Recognizing Spontaneous Facial Expressions using Deep Convolutional Neural Networks

Andreas Jönsson, Erik Söderberg

Master's thesis
2018:E25

LUND UNIVERSITY

Faculty of Engineering
Centre for Mathematical Sciences
Mathematics

CENTRUM SCIENTIARUM MATHEMATICARUM

Lund University

Faculty of Engineering

Centre for Mathematical Sciences

Master's thesis

Recognizing Spontaneous Facial Expressions using Deep Convolutional Neural Networks

Andreas Jönsson
andreasjnsson@gmail.com

Erik Söderberg
eriksoderberg@live.se

Supervisor
Kalle Åström
kalle@maths.lth.se

Assisting supervisor
Martin Ahrnbom
ahrnbom@maths.lth.se

Assisting supervisor
Siri Dovner
siri.dovner@axis.com

May 30, 2018

Abstract

Emotion recognition is a relatively new research area within image analysis where machine learning models learn to interpret facial expressions. In this thesis paper the authors have investigated how close to human accuracy selected deep convolutional neural network models can reach on the Emotion Recognition Problem, what the purpose of such an application might be and what compromises can be made when balancing computational power and classification performance.

A number of deep convolutional neural network models of varying depth were trained on mainly two datasets; FER2013 and AffectNet. Results showed that the best model managed to reach 74.6 % accuracy on a three label testset, where a human performed 80.7 %. The conclusion was made that for the most interesting use cases, label binning facial expressions into three classes *positive*, *neutral* and *negative* might be favorable. Lastly, as for trade-off when balancing accuracy and computational power, deeper networks perform insignificantly better than the more shallow ones, while the latter dramatically reduce the computational effort.

Acknowledgments

We would first of all like to thank our supervisors Kalle Åström, Martin Ahrnbom and Siri Dovner for all the guidance and input. Additionally we would like to thank Håkan Ardö who has been a source of knowledge for us, both for theoretical and practical problems. Finally we would like to thank *Axis Communications* for the thesis opportunity and the resources we have been able to use.

Contents

Glossary

CNN Abbreviation of convolutional neural network. 12

dropout The process of randomly dropping neurons at training time in a neural network, given dropout probabilites. 11

dynamic range The ratio between the largest and smallest value of some quantity, in this case pixel values. 18

epoch The timescale in which a machine learning model has trained on all components in a dataset. 7

facial landmarks Coordinates for data points of features in an image, for example coordinates for a mouth or an eye. 3

gate An unary, atomic mathematical operation, i.e. it has one input and cannot be reduced into a combination of other operations. 10

hyperparameter A parameter whose value is set before the learning process begins. 24

in-the-wild Images from various sources which are captured in a non-staged laboratory environment. 3

loss function A differentiable function of trainable parameters outputting a value representing the error of a model. 6

neocognitron The predecessor of the convolutional neural network. 12

neuron An atomic part of the neural network outputting a scalar value after operations on input value(s). 7

overfitting A model is considered overfitted when it has adapted to its training data in too much detail. 6

padding The process of adding a border of zeros around an input. 12

pooling The process of down-sampling a layer in a CNN. 12

saddle point A point on a graph surface where the gradient is zero but the point cannot be the same kind of local extremum on both axes. 25

1 Introduction

Although the precursor to the neural network was theorized as early as in 1943 by Warren McCulloch and Walter Pitts [1], it is in recent years with the advent of powerful GPU's that these systems have become popular. One of the fields where neural networks have shown great promise is the field of computer vision, where in particular convolutional neural networks are useful for solving a number of different problems.

One of the potential areas where convolutional neural networks can be utilized is for recognizing facial expressions in images of faces, more commonly known as *emotion recognition*. The possible applications for recognizing facial expressions are many: it could be used as a direct tool to evaluate a persons mood or to detect responses to different kinds of stimuli. Some of the more evident areas of use would be within marketing and surveillance, but it is not unthinkable that emotion recognition would be of high use for synergy between man and machine.

Like most machine learning problems, the results of neural network models rely heavily on the amount and quality of training data. For the emotion recognition task most datasets have six categories, the six basic facial expressions defined by Ekman in 1971: happiness, sadness, anger, surprise, disgust and fear [2]. On top of these six basic emotions a few others also commonly appear, such as neutral and contempt. By extension this means that most previous work on emotion recognition using neural networks have tried to classify facial expressions based on these emotions.

Our approach to the classification task is slightly different. What if the interesting case is not to classify these six different emotions (or up to eight depending on which dataset you use), but instead to classify using broader terms such as *positive, neutral* or *negative* response? One could argue that a useful use case would not be whether a computer could interpret a very specific facial expression such as 'disgust', but rather whether a person is satisfied or not.

Since we have chosen to focus on the application and usefulness of the emotion recognition problem rather than trying to produce new competitive model designs, we focus on testing models designed by others. We have namely noticed that despite abundant research on emotion recognition, the usefulness *in-the-wild* is not well understood, which we want to investigate.

2 Background

Neither the classification task nor the model designs in this project are unique. The fundamental convolutional layer was first introduced in 1988 by Y. Lecun et al. [3] and the earliest efforts at emotion recognition which we can find are by I. Cohen et al. in 2002 [4]. All research on emotion recognition we have come across is evaluated on a specific test set with no efforts of applying the classification task on a real life problem, which is is why we have chosen not to try to produce a top of the line classifier, but rather to use existing ones to evaluate their performance and usefulness in a real life setting.

3 Related work

In this report we have focused on mainly three different papers for choice of neural network models; *Facial Expression Recognition Using Deep Convolutional Neural Networks* by Sang et al. [5], *Densely Connected Convolutional Networks* by Huang et al. [6] and *Deep Residual Learning for Image Recognition* by He et al. [7]. Out of these three only the first article by Sang et al. focuses on the emotion recognition problem, while the other two are focused on object detection.

Earlier work utilizing machine learning has tried to solve the emotion recognition problem in a number of ways. Perhaps the most straight forward approach, which has been used by Sang et al. [5], is to simply analyze every image individually with a convolutional neural network and in that way predict the facial expressions. Other researchers have focused on analyzing a series of images in a video sequence utilizing time-delay neural networks and recurrent neural networks [8] [9]. The analysis is in most cases applied to images of faces displaying facial expressions, but sometimes it is also used in conjunction with *facial landmarks*, which are coordinates for facial features in an image, such as for example the mouth, the eyebrows or the eyes.

Our approach to the problem has been to focus on simple convolutional neural networks on individual

images instead of video sequences. The reason for this stems from three arguments: (1) datasets which contains facial expressions in video sequences are rather limited, there is far more available data on single images, (2) we want to create a neural network model which does not require too much computational power, since the area of use might be to embed the software on a network camera, and (3) when trying to analyze facial expressions in a video stream it is hard to distinguish when a facial expression starts and when one ends. If you still have to do a single image analysis to determine this, then you might as well try to create a classifier which is good at classifying single images.

4 Problem definition

The main questions we are trying to answer in this thesis work are:

1. Using selected deep convolutional neural network models; how close to human performance can we reach on the emotion recognition problem?
2. What real life applications could arise from the emotion recognition classification task?
   (a) What is the purpose of the application?
   (b) What are the trade-offs that can be made between prediction accuracy and execution speed?

5 Theory

5.1 The image classification problem

5.1.1 Goal

The goal of the image classification problem is to, given an image (in our case an image of a face), classify it i.e. providing it with a single label from a known set of labels (in our case the labels are different emotions).

5.1.2 Approach

One way of solving the image classification problem would be to somehow find patterns among labels and hard code some way of detecting these patterns. A simple example would be to use pixel value gradients to find eyes and then for example use eye distance and proportions to classify between different animals. This kind of approach proves to be a tricky task and the addition of a new label would result in a lot of work. A more simple approach, made possible by the computational power of the modern day computer, is to train a classifier much like the way you teach a child what different things are. With a child you point at a horse and say "horse". With a computer you could input a 3-dimensional array of integers representing an image of a horse together with a label corresponding to "horse". This way of tackling the image classification problem is referred to as a *data-driven approach*, since it requires the collection of already labeled images - a training set.

To evaluate the performance of the trained classifier, predictions are made on an already labeled set of images. Predictions and true labels are then compared to obtain a final score. The latter set of images is usually called a test set and cannot contain images from the training set. This because such an evaluation wouldn't test the classifiers performance to see general patterns in labels, but rather its ability to remember exact images. An analogy would be if you took a math test where you've seen all the answers beforehand and thus does not need to derive them. Your ability to remember the answer to a problem is therefore tested rather than your ability to solve an arbitrary problem of a given kind.

5.1.3 Challenges

There are a lot of factors affecting the performance of a classifier. The way the classifier works and whether it is optimal for a certain data set is very important. A good training set is however fundamental to high classification performance regardless of classifier.

A good training set should mirror the reality of the classification task at hand as well as possible. If you imagine a master set, which consists of every possible image displaying a classification label, then a good training set is a subset of this master set, as big as possible and with as high diversity as possible.

Even with an optimal classifier and a great training set, the classification performance might be inadequate. This could be caused by one of the following image problems:

- **Perspective**. The same motif looking different depending on the camera angle.
- **Scale variation**. A class of objects showing big differences in size.
- **Deformation**. Non rigid bodies exhibiting different levels of deformation.
- **Occlusion**. The object of interest being blocked in some way.
- **Light conditions**. Images being under- or overexposed dramatically differing in pixel values.
- **Camouflage**. The object of interest blending into the background.
- **Label diversity**. Instances of the same label differing in looks.

5.2 Linear Classification

In order to understand how a neural network works, one needs to be familiar with the basic concepts of linear classification. Therefore, a short theoretical background of linear classification follows, before moving on to the neural network architecture.

5.2.1 Score function

When predicting the label of an image, a linear classifier outputs a class score for every possible label. The highest class score represent the most likely label. A simple linear class score function can be defined as

$$f(\boldsymbol{x_i}, \boldsymbol{W}, \boldsymbol{b}) = \boldsymbol{W}\boldsymbol{x_i} + \boldsymbol{b} = \boldsymbol{s}, \tag{1}$$

where $\boldsymbol{x_i}$ is a n x 1 vector of input data, $\boldsymbol{W}$ a m x n matrix of weights and $\boldsymbol{b}$ a m x 1 vector of biases. The letter n corresponds to the amount of data and m to the number of classes. Usually the pixel matrices of an image are flattened to a single dimension vector, in this case $\boldsymbol{x_i}$ of length n. The class score vector $\boldsymbol{s}$ is a m x 1 column vector representing a metric of how well the input data matches the m labels. Simply put, the index of the highest value in the class score vector is the predicted label in the classifiers current state.

5.2.2 Loss function

Now that we have a way of predicting labels, we can also measure the classification performance. This is usually done using a *loss function*. The "loss" represents the error of the classification and should thus be minimized. The function defining the loss is usually referred to as the loss function, cost function or objective function. Two common loss functions for this kind of classification are the Multiclass Support Vector Machine (Multiclass SVM), also known as multiclass hinge loss or max margin loss, and the Cross Entropy loss function. They both use the class scores from (1) to define a loss, but with slightly different approaches.

5.2.2.1 Multiclass SVM loss function

The Multiclass SVM loss function is defined as

$$L_i = \sum_{j \neq y_i} max(0, s_j - s_{y_i} + \Delta), \tag{2}$$

where L_i is the loss for the i:th data example, j:s domain is the class labels, the s terms are the class scores from (1) and y_i is the correct class label for example i. The Δ term is a threshold for when to penalize class scores. If an incorrect class score is more than Δ smaller than the correct class score, then there is no penalty. Otherwise there is penalization proportional to the difference between the correct and incorrect class scores. Let's look at an example using $\Delta = 10$ and $y_i = 1$;

$$\boldsymbol{x_i} = \begin{bmatrix} -1 \\ 1 \\ 3 \end{bmatrix}, \quad \boldsymbol{W} = \begin{bmatrix} 1 & -2 & 3 \\ -2 & -1 & 4 \\ 1 & 2 & 3 \end{bmatrix}, \quad \boldsymbol{b} = \begin{bmatrix} 0 \\ 1 \\ -1 \end{bmatrix}, \tag{3}$$

$$\left.\begin{matrix} (3) \\ (1) \end{matrix}\right\} \implies \boldsymbol{s} = \begin{bmatrix} 6 \\ 14 \\ 9 \end{bmatrix}, \tag{4}$$

$$\left.\begin{matrix} (2) \\ (4) \end{matrix}\right\} \implies L_i = (6-14+10)+(9-14+10) = 7. \tag{5}$$

We can see that even though the correct class score is the highest, i.e. this sample is predicted correctly, the loss is nonzero. This is because a relatively high threshold $\Delta = 10$ is used.

To get a final loss function for the whole dataset, the losses L_i are summed up. A regularization term is usually added as well to prevent *overfitting*. An overfitted model has adapted to its training set in too much detail, instead of capturing more general patterns of the labels. To avoid this, high value weights are penalized, promoting a more balanced weight distribution and thus a less extreme fit. A common regularizer is the L2 regularizer which uses the L2 norm to penalize the weights. The total loss L can be expressed as

$$L = \frac{1}{N}\left(\sum_{i}^{N}\sum_{j \neq y_i}^{M} max(0, s_j - s_{y_i} + \Delta) + \lambda \sum_{i}^{K}\sum_{j}^{M} w_{i,j}^2\right), \tag{6}$$

where N is the size of the dataset, λ the regularization weight, K the number of data values per example, M the number of labels and w the corresponding weights.

5.2.2.2 Cross Entropy loss function

The Cross Entropy loss function is minimizing the cross entropy between the estimated distribution q

and the true distribution p. In our case, for example i, these distributions can be defined as

$$q: \quad P(X = x) = \frac{e^{s_x}}{\sum_j^M e^{s_j}} \tag{7}$$

and

$$p: \quad P(X = y_i) = 1, \tag{8}$$

where X is a discrete random variable representing the label. So if $X = 1$ then $s_x = s_1$ is the score function for label 1. The cross entropy between an estimated distribution q and true distribution p is defined as

$$H(p, q) = -\sum_x p(x) \log q(x). \tag{9}$$

Inserting (7) and (8) into (9) gives us

$$H(p, q) = -\log \frac{e^{s_{y_i}}}{\sum_j^M e^{s_j}} = L_i \tag{10}$$

which is our loss L_i for example i. A total loss L including a regularization term can be defined as

$$L = -\frac{1}{N} \sum_i^N \log \frac{e^{s_{y_i}}}{\sum_j^M e^{s_j}} + \lambda \sum_i^K \sum_j^M w_{i,j}^2. \tag{11}$$

Minimizing the cross entropy between the true distribution p and the predicted distribution q can be interpreted as forcing the mass of q towards the distribution of p, i.e. towards the correct labels. Studying (10), we can see that a perfect prediction ($q = p$) yields zero loss, while a non overlapping distribution situation such as $p = [1, 0, 0], q = [0, 0.5, 0.5]$ yields an infinite loss.

5.2.3 Optimization

The goal is to train the classifier, i.e. to minimize the total loss over the training set. This is done with the help of an optimizer. The optimizer uses the model to predict labels for a batch of images at a time. The loss function is then used to figure out in what way to update the weights to achieve a lower loss. The weights are updated accordingly and the optimizer starts over with a new batch. Once everything in the training set has gone through the training procedure once, the process starts over. This is called an *epoch*.

A foundation in many optimizers is the use of gradient descent, an approach where weights are updated in the opposite direction of the gradient of the loss function. The optimization process is described in more detail and for a neural network architecture in Section 5.3.2.

5.3 Neural Networks

5.3.1 Architecture

A Neural Network consists of an input layer, an output layer and any number of hidden layers. Every layer consists of one or more *neurons*. The neurons are the building blocks of the neural network, each performing mathematical operations. Neuron layers can be connected in different ways, for example in the commonly used fully connected manner, where every neuron in layer i is pairwise connected to every neuron in layer $i+1$, while no connections within a layer exists. In this case, layer $i + 1$ is fully connected. A neuron with n inputs x_i has n weights w_i corresponding to the inputs and one bias term b. Every neuron performs the operations

$$f\left(\sum_i^n w_i x_i + b\right) = O, \tag{12}$$

where f is an arbitrary function called "activation function" and O is the neuron output. It is a common practise to use activation functions which are nonlinear. Non-linearity is needed to acquire depth in a network. One way to look at this is to consider the following: a linear neural network is effectively never deeper than one layer, since summing up a set of linear layers is still only one linear function. The same does not apply when non-linearity has been introduced, thus allowing the network to easily model more complex patterns.

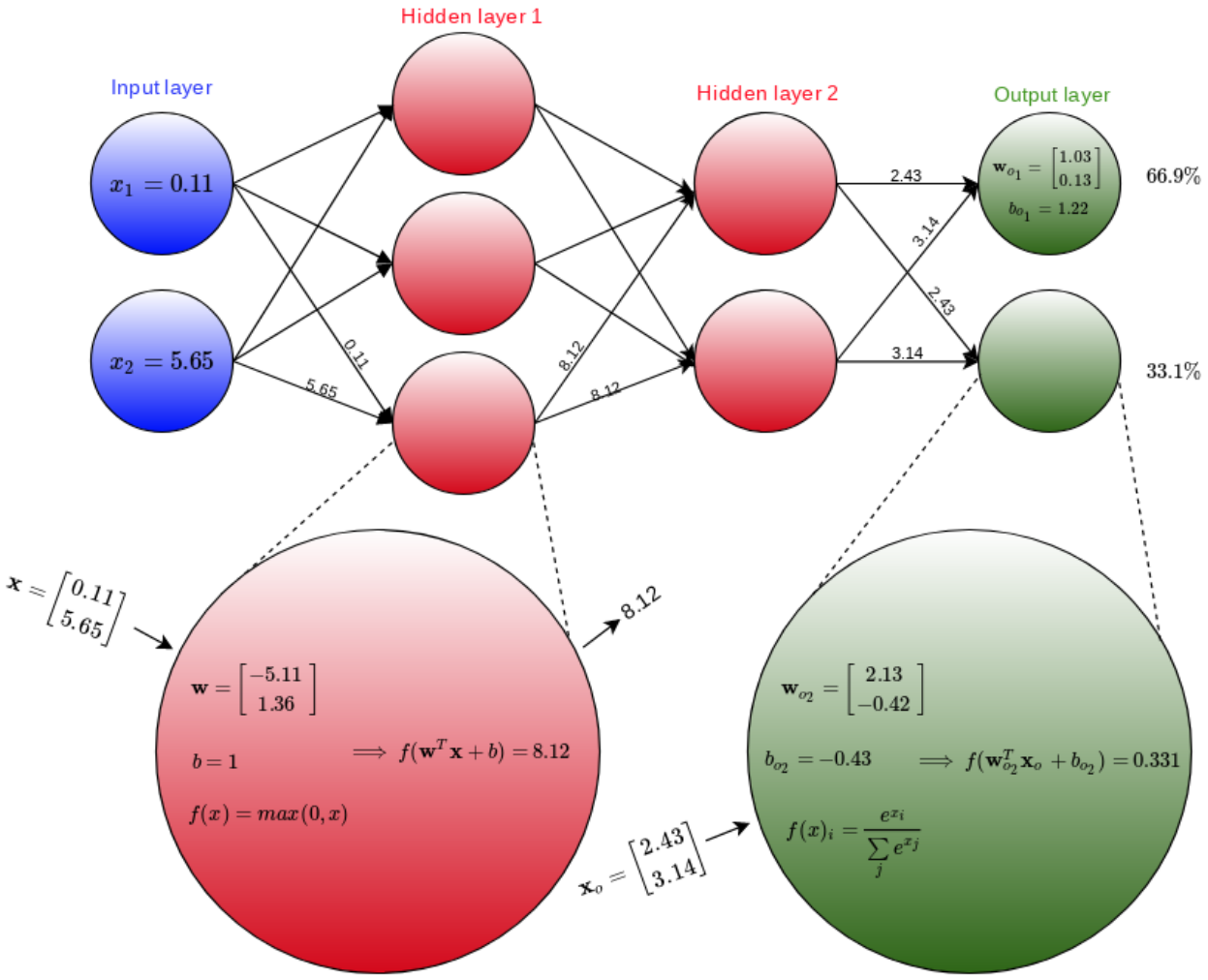

Figure 1: A simple fully connected neural network architecture using cross entropy loss.

Commonly used activation functions are *tanh*

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \tag{13}$$

Rectified Linear Units (ReLU)

$$f(x) = max(0, x), \tag{14}$$

and on the last layer using cross entropy loss function as seen in (7), *softmax*

$$f(x)_i = \frac{e^{x_i}}{\sum_j e^{x_j}}, \tag{15}$$

which squashes the outputs, i.e. the class scores into probabilities. To clarify, it seems to be common practise for the cross entropy loss from (11) not to include the softmax part, but to expect input equal to q in (7). This results in the fact that you have to use a softmax activation function in the output layer, which holds the class scores, to utilize the cross entropy loss function as defined.

Let's look at a simple neural network design of fully connected layers, portrayed in Figure 1. The classification task at hand could be anything, but let's consider the case of determining whether a given text is written in French or English. The first input value represents the average number of the letter "a" in a sentence. The second input value represents the average word length. Cross entropy loss is used and therefore also softmax activation function on the output layer. Given this and that we have 2 labels, "English" and "French", the output layer consists of 2 neurons expressing the class probabilities. All other layers use ReLU as activation function. Since this network is an example with made up weights, i.e. it has not been trained, it is not defined which output probability corresponds to what label.

5.3.2 Optimization

A fundamental part of training a neural network model is updating weights and biases in such a way that the model is "improved". As described in Section 5.2.2, a loss function is the used measure of how big the model error is. Whether the model is improved after updating weights and biases is thus easy to see using the loss function. This however does not help in the task of updating weights and biases in a smart way. Simply put, we want to find a "good" minimum of the loss function, in as few iterations as possible. A "good" minimum is a general minimum, meaning that it's not specific for the training set, but should be present in any test set as well. Ending up at a loss function minimum specific for the training set is what's called overfitting.

A first, very simple approach of finding a loss function minimum would be to iterate over the weights and biases using a grid. For example, without biases, the 2D grid in Figure 2 could be traversed to find the optimal combination of w_1 and w_2, given the grid size and domain. For a more precise training, you can just use a finer grid. This approach is a classic example of a brute-force search, which usually isn't an optimal method due to its rapidly increasing run cost for an increasing problem size. We will, as you will see later on, have millions of weights to train which makes the brute-force search impossible to use in practise. As stated above, we want to find a good minimum in as few iterations as possible, and this is where the backpropagation algorithm comes into play.

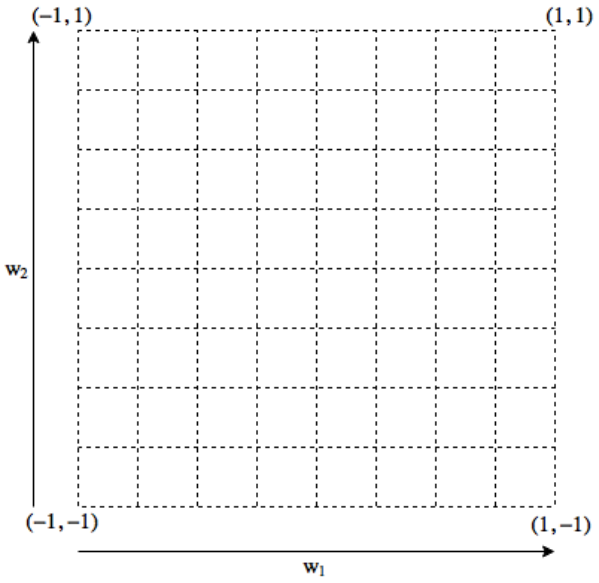

Figure 2: A 2D grid search domain of [-1 1], for weights w_1 and w_2. Every intersection in the grid represents a point where the loss given w_1 and w_2 is calculated.

5.3.2.1 Backpropagation

Backpropagation, also called backward pass, is the process of sending expression derivative values backwards through a network. This is done by using the chain rule

$$\frac{df}{dx} = \frac{df}{dy}\frac{dy}{dx} \tag{16}$$

to calculate the local derivative in every part of the network and then propagating this backwards. Let's examine this process with the help of a simple example. Consider the neural network in Figure 3.

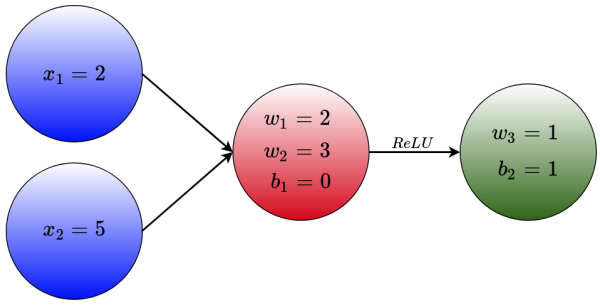

Figure 3: A simple fully connected neural network.

Let's assume the use of a L2 norm loss function together with this network. The L2 norm loss function is defined as

$$L = \sum_{i=1}^{n}(s_i - y_i)^2, \tag{17}$$

where n is the batch size to compute the loss over, y_i is the correct score and s_i is the predicted score. The prediction s_i is outputted from the output layer in Figure 3. For simplicity we assume the batch size $n = 1$. All mathematical expressions in this network are a combination of the following unary, atomic expressions, usually called *gates*

$$f_a(x) = ax, \qquad \frac{df_a(x)}{dx} = a, \tag{18}$$

$$f_c(x) = c + x, \qquad \frac{df_c(x)}{dx} = 1, \tag{19}$$

$$f_d(x) = d - x, \qquad \frac{df_d(x)}{dx} = -1, \tag{20}$$

$$f(x) = x^2, \qquad \frac{df(x)}{dx} = 2x, \tag{21}$$

where a, c and d are real number constants. The ReLU function can be expressed using (18), but for clarity ReLU and its derivative is defined in (22), where ReLU is abbreviated as R.

$$R(x) = \begin{cases} 0, & x \leq 0 \\ x, & x > 0 \end{cases}, \qquad \frac{dR(x)}{dx} = \begin{cases} 0, & x \leq 0 \\ 1, & x > 0 \end{cases}. \tag{22}$$

The full expression for the network is

$$(R(w_1x_1 + w_2x_2 + b_1)w_3 + b_2 - y)^2. \tag{23}$$

Using (18) through (22), (23) can be expressed as a circuit diagram shown in Figure 4, where y is the given correct value, green values are flowing forward and red values are flowing backward through the network. The green values (above the lines) originate from the weights, biases and the y-term. These values are sent through the gates to finally result in a loss, $L = 25$. Similarily, the red values (under the lines) originate in the outmost local derivative value of 10, calculated using (21). Every local derivative value can with the help of (16) generate the connected derivative values to the left. Let's for example look at the top left of Figure 4, which we can investigate using the following functions

$$\begin{aligned} f(w_1, x_1) &= w_1x_1, & \frac{df}{dx_1} &= w_1, & \frac{df}{dw_1} &= x_1, \\ g(w_2, x_2) &= w_2x_2, & \frac{dg}{dx_2} &= w_2, & \frac{dg}{dw_2} &= x_2, \\ q(f, g) &= f + g, & \frac{dq}{df} &= 1, & \frac{dq}{dg} &= 1, \end{aligned}$$

where f and g represent $f(w_1, x_1)$ and $g(w_2, x_2)$ respectively. In this section of the circuit our start value is the derivative of the total circuit expression with regard to q, $\frac{dL}{dq} = 10$. From here we can go deeper and calculate the derivatives next in line, by applying (16)

$$\begin{aligned} \frac{dL}{df} &= \frac{dL}{dq}\frac{dq}{df} = 10, \\ \frac{dL}{dg} &= \frac{dL}{dq}\frac{dq}{dg} = 10. \end{aligned}$$

In the same way the next layer of derivatives can be

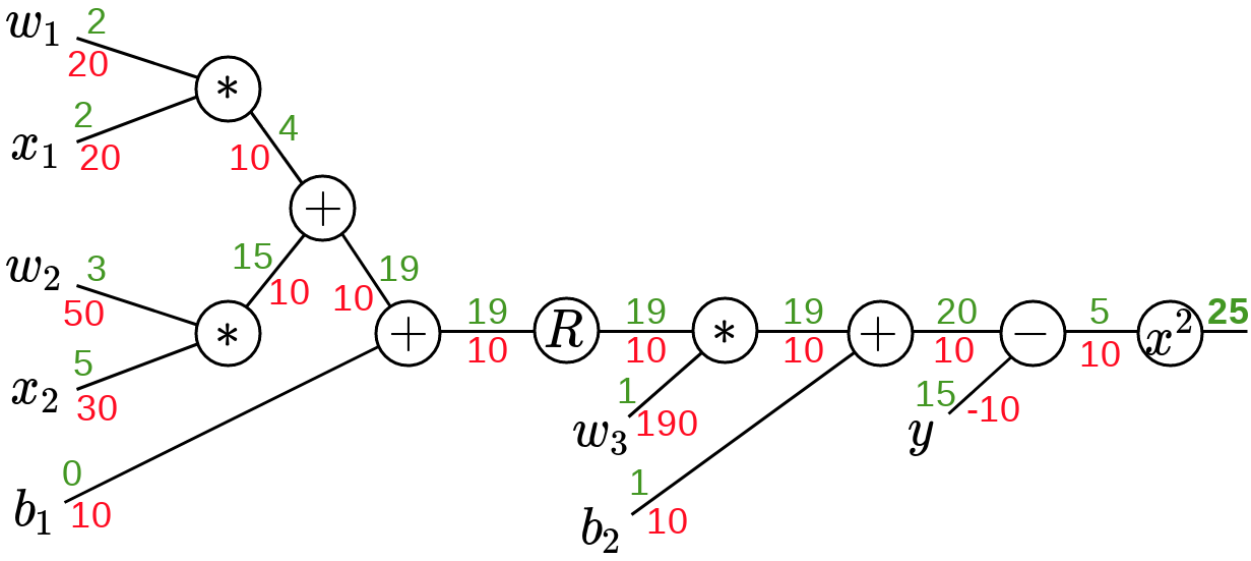

Figure 4: A circuit diagram describing the network in Figure 3 with L2 norm loss function. Green values are propagated forward and red backwards through the network.

determined

$$\frac{\boldsymbol{dL}}{\boldsymbol{dw_1}} = \frac{\boldsymbol{dL}}{\boldsymbol{df}}\frac{\boldsymbol{df}}{\boldsymbol{dw_1}} = \mathbf{20},$$
$$\frac{dL}{dx_1} = \frac{dL}{df}\frac{df}{dx_1} = 20,$$
$$\frac{\boldsymbol{dL}}{\boldsymbol{dw_2}} = \frac{\boldsymbol{dL}}{\boldsymbol{dg}}\frac{\boldsymbol{dg}}{\boldsymbol{dw_2}} = \mathbf{50},$$
$$\frac{dL}{dx_2} = \frac{dL}{dg}\frac{dg}{dx_2} = 30.$$

Since only weights and biases are to be updated, these are the only derivatives of interest. Combining all such derivatives result in a gradient, which tells us in what direction the total expression L grows the fastest, with regard to all trainable parameters.

5.3.2.2 Parameter update

Using the gradient descent technique, the trainable parameters will be updated in the opposite direction of above mentioned gradient. Gradient descent can mathematically be expressed as

$$\mathbf{x}_{i+1} = \mathbf{x}_i - \gamma \nabla L(\mathbf{x}_i), \tag{24}$$

where $\mathbf{x}_{i+1}$ is the new updated vector of trainable parameters, $\mathbf{x}_i$ the current vector of trainable parameters, γ the step size factor and $\nabla L(\mathbf{x}_i)$ the gradient of the loss function L with regard to the trainable parameters in $\mathbf{x}_i$.

Most high performance optimizers use some variation of the gradient descent technique. Examples are *Adagrad*, *RMSProp*, *AdaDelta* and *Adam*. These optimizers have the additional property of taking knowledge from previous steps, usually called momentum, which makes the stepping process smoother and less sensitive to highly divergent training examples. We have mostly used the AdaDelta and Adam optimizers.

5.3.2.3 Regularization

Overfitting can be prevented as mentioned in Section 5.2.2 using a regularization term in the loss function, which penalizes high value weighs resulting in a less specific fit. Another regularization technique is the *dropout* technique [10], which is applied in the training phase at run time. The main idea is to train random subsets of the neural network every training iteration, to prevent neurons from coadapting too much. Any neuron, except in the output layer, can be given a nonzero probability p_d to be dropped in any training iteration. If a certain neuron is to be dropped, all connections to and from that neuron is removed for that iteration. When training is done, all neuron are used together again

and their weights and biases are multiplied by $1-p_d$. This ensures every neuron to have the same expected output for both training and testing. Consider the simple example in Figure 5.

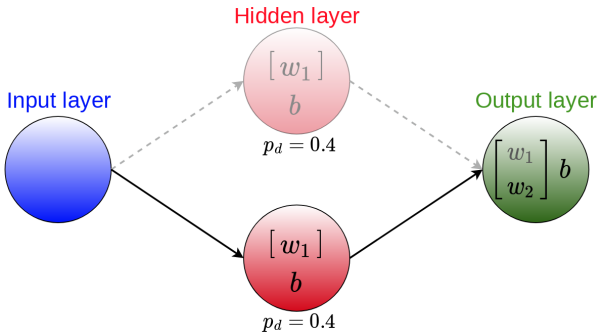

Figure 5: Example of a dropped neuron during training using the dropout technique.

Here we are observing the model for a training iteration where 1 out of 2 neurons with nonzero dropout probability p_d has been dropped. The dropped neuron is not updated during this iteration, and neither is the corresponding weight w_1 in the output neuron. All weights and biases in the hidden layer are multiplied by $1 - p_d = 0.6$ when predicting labels since the dropout probabilities here are $p_d = 0.4$.

5.4 Convolutional Neural Networks

One downside to a fully connected neural network is the number of weights, which quickly grows out of hand with increasing input data dimensions. With a small image of say 100x100 pixels, each neuron in a fully connected first hidden layer would have 10000 weights. To solve this problem researchers once again turned to nature, which gave rise to Fukushima's *neocognitron* [11] and LeCun's *convolutional neural network* (CNN) [3]. Although the latter would be the name that stuck, the principle is the same.

5.4.1 Convolutional layer

A CNN tries to mimic the photo receptors of the eye, where the neurons map to a small region of the visual field. The equivalent of this in a CNN are the convolutional layers which has a number of kernels (also known as filters). These kernels consists of small matrices of weights, usually the size 3x3 or 5x5. The kernels takes an input region with the same size as the kernel, and performs a element-wise multiplication of all the element (a Hadamard product). After the Hadamard product is calculated, the sum is computed and the bias of the kernel is added. The mathematical expression is

$$O(k^l) = b^l + \sum_i \sum_j x_{i,j} w^l_{i_j}, \tag{25}$$

where $O(k^l)$ is the output of kernel k^l, $x_{i,j}$ is the input value for position $[i, j]$, $w^l_{i,j}$ is the weights of the kernel k^l for position $[i, j]$ and b^l is the bias of the kernel k^l. $[i, j]$ is the x and y values in the kernel, and the corresponding values in the input. In order to construct an entire convolutional layer, these kernels are moved across the input image with a set stride. For each new position, (25) is performed and added to the output. This greatly reduces the number of weights needed, since the weights of a kernel are the same for the entire input. A convolutional layer with 10 kernels of the size 3x3, where the input is two-dimensional (say for example a gray-scale image), then only needs $10 \cdot 9 + 10 \cdot 1 = 100$ parameters. Figure 6 depicts how the operation is performed with a 3x3 kernel on a 5x5 input, padded with zeros. One thing that should be mentioned is that non-linear activation function is still applied to the output after a convolutional layer.

5.4.2 Padding

One direct consequence of moving the kernel across the input layer is that the output size will be lower than the input size. To counteract this *padding* is sometime used. Padding simply appends a set border of zeros around the input image. This is done to increase the size of the output from the convolutional layer.

5.4.3 Pooling

In addition to the convolutional layers inside a CNN, another operation called *pooling* is usually used. Pooling is performed to reduce the spatial size of the data through out the network, something that reduces the number of computations in each iteration, and also prevents overfitting (due to less parameters).

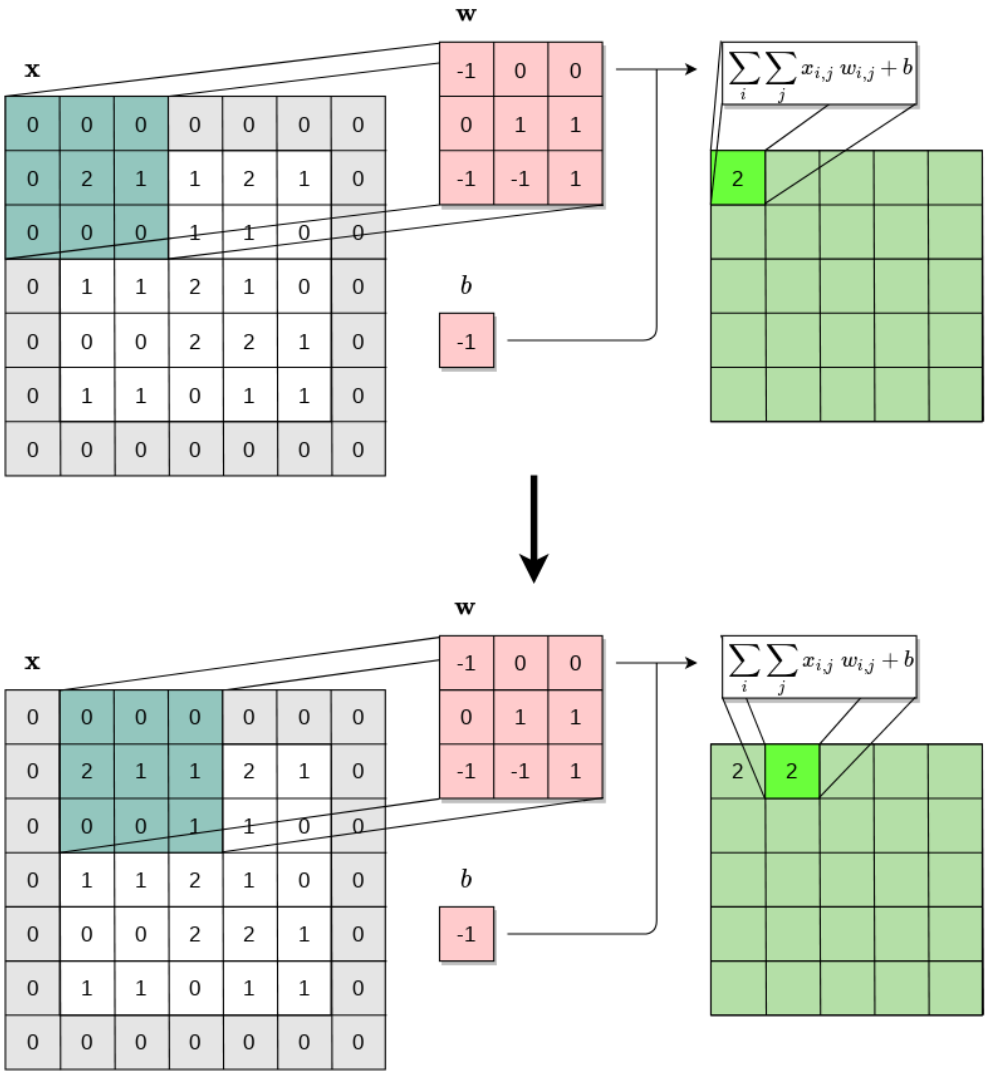

Figure 6: An example of a convolution in a CNN. The input is padded with zeros to preserve the grid size. The convolution is performed by a 3x3 kernel using a stride of 1.

There are mainly two types of pooling operations that are used; average pooling and max pooling. In both versions of pooling a kernel is moved across an input layer with a specified stride length, but how the output is calculated differs slightly. In average pooling, the output is equal to the average of the total kernel, while in max pooling the maximum value of the kernel is chosen as the output. In Figure 7 a max pooling operation is shown.

An intuitive explanation to why pooling works is by local similarity. An area adjacent to a point (or in our case, a pixel) often share similarities with that point. This means that we with high probability can compress the information of that local feature into one point (in our case pixel).

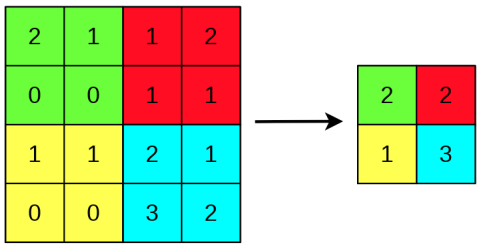

Figure 7: Max pooling operation using a 2x2 pooling kernel, with stride length 2.

5.4.4 Batch normalization

Batch normalization is a fairly new CNN concept in which the goal is to reduce the amount the hidden unit values are shifted around inside the network. This is called covariance shift. Since the input of hidden layers in a neural network is dependent on the output of previous layers, this means that distribution of the input to hidden layers change as earlier layers parameters are changed. This is done by normalizing on each batch in between hidden layers. It also has a regularizing effect, which counteracts overfitting. In this project we will not go into to much detail about batch normalization, but the original paper by Ioffe et al. is a great source of information [12].

5.4.5 Bottleneck layers

Another concept which is used within CNN's are *bottleneck layers*. Bottleneck layers tries to reduce the dimensionality of the information in the network, but instead of reducing it spatially like a pooling layer, it reduces the dimensionality in the kernel space through a convolutional layer consisting of 1x1 kernels. Since one can decide the number of kernels for a specific convolutional layer, these 1x1 kernels with stride one can be used to 'merge' higher dimensional input into an output of reduced dimensions.

5.5 Residual networks

Since the introduction of neural networks for image recognition the field has been on constant move to create deeper and more complex models. One of the earlier, deeper, models from 2015 was VGG, which had in between 11 to 19 layers [13]. These new models broke older records and further evolved the computer vision field. One of the obstacles that arose with deeper models was the notorious vanishing/exploding gradients problem. This has since then been mostly addressed by normalized initialization and intermediate normalization layers such as batch normalization.

However, a new problem has been exposed when deeper networks start converging; a degradation problem. With the network depth increasing, accuracy gets saturated and then degrades rapidly. This is not caused by overfitting (which has been confirmed through experiments on the accuracy of *training data*) [7, 14, 15], which means that the deeper architecture seems unable to find additional features in an image.

A proposed solution to this problem was introduced with deep residual learning framework (ResNets) [7]. Where ordinary architectures hope that stacked layers directly fit a desired underlying mapping, ResNets instead tries to fit these layers to a residual mapping. This is implemented by having shortcut connections in the network, which means that the input signal takes a shortcut around a number of layers. See Figure 8 for an explanation of the shortcut connection. These *residual blocks* will be instantiated with weights close to zero, and they are therefore an approximative identity of the problem. If it shows that any layers are redundant, they can easily converge to zero without hindering the forward propagation of information in the model (since the information can take a shortcut around layers).

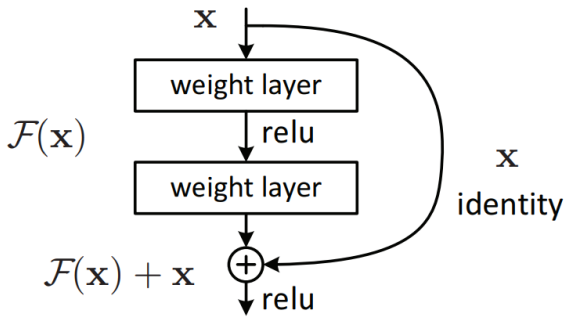

Figure 8: Residual learning; the idea of shortcut connections. Image courtesy of [7].

This underlying residual mapping that we want to fit is denoted $F(\mathbf{x}) + \mathbf{x}$, where $F(\mathbf{x})$ is the output of the previous layer. The building block shown in Figure 8 is therefore defined as

$$\mathbf{y} = F(\mathbf{x}, \{W_i\}) + \mathbf{x}, \tag{26}$$

where $\mathbf{x}$ and $\mathbf{y}$ are the input and output vector, and W_i are the weights for layer i. The operation $F + \mathbf{x}$ is performed through element-wise addition, something that means that the dimensions of $\mathbf{x}$ and F must be equal. When this is not true, one can perform a linear projection W_s on the shortcut connection to receive

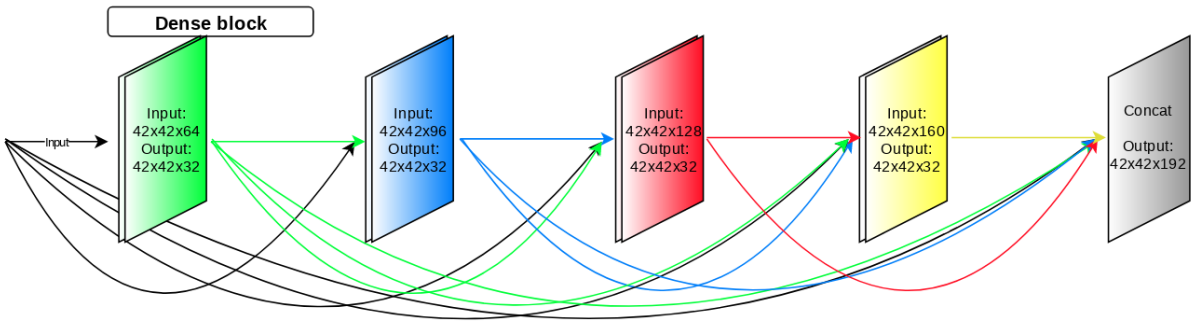

Figure 9: A 5-layer dense block. Each layer takes all preceding feature-maps as input.

$$\mathbf{y} = F(\mathbf{x}, \{W_i\}) + W_s\mathbf{x}. \tag{27}$$

5.6 DenseNets

As mentioned in the section about ResNets, CNNs are currently moving towards deeper and deeper architectures. The approach for solving the problem of vanish gradients and input usually share a key characteristic; they try to create short paths from early layers to later layers (or the other way around). The Dense Convolutional Network, or short DenseNet, tries to distill this feature into a simple connectivity pattern [6]. In DenseNets, all layers (with matching feature-map sizes) are connected to each other. To preserve the feed-forward nature of neural networks, layers are only connected forward, i.e. a layer does not get any input from succeeding layers. These modules of highly connected layers are called blocks, and the structure is illustrated in Figure 9. This means that the l^{th} layer in a dense block has l inputs, and the total amount of connections in an L-layer network is $\frac{L(L+1)}{2}$ (instead of L connections like an ordinary CNN).

Something that might be rather counter-intuitive with DenseNets is that they require fewer parameters than traditional convolutional networks. This is due to the fact that they don't need to relearn redundant feature maps. In traditional architectures one could view the entire network as an algorithm where a state is passed between the layers; it changes the state slightly but also passes on information that needs to be preserved. This means that some features will be relearned between layers. In DenseNets the layers are very narrow (with in between 6 to 64 filters per layer), which means that every layer only adds a small set of feature-maps to the entire model.

The second big advantage with DenseNets is the improved flow of information and gradients in the network. Since each layer has direct access to the gradient from the loss function and original input signal, the network is trained utilizing deep supervision.

An input image x_0 passes through a convolutional network consisting of L layers. These layers implements a non-linear transformation $H_l(\cdot)$ which consists of a composition of the operations mentioned earlier. Here $\cdot$ depicts the input into the non-linear transformation. The output of the l^{th} layer is denoted as x_l.

In a traditional convolutional neural network the l^{th} layer output is connected to the $(l+1)^{th}$ layer, which means that the following will be true: $x_{l+1} = H_l(x_l)$. In DenseNets a different connectivity pattern is used; within a dense block a layer l has a direct connection to all subsequent layers within the block. This is illustrated in Figure 9, and means that a layer l receives the feature maps of all preceding layers within a dense block. The output from layer l is therefore

$$x_l = H_l([x_0, x_1, ..., x_{l-1}]), \tag{28}$$

where the vector $[x_0, x_1, ..., x_{l-1}]$ refers to the output of all previous layers in the dense block.

6 Setup

6.1 Datasets

6.1.1 Facial Expression Recognition 2013 (FER2013)

6.1.1.1 Reasons for use

The Facial Expression Recognition 2013 dataset was created by Pierre Luc Carrier and Aaron Courville, and was the dataset used in the *Challenges in Representation Learning: Facial Expression Recognition Challenge* competition hosted by Université de Montréal and Google in 2013. We chose to train and validate on this particular dataset because of its relatively big size and due to the fact that it has been collected "in-the-wild". After examining the dataset further we came to the conclusion that the dataset contains a broad range of subjects and the images have been taken with differing light conditions.

6.1.1.2 Specifications

The images in FER2013 was collected using Google's image search API, where a set of 184 emotion-related keywords such as "blissful" and "enraged" were used. These keywords were used with words for gender, age and ethnicity to obtain a total of almost 600 facial image search queries [16]. Out of these queries, the first 1000 images were saved and cropped using OpenCV's face detection. The result was a dataset of 35887 grayscale images divided as shown in Table 1.

These images were then further divided into three sets: a training set of 28709 images, a public test set of 3589 images, and a competition test set of 3589 images. Figure 10 shows an example of images from the dataset.

Figure 10: An example of images of the seven labels from FER2013.

6.1.2 AffectNet

6.1.2.1 Reasons for use

The name AffectNet is derived through the words "facial ***Affect*** from the Inter***Net***", and is another dataset that is produced from images "in-the-wild". It tries to fill in the gaps of other big, "in-the-wild", datasets which usually only has one or two of the affective-related categorical system: basic emotions defined by Ekman [2], the dimensional model which measures valence and arousal over a continuous scale, and Facial Action Coding System (FACS), where all facial actions are described in terms of action units (AUs) [17]. AffectNet has values for all these three categorical systems.

Label	Anger	Disgust	Fear	Happiness	Sadness	Surprise	Neutral
# of images	4953	547	5121	8989	6077	4002	6198

Table 1: Distribution of data over the different labels for FER2013.

6.1.2.2 Specifications

A total of 362 emotion related keywords were put together and translated into five languages: Spanish, Portuguese, German, Arabic and Farsi. All in total 1250 different emotion related tags were created, and used for parsing the three search engines Google, Bing and Yahoo. Figure 11 shows an example of images from the dataset.

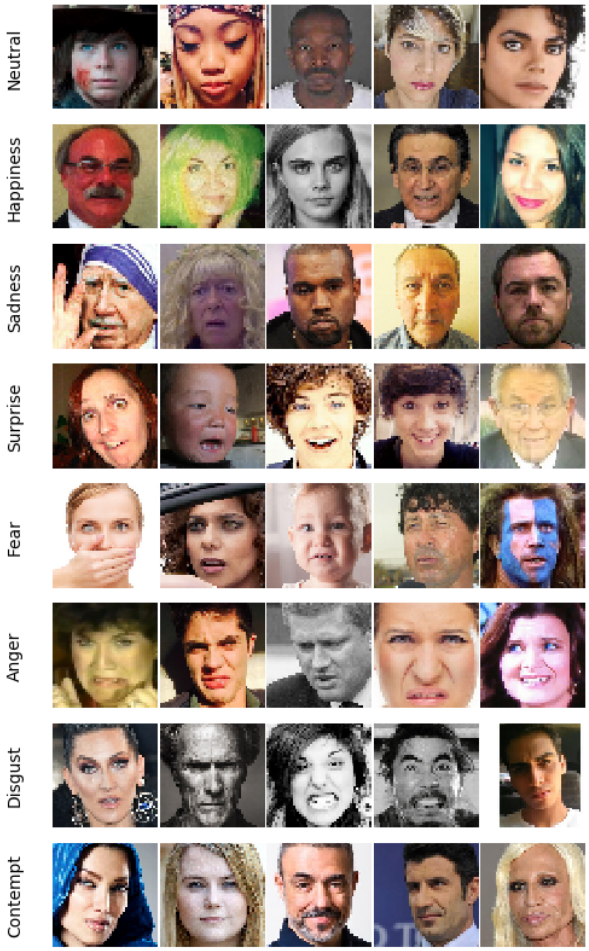

Figure 11: An example of images from eight of the labels of AffectNet.

Roughly 450000 images were manually annotated by 12 professional annotators [18] using 11 different labels: Neutral, Happiness, Sadness, Surprise, Fear, Anger, Disgust, Contempt, None, Uncertain and Non-face. The *None* label defines facial expressions/emotions which do not belong to the other eight (such as sleepy, seducing, focused etc.) The *Non-face* category was defined as images that: (1) do not contain a face in the image, (2) contain a watermark on the face, (3) the face detection algorithm utilized to crop images does not find a face, (4) the face is drawn, animated or painted, or (5) the face is distorted beyond normal or natural shape. The *Uncertain* label was used by the annotator if they were uncertain about the facial expression. The dataset includes a validation set of about 4500 images. There is also bounding box information for every image for easy face cropping. Table 2 shows the distribution of labels across the dataset.

6.1.3 Imaging, Robotics, and Intelligent Systems (IRIS) Thermal/Visible Face Database

6.1.3.1 Reasons for use

The new General Data Protection Regulation (GDPR) [19] protects the personal data of EU residents, anywhere in the world. The regulations apply to all companies based in the EU, collecting data from the EU or of EU residents. In short this means that working with images, in whatever shape or form, capable of revealing a persons identity may require consent. Machine Learning work and possibly research using this kind of images will probably have to adapt to this law, for example by buying GDPR-safe datasets.

A way of possibly getting around this extra cost is to use thermal images. Whether these images are exempt from the GDPR regulations seems to be rather ambiguous since it can be debated whether such an image is a personal data. In our experience, if you don't know who you're looking for, identifying a person from a thermal image is practically impossible. However, in the case of comparing a thermal image to a given set of people, identification is indeed a possible task.

Whether thermal images are a solution to GDPR compliance or not, it is in our interest to investigate the usefulness of these in comparison to visible images. Thermal images are for example unaffected by light conditions, something that can make or break the recording of visible images.

6.1.3.2 Specifications

Despite the lack of thermal data out there, especially with the appropriate labels, we found the small data set *IRIS Thermal/Visible Face Database* [20] to work

Label	Neutral	Happiness	Sadness	Surprise	Fear	Disgust	Anger	Contempt
# of images	75374	134915	25959	14590	6878	4303	25382	4250

Table 2: Distribution of data over the different labels for AffectNet.

with. This data set contains synced visible and thermal images in a lab setting with three labels; surprise, happiness and anger. For every person and label there are 11 images taken from an angle of -90 to 90 degrees where 0 degrees is straight onto the face. The data set is divided into different light conditions, for example normal lighting, lighting from the side and poor lighting. From the normal lighting collection of images we extracted a training and validation set of images. Our extracted data set features 30 individuals, male and female, and consists of 715 images. Out of these 715 images, 640 and 75 belong to the train and validation set respectively. Figure 12 shows an example of images from the dataset, where the visual image is shown together with its thermal counterpart.

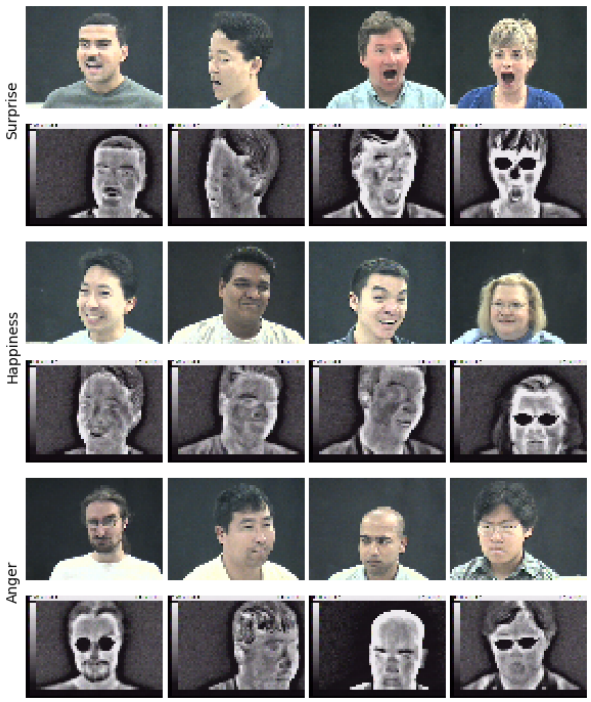

Figure 12: An example of images from the three labels of IRIS. The visual image is shown in the upper row and the thermal counterpart in the lower row.

6.2 Data preprocessing

All datasets cannot be processed exactly the same due to varying image formats, resolutions, color channels and facial crops. There are however two major processing steps all training data go through; augmentation and normalization.

6.2.1 Augmentation

To reduce overfitting, different augmentation techniques can be applied. We opted to replicate the augmentation made by Sang et al. in [5], and our training images all go through the following steps every training iteration:

1. Horizontal flip with probability 0.5
2. Random rotation of -45 to 45 degrees
3. Random resize of 0 to 12 pixels increase
4. Crop back to model input size

As stated, this should counteract overfitting since a certain image most likely will look more or less different in every training iteration. This means that the model won't adapt to images in too much detail which will result in a more general fit.

6.2.2 Normalization

Normalization is applied to both training sets and test sets. The purpose of normalizing the images is to "center" the data, to force all pixel values into a similar range. This makes the training less sensitive to varying light conditions, and the test set is normalized the same for consistency. Imagine for example two images of the same exact motif, in different light conditions. We want the network to react similarily to these two images, even though they might have different *dynamic ranges* and major pixel wise inequalities. As a result the class scores could differ a lot and gradients during backpropagation would do the same resulting in a significantly different parameter update.

First the below defined *Image normalization* is applied while reading the images from file. When all

images from the same dataset is in memory, the below defined *Pixel normalization* is applied. The normalization process is applied exactly as in [5], since it seems to be favorable for the emotion recognition problem.

6.2.2.1 Image normalization

First the mean value of an image is subtracted from the image, channel wise. This centers the values around zero. To minimize said variation in dynamic range, all pixel values are channel wise multiplied by a factor to end up with a standard deviation of $\sigma = 3.125$. This value was chosen since it was used by Sang et al. [5]. The value of the image after subtracting the mean is given by

$$f(x, y) = I(x, y) - \frac{1}{x \cdot y} \sum_{x,y} I(x, y), \tag{29}$$

where x and y are the coordinates for a given image I, and $f(x, y)$ is the value of the image after subtracting the mean.

The biased standard deviation is given by

$$s = \sqrt{\frac{1}{N} f(x, y)^2}, \tag{30}$$

where s is the standard deviation for an image, $f(x, y)$ is the image with the mean subtracted from Equation 29 and N is the number of pixels in an image. As a final step the solution of 29 is divided with 30 and then multiplied by the value $\sigma = 3.125$ (which was taken from Sang et al. [5]).

6.2.2.2 Pixel normalization

First the mean value for each pixel over a given data set is subtracted from said pixel, channel wise. This centers the values around zero. To minimize said pixel inequalities the training and test sets are normalized pixel wise. For each pixel, the corresponding value in every image in the data set is channel wise multiplied by a factor to end up with a standard deviation of $\sigma = 1$ for that pixel. The equations for calculating the mean of a pixel across a dataset is given by

$$f(x, y) = I(x, y) - \frac{1}{M} \sum_{m=1}^{M} I_m(x, y), \tag{31}$$

where x and y are the coordinates of an image I, M is the number of images which we are normalizing over and $f(x, y)$ is the value of a pixel in an image after subtracting the pixel mean.

The biased standard deviation for a pixel is given by

$$s_{x,y} = \sqrt{\frac{1}{M} f(x, y)^2}, \tag{32}$$

where s is the standard deviation for a pixel (x, y), $f(x, y)$ is the value of a pixel with the mean subtracted from Equation 31 and M is the number of images in the dataset. As a final step the normalized pixel value is calculated by dividing $f(x, y)$ by the standard deviation $s_{x,y}$ and multiplying with one.

6.3 Models

6.3.1 BK models

6.3.1.1 Background

In this project, two of the models we've chose to focus on are BKStart and BKVGG12. The BKStart model described is the authors recreation of the winning model of the 2013 *Kaggle Facial Expression Competition*. The BKVGG12 model is one of four BKVGG models proposed by the authors, which are based on the VGGNet models that placed second in the 2014 *ImageNet Large Scale Visual Recognition Competition*. The four BKVGG models proposed have a heavily reduced number of filters in the convolutional layers. The VGGNets were trained on the ImageNet dataset which is much bigger than the FER2013 dataset, see Section 6.1.1, used in this paper. Training on a smaller dataset is more prone to overfitting which is what the authors try to avoid with this change.

6.3.1.2 BKStart

The BKStart model is a rather simple CNN working with image input size 42x42. The architecture is combined with dropout on the fully connected layer (3072 neurons) before the output layer and can be seen in Table 3.

Layer	Filters	Filter size	Stride	Act.	Neurons
Input	-	-	-	-	-
Conv2D	32	5x5	1	ReLU	-
Max Pool	-	3x3	2	-	-
Conv2D	32	4x4	1	ReLU	-
Avg. Pool	-	3x3	2	-	-
Conv2D	64	5x5	1	ReLU	-
Avg. Pool	-	3x3	2	-	-
FC	-	-	-	ReLU	3072
FC	-	-	-	?	7

Table 3: The BKStart model architecture, where FC is short for fully connected layer and Act. is short for activation function. The ? means that the activation function depends on the chosen loss function. For cross entropy loss, softmax activation is used and for SVM loss there is no activation.

Our implementation of the BKStart model comes in two different versions; with and without added padding. Padding to preserve the grid size after every convolution has been added as a measure to extract more information from the original pixels. The drawback is that the number of trainable parameters grows by roughly factor 10, see Figure 20. The BKStart model without padding is denominated as BKMin.

6.3.1.3 BKVGG12

The BKVGG12 model is just like BKStart built using the definition given in by Sang et al. [5], with input size 42x42. Dropout is used on all fully connected layers except for the output layer. BKVGG12 is very similar to BKStart, but has a deeper layer architecture which is displayed in Table 4.

Just like BKStart, the BKVGG12 model has padding preserving the grid size after a convolution. Due to the higher number of layers however, there is no unpadded version. Without padding an image propagated through the network would run out of pixels to operate on.

Layer	Filters	Filter size	Stride	Act.	Neurons
Input	-	-	-	-	-
Conv2D	32	3x3	1	ReLU	-
Conv2D	32	3x3	1	ReLU	-
Max Pool	-	2x2	2	-	-
Conv2D	64	3x3	1	ReLU	-
Conv2D	64	3x3	1	ReLU	-
Max Pool	-	2x2	2	-	-
Conv2D	128	3x3	1	ReLU	-
Conv2D	128	3x3	1	ReLU	-
Max Pool	-	2x2	2	-	-
Conv2D	256	3x3	1	ReLU	-
Conv2D	256	3x3	1	ReLU	-
Conv2D	256	3x3	1	ReLU	-
FC	-	-	-	ReLU	256
FC	-	-	-	ReLU	256
FC	-	-	-	?	7

Table 4: The BKVGG12 model architecture, where FC is short for fully connected layer and Act. is short for activation function. The ? means that the activation function depends on the chosen loss function. For cross entropy loss, softmax activation is used and for svm loss there is no activation.

6.3.1.4 Andreas100

The Andreas100 model is simply the BKVGG12 model adapted for input size 100x100. The only difference is that the filter sizes for the convolutional layers have been doubled to 6x6. This has been done to roughly preserve the filter size to image size ratio.

6.3.2 ResNet

6.3.2.1 Technical information

The ResNets utilizes the operations batch normalization, rectified linear units, bottleneck and convolutional layers. All downsampling is done directly by convolutional layers instead of pooling layers, a concept introduced in Springenberg's paper on all convolutional nets [21]. Whenever the feature maps are halved in size, the number of filters for succeeding layers are doubled.

The shortcut connection described in Figure 8 only works whenever the input and the output is of the same dimension. To solve this problem whenever a downsampling occurs, the network utilizes two possible options: (A) it adds zero padding to increase the dimension of the input, or (B) use the projection shortcut defined in 27, which is done by 1x1

Layer	Input	Output	ResNet-18	ResNet-34	ResNet-50
Conv 1	42 x 42	21 x 21	7 x 7 conv, stride 2		
	21 x 21	11 x 11	3 x 3 max pooling, stride 2		
Convolutions (1)	11 x 11	11 x 11	[3 x 3, 64 ; 3 x 3, 64] x 2	[3 x 3, 64 ; 3 x 3, 64] x 3	[1 x 1, 64 ; 3 x 3, 64 ; 1 x 1, 256] x 3
Convolutions (2)	11 x 11	6 x 6	[3 x 3, 128 ; 3 x 3, 128] x 2	[3 x 3, 128 ; 3 x 3, 128] x 4	[1 x 1, 128 ; 3 x 3, 128 ; 1 x 1, 512] x 4
Convolutions (3)	6 x 6	3 x 3	[3 x 3, 256 ; 3 x 3, 256] x 2	[3 x 3, 256 ; 3 x 3, 256] x 6	[1 x 1, 256 ; 3 x 3, 256 ; 1 x 1, 1024] x 6
Convolutions (4)	3 x 3	2 x 2	[3 x 3, 512 ; 3 x 3, 512] x 2	[3 x 3, 512 ; 3 x 3, 512] x 3	[1 x 1, 512 ; 3 x 3, 512 ; 1 x 1, 2048] x 3
Classification	2 x 2	1 x 1	2 x 2 global average pooling		
	1 x 1		7 classes, fully connected, softmax		

Table 5: Different model setups for ResNet.

convolutions.

One thing to mention is that ResNet's were originally constructed to solve the object classification problem, and was tested on CIFAR-10 and ImageNet. This might mean that the model is not optimized for finding good features for facial expressions, something that would warrant further investigations.

If the reader wishes to learn more about ResNets structure and the residual mapping, we refer you to read the original paper by He et al. [7].

6.3.2.2 Model settings

The different models that we tried out in this thesis work were the ResNet-18, ResNet-34 and ResNet-50 models, where the numbers refers to the number of convolutional layers in each of the models. ResNet-34 and ResNet-50 are very similar in the way they are constructed, but the ResNet-50 utilizes bottleneck layers to a higher degree. As can be seen in the 6th column of Table 5, it first reduces the number of dimensions, then performs a 3x3 convolution, and finally restores the number of dimensions. Dropout appears in between each residual block, which are denoted by [3 x 3, 64 ; 3 x 3, 64] in Table 5.

The source code for the ResNet was implemented by R. Kotikalapudi [22].

6.3.3 Dense

6.3.3.1 Technical information

DenseNets utilizes the operations batch normalization (BN), rectified linear units (ReLU), pooling, bottleneck and convolution layers (Bottleneck-conv/Conv) which are further explained in Section 5.4. The composite function $H_l(\cdot)$, which was explained in Section 5.6 is defined as the following consecutive operations:

`BN` $\rightarrow$ `ReLU` $\rightarrow$ `Bottleneck-Conv` $\rightarrow$ `Dropout` $\rightarrow$ `BN` $\rightarrow$ `ReLU` $\rightarrow$ `Zero Padding` $\rightarrow$ `Conv-(3x3)` $\rightarrow$ `Dropout`

In the DenseNet the composite functions is called a convolutional block.

Between all dense blocks a transition block is utilized. Transition blocks are used to incorporate down-sampling throughout the network, since it isn't possible to concatenate feature maps of different sizes. The transition block also contains batch normalization, activation functions, compression convolutions, dropout and average pooling. The following consecutive operations are performed:

`BN` $\rightarrow$ `ReLU` $\rightarrow$ `Compression-Conv` $\rightarrow$ `Dropout` $\rightarrow$ `Average Pooling-(2x2), stride 2`

Compression convolutions follow the same principle as bottleneck layers, but instead of always reducing

Layer	Input	Output	Dense-121	Dense-121 red	Dense-169	Dense-169 red	Dense-161	Dense-161 red
Input Block	42 x 42	42 x 42	7 x 7 conv, stride 1					
	42 x 42	21 x 21	3 x 3 max pooling, stride 2					
Dense Block (1)	21 x 21	21 x 21	[1 x 1 conv, 3 x 3 conv] x 6					
Transition Block (1)	21 x 21	21 x 21	1 x 1 conv					
	21 x 21	10 x 10	2 x 2 average pooling, stride 2					
Dense Block (2)	10 x 10	10 x 10	[1 x 1 conv, 3 x 3 conv] x 12					
Transition Block (2)	10 x 10	10 x 10	1 x 1 conv					
	10 x 10	5 x 5	2 x 2 average pooling, stride 2					
Dense Block (3)	5 x 5	5 x 5	[1 x 1 conv, 3 x 3 conv] x 24		[1 x 1 conv, 3 x 3 conv] x 32		[1 x 1 conv, 3 x 3 conv] x 36	
Transition Block (3)	5 x 5	5 x 5	1 x 1 conv					
	5 x 5	2 x 2	2 x 2 average pooling, stride 2					
Dense Block (4)	2 x 2	2 x 2	[1 x 1 conv, 3 x 3 conv] x 16	-	[1 x 1 conv, 3 x 3 conv] x 32	-	[1 x 1 conv, 3 x 3 conv] x 24	-
Classification Block	2 x 2	1 x 1	2 x 2 global average pooling					
	1 x 1		7 classes, fully connected, softmax					

Table 6: Different model setups for DenseNet.

the number of output layers to a fixed amount, it compresses the number of layers to a number θm, where $0 < \theta \leq 1$ is the compression factor and m is the number of layers before compression. In all of our experiments we've set the compression factor to $\theta = 0.5$

A factor which is used throughout the DenseNet is the growth factor k. The growth rate is primarily used for two things within the DenseNet: to regulate how many new feature maps that are added to the total network knowledge, and to regulate how compression and bottleneck layers are changing the output.

A final parameter that decides how many features that are created is the initial filter number, which decides how many filters the initial convolutional layers should output. If the reader wish to learn more about how DenseNets are built we refer you to read the original paper by Huang et al. [6]. Figure 13 shows all the parts of the DenseNet tied together.

One thing to mention is that DenseNet's were originally constructed to solve the object classification problem, and was trained and tested against CIFAR, SVHN and ImageNet. This might mean that the model is not optimized for finding good features for facial expressions, but investigation this further is of interest.

6.3.3.2 Model settings

Mainly six different versions of DenseNets were tried to evaluate how different settings affect the performance of the model. These can be seen in Table 6 and are Dense-121, Dense-121-reduced, Dense-169, Dense-169-reduced, Dense-161 and Dense-161-reduced. The word "reduced" refers to a reduced version where the final block of the original model is removed; the reduced models only contains three dense blocks. Dense-121(-reduced) and Dense-169(-reduced) all have the initial filter size of 64, and a growth rate of 32, where as Dense-161(-reduced) have the initial filter size of 96 and a growth rate of 48. The number of Convolutional blocks differ between Dense-121(-reduced), Dense-169(-reduced), and Dense-161(-reduced); the exact number can be seen in Table 6.

The source code for the DenseNet was implemented by F. Yu [23], although some modifications were made to how dropout is handled (more on that in Section 7.1.3).

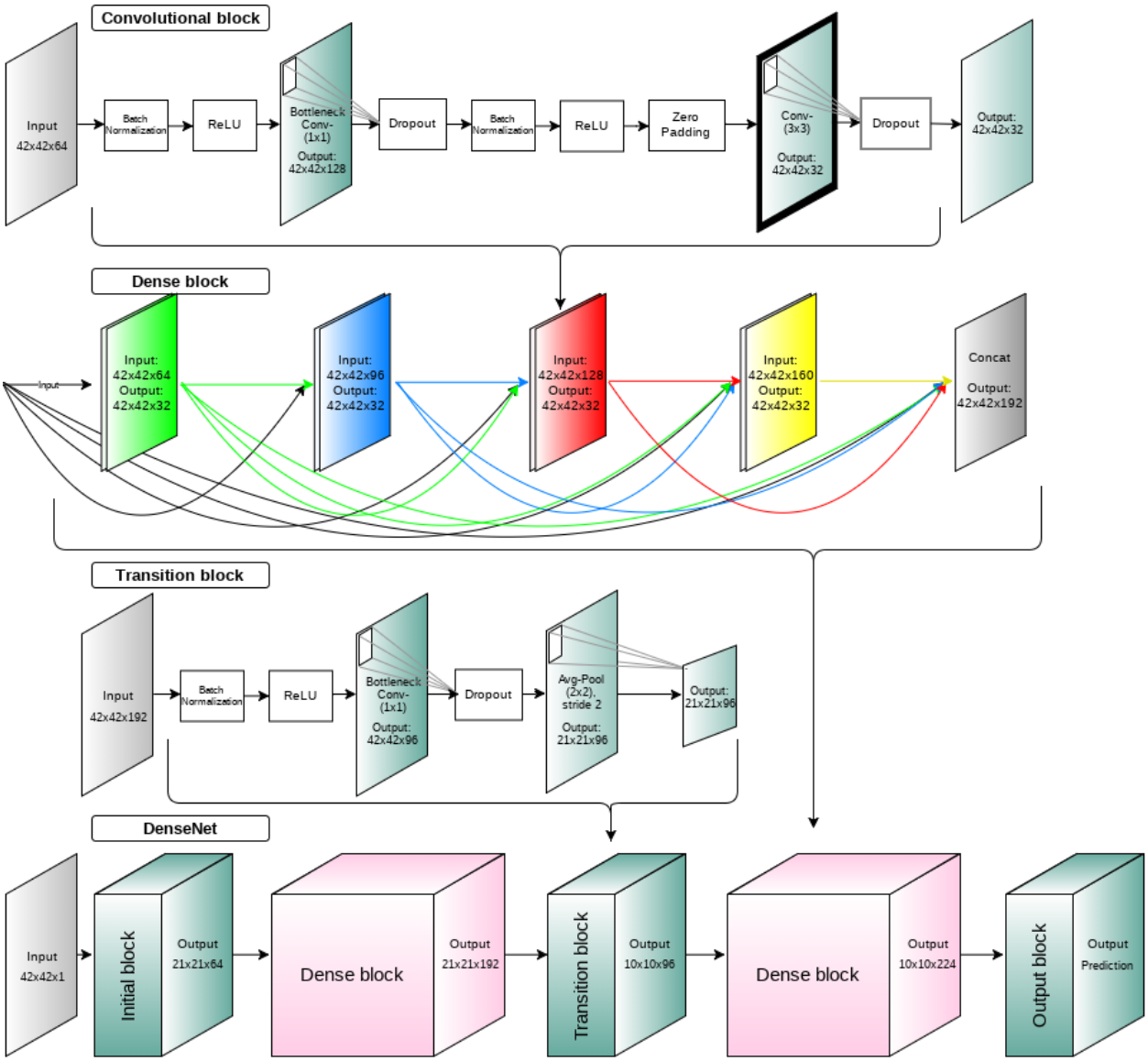

Figure 13: All parts of the DenseNet tied together. This model contains one initial block which downsamples the input, and is followed by two dense blocks, one transition block, and a final output block.

7 Method

This section describes our experimental work going from the defined model architectures in Section 6.3 to optimally trained neural networks. Our approach has been to isolate and fine tune one variable at a time, since trying a big number of variable combinations is unfeasible due to the time consuming model training. Properly training a model could take anywhere from a few hours to a week depending on the model and training dataset. This is why we've also tried to limit most variable tuning to the FER2013 dataset, which is roughly 7 % the size of our main dataset AffectNet. Our hypothesis is namely that the FER2013 dataset is big and diverse enough to indicate what model settings are in general preferable. If the influence of a variable still is uncertain or extra interesting it could be investigated further on AffectNet.

7.1 Model optimization against FER2013

Even though the models described in Section 6.3 are well defined, there are still a variety of settings and *hyperparameters* to tune. The tuning of these variables including optimal settings for the FER2013 dataset are presented in this section one variable at a time. For FER2013, the public test set mentioned in Section 6.1.1 has been used as validation set.

7.1.1 Loss function

As explained in Section 5.2.2 there are two commonly used loss functions for multiclass classification problems; Cross-Entropy and Multiclass SVM. Sang et al. used these two loss functions with similar results. [5] They conclude however that the L2 (squared) Multiclass SVM loss function produces slightly better results on the FER2013 dataset. On the public test set, which is what we evaluate against, the Multiclass SVM loss produces 0.5 percentage points higher accuracy using the BKStart model. No corresponding comparison is presented for BKVGG12.

The Cross-Entropy loss function is already defined in the Keras API which we are using, making it easy to start using. The Multiclass SVM loss function as defined in Section 5.2.2.1 is however not defined in Keras. Instead there is a loss function called *multiclass hinge* which is a variation of the Multiclass SVM loss which only takes the most likely incorrect class score into account, instead of all incorrect class scores. This does not work for our problem and probably not very well in general for classification tasks with many labels. This seems quite intuitive since every time the the most likely incorrect class score is trained down another class score takes over and so on, hindering the training process from making any progress. All dimensions i.e. class scores need to be taken into account, always. Fortunately Keras allows you to write your own loss function using Tensorflow, which made it possible for us to implement Multiclass SVM loss as well as the L2 norm version

$$L = \frac{1}{N}\sqrt{\sum_{i}^{N}\sum_{j\neq y_i}^{M} max(0, s_j - s_{y_i} + \Delta)^2}. \quad (33)$$

With $\Delta = 1$ we concluded that the L2 Multiclass SVM loss performs better than the normal one, which agrees with the fact that Sang et al. [5] only presents results with this SVM loss version. The used value for Δ is however not specified, so we tested a few different values for this hyperparameter.

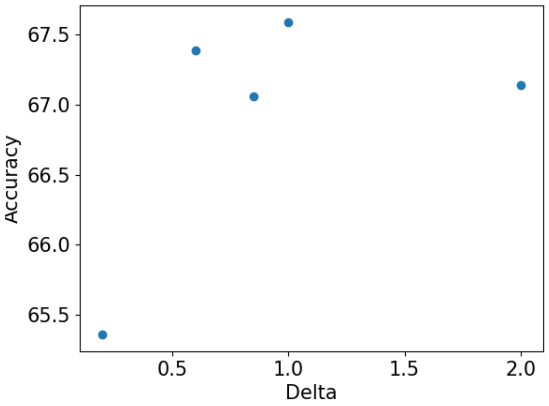

Figure 14: L2 Multiclass SVM loss validation accuracy using BKVGG12 model for different Δ values.

As displayed in Figure 14, our initial value of $\Delta = 1$ was the optimal one of our tests.

The Cross-Entropy and L2 Multiclass SVM loss functions performs almost equally in terms of test

accuracy, where the former seems to have a slight advantage. With everything but the loss function isolated, the Cross-Entropy and L2 Multiclass SVM ended up with highest validation accuracies of 68.23 % and 67.59 % respectively, using the BKVGG12 model. This makes our finding contradictory to the one by Sang et al. [5] regarding what loss function performs the best. However, they reached accuracies roughly 2 percentage points higher than us, which means that there are other factors possibly affecting this outcome. The loss performance difference is however relatively small in both cases and will probably be negligible in a real life application of the emotion recognition problem.

7.1.2 Optimizer

We have tried three different optimizers when running our model training. Two of these are what you would call high performance optimizers, *Adam* and *AdaDelta*, and we have thus only used the default optimizer parameters when running these. The third optimizer tried is the *Stochastic Gradient Descent* or *SGD* for short, which is a less flexible algorithm which for example struggles to break symmetry on *saddle points*. The latter optimizer was used since multiple sources, for example by Sang et al. [5] and Huang et al. [6], have seen improvements in training by using the SGD optimizer in a specific way. Figure 15 shows these symmetry breaks on saddle points.

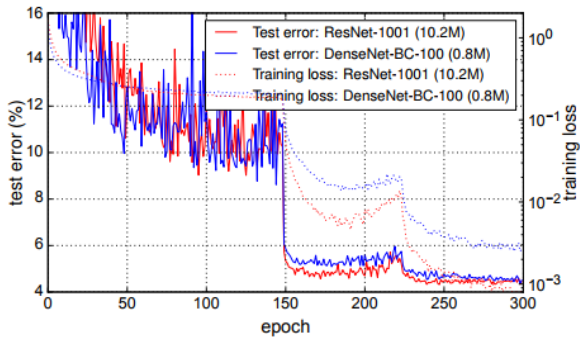

Figure 15: SGD optimizer which breaks symmetry on saddle points. Image courtesy of [6].

The approach by Sang et al. [5] is described as follows:

"During training phase, we use a strategy that decrease the learning rate 10 times if the training loss stops improving. The experiments show that the learning rate is often decreased about 5 times, and the training phase is often finished after about 1400 epochs."

We have tried to apply this technique without success. The statement *"if the training loss stops improving"* is rather vague which makes it hard to exactly replicate the algorithm. We tried an approach where the learning rate is decreased by a factor k if the training loss hasn't decreased at least d in n epochs. It seems as no matter what combinations of k, d and n we use, the same thing happens; after the first learning rate decrease the training just does not improve anymore. Also the training performance at this point is inferior to the other optimizers. This realization made us rather quickly abandon this technique.

The two remaining optimizers seem to be rather equal in their performance, where AdaDelta has been almost negligibly better. On a training setup using BKVGG12 on FER2013 with the optimizer as the only non-fixed variable, the average validation accuracies ended up being 67.78 % and 67.17 % using AdaDelta and Adam respectively. This is for three training runs using each optimizer, which means that another set of training runs could contradict this result since training a model is a non-deterministic process. Based on this we decided to settle on using AdaDelta, and in practise you probably wouldn't notice the small performance difference in the two optimizers.

7.1.3 Dropout

Dropout has been used by a varying degree by the different models; on exactly which layers it has been utilized can be seen in Section 6.3.1.2, 6.3.1.3, 6.3.2 and 6.3.3.

The initial approach involved setting the dropout rate to a fixed probability for all layers in a model, we experimented with different settings in between 0 and 0.5. During this experimentation phase we came to the realization that dropout should most likely be tuned lower on earlier layers. To support this claim we had the following hypothesis:

Big, important features in an image will most likely be extracted in the beginning of a neural network, since we have not yet done any downsampling or other convolutional operations on the input. Therefore we want to be able to save as much as possible of

this initial information, which led us to want to remove the random property of introducing dropout. The solution to this was to add dropout which gradually increases, with a low amount of dropout on the initial layers or Dense/Residual block, and a higher amount on the later layers or Dense/Residual blocks. The final parameter settings for dropout was chosen to [0.05, 0.2, 0.4] for BKStart and BKVGG12, and [0.05, 0.2, 0.2, 0.3] for DenseNets and ResNets.

The tests we conducted to support this hypothesis can be seen in Figure 16. One can see that a gradually increasing dropout do perform better then a static one.

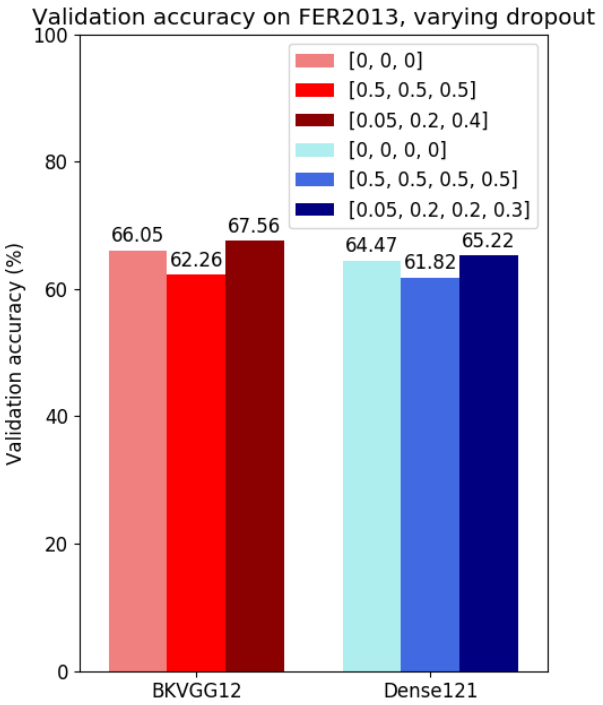

Figure 16: Validation accuracy on FER2013 for BKVGG12 and Dense121 when trained with varying dropout rates.

7.1.4 Model training

To give the reader a better understanding of how the accuracy and loss changes for a model as it trains over several epochs, we've added an image which illustrates how these values are changed over time. This can be seen in Figure 17. Here one can see that after around 200 epochs, the values starts to diverge and the model overfits on the training set.

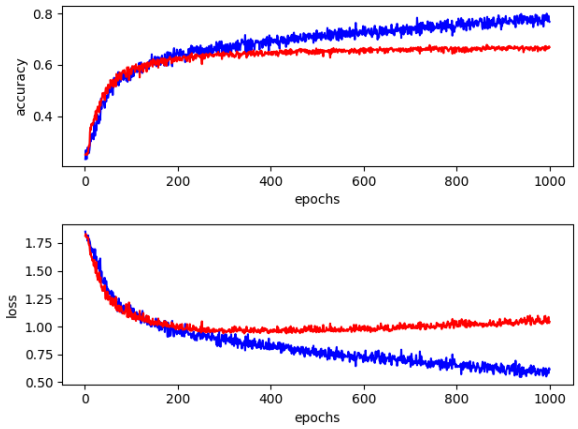

Figure 17: Accuracy and loss for training set and validation set when training BKVGG12 on FER2013 for 1000 epochs. The training accuracy/loss is depicted in blue and the validation accuracy/loss in red.

7.1.5 Label binning

One of the questions defined in the problem definition was whether we could find a real life application for utilizing emotion recognition. Early on in the project we realized that it might be sufficient to just be able to distinguish between the three emotions *positive*, *neutral* and *negative*. Since the datasets we are training against are divided into seven or eight labels, we opted to use label binning.

Using label binning, we merged the labels in FER2013 and AffectNet as shown in Table 7. Our reasoning for putting *surprise* as neutral was that a facial expression conveying surprise can be triggered by both a positive or negative situation, and should therefore be labeled as neither.

Initial tests were conducted to verify whether it would make any difference to train on seven or three labels on FER2013, or to do label binning on a model trained on seven labels. These results can be seen in Figure 18.

Binned label	Anger	Disgust	Fear	Happiness	Sadness	Surprise	Neutral	Contempt
Positive	-	-	-	+	-	-	-	-
Neutral	-	-	-	-	-	+	+	-
Negative	+	+	+	-	+	-	-	+

Table 7: Label binning of the seven/eight emotions present in FER2013/AffectNet.

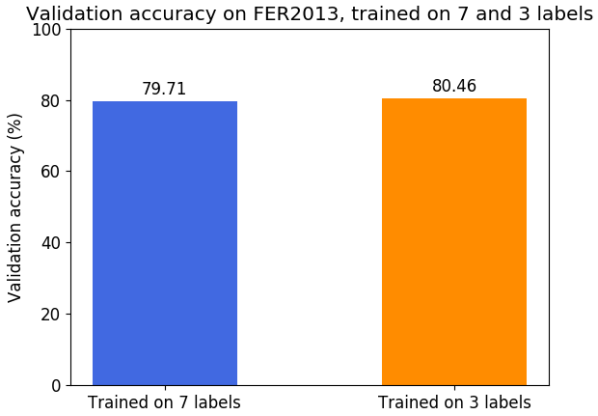

Figure 18: Validation accuracy on three labels when trained on a binned training set versus validation accuracy on three labels when binning is done retroactively.

Since it does not seem to make a difference whether you train on three or seven labels, a follow-up investigation was conducted on AffectNet where *surprise* was excluded from the *neutral* label. This was done to see whether our hypothesis of putting surprise as a neutral label would influence the test results. For comparable results, the effect of omitting the surprise label was investigated on our testset, which are described in Section 7.2, and the results from omitting surprise can be found in Section 8.7.

One final label binning scenario was tested; dividing facial expressions into positive/non-positive. This was done due to the fact that it might be sufficient for a classifier to register whether a person is happy or not.

7.1.6 Choice of models

Once optimal settings for the loss function, the optimizer and dropout was found, we did a test to see how the different models performed against FER2013. This was done to see whether different settings for the models would have a non-negligible impact on the result, or if additional parameters are unnecessary.

As can be seen in Figure 19, the results for different models of DenseNets and ResNets are quite constant with a mean of 65.47 % accuracy 61.28 % respectively. With these results we opted to focus training on one of the DenseNet models, Dense121, and not conduct any more training on any of the ResNet models due to their weak performance. We felt that this choice was justified since we still have the deep neural network model DenseNet to measure the performance on a larger dataset. Dense121 was chosen since it was the smallest of the DenseNet models.

In Figure 20 the number of number of FLOP (floating point operations) needed to predict the label of one input image is plotted against the validation accuracy shown in Figure 19. This further strengthened our choice of conducting less experiments on the larger neural network models, since a lower number of FLOPs with a higher accuracy is sought-after.

7.2 Creating our own testset

Since our models are trained on two different datasets and these do not reflect the real life scenarios we most likely would apply the emotion recognition problem to, we decided to collect a testset of our own. One plausible real life scenario would be the adaption of advertisements in for example shoppings malls, depending on the feedback of the visitors. The dataset FER2013 mostly contains images which are captured straight onto the faces and a lot of them seem to be staged, i.e. the facial expressions are forced, which is unwanted since the classification most likely will be applied to unstaged images. In reality we will not end up with images captured exactly in front of peoples faces, but rather from varying angles, and more so from an upward position due to how surveillance cameras are usually mounted. There are in other words several reasons to why evaluating a model on FER2013 is suboptimal for a real life application. The same goes for AffectNet, although it seems to have a slightly higher

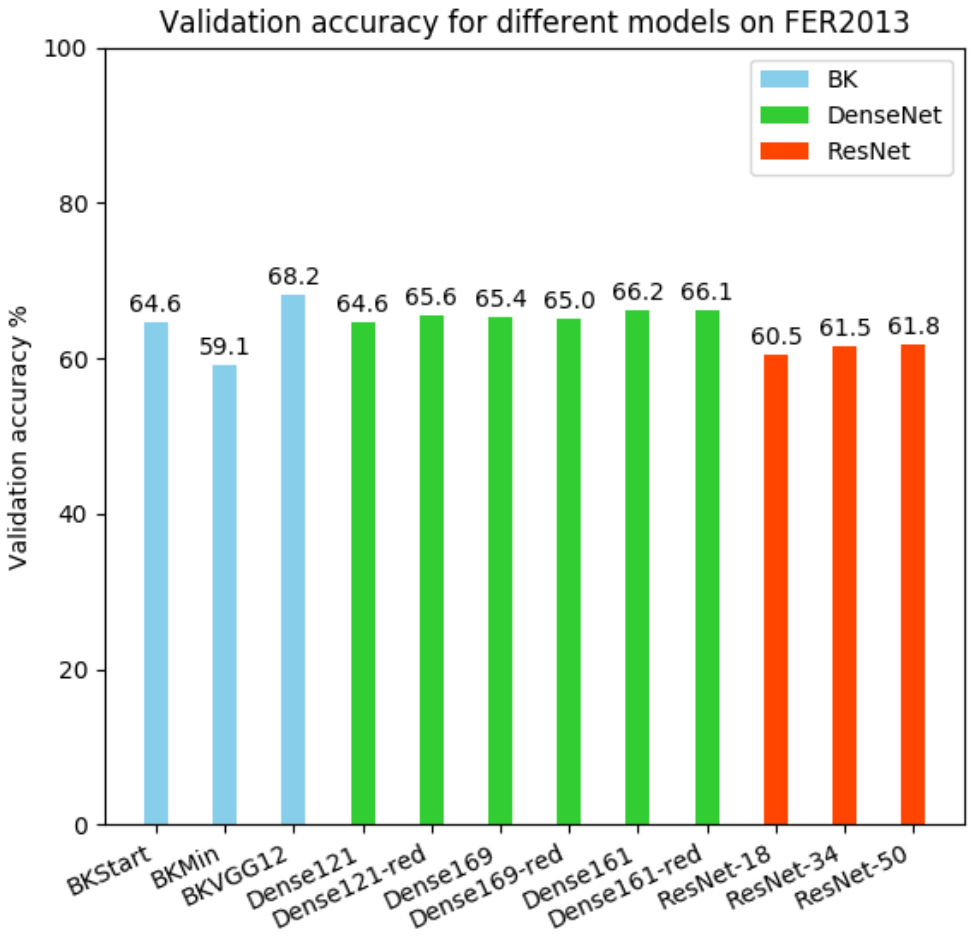

Figure 19: Results for different models when validating against FER2013, using the 7 original labels.

ratio of unstaged images. Simply put, we want a dataset to benchmark our models with which is as close as possible to what you would end up with in a real life application. This means that the camera angles should mimic those of normal surveillance cameras and the facial expressions should be unstaged. Furthermore we chose to annotate this dataset using three labels; negative, neutral and positive, since our hypothesis is that this is more useful in a real life application.

Fortunately we did not have to produce new images for our testset, but were able to go through some datasets our employer already had. From this dataset we extracted 258 images which displayed seemingly unstaged faces with reasonable facial expressions, i.e. nothing too extreme. Roughly half of the extracted images are captured by actual surveillance cameras mounted accordingly in our employers building. The other half are images similar to those of AffectNet since they are taken from the internet, but they are as mentioned not staged. Since some of our models have an input size of 100x100 pixels while not all the 258 images are this big after facial cropping, we had to omit some images. This resulted in a testset of 192 images. Furthermore we wanted to have a balanced testset, meaning that there are the same number of examples for every label. The least represented label was negative with 35 images, which meant that we had to omit images with other labels. After doing this in a random manner we ended up with a final testset of 105 images. Since we are no annotation specialists and also since it greatly increases the credibility of the annotation, we let 9 people, including ourselves, individually annotate the testset. The labels to choose from were negative, neutral and positive. The testset was then labeled using the most chosen label for every image. Fortunately, there were no ties in the label voting. By dividing the number of "correct" label votes by

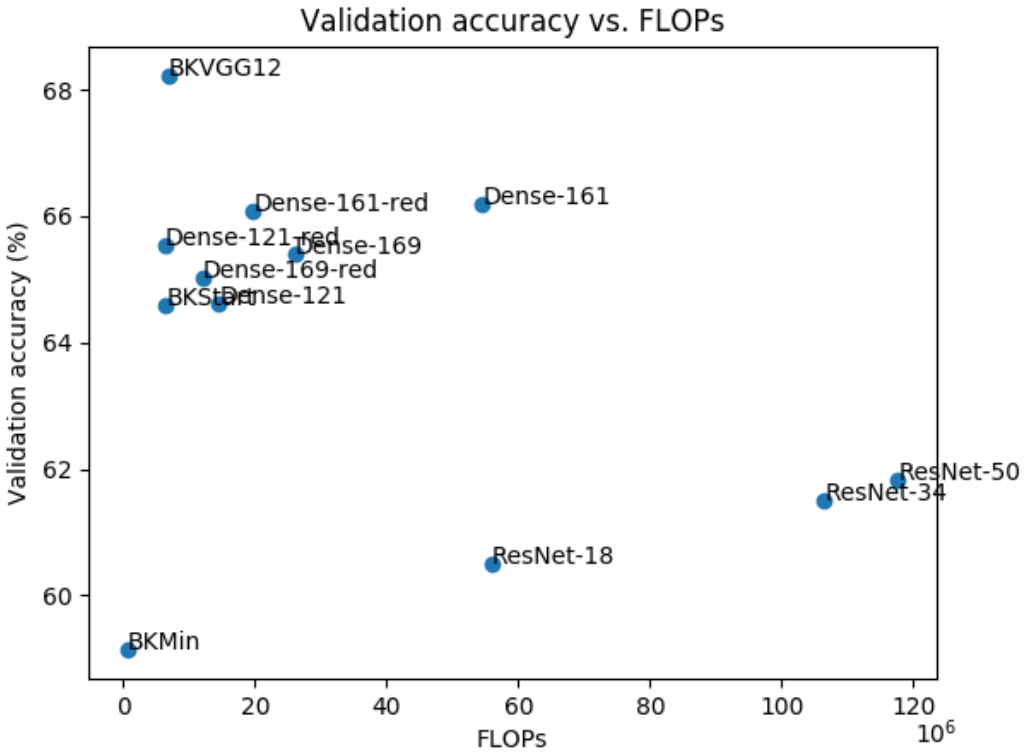

Figure 20: Validation accuracy on FER2013 plotted against the number of FLOPs needed to run one prediction.

all label votes, we could define an "agreement accuracy" which ended up being 85 %.

7.3 Tests on IRIS

As stated in Section 7.3, we want to investigate whether thermal images could be a viable alternative to visible images on the emotion recognition problem. Furthermore, the IRIS dataset consists of color images of 320x240 pixels which allows the opportunity to test the impact of color channels and image resolution. Training has thus been run on grayscale and RGB color channels for the visible dataset and grayscale images for the thermal dataset as seen in Figure 21 and 22. Test results on 42x42 versus 100x100 pixels are found in Figure 23. Since the face detection algorithm used can't find faces in the thermal images, no images have been cropped.

When evaluating visible vs. thermal images and 42x42 vs. 100x100 pixels, grayscale images were chosen.

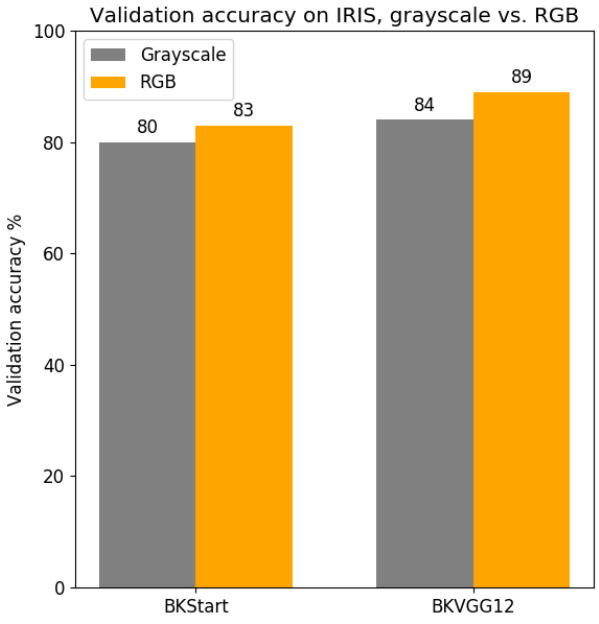

Figure 21: Validation accuracy of BKStart and BKVGG12 when evaluating the impact of color in visible images.

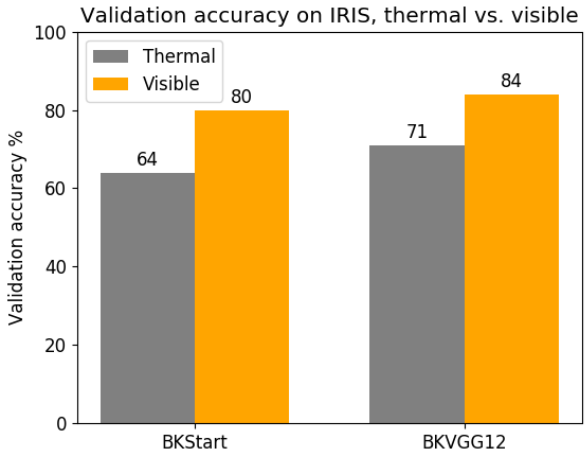

Figure 22: Validation accuracy of BKStart and BKVGG12 when evaluating thermal versus visible images in grayscale.

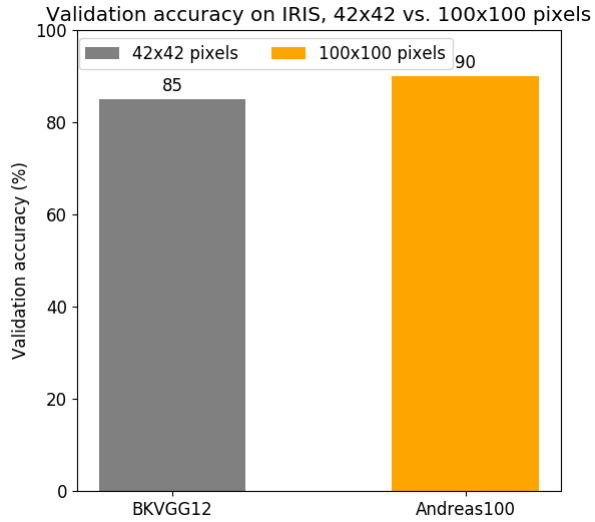

Figure 23: Validation accuracy of BKVGG12 and its 100 pixel counterpart, Andreas100, trained on visible grayscale images.

It is quite clear that for this dataset, the results are better using the RGB color channels compared to a single grayscale channel. The visible images also have a significant advantage on the thermal images, using these models. Finally a higher image resolution seems to increase the model accuracy. The IRIS dataset is however relatively small and one needs to be careful drawing conclusions only based on such a small amount of data. When it comes to the research on thermal images, which for us was a bit of a side track, we abandon it here since training on these images clearly show a lower performance on this only thermal dataset we have. Color channels and higher resolution images seems promising and we will thus investigate these parameters further on a bigger dataset.

7.4 Model optimization against AffectNet

One of the biggest challenges encountered with this thesis is the sheer number of parameters one can fine tune to reach higher results. As mentioned in Section 7 our approach to solving this problem has been to isolate one parameter at a time, find an optimal value, and then fix that parameter to find an "optimal configuration" using FER2013. AffectNet is about 14 times bigger than FER2013 and we have thus tried to optimize only a few, most important settings. These are first and foremost; color versus black and white images, image size, dropout and training on three labels where the surprise label has been omitted.

Since the images in AffectNet are in color, further investigation on color versus black and white was an obvious choice. The images are also larger than the 48x48 pixel images in FER, which made investigation on image size a choice. Finally, *no surprise* was investigated due to the reasons clarified in Section 7.1.5.

When training on AffectNet, the dropout amount was decreased. It was set to [0, 0, 0.5] for BKStart, BKMin and BKVGG12, and [0, 0, 0, 0] for DenseNet and ResNet. This was chosen due to the reasoning that with a bigger dataset the model should be less prone to overfitting, thus less dropout is needed. The first few training runs on AffectNet used the dropout settings optimal for FER2013, which proved inferior on AffectNet compared to above mentioned values. For example using BKVGG12 isolating the dropout values, [0, 0, 0.5] outperformed [0.05, 0.2, 0.4] with accuracies 39.63 % and 38.0 % respectively. Testing more dropout settings than this on AffectNet was not a priority and since a training on AffectNet usually takes two days there was simply no time to investigate this further.

7.5 Demo Application

7.5.1 Background

As a final part of the thesis project we decided to implement a demo application to act as an example and inspiration for anyone thinking of applying emotion recognition on a real life problem. The application will collect images from a network camera, predict emotions on any detected faces (by OpenCV's face detection) in the images, and display the video stream with facial bounding boxes and corresponding emotions in real time.

For this task we decided to use a trained model which has a good balance between performance and required computations, since we want it to be as fast as possible without losing significant accuracy. However, even for our smallest models, running face detection and emotion prediction embedded on a camera is most likely not possible in real time. That's why we decided to let a desktop computer with GPU support run the application, while anyone on the network can access the video stream through a web client.

In order to get a smoother real time prediction we wanted to analyze multiple images before presenting a prediction. To be able to do this, one needs to keep track on which face corresponds to which in different frames. This is why we implemented a very simple tracking algorithm, which pairs together faces frame by frame, by looking at the facial bounding box center values. If a value is present in the next frame which is within a given euclidean range from present value for a given bounding box, then these two faces are assumed to be the same. Otherwise the tracker loses that face until the next prediction iteration. A "lost" face will results in an empty prediction, i.e. no prediction, unless the tracker already had picked up enough images of the face before losing it to make a prediction that prediction iteration. How often a new label is presented and how many images there are in a prediction iteration can be decided by the user.

7.5.2 Application Features

The user of our application has a range of different settings to play around with, to be able to optimize the performance for different conditions:

- **Camera**. Any camera connected to the network can be chosen for the analysis.
- **Emotion spectra**. The user can chose between a full range of emotions, or a binned version with three possible emotions; negative, neutral and positive.
- **Prediction Sense**. If the predicted label does not have a probability greater or equal to this value, the label is automatically changed to *uncertain*.
- **Tracking Sense**. The maximum euclidean distance in pixels between a face in two consecutive frames.
- **Prediction Frequency**. Every so many images predictions are made.
- **Label Frequency**. Every so many images a new label is presented.

8 Results

8.1 Model setups

In the following section most parameters have been fixed in order to evaluate the impact of other settings, such as color, image size and label binning configuration. As mentioned in Section 7.1 Cross-Entropy loss was chosen, AdaDelta was picked as the optimizer, dropout was set to [0.05, 0.2, 0.4] for BKStart, BKMin, BKVGG12 and Andreas100 when training on FER2013 and [0, 0, 0.5] when training on AffectNet, and [0.05, 0.2, 0.2, 0.3] for DenseNet when training on FER2013 and [0, 0, 0, 0] when training on AffectNet. Dense121 was trained on both images of sizes 42x42 pixels and 100x100 pixels, and is therefore denoted Dense121 and Dense121-100 in the following section. Both Andreas100 and Dense121-100 were trained on RGB images unless stated otherwise.

8.2 Validation accuracy on AffectNet

Figure 24 shows the validation accuracy of our models trained and validated on AffectNet. The DenseNet models perform slightly better on the validation problem with eight labels, where as DenseNet trained on 100 pixels performs the best with 48.58 % accuracy.

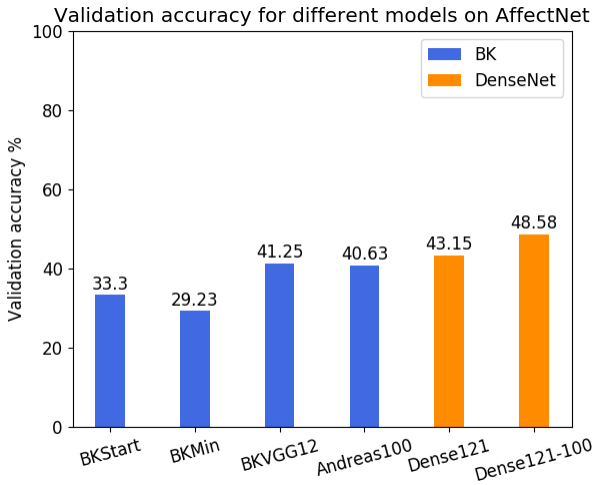

Figure 24: Validation accuracy of our models trained and validated on AffectNet.

8.3 Confusion matrix on AffectNet

Table 8 shows the confusion matrix for BKVGG12 when validating against AffectNet. This means that the correct label is written in the first column, and the guesses which are made on each individual label is shown for that row in the other eight columns. The numbers in bold are therefore correct predictions for each label. Something to notice here is that the model is very good at predicting the labels Neutral and Happiness, where 73.4 % and 94.4 % respectively are guessed correct. Another interesting observation which can be made is that almost no predictions are made on Contempt for the entire validation set.

8.4 Grayscale vs RGB

Figure 25 shows the results when evaluating the impact of training on grayscale versus RGB images. As can be seen, the color does not seem to have any impact on the result what so ever.

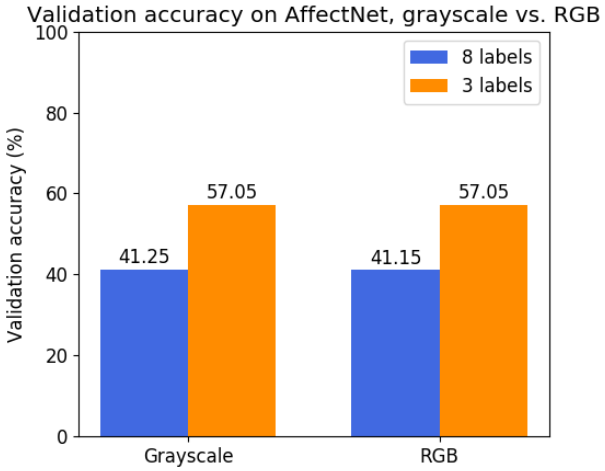

Figure 25: BKVGG12 trained on 42 pixel of AffectNet, grayscale images versus RGB images. The two label categories are standard 8 labels, and binned three labels; positive, neutral and negative.

8.5 42 vs 100 pixels

Figure 26 shows the results from training DenseNet models on 42x42 and 100x100 pixels. Higher resolution does provide a higher accuracy, something

Ground truth \ Prediction	Neutral	Happiness	Sadness	Surprise	Fear	Disgust	Anger	Contempt
Neutral	**0.734**	0.15	0.05	0.02	0.012	0	0.034	0
Happiness	0.052	**0.944**	0	0.004	0	0	0	0
Sadness	0.384	0.094	**0.412**	0.014	0.028	0.002	0.066	0
Surprise	0.344	0.228	0.044	**0.252**	0.106	0.002	0.024	0
Fear	0.25	0.078	0.104	0.1	**0.362**	0.014	0.092	0
Disgust	0.252	0.166	0.106	0.03	0.016	**0.136**	0.294	0
Anger	0.356	0.068	0.062	0.016	0.024	0.016	**0.458**	0
Contempt	0.284	0.636	0.02	0.002	0.	0.008	0.048	**0.002**

Table 8: Confusion matrix for BKVGG12 against the validation set of AffectNet. Each row shows how often the labels in each column have been predicted for that given ground truth. The correct predictions are found along the diagonal in bold text.

that confirms the results made in Figure 23, Section 7.3.

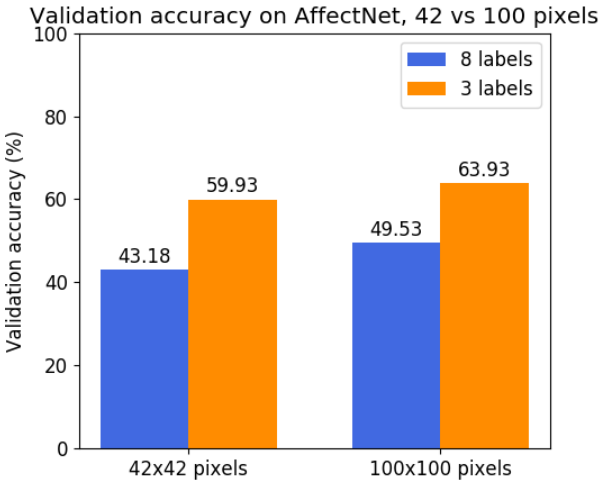

Figure 26: Dense models trained on grayscale 42x42 and 100x100 pixels of AffectNet. The two label categories are standard 8 labels, and binned three labels; positive, neutral and negative.

8.6 Testset results

8.6.1 AffectNet trained models

In Figure 27 the performance of models trained on AffectNet on our testset can be seen. Things worth to notice is that the Dense121 100x100 pixel performs marginally better on the three class classification problem than the rest of the 42x42 pixel models.

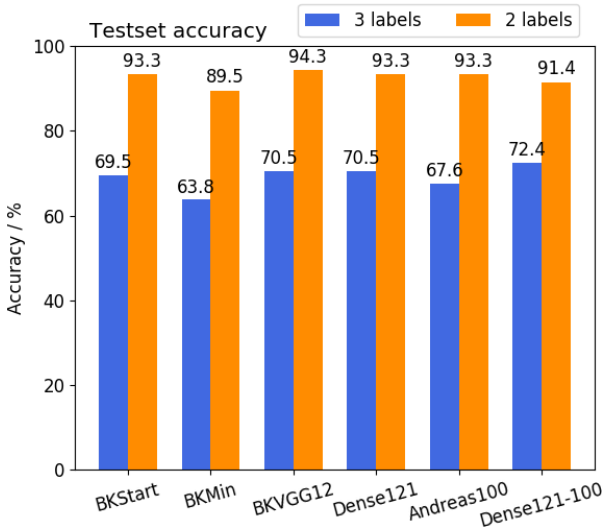

Figure 27: AffectNet trained models evaluated on our testset for 3 labels; negative, neutral and positive, and 2 labels; non-positive and positive.

8.6.2 FER2013 trained models

In Figure 28 the performance of models trained on FER2013 on our testset can be seen. Worth noticing is that the more shallow neural network BK-models perform equally good or better than the deeper neural network models of DenseNet.

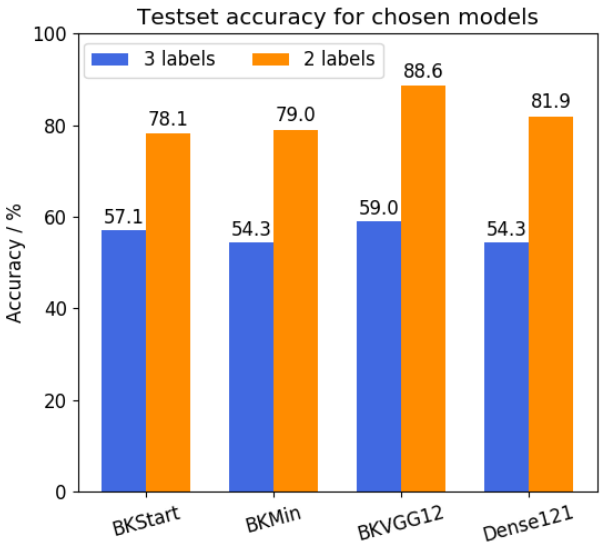

Figure 28: FER2013 trained models evaluated on our testset for 3 labels; negative, neutral and positive, and 2 labels; non-positive and positive.

8.7 Omitting surprise label

As mentioned in Section 7.1.5, we investigated how omitting the surprise result would affect the classification accuracy. To do this, we compared the testset accuracies of our models trained on AffectNet with and without surprise label in the training set. The results are presented in Figure 29.

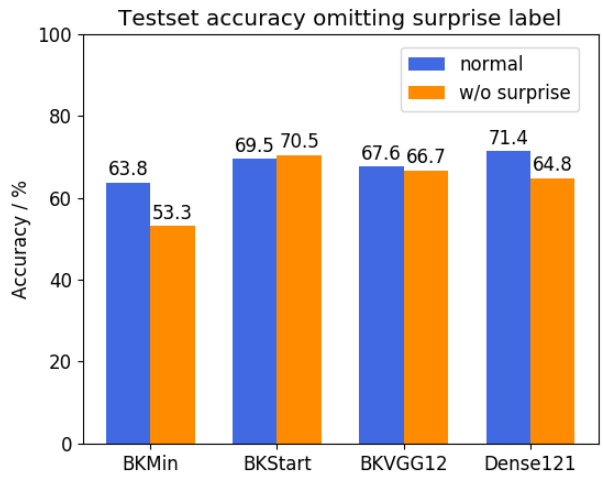

Figure 29: Testset accuracy investigating the effect of removing surprise label. All models have been trained on AffectNet, 42x42 pixels, grayscale images and the presented accuracies are for 3 labels.

8.8 Omitting pixel normalization

Since our demo doesn't have a given dataset to predict on, but a constant flow of new images, the pixel normalization part in the preprocessing step could not be performed in the same manner as before. Instead we chose to completely drop this normalization step which actually seems to slightly improve the classification performance, at least for the few configurations we've tested. For example using our optimal settings for BKVGG12 on FER2013, the best validation accuracy during a training was 68.34 % and 67.78 % without and with pixel normalization respectively. Similarly for BKStart on AffectNet, we could see an increase in validation accuracy from 33.3 % to 34.5 %. The same applies for Dense121 with an increase from 43.2 % to 45.5 %. To get a final verdict on whether omitting pixel normalization is a gain or a loss for us, we investigated this effect on our testset, as seen in Figure 30.

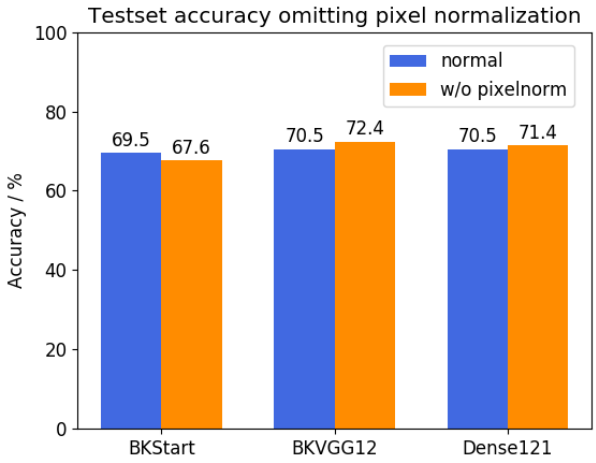

Figure 30: Testset accuracy investigating the effect of pixel normalization. All models have been trained on AffectNet, 42x42 pixels, grayscale images and the presented accuracies are for 3 labels.

8.9 Demo application

In Figure 31 we have printed both the eight and three class labels for clarity. The BKVGG12 model trained on AffectNet without pixel normalization has been used for the analysis.

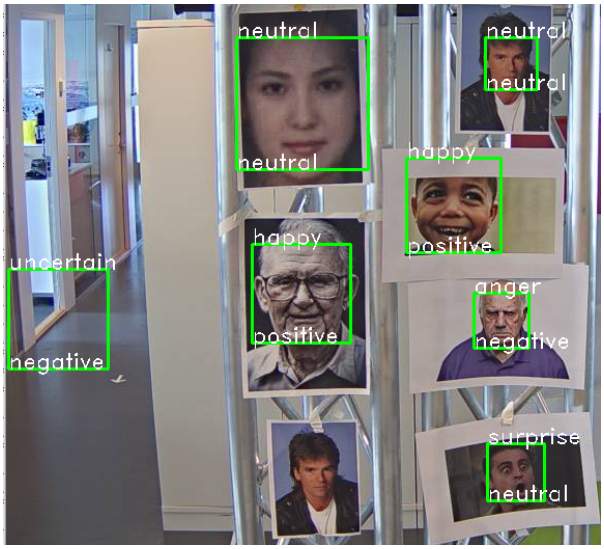

Figure 31: Demo application screen shot.

The faces in the image have rather clear facial expressions and it is no surprise that the model seems to classify them all very well. The effect of the camera angle is evident since the missed face in the bottom is found in the top where the angle is more straight on. In general, face detection algorithms seem to be quite bad at detecting faces from deviant angles. Lastly, there is the rather conspicuous effect of mistaking a doorstep for a face. Sometimes the face detection returns these false positives which potentially could harm for example a statistical recording. However, this problem could be reduced by calibrating the camera angle since most often the background is more or less static.

In general, we are very happy with the performance of our application. The biggest practical issue would be the face detection performance, which is not part of our core problem - emotion recognition. It has become quite clear that the "happy" label is the most consistent label in terms of prediction probability for eight classes. For example if the prediction sense parameter is set to 70 %, then in general all facial expressions result in the "uncertain" label except for "happy". A good compromise seems to be a prediction sense of 30-40 %, i.e. this results in a nice and seemingly correct experience for the viewer. The same goes for 3 and 9 for prediction and label frequency respectively. Lower values tend to drive the stream out of "real time" rather fast, due to the analysis lagging behind.

9 Discussion

9.1 Dataset size

It has become quite clear during this thesis work that dataset size is extremely important. Looking at Figure 28 we see that for three labels no model reaches a test accuracy of more than 59 %. These models have been trained on FER2013 which contains roughly 29 000 training images. The AffectNet dataset we've used contains over 287 000 training images and the corresponding testset results can be seen in Figure 27. Here all models perform well above 60 % with a highest accuracy of 72.4 %. The same trend can be seen for the 2 label accuracies. In summation, a larger dataset seems to be more important than choosing the right model, even though this of course also is important. This conclusion matches the one made by Domingos, P. in [24], where he states that:

"As a rule of thumb, a dumb algorithm with lots and lots of data beats a clever one with modest amounts of it".

The reason for this is quite intuitive and is easily explained using the metaphor from Section 5.1.2. You train a model much like how you teach a child. With a child you point at a horse and say "horse" and with a computer model you give it an image of a horse together with a label that says "horse". Let's say the problem is to classify animals. The more different animals and specimens the child sees, the better will it learn to classify them. It is a process which with proper teaching (correct labels) leads to more knowledge, i.e. better classification performance. As we see it, a classifier should in theory be able to perform better with increasing dataset size, no matter the dataset size. However, a classification model only has so many parameters and can thus only learn a limited amount of information, just like a human. This should mean that in practise a model cannot improve indefinitely with increasing dataset size. Furthermore, simple classification problems might be more or less perfectly solved before the dataset size reaches extreme values.

Another noteworthy effect of dataset size appears when comparing the performance of models trained using grayscale versus RGB images. One of the things mentioned in Section 7.3 was that the results from training on color images of IRIS seemed to contribute to a higher validation accuracy. When the same test was conducted on AffectNet the results seemed to say something else, see Figure 25. On AffectNet, grayscale and RGB images performed equally well, leaving us with the conclusion that the color images have no additional information about a person's facial expression. However, since the mentioned IRIS tests claimed the opposite, it seems as if you need a certain amount of training data for color and grayscale to perform equally well. A training set of RGB images technically have three times the information of corresponding grayscale images. Even though most of this extra data could be useless for the emotion recognition problem, for a small enough training set it might have an influence and boost the performance noticeably. This is our best explanation for these results.

9.2 Pixel normalization and image size

Looking at Figure 30, it seems as omitting the pixel normalization step from Section 6.2.2.2 slightly increases the testset accuracy. In theory it should rather be the opposite since normalizing, i.e. centering and fixing the variance of the data is done to aid the network in learning. For example using the *ReLU* activation function you want your data close to zero to fully utilize the non-linearity, which in general helps to model more complex problems. However, when a model uses batch normalization (explained in Section 5.4.4), for example DenseNet, image normalization steps are redundant. This makes the Dense121 model results in Figure 30 even more inexplicable since here the data is normalized either way. In summation, we have no theory supporting increased performance by omitting pixel normalization, this is probably just a coincidence.

As for higher resolution images granting higher accuracies, which can be seen in Figure 26, the results are a bit easier to interpret. First of all, with higher image resolution you can fit more information within an image; an image of input size 100x100 pixels contains about 5.7 times more pixels than an image of size 42x42 pixels. Secondly; the results on image size originates from DenseNet models. Since DenseNet is optimized against ImageNet with input size 224x224 pixels, it is reasonable to believe that the 100x100 images should fit the model better.

9.3 An unbalanced training set

One of the obstacles one can encounter in a machine learning problem is an unbalanced training set. If one of the classes in a training set is more common, then the model will reach a higher accuracy if it predicts that class more often when unsure; the model will become biased. This is something that you generally want to avoid; even though a facial expression showing happiness might be more prevalent, the predictor should still be unbiased in its guess.

If we take a look at Table 2 we can see that some classes are heavily unbalanced. There are for example almost 32 times more images of facial expressions showing happiness than facial expressions showing contempt, where happiness is 1.8 times more occurring than the second biggest label neutral. This can be connected to the results shown in Table 8. Here happiness is the label guessed correctly most of the times. This could be because happiness is a very distinct facial expression, i.e. the network manages to extract good features from happiness but not to the same degree for other labels, and/or because happiness is more prevalent in the training set.

This is also most likely the reason for the low accuracy on the label *contempt*. Contempt is the label with the fewest number of images in AffectNet at 4250. It is however closely followed by *disgust* at 4303 images, which reaches 13.6 % accuracy. Confusion matrices on other models have similar results, but better models reach around 7 % accuracy on contempt. It is therefore probably not an implementation issue, but rather that the dataset is unbalanced combined with that the CNN's don't manage to pick up any good features for contempt.

9.4 Evaluation of models

One of the more interesting observations that can be made is the performance by the different models. All models with the prefix BK could be considered shallower CNN's, while DenseNet and ResNet can be considered deeper CNN's. As can be seen in Figure 19, our deeper neural network models don't perform better than the shallower ones (where ResNet performs well below BKVGG12) on FER2013. On AffectNet, see Figure 27, the results have changed slightly where the DenseNet121 trained on 100 pixel RGB images performs the best on the three class problem. However, they all perform relatively similarly.

The reasons for this could be many, but our theory is that this is mainly because of one or two things:

When the creators of DenseNet and ResNet tested their models, they were benchmarked against the VGG16 model, which the BKVGG12 model is based on. In both articles by Huang et al. and He et al. [6] [7] DenseNet and ResNet performed better than the VGG16 model, but in our case it performs equally good or even better. The first thing to notice is that these neural networks were trained and tested against ImageNet. The images in ImageNet, which contains a total of 1000 classes, are considerably different from the images and the problem of classifying facial expressions. Therefore it might be satisfactory to utilize a smaller neural network model, and still achieve good accuracy.

The second reason which we believe could be a culprit is the fact that DenseNet and ResNet are constructed to classify higher resolution images. The images in ImageNet are of the resolution 224x224 pixels, which is well above our maximum of 100x100 images. Since we have chosen not to focus on creating an optimal neural network model for this specific problem, it might be that the designs of DenseNet and ResNet are not optimized for the images we have.

9.5 The human error

One must remember that the goal working with the emotion recognition problem should not be to reach a "perfect" 100 % classification accuracy. While many classification problems like the ImageNet object recognition problem are highly unambiguous to annotate, the emotion recognition problem is not. In a lot of cases people will not agree on the same emotion label and annotation is thus rather subjective. Consequently, a perfect 100 % classification accuracy could just mean that the model has overfitted to the annotator's opinions. If that annotator happens to be deviant in their facial expression interpretation, then an objectively good model could perform worse than an objectively bad one. As we see it, this problem can be reduced to some extent in two ways; using professional annotators and/or using multiple annotators. The former is applied on AffectNet and the latter on our own testset.

Even though a given AffectNet image is annotated by only one person, they did investigate the agree-

ment among annotators by letting two of them annotate a subset of AffectNet. The two annotators agreed on 60.7 % of the images, with 11 possible labels. Even though this might sound like a poor agreement, we believe it to be very difficult for non-professionals to achieve this level of agreement with that many labels. Let's for example compare it to the agreement of the 9 annotators on our own testset. In Section 7.2 we defined an agreement accuracy as: number of "correct" label votes divided by total number of label votes. The agreement accuracy on our testset was 85 % using three possible labels. The AffectNet agreement of 60.7 % converted to this metric is 80.4 %, using 11 labels. Whether this metric is a fair comparator could be discussed, but the point is that the AffectNet label agreement is very good; the agreement accuracy is almost as high as for our testset but with more than three times the labels. We believe that AffectNet is the best dataset publicly available for the emotion recognition problem thanks to its size and annotation quality - the human error has been minimized.

9.6 Human vs AI

An interesting test, maybe the most interesting of them all, is to benchmark our model against a human. To begin with, we'll compare the validation accuracy of our top performing model, Dense121-100 trained on AffectNet grayscale images. This model had a validation accuracy of 49.53 % which can be seen in Figure 26. Comparing this to the agreement that the professional annotators had on AffectNet, 60.7 %, the neural network performance is not too far off.

Secondly, we would like to compare our model against an annotator on our testset. To do this, we chose one person out of the 9 annotators to omit from the annotation so that this person's classifications accuracy could be determined. This person was neither the most nor least deviant among the annotators. The person's accuracy turned out to be 80.7 %. The best model from Section 8.6.1 was Dense121-100 (trained on color images), which achieved 74.6 %. The performance difference between an average human and a neural network can thus be rather small on the emotion recognition problem, at least for our interpretation of it. Chances are also that this difference will decrease with increased computational power. As we see it, an AI performing as well as a human is definitely realistic, which basically is the goal.

An additional thing to mention when studying human emotions is that we as human beings rarely interpret the mood of other people from a snapshot; it is mostly based on a sequence of images, body language and the current situation. Therefore our machine learning models can be seen as competitors when trying to analyze single images, but for interpreting human emotions as a whole, a human still has to be seen as the expert.

9.7 Label binning methods

As mentioned in Section 7.1.5 one of the areas we wanted to further investigate was how different label binning methods could impact the accuracy of our CNN models. As mentioned, one of things we were uncertain about was whether surprise should be labeled as a neutral facial expression or not. One could after all advocate that a surprised facial expression could originate from both a positive and negative reaction.

To investigate this we conducted training on our models where the surprise label was removed from the training set, to see how these models performed against models which had seen the surprised facial expression. This can be seen in Figure 29. Even though it's not possible to draw any final conclusions from this figure since the results are not unanimous, it seems as if the models trained on surprise images does perform better than the models who are not. This is most likely because of one of two things. First of all, the models trained on images depicting a surprised facial expression have had more data to train against, and could therefore have managed to recognize facial expressions more generally. Secondly, this result could be an indication that binning surprise as neutral is reasonable, thus giving models trained with surprise images an obvious advantage.

As a final argument on label binning, we believe that it in most cases is sufficient to condense the standard seven/eight facial expressions into the three labels *positive*, *neutral* and *negative*. By doing so you can easily eliminate unnecessary uncertainty. Knowing whether a person is disgusted or angry probably makes little difference for an application user. It is most likely sufficient to know that the person conveys a negative facial expression.

9.8 Demo application

For our in Section 8.9 mentioned frequency settings there will always be three predictions contributing to the presented label, i.e. the probability of encountering a tie is somewhat minimized. If a tie happens anyway, our implementation handles it by choosing the first possible "winner" which is not optimal, but could easily be changed if wanted. As mentioned in Section 8.9, the "happy" or "happiness" label most often has the highest prediction probability; the network is very confident when predicting happiness. This agrees with the fact that this label always seems to have the highest prediction accuracy, which for example can be seen in Table 8. As mentioned in Section 9.3, we believe there are two main reasons for this. First of all, as seen in Table 2, "happiness" is by far the most frequent label in the AffectNet dataset. This means that a model trained on AffectNet most likely will become somewhat biased since it becomes advantageous to predict "happiness" when uncertain. Second of all, being happy might display relatively distinct visible features such as a certain shape of the lips combined with displaying teeth which results in high contrasts in this area.

9.9 Practical application considerations

When working with the emotion recognition problem "in-the-wild", theoretically optimal solutions might not be the best choices. For example would it in theory be motivated to use as big images as possible with all color channels for analysis, to maximize the available information. This would go hand in hand with deeper network architectures which would be able to pick up more features. The obvious problem with this is the computational effort required to train such a model and to use it for predictions. One must find a sensible balance between algorithm speed and prediction accuracy, which we seem to have found in BKVGG12 as an example. In Figure 27 BKVGG12 is just 1.9 percentage points less accurate than Dense121-100 for three labels. BKVGG12 uses grayscale images of 42x42 pixels while Dense121-100 uses RGB images of 100x100 pixels. Furthermore, BKVGG12 requires 6.93M FLOPs to predict one image, while Dense121-100 needs 16.6M FLOPs. Although dependent on the use case, the gains in accuracy are probably not worth the increased running time. Additionally, another drawback of using a higher image resolution is that the face detection would find less faces since a face needs to be closer to the camera to be detected.

One big drawback of our demo application is as mentioned earlier that it only works from a very straight on angle. If the angle is a bit off, the face detection algorithm will miss faces and such a face doesn't even reach the analysis step. Even if it did, the prediction performance would most likely be inferior since our models have been trained using datasets with mostly "straight angles". So in order to solve the angle problem, one would have to build or find a face detector capable of detecting faces from varying angles as well as train a model with facial images from varying angles. We don't know how much work this would require, but it is definitely an interesting problem that might be worth trying to solve if one wants to build an application of this kind.

9.10 Possible use cases

One of the questions we asked ourselves in the problem definition was what real life applications which could emanate from the emotion recognition problem. The following section should not be seen as final answers to that question, but rather as thoughts and reflections on where we as authors think that the problem might be applicable.

One of the more obvious areas of use for recognizing people's emotions would be within the commercial sector. A tool which can instantaneously measure a group of people's response to for example products, commercials etc. could be highly useful. However, one should not overlook the potential inaccuracy in such data. Even though a customer in a shop might be happy while he/she is browsing wares, it does not have to mean that the reason for expressing a positive facial expression stems from a positive shopping experience.

Instead, we think that the emotion detection problem is more useful when it comes to measuring spontaneous reactions. If the user would be able to for example measure a crowd reaction during a funny or scary part of a movie, then the producer of the movie would be able to evaluate how well a certain scene lives up to its purpose.

Another potential area of use which lies further on the horizon is within the synergy between human

and AI. As machines will be utilized in areas where they need to be able to interpret human responses, the ability to understand facial expressions could be highly useful.

10 Conclusion

The emotion recognition problem does indeed seem solvable by a neural network. The models trained in this thesis achieve relatively close to human performance when tested on our three label testset, 74.6 % versus a human performance of 80.7 % accuracy. When comparing with eight labels instead of three, the difference is slightly higher where our top performing model Dense121-100 on grayscale images reaches 49.5 % validation accuracy on AffectNet and the agreement of professional annotators on a subset of AffectNet is 60.7 %.

As for possible use cases for the emotion recognition problem, we've deduced that it in most cases might be sufficient to be able to interpret three labels; *positive*, *neutral* or *negative* facial expressions. Although the recognition of facial expressions could be used in a number of different areas, we believe that it is best suited for situations where you want to measure the response of a sudden event, such as the crowd's reaction during a specific scene of a movie, or a potential customer's response to a commercial.

When it comes to trade-offs between prediction accuracy and execution speed, we've concluded that models does not have to be necessarily deep in order to perform well on the emotion recognition problem. The more shallow BK-models perform insignificantly worse than the deeper DenseNet models while executing significantly faster. Some gains in accuracy can be made if larger images are analyzed, but it is questionable if this small performance improvement is justifiable when set in comparison with the extra computational power needed to classify an image.

11 Future work

If we were to continue this work, there are a few things we'd like to look into more. For example, the DenseNet and ResNet models are designed for an input size of 224x224 pixels. We have only used them for 42x42 and 100x100 pixels and it would be very interesting to see how they perform using the image size they were built for. As camera resolutions increase and computers get more powerful, it also becomes more reasonable to analyze bigger images.

There are some things which possibly could be improved, for example getting the SGD optimizer approach described in Section 7.1.2 to work since this seems to have been a successful approach for multiple authors. Also as mentioned in Section 9.9, we'd like to improve our application to work for faces of varying angles.

To reach a wanted product it would be interesting to speak to potential users to see exactly what functionalities and features are of interest to them. Also combining our application with already existing functionalities created by our employer, we could record emotions together with gender and age for more valuable statistics. For practical purposes we would also like to investigate whether our application could be run embedded on a camera with acceptable performance.

References

[1] W. McCulloch and W. Pitts. "A Logical Calculus of the Ideas Immanent in Nervous Activity". In: *Bulletin of Mathematical Biophysics* 5 (1943), pp. 115–133.

[2] P. Ekman and W. V. Friesen. "Constants Across Cultures in the Face and Emotion". In: *Journal of Personality and Social Psychology* 17(2) (Feb. 1971), pp. 124–129.

[3] Y. LeCun et al. "Gradient-based Learning Applied to Document Recognition". In: *Proceedings of the IEEE*. Vol. 86. 11. Nov. 1998, pp. 2278–2324.

[4] I. Cohen et al. "Facial expression recognition from video sequences". In: *Computer Vision and Image Understanding* (2002).

[5] D. V. Sang, N. V. Dat, and D. P. Thuan. "Facial Expression Recognition Using Deep Convolutional Neural Networks". In: *Knowledge and Systems Engineering (KSE) 9th International Conference*. Oct. 2017, pp. 130–135.

[6] G. Huang et al. "Densely Connected Convolutional Networks". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVP)*. July 2017, pp. 2261–2269.

[7] K. He et al. "Deep Residual Learning for Image Recognition". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVP)*. June 2016, pp. 770–778.

[8] K. Zhang et al. "Facial Expression Recognition Based on Deep Evolutional Spatial-Temporal Networks". In: *IEEE Transactions on Image Processing* 26 (Sept. 2017), pp. 4193–4203.

[9] H. Meng et al. "Time-Delay Neural Network for Continuous Emotional Dimension Prediction From Facial Expression Sequences". In: *IEEE Transaction on Cybernetics* 46 (Apr. 2016), pp. 916–929.

[10] N. Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15 (2014), pp. 1929–1958.

[11] K. Fukushima. "Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position". In: *Biological Cybernetics* 36 (1980), pp. 193–202.

[12] S. Ioffe and C. Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. URL: https://arxiv.org/pdf/1502.03167.pdf.

[13] K. Simonyan and A. Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. URL: https://arxiv.org/abs/1409.1556.

[14] R. K. Srivastava, K. Greff, and J. Schmidhuber. *Highway Networks*. URL: https://arxiv.org/abs/1505.00387.

[15] K. He and J. Sun. *Convolutional Neural Networks at Constrained Time Cost*. URL: https://arxiv.org/abs/1505.00387.

[16] I. J. Goodfellow et. al. "Challenges in Representation Learning: A report on three machine learning contests". In: *Neural Networks* 64 (Apr. 2015). Special Issue on "Deep Learning of Representations, pp. 59–63.

[17] P. Ekman and W. V. Friesen. "Facial Action Coding System: A Technique for the Measurement of Facial Movement". In: *Consulting Psychologists Press, Palo Alto* (1978).

[18] A. Mollahosseini, B. Hasani, and M. H. Mahoor. "AffectNet: A Database for Facial Expression, Valence, and Arousal Computing in the Wild". In: *IEEE Transactions on Affective Computing* (2017).

[19] Datainspektionen. *Dataskyddsförordningen - General Data Protection Regulation (GDPR)*. URL: https://www.datainspektionen.se/Documents/Dataskyddsf\%C3\%B6rordningen\%20-\%20Datainspektionen.pdf.

[20] B. Abidi. *Dataset 02: IRIS Thermal/Visible Face Database*. IEEE OTCBVS WS Series Bench; DOE University Research Program in Robotics under grant DOE-DE-FG02-86NE37968; DOD/TACOM/NAC/ARC Program under grant R01-1344-18; FAA/NSSA grant R01-1344-48/49; Office of Naval Research under grant #N000143010022. URL: http://vcipl-okstate.org/pbvs/bench/index.html.

[21] J. T. Springenberg et al. *Striving for Simplicity: The All Convolutional Net*. URL: https://arxiv.org/abs/1412.6806.

[22] R. Kotikalapudi. *Keras implementation of ResNet*. URL: https://github.com/raghakot/keras-resnet.

[23] F. Yu. *DenseNet-Keras with ImageNet Pretrained Models*. URL: https://github.com/flyyufelix/DenseNet-Keras.

[24] P. Domingos. *A Few Useful Things to Know about Machine Learning*. URL: https://homes.cs.washington.edu/~pedrod/papers/cacm12.pdf.

Master's Theses in Mathematical Sciences 2018:E25
ISSN 1404-6342

LUTFMA-3348-2018

Mathematics
Centre for Mathematical Sciences
Lund University
Box 118, SE-221 00 Lund, Sweden

http://www.maths.lth.se/