Comparing Convolutional Neural Network and Logistic Regression on the Fashion MNIST and Facemask Detection Dataset

Knut Magnus Aasrud
Philip Karim Niane
Ida Due-Sørensen

December 16, 2020

Abstract

We classify the Fashion-MNIST dataset with a logistic regression and a convolutional neural network; this as a learning exercise before using the same methods to classify the AI face mask detection dataset. Their performance is compared and the accuracy of each is discussed.

1 Introduction

Spectacular advancements in deep learning have been constructed and perfected with time, primarily over the convolutional neural network (CNN) algorithm. The use of CNNs has grown due to its success with a vast array of applications including image classification [1], speech recognition [2] and self-driving cars [3]. The aforementioned list of applications is far from complete, but it demonstrates the versatility and the importance of the CNN algorithm.

The purpose of this work is to explore computational methods of classifying whether a picture contains a person wearing a facemask, incorrectly wearing a facemask or not wearing one at all. In the current climate, this could prove useful in a plethora of ways. An example could be cameras situated at the entrances of a mall, counting the number of people wearing masks and thus calculating the risk of housing a said number of people at the same time.

We will compare the performance of an ordinary logistic regression with a convolutional neural network on the Fashion-MNIST and AI face mask detection dataset. The Fashion-MNIST is a good benchmark dataset which poses a more challenging classification task than the simple MNIST digits data. It will here be used as a proof-of-concept modelling task, and as a learning exercise before we tackle the facemask dataset.

2 Theory

2.1 Logistic regression

Logistic regression is a method for classifying a set of input variables $\boldsymbol{x}$ to an output or class $y_i, i = 1, 2, \ldots, K$ where K is the number of classes. The review in this section is based on Hastie et al. [4, ch. 4], and the reader is referred to this book for a more detailed explanation of topic. The prediction of output classes which the input variables belongs to is based on the design matrix $\boldsymbol{X} \in \mathbb{R}^{n \times p}$ that contains n samples that each carry p features. We distinct between *hard* and *soft classification*, which determines the input variable to a class deterministically or the probability that a given variable belongs in a certain class. The logistic regression model is given on the form

$$\begin{aligned} \log \frac{p(G = 1|X = x)}{p(C = K|X = x)} &= \beta_{10} + \beta_1^T x \\ \log \frac{p(G = 2|X = x)}{p(C = K|X = x)} &= \beta_{20} + \beta_2^T x \\ &\vdots \\ \log \frac{p(G = K - 1|X = x)}{p(C = K|X = x)} &= \beta_{(K-1)0} + \beta_{K-1}^T x. \end{aligned} \tag{1}$$

We consider the binary, two-class case with $y_i \in [0, 1]$. The probability that a given input variable x_i belongs in class y_i is given by the Sigmoid-function (also called logistic function):

$$p(x) = \frac{e^x}{1 + e^x} = \frac{1}{1 + e^{-x}}. \tag{2}$$

A set of predictors, $\boldsymbol{\beta}$, which we want to estimate with our data then gives the probabilities

$$p(y_i = 1|x_i, \boldsymbol{\beta}) = \frac{\exp\left(\boldsymbol{\beta}^T x_i\right)}{1 + \exp\left(\boldsymbol{\beta}^T x_i\right)} \tag{3}$$

$$p(y_i = 0|x_i, \boldsymbol{\beta}) = 1 - p(y_i = 1|x_i, \boldsymbol{\beta}). \tag{4}$$

We define the set of all possible outputs in our data set $\mathcal{D}(x_i, y_i)$. Further, we assume that all samples $\mathcal{D}(x_i, y_i)$ are independent and identically distributed. Now we can approximate the total likelihood for all possible outputs of $\mathcal{D}$ by the product of the individual probabilities [4, p. 120] of a specific output y_i:

$$P(\mathcal{D}|\boldsymbol{\beta}) = \prod_{i=1}^{n}[p(y_i = 1|x_i\boldsymbol{\beta})]^{y_i}[1 - p(y_i = 1|x_i, \boldsymbol{\beta})]^{1-y_i}. \tag{5}$$

We want to maximize this probability by using the maximum likelihood estimator (MLE). By taking the logarithm of eq. (5), we obtain the log-likelihood in $\boldsymbol{\beta}$

$$\log P(\mathcal{D}|\boldsymbol{\beta}) = \sum_{i=1}^{n}[y_i \log p(y_i = 1|x_i\boldsymbol{\beta}) + (1 - y_i)\log(1 - p(y_i = 1|x_i, \boldsymbol{\beta}))]. \tag{6}$$

By reordering the logarithms and taking the negative of eq. (6), we obtain the *cross entropy*

$$\mathcal{C}(\boldsymbol{\beta}) = -\sum_{i=1}^{n}\left[y_i\boldsymbol{\beta}^T x_i - \log\left(1 + \exp\left(\boldsymbol{\beta}^T x_i\right)\right)\right]. \tag{7}$$

The cross entropy is used as our cost function for logistic regression. We minimize the cross entropy, which is the same as maximizing the log-likelihood, and obtain

$$\frac{\partial\mathcal{C}(\boldsymbol{\beta})}{\partial\boldsymbol{\beta}} = -\sum_{i=1}^{n} x_i(y_i - p(y_i = 1|x_i, \boldsymbol{\beta})) = 0. \tag{8}$$

The second derivative of this quantity is

$$\frac{\partial^2\mathcal{C}(\boldsymbol{\beta})}{\partial\boldsymbol{\beta}\partial\boldsymbol{\beta}^T} = \sum_{i=1}^{n} x_i x_i^T p(y_i = 1|x_i, \boldsymbol{\beta})(1 - p(y_i = 1|x_i, \boldsymbol{\beta})). \tag{9}$$

These expressions can be written more compactly by defining the diagonal matrix $\boldsymbol{W}$ with elements $p(y_i = 1|x_i, \boldsymbol{\beta})(1 - p(y_i = 1|x_i, \boldsymbol{\beta}))$, $\boldsymbol{y}$ as the vector with our y_is values and $\boldsymbol{p}$ as the vector of fitted probabilities. We can then express the first and second derivatives in matrix form

$$\frac{\partial\mathcal{C}(\boldsymbol{\beta})}{\partial\boldsymbol{\beta}} = -\boldsymbol{X}^T(\boldsymbol{y} - \boldsymbol{p}) \tag{10}$$

$$\frac{\partial^2\mathcal{C}(\boldsymbol{\beta})}{\partial\boldsymbol{\beta}\partial\boldsymbol{\beta}^T} = \boldsymbol{X}^T\boldsymbol{W}\boldsymbol{X}, \tag{11}$$

also known as the Jacobian and Hessian matrices, respectively. We will use the stochastic gradient descent (SGD) (section 2.2) to find the optimal parameter $\boldsymbol{\beta}$.

2.2 Stochastic Gradient Descent

Gradient descent describes the process of finding a local minimum of a function (the cost function, in our case) by following the negative value of the gradient at each point, stepwise. *Stochastic gradient descent* or SDG is a way of increasing the numerical efficiency of this process, by doing this process stochastically.

This involves randomly dividing the training data into a given number of *mini batches*. For each mini batch, the gradient is found by averaging the gradient value each mini batch sample has. Then the weights and biases are updated (take a step down the "slope") and the process is repeated for the rest of the mini batches. The updating done at each mini batch is expressed mathematically as

$$w \to w' = w - \frac{\eta}{m} \sum_{i}^{m} \nabla C_{i,w}$$
$$b \to b' = b - \frac{\eta}{m} \sum_{i}^{m} \nabla C_{i,b},$$

where m is the number of data points in the mini batches and ∇C_i is the gradient of the cost function at each individual data point. After exhausting all the training data, we have finished a so-called *epoch*, of which we can perform as many as necessary.

2.3 Convolutional neural network (CNN)

In deep learning, a convolutional neural network (CNN), is a specialized kind of neural network most commonly applied to analyzing images. Convolutional networks can be thought of as neural networks that use convolution in place of a general matrix multiplication in at least one of their layers [5].

In this section we assume that you are already familiar with the multilayer perception (MPL). If you wish to refresh your memory you can look into our previous work on MPL [6].

2.3.1 Motivation

For each layer in the MPL we have to perform a matrix multiplication of the activation's, a^l with the new weights, w^l. As an example we consider an image size of $28 \times 28 \times 3$ (28 pixels wide, 28 pixels high, 3 color channels). A single fully-connected neuron in the first hidden layer of the MPL would then have $28 \times 28 \times 3 = 2352$ weights. This amount of operations is manageable and might perform well for basic binary images, but does not scale to a high pixelated image of size $M \times N \times 3$.

Further, MPLs ignore the information brought by pixel position and correlation with neighbour pixels. Another drawback is that MPLs are not translation invariant. If an element of interest appear in the top left of the image in one picture and the bottom right of another picture, the MPL will try to correct itself and assume that the element will always appear in this section of the image.

The CNN successfully captures the spatial and temporal dependencies through the application of relevant filters. A better fit to the image dataset is obtained by the reduction in the number of parameters involved and the reusability of weights. Another advantage of convolution is that it provides a mean for working with inputs of variable size. In the following we describe a simple CNN architecture for image classification.

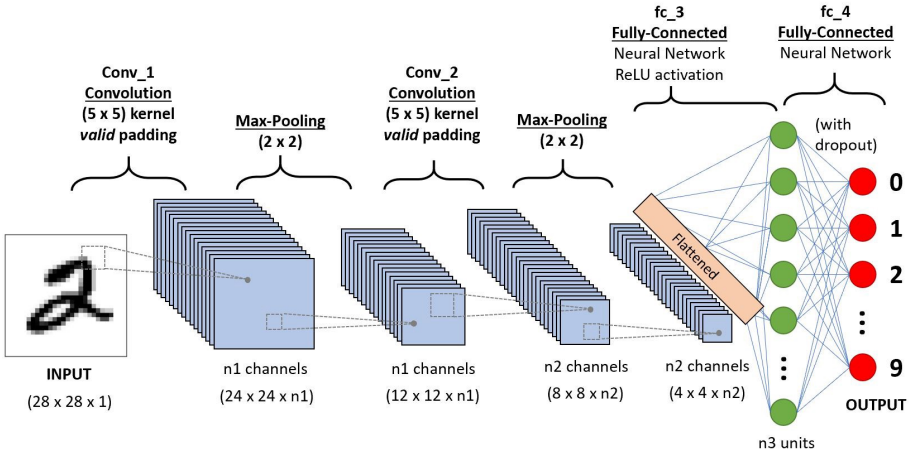

Figure 1: An example of a CNN architecture to classify handwritten digits. Figure taken from [7].

2.3.2 Input layer

The first layer is the *input layer*. This layer will hold the raw pixel values of the image to be studied. In the case of a RGB color image we have three pixel grids representing red, green and blue. The number of pixel grids is referred to as the image *depth*. A RGB image thus have a depth of three and a grayscale image has a depth of one. Figure 2 show an example of a $4 \times 4 \times 3$ RGB image.

2.3.3 Convolution layer

The main purpose of the *convolution layer* is to relate pixels to each other spacially. Here, the element involved in carrying out the convolution operation

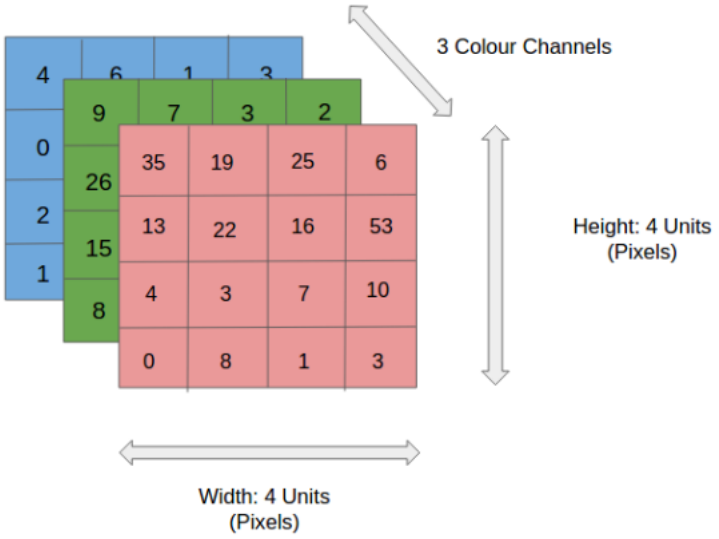

Figure 2: An example of an $4 \times 4 \times 3$ RGB image. Figure taken from [7].

is reffered to as a *filter* or *kernel (K)*. The kernel matrix typically has shape 3×3 or 5×5 and has the depth of the input image.

The filter starts in the upper left corner, the algorithm looks at a section of the input layer (I_n) that has the same shape as K. It then performs a matrix multiplication between I_n and K and the results are summed with the bias to give a squashed one-depth channel convoluted feature output [7]. Then, the filter moves to the right by an amount referred to as the *stride length* till it parses the complete width of the image. Moving on, the filter hops down to the leftmost section of the image with the same stride value. The process is repeated until the entire image is traversed.

The convolution layer reduces the size of the input. For a 5×5 input with stride length 1, such as the example in fig. 3, will result in a 3×3 convoluted feature output. If this is not desirable, we can use *padding*. *Same padding* involves adding a layer or several layers of pixels of value 0 around the entire input. The input is now a 6×6 matrix, and the convoluted feature output will have size 5×5. If the stride length is larger than 1, *valid padding* can be used to make the size of the convoluted feature output less reduced. In a typical CNN architecture, both valid and same padding is used.

2.3.4 ReLU layer

The *ReLU layer* applies the non-saturating activation function $f(x) = \max(0, x)$. This activation function effectively removes negative values from an activation map. this leaves the size of the volume unchanged. Further, it increases the nonlinear properties of the decision function and of the overall network without

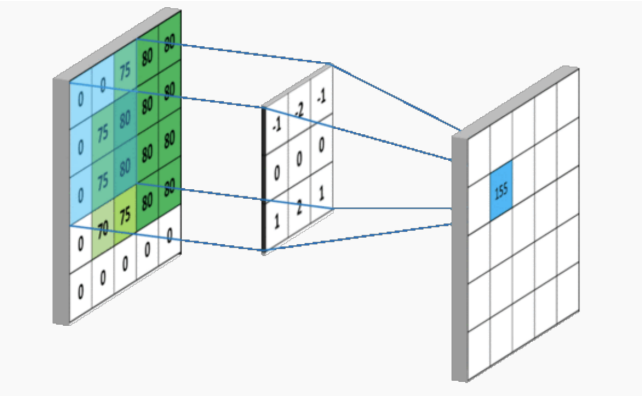

Figure 3: An example of how a convolution layer is applied to a image using kernel filters. Figure taken from [8].

affecting the receptive fields of the convolution layer [9].

2.3.5 Pooling layer

The *pooling layer* is used to non-linearly downsample a activation map. Pooling utilizes the idea that the exact location of a feature is less important than a rough estimate of the relative location of other features. When the spatial dimension is reduced, forcing translation invariance, overfitting and the amount of computations needed is reduced. The pooling map sizes are typically 2×2, larger sizes may also be used. In the following we consider a pooling size of 2×2. Pooling may compute a max or an average from the portion of the image covered by the Kernel, see the example in fig. 4. Max pooling is better at reducing noise, and will therefore typically perform better. For large images it is typical to resend the feature through convolution and pooling layers several times, this comes with a increased computational cost.

2.3.6 Fully-connected layer and output layer

Finally, after several convolutional and pooling layers, we have the *fully-connected* layer (FCL). Here, the input is first flattened before it is sent through a traditional feed-forward neural network layer (FFNN). There are typically several FCLs and the final layer is referred to as the *output layer*. Backpropagation is applied to every iteration of training, and over a series of epochs, the algorithm is able to classify images.

2.4 Quality of models

We measure the performance of the different models used in this work. For classification, we measure how accurate the predictions given by the model are by the so called accuracy score. The accuracy score is given by

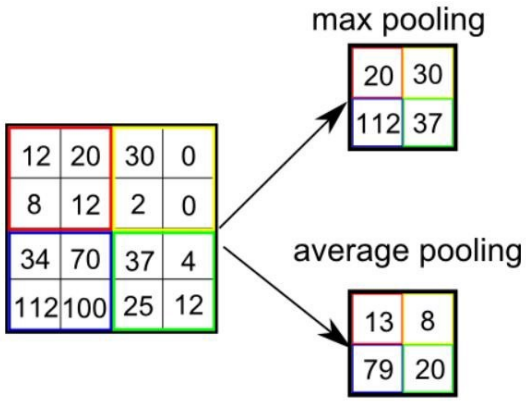

Figure 4: Two types of pooling: computing a max or an average from the portion of the image covered by the Kernel. Figure taken from [7].

$$\text{accuracy} = \frac{1}{n}\sum_{i=0}^{n} I(t_i = y_i), \tag{12}$$

where n is the number of samples, t_i represents the target output and I is the indicator function, which returns 1 if $y_i = t_i$ and 0 otherwise.

The *confusion matrix* of a categorical predictor is a table layout that presents a more in-depth view of the method performance than the accuracy score. Each row of the matrix represents the instances in a predicted class while each column represents the instances in an actual class. The advantage of the confusion matrix is that it better illustrates the misclassified instances. Below, we illustrate the confusion matrix for the three-class A, B and C:

		True		
		A	B	C
Pred	A	AA	AB	AC
	B	BA	BB	BC
	C	CA	CB	CC

All the correct predictors are located in the diagonal of the table, while the non-diagonal elements represent the misclassificatons of the predictors.

3 Method

3.1 Data sets

3.1.1 Fashion-MNIST

The fashion-MNIST is a dataset of Zalando's article images – consisting of a training set of 60,000 examples and a test set of 10,000 examples [10]. A sample of the dataset can be seen i fig. 5. Each article image is a 28×28 grayscale image, associated with a label from 10 classes. The class labels can be seen in table 1. The items are distributed evenly, 6000 of each in the training set and 1000 in the test set. As the fashion-MNIST shares the same image size and structure of training and testing splits, it can serve as a direct drop-in replacement for the original MNIST dataset for benchmarking machine learning algorithms.

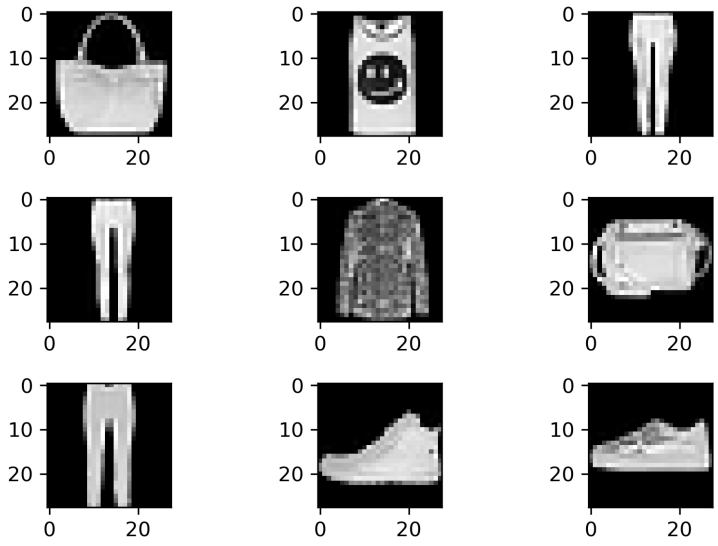

Figure 5: A sample of the fashion-MNIST dataset.

3.1.2 Facemask detection dataset

The facemask dataset is a dataset consisting of approximately 4000 RGB images of sizes 1024×1024. The dataset consists of pictures of people with and without face masks. The images is split into two folders, a train- and a testset. The dataset can be found at kaggle[11]. For easier handling of the images, the images were preprocessed and saved as arrays to make it easier to load when applying the models. The prehandling includes reshaping the images into sizes of 224×224 pixels. The masks on the images were edited on by computer tools. Some of the photos can be seen in figure 6

Table 1: Class names in the fashion-MNIST dataset.

Label	Description
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

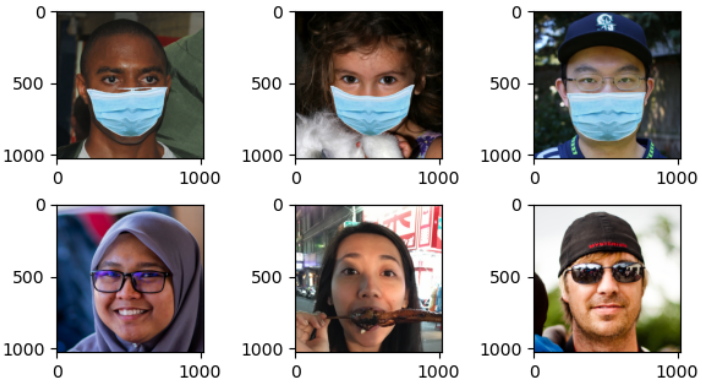

Figure 6: A sample of the facemask detection dataset. The value 1=mask, while the value 0=no mask.

3.2 Logistic regression

The implementation of the logistic regression is implemented pretty straight forward. From project 2 [6], it was shown that scikit learn had the ability to provide predictions using logistic regression much faster, and a bit more accurate than a self written SGD algorithm. The logistic regression in this article therefore revolves around scikit learn's logistic regression library. Beside using scikit learn's machine learning library, scikit learn's image processing library which is as the name says, an image processing library. This well known library is quite useful when dealing with images, and was implemented when processing the images from the facemask dataset.

The facemask implementation is done by resizing the images into sizes of 224x224

to reduce the computer effort. Then the images is scaled by dividing the pixels by the maximum pixel value, and sorted into arrays. Then removing the presence of colours by transforming the images into grayscale. This way there is only one colour channel to work with, instead of the original three. The mnist dataset doesn't need as much prework as the facemask dataset except scaling the data.

After the data is ready, the pixel values are sent into the logistic regression function and trained. After the training is done, the fitting and predicting is ready to be set in motion. The predicted response vector for the datasets is retured, which then can be used to calculate the accuracy.

4 Results

4.1 Fashion-MNIST

4.1.1 CNN

The training session for the CNN on the Fashion-MNIST dataset initially developed promisingly. However, the loss shown in the validation set quickly increased in the last epochs, as shown in figure 7

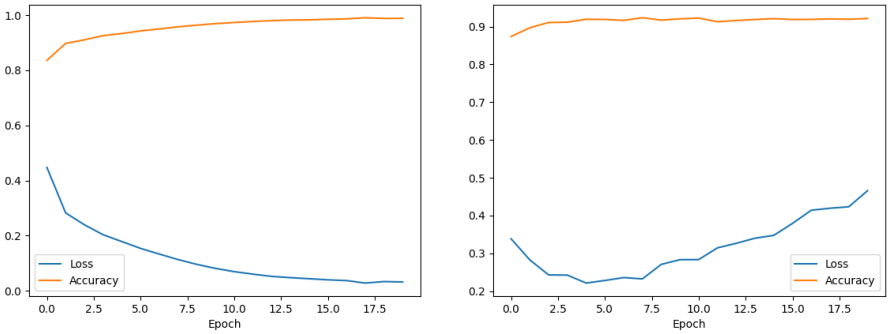

Figure 7: History of loss and accuracy of the training and validation set as they develop through the epochs.

After modifying the network structure by implementing dropout in the model, the validation accuracy and loss progressed nicely and as expected. This is shown in figure 8

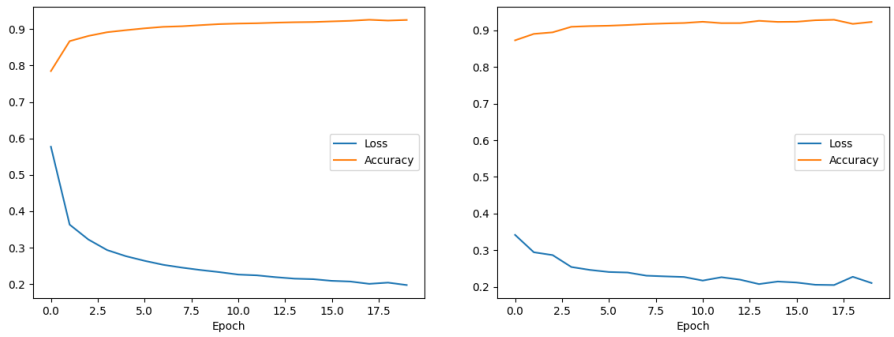

Figure 8: History of loss and accuracy of the training and validation set as they develop through the epochs. Here, dropout is implemented into the model.

After evaluating the model on the test set and comparing against the real values, the accuracy on never before seen data is 92% (this result can be reproduced by running `src/fashion_mnist/predict.py`). In figure 9, we see a small selection of test data, the predicted category and the actual category.

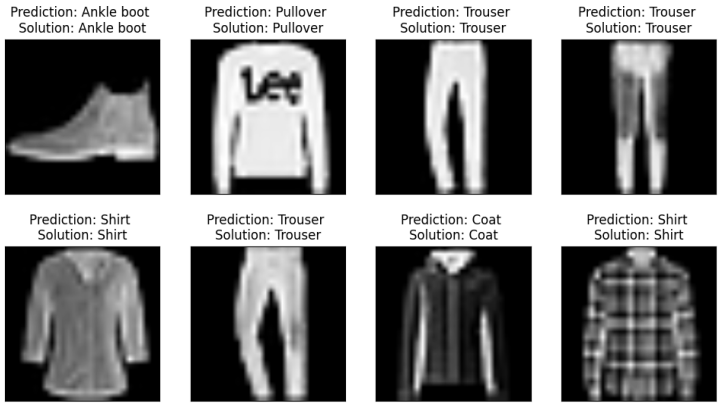

Figure 9: A small selection of the Fashion-MNIST testing set, with the predicted category and the actual category.

4.1.2 Logistic regression

The logistic regression model was applied on the mnist fashion dataset. The logistic regression resulted in an accuracy of 84% when applied to the test set. The accuracy from the train set was 88%. The confusion matrix which shows

the accuracy of each article can be seen in figure 10.

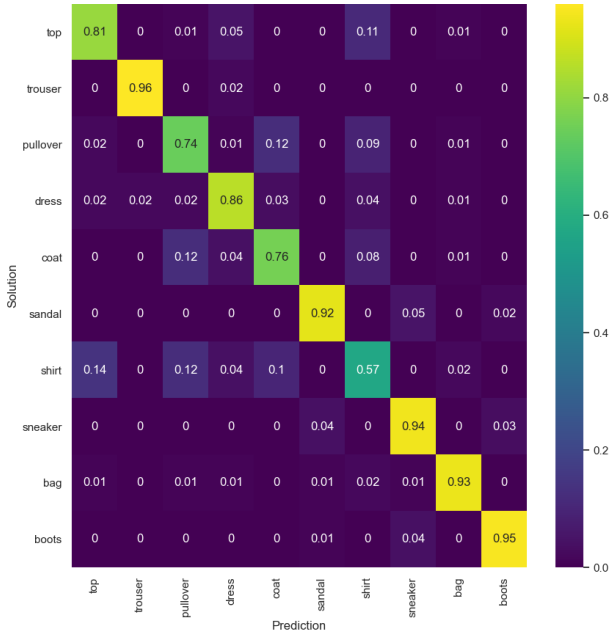

Figure 10: The confusion matrix after using logistic regression to classify the fashion mnist. The matrix shows the accuracy of the model within each predicted class

Some of the missclassified pictures from using logistic regression on the fashion mnist dataset can be seen in figure 11

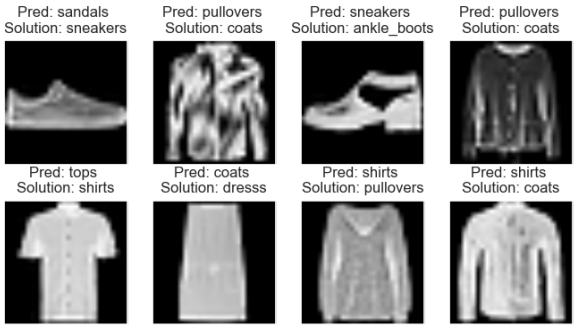

Figure 11: Some missclassified images from the mnist fashion dataset, predicted by logistic regression

4.2 Face mask detection

4.2.1 CNN

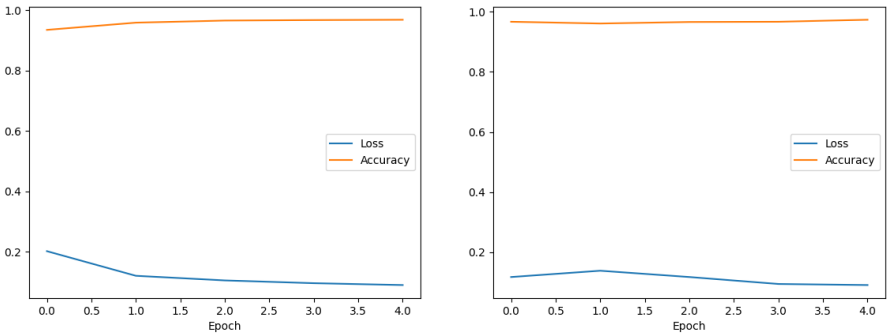

Figure 12: History of loss and accuracy of the training and validation set as they develop through the epochs.

After training on the training dataset, our model got an accuracy of approximately 98% on the testing set. In addition to classifying the testing set, we snapped a couple of pictures of ourselves to test the network on. Figure 13 shows some examples and the results the network provided.

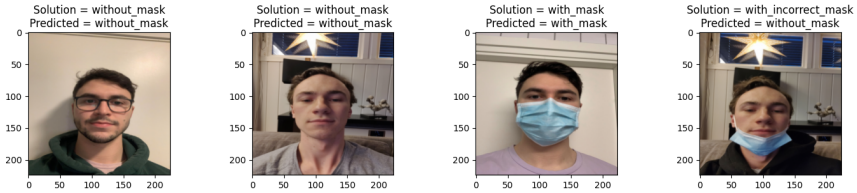

Figure 13: Pictures of ourselves and the predicted classifications of them.

4.2.2 Logistic regression

The logistic regression model was applied on the face mask dataset. The logistic regression resulted with an accuracy score of 48% when applied to the test set. The confusion matrix which shows the accuracy of the predicted classifications can be seen in figure 14. Some of the miss classified images can be seen in figure 15

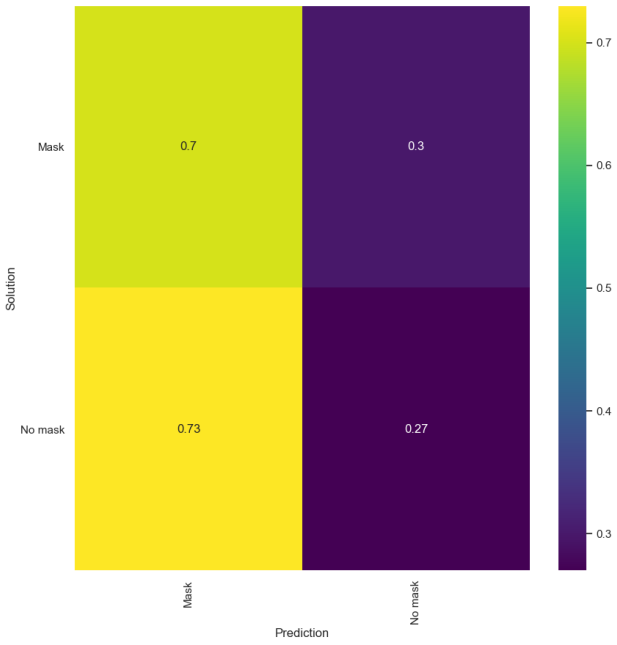

Figure 14: The confusion matrix after using logistic regression to classify the face mask detection dataset. The matrix shows the accuracy of the model within each predicted class

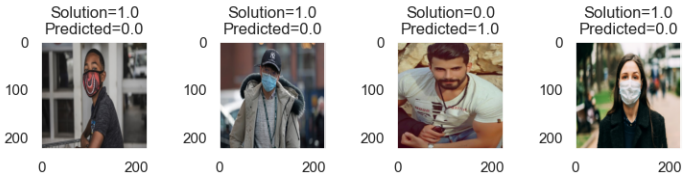

Figure 15: Some miss classified images from the facemask detection dataset, predicted by logistic regression

Since the dataset consisted of images of people with facemasks edited on instead of wearing real physical masks, the model was tested on a few pictures taken by ourself to check how the model performed after training. The results can be seen in figure 16

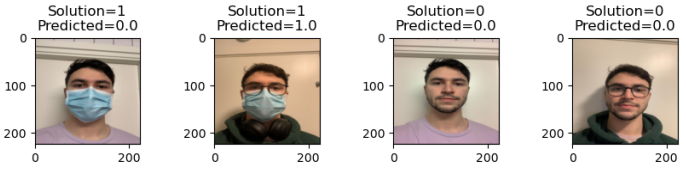

Figure 16: Pictures taken by ourselves to test the model. The model predicted three out of four pictures correct

5 Discussion

5.1 Fashion-MNIST

5.1.1 Logistic regression

When using logistic regression on the fashion mnist, the accuracy ended up with a decent 84% score on the testset. This is quite good considering that logistic regression is a regression method. Having a look at the confusion matrix in figure 10, it is easy to see which articles was the hardest to classify, and which ones the easiest. The correct predictions is, as mentioned in the theory along the diagonal. The easiest articles to recognize was the trousers with an accuracy of 96%, boots with an accuracy of 95%, and sneakers, bags and sandals with accuracy's of 94-, 93- and 92% respectively. The most difficult one to classify was without doubt the shirts with an accuracy of 57%. Followed by pullovers and coats with 74- and 76% accuracy respectively. The reason the model struggled some bit to classify shirts is most certain because shirts resembles tops, pullovers and coats a lot, which is backed up by the confusion matrix which shows that about 35% of the shirts were predicted into those three categories.

5.1.2 CNN

5.2 Facemask detection

5.2.1 Logistic regression

Applying the logistic regression model on the facemask detection dataset, ended up with a disappointing accuracy of 48% on the testset. This is a bit worse than expected due to the fact that making the model aimlessly guess which category to label the images, would give a couple of percentages higher accuracy. The model having some difficulties learning, is probably due to the facemask pictures being a bit too complex with every picture having a new face, in a new location with different objects in the picture. Having a look at the confusion matrix in figure 14, it is easy to notice that the model is predicting most of the images as images containing masks, with 73% of the testset predicted as being an image containing a mask. Because the model is predicting most of the pictures as masks, it managed to predict 70% of the masks correct, while it only predicted about 30% of the images without a mask correctly.

Moving over to figure 16, which shows the predictions of real life pictures taken by us and predicted by the model, shows that the model managed to guess 3/4 of the images which is impressing, but can't really be taken seriously due to the fact that it was only tested on four photos, in addition the four photos were really simplistic being a front photo with a plane background.

5.2.2 CNN

The networks training session progressed promisingly and our accuracy was satisfactory. It is however worth mentioning that many of the pictures in the dataset are stock photos with facemasks edited in, which means the masks are very similar and the photos have similar properties (focus, aspect ratio, and so on). When testing on the photos taken of ourselves, the accuracy quickly declined to around 50% (there were rather few photos, mind you, and a greater dataset is needed to prove any significance). This might be due to the fact that these photos were taken in portrait, while the training set mostly included prefectly square photos. The scaling necessary to do a prediction might have led to artifacts that made it difficult to discern the prescence/absence/incorrectness of a mask.

To improve upon the model, the perhaps most important step is to gather a more diverse dataset, including different angles, aspect ratios and mask types. A higher accuracy could also be gained by implementing a separate model that recognizes faces/heads and crops the picture to match them. This would alleviate the issue of aspect ratio and ensure a predictable "correct" position of the mask, improving the accuracy of guessing incorrectly placed masks.

5.3 Logistic regression vs. CNN

6 Conclusion

After exploring the use of CNN and logistic regression for image classification, it is safe to say that CNN with an outstanding 98% accuracy on the facemask easily outperforms logistic regression which only managed an accuracy of 48% on the same facemask dataset.

(Need more conclusion here- Conclusion is a bit short)

Comparing our results with scientific articles like [12] and [13] that also have been studying the use of machine learning for face mask detection, by using CNNs, put our CNN really close to their performance. They managed to achieve accuracy scores of 99.49% and 99% respectively.

6.1 Prospects for the future

The main prospect for the future would be to dive into the fine tunings of CNNs and explore further how different parameters affect the accuracy, and search even further for ways to push the accuracy closer towards the 100%. The logistic regression was implemented by using scikit learn, which gives less elbow room for optimization.

Another interesting prospect for the future would also be to try these methods on more complicated and larger datasets, to see how they fare in other scenarios.

References

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2016.

[2] O. Abdel-Hamid, L. Deng, and D. Yu, "Exploring convolutional neural network structures and optimization techniques for speech recognition," in *INTERSPEECH*, 2013.

[3] B. Wu, F. Iandola, P. H. Jin, and K. Keutzer, "Squeezedet: Unified, small, low power fully convolutional neural networks for real-time object detection for autonomous driving," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, Jul. 2017.

[4] T. Hastie, R. Tibshirani, and J. Friedman, *Elements of Statistical Learning: Data Mining, Inference, and Prediction*, eng, ser. Springer series in statistics. New York: Springer, 2009, ISBN: 0387848576.

[5] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, `http://www.deeplearningbook.org`.

[6] K. M. Aasrud, P. K. Niane, and I. Due-Sørensen, *Logistic regression and feed forward neural network (ffnn) analysis on the mnist and franke's function dataset*, 2020. [Online]. Available: https://github.com/kmaasrud/ffnn-fys-stk4155/blob/main/doc/main.pdf.

[7] S. Saha. (Dec. 15, 2018). "A comprehensive guide to convolutional neural networks — the eli5 way," [Online]. Available: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

[8] M. Stewart. (Feb. 27, 2019). "Simple introduction to convolutional neural networks," [Online]. Available: https://towardsdatascience.com/simple-introduction-to-convolutional-neural-networks-cdf8d3077bac.

[9] Wikipedia contributors, *Convolutional neural network — Wikipedia, the free encyclopedia*, https://en.wikipedia.org/w/index.php?title=Convolutional_neural_network&oldid=992073762, [Online; accessed 9-December-2020], 2020.

[10] H. Xiao, K. Rasul, and R. Vollgraf. (Aug. 28, 2017). "Fashion-mnist: A novel image dataset for benchmarking machine learning algorithms." arXiv: cs.LG/1708.07747 [cs.LG].

[11] W. Intelligence, *Face mask detection dataset*, https://www.kaggle.com/wobotintelligence/face-mask-detection-dataset, [Online; accessed 11-December-2020], 2020.

[12] A. Chavda, J. Dsouza, S. Badgujar, and A. Damani, *Multi-stage cnn architecture for face mask detection*, https://www.researchgate.net/publication/344276976_Multi-Stage_CNN_Architecture_for_Face_Mask_Detection, [Online; accessed 15-December-2020], Sep. 2020.

[13] A. Rosebrock, *Covid-19: Face mask detector with opencv, keras/tensorflow, and deep learning*, https://www.pyimagesearch.com/2020/05/04/covid-19-face-mask-detector-with-opencv-keras-tensorflow-and-deep-learning/, [Online; accessed 15-December-2020], May 2020.

A Appendix

A.1 Source code

All the source code is located in this GitHub repository.