(l\) stands for the current layer.

With this convention, suppose we are given an \(n_{h^{l-1}} \times n_{w^{l-1}} \times n_{c^{l-1}}\) input image, where \(h, w, c\) stands for height, width and number of channels respectively. Then the new height, width and, number of channels after a forward pass through the conv layer can be calculated as \(n_{h^{l}} = \left\lfloor \frac{n_{h^{l-1}} + 2p^l - f^l}{S^l} + 1 \right\rfloor\). The new channel \(n_c^l\) will be equal to the number of kernels in the conv layer and each kernel in the layer will be \(f^l \times f^l \times n_c^l\) and the activation \(x^l = \sigma\) for this layer will be \(a^l \rightarrow n_h^l \times n_w^l \times n_c^l\). When using batch gradient descent or a vectorized implementation, the activation is \(A^l \rightarrow m \times n_h^l \times n_w^l \times n_c^l\) where \(m\) denotes the examples in the batch.
For many cases, the features that are derived from these convolutional operations by the network are not noticeable to the human eye, which is exactly why the networks are so efficient at performing classification. With enough training, they can detect theoretically anything in an image, given that enough data is provided.

When using the convolutional layers in CNN’s, one must carefully consider what convolutions to use as more kernels equal more parameters. For example, if a layer would have 128 3x3x3 kernels, this would be equals to 3x3x3x128 + the bias parameter for each kernel. In other words, 3584 learnable parameters. Notice that the number of parameters is not bound to the size of the input image, which makes CNN’s less prone to overfitting.

Regarding the choice of size for the kernels, many of the state-of-the-art CNN’s, use kernel sizes that are odd, e.g. 1x1, 3x3, 5x5, 7x7 and so on. The reason for this is debatable and is rarely discussed or given a straight answer to in the state-of-the-art literature. However, one might reason that it is because when padding (making the output the same as the input in dimensions), one is trying to avoid losing information from the image.

2.4.4.2. ReLU

The Rectified Linear Unit (ReLU) is not a separate layer or component of the convolutional network. It is part of the convolution process and is applied after the element-wise addition of the product of the feature map tensors has been performed with the input tensors. Before the output of the conv layer is passed forward to the next layer unit, the ReLU (non-linearity function) is applied and then the bias is added before the next forward pass is being made. ReLU can be described mathematically as:

\[ f(x) = x^+ = \max(0, x) \]

ReLU is arguably the most popular activation function used in CNN’s[7, 8, 37, 38]. A few reasons why ReLU has gained such influence is that firstly it is computationally cheap since it essentially is just a max function of two values[39]. Secondly, it converges fast, meaning, since its linear, it does not plateau for large values of \( x \). Therefore, it does not experience the vanishing gradient problem that is present in other activation functions such as Sigmoid[10].

![Figure 2.6 - Rectified linear function (ReLU)](https://www.tinymind.com/learn/terms/relu)

2.4.4.3. Pooling

Another layer that often is incorporated into convolutional neural networks is the pooling operation. Note that it is possible to implement this layer alongside convolutional layers, but it is by no means required to get a conv network up and running. There are several different styles pooling, in all the CNN discussed here, the max pooling operation is used. In the pooling layer there are no learnable parameters, therefore not making a compelling difference to the computational process. The pooling procedure works in a similar way to the conv layer by applying a kernel over the input tensor volumes. In figure 2.7, the max pooling operation using a 2x2 kernel applied to a 4x4 matrix can be observed. The kernel is first put in the upper left corner of the input matrix, the max value is extracted (20) and is inserted into the 2x2 output matrix. The kernel continues to move from left to right and top.
The max pooling operation takes an input volume of $W_1 \times H_1 \times C_1$ and requires two hyperparameters to be set before runtime, the spatial extent $F \times F$ and the stride $S$. The operation produces an output volume of $W_2 \times H_2 \times C_2$ to be inputted to the next layer where:

$$W_2 = \frac{W_1 - F}{S} + 1$$
$$H_2 = \frac{H_1 - F}{S} + 1$$
$$D_2 = D_1$$

There are two main purposes of the pooling operation, the first being helping prevent the model from over-fitting by providing as it makes an abstraction of the input volume[9]. It also reduces the input volume, hence reducing the number of learnable parameters and saving computation resources[37]. This can also be observed in figure 2.7 as the input volume of 4x4x1 outputs 2x2x1.

### 2.4.4.4. Flattening

After putting the input image through a CNN to extract all the features, a layer called flattening is applied to restructure an order-n tensor into an order-1 tensor(array). This layer is usually not referred to as a specific layer but a part of the fully connected layers and sometimes the fully connected layers are referred to as “hidden layers” (taken from ANNs mentioned previously)[7, 8]. This layer marks the end of the feature extraction process of the CNN and the beginning of the classification process. If the final output of the final layer in the extraction process would be a 3x3x256 tensor, one would derive some array of 2304 elements from it. From this layer and onward, the operations unique to the CNN essentially is finished and the network functions like a simple ANN that was explained in section 2.4.1.

### 2.4.4.5. Full connection

With the flattened array as input, one or more fully connected layers can be further applied to help the classification procedure. These layer(s) function in the exact same way as a normal ANN hidden layer, where each node in the previous layer maps to the next one and so forth. Please see section 2.4.1 for a more elaborate explanation of this.

### 2.4.4.6. SoftMax and cross-entropy

SoftMax in combination with cross-entropy functions as a way for the CNN’s final output to be displayed as probabilities summing up to 1. Logistic regression is closely related with SoftMax as a synonym to it is “multi-class logistic regression”, see appendix for a figure on how they relate. The SoftMax function takes an un-normalized vector as input. In the case of a neural network, this is the vector derived from the last of the fully connected layers array transposed. After applying the SoftMax function to the vector, each element $i$ of the vector $V$ will be in the interval $[0,1]$ and the sum of the elements will be:

$$\sum_i V_i = 1.$$
This vector's values can then be extracted and evaluated as they correspond the probabilities of each class predicted from the original input image. The SoftMax unit is essentially a classifier that outputs the probabilities of your classes. It works together with the cross-entropy loss function, where SoftMax uses the gradient of the cross-entropy function to update its weights when applying stochastic gradient descent[41].

2.4.5. Transfer learning
In some cases where the problem domain is similar and/or the dataset for training is very small, transfer learning can be applied instead of constructing a new model. By using transfer learning, one can use a pre-defined model that has already been trained on a dataset and instead of retraining the model from scratch. Simply by downloading the final weights that were acquired from the training of the model and swap out the last output layer to a different classifier or a fully connected layer and SoftMax with the number of classes that you are using in your problem. If one wish to retrain some of the layers in the pre-defined architecture but not all, this can be achieved by “freezing” the layers whose weights should not be updated. Unless one has an exceptionally large dataset or a very different problem domain, it is advantageous to at least try out transfer learning before moving on to develop new models[42].
3. Related Work

In this section, previous works and research in FER, alongside some state-of-the-art CNN models are described and investigated in detail. The challenges and achievements of the area are covered as well.

3.1. ImageNet challenge and state-of-the-art CNN’s

Before covering separate papers regarding the specific problem of FER, a few successful state-of-the-art models using CNN’s related to object-detection and classification will be looked upon. DL and CNN’s have since 2012 to present date been winning the ImageNet challenge. The contest named the ImageNet challenge, short for Large Scale Visual Recognition Challenge (ILSVCR) is one of the largest classification competitions that evaluate algorithms for object detection and image classification in the world. According to the Stanford Vision Lab that hosts the contest, their motivation behind it is described as follows:

“One high-level motivation is to allow researchers to compare progress in detection across a wider variety of objects -- taking advantage of the quite expensive labeling effort. Another motivation is to measure the progress of computer vision for large scale image indexing for retrieval and annotation.” [43]

Alexnet is considered one of the bigger break-throughs in using CNN for image classification and entered the ImageNet competition and took first place in 2012 when the model was first introduced[44]. Alexnet is by no means a network considered to be overly complex by the standards of recent years, but nonetheless proved very powerful. Alexnet uses 11x11, 5x5 and 3x3 convolutional layers and applies max pooling, dropout and ReLU as the activation function. The architecture is built up of two pipelines making computations in parallel before connecting back together into a sequential pipeline at the dense layers and SoftMax classification. The reason for the architectures unique parallel pipeline is due to the training being processed on two Nvidia GeForce GPUs simultaneously. Moving forward two years in time (2014) the Inception V1(GoogleNet) from Google made its first appearance. It achieved a top-5 error rate very close to human level performance which even required the organizers of the challenge to evaluate. It achieved a top-5 error rate very close to human level performance which even required the organizers of the challenge to evaluate. In the end, it did not beat human level performance but came very close. The architecture took inspiration from LeNet and used 1v1 convolutional layers to decrease the number of parameters while still being able to use a very deep architecture[45]. Inception has also been further improved upon in Inception v2, v3, and v4. The same year as Inception V1, VGGNet also made its debut in the ImageNet challenge. VGG uses explicitly 3x3 convolutions and utilizes lots of filters. VGG has since its appearance become widely publicly available in different variations and is frequently used for feature extraction from images. One of the major limitations of VGG is the number of trainable parameters, which is close to 138 million. Compared to Inception v1 using 4 million and was published the same year, this is quite a lot and does impact the amount of time required for training significantly[46].

While all these results are very impressive, there is one major limitation on many of the ImageNet images that are used for classification. The images that are used often contains only the object that is to be classified or at least the object that is to be classified is centered or in great focus. In real-world scenarios, this is usually not the case as the object might be obscured or there might be a lot of other objects that might be like the one to be classified. It is however understandable that the images from the dataset are given this way, as one image is related to only one specific class and not several classes, even if there are images with several objects that all are viable for classification. This problem is a fact for all tasks that build upon object detection and classification, FER included.

3.2. FER state-of-the-art

This subsection is dedicated to reviewing some related works and their approaches to the FER problem and identify successful techniques but alsoshortcoming in the current state-of-the-art. Furthermore, it will go into detail why FER is such a challenge task and look at how previous works have been trying to overcome these challenges.

In a conference in Malaysia 2009, Adeshina, Lau [3] et al. showed their review of real-time facial expression recognition approaches. In their review, they cover extensive comparison of different approaches to both face detection and tracking, facial feature extraction and finally facial classification approaches. They concluded that most of the reported systems with high accuracy are restricted to only frontal views of the expression at a single scale. This is a challenge that has been around for some time and others have arrived at the same conclusions regarding challenges[2] to conditions in the position, poses, obstruction and illumination of the faces in images. Also, they mention that the distance from the camera has a great impact using some of the state-of-the-art detection algorithms. Furthermore, many algorithms are covered and compared for especially detection, but deep learning
and especially CNN’s are mostly covered for expression classification and not detection. They purposed that IR-images might be used to further enhance expression recognition systems in the future.

Previous research backs the fact that CNN’s outperform previous methods in classification related problems, especially in image classification problems. In 2017 Ronghe, Nakashe [6] et al. performed a hybrid architecture approach using RNN-CNN combined. In their work, the Recurrent neural network (RNN) is used in combination with the CNN to learn temporal and spectral features of emotions and to predict the subjects in the video’s emotional reactions.

Jung, Lee [37] et al. performed a comparison of DNN and CNN in 2015 and achieved impressive results with their CNN model. They used the CK+ dataset that contains 327 image sequences that include the 7 universal emotions while performing an evaluation on their CNN and DNN respectively and achieved an 86.54% recognition rate with CNN compared to a 72.78% rate on the DNN. Speculations that there is the possibility that the DNN was overfitting were taken into consideration. On top of the neural networks, they also developed a system for real-time facial detection and feature extraction. They used a Haar-like detector and then cropped the images and performed normalization before feeding them into the CNN and DNN.

In 2017, Fathallah, Abdi [47] et al. presented a new architecture network based on CNN for facial expression recognition. They evaluated and trained their model using several large public databases, such as CK+, MUG, and RAFD. The motivation behind the use of many databases is that a deep network needs a huge amount of training data, therefore, combining many databases to get a final larger one was their approach. Their approach in the CNN model used four convolutional layers where the first three are followed by max pooling layers and the last in the line is the fully connected layer. They recorded a total of 300000 iterations of the database that took 4 days to reach accuracy close to 99% in their training step. They also compared several approaches with 3, 5 and 6 layers before testing their final 8-layer structure.

Using the FERC-2013 for training, Kumar, Kumar [7] et al. applied a CNN to better predict human emotion(frame by frame) and researched how intensity changes on a face from low to high level of emotional intensity. FERC-2013 database provides 32 000 low-resolution images that are described as “emotions in the wild”, i.e taken under some changes in parameters for the images, such as distance and so on. They applied the well-studied Viola-Jones algorithm for face detection in the database set [1, 4, 15]. In their result for the detection stage, it showed the drawbacks of the Viola-Jones method when trying to detect non-frontal face images. Their final 9-layer CNN model shows a promising 90%+ accuracy on a set of seven different emotions (universal set).

Several studies point to that FER are interlinked with not only the face but also the gender and age [48-51]. Age acts on certain emotion in certain years of age, for example, children have an increase in sensitivity to all emotions from the age of 5 until adulthood, except for happiness and fear. Meaning that the emotions of happiness and fear are recognized much earlier than the rest in a humans life [52]. It is found that women express their emotion more intensely than men, true for both the facial expression and in a more intense mood experience. It is also stated by et al. that women express more positive emotional force than men in general.

All the above studies show an increase in accuracy when using DL for classification of the human emotion to certain facial expressions. Most of the studies are restricted to databases containing static images in controlled circumstances (e.g. CK+, JAFFE, MUG) while others provide real-time footage or in some cases databases containing video footage frames. Something to take note of is that many of the algorithms that are being used in the state-of-the-art that are published today are limited to using haar-based face detection and using grayscale datasets for training and validation. To extend the possibilities for FER this thesis work will utilize the fact that CNN has been proven to be an outstanding feature extractor in many of the works mentioned in this section of the thesis work. Further, by using deep learning for performing the detection instead of the Viola-Jones algorithm in attempts to try and overcome the restriction to detection in full frontal images only. This work also considers the use of alternative classification methods instead of the SoftMax used in many other research articles relating to FER, to try and increase the accuracy and compare these models to the SoftMax function in terms of accuracy [7, 47, 37]. On top of these mentioned attempts on the identified challenges, the dataset that will be used will be hosting images given from “the wild” as many of the related works that have been covered only provide datasets given from controlled environments.
4. Problem Formulation

This thesis work is built up from several smaller problem subtasks that together form a set of research questions that have been answered or partially answered. The research questions are built around the steps that are required to implement a robust FER system and touches on the smaller sub components that make the whole system. The methods and results that are presented are based on the following research questions:

1. How well does a deep learning approach using CNN compare against the well-studied detection algorithm Viola-Jones in terms of detection accuracy and detection of faces in different orientations than full-frontal view?
2. Can different machine-learning algorithms be used to increase performance in terms of accuracy of the extracted features of a CNN in comparison to the more commonly used SoftMax function?
3. What is the difference in terms of speed and accuracy between using the different detection approaches mentioned in question 1 in combination with classification using a CNN for three emotions in real-time?
5. Methods and Materials

The methodology is divided into five parts: Data collection, image pre-processing and ROI extraction, CNN testing and validation, CNN testing and evaluation, and finally ML classification. The selected dataset for the training and validation is AffectNet and the three classes to be classified are happy, sad and neutral. For the training and validation, two CNN models are used, one through transfer learning (VGG16) and the other built from scratch. In testing, the provided test set that comes with AffectNet is used and FER2013 is also used to further confirm the robustness of the results[46]. Finally, the model developed from scratch is used for intermediate layer feature extraction and these features are used to train and evaluate a set of ML models. If not stated otherwise, the reader can assume that all the implementations were made using the Python programming language in the Anaconda data science platform for the Windows operating system. The development environment used was spyder 3.0.

5.1. Known limitations

Before delving into the methods used to derive the results, some of the limitations known to have a greater impact on this thesis work will be covered. In the classification stage of the process, only two of the six universal emotions have been used (and neutral). This will affect the overall accuracy of the system and it sets a lower limit for training data to be used. Also, while classifying these emotions, there is no intensity level of arousal. Meaning that to more accurately measure an emotion, one would need to have the amount of arousal combined with the labeled emotion, not just the static label. Additionally, the developed system does not consider age or gender as individual classes nor take any of these two into account when performing expression classification, as seen in the final paragraph of the related works section, this has an impact on the emotion related to the facial expression.

In terms of more technical limitations for the method used, the CNN uses only the raw images of the detection made. Meaning no pre-processing in terms of extracting facial landmarks or similar is performed, which could potentially make the CNN learn better. Furthermore, the biggest technical limitation is the data provided to the CNN to perform feature extraction on. The images used in this thesis work comes vastly different resolutions because they are originating from the web. When feeding the images into the CNN, they all must be the same size in terms of height, width, and dimensions. This means that some images are downsampled while others are upsampled, causing loss of information and in other cases stretching of the images. Lastly, the number of images collected affects how well the models can generalize and how much complexity can be increased without overfitting.

5.2. Data collection

As mentioned previously in the background section, the dataset size and conditions regarding e.g. different illumination settings and variations in pose may impact the result greatly. Because one of the purposes of the problem formulation is to be able to make classification in real-time, the AffectNet with images taken from the wild was used for training and validation purposes.

AffectNet is a dataset of 1 million facial RGB images collected from the internet by querying three major search engines using 1250 emotion-related keywords in six different languages[12]. From these 1 million images, 440 thousand of them are manually annotated images and these are the ones used for training in this thesis work. The images vary in sizes as they are taken as is from the internet. The reason for using the manually annotated instead of the automatically annotated is that the algorithm used for the automatic annotation is not 100% accurate, which in some cases, leads to the annotated images being labeled incorrectly. Even if the images are manually annotated they come with some samples of images that are distorted or does not contain faces at all (see figure 5.1). Additionally, the manually annotated images are incorrectly labeled due to human error in the labeling process in some of the cases (see figure 5.2).

<table>
<thead>
<tr>
<th>Emotion</th>
<th>Happy</th>
<th>Neutral</th>
<th>Sad</th>
</tr>
</thead>
<tbody>
<tr>
<td>Samples(train/valid)</td>
<td>134915</td>
<td>75374</td>
<td>25959</td>
</tr>
<tr>
<td>Samples(test)</td>
<td>500</td>
<td>500</td>
<td>500</td>
</tr>
</tbody>
</table>

Table 5.1 - Number of samples in the AffectNet manually annotated dataset for 3 emotion classes
AffectNet is one of the largest datasets to be found online for FER as of writing and this is the core motivation behind why the dataset was selected [12]. DL models benefit from more data being used in the training, the bigger the dataset, the higher the accuracy and the benefit of less risk for overfitting the model. Many of the popular older datasets used, such as CK+ and FER2013 are limited to grayscale images that are very small (46x46 in the case of FER2013) which might impact the results and are limited to a smaller amount of data. Furthermore, these datasets are limited to full frontal poses which would not benefit classification in real-time where uncontrolled head movement is very likely to occur. On top of this many of the published papers in the state-of-the-art have been using datasets where data was collected solely from lab environmental based datasets (e.g. CK+, Jaffe) when performing their training and validation [7, 37, 38].

To perform extended testing and to confirm robust results, the private and public test portions of the FER2013 dataset have been used in the testing phase. FER2013 dataset consists of 28,709 samples divided over 6 classes and have been featured on Kaggle.com as a part of their FER challenge. The data consists of 48x48 grayscale images of faces where the faces are centered in the image. The FER2013 comes with a private and public test set to try the accuracy of developed models on. The original purpose was to use the private set for validating the results before testing on the public set[11]. I have used both sets to test the model trained on AffectNet. The test classes and the number of samples for them can be observed in table 5.2.

<table>
<thead>
<tr>
<th>Emotion</th>
<th>Happy</th>
<th>Neutral</th>
<th>Sad</th>
</tr>
</thead>
<tbody>
<tr>
<td>Private samples</td>
<td>897</td>
<td>626</td>
<td>594</td>
</tr>
<tr>
<td>Public samples</td>
<td>895</td>
<td>607</td>
<td>653</td>
</tr>
</tbody>
</table>

Table 5.2 - Number of samples in the FER2013 test sets for 3 emotion classes

5.3. Detection and extraction

Two detection algorithms were tested for detection and one of them was also used to extract the ROI from the images to prepare a folder structure for training with the DL models. For the extraction, a DL-based approach using ResNet was used. To prepare the dataset for extraction and placement in folders, the Pandas library for python was used to store the file locations with their specific class label in a data frame. Using this data frame, one image at a time can be loaded into memory and detection can be performed, before moving the file in the data frame to the appropriate folder. The weights and model used for the CNN were downloaded via the OpenCV framework utilizing the DNN module and was used with the Python programming language. The model in OpenCV uses the original implementation given in the original research article of ResNet[53]. A detection prediction rate of 0.9 was used to filter out any images that do not contain faces (see figure 5.1 for some examples). After detection, the images were cropped and resized to the input size of the CNN classifier model before being placed in separate folders for training and validation (see figure 5.3). The detection and ROI extraction algorithm
were used on the test set as well, preparing these images for the final evaluation. A pre-trained haar-feature based
algorithm (Viola-Jones) was also used in a test for detection accuracy and robustness in addition to the DL-based
algorithm. The Viola-Jones algorithm was also implemented with the OpenCV version of the weights and the
implementation.

![Diagram of detection and ROI extraction preparing the dataset for classification](image)

**Figure 5.3 - Illustration of the detection and ROI extraction preparing the dataset for classification**

### 5.4. CNN’s

For the training and classification, two models were trained and evaluated, VGG16 and my own developed
architecture. In both cases, the same training, validation and test data were used. The networks were trained on a
desktop computer utilizing a Nvidia GeForce GTX 1060 6GB graphics card. The implementations for both the
networks were made using Tensorflow and Keras libraries with Python. When performing the feature extraction,
the features were stored in Numpy arrays and these arrays, in turn, was stored in Pandas data frames. Storing the
arrays in data frames provides a simple way of iterating over them and gives the ability to look upon the arrays
in the variable explorer of the Spyder development environment. What this means is that one can look at the
extracted features in a table format where not only the values, but their types and shapes can be displayed for the
user. This provided great help when debugging and making sure that the features were extracted correctly.

The pre-processing before the images were fed into the CNN’s includes the detection and ROI extraction
mentioned in the previous section, as well as image data augmentation and further image resizing to fit the
networks.

The following augmentation techniques were applied to the dataset:
- Rotation range shift of 20%
- Shear range shift of 0.2
- Zoom range shift of +- 0.2
- Width and height shift range of 0.1
- Horizontal flips allowed

The same augmentation techniques were made for the test set before predicting and evaluating the final accuracy
of the models. The low threshold values of each of the techniques were selected to reduce the model’s risk of
overfitting but at the same time not augmenting it too far from the original shape, see figure 5.4 for some examples
of good vs bad techniques. See the background section 2.3.4 for more information on how this can be applied to
prevent overfitting.
For the first test with the CNN’s, an implementation of VGG16 was used. VGG16 was selected as the first network to try due to its simple straightforward architecture that gives an intuitive understanding of how the network works. The original VGG16 paper was also the first one after AlexNet that was read during the pilot study of CNN’s. Further, its architecture model is widely available online, not only in terms of descriptions of the architecture but also implementations for popular libraries such as Keras and Tensorflow. Applying transfer-learning with VGG16, all layers but the last conv layer was frozen and was not trained. This decision was made due to the amount of time that would be required to retrain all the layers of such a deep network using limited hardware. Recall from the related works section that VGG16 has 138 million trainable parameters[46]. There is also a risk of overfitting as the 3 classes from AffectNet that were used was trained on ~25 000 images of each sample while VGG16 was originally trained on ImageNet, which as of writing contains 14 197 122 images and 1000 classes [31]. The network was trained using the initial ImageNet weights and only retraining the final conv layer (see figure 5.5). The last fully connected layer and SoftMax layer was swapped out from 1000 nodes to 3 because of the difference in classes to be predicted. None of the other hyperparameters were touched at the training session of VGG16 except the batch size and number of epochs. The number of epochs simply refers to how many times the entire training set is passed forward and backward (backpropagation) through the network12. The network was run for 100 epochs with a batch size of 64.

The second CNN architecture takes partial inspiration from VGG16, using only 3x3 convolutions but adding more dropout layers to reduce overfitting and decreasing the number of layers drastically. This architecture was trained on the same ~75 000 images of 3 classes as VGG16 over 100 epochs. VGG16 was trained before this architecture and taking notes from that training session and the overfitting occurrence after about 20 epochs, this architecture implements early stopping if the validation accuracy is not improved over 15 epochs. It also applies learning rate decay with a factor of 10% with regards to the quality of the validation accuracy each 5:th epoch if the learning rates plateau with an absolute min value of 0.0001. The final model takes an input of 150x150xn tensors and has 6 Conv layers using ReLU activation function and 3 max pooling layers. On top of these 9 layers, it implements batch normalization on the channel axis after each conv layer and dropout with a rate of 0.25 in combination with the max pooling operations. Dropout simply refers to dropping a few randomly selected nodes during training not to be trained. Finally, it has 2 fully connected layers, one with 512 as output and the other one with 3 just before the SoftMax classifier. Take note that the number of Dense (fully connected) layers have been greatly decreased, as well as the nodes in each of these layers. This is due to the great increase in learnable parameters that occurs

---

with the fully connected layers and that one of these layers activation is supposed to be used as input to other classifiers. The 512-node layer activation is used as input features for the ML algorithms. See figure 5.4 for a complete overview of the architecture. In the figure, the numbers displayed in each layer are input volumes, i.e. the next layers input is the previous layers numbers displayed.

The evaluation and testing were performed with 3 sets, FER2013 private test, FER2013 public test, and AffectNet test set\cite{11, 12}. The testing of the datasets was performed individually on each set 5 times for all the models and then the mean was evaluated from those 5 results as the final accuracy. The test process was only performed on the final model with the best accuracy. For the developed DL model seen in figure 5.6, there is also a smaller test involving real-time footage where ResNet is used as a face detector.

5.5. **Machine-learning classification**

The ML classification experiments were made using the features extracted from my own CNN architecture with 300 images of each category (happy, sad and neutral) from the AffectNet set. For testing of the algorithms another 30% of the images from the total 900 images were picked and used as input. The 900 sample images were first run through the face detector and cropped as in figure 5.3, then they were put through the CNN model and forward passed until the activation of the fully connected 512 layers. From this layer, the features were extracted into data frames with Pandas and paired with their respective label to be used to train the ML algorithms. All ML models were implemented in Python using Scikit-learn module. This allowed for fast implementations along with easy tuning of hyperparameters for all the models and some given defaults for some of the hyperparameters. If no other is stated, the default parameters for hyperparameters are used.

5.5.1. **Regression**

For regression, logistic regression was trained and evaluated on the CNN features. The logistic regression was evaluated using all three different solvers that support multinomial classification. Multinomial simply indicates that multi-class classification (i.e. training 3 separate logistic regression classifiers) using cross-entropy loss function is used. The three solvers used were sag, Newton-cg, and lbfgs. These three terms represent how the loss function will work, i.e minimizing the error rate of the classification. The three solvers used are all implemented using scikit learn in Python. The solvers have different characteristics and depending on the number of samples and how many samples you are using for the training of the logistic regression model, one should consider different solvers. For this problem, the feature data is very Dense and contains 512 features, where all the three solvers that have been mentioned are found to be faster for solving and finding the global minimum of the loss for such problems. They all build upon the classic learning technique of SG, short for stochastic descent or stochastic gradient descent\cite{54}.
The sag solver, short for stochastic average descent, where the iteration cost is independent of the number of terms in the sum. It does, however, store the values of previous gradients and can, therefore, converge faster than the classic stochastic gradient descent. One downside of this solver is that its memory cost is $O(n)$ which may make it infeasible for very large values of $n$. Since this thesis is working with a total of 900 samples for these algorithms, it is not something that will cause a problem[55].

The Newton-cg (conjugate gradient) solver also builds upon stochastic descent but utilizes the Hessian matrix (A square matrix of second-order partial derivatives of order $n \times n$, where $n$ is the number of variables in the problem). This algorithm utilizes the conjugate gradient method and uses quadratic approximation, which usually results in a better function minimization. Drawbacks of this are that it is more computationally taxing because it uses a a quadratic approximation. Again this is not a problem in this case with a fewer number of samples[56].

Finally, the lbfgs solver, short for Limited-memory Broyden–Fletcher–Goldfarb–Shanno is used by default in the implementation toolkit form scikit learn. It is a more memory friendly method of the bfgs algorithm which works in the same way but consumes more memory in the running process. Lbfgs also uses the Hessian matrix, although the inverse of it to help converge to the global minimum. However, instead of storing the full $n \times n$ inverse Hessian matrix of variables, it stores a few vectors that represent the approximation to conserve memory[57].

### 5.5.2. SVM

For SVM several kernels were tried to tackle the problem, starting with the simplest linear kernel then trying the polynomial kernel. The linear kernel and polynomial kernel work best if the data is linearly separable, so in this case, it might seem counter-intuitive to try it on a dataset which contains 512 features per sample. However, since the runtime of SVM is very fast using few samples, there was time to try the polynomial kernel as well. Kernels are essentially just a trick to fit the hyperplane to a higher dimensional space. What this essentially means is that one can find a higher dimensional space where the input features are linearly separable and the simply draw the hyperplane as accurate as possible. In the case of the linear kernel, it is as simple as it sounds, the kernel linearly separates the given features and is working as a second order or quadratic order kernel, unlike the linear kernel. The polynomial kernel is more fit for non-linear models and differs in a way that it does not only look at the given features but also looks at the combination of the features[58].

For both the kernels only a single hyperparameter must be taken into consideration, namely $C$. This is scalable float value and a low value of $C$ indicates that the decision surface should be smooth, hence making it less complex. While a large value of $C$ will make the kernel try to classify the input samples more correctly\(^{13}\). $C$ was tested at values 1.0, 10.0 and 100.0 respectively based on the guidelines from the scikit module.

There is another type of kernel based on the radial basis function (RBF) that can be used when there are very many training samples and can outperform the polynomial and linear kernels. However, since only 900 samples were used in this thesis work and the training time can be significantly longer, it will not be included[59]. Given more time this might have been included as well, as the RBF kernel has the potential to increase accuracy with more samples.

### 5.5.3. Decision tree and random forest

The decision tree implementation uses a binary decision tree with a max depth of 3 to control the depth of the tree and help prevent overfitting. The recommended random state of 100 from the toolkit scikit learn and a minimum of 5 samples per leaf, ensuring that multiple samples are used to get to a leaf node (endpoint) in the classification process. For the random forest classification, 100 decision trees were used in an ensemble, the same number of max depth and minimum samples per leaves were used. Several values were tested before arriving at the best results from 100 decision trees. Starting with low values at 10 and working up to 100, the accuracy curve stagnated at 100.

### 5.5.4. KNN

Using KNN, there is only a single hyperparameter to optimize, which is $K$. $K$ refers to the number of nearest “neighbors” of features to the actual feature that is to be classified. For example, with a value of $K=5$, the 5 closest points to the point that represents the feature to be classified will vote on what the output is. To make the search for the optimal $K$ value for the use of KNN for this problem, a grid search algorithm was applied as the KNN was fit to the data. Using this method, one can get the optimal $K$, saving time from trial and error testing for the optimal accuracy. There is nothing else to specify for the KNN as it is such a simple model.

5.5.5. **Naive Bayes**
The naive Bayes models used are Gaussian and multinomial. Because FER is based on categorical data, a standard multinomial naive Bayes algorithm should have the upper hand in this classification problem. However, since the implementation is simple, and the training speed is fast as it does not need to fit any coefficients by optimization procedures during training, there is space to try both methods out to compare them. Gaussian is unfit for the problem due to its standard application being on continuous values and not discrete values as in this case. For both the naive Bayes based classifiers no hyperparameters values were tweaked. Since the naïve Bayes classifiers are usually used for text and the results were not high, no further attempts were made to try increase accuracy with these[60].

5.6. **Real-time classification and detection**
For the real-time tests, the detection algorithms were tested together at the same time. The camera used for the test is an integrated web camera manufactured by Realtek on a Lenovo IdeaPad 330s laptop. Unfortunately, there are no further specifications to the web cameras hardware specifications. The algorithms were tested in different angles, illuminations and with different covers for the face to see how accurate their detection rate was. The illumination was measured in lux\(^{14}\) and it was tested in 3 different types of lux values for both detection algorithms. The lux metric was recorded with the Lux Light Meter application for android using a Google Pixel XL smartphone. For the haar-classifier a scale factor of 1.2 was used, essentially meaning that it has the capability of expanding the boundaries when looking for faces in the images, resulting in a larger range for the detections. For the DL detection, a confidence level of 0.9 was used for the bounding box to appear. On top of accuracy in detections, there was also a benchmark test to determine how fast the algorithms perform in terms of seconds and fps on 120 recorded frames. When the detection algorithms had been evaluated, the DL model was used to do live classification with the CNN trained with AffectNet. In this test, the benchmarks for seconds and fps on 120 frames were used as well. The illumination in lux was not recorded for the live classification test, only the detection comparison.

\(^{14}\) Measurement of illumination - equal to one lumen per square metre
6. Results

In this section, the results of the training and testing of all the algorithms are presented. In the face detection comparison between the Viola-Jones algorithm and ResNet, the metrics are given in runtime (FPS), the detection rate for live detections (profile face, 45-degree face, tilted face, and different illumination) and amount of detections given when detecting and extracting faces in the AffectNet dataset.

The final CNN model used for feature extraction results contains information derived from the training, such as loss, learning process information, and performance on different datasets. Some of these metrics are also given for the VGG16 model that was tried before taking the decision to use the final model for the extraction and classification process.

ML results are given from \( K \)-fold cross-validations and confusion matrices where the best results are conferred. The \( K \) in the cross-validation is set to 5 for all the ML tests. The confusion matrices are provided to see not only the accuracy of the model, but also more information regarding their misclassifications. As mentioned in the Methods and materials section the input for these algorithms is 900 samples derived from running images from AffectNet through the final CNN model and extracting 512 floating point values from each image. The procedure of the cross-validation then used on these samples is as follows:

1. Shuffle the 900 samples randomly
2. Split the samples into 5 groups
3. For every group:
   a. Take the group as a holdout or test data set
   b. Take the remaining groups and use it as a training set
   c. Fit an ML model on the training set and evaluate it on the test set
   d. Record the score and discard the model
4. Summarize the skill of the model using the given scores

6.1. Face detection

In the results derived from the detection algorithms, two main properties were considered, speed and, detection rate. Speed was tested on a Lenovo IdeaPad 330s laptop using the built-in web camera (as mentioned in the methods and materials section) for the haar-cascade based Viola-Jones algorithm and the deep-learning based algorithm pre-trained with ResNet. The cascade features and the ResNet weights were provided from OpenCV and GitHub. The same laptop web camera was used in section 6.4 for real-time detection.

When using the classifiers for live detection, there was no noticeable slowdown to the eye when using the DL approach compared to the haar-cascade based approach, as they only differed by about 2 fps, however, the DL approach appears to be faster in this case. One must consider that the Viola-Jones algorithm only performs detection on grayscale images, therefore, the image converting step is included in the time taken to perform the detection.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>DL(ResNet)</th>
<th>Viola-Jones</th>
</tr>
</thead>
<tbody>
<tr>
<td>Captured Frames</td>
<td>120</td>
<td>120</td>
</tr>
<tr>
<td>Time</td>
<td>6.3237273</td>
<td>7.1407494</td>
</tr>
<tr>
<td>Estimated FPS</td>
<td>18.976150</td>
<td>16.804958</td>
</tr>
</tbody>
</table>

Table 6.1 - Detection algorithms benchmark

Both algorithms failed some detections when in the extraction of the ROI process when preparing the images for input to the CNN. The DL algorithm that was used for the final extraction of the ROIs failed about detection in about 3000 images from the 75 000 total images given from the dataset. See figure 6.1 for two examples on the left from ResNet failed detections and two examples from the right from the Viola-Jones algorithms failed detections.
6.2. CNN

6.2.1. VGG16
VGG16 was trained once on the AffectNet training set with all the layers and achieved a final accuracy of ~50% over 100 epochs without the ImageNet weights as initial weights. After freezing all the layers except the final fully connected layers and initializing all of them with the pre-existing ImageNet weights the accuracy rose from ~50% to ~61%. A test was conducted with VGG16, training the whole network from scratch and letting it run for 100 epochs without early stopping, which resulted in a result of close to ~97% and massive overfitting where the validation accuracy peaked at about ~75%. In figure 6.2 the loss and accuracy for the VGG16 can be observed, where the orange curves in the graph represent the validation set and the blue curves are the training.

![Figure 6.2 - Overfitting occurrence of VGG16 trained on AffectNet](image)

6.2.2. Final model
The final CNN model reached an accuracy of 81.67% on the training/validation set and underwent early stopping after running for 80 epochs. The training process took about 8-9 hours where each step took ~359 ms and approximately 1100 steps (one step = one image) were taken in each epoch. This process can be inspected in figure 6.3. On the test sets, the model performed strongest on the AffectNet test set (1410 images) and performed below training/validation accuracy on the FER2013 test sets (2155 and 2099 images). The average from 5 runs on each of the test sets are displayed in table 6.2 and a confusion matrix from the best run on AffectNet can be seen in figure 6.4. See the appendix for the confusion matrices from the FER2013 results.

<table>
<thead>
<tr>
<th>Test set</th>
<th>AffectNet</th>
<th>FER2013 Public</th>
<th>FER2013 Private</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total mean acc</td>
<td>0.8132</td>
<td>0.7204</td>
<td>0.7414</td>
</tr>
<tr>
<td>Accuracy Best</td>
<td>0.8216</td>
<td>0.7560</td>
<td>0.8031</td>
</tr>
<tr>
<td>Misclassified Best</td>
<td>250</td>
<td>640</td>
<td>51</td>
</tr>
</tbody>
</table>

*Table 6.2 - Test accuracy of CNN on test sets*
6.3. ML classification

The results from all the ML algorithms are presented here, the evaluation was made using K-fold cross-validation, then the mean from this confirmed the result of the best algorithm. The best result and the results from the K-fold cross-validation are both presented separately for each algorithm as well as confusion matrices (cm) from the best results. All the ML algorithms are run using a Desktop computer with an Intel Core i5-4670K CPU with 4 3.40Ghz cores and 16GB physical memory.

6.3.1. Logistic regression

The logistic regression models’ results are simple and straightforward, the results of the different solvers are displayed in table 6.3 and following in figure 6.5 for the sag solver (see appendix for the other solvers CMs). The confusion matrices are derived from the best training sessions and are correlated to the “Accuracy best” and the “Misclassified” rows in the table. Due to warning feedback given by the Sag solver regarding the maximum amount of iterations to reach convergence, the max_iter was set to 5000 for this solver. This warning is given when the algorithm is not run for enough iterations to converge. There was no impact in terms of accuracy or misclassified examples when raising this, but it removed the warning, meaning that the solver now converges. One note was that Sag solver takes considerably longer to run even with such few training samples. The first two solvers (Lbfgs and newton-cg) take 1< seconds and the sag solver close to 4 seconds in runtime.
6.3.2. SVM

With the SVM based models, there were problems when giving a lot more samples, which is one of the reasons why the training samples were limited to only 900 in total for the ML classification tests. When choosing 1000 images of each class (i.e. 3000 images) it would run endlessly but with 900 images, it finishes in a couple of seconds. The results can be observed in table 6.4 for the different kernels and the best runs result using the polynomial kernel can be seen in figure 6.6.

<table>
<thead>
<tr>
<th>Kernel</th>
<th>Linear</th>
<th>Poly</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy K-fold</td>
<td>0.7431</td>
<td>0.7381</td>
</tr>
<tr>
<td>Accuracy Best</td>
<td>0.7826</td>
<td>0.8024</td>
</tr>
<tr>
<td>Misclassified Best</td>
<td>55</td>
<td>50</td>
</tr>
</tbody>
</table>

Table 6.4 - SVM kernels
6.3.3. **Random forest ensemble**

Using the random forest classifier, many different $N$ estimators (decision trees) were tested to see if accuracy improved. After going above $N=30$ estimators the accuracy momentum decreases quickly but keeps improving at a low rate and stops at around $N=100$. The final hyperparameter used for $N$ was 100 and the confusion matrix for this result can be observed in figure 6.7. After running several k-fold cross-validations, the algorithm seems to be able to make high classifications using lower numbers of trees as well (see table 6.5). It gave quite varying numbers depending on how many times you run it.

<table>
<thead>
<tr>
<th>N estimators</th>
<th>30</th>
<th>65</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy K-fold</td>
<td>0.7974</td>
<td>0.7907</td>
<td>0.8009</td>
</tr>
<tr>
<td>Accuracy Best</td>
<td>0.8340</td>
<td>0.8261</td>
<td>0.8379</td>
</tr>
<tr>
<td>Misclassified Best</td>
<td>42</td>
<td>44</td>
<td>41</td>
</tr>
</tbody>
</table>

*Table 6.5 - Random forest*

*Figure 6.7 - Random Forest using 100 N estimators CM*
6.3.4. Decision Tree
The decision tree results were straightforward and did not diverge with running the algorithm several times. A single tree was tested with different values of min depth and max depth as described in the method. The results were essentially the same, differentiating by less than 1% when using default or close to default values. When applying high values of max depth and leaves, the accuracy is decreasing slow but steady. See the results given from the decision trees below in table 6.6 and figure 6.8.

<table>
<thead>
<tr>
<th>Max depth, min samples of leaves</th>
<th>3, 10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy K-fold</td>
<td>0.7549</td>
</tr>
<tr>
<td>Accuracy Best</td>
<td>0.7826</td>
</tr>
<tr>
<td>Misclassified Best</td>
<td>55</td>
</tr>
</tbody>
</table>

Table 6.6 - Decision tree classifier

![Confusion matrix for decision tree](image)

Figure 6.8 - Decision tree CM

6.3.5. KNN
For the KNN the best $K$ neighbors derived from the grid search was 24 and this was used to perform the k-fold cross-validation. See table 6.7 and figure 6.9.

<table>
<thead>
<tr>
<th>N estimators</th>
<th>30</th>
<th>65</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy K-fold</td>
<td>0.7974</td>
<td>0.7907</td>
<td>0.8009</td>
</tr>
<tr>
<td>Accuracy Best</td>
<td>0.8340</td>
<td>0.8261</td>
<td>0.8379</td>
</tr>
<tr>
<td>Misclassified Best</td>
<td>42</td>
<td>44</td>
<td>41</td>
</tr>
</tbody>
</table>

Table 6.7 - KNN
6.3.6. Naive Bayes

The naive Bayes results are given from a Gaussian model and a Multinomial model. The multinomial model showed robust results in the k-fold cross-validation. See table 6.8 and figure 6.10 for these results.

<table>
<thead>
<tr>
<th>Model</th>
<th>Gaussian</th>
<th>Multinomial</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy K-fold</td>
<td>0.7770</td>
<td>0.8111</td>
</tr>
<tr>
<td>Accuracy Best</td>
<td>0.7470</td>
<td>0.8194</td>
</tr>
<tr>
<td>Misclassified Best</td>
<td>64</td>
<td>48</td>
</tr>
</tbody>
</table>

Table 6.8 - Naive Bayes

6.4. Real-time classification and detection

Using the same metrics as for the comparison between the detection algorithms previously in this section, the results from the real-time test with classification using ResNet and the CNN model can be observed in table 6.9 and figure 6.11. In figure 6.12 the detection tests between the Viola-Jones and ResNet are displayed. The left
Vision based facial expression detection using deep convolutional neural networks

images (using red bounding box) displays the ResNet model and the right images (using green bounding box) displays the Viola-Jones detections. See appendix for more images from the test session.

<table>
<thead>
<tr>
<th>Model</th>
<th>ResNet + CNN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Captured Frames</td>
<td>120</td>
</tr>
<tr>
<td>Time</td>
<td>7.5900</td>
</tr>
<tr>
<td>Estimated FPS</td>
<td>15,8101</td>
</tr>
</tbody>
</table>

Table 6.9 - Real-time classification

In the test with detection in different illuminating settings, all the images were taken in a full-frontal position in at night (to reduce interference from natural sunlight). A simple desk lamp and the computer screen was used to achieve different illumination settings. The measurement recorded was lux and the results can be observed in table 6.10 where the table shows if detection was possible in different values of lux. Images from the session can be seen in figure 6.13.
Fredrik Julin Vision based facial expression detection using deep convolutional neural networks

<table>
<thead>
<tr>
<th>Model</th>
<th>ResNet</th>
<th>Viola-Jones</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lux 1</td>
<td>Fail</td>
<td>Success</td>
</tr>
<tr>
<td>Lux 5</td>
<td>Success</td>
<td>Success</td>
</tr>
<tr>
<td>Lux 46</td>
<td>Success</td>
<td>Success</td>
</tr>
</tbody>
</table>

*Table 6.10 – Detection algorithms illumination benchmark*

*Figure 6.13 - Real-time detection illumination test, from the top 46 lux down to 1*
7. Discussion

In this section of the report conclusions about the results regarding the state-of-the-art and related works are discussed. Furthermore, potential future work and discussion about improvements or extensions of the work are suggested.

7.1. Conclusions

7.1.1. Detection

Regarding the first question in the problem formulation, the results from the detection is here discussed. The ResNet based approach showed a good overall performance in both speed and accuracy in the ROI extraction process and the live detection and classification test. The algorithm also gave a very high detection rate when extracting images into folders in preparation for the training sessions, also considering that some of the images did not have faces in them. This high accuracy is mainly due to its ability to detect faces in angles as the AffectNet set contains varied poses of the subjects[12]. The choice of using ResNet for this was due to the easy accessibility of the weights and for its low number of parameters compared to other state-of-the-art models[44, 46].

The results in comparison to the Viola-Jones based approach in the real-time detection test further gave some interesting results. The higher accuracy for the DL algorithm in terms of detections in 45-degree angles, profile, tilted and covering of the face was expected. However, in terms of illumination. the Viola-Jones algorithm outperforms the DL algorithm classifier, as the DL algorithm has trouble spotting faces in images with darker pixel values. One reason for this could be that the pre-trained weights that were downloaded for ResNet simply has not been exposed to darker images containing faces in the original training set. It should also be noted that the detection threshold for ResNet to output the bounding box to the face was a 0.9 in the given prediction. Lowering this threshold value could potentially make the CNN approach using ResNet to detect faces in darker illumination as well and perhaps improve even more on faces in other angles. There were also very bright images that could occur in, for example, broad daylight since the experiment was conducted indoor with artificial light from a lamp and a computer screen, one cannot say how very bright illumination might affect the detection rate of the algorithms.

In terms of speed performance, the DL algorithm was faster than the Viola-Jones algorithm, even if only with a few fps over 120 frames. This result was quite unexpected as the Viola-Jones algorithm is praised for its speed and low computational cost. Now, it should be mentioned that the script running the Viola-Jones and DL algorithms detectors are performing the necessary pre-processing operations upon the frames that are given from the web camera during runtime. This step includes image resizing for the DL algorithm and grayscale conversion for the Viola-Jones algorithm. However, this seems like a reasonable item to consider when performing the real-time classifications as most cameras today output colored images.

7.1.2. CNN’s

This section mainly covers the CNN’s performance and includes the discussion of the results regarding the third research question. VGG16 achieved a very poor performance in comparison to state-of-the-art results covered in the related work section. One of the reasons this might be is that the dataset domain is very different from what the ImageNet looks like. The ImageNet dataset contains many classes of animals and different objects but not many humans or faces of humans. And if a human is present in some of the pictures by chance, human and/or face are not present as a valid classification option for the set. The second speculation is that the ImageNet dataset contains a lot more images from each class, meaning that it gives more variation, which in turn means that there will be more features to be learned about each object. For a dataset much smaller, as AffectNet, there is the risk of learning too much about the training sample, therefore overfitting the set to the model.

Instead of starting trial and error from scratch with some new convolutional strides, padding, layers and so forth, the developed model took inspiration from VGG16. As it is a straightforward architecture and has proven to be effective for classification tasks in the past. Because overfitting was a big concern, decreasing the complexity of the new model and taking preventive steps against overfitting seemed like a good idea. With the first attempt for training, the model peaked at about 40 epochs and terminated training early because it was overfitting again. By tweaking the model complexity, adding and removing pooling and convolutional layers systematically and adding batch normalization and dropout, the model was able to learn for a greater number of epochs and increasing its accuracy step by step. Some of the final hyperparameters that were tweaked to try to reach an accuracy of the state-of-the-art models were momentum and learning rate decay. The purpose for this was to decrease the training time by letting the model start learning fast and slow down as it hits certain thresholds in the loss function and let the model try to reach the minimum value of the loss function without overshooting when close to plateauing.
The result of about 81% is not enough to compete with state-of-the-art architectures, however, it is not a complete catastrophe either. For many of the state-of-the-art models, there are 3 more classes in their datasets, forming the 6 universal emotions defined by Ekman et al., while my model uses only 3 of these classes [61]. It is hard to speculate beforehand how the accuracy might be affected by bringing in more classes because it also depends greatly on the quality, quantity and properties of the datasets that are used. For example, many of the state-of-the-art models that have been presented have been trained on datasets where images are from controlled environments [37, 47].

There are some interesting takeaways from the results of the confusion matrix given from the training with the model. Between the classes happy and sad, the model is very strong in its classification capabilities. But between sad and neutral it has a harder time, especially when classifying the sad images. This is probably due to more similar features learned between sad and neutral, than for example between sad and happy. Introducing more classes could both benefit and make this process harder, by giving more examples of what classifies as not sad for example it should become stronger in that specific classification. However, if the new classes introduce a lot of similar features with e.g. the class sad, it might impact the classification accuracy negatively. The features extracted from what the CNN learned will have a direct impact on other algorithms as well, as we will see in the next section.

For the real-time purposes, the classification and and detection using CNN’s showed promising results. When applying classification on top of the face detection in real-time, an FPS drop of only about 3 was given as a result. Given these results, it would seem like the detection is the more computationally intensive operation in real-time. Of course, this also depends on the complexity of the classification model, but in this case a drop of 3 FPS from a total of ~19 to ~16 is deemed as acceptable. For greater FPS values, one would hardly notice the slowdown in terms of speed.

7.1.3. Classification and ML algorithms

To summarize the ML algorithms performance and answer the second question asked, the classification process using ML algorithms is discussed here. The dataset for all the ML models was limited to 900 images because of attempts to run the SVM models with higher than a certain number of images, they would run endlessly. The SVM models seemed to have immense trouble to fit the hyperplane to the problem as with for example 1000 images of each class (i.e. 3000 images) it would run endlessly, but with 900 images, it would finish in a couple of seconds.

Some of the ML algorithms did not deviate too far from the SoftMax classifier originally used with the CNN to make predictions on the dataset, while others did not reach the same amount of accuracy. Why this happened It was expected that some of the linear models would not fare as well as other models due to the size of each feature sample. One pleasant surprise was the random forest ensemble, which performed slightly below the SoftMax classifier in the k-fold cross validation score but had a higher top accuracy of almost 2%. The KNN also performed above the SoftMax classifier with the top result and the multinomial naive Bayes algorithm was very close as well. The two most robust algorithms in terms of the k-fold cross validation were naive Bayes and the KNN algorithms. The simpler models seem to benefit more from the extracted features from the CNN than the more complicated ones such as SVM.

Some of the ML algorithms gave an accuracy of equal to or better than the SoftMax classifier of the CNN in some of the cases, given a very little amount of training data, less than 300 images per sample. Unfortunately, there has not been enough time to conduct an experiment on how well these algorithms would fare on manually extracted features from the AffectNet dataset compared to how they did on the CNN extracted features. If the accuracy would prove to be higher, it could prove highly beneficial in both terms of accuracy and time to download pre-trained high accuracy weights of a specific problem and extract known features to use for other models. Another way to experiment with the accuracy of the models would have been to have another fully connected layer in the CNN that reduces the number of features to a smaller amount to extract from.

7.1.4. Summary

Deep learning shows great promise in both terms of detection and classification in the FER pipeline. With more quantity and quality for training data, one can reap more benefits in dealing with FER as images are arguably the most feature-rich form of data for FER. Even though transfer learning is a great first choice when trying predictions on a new problem as it saves time and the development of a model, it should be approached with caution if the problem domain is very different from the original. For the real-time detection results, it is clear to say that DL is
also feasible when making real-time detections and predictions with CNN’s. Machine learning models can benefit in terms of accuracy from the features extracted from CNN’s to a degree, depending on the model.

Even if the purposed CNN’s accuracy did not achieve state-of-the-art accuracy it could successfully be used as a feature extractor for the set of machine learning algorithms classifications and these features proved to be very useful, especially for the cases of simple algorithms. These simple algorithms have the benefit of low training cost and still maintaining high accuracy that is useful when the dataset is small, or the computational resources are limited. While evaluating the deep learning models for real-time classification with the purpose of determining their practical value some impressive results were achieved. A laptop with an integrated web camera managed to run the full FER process with detection, image pre-processing and CNN classification in real-time at an acceptable fps. These results show that a desktop running single or multiple GPU’s are not required to run real-time classification.

7.2. Future work

FER is a problem formed from a union of many other problems that together form a FER system pipeline, including but not limited to, image preprocessing, computer vision and AI. In this work, the goal was to achieve solutions and/or partial solutions to the subproblems described back in the problem formulation in section 4. Improvement in any of these subtasks would benefit the study of FER, as all the parts impact the overall practice in the art. In the pursuit of this, some of the results prove promising, giving some new insight in not that well-studied areas of interest, while other subtasks failed. The CNN accuracy did not achieve what is considered state-of-the-art accuracy (see related works) and was performed on a smaller set of classes than most of the state-of-the-art algorithms. Due to the time constraint and the lack of manpower and hardware, more experimentation with hyperparameters, architectures and different datasets were limited and would have been beneficial.

For future studies, more data would always be beneficial, not only to the classification process but also the detection stage. As mentioned previously, all deep learning models benefit from a higher quantity of quality data. To use more GPUs and more powerful hardware is also a great way to improve the training time of algorithms, hence giving more time for the trial and error algorithms, getting more feedback from the training sessions on what needs tuning. All machine learning models evaluated in this thesis work benefits from higher accuracy of CNN’s when performing the feature extraction, some more than others. Without a doubt, better feature extraction would mean better classification capabilities for all the tested machine learning models. Finally, given the constant increase in computational power, the great work of data gathering from the computer vision community and the recent increase in popularity and accuracy of CNN’s, FER will very likely have many real-time application areas in the future.
8. References


9. Appendix

In this section all figures referenced in the text can be found.

*Figure 9.1 - Down sampling using max pooling operation in a CNN layer*

*Figure 9.2 - Relation between SoftMax regression and logistic regression*
Figure 9.3 - Linear regression between dependent variable weight(y) and independent variable height(x)

Figure 9.4 - Logistic regression with sigmoid function applied between passing an exam and hours studied

Figure 9.5 - KNN using K=3 to determine X
Fredrik Julin Vision based facial expression detection using deep convolutional neural networks

Figure 9.6 - Real-time detections ResNet vs Viola-Jones

Figure 9.7 - FER2013 public test set result CM
Fredrik Julin  
Vision based facial expression detection using deep convolutional neural networks

Figure 9.8 - FER2013 private test set result CM