Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Measurement and Information Systems

Smile detection from video using deep neural network-based methods

Scientific Students' Association Report

Author:

Mátyás Pólya

Advisor:

Dr. Hullám Gábor
Gábor Révy

2022

Contents

Kivonat

A mosolygás egy olyan arckifejezés, amit a száj két oldalán található izmok befeszítése vált ki. A legkorábbi ismert, mosolyra emlékeztető arckifejezés 30 millió évvel ezelőttre vezethető vissza. Ilyen sok idő alatt az emberi lét szerves része lett ez az arckifejezés. A mosoly lényeges jelzője az érzelmi állapotnak, megelégedettségnek és segít csökkenteni a stresszt. Modern gépi tanuló módszerek segítségével több ilyesfajta kapcsolat feltárása is megoldható lenne. Ehhez az első lépés a mosoly felismerése.

A munkám célja egy mosolydetekciós eljárás megalkotása, neurális hálózatok segítségével. Dolgozatomban megvizsgálok több, eltérő komplexitású módszert. Elsőként egy teljesen összekötött neurális hálózatot, majd több konvolúciós neurális hálózat alapú megoldást tekintek át. Ezek során elemzem a különböző adataugmentációs technikák hatásait, továbbá megvizsgálom, hogy a transzfer learning metodológiáját felhasználva javítható-e az eljárás teljesítménye.

Újdonságértékkel bír, hogy kihasználva az adathalmaz videó voltát, visszacsatolt neurális hálózatok segítségével végzek mosolydetekciót. A szakirodalomban fellelhető algoritmusok jellemzően képek alapján működnek, és nem veszik figyelembe a megelőző képkockák nyújtotta információt.

Az implementált módszerek teljesítményének vizsgálatára a kutatási célra elérhető GENKI-4K és AM-FED+ adathalmazokon végzek összehasonlító elemzést.

Abstract

Smiling is a facial expression that is triggered by the tightening of muscles on both sides of the mouth. The earliest known facial expression resembling a smile dates back to 30 million years ago. Since then it has become an integral part of human expressions. Research has shown that smiling is an essential indicator of emotional state, contentment and it also helps to reduce stress. Modern machine learning methods can be used to explore such relationships. Naturally, the first step is to recognise the smile.

In my research I aim to build a smile detection method using neural networks. In my thesis I investigate several methods with different complexities. First, I consider a fully connected neural network and then several convolutional neural network based solutions. Then I analyze the effects of different data augmentation techniques and investigate whether the performance of learning methods can be improved by using transfer learning methodology.

The novelty of my work is that it exploits the video nature of the dataset to perform detection using recurrent neural networks. Algorithms in the literature typically operate on images and do not take into account the information provided by previous frames.

In order to investigate the performance of the implemented methods, I perform a comparative analysis on the GENKI-4K and AM-FED+ datasets available for research purposes.

Chapter 1

Introduction

Smiling is one of the most common facial expression of a person. It is thought to be evolved from a "fear grin", that was used to signal harmlessness, or submission [22]. Nowadays, it is mainly used to express positive emotions, such as happiness or amusement. It is not a learned facial expression, people smile instinctively, even before being born [13], in the womb.

Many studies have investigated the link between a person's smile, and other traits, arriving at interesting conclusions: researchers found in an experiment [16], that smiling participants had lower heart rates while recovering from a stressful task than those with neutral expressions. This indicates, that smiling can actually reduce stress. In the age of the big data, tools are already commonly available to get the dataset related to the topic of interest, and to create predictive models. However, smile and emotion recognition are still fairly open problems.

In my thesis I aim to build an automated smile detection algorithm. Two types of methods can be considered for such a task: expert system-based, an neural network-based algorithms. The former requires extensive knowledge of the workings of the human face and its muscles, which I do not possess, so my choice fell on the latter. However, the neural network-based methods require an adequately large sample size. The quality of these data can make or break the detection algorithm, so it was essential for me to acquire a solid dataset. Unfortunately, each dataset acquired had its shortfalling, be it repetitiveness, or quantity.

The GENKI-4K [21] and the AM-FED+AM-FED+ [20] datasets were used during the development of the algorithm. The GENKI-4K dataset consists of 4000 images retrieved from the internet. In each image there is a person, who is either smiling, or not. The dataset is labeled accordingly, with binary values. The AM-FED+ dataset consists of 545 videos, or 263 705 frames of people watching advertisements, recorded through their webcams. They are labelled according to the degree to which the participants smile on a scale of 0 to 100. For the autoencoder method, I used the Labeled Faces in the Wild [10] dataset and the Human Faces dataset[1]. These datasets contain 13233 and 7219 face images respectively.

Neural network based methods utilize several different architectures, varying in complexity and generalization ability. In my thesis, I examine a few of them, starting with the simplest one, the fully connected neural network (FNN). Its disadvantage for the desired application is that it does not consider the structure of the input. The input - being an image - bears

[1] `https://www.kaggle.com/datasets/ashwingupta3012/human-faces` (accessed 20.10.2022.)

extra information, such as when moving or rotating it slightly, it still depicts largely the same thing. The convolutional neural networks (CNN) solve this problem, as they utilize the spacial information of the input, using convolutional layers. The inspiration for them comes from the way the eyes work. Through transfer learning methodology, we can reuse models that were trained on millions of pictures, to take advantage of their already learned representations. Using frames from videos as input for a neural network still holds extra information that we can use to further improve the algorithm. The frames depict the same face at different points of time. To utilize this extra information, we can use recurrent neural networks (RNNs), that were developed for inference on sequential data.

An alternative approach for smile recognition involves the usage of the autoencoder architecture, which includes an encoder and a decoder part. Autoencoder is a type of unsupervised learning algorithm, so labels for the data are not needed, only images of faces. When the model has successfully learned the hidden representation of faces, training of smile recognition can begin, using the outputs of the encoder as inputs for the classifier model.

When working with neural networks, the preprocessing of the input data is a crucial step. The location of the face in each picture found in the aforementioned datasets are different from one another, so if they are utilized as inputs for a neural network, then the model would not only need to learn how to recognise whether a person is smiling, but would need to find where that person's face is located. For the latter, there are readily available solutions. Such method is the facial landmark points detector, which allows to easily determine the location of the face, or even the location of the eyes. Using the landmark detector during the preprocessing of the data, our learning algorithm only needs to focus on smile recognition and not on face localization.

After the initial training of the acquired and preprocessed data is done, we can apply data augmentation, which is used when the quantity of the available data is not sufficent. With the help of data augmentation, we can expand the number of available inputs without introducing new data samples, by changing the input images in such ways that do not affect the desired output (e.g. rotation, translation, mirroring).

The thesis is structured as follows. In Chapter 2, I discuss the background of my work. The concepts related to basic neural networks and their variants are described, and the datasets used for the trainings are examined. After that, I review the previous works done in the smile recognition field. In Chapter 3, I go into the details regarding the implemented data preprocessing and neural networks. For each examined model, the effect of its architecture and hyperparameters are investigated. In Chapter 4, I draw conclusions about the examined models, methods and their effects. In Chapter 5, I discuss the improvement possibilities of my work.

Chapter 2

Background

2.1 Neural Networks

Neural networks are a subset of machine learning methods. They were created to imitate the behaviour of the human brain, where millions of neurons are interconnected with one another. In this section a concise overview is presented, introducing basic concepts and components, which is then followed by the description of advanced architectures and related techniques.

2.1.1 Overview

In neural networks, neurons are structured into 3 kinds of layers: the input layer, the output layer, and the hidden layers, of which there can be arbitrary numbers present.

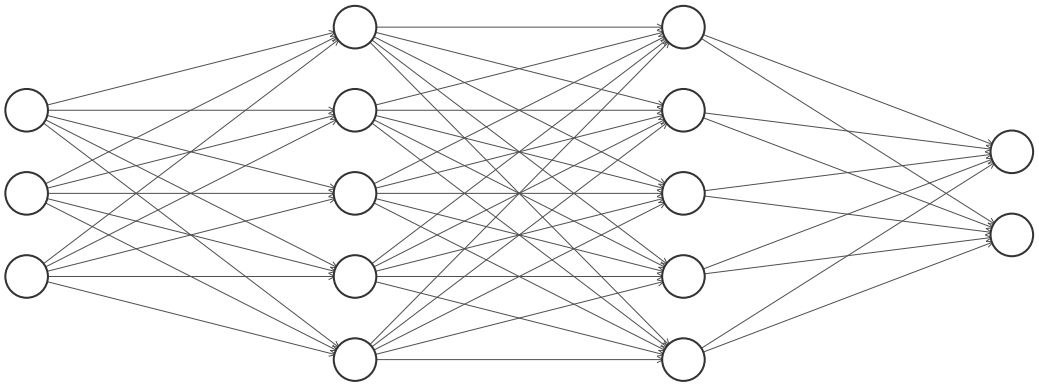
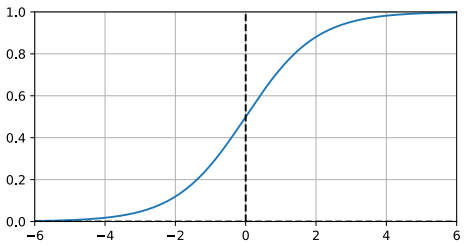


Figure 2.1: A fully connected neural network with 2 hidden layers

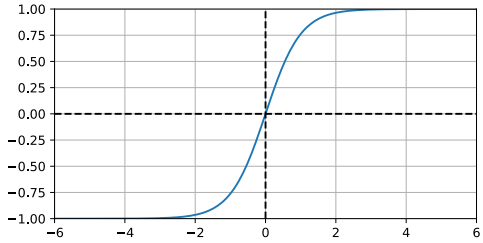
The inputs of neural networks are real ($\mathbb{R}$) numbers. Connections between neurons can be present based on different kind of rules. In a basic feed forward network (Figure 2.1), every neuron in the neighbouring layers are connected to each other in a forward facing way. Each connection applies a multiplication to the signal sent through it. The coefficient of the applied multiplications are called weights. The neuron sums the incoming signal values, and applies an activation function to this sum. Then, it sends this new value to the neurons that it is connected to. To let the network calculate more complex functions, a bias input is introduced into every layer, which takes on a constant 1 value. It enables

the intermediate neurons to apply an offset to their function, independently of their input values.

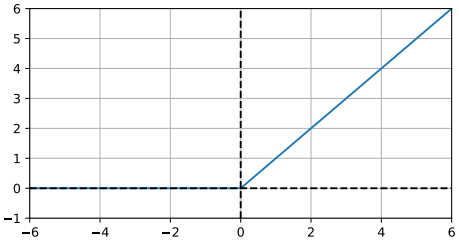
The activation functions introduce non-linearity to the model, which enables it to be an "universal approximator", meaning that it can approximate any arbitrary function.



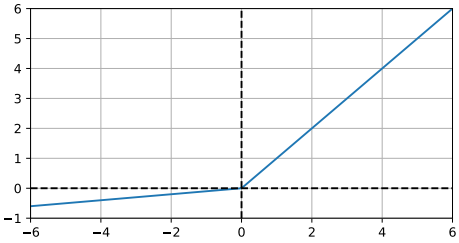
(a) Sigmoid function: $\frac{1}{1+e^{-x}}$



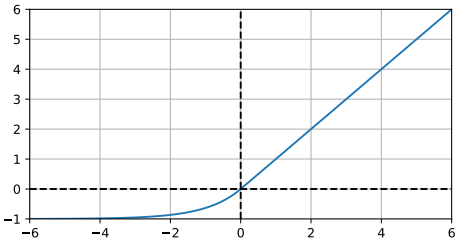
(b) Tanh function: $\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$



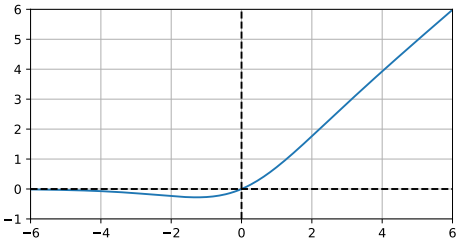
(c) ReLU function: $\max(0, x)$



(d) Leaky ReLU function: $\max(0.1 \cdot x, x)$



(e) ELU function: $\begin{cases} x & \text{if } x \geq 0, \\ \alpha(e^{x}-1) & \text{otherwise} \end{cases}$



(f) Swish function: $\frac{x}{1+e^{-x}}$

Figure 2.2: Activation functions used in practice

The notation for neural network components is the following:

- $W^{(l)}_{(n,m)}$ is the weight of the connection between the n-th neuron of the l-th layer and the m-th neuron of the $(l-1)$-th layer
- $W^{(l)}_{(n,0)}$ is the weight of the bias in the n-th neuron of the l-th layer
- $y^{(l)}_{(n)}$ is the output of the n-th neuron of the l-th layer
- $g(\cdot)$ is the activation function of a neuron

To make the math easier, the bias inputs are considered in each layer as the 0th neuron

The following equation holds for every neuron that has other neurons connected to it as inputs:

$$y^{(l)}_{(n)} = g\left(\sum_{i}\left(W^{(l)}_{(n,i)} \cdot y^{(l-1)}_{(i)}\right)\right) \tag{2.1}$$

When assessing the performance of a neural network, we would like to quantify how well the model performs. For this, we use a loss function, which tells us how close are the predictions ($\hat{y}_n$) to the actual known values (y_n) :

- Cross Entropy Loss $= \frac{-1}{N} \sum_{n=1}^{N} [y_n \log \hat{y}_n + (1 - y_n) \log(1 - \hat{y}_n)]$ is used for classification problems, such as smile recognition
- Mean Squared Error $= \frac{1}{N} \sum_{n=1}^{N} (y_n - \hat{y}_n)^2$ is used for regression problems, such as image reconstruction through autoencoders

For the training of neural networks, the available dataset is typically divided into 3 disjoint sets: training, validation and test set. The network is only trained on the training dataset, and evaluated on the validation, and test sets. This helps preventing overfitting to the training data, as the training is ended when the model begins the underperform on the datapoints not present during learning.

Neural networks learn with the help of the backpropagation algorithm. The weights of the model are updated based on the value of the derivative of the loss function, with regard to the value of the weight $\left(\frac{dL}{dW}\right)$.

2.1.1.1 Example XOR implementation

The problem with the predecessors of neural networks, i.e. perceptrons, was that they could not realize even simple functions, such as the XOR function. However, the application of the backpropagation method, the utilization of appropriate activation functions, and creating more complex networks by stacking several layers enabled the learning of arbitrarily complex functions.

x	y	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

Table 2.1: XOR operation on 2 variable

For example, neural network found in Figure 2.3 satisfies the given conditions, using ReLU (Figure 2.2c) as activation function. Figure 2.4 shows the output value of the aforementioned neural network, as a function of (x, y).

2.1.2 Fully connected neural networks

In a fully connected neural network, each neuron of a layer is connected to every neuron in the previous layer (except the neurons of the input layer). With this kind of architecture, they are "structure agnostic", there are no assumptions made about the input's property. This kind of behaviour makes them widely applicable, but they tend to underperform in case of sequential or other highly structured data, compared to other architectures that utilize the information acquired from the input's structure. Figure 2.1 depicts a fully connected neural network.

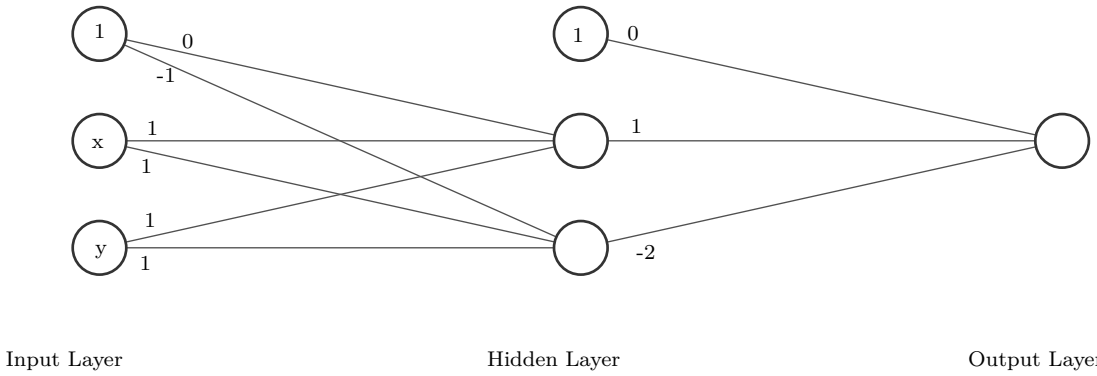


Figure 2.3: A fully connected neural network calculating the XOR function

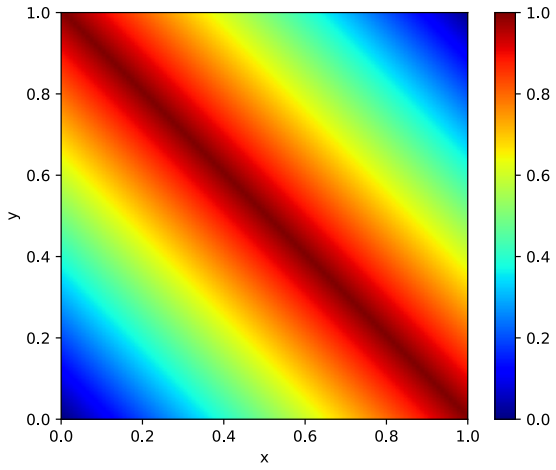


Figure 2.4: The plot of the network found in Figure 2.3

2.1.3 Convolutional neural networks

The idea for convolutional neural networks comes from the way the eye works. When researchers were examining how cats' eyes worked [23], they found that similar shapes caused the activation of same regions of the cat's brain (for an illustration of the experimental setup see Figure 2.5). After further investigation, they concluded that these regions act as a kind of feature extractor, and only their outputs get passed deeper into the brain.

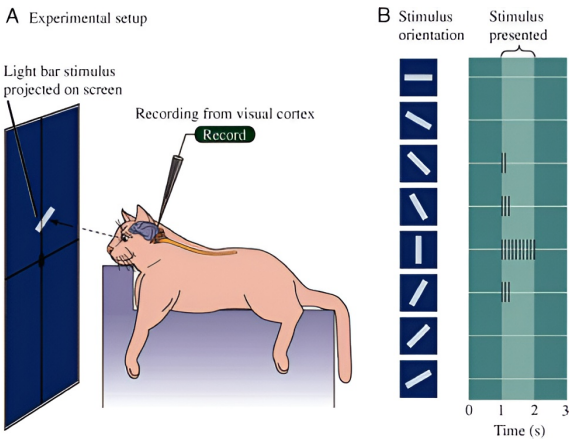


Figure 2.5: The setup of the experiment, in which the brain activations of cats induced by different shapes were investigated [23]

To simulate this kind of feature extraction, the convolutional layers were created. These layers use kernels, which are applied to the image to perform the feature extraction. The output of these operations produce a tensor (multidimensional matrix), which can also fed into another convolutional layer. Depending on the application, several consecutive convolutional layers can be built into a network.

2.1.3.1 Convolutional operation

The result of the convolution operation is calculated as follows:

$$g[m,n] = (f * h)[m,n] = \sum_{i=0}^{k-1} \sum_{j=0}^{l-1} f[m+i, n+j] \cdot h[i,j] \tag{2.2}$$

where

- h is a $k \times l$ convolutional kernel
- f is an $o \times p$ input matrix ($o \geq k$ and $p \geq l$)
- g is the $(o-k+1) \times (p-l+1)$ convolved matrix

Convolving a 2×2 kernel on a 3×3 matrix would look like as follows:

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix} * \begin{bmatrix} b_{00} & b_{01} \\ b_{10} & b_{11} \end{bmatrix} =$$

$$\begin{bmatrix} a_{00} \cdot b_{00} + a_{01} \cdot b_{01} + a_{10} \cdot b_{10} + a_{11} \cdot b_{11} & a_{01} \cdot b_{00} + a_{02} \cdot b_{01} + a_{11} \cdot b_{10} + a_{12} \cdot b_{11} \\ a_{10} \cdot b_{00} + a_{11} \cdot b_{01} + a_{20} \cdot b_{10} + a_{21} \cdot b_{11} & a_{11} \cdot b_{00} + a_{12} \cdot b_{01} + a_{21} \cdot b_{10} + a_{22} \cdot b_{11} \end{bmatrix}$$

The function can be extended to work on multi-dimensional matrices (tensors), as is the case with RGB images having three color channels.

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 0 & 1 & 2 \\ 3 & 4 & 5 & 6 \end{bmatrix} \Rightarrow \begin{bmatrix} 6 & 8 \\ 9 & 6 \end{bmatrix}$$

Figure 2.6: 2×2 max pooling on a 4×4 matrix

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 0 & 1 & 2 \\ 3 & 4 & 5 & 6 \end{bmatrix} \Rightarrow \begin{bmatrix} 3.5 & 5.5 \\ 4 & 3.5 \end{bmatrix}$$

Figure 2.7: 2×2 average pooling on a 4×4 matrix

2.1.3.2 Pooling layers

Using pooling layers, we reduce the dimensionality of the features, while still retaining most of the important information found in them. Further operations are performed on these reduced features, making the model less dependent on the location of the original feature.

- Max pooling layer selects the maximum element found amongst the examined values (see Figure 2.6).
- Average pooling calculates the average of the examined values (see Figure 2.7).

2.1.3.3 Network architecture

A typical convolutional neural network (CNN) consists of multiple convolutional layers, each one followed by a pooling layer, then finally, several fully connected (dense) layers (see Figure 2.8).

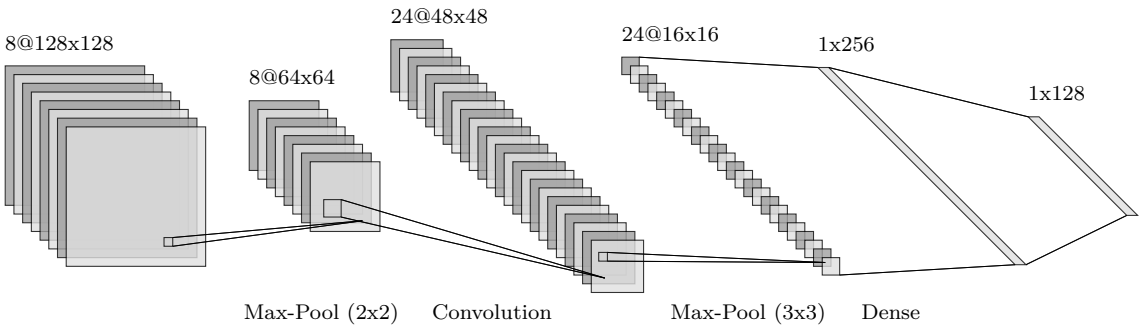


Figure 2.8: A convolutional neural network

CNNs are mainly used for applications, where the input contains structural information, like in the case of images, where neighbouring pixels can be highly correlated with each other (e.g. they belong to the same object).

2.1.4 Transfer Learning

There are several models with millions of parameters trained on millions of data points made available to use. These models can be used out-of-the-box for the specific application that they were created for. In the case of image recognition, a model is typically constructed to recognize the objects found in their training data, and inside the model, important image features are recognized. With these models, predictions can only be made on objects that they were trained on, but with the help of transfer learning, additional classes can be "inserted" into the prediction space.

In order to achieve this, we take the original model, and "freeze" its weights: this means that they are not updated during training. Then, we remove the original output layer, and add hidden layers after the last remaining one, as well as a new output layer, tailored to our desired application. With this kind of setup, we take advantage of the learned inner representation of the original model, while not having to sacrifice valuable computational resources to train it. After the initial training with the original layers frozen, fine-tuning can be applied, meaning that the model is trained again, this time with its layers unfrozen. Fine-tuning can greatly increase the performance of the model.

Transfer learning is ideal, when working with small sample size.

Applications include:

- Natural Language Processing
- Computer vision
- Speech/Audio recognition
- Computer games (e.g. AlphaGo)

2.1.5 Recurrent neural networks

A recurrent neural network (RNN) has connections between neurons, that create a cycle, allowing the output of a neuron to affect its subsequent inputs (see Figure 2.9). This makes it ideal for working with sequential data, as the inner features of RNNs can store features calculated from preceding datapoints. RNNs are also useful when working with variable-length data, contrary to feed forward neural networks, which need fixed-length data.

Training a basic recurrent neural network involves a concept called "unrolling", meaning that from each neuron, time indexed copies are created, and backward connections are replaced with forward connections. This causes the worsening of the exploding/vanishing gradient problem. It also fails to remember information for a longer period of time. To combat these issues, cell-based RNNs are used in practice, such as LSTM and GRU.

2.1.5.1 Long short-term memory (LSTM)

LSTMs introduced gates, which allow it to remember information for arbitrary time intervals. They also solve the vanishing gradient problem.

The LSTM consists of 3 gates:

- Input gate: determines what information from the cell states that were given as input should be kept.

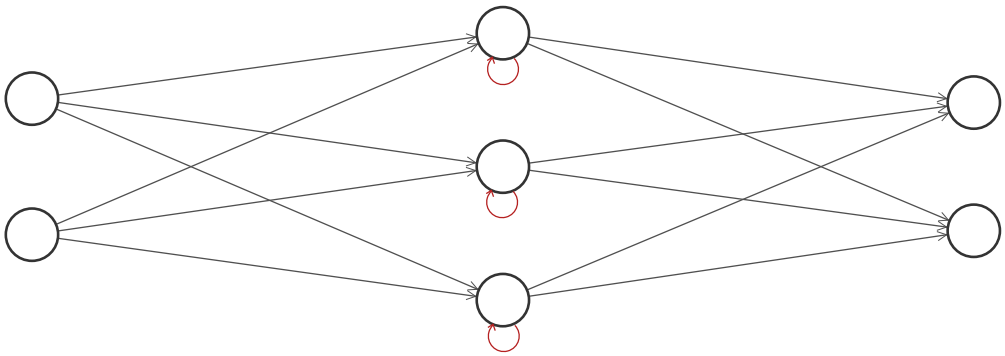


Figure 2.9: A simple recurrent neural network with recurrent nodes in the hidden layer.

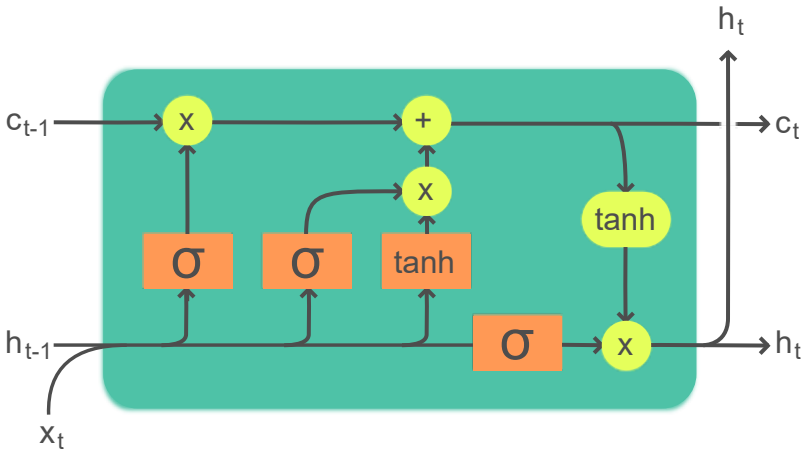


Figure 2.10: The architecture of an LSTM cell, source: Wikimedia Commons [5]

- Forget gate: determines what information from the cell state should be forgotten.
- Output gate: determines what information from the hidden state should be exposed to the next layers.

Figure 2.10 shows how the hidden state (h_t) and the cell state (c_t) are calculated from the inputs.

2.1.5.2 Gated Recurrent Unit (GRU)

GRU is very similar to LSTM, but in contrast with them, they lack an output gate, thus they have fewer parameters. GRU's performance on certain tasks was found to be similar to that of LSTM. GRUs have been shown [3, 26] to perform better on certain smaller and less frequent datasets.

The GRU consists of 2 gates:

- Reset gate: determines the information to be be passed from the current cell state to the next time step.

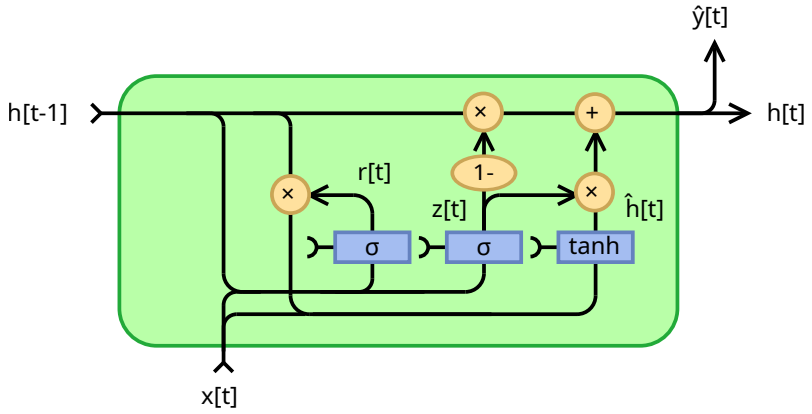


Figure 2.11: The architecture of a GRU cell, source: Wikimedia Commons [4]

- Update gate: determines what information from the previous time step cell state should be used in the current time step.

Figure 2.11 shows how the cell state (h_t) is calculated from the inputs.

With the spread of cell based RNNs, new applications arose, and the performance of existing applications improved. Such applications include:

- Time series prediction
- Machine translation
- Speech recognition
- Generating image descriptions
- Text summarization
- Music composition

2.1.6 Autoencoders

Autoencoders are a type of neural networks, that aim to learn the hidden encoding of unlabeled data. Introducing information bottleneck into the model forces it to compress the data into a lower dimensional feature space. The fitness of encodings are calculated on how accurately the data can be reconstructed based on them.

Autoencoders consist of 3 parts:

- Encoder: transforms the input data into an encoded representation that is typically several orders of magnitude smaller.
- Bottleneck layer: contains the compressed feature representation.
- Decoder: reconstructs the data based on the compressed features.

Figure 2.12 depicts an autoencoder neural network, which encodes a 10 dimensional feature space into a 3 dimensional one.

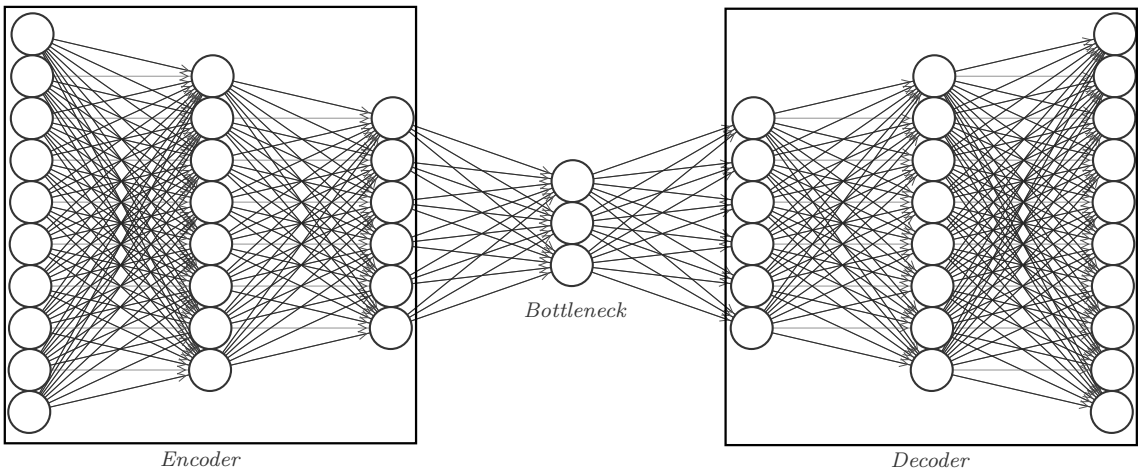


Figure 2.12: An autoencoder neural network

2.1.7 Dropout

Dropout is a regularization technique used to reduce overfitting to the training dataset. Outputs of neurons are randomly zeroed out, making the model less dependant on individual neurons. It approximates simultaneously training a large number of models with different architectures.

2.1.8 Data augmentation

Data augmentation is a technique of expanding the quantity of datapoints. It involves modifying the existing input data in such a way, that the labels belonging to them remain the same, thus creating more datapoints, without the need for extra labeling work. For images, such modifications include: flipping, translating, noise adding, scaling, cropping, bluring or contrast changing.

2.2 Facial landmark points detectors

Facial landmark point detection algorithms identify predefined points on the face. They are usually created using deep learning methods, where the dataset consists of face images and the location of the points to be identified.

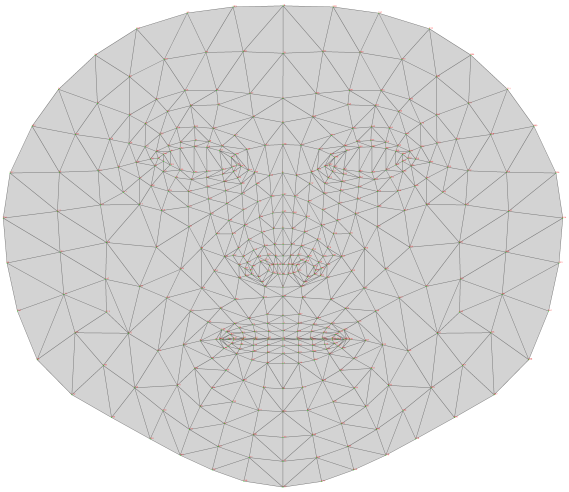


Figure 2.13: The face mesh, that is fitted to the face by a neural network [12] in the MediaPipe [17] library. source: MediaPipe

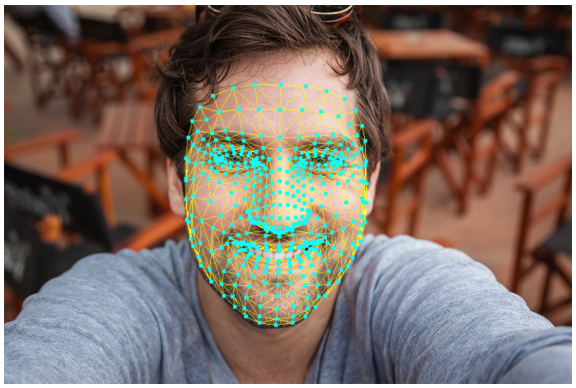


Figure 2.14: The face mesh, that is fitted to the face by a neural network [12] in the MediaPipe [17] library. source: MediaPipe

Figure 2.14 shows the landmark points of Figure 2.13 detected on a face.

2.3 Datasets

To recognize smile using machine learning techniques, adequately labeled datasets are needed. The diversity and quantity of the provided data is positively correlated with how well the model can perform.

2.3.1 GENKI-4K

The GENKI-4K [21] dataset consists of – as its name implies – 4000 images, collected from the internet. The images were taken in a natural environment instead of a controlled one, making them ideal for the smile activated camera shutter mechanism, for which it was collected.

Figure 2.15: Sample of the images found in the GENKI-4K dataset (top 2 rows: smiling, bottom 2 rows: not smiling)

The dataset consists of 2162 smiling and 1838 non-smiling pictures. The labels include whether the subject is smiling, and the subject's head orientation expressed in Euler angles (roll/pitch/yaw).

As it can be seen in Figure 2.15, the images are of different quality, making them ideal for generalization.

2.3.2 AM-FED+

The AM-FED+ [20] dataset consists of 1044 webcam videos recorded in real world conditions. The subjects were tasked to watch video advertisement, while their reactions were being filmed.

Figure 2.16: Sample of the frames found in the AM-FED+ dataset (top 2 rows: smiling, bottom 2 rows: not smiling)

545 videos, or 263705 frames are labeled as follows:

- 10 symmetrical and 4 asymmetric (unilateral) action units (AUs) from the Facial Action Coding System [6] (FACS). Action units are the smallest unit of movements produced by a muscle or muscle group on the face.
- 2 head movements: whether the head is tilted forwards or backwards
- the extent of smile (on a scale of 0-100)
- expressiveness: the presence of a non-neutral facial expression
- gender
- facial landmark points
- familiarity with, liking of and desire to watch again for the stimuli videos

2.3.3 Dataset for autoencoders

For the training of autoencoders, labeled data is not required. Datasets containing images of faces were downloaded from the kaggle.com website.

2.3.3.1 Labeled Faces in the Wild (LFW) dataset

The Labeled Faces in the Wild [10] dataset consists of 13233 images. The "Labeled" word in the name refers to the fact that the name of the famous person belonging to the image is given. Figure 2.17a shows a small sample of the dataset.

2.3.3.2 Human Faces dataset

The Human Faces dataset[1] consists of 7219 images, with diverse qualities. Figure 2.17b shows a small sample of the images found in the dataset.

(a) Sample of images found in the LFW dataset

(b) Sample of images found in the Human Faces dataset

Figure 2.17: Datasets used for training the autoencoder

[1] https://www.kaggle.com/datasets/ashwingupta3012/human-faces (accessed 20.10.2022.)

2.4 Related works

There are several papers discussing smile, and other expression detection algorithms, ranging from simple to complex ones. In this section, related works are discussed.

2.4.1 Smile and expression detection

Glauner [8] detected smiles with the help of deep neural networks. They tested their algorithm on the Denver Intensity of Spontaneous Facial Action (DISFA) database [18], achieving a 99.45% accuracy. They compared the performance of models receiving images of a person's whole face, with those that obtained images of only the area around the person's mouth, arriving at the conclusion that smile detection performs better when the whole face is shown to the models as inputs.

Whitehill et al. [25] presented the GENKI [24] dataset (a superset of GENKI-4K [21]). They created a standard linear regression model that achieved a 97% accuracy based on 8×8 pixel downsampled images retrieved from the DFAT [11] dataset, illustrating the underlying problem with it. With the help of Gabor Energy Filters, Box Filters, Edge Orientation Histograms and Local Binary Patterns, they created a smile detection algorithm, using GentleBoost [7] and SVMs. They also discussed the performance improving effect of fixing the eyes to constant locations.

Bianco et al. [1] created a smile detection convolutional neural network trained on the GENKI-4K [21] dataset, achieving an accuracy of 93.77%. They discussed the usage of facial landmark points for the input image generation. Image augmentation and distortion effects were examined throughout the work. The obtained model is compared to other smile recognition solutions.

McDuff et al. [19, 20] presented the AM-FED and AM-FED+ datasets. The data obtaining and labeling steps were thoroughly documented. They examined and compared the qualities of popular datasets used for emotion detection and concluded that posed datasets have disadvantages when compared to fully spontaneous datasets. A baseline solution was given for the recognition of facial expressions.

Chapter 3

Implemented methods

The methods presented in this thesis were implemented using the Python programming language. The implementations for the neural networks were made using the Tensorflow/Keras library. The models were mainly optimized for the GENKI-4K dataset, as the AM-FED+ dataset has too much data samples, thus making the training slower (by a factor of 50).

3.1 Feature processing

In machine learning, the quality of the training data has great impact on the results, therefore it is important to preprocess the datasets before feeding them into the model.

3.1.1 Input generation

As seen in Figure 2.15 and Figure 2.16 the location of the faces found in the data samples varies from image to image. This is unfortunate, as this introduces extra features that the model has to learn. To work around this, a face or landmark detector can be used to locate the faces in the images, and then translate and clip them in such a way that their position is well-defined. In my implementation, I fixed the position of the eyes to a constant location.

Figure 3.1 depicts the steps involved in generating the input images. In the second row of the figure, the detected landmark points are projected onto the images. The centers of the eyes (colored green) are not part of detected points, but are calculated from the points belonging to the perimeters of the eyes (colored blue). In the 3rd column, a differently shaped "mask" can be seen. As a fail-safe mechanism, two landmark detector was used (Mediapipe [17] and dlib [15, 14]), for handling situations, when one of them fails to detect the landmark points.

3.1.2 Augmentation

Throughout the thesis, data augmentations were implemented using preprocessing layers. Preprocessing layers can be inserted into a neural network the same way as non-preprocessing layers (e.g. dense layers, convolutional layers), but contrary to them, their behaviour are not learnt during the training period, but given at construction time. Such layers include the random translation layer, which takes 2 parameters, denoting the extent

Figure 3.1: Examples of input image generation using facial landmark points detector

of the maximum horizontal and vertical translation, which are random generated from a uniform probability function. The random contrast layer adjusts the contrast of the image, based on a randomly generated factor, and the random flip layer flips the image horizontally 50% of the time. It is important to mention, that new random values are continuously generated throughout the training period, not just once at the start of it. During inference time, the preprocessing layers are disabled, they don't make modifications on the received images.

3.2 Fully connected neural networks

As discussed previously, fully connected neural networks have the simplest architecture. The input is a preprocessed image, whose values are flattened, transforming it from a tensor (image height, image width, 3) into a (image height×image width×3) vector, which can be used as input for the dense layers. In Figure 3.2 a sample implementation can be seen.

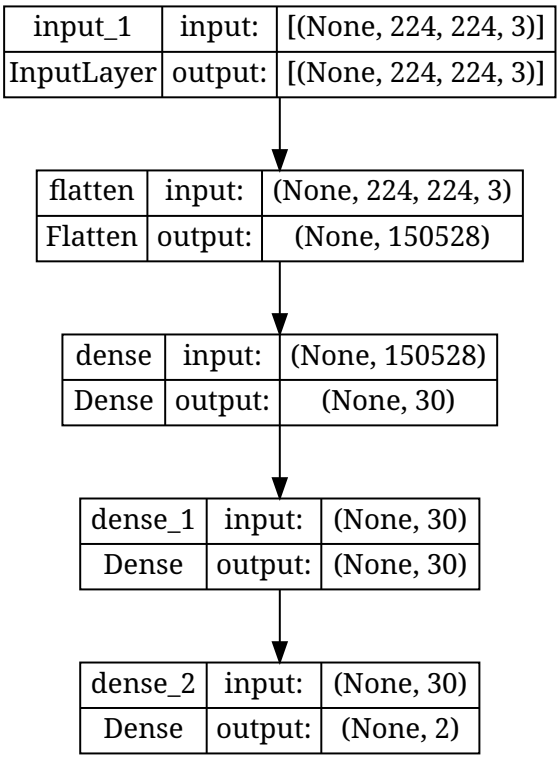


Figure 3.2: Implemented fully connected neural network, with image height = 224; image width = 224; 2 hidden layers, each with 30 neurons

3.2.1 Architecture

First, 30 different models were trained on the GENKI-4K dataset, having different number of layers [1,2,3,4,5,6], and neurons [10,20,30,40,50]. Each model was trained 10 times, to account for statistical irregularities.

We can see in Figure 3.4a, that the models having 1 hidden layer, or 10 neurons, underperformed the other models. The models having more than 10 neurons, or more than 1 layers had almost identical validation accuracies (~ 0.9). Examining Figure 3.3, we can see that every model could reach that kind of accuracy, but for simpler models, the value highly differed between each run. The reason for this phenomenon lies in the weight initialization process: if the initial values of the weights were wrongly set, the model could not learn. For further examination, models having 2 or 3 layers, with 20 or 30 neurons were used.

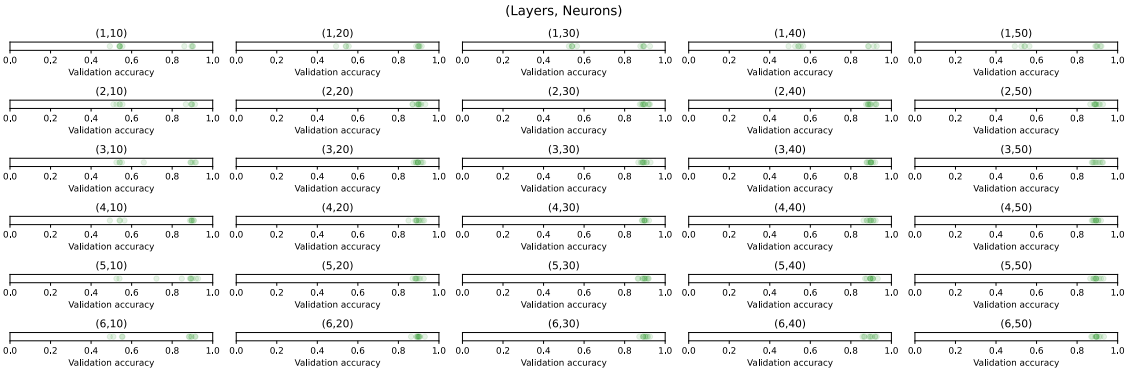
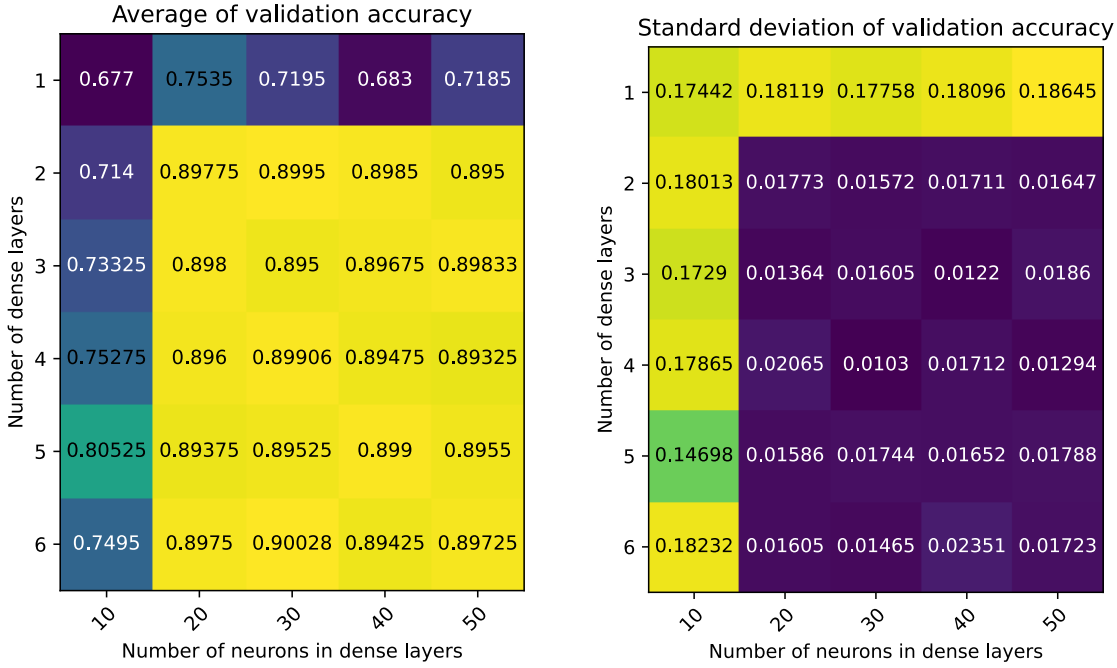


Figure 3.3: Validation accuracies of the examined model



(a) Average of validation accuracies

(b) Standard deviation of validation accuracies

Figure 3.4: Statistics based on the performance measures of models having different number of layers and neurons (N=10)

3.2.2 Dropouts, activation functions

Next, the effect of dropout layers [not present, 0.2] and different activation functions [leaky ReLU, ReLU, tanh, sigmoid] were tested.

The sigmoid and tanh functions underperformed the other functions. The dropout layers made the models perform more consistently, as the standard deviations decreased while using them. The leaky ReLU function had better results than the ReLU function. For further analysis, models with leaky ReLU activation functions and 0.2 dropouts are considered.

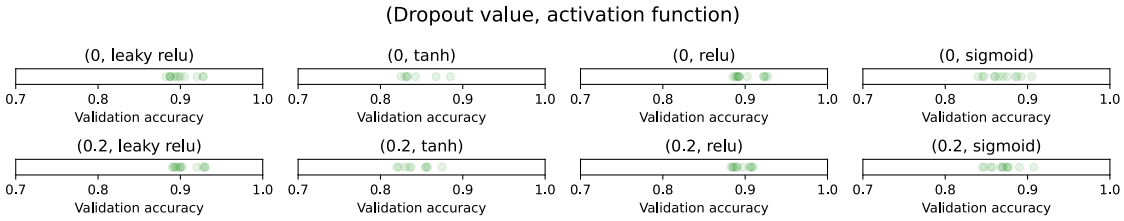
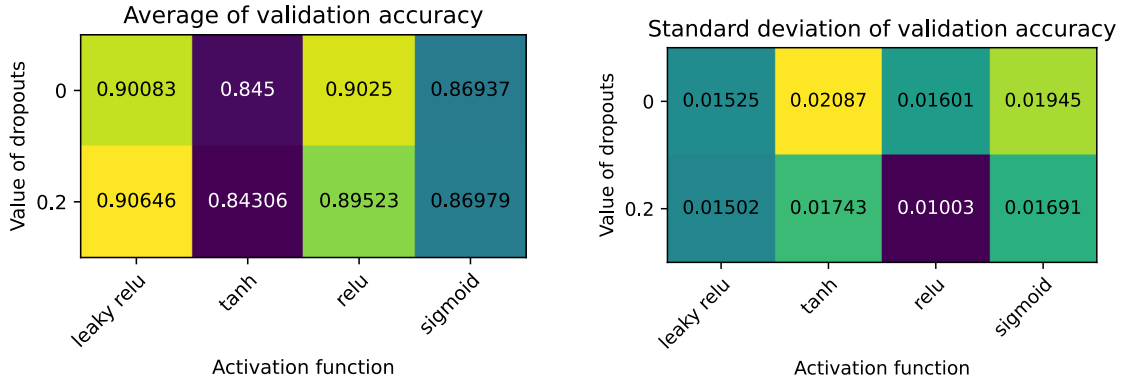


Figure 3.5: Validation accuracies of models with different activation functions and dropout values.



(a) Average of validation accuracies.

(b) Standard deviation of validation accuracies

Figure 3.6: Statistics based on the performance measures of models with different activation functions and dropout values (N=3).

3.2.3 Image flipping, contrast

Next, the effects of augmentation are examined. For the implementation, preprocessing layers are used, meaning that the augmentation is done inside the model. GPU resources are used instead of CPU, making it faster. It is also not needed to temporary store the augmented dataset in the memory. Figure 3.7 and Figure 3.8 shows the results of the trained models.

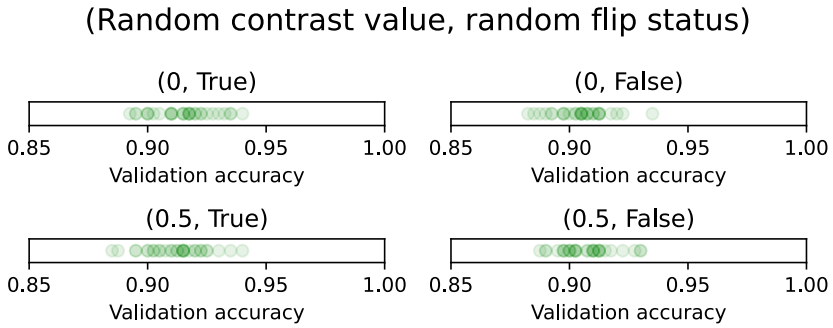
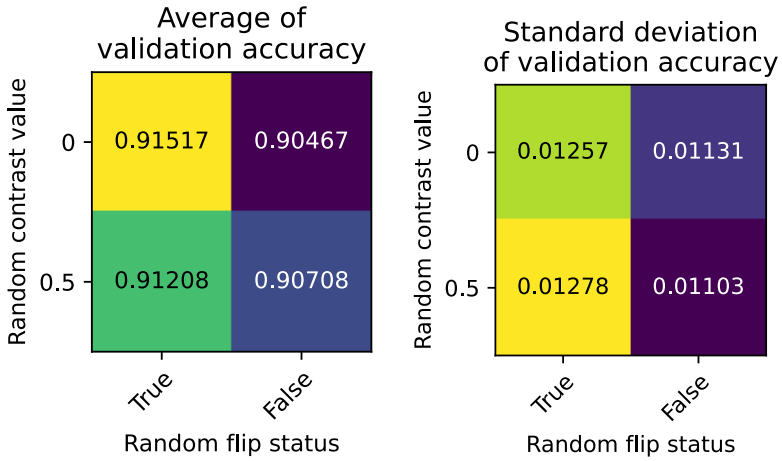


Figure 3.7: Validation accuracies of models with flipping and contrast modifying preprocessing layers

In Figure 3.8a it can be clearly seen, that the flipping layer improves the models. The same thing cannot be said about the contrast layer. Further examined models implement the flipping layer, but not the contrast layer.



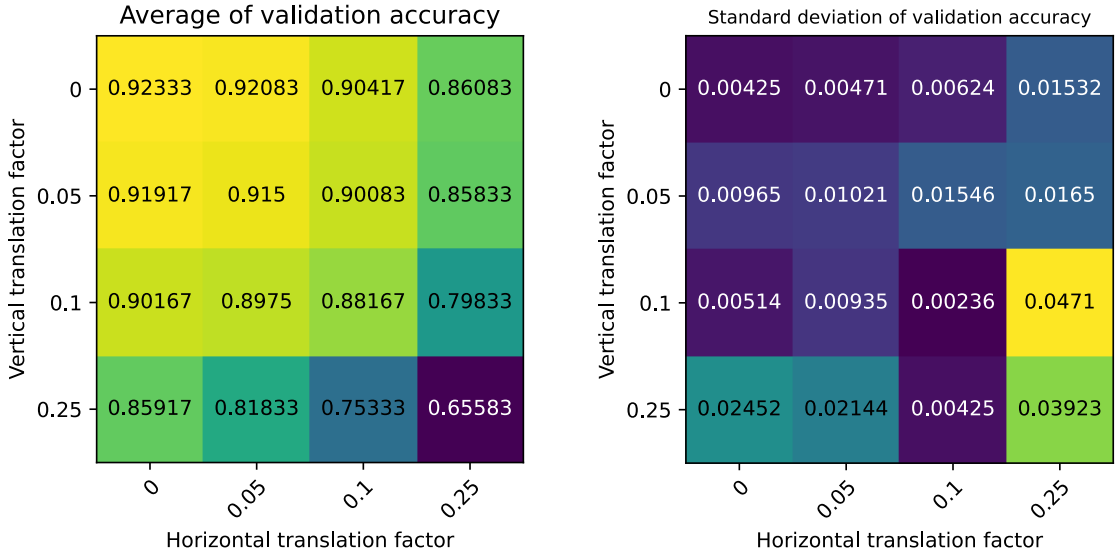
(a) Average of validation accuracies

(b) Standard deviation of validation accuracies

Figure 3.8: Statistics based on the performance measures of models, with and without flip and contrast preprocessing layers (N=30)

3.2.4 Translation

Another examined image augmentation operation is the translation. The images are translated randomly by an uniform probability bounded by the given parameters. Different values for the vertical and horizontal parameters were used.



(a) Average of validation accuracies

(b) Standard deviation of validation accuracies

Figure 3.9: Statistics of performance measures of models with different translation layers (N=3)

Figure 3.9a shows that the models performed worse the bigger the translations were. This is the opposite of what we would expect from an augmentation, but it can be explained by the fact that this fully connected neural network model was first optimized for non-

augmented learning. For it to perform better on translated images, the architectures would have to be examined again, while trained on translated datapoints.

3.2.5 AM-FED+

The obtained models with the best results are trained on the AM-FED+ dataset.

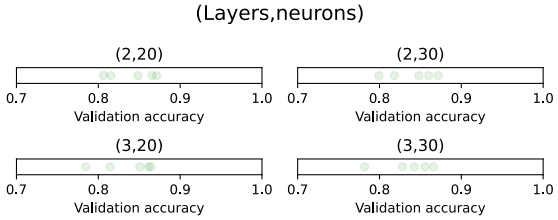
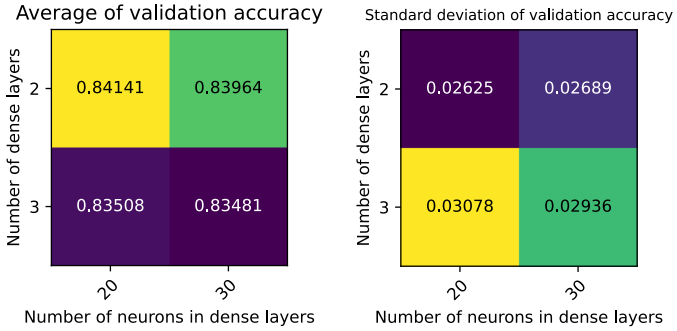


Figure 3.10: Validation accuracies of models trained on the AM-FED+ dataset



(a) Average of validation accuracies

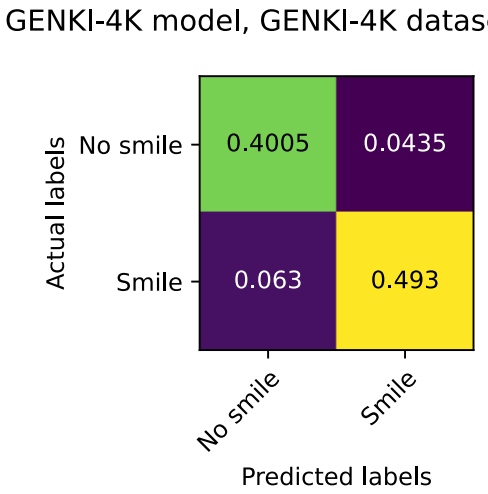
(b) Standard deviation of validation accuracies

Figure 3.11: Statistics of models with different number of layers and neurons, trained on the AM-FED+ dataset (N=5)

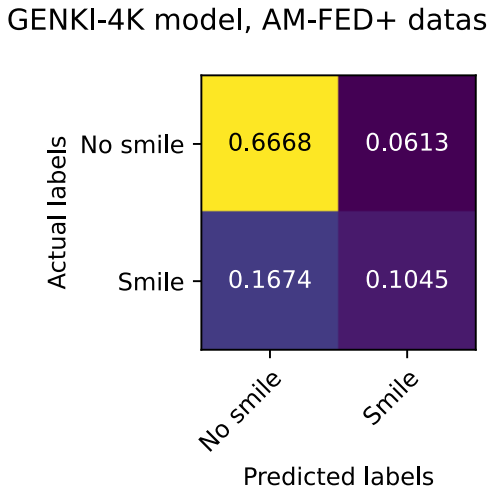
The best performing model had 2 hidden layers, with 20 neurons each.

3.2.6 Model evaulation

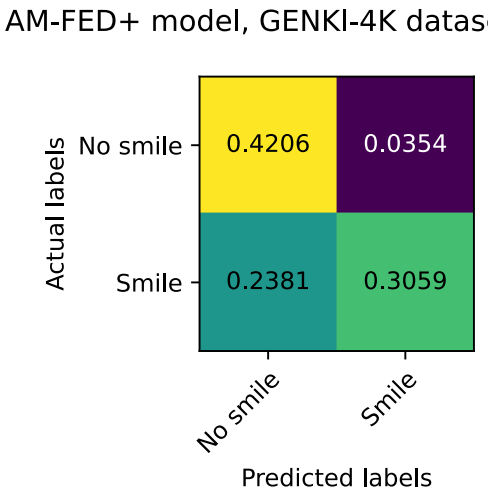
Figure 3.12 shows the accuracies achieved by different models tested and trained on different datasets. In Figure 3.12a and Figure 3.12d the models were evaluated on the test partition of their associated datasets. The models in Figure 3.12b and Figure 3.12c were evaluated on the dataset not trained on. On the AM-FED+ dataset, the GENKI-4K model (~ 0.77) almost performed as well as the AM-FED+ model (~ 0.79). This does not hold for the opposite case, as the AM-FED+ model performed significantly worse (~ 0.72), than the GENKI-4K model (~ 0.89) on the GENKI-4K dataset. This can be explained by the diverse quality of the images found in the GENKI-4K dataset.



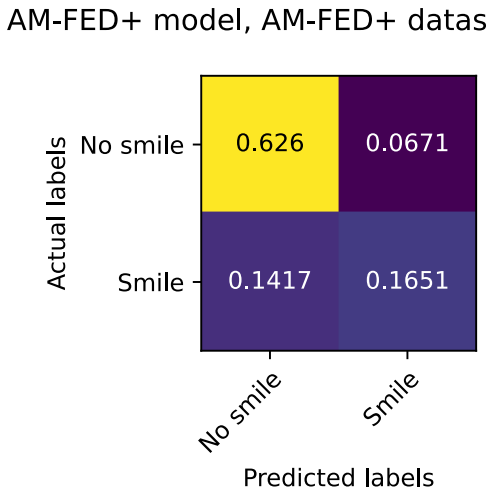
(a) Test accuracy of GENKI-4K model



(b) GENKI-4K model accuracy on AM-FED+ dataset



(c) AM-FED+ model accuracy on GENKI-4K dataset



(d) Test accuracy of AM-FED+ model

Figure 3.12: Confusion matrices obtained on data not seen by the fully connected model

3.3 Convolutional neural networks

Convolutional neural networks utilize convolutional and pooling layers. They are constructed as follows: the input layer receives the preprocessed image, with its shape information still intact (image height, image width, 3), then several convolution and pooling layers are implemented. The last pooling layer gets flattened, then fed into a dense layer, which is followed by more dense layers. The output layer has 2 neurons, as we need to predict 2 classes. As with the fully connected neural network, the models are optimized on the GENKI-4K dataset. Figure 3.13 shows a sample CNN implementation.

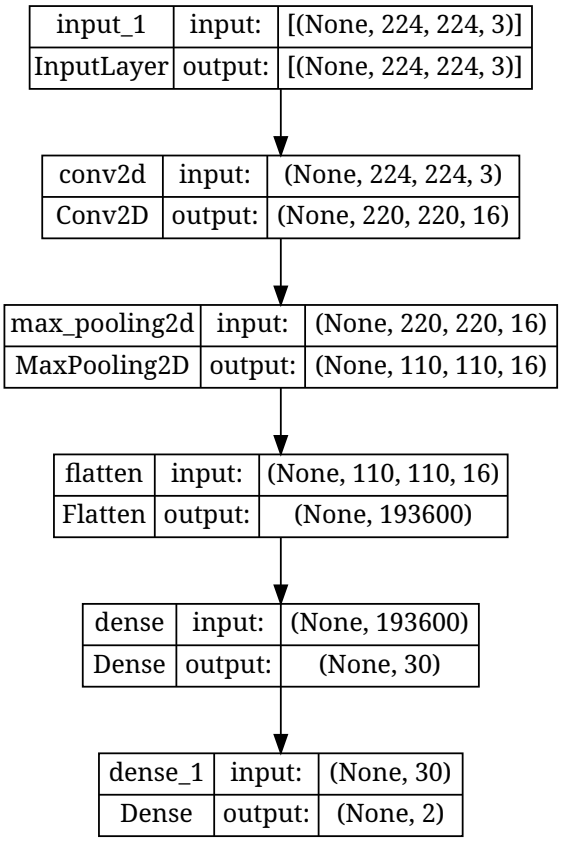
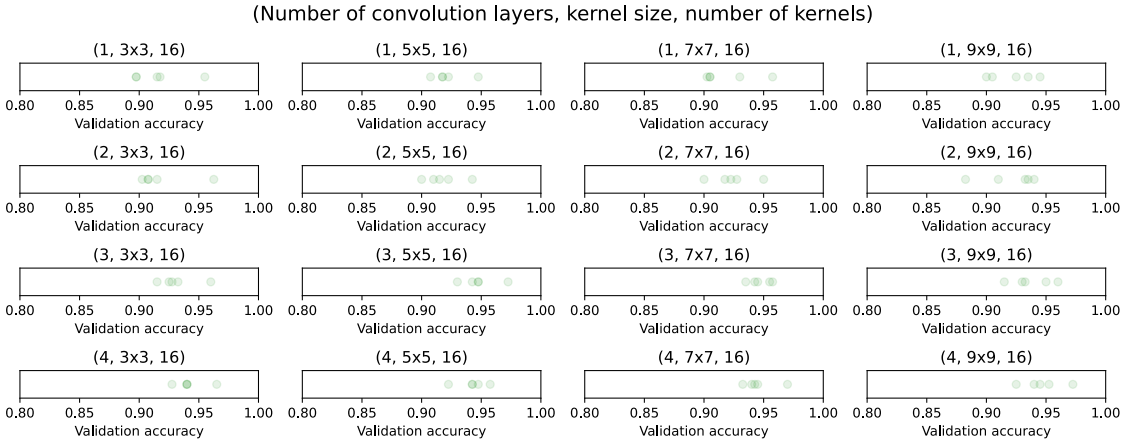


Figure 3.13: A CNN, with 1 convolutional and max pooling layer, and 1 dense hidden layer

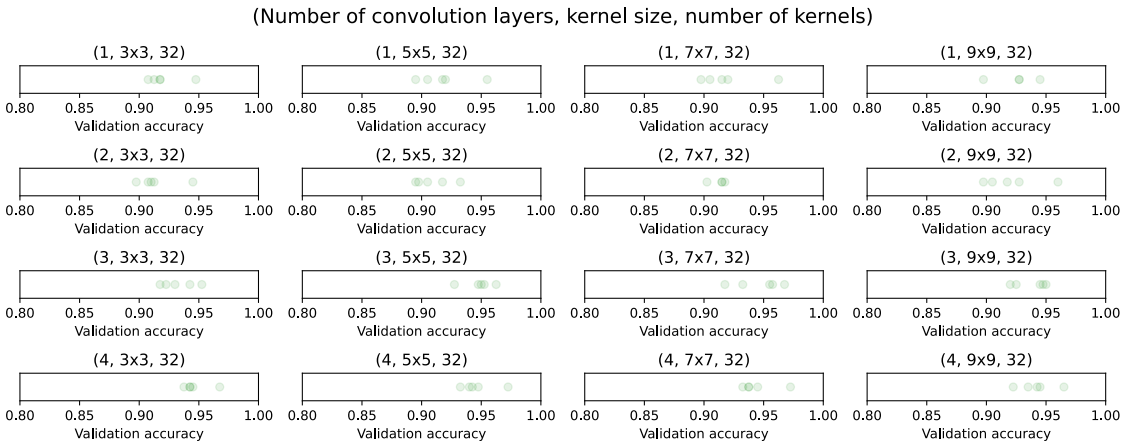
3.3.1 Architecture

First, the performance of different architectures is evaluated. Models with different number of convolution layers [1,2,3,4], differently sized kernels [3×3, 5×5, 7×7, 9×9], and different number of kernels [16,32] are examined. Figure 3.14 shows the validation accuracies achieved by the inspected models.

Figure 3.15 depicts the statistics calculated from the validation accuracies of the examined models. Models with 2 or less convolutional layers underperformed the models with 3 or more convolutional layers. In both case, models with 3 convolutional layers and 3×3 or 9×9 kernel sizes achieved less than 0.94 average validation accuracy, while all the other models having 3 or more convolution layers achived above 0.94 results. The best performing model had 3 convolutional layers, and 5×5 sized kernell , and achieved 0.948 accuracy. There was no considerable difference between the models with different number of kernels. For further analysis the model with 3 convolutional layers, 5×5 kernel size, and 32 kernels was choosen.

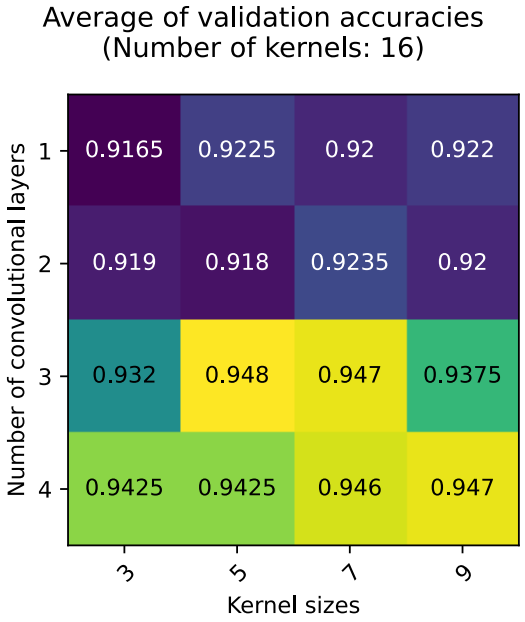


(a) Validation accuracies of the models with 16 number of kernels in each convolutional layer

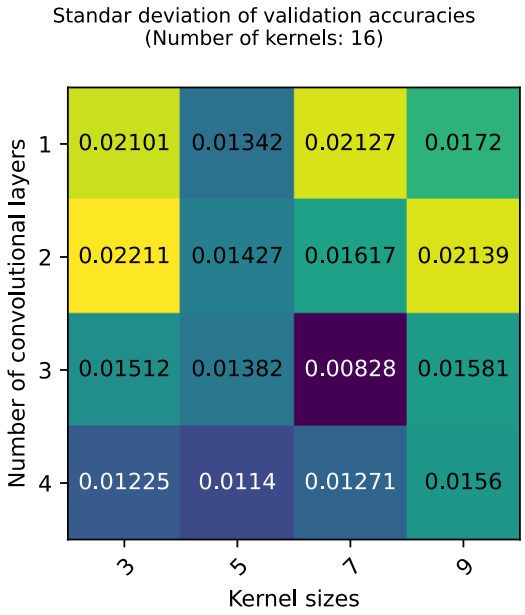


(b) Validation accuracies of the models with 32 number of kernels in each convolutional layer

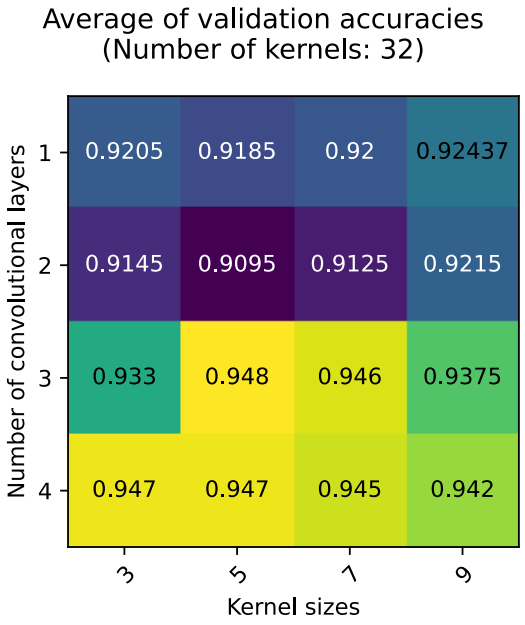
Figure 3.14: Validation accuracies of the models with different kernels and convolutional layers



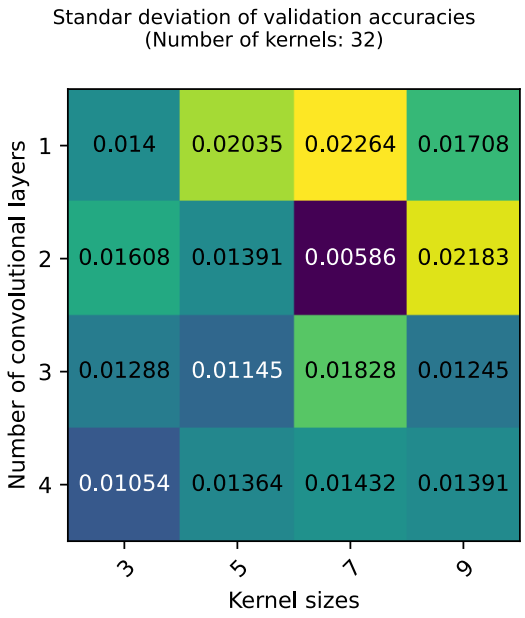
(a) Average of validation accuracies of models with 16 kernels



(b) Standard deviation of validation accuracies of models with 16 kernels



(c) Average of validation accuracies of models with 32 kernels



(d) Standard deviation of validation accuracies of models with 32 kernels

Figure 3.15: Statistics of models with different kernels and convolutional layers (N=5)

3.3.2 Image flipping, contrast

The effects of the horizontal flipping and contrasting preprocessing layers are examined. Figure 3.16 show the validation accuracies achieved with different models. In Figure 3.17, it can be seen, that flipping the images improves the performance of the model. The contrast changing layer slightly improves the model until the 0.5 value when the flipping

layer is active, but has varying effects when it is not. It also decreases the standard deviation in the first case. For further model analysis, flipping layer and contrast layer with 0.5 value is used.

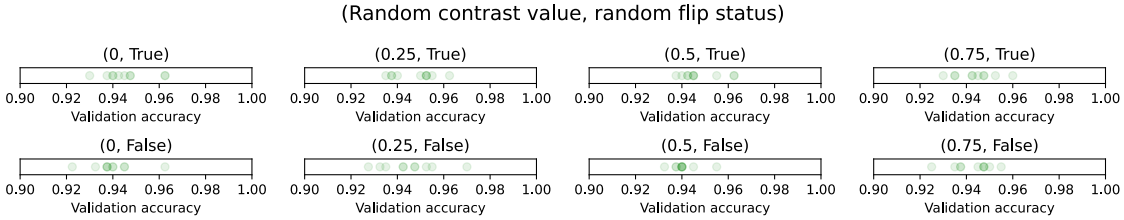
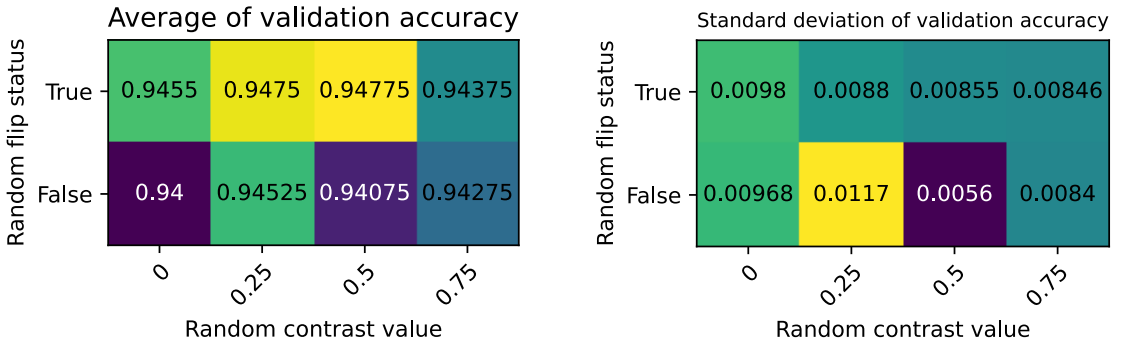


Figure 3.16: Validation accuracies of models with flipping and contrast modifying preprocessing layers



(a) Average of validation accuracies

(b) Standard deviation of validation accuracies

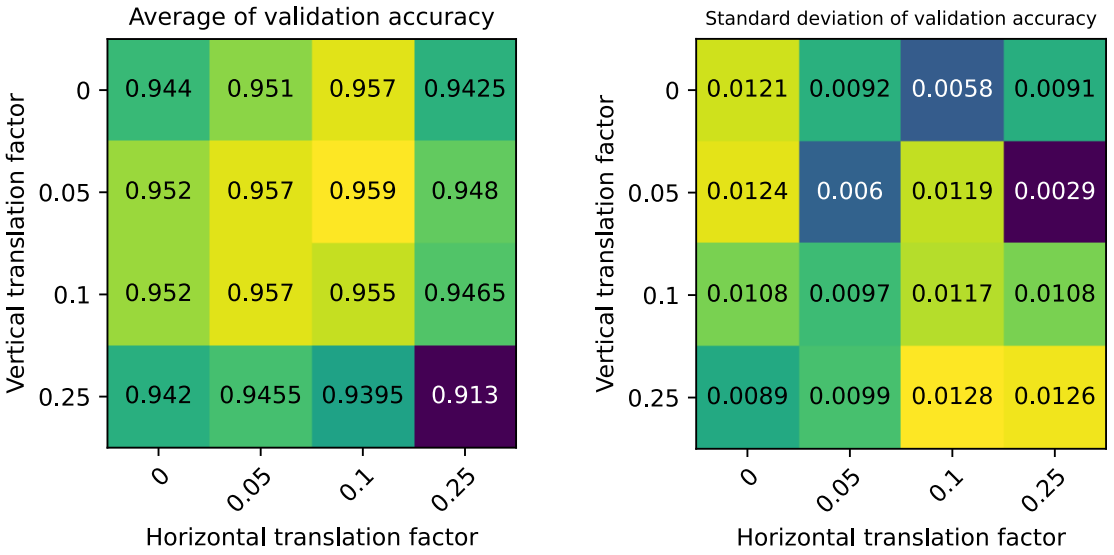
Figure 3.17: Statistics of models with flipping and contrast modifying preprocessing layers (N=10)

3.3.3 Translation

The performance of models with different translation preprocessing layers are tested. Figure 3.18 shows the effect of differently parameterized layers. It can be seen that the augmentations had a positive effect on the performance of the model, in contrary to the case of the fully connected neural network (Figure 3.9), further proving the advantages of CNNs. On the other hand, above a certain translational value, the model started to perform worse, which can again be explained by the initial model selecting process, optimized on the non-augmented images. For ease of implementation, 0.05 value was selected for both horizontal and vertical translation in subsequent models.

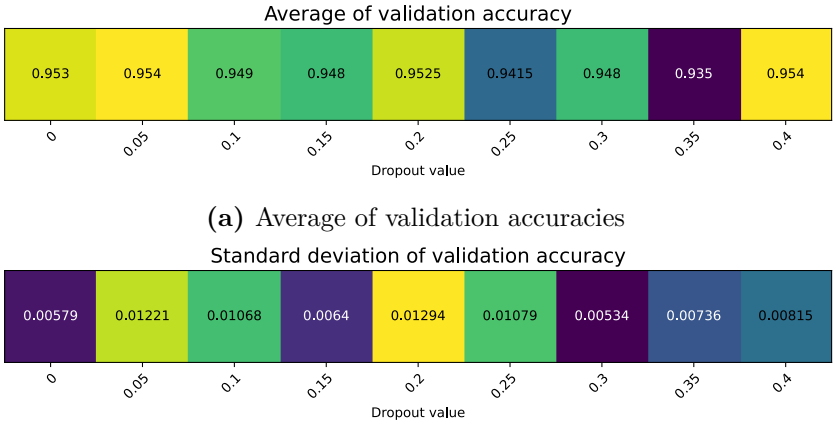
3.3.4 Dropout layers

The effect of dropout layers are examined next. Figure 3.19 shows the performance of models with different dropout layers. No clear relation between the performance and dropouts can be seen, but the models with higher dropout rates took significantly longer to train. Subsequent models don't implement dropout layers.



(a) Average of validation accuracies **(b)** Standard deviation of validation accuracies

Figure 3.18: Statistics of models with translation preprocessing layer (N=5)



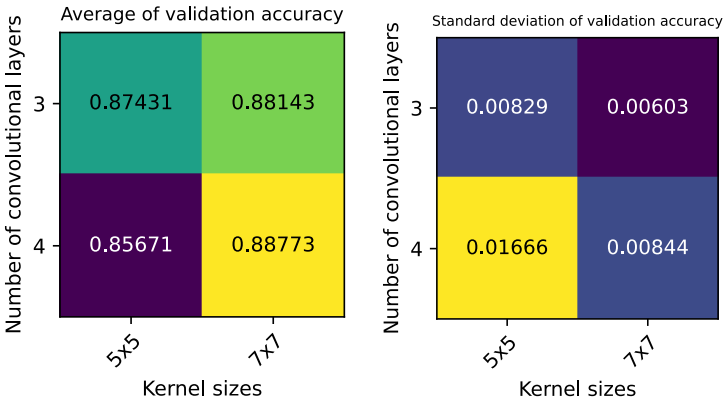
(a) Average of validation accuracies

(b) Standard deviation of validation accuracies

Figure 3.19: Statistics based on the performance measures of models having different dropout layers (N=5)

3.3.5 AM-FED+

Models with the optimized parameters are trained on the AM-FED+ dataset. As its datapoints are different than that of GENKI-4K, the number of convolutional layers and kernel sizes are examined again. Figure 3.20 shows the validation accuracies of the trained models. The model with 4 convolutional layer, and 7×7 kernel size had the best performance, and is used for further analysis.



(a) Average of validation accuracies

(b) Standard deviation of validation accuracies

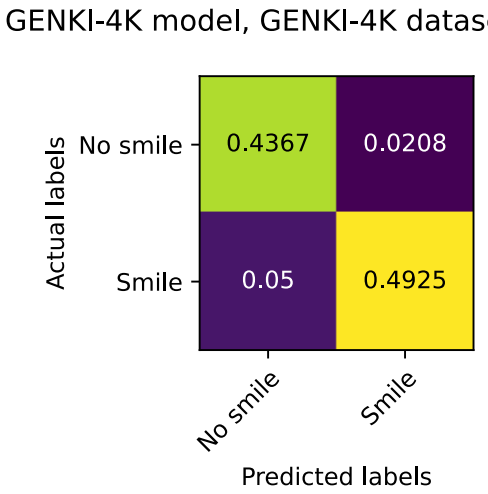
Figure 3.20: Statistics of models with different number of convolutional layers and different sized kernels, trained on the AM-FED+ dataset (N=3)

3.3.6 Evaluation

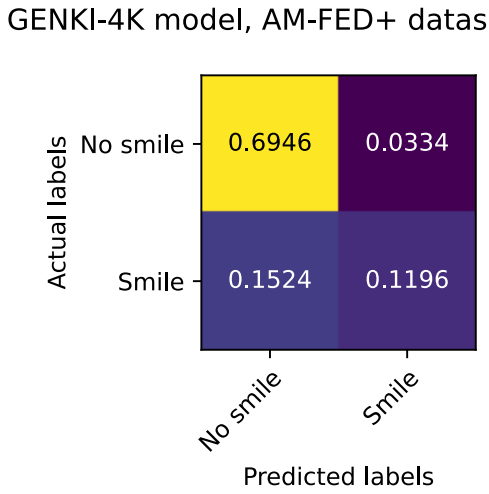
Figure 3.21 shows the accuracies achieved by different models tested and trained on different datasets in the same format as Figure 3.12. Almost all models performed better than their FNN counterpart:

- Models trained and evaluated on GENKI-4K dataset improved from 0.89 to 0.93 accuracy
- Models trained on GENKI-4K dataset and evaluated on AM-FED+ improved from 0.77 to 0.81 accuracy
- Models trained on AM-FED+ dataset and evaluated on GENKI-4K deteriorated from 0.72 to 0.68 accuracy
- Models trained and evaluated on AM-FED+ dataset improved from 0.79 to 0.8 accuracy

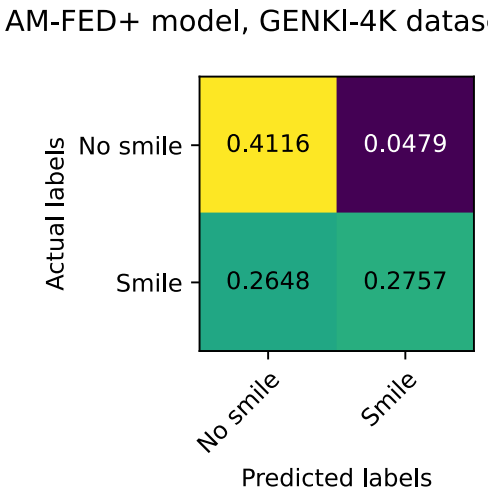
It can be seen once more, that the quality of the GENKI-4K dataset is superior to that of AM-FED+, as the models trained on the former even performed better on the AM-FED+ dataset, than the models trained on it.



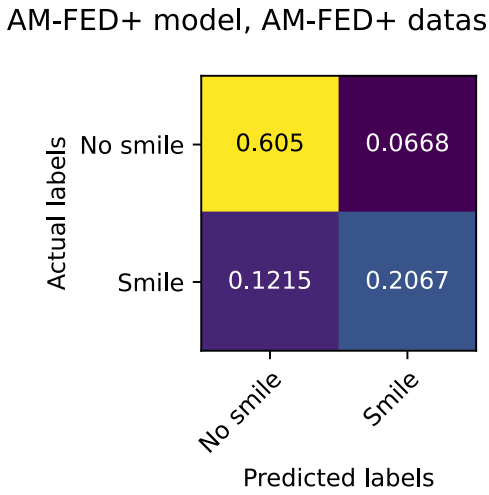
(a) Test accuracy of GENKI-4K model



(b) GENKI-4K model accuracy on AM-FED+ dataset



(c) AM-FED+ model accuracy on GENKI-4K dataset



(d) Test accuracy of AM-FED+ model

Figure 3.21: Confusion matrices obtained on data not seen by the CNN model

3.4 Transfer learning

Transfer learning involves reusing a completed model that was typically trained on a great quantity of quality data. The models used here are made available through the Tensorflow/Keras library.

3.4.1 ResNet models

The ResNet [9] models introduced a revolutionary concept, called "identity shortcut connection". With the help of it, extremely deep neural networks can be trained, as it removes the problem of vanishing gradient.

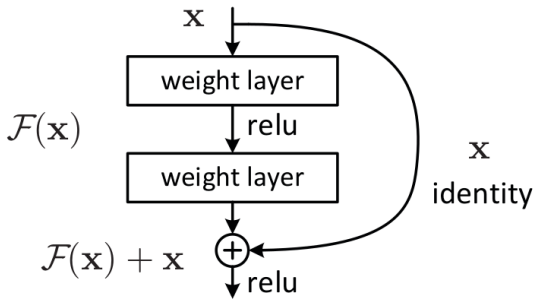


Figure 3.22: Identity shortcut connection introduced in ResNet.
source: [9]

In the thesis the ResNet-50 variant is used, which has 50 layers.

3.4.2 Xception model

The Xception [2] models are built on the foundation of inception modules, first found in the GoogLeNet architecture. Inception modules perform a convolution on the input with not one, but many different sized filters, whose outputs are concatenated and sent to the next layer. The Xception model introduced another interpretation of these inception modules, which performed better than the original one.

Model	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
ResNet50V2	76.0%	93.0%	25.6M	103
Xception	79.0%	94.5%	22.9M	81

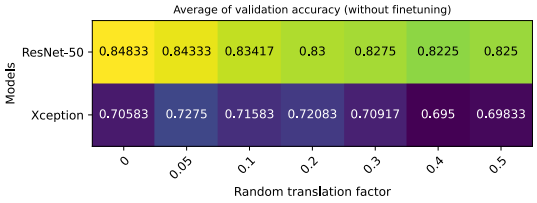
Table 3.1: The models' performance on the ImageNet validation dataset.The top 1 accuracy is the general accuracy, only taking into account the match between the ground truth and the prediction. Top 5 accuracy considers a prediction to be correct if the ground truth label matches any of the 5 highest score predicted labels.

The models were trained on the ImageNet dataset, consisting of 14197122 annotated images, belonging to 1000 different classes.

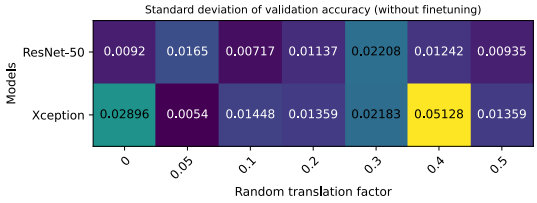
The output layers were changed to accomodate 2 classes for the purpose of smile recognition, but no extra dense layers were added, as the last hidden layer had 2048 neurons, making the usage of extra layers prone to overfitting.

3.4.3 Translation

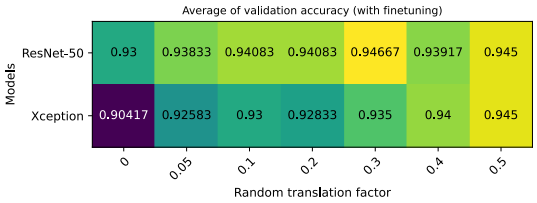
As the random flipping preprocessing layer proved its performance improving effects, the models were equipped with it. Contrary to the previous examinations of the translation preprocessing layer, the horizontal and vertical translations were not examined separately, they were fixed to the same value.



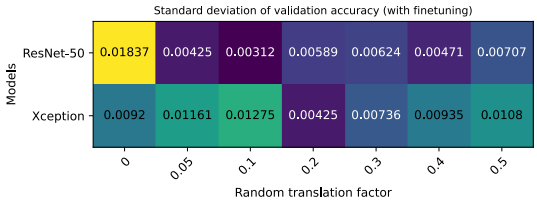
(a) Average of validation accuracies without finetuning



(b) Standard deviation of validation accuracies without finetuning



(c) Average of validation accuracies with finetuning



(d) Standard deviation of validation accuracies with finetuning

Figure 3.23: Statistics based on the performance measures of models having different translation layers, with and without the usage of finetuning (N=3)

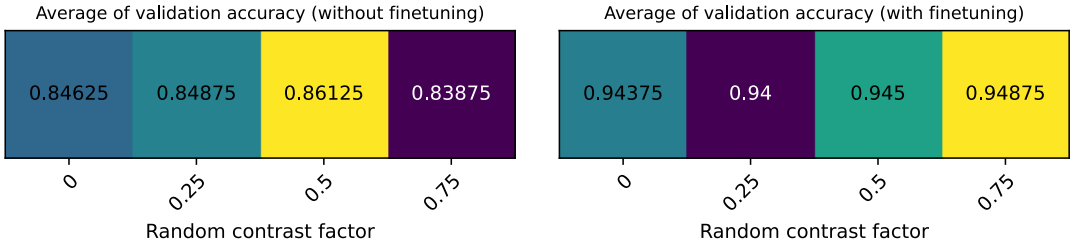
Figure 3.23 shows the validation accuracies of the examined models. Comparing Figure 3.23c to Figure 3.23a, the positive effect of finetuning is apparent, as it increased the validation accuracy by ~ 0.1 and ~ 0.2 in the ResNet and the Xception model respectively.

A positive correlation can be seen between the value of the random translation factor and validation accuracy, contrary to the previously examined CNN models (Figure 3.18), where above the 0.25 mark, the models' performance began to decrease. This can be explained by the pretrained models' training on extensive data, which made it more location agnostic compared to the simple CNN models.

Comparing the validation accuracies with finetuning, it can be seen that the ResNet model outperforms the Xception model, therefore, for further analysis, only the ResNet model is used.

3.4.4 Contrast

Figure 3.24 shows the impact of the random contrast preprocessing layer. The best performing model is the one with the highest contrast modifying layer. Comparing the effects to the ones found in the CNN models (Figure 3.17a), the same situation arises as with the translation preprocessing layer: thanks to the pretraining on the ImageNet dataset, the model is more contrast independent.



(a) Average of validation accuracies without finetuning

(b) Average of validation accuracies with finetuning

Figure 3.24: Statistics based on the performance measures of models having different contrast layers, with and without the usage of finetuning (N=2)

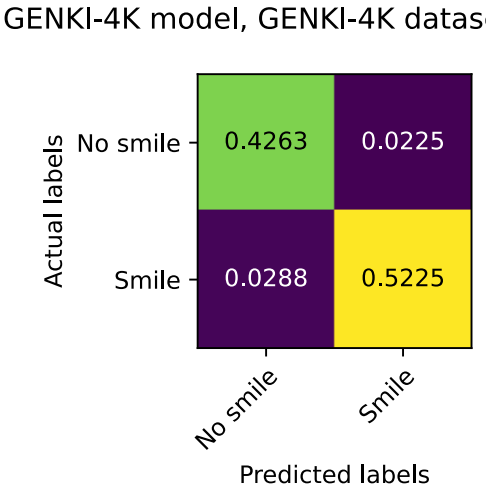
3.4.5 Evaluation

Figure 3.25 show the performance of the obtained models, tested on different datasets. As previously mentioned, the model reacted better to higher levels of image augmentations, as it has more optimal architecture and better initialized weights acquired from the pretraining on millions of images.

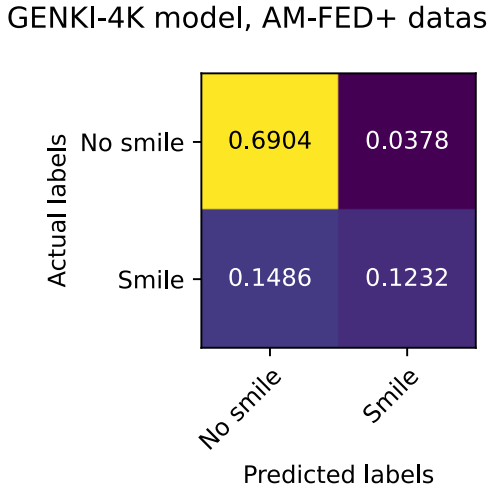
Compared to the accuracies of the simple convolution neural network, all aspect of performance improved as follows:

- Models trained and evaluated on GENKI-4K dataset improved from 0.93 to 0.95 accuracy
- Models trained on GENKI-4K dataset and evaluated on AM-FED+ remained at 0.81 accuracy
- Models trained on AM-FED+ dataset and evaluated on GENKI-4K improved from 0.68 (0.72 with FNN) to 0.85 accuracy
- Models trained and evaluated on AM-FED+ dataset improved from 0.8 to 0.82 accuracy

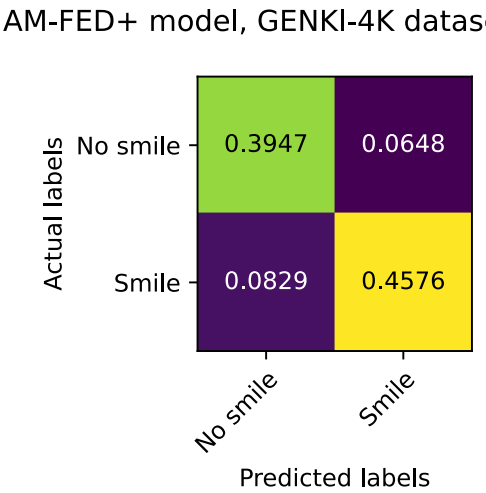
The most notable performance increase happened to models trained on the AM-FED+, whose accuracy increased by ~ 0.13 when evaluated on the GENKI-4K dataset.



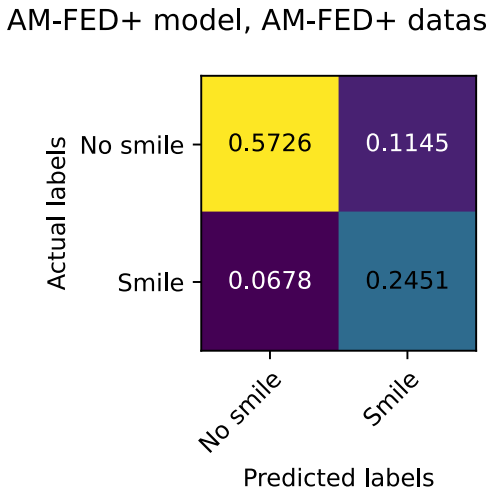
(a) Test accuracy of GENKI-4K model



(b) GENKI-4K model accuracy on AM-FED+ dataset



(c) AM-FED+ model accuracy on GENKI-4K dataset



(d) Test accuracy of AM-FED+ model

Figure 3.25: Confusion matrices obtained on data not seen by the ResNet50 model

3.5 Recurrent neural networks

Recurrent neural networks are trained on sequential data, such as videos. Because only the AM-FED+ dataset consists of videos, the whole recurrent part of the model can only be trained on them.

The examined models are based on the ones found in the CNN section. The last hidden layer of the CNN model is changed to a recurrent layer. The usage of Keras' TimeDistributed layer is necessary, as the model and the recurrent layers expect sequences of images, while the non-recurrent layers expect non-sequential data. Figure 3.26 shows a sample implementation of the discussed RNN models.

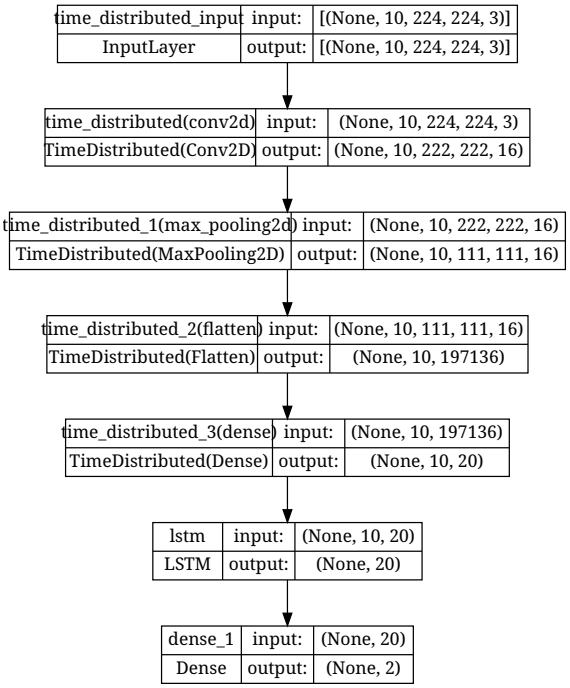
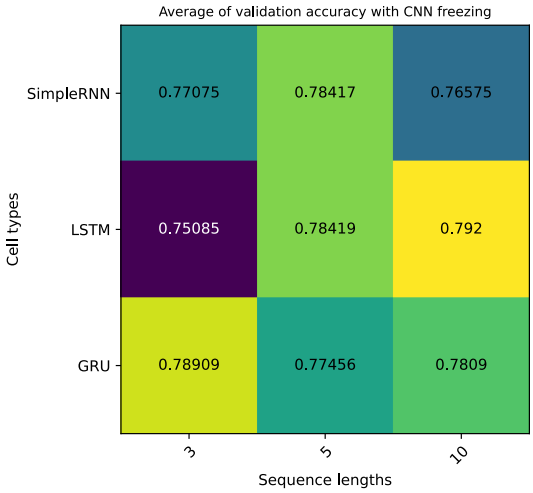


Figure 3.26: A sample implementation of the recurrent neural network, using LSTM cells

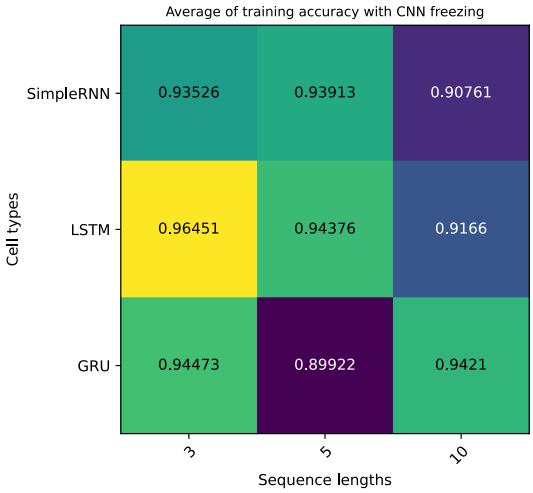
Different recurrent cells [SimpleRNN, LSTM, GRU] and input sequence lengths [3, 5, 10] were examined. Figure 3.27 shows the performance of the models acquired through training. If uninitialized convolutional layers are used, the models learn very slowly, or don't learn at all. To circumvent this, the convolutional layers are pretrained on the GENKI-4K dataset, and then inserted into the recurrent network. The models are then trained in 2 different ways: with the convolutional layers freezed (as in transfer learning), and not freezed.

As seen in Figure 3.27a and Figure 3.27c, the models with freezed CNN layers substantially underperform their non-freezed counterpart. As seen in Figure 3.27b and Figure 3.27d the model was able to learn the training data, but was not able to generalize based on it. The disparity between the training and validation accuracy is apparent, as recurrent cells have greater representational ability, making them more prone to overfitting.

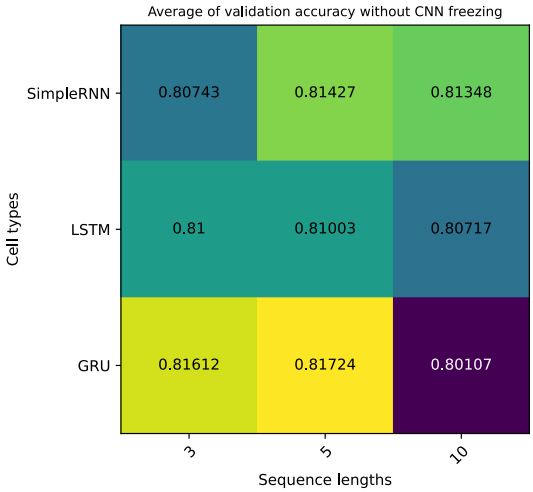
No clear connections were discovered between the examined parameters and the models' performance.



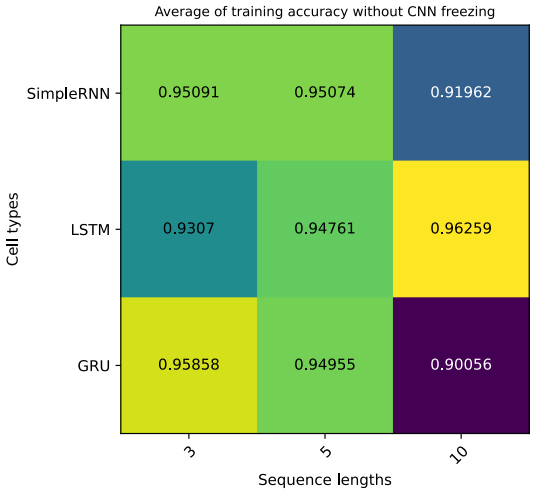
(a) Average of validation accuracies with CNN freezing



(b) Average of training accuracies with CNN freezing at the point where the model had the maximum validation accuracy



(c) Average of validation accuracies without CNN freezing



(d) Average of training accuracies without CNN freezing at the point where the model had the maximum validation accuracy

Figure 3.27: Statistics based on the performance measures of models having different recurrent cells and sequence lengths (N=3)

3.6 Autoencoders

The autoencoder model was trained on the datasets found in Section 2.3.3. The encoder part of the model consists of convolutional and maxpooling layers, flattened at its last layer. The encoded part is a simple dense layer. The decoder part's first layer is a dense layer, which gets reshaped into a 3 dimensional tensor, to serve as input for the following convolutional and upsampling layers.

Models with different dimension of latent space were examined. Figure 3.28 shows the compression and reconstruction ability of the autoencoders, in different latent feature dimensions.

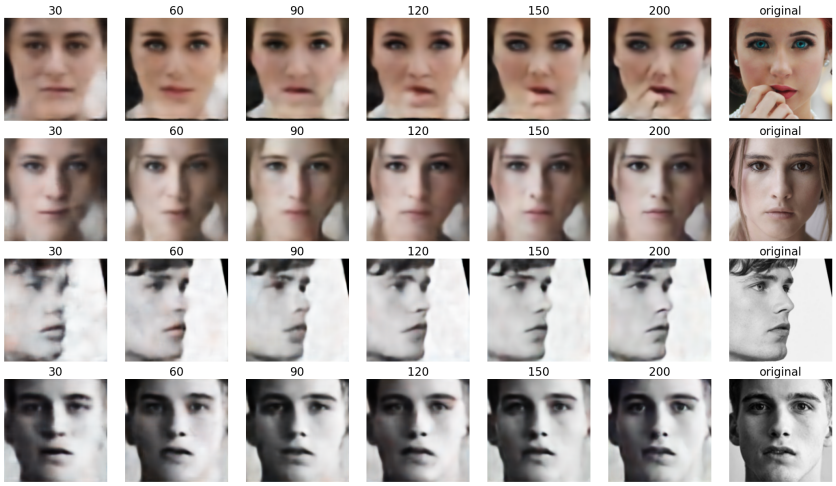


Figure 3.28: The learned representations of the autoencoders

After the autoencoder is trained, its decoder part is discarded, the inference is done using the encoder and encoding part. 2 hidden dense and an output layer is connected to the output of the encoding layer. To limit the possibility of overfitting on the training data, all the newly connect layers have 2 neurons. This means that the maximum number of trainable parameters (in the case of 200 dimensional latent feature space) is 414. For comparison, the ResNet50 model has ∼25 million parameters (Table 3.1). Figure 3.30 shows the validation accuracies of models trained based on different autoencoders. The model trained on the GENKI-4K dataset exhibits a positively correlated connection between the validation accuracy and the value of the latent dimension. The same can't be said about its AM-FED+ counterpart.

3.6.1 Evaluation

To maintain symmetry, models with 200 latent dimensions were evaluated on both datasets. Figure 3.31 shows the obtained accuracies. Based on the values, the models' performance is between the fully connected neural networks and the convolutional neural networks:

- Models trained and evaluated on GENKI-4K dataset had 0.9 average accuracy (FNN had 0.89 and CNN had 0.93)
- Models trained on GENKI-4K dataset and evaluated on AM-FED+ had 0.79 average accuracy (FNN had 0.77 and CNN had 0.81)

- Models trained on AM-FED+ dataset and evaluated on GENKI-4K had 0.68 average accuracy (FNN had 0.72 and CNN had 0.68)
- Models trained and evaluated on AM-FED+ dataset had 0.8 average accuracy (FNN had 0.79 and CNN had 0.8)

Comparing the number of trainable parameter of the autencoder (414) to the parameter numbers of the FNN model (4517792), and CNN model (196802), the autoencoder based model performed very well.

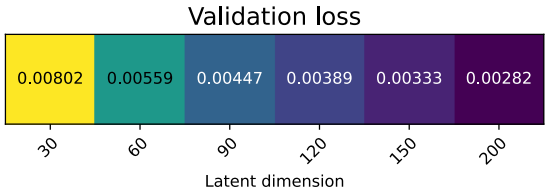
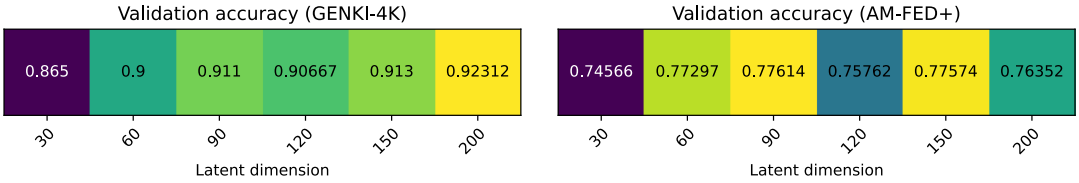


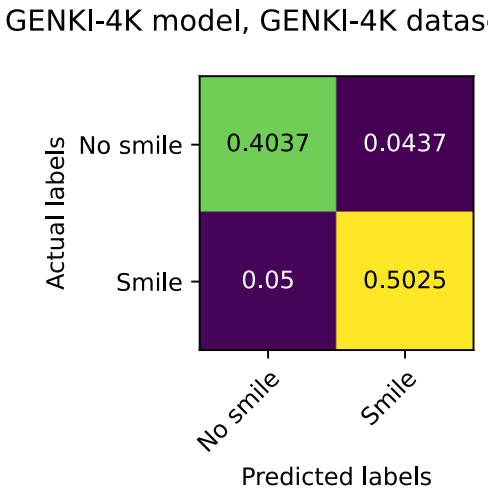
Figure 3.29: The validation losses of the autoencoders with different latent dimensions



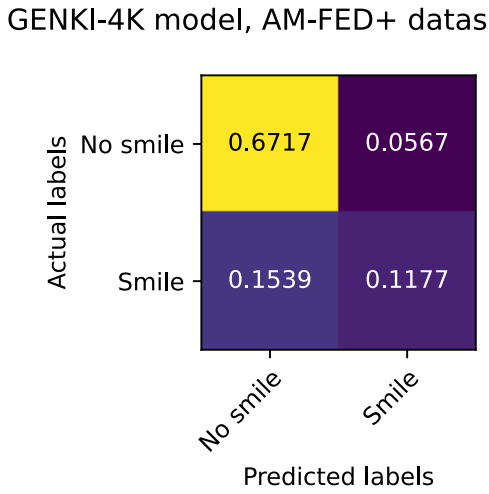
(a) Average of validation accuracies with different latent dimensions, trained on the GENKI-4K dataset

(b) Average of validation accuracies with different latent dimensions, trained on the AM-FED+ dataset

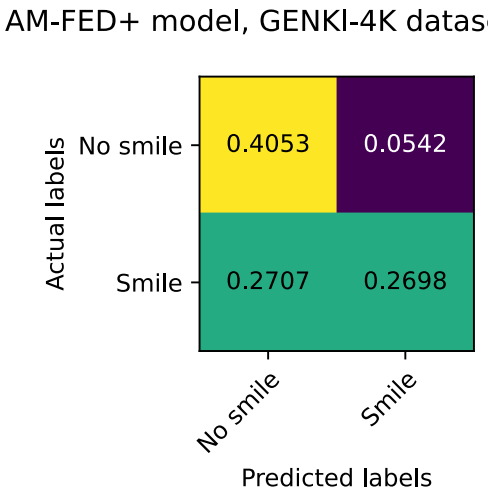
Figure 3.30: Statistics based on the performance measures of autoencoders having different latent dimension (N=5)



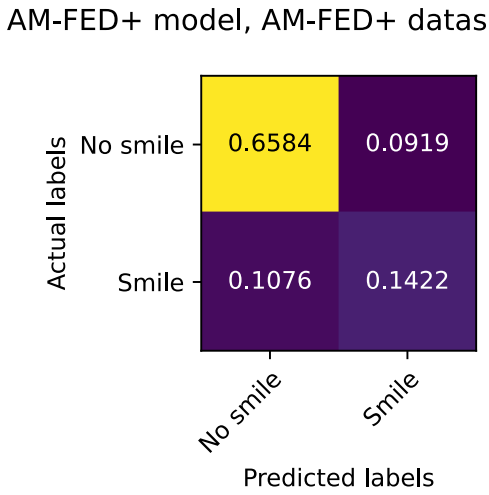
(a) Test accuracy of GENKI-4K model



(b) GENKI-4K model accuracy on AM-FED+ dataset



(c) AM-FED+ model accuracy on GENKI-4K dataset



(d) Test accuracy of AM-FED+ model

Figure 3.31: Confusion matrices obtained on data not seen by the autoencoder based model

Chapter 4

Conclusion

In my work I examined and compared different approaches for smile detection using neural networks. The effects of various model architectures, parameters, datasets and data augmentation operations were studied.

The importance of data quality was many times apparent when comparing models trained on the GENKI-4K and AM-FED+ datasets. The GENKI-4K trained models were often producing better results on the AM-FED+ dataset, than the models trained on it. This is to some extent unexpected, because all of the datapoints found in the AM-FED+ dataset share the same underlying qualities: people watching advertisements, while being recorded by their webcamera. This quality infers traits such as characteristic head posture, or gaze direction. The results obtained from the transfer learning section further prove the disparity of the datasets, as the state-of-the-art ResNet model displayed the same phenomenon, as our simple convolutional neural network implementation.

The fully connected neural network was not able to take advantage of the effects of image augmentations, as it is "structure agnostic", so it couldn't recognize the hidden information found in the images' structure.

The convolutional neural network architecture could take advantage of the information in the structure, as it was specifically made for these kind of applications, but after a certain augmentation factor, it began performing worse. This can be explained by the CNN model's simplicity, as it was selected based on its performance on non-augmented data.

The transfer learning model (using the ResNet50 pretrained weights) performed well, as it has already seen millions of images, from which it has learnt useful feature representations. It also responded well to the increasing level of augmentation.

The examined RNN model was too simple to be able to take advantage of the sequential data, as it overfit heavily to the training data.

The autoencoder model performed very well considering its small number of trainable parameters. Its performance was between the performance of the examined FNN, and CNN model.

Chapter 5

Future works

Performance improvements could be made with the usage of more diverse augmentation operations, such as rotation, shearing, bluring, or noise adding. Software incompatibility and time constraint prevented the implementation of these. The help of subject specific methods, such as the computation of histogram of oriented gradients or other image processing operations could offer valuable informations for the neural networks in the form of feature extraction.

The facial landmark points, which were only used for eye localization, have vast amount of unutilized information. With the help of them, the background of the images could be removed, eliminating non significant input noise, or a high level structure of the face could be engineered.

Further examination is needed in the case of the recurrent neural network architecture, as it is almost the simplest implementation possible. The need for engineering extra features incorporating the changes between the frames (e.g. optical flow) is highly likely.

The usage of the autoencoder model has great potential, different architectures could be investigated, in both the encoder and decoder parts, as only the dimensionality of encoding layers were examined.

Acknowledgements

I would like to thank Gábor Révy and Dr. Gábor Hullám for their help and support during the preparation of this thesis and the related research.

Bibliography

[1] Simone Bianco, Luigi Celona, and Raimondo Schettini. Robust smile detection using convolutional neural networks. *Journal of Electronic Imaging*, 25:063002, 11 2016. DOI: 10.1117/1.JEI.25.6.063002.

[2] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1251–1258, 2017.

[3] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[4] Wikimedia Commons. Gated recurrent unit, 2018. URL https://commons.wikimedia.org/wiki/File:Gated_Recurrent_Unit,_base_type.svg.

[5] Wikimedia Commons. The lstm cell, 2018. URL https://commons.wikimedia.org/wiki/File:The_LSTM_Cell.svg.

[6] Paul Ekman and Wallace V Friesen. Facial action coding system. *Environmental Psychology & Nonverbal Behavior*, 1978.

[7] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors). *The annals of statistics*, 28(2):337–407, 2000.

[8] Patrick O Glauner. Deep convolutional neural networks for smile recognition. *arXiv preprint arXiv:1508.06535*, 2015.

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

[10] Gary B. Huang, Manu Ramesh, Tamara Berg, and Erik Learned-Miller. Labeled faces in the wild: A database for studying face recognition in unconstrained environments. Technical Report 07-49, University of Massachusetts, Amherst, October 2007.

[11] Takeo Kanade, Jeffrey F Cohn, and Yingli Tian. Comprehensive database for facial expression analysis. In *Proceedings fourth IEEE international conference on automatic face and gesture recognition (cat. No. PR00580)*, pages 46–53. IEEE, 2000.

[12] Yury Kartynnik, Artsiom Ablavatski, Ivan Grishchenko, and Matthias Grundmann. Real-time facial surface geometry from monocular video on mobile gpus. *arXiv preprint arXiv:1907.06724*, 2019.

[13] Fumito Kawakami and Takumi Yanaihara. Smiles in the fetal period. *Infant Behavior and Development*, 35(3):466–471, 2012.

[14] Vahid Kazemi and Josephine Sullivan. One millisecond face alignment with an ensemble of regression trees. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1867–1874, 2014.

[15] Davis E. King. Dlib-ml: A machine learning toolkit. *Journal of Machine Learning Research*, 10:1755–1758, 2009.

[16] Tara L Kraft and Sarah D Pressman. Grin and bear it: The influence of manipulated facial expression on the stress response. *Psychological science*, 23(11):1372–1378, 2012.

[17] Camillo Lugaresi, Jiuqiang Tang, Hadon Nash, Chris McClanahan, Esha Uboweja, Michael Hays, Fan Zhang, Chuo-Ling Chang, Ming Guang Yong, Juhyun Lee, et al. Mediapipe: A framework for building perception pipelines. *arXiv preprint arXiv:1906.08172*, 2019.

[18] S Mohammad Mavadati, Mohammad H Mahoor, Kevin Bartlett, Philip Trinh, and Jeffrey F Cohn. Disfa: A spontaneous facial action intensity database. *IEEE Transactions on Affective Computing*, 4(2):151–160, 2013.

[19] Daniel McDuff, Rana el Kaliouby, Thibaud Senechal, May Amr, Jeffrey F. Cohn, and Rosalind Picard. Affectiva-mit facial expression dataset (am-fed): Naturalistic and spontaneous facial expressions collected "in-the-wild". In *2013 IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 881–888, 2013. DOI: 10.1109/CVPRW.2013.130.

[20] Daniel McDuff, May Amr, and Rana El Kaliouby. Am-fed+: An extended dataset of naturalistic facial expressions collected in everyday settings. *IEEE Transactions on Affective Computing*, 10(1):7–17, 2018.

[21] T MPLab. The mplab genki database, genki-4k subset, 2009.

[22] Signe Preuschoft. Primate faces and facial expressions. *Social Research*, pages 245–271, 2000.

[23] Dale Purves. *Brains: how they seem to work*. Ft Press, 2010.

[24] http://mplab.ucsd.edu. The MPLab GENKI Database, 2009.

[25] Jacob Whitehill, Gwen Littlewort, Ian Fasel, Marian Bartlett, and Javier Movellan. Toward practical smile detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 31:2106–2111, 2009.

[26] A Yazidi, R Goyal, A Paes, N Gruber, NG De, and A Jockisch. Are gru cells more specific and lstm cells more sensitive in motive classification of text?, front. *Artif. Intell*, 3(40):10–3389, 2020.