MUNDBINET: ITERATIVELY OPTIMIZING THE TRAINING OF A CNN TO CLASSIFY COVID-19 FACE MASK PLACEMENT

Jacob Qvist Jensen
Department of Computer Science
Aarhus University
Denmark
201508195@post.au.dk

Morten Astrup
Department of Computer Science
Aarhus University
Denmark
201705289@post.au.dk

Martin Kjær
Department of Computer Science
Aarhus University
Denmark
201709372@post.au.dk

May 2, 2022

ABSTRACT

In this paper it is described how a Convolutional Neural Network called *MundbiNet* has been developed to classify the placement of a face mask given an image. Through the development and testing of different versions of the *MundbiNet* model, the importance of well fitted data and regularization became clear. The use of a homogeneous dataset generated from an AI model augmented with face masks using a third party face-detection CNN, tended to make the model overfit which was discovered during several different experiments. This lead to experiments and discussions on pre-processing and regularization on the training data in order to make the model perform better on unseen data. One of the major issues of the model was the poor performance on images with bad lightning. The result of the final model and its performance is described using different visualization tools like, heatmaps and a t-SNE model. The conclusion on the performance of the model is that it has great performance on unseen data under the right conditions, but still has some limitations on the performance on images from the wild, which is likely due to the fact that the training data is very generic without much variation.

***K*eywords** MundbiNet · CNN · Multi-class classification · Heatmaps · SoftMax · Batch normalization · t-SNE · COVID-19

1 Introduction

Severe acute respiratory syndrome coronavirus 2 (SARS-CoV-2) has nearly paralyzed most of the society since March 2020. Restrictions are rolled out by the governments all over the World to prevent a rapid spread of the severe coronavirus 2. Early on in the crisis face masks were worn by the inhabitants of Wuhan, which was formerly known as the epicenter of the virus. Later on at the 20th of July 2020 it became mandatory to wear a mask in all public buildings in France[1]. This restriction later became mandatory in Denmark as well. A face mask can be worn in many different ways, but if it is placed incorrectly it looses its effect[2]. As it has become a tendency that people wear the mask incorrectly, some governmental institutions have chosen to produce material on how not to wear a face mask[3]. Why do the governments not use a convolutional neural network optimized for predicting the placement of a mask? This tool could be used to monitor how many people are wearing their mask incorrectly or help people adjusting to the correct and safe placement of the mask. In this project we present *MundbiNet*.

The purpose of project *MundbiNet* is to develop and train a convolutional neural network that can predict four different placements of the face mask, which are:

[1] http://thinkeuropa.dk/politik/tidslinje-over-coronakrisen-hvad-skete-der-og-hvornaar

[2] https://www.hopkinsmedicine.org/health/conditions-and-diseases/coronavirus/proper-mask-wearing-coronavirus-prevention-infographic

[3] https://www.rochesterregional.org/news/2020/07/how-not-to-wear-a-mask

- **Correct**
- **Below Nose**
- **Chin**
- **No Mask**

The first one is called **Correct** and represents all correctly placed masks, which is a mask that covers nose, mouth and chin as described by Apotek[4]. The second one is called **Below Nose** and is a variant were the nose is free form the mask, but the rest is covered. The third variant is called **Chin** and it only covers the chin. The last variant is called **No Mask** and that is a face without a mask.

We chose to create live video functionality, such that it is possible to predict the mask position through a webcam in real-time. This was built based on an existing model described in subsubsection 3.3.2 that uses a binary classifier. By manipulated the python script it was adapted to function with *MundbiNet*.

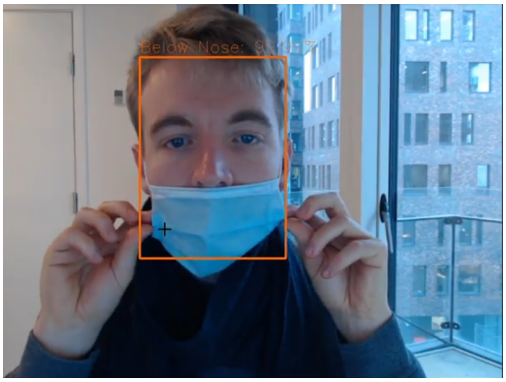

Figure 1: Image of the live video prediction

The CNN in this project is based on knowledge gained through the course of Deep Learning for Visual Recognition at Aarhus University, which have been taught in a top-down approach.

The research questions of the project are as follows:

- Is it possible to build a multi-class classifier that can classify 4 different variations of mask placement?
- How accurate is the classifier on images it has never seen?
- How well is the classifier at predicting on real images, when it has been trained on generic AI generated images?

An answer to the research questions are presented in section 6 based on the classifier that has been fine-tuned over several iterations in subsection 3.3.

2 Related work

LeNet-5[1] was the first convolutional neural network to be published as scientific work and changed the way of training a neural network. The main idea behind this CNN was to predict the numbers of a financial check, to make this task automated instead of manually reading the numbers. Before LeNet-5, visual recognition was an expert machine learning task which included feature engineering by hand. LeCun et al. presented a CNN that was built on 3 main ideas: Spacial sub-sampling, local receptive fields and shared weights. Local receptive fields combined with shared weights are crucial for the convolutional layers in most CNNs and LeCun et al's work paved the way for the field of deep learning for visual recognition. Their results were compared with different other classifiers as e.g. earlier versions of LeNet, K-NN classifiers with a Euclidean distance measure between input images and also simple linear classifiers. LeNet-5 achieved an error rate below 1%, which at the time of doing the research was nearly the best achieved. Despite the good results, the architecture of LeNet-5 was not perfect.

A lot has happened since the early days of LeCun et al's LeNet-5. Computation power is much cheaper and it is therefore possible to train models faster and also more complex. Since deep learning is used in many different cases it

[4]https://www.apoteket.dk/sundhed/forebyggelse/brug-af-mundbind

can sometimes be hard to find large and also reliable datasets. Pre-processing and data augmentation are widely used techniques for tweaking and extending the datasets by changing the images. Data augmentation can for instance be used to enlarge the dataset, which is commonly used for smaller datasets[3]. Being able to understand what the triggers the model can also be crucial. Heatmaps can be used to figure out what a deep learning model have actual learned by visualizing it for the developer[4].

The latest research shows that the pandemic outbreak also pushes the limits of deep learning research. M. Loey et al[2] presents their work on how a hybrid model that combines deep learning and traditional machine learning can predict whether a person has a mask on or not. They look at different datasets, where 1/3 is images of real people with real masks and 2/3 are generated. The World is not as binary as M. Loey et al presents it. Humans are known to create uncertainty and are not programmable machines that only have a mask correctly placed or are not wearing it. A campaign in America trained the population to wear the mask incorrectly[5], which makes the mask ineffective. Instead of creating a binary classifier, we believe that governments have to be able to predict how the mask is placed. Also, to evaluate our model, we will use heatmaps to show what triggers the classifier, where M. Loey et al only look into accuracy since they are trying to get a better accuracy compared to other researchers.

3 Methods

This section describes the methods that characterises the project. Firstly, the network architecture is described in subsection 3.1. In subsection 3.2 the dataset is described and analyzed. Afterwards in subsection 3.1 the network architecture is explained. Lastly, the experiments that were carried out in the project are described in subsection 3.3.

3.1 Network architecture

The network architecture in Figure 2 is based on several iterations from subsection 3.3. The architecture is inspired from a Github Repository of a *Face Mask Detector*[6] by Chandrika Deb, and our experiments share a lot of code with this repository. Especially when predicting from images and through the webcam. The many iterations have shaped the architecture and two parts separates this model from most of the other explored 1) Batch normalization layer and 2) Dropout layer. Batch normalization enforces zero mean and unit variance. By normalizing activations throughout the network, batch normalization prevents small changes to the parameters, which potentially could have been one of the issues early on in subsection 3.3. Dropout enforces the network to not only rely on a single neuron, which could have been the case in the early experiments in subsection 3.3. Batch Normalization and Dropout were both implemented as regularisation techniques as an effort to decrease how overfit the model was. Our network ended up being very shallow, which is also true for many other face mask detectors we found online. Our reasoning behind this is that we did not want the network to over interpret many of the details. Its only task is to spot a face mask and its position. It must be said that all these changes to the model over time is based on the material presented in the lectures and the model therefore also evolved with the progress of the semester.

3.2 Dataset

The dataset consists of 40.000 images from `www.generated.photos`, which is a service that sells AI generated images for deep learning. Luckily, they provide a free set of images for research purposes, which we applied for. We received 10.000 images, which we used for creating our dataset that contains 4 different labelled categories: 1) Without mask 2) With mask perfect placed 3) With mask placed below nose 4) With mask placed below mouth. In Figure 3 a snippet of the images with the four different categories can be seen, but with a higher resolution. The images in the dataset are 256x256 pixels and are in *.jpg* format. The dataset is split into a training and a testing set. 80% of the dataset is placed in training and 20% is in testing. We are aware of the fact that it is better to split the dataset into 3 different sets: training, validation and test, but we have chosen to extract a little mini-set for the validation part with only 16 images, which is 4 from each category. Our dataset is not that big and this is mainly the reason by the decision. Cross validation could also have been an opportunity since we have a small dataset, but as it is not frequently used in deep learning, we chose not to use it. While working with the data, we have created a set of assumptions about the data our model should be able to handle and what not to:

1. Should be able to handle blue face masks.
2. Should be able to handle studio lighted images of faces.

[5] https://www.forbes.com/sites/mishagajewski/2020/07/11/were-trying-to-get-people-to-wear-masks-the-wrong-way/?sh=5e818ddd232e

[6] https://github.com/chandrikadeb7/Face-Mask-Detection

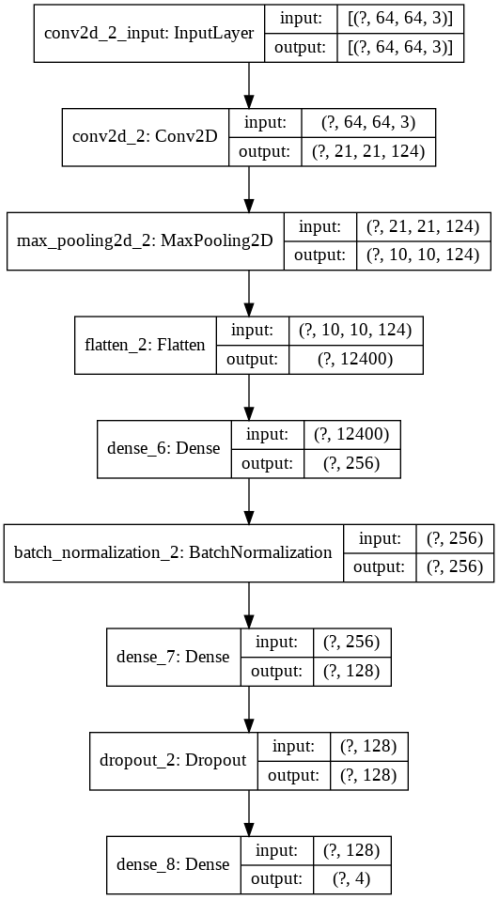

Figure 2: CNN architecture

3. The model is not build to being able to predict any arbitrary kind of faces. This means that we have not focused on generalizing the dataset, but we still have all races and ages represented in the dataset. For instance, it is not a goal to be able to predict a face from the side.

4. Should be able to handle sheared images, which was handled by a shear range of 0.1.

5. Should be able to handle different zoom levels.

6. Should be able to handle differently scaled images, which was handled with a rescale of 1./255.

Data augmentation of the dataset is described in subsubsection 3.2.1 and pre-processing of the dataset is described in subsubsection 3.2.2.

3.2.1 Data augmentation

The first and most significant change to the dataset is the augmentation of face masks, which extended the dataset from 10.000 to 40.000 images and thereby extended the dataset with 3 extra categories of people with face mask placed in different positions. A python script was developed that used a face recognition model to predict positions of crucial points in the facial region. These points were the position of the tip of the nose, the center of the chin and the center of the lower lip. This allowed the script to use three different strategies, one for each class, to augment the mask on the input image. Other parameters as the rotation of the head was also taken into account when, the masks were mounted. Before the training of the model started, we built a script that cropped out the face of the person since this was the only part of the image that had to be used for the training. This was chosen to lower the background clutter effect on the results and it also created an opportunity to use the prediction model on other images than the ones in the dataset.

Figure 3: One image from each category of the same "person". From left: No mask, perfect placed, below nose and below mouth.

3.2.2 Pre-processing of images

In the following section we describe how we have pre-processed the images before feeding them into the network. In order order to make the model less prone to overfitting, we have applied different pre-processing techniques to the dataset. The process of choosing pre-processing techniques are explained in subsubsection 3.3.5.

After some early testing on the model and applying a heat map to the output it became clear that the model was focusing too much on edges in the images and there was a need for data to be pre-processed to address this problem that could lead to overfitting.*Gaussian blur* was applied to the images in order to change model from looking at these high frequency features like the sharp edges from the augmented masks data. Adding this blur to the images prevented the model from learning these high frequency features that is irrelevant for the model and caused overfitting. For the same reason *Salt & pepper* was applied. This pre-processing sets random pixel values to 0 or 255. This should change the focus of the model from looking at specific pixels and make decisions based on these. Later we discovered that there is a function in Keras called *ImageDataGenerator*, that makes it possible to automatically pre-process the data-input with some given probability. The model runs different types of pre-processing on the training data to make train the model to perform better on new unseen data. The first pre-process is called *Shearing*. This makes the image distorted along an axis in a given range. This created perception angles in the images and will help the model perform better on images from different angels.

Figure 4: Pre-processed images which the model would train on

Another pre-processing that is performed in the ImageDataGenerator is *Zoom*, this will make the image zoom random within a given range. This will train the model to perform better on input images with different scaling. *Height shift* make the image shift random en the vertical-direction inside a given range. This again adds some randomness to the data.

Pre-processing using *rotation* on the data, make the image randomly rotated within a range given in degrees. This allows the model to learn on images with different orientation. In this case the model will also train on images where people are tilting their head. To overcome the fact that pictures are captured in different types of lightning, some of the input images are added random *brightness* level within a given range. This is important for the model to also perform well on unseen data with another level of brightness. *Horizontal flip*, flips the picture horizontal randomly. This is useful in our case because the augmented mask data is not symmetric. Horizontal flipping gives the dataset extra randomness. It would not make sense to flip vertically with random probability because it is unlikely to get an input image of a face upside down.

3.2.3 Pre-processing when predicting

When actually predicting images, most of the pre-processing was omitted, since that was just regularisation techniques when training. However, there are some types of pre-processing that is important to have both when training and predicting. The trained images had been cropped such that only the face was trained on. This was an important element to replicate since the model has not been trained to handle background clutter. So using another CNN called 'Face Recognition'[7], the faces of all people in the image would be detected and cropped. Using *MundbiNet* these cropped images would be predicted. Doing this also drastically reduces the amount of training and data the CNN would need to give good results, since it only needs to focus on faces and not entire images.

3.3 Experiments

In the following, we will present the experiments that we have carried out through the project. Since the course was structured as a top-down approach, we have increased our skill set as the project evolved. This also meant that we have learned from early implementations and adapted. We will discuss this in the following.

3.3.1 The initial experiment

The initial plan was to create a project combining the Internet of Things with Deep Learning by running the model in the cloud and using Raspberry Pi[8] with a camera as input. The first experiment consisted mainly of testing pre-trained models on a Raspberry Pi. Different kinds of face detection models were tested, and a model called 'Face Recognition'[9] showed promising results. This made it possible to run the model on a raspberry pi, but only with 1 frame per second on the camera. The usage of the camera was chosen to make it possible to build a fully fledged system that could be evaluated in the wild. During this phase, it became clear that we had to focus more on the model itself, and less on everything else. The key findings from these initial experiments was the knowledge we gained from trying out different pre-trained models, and finding which models worked well. We found that the model 'Face Recognition' [10] worked very well.

3.3.2 The second experiment

Before starting with the second experiment we tried found a model and trained it based on code from PyImageSearch [11]. Parts of the code could be reused, but the training of the model had to be modified to fit the purpose of our case. We rewrote the binary classifier to a classifier that could potentially classify many different classes. Our goal was to be able to predict if the face in a given image had a mask placed correctly, and if it was not placed correctly how was it then placed? People tend to place it below the nose or under the mouth, so we included these as two separate classes. Additionally we added correctly positioned masks as a class, and masks placed on the forehead (later removed) as a class. After training the model we got not so promising results as seen in Figure 5. At this time our dataset was not very good and small as it only contained 300 images of poor quality as seen in Figure 6.

[7] https://github.com/ageitgey/face_recognition

[8] https://www.raspberrypi.org/products/raspberry-pi-4-model-b/?resellerType=home

[9] https://github.com/ageitgey/face$_r ecognition$

[10] https://github.com/ageitgey/face$_r ecognition$

[11] https://www.pyimagesearch.com/2020/05/04/covid-19-face-mask-detector-with-opencv-keras-tensorflow-and-deep-learning/

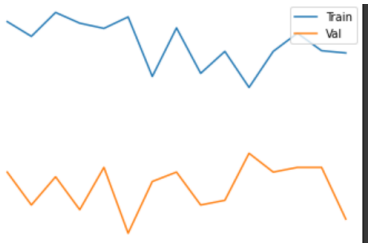

Figure 5: Results from the second experiment with no x and y axis

Figure 6: A snippet of images from the initial dataset (source: `https://www.pyimagesearch.com/2020/05/04/covid-19-face-mask-detector-with-opencv-keras-tensorflow-and-deep-learning/`)

3.3.3 Larger dataset

As we continued the studies, we gained knowledge on how to increase the accuracy in a convolutional neural network. Our first step to create a better model was to get a larger dataset. After long search we figured that it was not possible to find a labeled dataset of people wearing masks in different positions. We therefore wrote a python script that could mount masks on images of people as described in subsubsection 3.2.1. We found a dataset of AI-generated images of people and applied for access, which was granted. The dataset consisted of 10.000 images of people on a white gray-white background with all races, genders and many different ages, see subsection 3.2. The dataset is generated by another neural network and the source can be found at https://generated.photos/. The results in Table 3.3.3 were slightly better than the test before. We quickly figured that the lack of accuracy was do to the way we evaluated on the test set, which we fixed in subsubsection 3.3.4.

3.3.4 Overfit model

The problem in subsubsection 3.3.3 was that in the evaluation of the test set since we trained the model with images that had been cropped to show only the face. The images we predicted on had not been cropped. Therefore we utilised the previously mention 'Face Detection' package to utilise its ability to detect and crop faces. We then looped over every output image, which is a cropped image of the detected face. By doing this, we saw a significant increase in accuracy.

Note that this altering of the model while training was done early in the course and we later got a better strategy for adjusting the model to perform better.

`5 Epochs, 10 StepsPrEpoch, Batch Size 32`

We started with few epochs and low learning rate and small batch size to get an idea on how the model would perform before training the model with full capacity. The model was very good at finding some classes. Perfect placement of the

Correct	27 %
Below Nose	25 %
Chin	27 %
Only Nose & Mouth	21 %

Table 1: Accuracy for Correct, below nose, chin and only nose and mouth

mask had an accuracy of 65%, which was much higher than the tests in subsubsection 3.3.3 but still bad for a classifier. Below nose was often classified as chin. The after we increased the number of epochs and steps pr epoch for better training, the accuracy of the model also increased.

`20 Epochs, 50 StepsPrEpoch, Batch Size 64`

In this sub-experiment we increased the amount of epochs, steps pr epoch even more and we also increased the batch size. This resulted in an overfit model, that had a high accuracy. A visualisation of the results (accuracy and loss) can be seen in Figure 7. The results clearly shows that 20 epochs in this setup is too much compared to our dataset. We knew at this time that we had to focus more on regularization techniques in order to get rid of the overfitting, which we focused on in subsubsection 3.3.5.

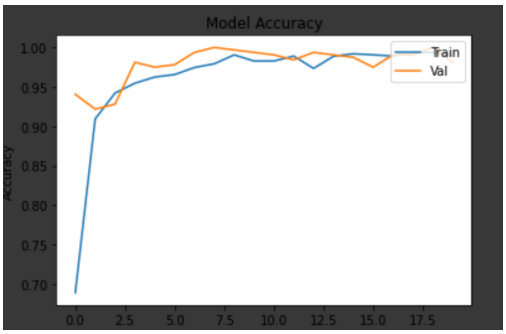

Figure 7: Overfit but accurate model

3.3.5 Implementing heatmaps and regularization techniques

In this experiment section we will explain the process of optimizing our model to reduce overfitting. The process was long and involved, looking into heatmaps, gaussian blur, salt and pepper, shear, rotation, vertical flip, rescale, brightness, dropout and also tweaking the amount of epochs, steps pr epoch and batchsize according to the new kind of data. We wanted the model to reach a high accuracy more slowly with the hypothesis that the result would be a more robust model. The first thing we looked into was heatmaps, because we wanted to get a clear picture of how the model predicted the class of the image. This was used actively in the process of determining how well the model was doing and also helped us to determine that our model was overfit in the start. In Figure 8 two heatmaps in a superposed version of the same face is shown and one reference image of a correctly predicted perfectly placed mask. To the left with no mask, in the middle with perfect placed mask that was predicted as being below the nose and lastly to the right a reference of how the model usually predicted a perfectly placed mask. The model only looked at the mask or the face. We wanted the model to look at both the mask and face since we believe that this would be an indication of a less overfit model.

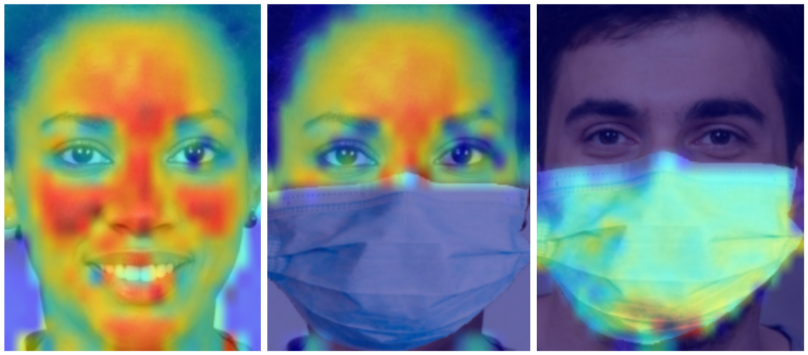

Figure 8: Difference in what triggers the model

The idea of using heatmaps appeared when we manually analysed some images as seen in Figure 10. The lower left prediction is clearly wrong, but still has a high accuracy. Therefore, we investigated it further and all agreed to look more into regularization techniques to make the model better. We found that with the incorrectly labelled image in

Figure 9: Focus on the bottom left image

Figure 10, the model would only look at the face and ignore the mask, as seen in Figure 8, while it would only look at the mask in the correctly labelled images with the label 'Correct'.

3.3.6 Systematic approach to increase accuracy

In this series of experiments, we attempted to explore new approaches to improve the accuracy of our model. In the previous iterations we have had problems with the incorrect predictions having a very high First we attempted to make the neural network deeper, by adding additional convolutional layers. The reasoning behind this was to increase the neurons receptive fields, allowing them to detect larger features. But we could not achieve a model more accurate than in our previous experiments. In fact, the resulting model could not predict anything.

Reverting our changes, we increased the intensity of our pre-processing. Our dataset had already received some pre-processing in the form of salt and pepper and a minor gaussian blur subsubsection 3.2.1. Apart from this, we increased the intensity of rotation, lighting changes, shear and zoom. This resulted in a model with much less overfitting. When we tried to predict images with the trained model, the results were very inaccurate. To combat a problem with the face detector cropping the faces with different height we attempted to add a height shift. This also lead to very low accuracy. At this point we found that we had misunderstood one of the standard parameters in Keras ImageDataGenerator - Namely vertical-flip, which we had included as a parameter. This meant that half the images had been flipped to be upside-down instead of just being mirrored as we had thought. Removing this restored our high accuracy. We settled for relatively low pre-processing:

Rescale	Shear Range	Zoom Range	Rotation Range	Brightness Range
1./255	0.1	0.2	20	[0.6, 1.3]

Our next approach was to see the difference in accuracy when altering the dimensions of the input image. Most of the models that have been trained previously have been with 64x64 input images. First we attempted to train the model with 256x256 images while making the convolutional layer three layers deep. We received poor results. Reducing the image size to 32x32 with a single convolutional layer gave great results. A reason behind this could be that in this particular use case, the network does not need many features in the data to recognise the position of the mask. Therefore a larger image with more details could confuse more than could help. To many redundant features in the data could result in more overfitting.

3.3.7 Fine-tuning

When training, the model tended to overfit in the last epochs. We noticed that by looking at the loss curve on the validation set started to increase. This gave us the idea of using early-stopping. Just stop the training as soon as the loss starts increasing.

We also introduced batch normalization in our model to allow us to train the model with higher learning rate. This is described further in subsection 3.1.

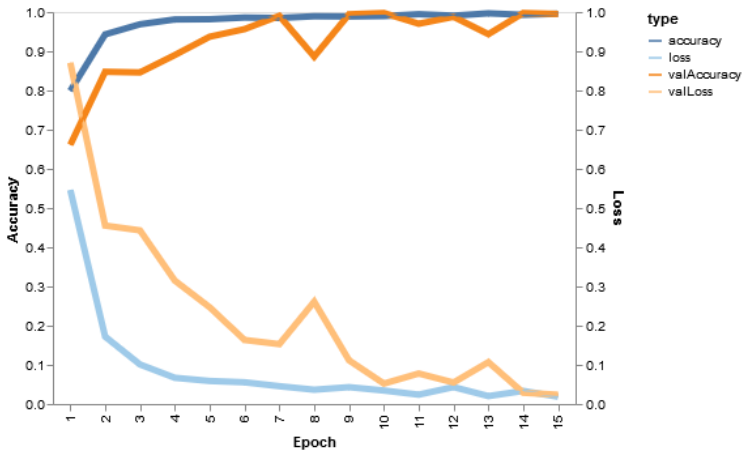

Figure 10: Loss and Accuracy for the final model

After getting the model to perform very well on our validation grid Figure 11, consisting of unseen generated images like the one from our dataset, we wanted to test how the model performed on unseen images of us. Here we discovered that the model is very light-sensitive and only perform well in good lightning conditions. We tried increasing the range of brightness added in the pre-processing to make the model perform better on images in different lightning. It was still hard for the model to perform well on images from the wild when images was not in the right lightning conditions ore people would wear i mask that had a different color than the blue mask from the dataset. This made us set up some criteria for the input data so that we could realise the limitations for our model as seen in section 5.

4 Results

MundbiNet had a very high accuracy (see section 4) on the AI generated images which were exclusively used when training the model. These predictions were also very consistent.

	Loss	Accuracy	Validation Loss	Validation Accuracy
Epoch 15	0,0186	0,996	0,0237	0,995

Unfortunately the model still had difficulties predicting actual images of real humans as seen in Figure 11. Lighting, angle and mask type were triggering the wrong features. During the final stages of the experimentation several types of regularization techniques were applied, but we saw no improvements on images in the wild. It was expected that a rotation of 20 degrees in pre-processing would help.

Figure 11: Predictions of wild images (left) and AI generated images (right)

4.1 Discussion of the results

In the following, the results produced by *MundbiNet* is discussed and compared to related work.

4.1.1 Visualising using t-SNE

After visualising the data using t-SNE it is clear that that the model can easily classify the different AI generated images from one another, correctly identifying the position of their mask. The entire test-set of 7040 images were used for the visualisation in Figure 12. Visualising a large dataset of actual images from the wild would likely reveal a visualisation with less clear boundaries between the classes.

4.1.2 Heatmaps of classes

Inspecting the superimposed heatmaps Figure 13, there is a clear difference in the weights depending on the mask position. There is a great focus on the hair when a person is not wearing a mask, which is odd because most of the hair is unobstructed by the mask. On a correctly placed mask, the model will focus on the edges at the bottom of the mask. This makes sense because wearing the mask below the nose or on the chin will block this part of the face, which makes these edges unique to a correct placement. The last image only focus on a very few edges on the image, which is not ideal, and indicates that the model is still overfit, even after much pre-processing.

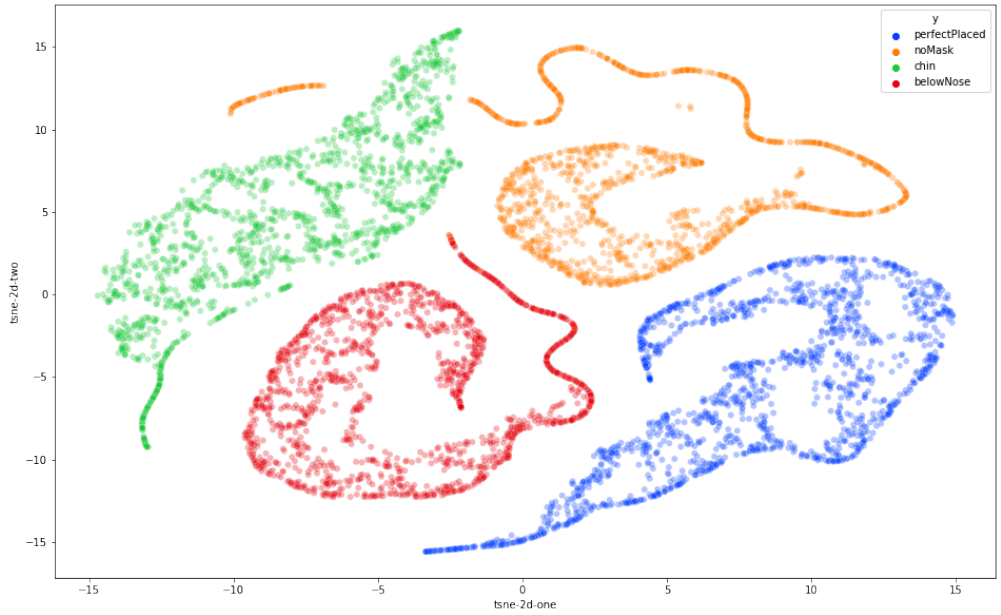

Figure 12: t-SNE visualisation of the final model

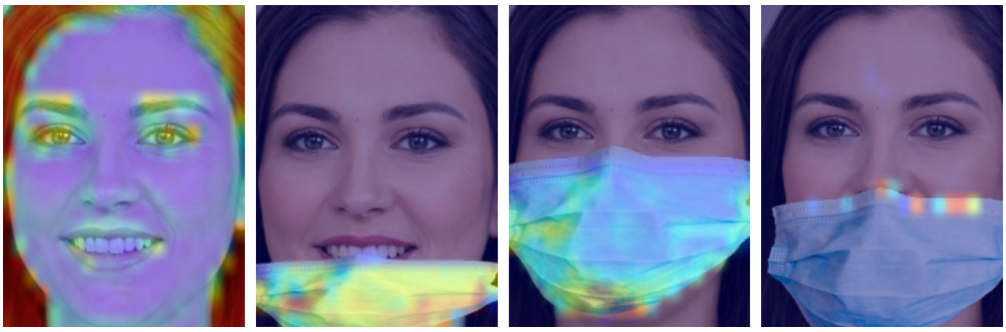

Figure 13: Superimposed heatmaps for the four different classifications

4.1.3 Model Accuracy

The final model *MundbiNet* is very accurate, but only under the right conditions. These conditions are relatively strict, which makes the model unfeasible to implement in the real world as it currently is. Training the model has been a constant battle with increasing its accuracy while attempting to reduce overfitting. This has been difficult, mainly due to the fact that the dataset is very generic and all images are very similar. The masks are also identical, and placed in almost the same position. These factors makes it easy for the model to 'cheat' instead of correctly analysing the images. Many of the regularisation techniques that were applied did not have such a big effect as intended.

4.1.4 Comparison to related work

The primary paper to compare *MundbiNet* with is M. Loey et al's paper[2]. They present their work on how a hybrid model that combines deep learning and traditional machine learning can predict whether a person has a mask on or not. The first comparison will focus on the pandemic impact followed by a comparison of the different approaches in terms of architecture and results. Loey et al's paper has a naive implementation if we look at the different ways of wearing a mask. People are not as binary as they present it and this is also the reason by the multi-class classifier, *MundbiNet*, which takes the different ways of wearing a mask into consideration. Ideally, the model should be extended with more classes and also a more images of real humans wearing the mask with correct labelling to make it work better on real images.

Loey et al had high accuracy, when they evaluated their model on 3 different datasets.

> "The SVM classifier in RMFD achieved 99.64% testing accuracy. In SMFD, it gained 99.49%, while in LFW, it reached 100% testing accuracy" - [2] Section: Conclusion and future works.

The different testing accuracy is very close to the accuracy of *MundbiNet*, but the level of deepness is far from the same. *MundbiNet* has only one convolutional layer where as Loey et al's model has 18 layers. Their model is a hybrid deep transfer learning model with machine learning methods and is build with the architecture of ResNet. *MundbiNet* is a pure CNN.

5 Limitations and future work

The primary limitations of this project have been that the model is not doing well at predicting in the wild. This limitation primarily comes down to a limited dataset, which we have produced on our own based on an AI generated set of photos. It is unknown how the model CNN would react with a strong dataset from the real world with labels. Future work will be to gather a large labelled dataset and train the classifier with this data. The current dataset is generated using the same mask image for augmenting all the training data. In future work the generating of images for training should include different types of masks in different settings to prevent overfitting. To make the model perform better on images in the wild, a more aggressive data augmentation is suggested for future work. An augmentation to explore could be *color jitter* to augment random change in brightness, contrast and saturation of training images.

Several regularization techniques have not been explored yet. Changing the hyperparameters regarding weight decay have not been implemented nor tested, which is left for future work. All in all, there are many different approaches that can be used to optimize the network. Augmentation to prevent occlusion could also have been implemented, but this is also left for future work.

It comes down to what the purpose of the model is and in our case it was to distinguish between different ways of wearing a face mask.

6 Conclusion

In this report the development, training and validation of *MundbiNet* is presented. In section 1 the research questions were formulated and in this section, we will give an answer to them based on the work of this report.

- *Is it possible to build a multi-class classifier that can classify 4 different variations of mask placement?* Based on the results in section 4 it can be concluded that the classifier is successful at predicting the four different classes with an accuracy of 99,5%.
- *How accurate is the classifier on images it has never seen?* The small test set in section 4 showed that all of the 16 images from the grid were correctly classified. This was not the case with the earlier model which were tested in Figure 10.
- *How well is the classifier at predicting on real images, when it has been trained on generic AI generated images?* As seen in Figure 1 *MundbiNet* can be used on a live feed from a camera and classify the image stream from the wild. It is not perfect, which is also what could be assumed with the dataset. Figure 11 shows that improvements still have to be implemented in order for the model to be good at predicting in the wild. We still think it is a very good baseline that it is able to predict the correct category for Mette Frederiksen as seen in Figure 11.

References

[1] Yann LeCun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[2] Mohamed Loey et al. "A hybrid deep transfer learning model with machine learning methods for face mask detection in the era of the COVID-19 pandemic". In: *Measurement* 167 (2020), p. 108288.

[3] Luis Perez and Jason Wang. *The Effectiveness of Data Augmentation in Image Classification using Deep Learning*. 2017. arXiv: 1712.04621 [cs.CV].

[4] W. Samek et al. "Evaluating the Visualization of What a Deep Neural Network Has Learned". In: *IEEE Transactions on Neural Networks and Learning Systems* 28.11 (2017), pp. 2660–2673. DOI: 10.1109/TNNLS.2016.2599820.