Deep Learning Assignment 1: Fashion MNIST

Doudos Panagiotis (f3352106)

Legkas Sotirios (f3352112)

Introduction:

This Project is a classification of different fashion items from the fashion MNIST dataframe using two different architectures, an MLP and a CNN. At first the problem will be tackled with the most basic deep architecture, one of a Multi-Layered-Perceptron. Then, a Convolutional Neural Network will be used, an architecture tailored to image and pattern classification and recognition. The two will be compared, and some troubleshooting will be performed to try and further improve the efficiency of the better model.

Data Preprocessing, MLP Initiation and Tuning:

Even though the data was already ready to use, a couple of preprocessing steps were performed. The image vectors were reshaped to a 28x28x1 matrix, then divided by 255 to confine their numerical values to the (0,1) space. Finally, our labels were converted to 1-hot vectors, so as to align the data with a softmax activation function in the final layer of the models.

To tune a model with Keras tuner, a model building function must first be instantiated, which will take into account the different hyperparameters to be tuned. In tuning of the MLP model, several parameters were selected. The number of layers, the number of nodes per layer, the existence and size of dropout per layer and the learning rate of our model. The activation function was selected to be gelu, as a safer relu alternative, as it does not “kill” possibly negative derivatives, giving us a better, more stable search space. Some parameters were not tuned, such as the input and output layer, which are only dependent on the data. As mentioned, the output layer was a simple layer with a softmax activation function that classified the item to one of ten categories. A BayesianOptimization function was used from the Keras tuner options, to quicken the tuning time and save computational resources while still not missing out on too much compared to a more extensive search like Hyperband. That is mostly because of the Gaussian process used in selecting the hyperparameter combinations, which generally tends to sample sets of parameters that come close to the optimal one. The number of trials was arbitrarily set to 6, as an inexpensive yet effective amount. The validation split was set to 0.2 and early stopping was also used, based on validation loss, with a patience of 10. The complete search space is shown in Figure 1 and the architecture decided was as described in Figures 2 and 3.

Figure 1

MLP Search Space

```
Search space summary
Default search space size: 4
num_layers (Int)
{'default': None, 'conditions': [], 'min_value': 2, 'max_value': 5, 'step': 1, 'sampling': None}
units_1 (Int)
{'default': None, 'conditions': [], 'min_value': 32, 'max_value': 512, 'step': 32, 'sampling': None}
dropout_1 (Float)
{'default': 0.0, 'conditions': [], 'min_value': 0.0, 'max_value': 0.3, 'step': 0.1, 'sampling': None}
learning_rate (Choice)
{'default': 0.001, 'conditions': [], 'values': [0.001, 0.0001, 1e-05], 'ordered': True}
```

Figure 2

Best Hyperparameters in MLP

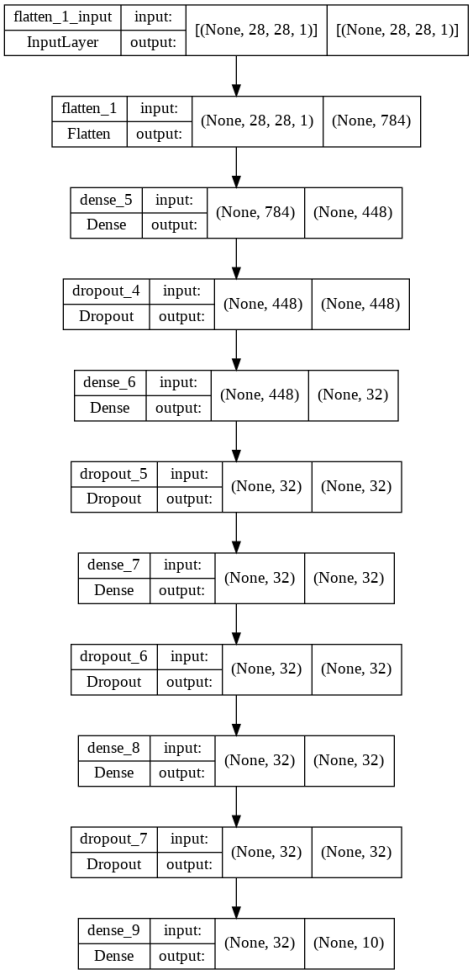
```
The number of layers is 4
The number of nodes in layer 1 is 448
The number of nodes in layer 2 is 32
The number of nodes in layer 3 is 32
The number of nodes in layer 4 is 32
------------------------------------------------
learning_rate is 0.0001
The dropout probability after the 1st layer is 0.3
The dropout probability after the 2nd layer is 0.0
The dropout probability after the 3rd layer is 0.0
The dropout probability after the 4th layer is 0.0
```

MLP fit and results:

After the model was tuned, it was fitted to our training data, with the same validation split as above. The model was allowed to run a maximum of 100 epochs with the same early stopping setup as well. During the model fit, the least validation loss was achieved at 0.2851 (validation accuracy at 90%) with the corresponding training loss being 0.2018 (accuracy at 92.49%). Overall, an accuracy of 89.50 % was achieved in the test set. In the test set, all other average metrics were also at about 90% as shown in the classification report of Figure 4. The results were relatively satisfactory, but the confusion matrix showed that categories 2 (Pullover) and 6 (Shirt) were misclassified the most. As such, a data augmentation on these two categories was decided.

Figure 3

MLP Architecture



Data Augmentation:

As the data was mostly images, augmentation could be easily achieved with rotations, zoom changes, flips etc. A function combining these techniques randomly was created to process the existing data and procure an equal number of new ones. After more data was created that was either an augmentation of a Pullover or a Shirt, the model was fit again (not incrementally, the weights were reset, but the architecture was retained). This time, the test accuracy was increased by 0.72%, (at 90.22%), a small but not negligible change. Some more details on the metrics and how much the two relevant category scores were increased are shown in the Figures below.

Figure 4

Classification Report

	precision	recall	f1-score	support
0	0.85	0.83	0.84	1000
1	0.99	0.98	0.98	1000
2	0.78	0.85	0.82	1000
3	0.91	0.89	0.90	1000
4	0.83	0.82	0.82	1000
5	0.98	0.96	0.97	1000
6	0.73	0.72	0.73	1000
7	0.93	0.97	0.95	1000
8	0.97	0.97	0.97	1000
9	0.97	0.95	0.96	1000
accuracy			0.90	10000
macro avg	0.90	0.89	0.90	10000
weighted avg	0.90	0.90	0.90	10000

Figure 5

Classification Report (Augmented)

	precision	recall	f1-score	support
0	0.83	0.86	0.85	1000
1	0.99	0.98	0.98	1000
2	0.82	0.85	0.83	1000
3	0.92	0.90	0.91	1000
4	0.83	0.85	0.84	1000
5	0.98	0.97	0.97	1000
6	0.77	0.71	0.74	1000
7	0.95	0.96	0.96	1000
8	0.97	0.98	0.98	1000
9	0.96	0.96	0.96	1000
accuracy			0.90	10000
macro avg	0.90	0.90	0.90	10000
weighted avg	0.90	0.90	0.90	10000

Figure 6

MLP Confusion Matrix

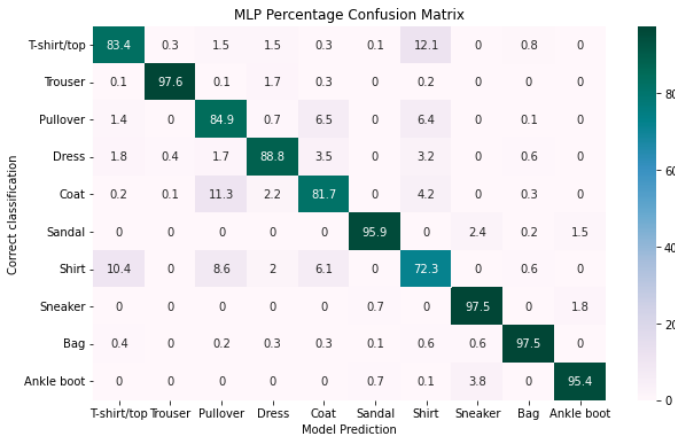


Figure 7

MLP Confusion Matrix (Augmented Data)

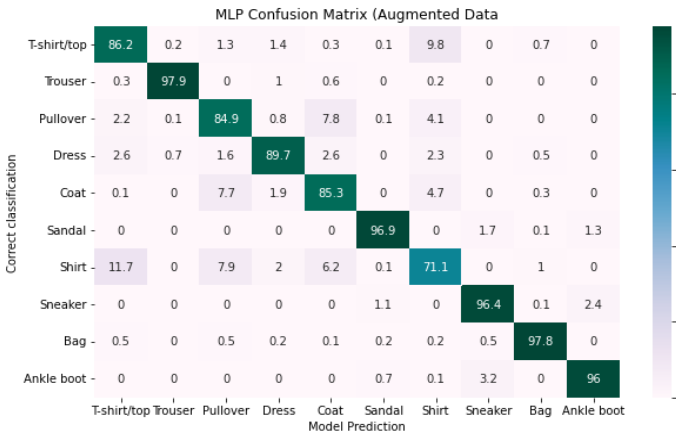


Figure 8

Training and validation loss

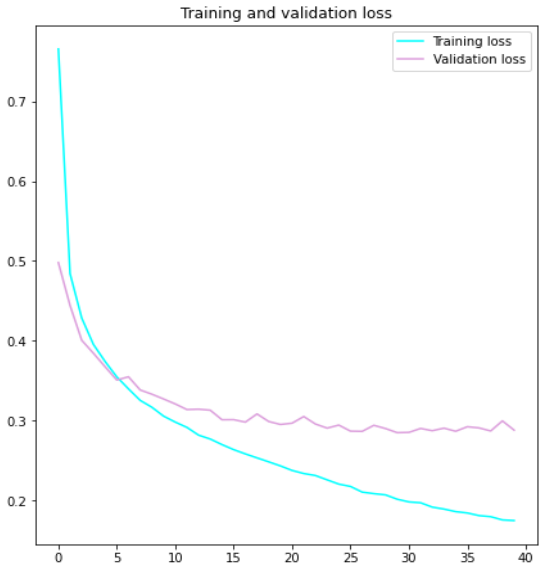
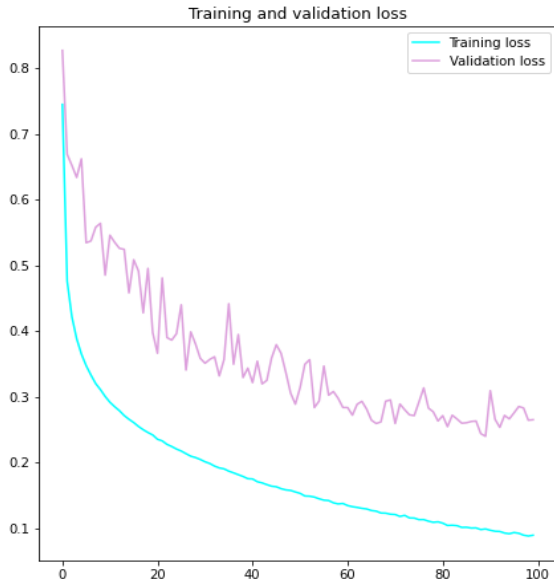


Figure 9

Training and validation loss (Augmented)



Convolutional Neural Network:

After the MLP, it was mandated to use a CNN, which is also natural, as CNNs are well known for their ability to classify images and more specifically image patterns. For our CNN, a pretty standard model format was selected. The basic architecture of our CNN was hard set, with a total of 3 convolutional layer with a max pooling layer for each one respectively, and dropout in between some of them. Some more layers were also added, such as a layer to flatten the matrices in our network and a dense layer on top so that the final classification could be made. A more extensive description of the search space can be seen in Figure 10. The number filters, the size of the kernel, the dropout rates and the number of nodes in our dense layer were tuned using Bayesian Optimization as before. The learning rate was tuned as well. It could be worth noting, that the number of filters in each layer was set to be tuned in progressively higher numbers (e.g. For the first convolutional layer the tuner had to choose between 16 and 32 filters, whereas for the second convolutional layer, between 32 and 64), since that is seen as a generally good practice to let the model narrow down and take notice of smaller details. As before, few trials were executed, this time due to even more severe computational limitations. With each trial running at over an hour, and the only occasional sympathy of Google Collab, 5 trials were considered a safe amount for a model which is not extremely complex such as this one. The tuned final model is shown in Figure 12, which was then fitted to the data. The model was once again allowed to run a maximum of 100 epochs with the same early stopping setup. During the model fit, the least validation loss was achieved at 0.1908 (validation accuracy at 93.17%) with the corresponding training loss being 0.1347 (accuracy at 94.89%). Overall, an accuracy of 92.63%

was achieved in the test set. All other average test set metrics were also at about 93% (rounded up) as shown in the classification report of Figure 13. There was an obvious improvement in the models' decision making, however the confusion matrix made it clear that still, categories 2 (Pullover) and 6 (Shirt) were misclassified quite a bit more than the others. As such, the model was once again fit to the augmented data created for the MLP initially.

Figure 10

CNN Search Space

```
Search space summary
Default search space size: 11
conv_1_filter (Choice)
{'default': 16, 'conditions': [], 'values': [16, 32], 'ordered': True}
conv_1_kernel (Choice)
{'default': 3, 'conditions': [], 'values': [3, 4], 'ordered': True}
conv_2_filter (Choice)
{'default': 32, 'conditions': [], 'values': [32, 64], 'ordered': True}
conv_2_kernel (Choice)
{'default': 3, 'conditions': [], 'values': [3, 4], 'ordered': True}
dropout_115 (Float)
{'default': 0.0, 'conditions': [], 'min_value': 0.0, 'max_value': 0.4, 'step': 0.1, 'sampling': None}
conv_3_filter (Choice)
{'default': 64, 'conditions': [], 'values': [64, 128], 'ordered': True}
conv_3_kernel (Choice)
{'default': 3, 'conditions': [], 'values': [3, 4], 'ordered': True}
dropout_215 (Float)
{'default': 0.0, 'conditions': [], 'min_value': 0.0, 'max_value': 0.4, 'step': 0.1, 'sampling': None}
units_15 (Int)
{'default': None, 'conditions': [], 'min_value': 128, 'max_value': 512, 'step': 32, 'sampling': None}
dropout_315 (Float)
{'default': 0.0, 'conditions': [], 'min_value': 0.0, 'max_value': 0.4, 'step': 0.1, 'sampling': None}
learning_rate (Choice)
{'default': 0.001, 'conditions': [], 'values': [0.001, 0.0001], 'ordered': True}
```

CNN and Data Augmentation:

Once fitted to the augmented data, we saw once again an increase of about 0.5%, reaching a test accuracy of 93.16%. The confusion matrices will be once again presented to show the differences in the models' predictions both in general and specifically for the two augmented categories. It is worth noting that there was no big difference in the percentage raise for the MLP and CNN when given the augmented data, regardless of how vastly different the two models work. This could perhaps speak for the way the data was augmented, and how that does not let any of the models utilize its maximum potential. As such, one of the changes that could be made given more time is changing the data augmentation function to explore how the models react.

Figure 11

Best Hyperparameters in CNN

```
The number of filters in 1st Convolutional layer is 1
Kernel size in 1st Convolutional layer is (3,3)
The number of filters in 2nd Convolutional layer is 32
Kernel size in 2nd Convolutional layer is (3,3)
The dropout probability after the 2nd layer Convolutional is 0.2)
The number of filters in 3rd Convolutional layer is 128
Kernel size in 3rd Convolutional layer is (3,3)
The dropout probability after the 3rd layer Convolutional is 0.4)
The number of nodes in dense layer 2 is 128
The dropout probability after the dense layer is 0.2
------------------------------------------------
learning_rate is 1e-04
```

Conclusion and Observations:

Through working with MLPs and CNNs for this Project, a few things became apparent. First of all, as expected, the CNN was much better equipped through its inherent structure to tackle the problem, giving an almost 4% better accuracy in its estimations. It was also quite clear how data augmentation can be an important asset when working with image data that can be easily and safely augmented. A very interesting note however, is how much more volatile the validation curves were when working with the augmented datasets compared to before (Figures 17 & 18). That should possibly be attributed to the fact that because the training data is now imbalanced, as classes 2 and 6 contain more datapoints, when the model is correctly classifying those two classes, it is rewarded more compared to the others. This causes a bit of an overfit on those two classes, which is then punished, as they are not so many more in quantity compared to the others, causing the slight drop in accuracy once again. Overall, in this Project the different natures and behaviors of the models became very apparent in many ways, not only in their differences, but their similarities as well.

Figure 12

CNN Architecture

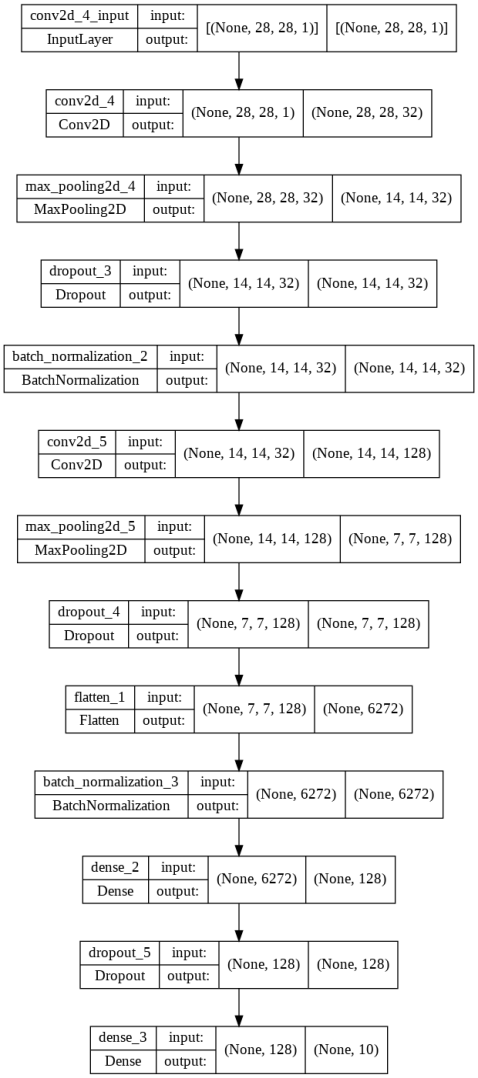


Figure 13

Classification Report

	precision	recall	f1-score	support
0	0.89	0.85	0.87	1000
1	0.99	0.98	0.99	1000
2	0.92	0.85	0.89	1000
3	0.92	0.95	0.93	1000
4	0.87	0.90	0.89	1000
5	0.99	0.98	0.99	1000
6	0.77	0.81	0.79	1000
7	0.95	0.99	0.97	1000
8	0.99	0.99	0.99	1000
9	0.99	0.95	0.97	1000
accuracy			0.93	10000
macro avg	0.93	0.93	0.93	10000
weighted avg	0.93	0.93	0.93	10000

Figure 14

Classification Report (Augmented)

	precision	recall	f1-score	support
0	0.89	0.86	0.87	1000
1	0.99	0.99	0.99	1000
2	0.92	0.87	0.89	1000
3	0.91	0.95	0.93	1000
4	0.88	0.92	0.90	1000
5	0.99	0.99	0.99	1000
6	0.80	0.81	0.80	1000
7	0.97	0.98	0.97	1000
8	0.99	0.98	0.99	1000
9	0.98	0.97	0.97	1000
accuracy			0.93	10000
macro avg	0.93	0.93	0.93	10000
weighted avg	0.93	0.93	0.93	10000

Figure 15

CNN Confusion Matrix

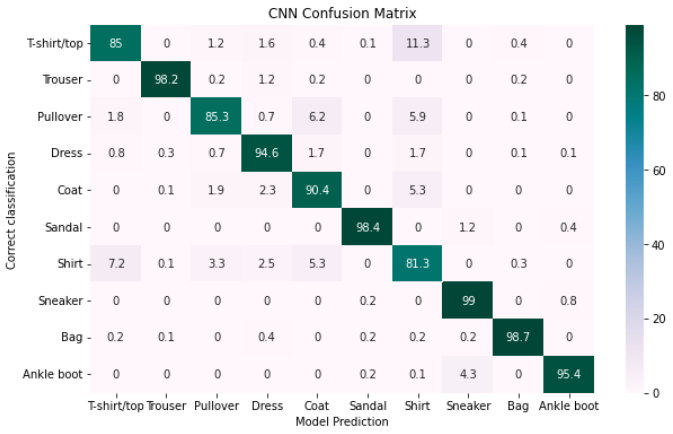


Figure 16

CNN Confusion Matrix (Augmented)

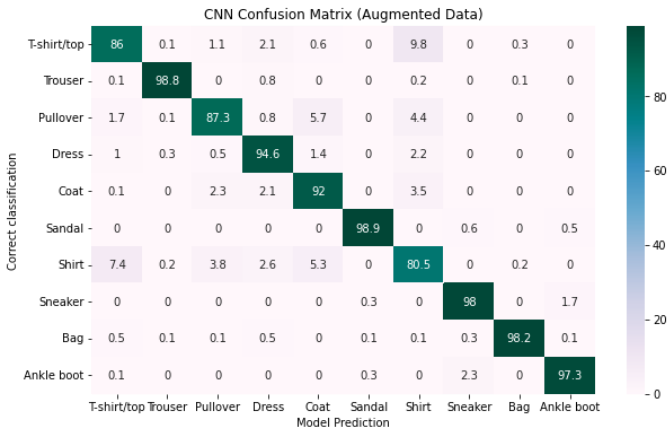


Figure 17

Training and validation loss

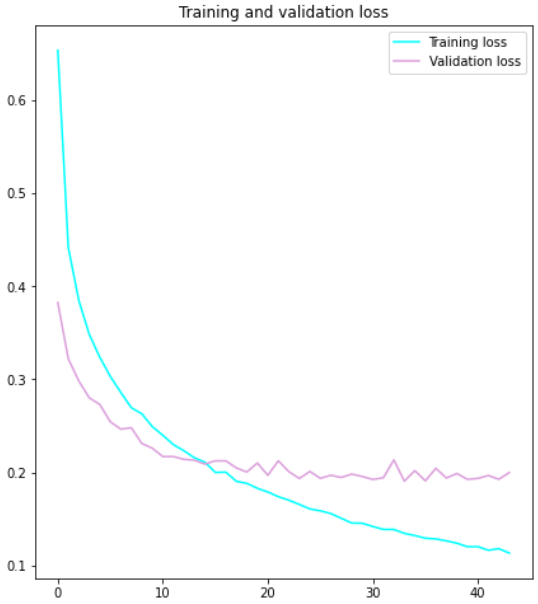
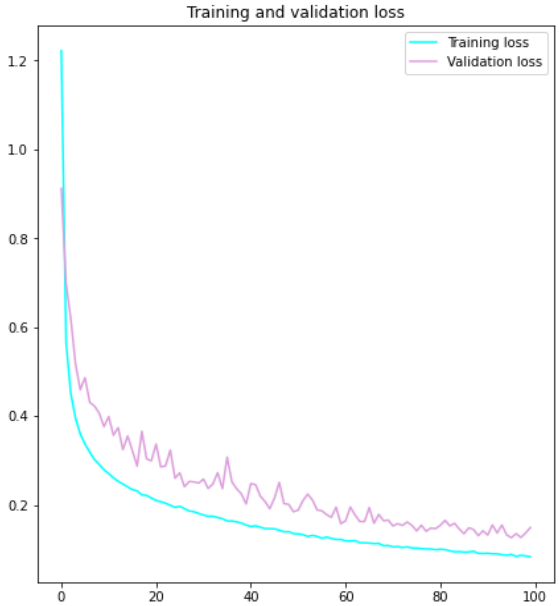


Figure 18

Training and validation loss (Augmented)



Google Colab Link:

https://colab.research.google.com/drive/1M8rK4uj4XOFdHR1rlO1PAViYn2feAgqL?usp=sharing