ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

ASSIGNMENT 1:

MLP & CNN ARCHITECTURES AS SOLUTION FOR THE CATEGORIAZATION PROBLEM OF FASHION MNIST

INTZOGLOU EVANGELOS – MASTRODIMITRIS GOUNAROPOULOS SPYRIDON

2023

Folder link:

https://drive.google.com/drive/folders/1h01FzXeGb59ciE5XKw-ZQfk4Sj-qL8C1?usp=sharing

FASHION - MNIST

Fashion-MNIST is a dataset commonly used in the field of machine learning and computer vision. It serves as a benchmark for developing and evaluating various image classification algorithms. The dataset is an alternative to the popular MNIST dataset, which consists of handwritten digits.

Fashion-MNIST contains a collection of 70,000 grayscale images of fashion-related items, categorized into 10 classes. Each image has a resolution of 28x28 pixels. The dataset is split into 60,000 training images and 10,000 test images. It was created as a more challenging replacement for MNIST, allowing researchers to explore and develop image recognition algorithms on a broader range of visual patterns.

The goal of Fashion-MNIST is to accurately classify the images into their corresponding fashion item classes. The classes include T-shirts/tops, trousers, pullovers, dresses, coats, sandals, shirts, sneakers, bags, and ankle boots.

Fashion-MNIST has gained popularity as a benchmark dataset because it provides a more realistic and complex task compared to MNIST. It allows researchers and developers to assess the performance of machine learning models in a broader context, specifically in the field of fashion image classification.

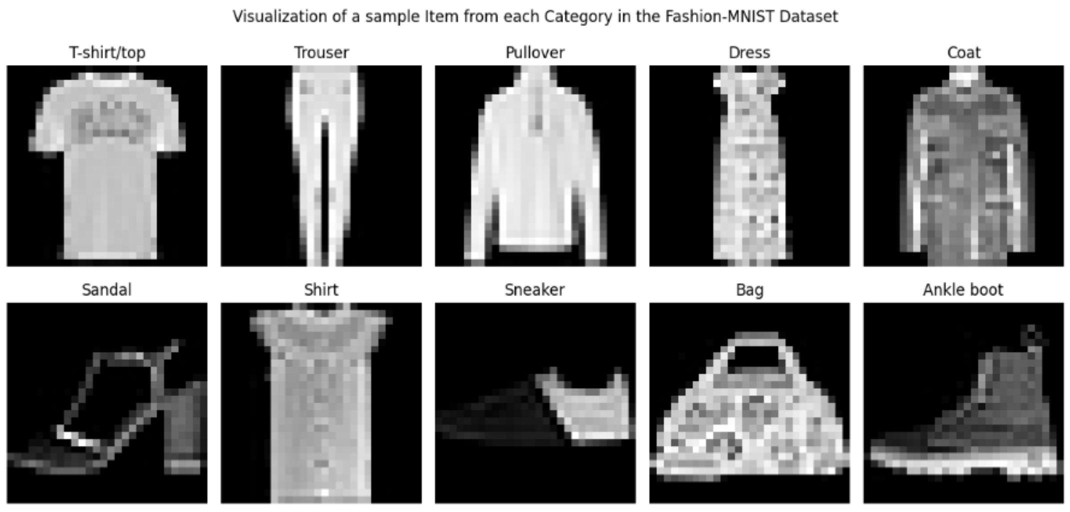
MLP:

Data Preprocessing:

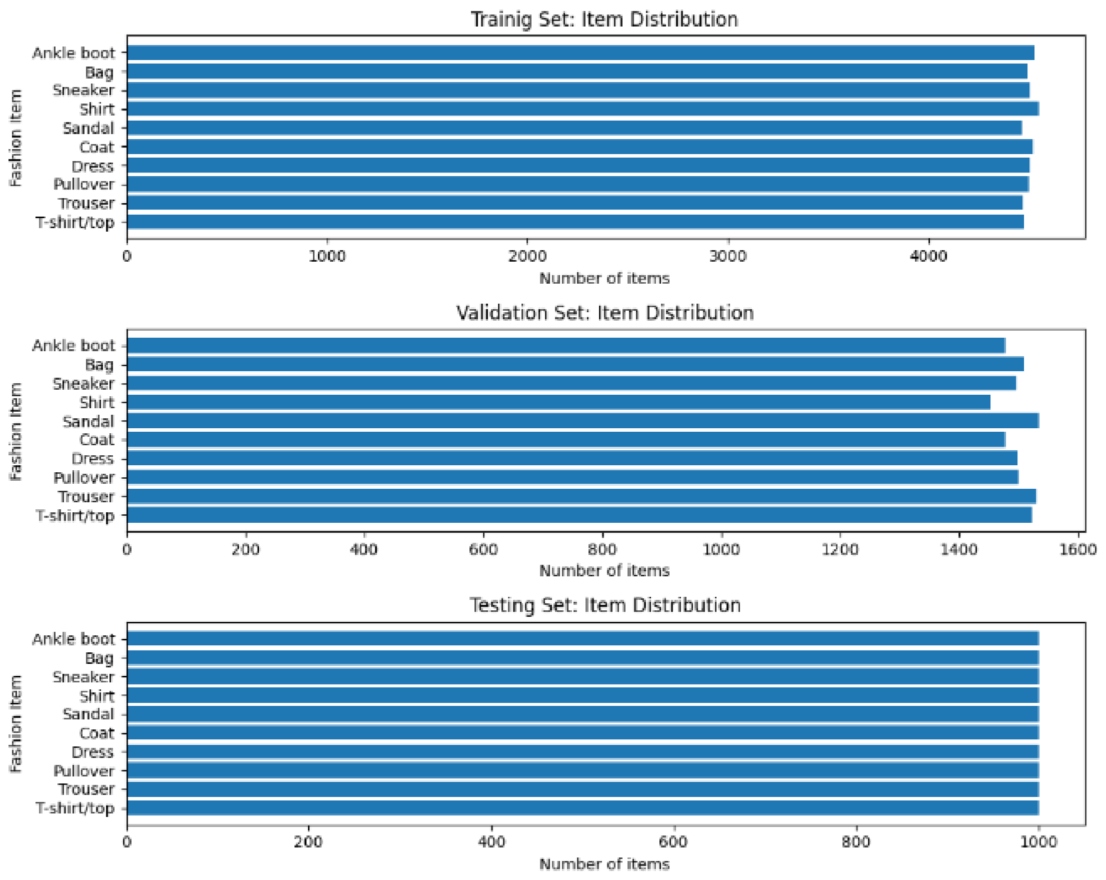
After the download of the Dataset and the import of all the necessary libraries, data preprocessing was made. We performed essential data pre-processing steps for a deep learning model that recognizes fashion items. The images from the Fashion-MNIST dataset were normalized by dividing the pixel values by 255.0, ensuring they are scaled between 0 and 1. This normalization step helps in efficient training of neural networks. Additionally, the 2D images were flattened into 1D arrays by reshaping them into a size of 784 pixels. The labels were transformed into one-hot encoded vectors using keras.utils.to_categorical(), enabling categorical training. Furthermore, the training data was further split into training and validation sets using train_test_split to facilitate model evaluation during training. These data pre-processing steps set the foundation for training a deep learning model to recognize fashion items.

Data Visualization:

Below, we focuse on visualizing a sample image from each category in the Fashion-MNIST dataset:


Visualization of a sample Item from each Category in the Fashion-MNIST Dataset
T-shirt/top
Trouser
Pullover
Dress
Coat
Sandal
Shirt
Sneaker
Bag
Ankle boot

Continuing with the data visualization we print in the following charts, the distribution of the training, validation, testing sets:



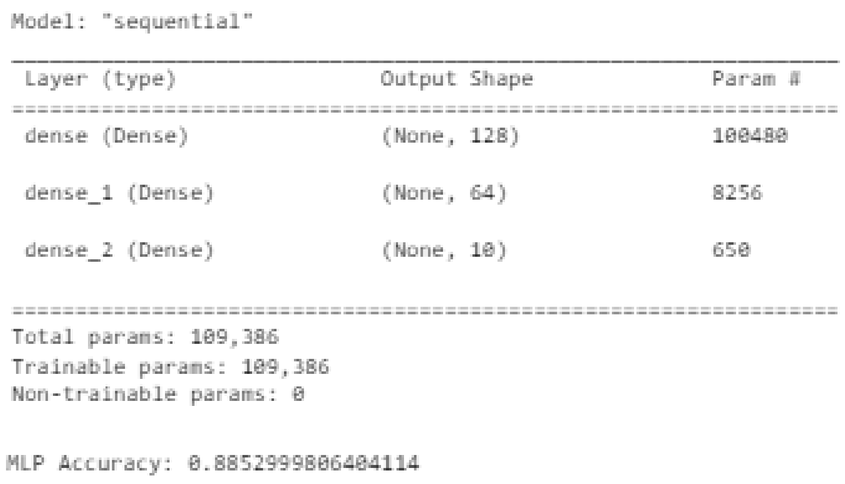
Baseline MLP Creation 1:

Model Architecture: An MLP model is constructed using the Sequential class from Keras. The model consists of three dense layers. The first dense layer has 128 units and uses the ReLU activation function. It takes an input shape of (28 * 28), corresponding to the flattened input images. The second dense layer has 64 units and also uses the ReLU activation function. The final dense layer has num_classes (10 in this case) units and uses the softmax activation function, which is suitable for multi-class classification tasks.

Model Compilation: The model is compiled with the specified loss function (categorical_crossentropy), optimizer (adam), and metric to measure the performance (accuracy). This configuration prepares the model for training.

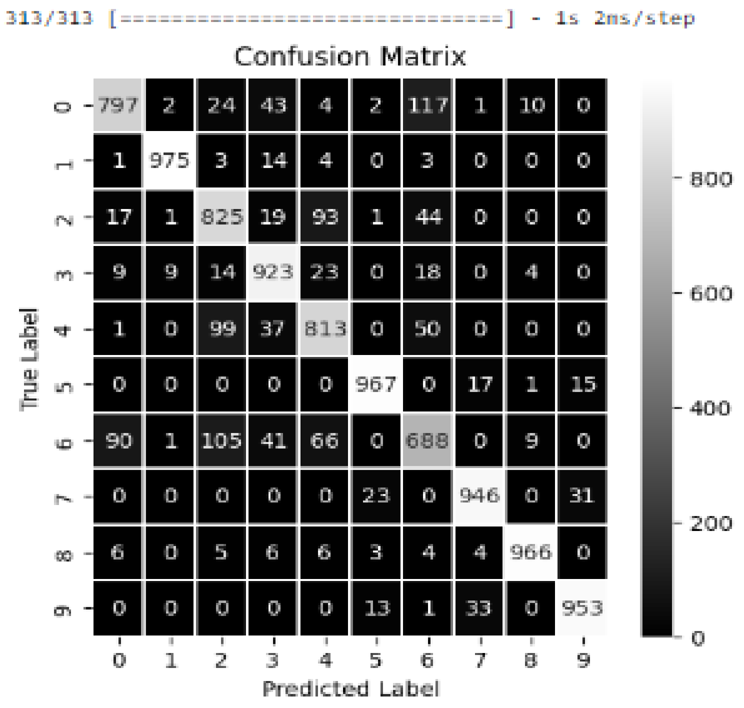
Model Training: The fit() function is used to train the MLP model. The training data (X_train and y_train) is provided, along with other parameters such as the batch size (128), the number of epochs (20), and the validation data (X_val and y_val).

Model architecture and MLP accuracy,



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 128)               100480

 dense_1 (Dense)             (None, 64)                8256

 dense_2 (Dense)             (None, 10)                650

=================================================================
Total params: 109,386
Trainable params: 109,386
Non-trainable params: 0

MLP Accuracy: 0.8852999806404114
```

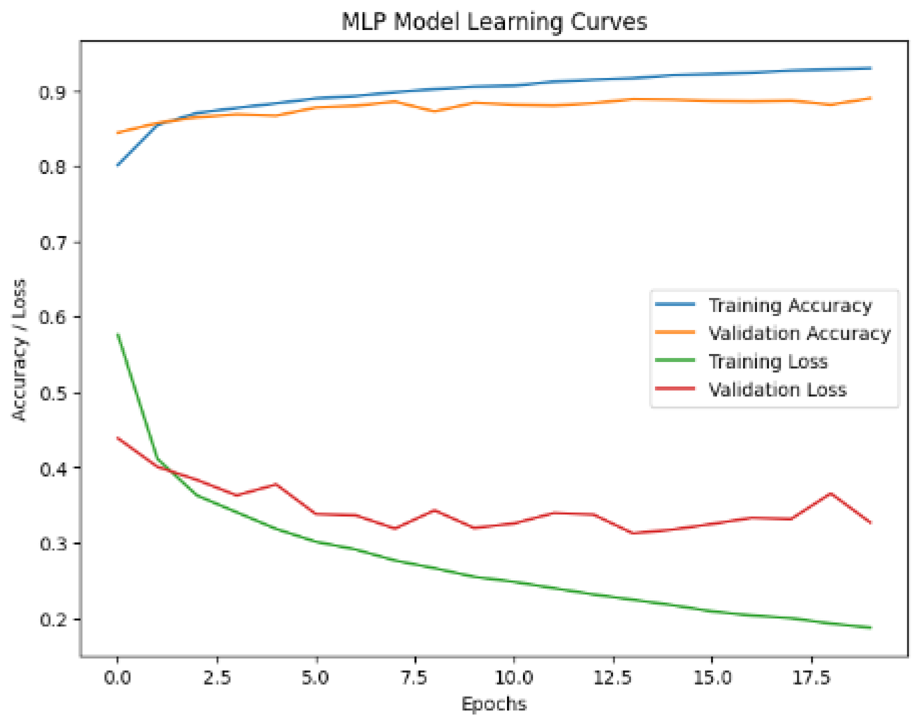
<u>Model Confusion Matrix,</u>



```
313/313 [==============================] - 1s 2ms/step
```

	0	1	2	3	4	5	6	7	8	9
0	797	2	24	43	4	2	117	1	10	0
1	1	975	3	14	4	0	3	0	0	0
2	17	1	825	19	93	1	44	0	0	0
3	9	9	14	923	23	0	18	0	4	0
4	1	0	99	37	813	0	50	0	0	0
5	0	0	0	0	0	967	0	17	1	15
6	90	1	105	41	66	0	688	0	9	0
7	0	0	0	0	0	23	0	946	0	31
8	6	0	5	6	6	3	4	4	966	0
9	0	0	0	0	0	13	1	33	0	953

<u>And the learning curves,</u>



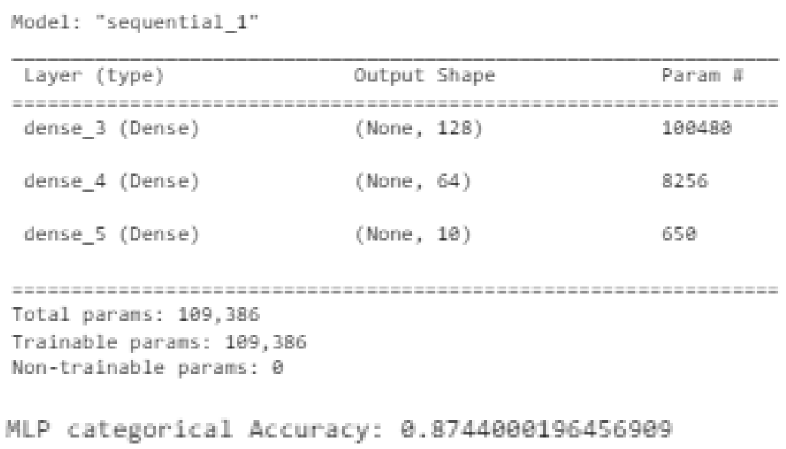
The Multi-Layer Perceptron (MLP) model achieved an accuracy of 88.53% on the test dataset. This model consists of three dense layers with 128, 64, and 10 neurons, respectively. The model was trained using the Adam optimizer and categorical cross-entropy loss function. The training process lasted for 20 epochs, with a batch size of 128.

Upon evaluating the model's performance, a confusion matrix was generated to provide insights into the classification results. The confusion matrix reveals the number of instances that were correctly classified and misclassified for each class. From the matrix, it can be observed that the model achieved relatively high accuracy for most classes, with some variations.

Baseline MLP Creation 2:

We use the same architecture as before but now we have as metric the **categorical accuracy.**

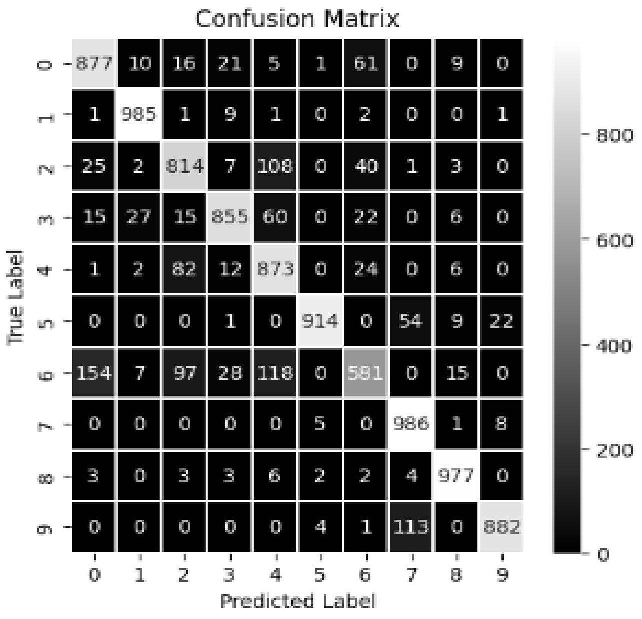
Model architecture and MLP categorical accuracy,



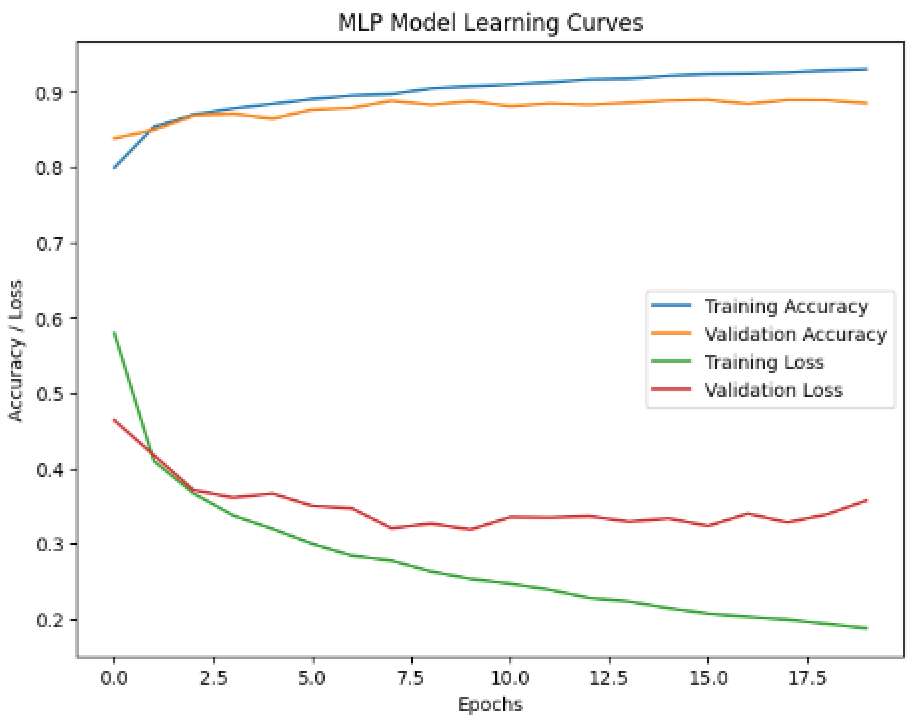
```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_3 (Dense)             (None, 128)               100480

 dense_4 (Dense)             (None, 64)                8256

 dense_5 (Dense)             (None, 10)                650

=================================================================
Total params: 109,386
Trainable params: 109,386
Non-trainable params: 0
```

```
MLP categorical Accuracy: 0.8744000196456909
```

Model Confusion Matrix,



And the learning curves,



The second MLP model achieved promising results in classifying the Fashion MNIST dataset. With an overall categorical accuracy of 87.44%, the model demonstrated its ability to accurately predict the clothing categories based on the input images. The model architecture consisted of three dense layers with 128, 64, and 10 neurons, respectively. It was trained using the Adam optimizer and categorical cross-entropy loss function.

Upon analyzing the confusion matrix, it is evident that the model performed well for most classes. Notably, it exhibited high accuracy for classes like "Trouser" (label 1). However, some confusion was observed between certain classes, resulting in misclassifications. For instance, there were instances where "Shirt" (label 6) was mistakenly predicted as "T-shirt/top" (label 0) or "Pullover" (label 2).

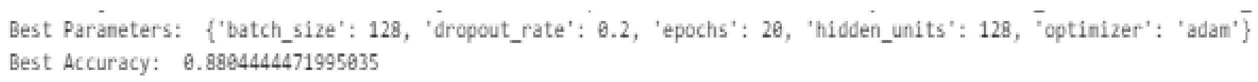
Overall, both models demonstrated their effectiveness in classifying fashion items, but the first model achieved slightly better accuracy and showed more consistent performance across different classes.

Baseline MLP Optimization 1:

This is the implementation of a grid search for hyperparameter tuning. The goal is to optimize the performance of the initial MLP model by finding the best combination of hyperparameters. The GridSearchCV object is instantiated with the MLP model wrapper, the parameter grid, and other settings such as the number of cross-validation folds (cv) and verbosity. By performing the grid search, the code aims to find the best combination of hyperparameters that maximizes the model's accuracy on the validation set, leading to improved performance compared to the initial MLP model.

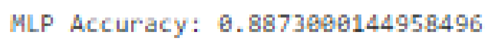
Model architecture and MLP accuracy,

The MLP architecture consists of multiple dense layers with ReLU activation, followed by dropout layers to mitigate overfitting, and a final dense layer with softmax activation for multi-class classification. The model is compiled with the categorical cross-entropy loss function and the chosen optimizer.



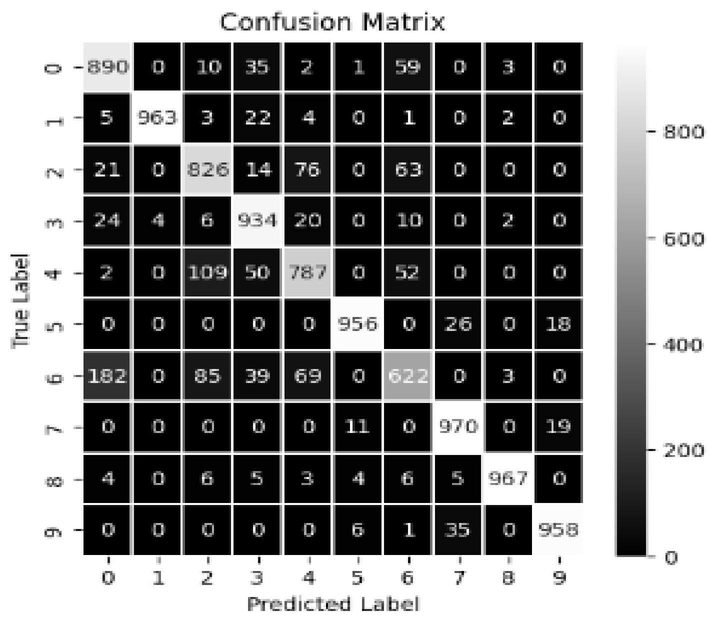
```
Best Parameters:  {'batch_size': 128, 'dropout_rate': 0.2, 'epochs': 20, 'hidden_units': 128, 'optimizer': 'adam'}
Best Accuracy:  0.8804444471995035
```

Then the best MLP model obtained from the grid search, best_mlp_model, is trained on the entire training dataset (train_images and train_labels). The model is trained for 10 epochs with a batch size of 128. The training progress is stored in the history object.

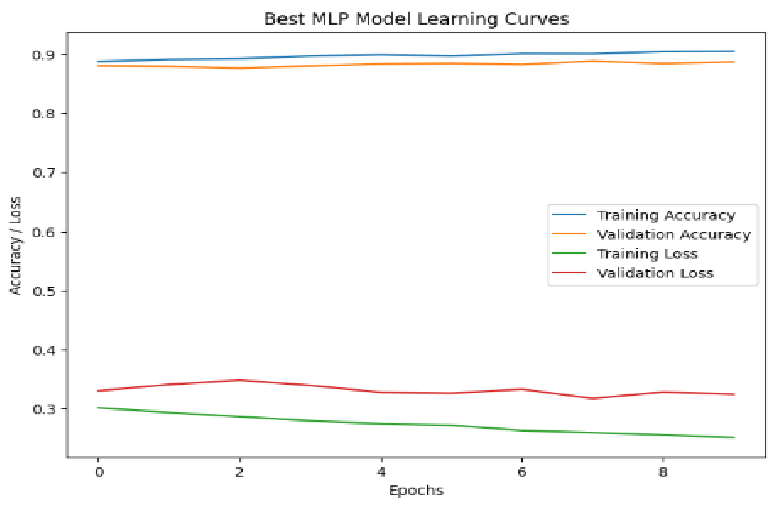


```
MLP Accuracy: 0.8873000144958496
```

<u>Model Confusion Matrix,</u>



<u>And the learning curves,</u>



After the grid search, the best set of hyperparameters is determined, which consists of a batch size of 128, dropout rate of 0.2, 20 epochs, 128 hidden units, and the 'adam' optimizer. The model achieved the best accuracy of approximately 88.04% with these hyperparameters.

The best MLP model, based on the best parameters found through grid search, was trained on the entire training dataset using these parameters.The model was then evaluated on the test dataset, yielding an accuracy of approximately 88.73%.

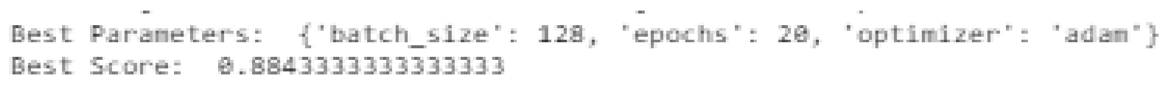
The learning curves plot for the best MLP model showed the training accuracy, validation accuracy, training loss, and validation loss over the 10 training epochs. It demonstrated that the model's accuracy increased gradually and stabilized, while the loss decreased during training.

Overall, the third model, with its optimized hyperparameters obtained through grid search, performed well in classifying the fashion items in the test dataset. The model achieved an accuracy of approximately 88.73%, indicating its effectiveness in recognizing different types of clothing items.

<u>Baseline MLP Optimization 2:</u>

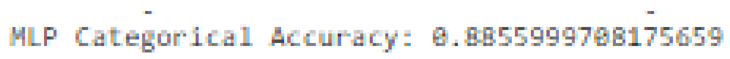
In this implementation we use again a grid search but this time we use categorical accuracy and also we put other parameters in the grid search.

<u>Model architecture and MLP categorical accuracy,</u>



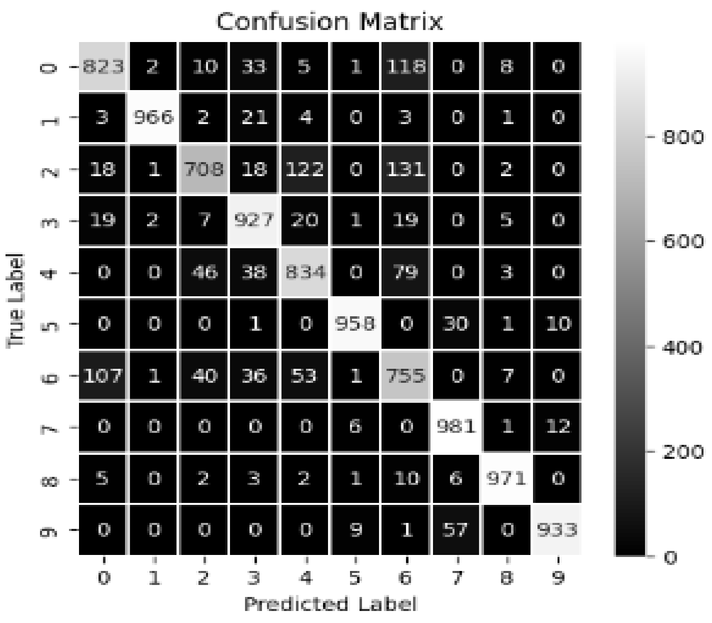
```
Best Parameters:  {'batch_size': 128, 'epochs': 20, 'optimizer': 'adam'}
Best Score:  0.8843333333333333
```

The MLP consists of an input layer with flattened input shape, a hidden layer with 128 units and ReLU activation, and an output layer with softmax activation for multi-class classification. The model was trained using the Adam optimizer with a batch size of 128 for 20 epochs.



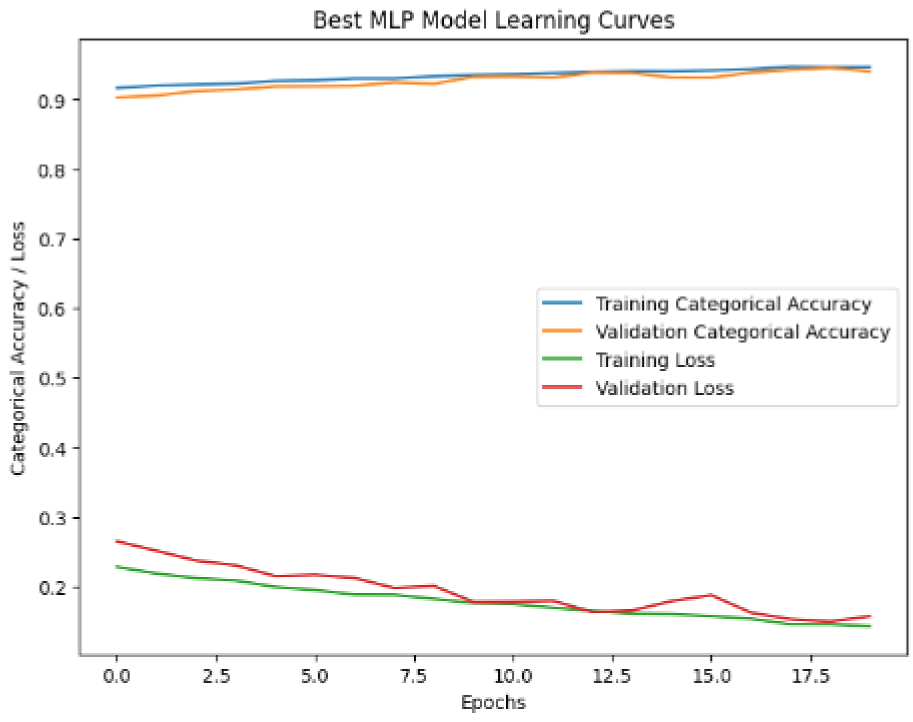
```
MLP Categorical Accuracy: 0.8855999708175659
```

<u>Model Confusion Matrix,</u>



True Label \ Predicted Label	0	1	2	3	4	5	6	7	8	9
0	823	2	10	33	5	1	118	0	8	0
1	3	966	2	21	4	0	3	0	1	0
2	18	1	708	18	122	0	131	0	2	0
3	19	2	7	927	20	1	19	0	5	0
4	0	0	46	38	834	0	79	0	3	0
5	0	0	0	1	0	958	0	30	1	10
6	107	1	40	36	53	1	755	0	7	0
7	0	0	0	0	0	6	0	981	1	12
8	5	0	2	3	2	1	10	6	971	0
9	0	0	0	0	0	9	1	57	0	933

<u>And the learning curves,</u>



The results for the model, including grid search and the final evaluation, are as follows:

Best Parameters: {'batch_size': 128, 'epochs': 20, 'optimizer': 'adam'},

Best Score: 0.8843

After performing grid search with cross-validation, the best parameters were determined to be a batch size of 128, 20 epochs, and the Adam optimizer. The model achieved a categorical accuracy of approximately 0.8856 on the test set. The learning curves of the model demonstrate that the training and validation categorical accuracy improved steadily over the course of training, indicating effective learning. The model also achieved a relatively low training and validation loss, suggesting good generalization capabilities.

CONVOLUTIONAL NEURAL NETWORK

INTRODUCTION

A convolutional neural network (CNN) is a deep learning algorithm designed to process and analyze visual data, particularly images. It is composed of multiple layers of interconnected neurons that extract and learn meaningful patterns and features from the input images. The key element of CNNs is the convolutional layer, which applies a set of learnable filters to the input image, capturing local spatial dependencies and detecting important visual features. This hierarchical approach allows CNNs to automatically learn and recognize complex patterns, such as edges, textures, and shapes, enabling them to excel in image classification tasks. CNNs have proven to be highly effective in image classification due to their ability to exploit the inherent spatial structure of images and their ability to learn and generalize from large-scale image datasets. Their success in various computer vision tasks has made them a cornerstone in the field, achieving state-of-the-art performance and providing valuable insights into the visual information processing mechanisms of the human brain. For the FASHION-MNIST problem a CNN will provide an excellent solution.

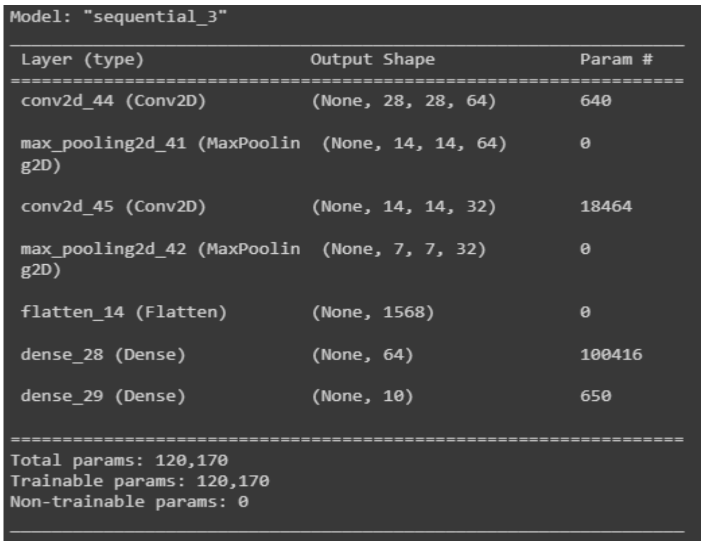
DATA PREPARATION

The FASHION-MNIST classification problem is well-known and defined, its dataset is clean and with no missing values and all images are set in the same 0^0 orientation. Since are imported using Keras, the data are already split up 60000/10000 to train data, test labels and test data, test labels. The first step was to change the labels to categorical and for the images to change the dimensions, adding one channel since the images are in grayscale.

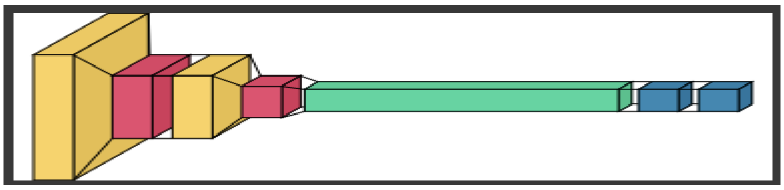
Finally before the creation of any models the train image and labels set was further split down to a new train-set 75% and a new validate-set 25% of the initial set.

FIRST ATTEMPT

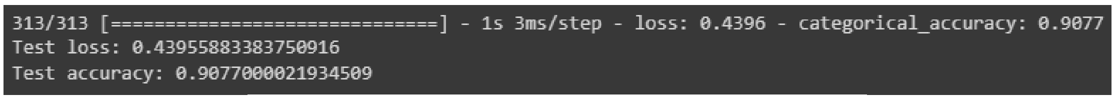
For this problem I wanted to understand the problem and make any changes by hand so to see the impact rather than use an automated optimization method, even since the random search had already been implemented for the first part of the assignment. Because I wanted a simple CNN to start I started with only a couple of convolution layer. For all activations I used the old reliable relu. Also I compiled the model using Adam and categorical accuracy.



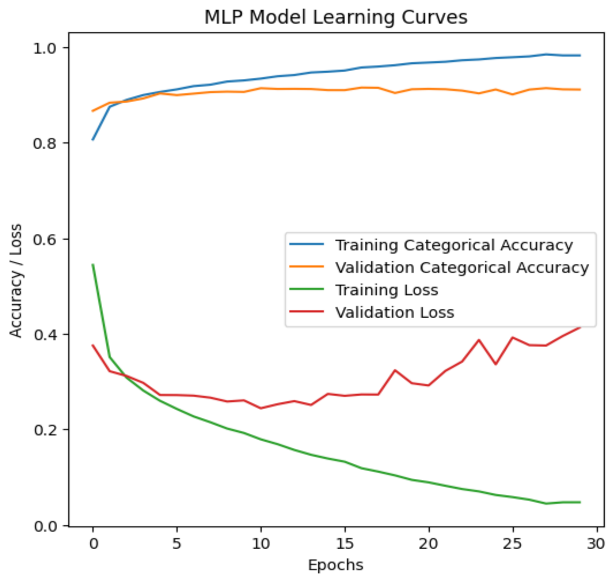
```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_44 (Conv2D)          (None, 28, 28, 64)        640

 max_pooling2d_41 (MaxPoolin  (None, 14, 14, 64)       0
 g2D)

 conv2d_45 (Conv2D)          (None, 14, 14, 32)        18464

 max_pooling2d_42 (MaxPoolin  (None, 7, 7, 32)         0
 g2D)

 flatten_14 (Flatten)        (None, 1568)              0

 dense_28 (Dense)            (None, 64)                100416

 dense_29 (Dense)            (None, 10)                650

=================================================================
Total params: 120,170
Trainable params: 120,170
Non-trainable params: 0
_________________________________________________________________
```



The results were exceptional for a first attempt as it achieved a high accuracy unfortunately this was also followed by a high overfitting to the training data.

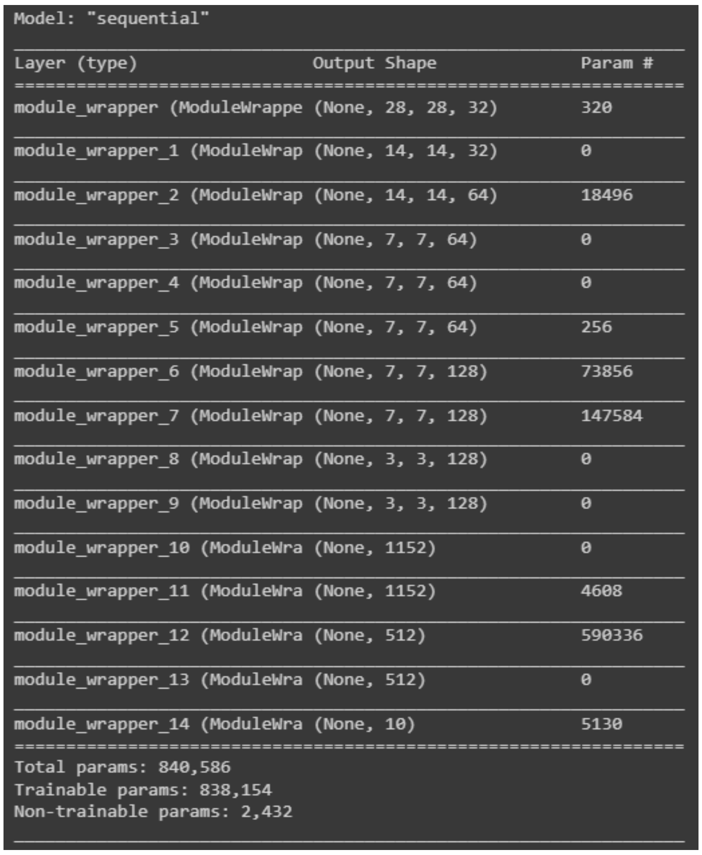


```
313/313 [==============================] - 1s 3ms/step - loss: 0.4396 - categorical_accuracy: 0.9077
Test loss: 0.439558833837500916
Test accuracy: 0.9077000021934509
```



SECOND ATTEMPT

Since the results were so encouraging, I tried to add more layer to increase the descriptive ability of the mode, hopping by increasing the accuracy of the trainset from the first epochs to increase the accuracy of the test set as well. I also added dropout to equalize the expected overfitting produced by the increase of the model's ability to understand the train data.



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
module_wrapper (ModuleWrappe (None, 28, 28, 32)        320
_________________________________________________________________
module_wrapper_1 (ModuleWrap (None, 14, 14, 32)        0
_________________________________________________________________
module_wrapper_2 (ModuleWrap (None, 14, 14, 64)        18496
_________________________________________________________________
module_wrapper_3 (ModuleWrap (None, 7, 7, 64)          0
_________________________________________________________________
module_wrapper_4 (ModuleWrap (None, 7, 7, 64)          0
_________________________________________________________________
module_wrapper_5 (ModuleWrap (None, 7, 7, 64)          256
_________________________________________________________________
module_wrapper_6 (ModuleWrap (None, 7, 7, 128)         73856
_________________________________________________________________
module_wrapper_7 (ModuleWrap (None, 7, 7, 128)         147584
_________________________________________________________________
module_wrapper_8 (ModuleWrap (None, 3, 3, 128)         0
_________________________________________________________________
module_wrapper_9 (ModuleWrap (None, 3, 3, 128)         0
_________________________________________________________________
module_wrapper_10 (ModuleWra (None, 1152)              0
_________________________________________________________________
module_wrapper_11 (ModuleWra (None, 1152)              4608
_________________________________________________________________
module_wrapper_12 (ModuleWra (None, 512)               590336
_________________________________________________________________
module_wrapper_13 (ModuleWra (None, 512)               0
_________________________________________________________________
module_wrapper_14 (ModuleWra (None, 10)                5130
=================================================================
Total params: 840,586
Trainable params: 838,154
Non-trainable params: 2,432
_________________________________________________________________
```

In the above cryptic message is stated that for this model we added 2 more convolution layers having sizes 32, 64, 128, 128 a larger dense layer after the flattening, having 512 units. The resulting model has around 700.000 more trainable parameters than the first.

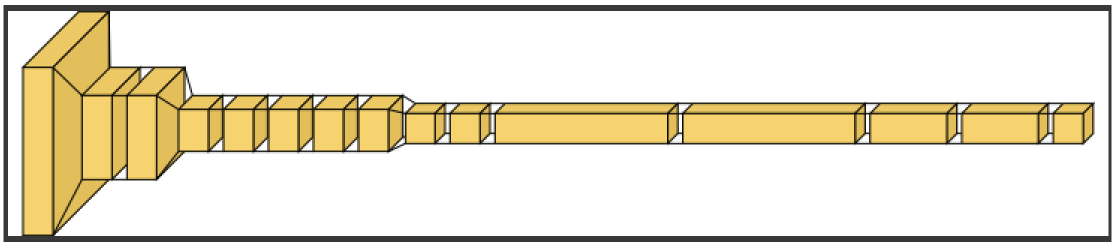
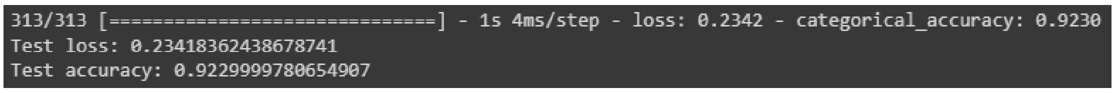


Figure 4: Model 2 Architecture

Indeed, this model archives better results compared to the previous one.


```
313/313 [==============================] - 1s 4ms/step - loss: 0.2342 - categorical_accuracy: 0.9230
Test loss: 0.23418362438678741
Test accuracy: 0.9229999780654907
```
Figure 5: Model 2 Results

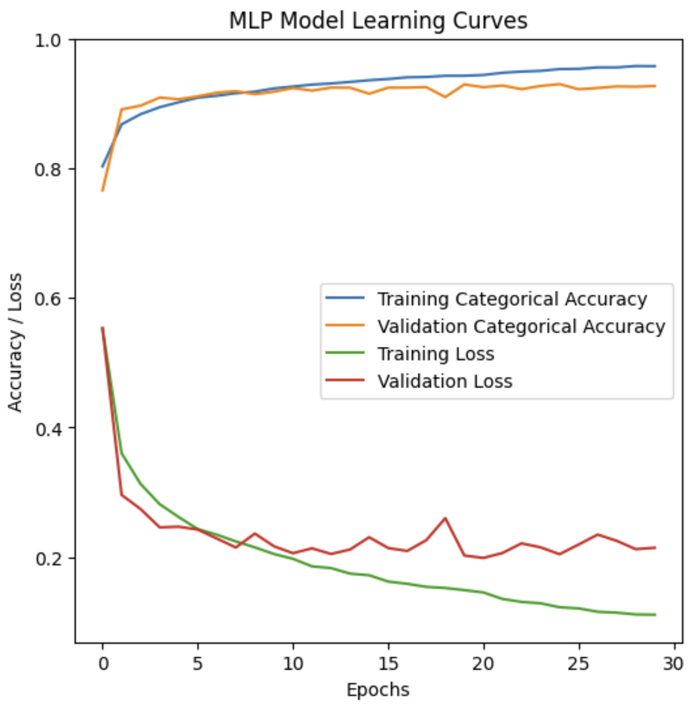
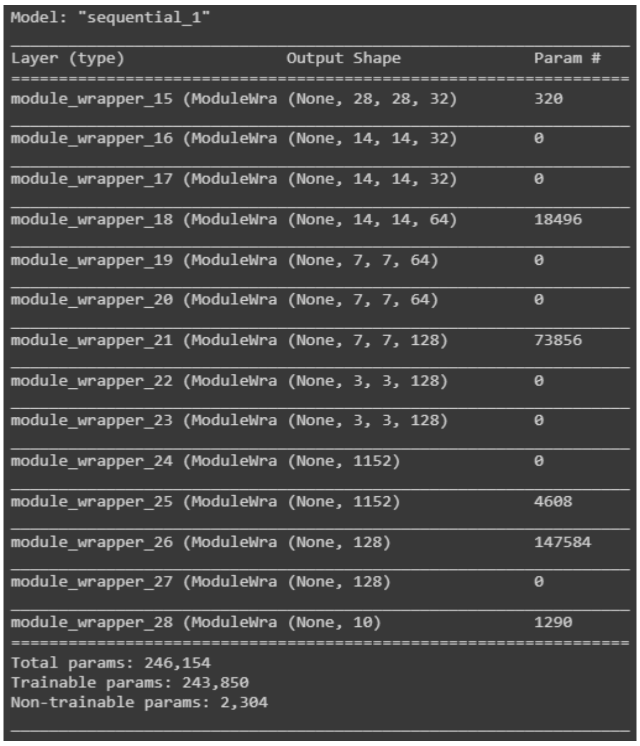


Figure 6: Model 2 Results

Unexpectedly the effect of the dropout was more influential than the 8 time increased number of neurons and the overfitting is still prevalent but less pronounced than before.

THIRD ATTEMPT

At this point I concluded that the train accuracy increasing brings diminishing returns for the test results. So I backtracked to a simpler model but without loosing accuracy. So I dropped a convolution layer increased the presence of dropout and decreased the final dense before the softmax to the one quarter of the previous model.



```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
module_wrapper_15 (ModuleWra (None, 28, 28, 32)        320
_________________________________________________________________
module_wrapper_16 (ModuleWra (None, 14, 14, 32)        0
_________________________________________________________________
module_wrapper_17 (ModuleWra (None, 14, 14, 32)        0
_________________________________________________________________
module_wrapper_18 (ModuleWra (None, 14, 14, 64)        18496
_________________________________________________________________
module_wrapper_19 (ModuleWra (None, 7, 7, 64)          0
_________________________________________________________________
module_wrapper_20 (ModuleWra (None, 7, 7, 64)          0
_________________________________________________________________
module_wrapper_21 (ModuleWra (None, 7, 7, 128)         73856
_________________________________________________________________
module_wrapper_22 (ModuleWra (None, 3, 3, 128)         0
_________________________________________________________________
module_wrapper_23 (ModuleWra (None, 3, 3, 128)         0
_________________________________________________________________
module_wrapper_24 (ModuleWra (None, 1152)              0
_________________________________________________________________
module_wrapper_25 (ModuleWra (None, 1152)              4608
_________________________________________________________________
module_wrapper_26 (ModuleWra (None, 128)               147584
_________________________________________________________________
module_wrapper_27 (ModuleWra (None, 128)               0
_________________________________________________________________
module_wrapper_28 (ModuleWra (None, 10)                1290
=================================================================
Total params: 246,154
Trainable params: 243,850
Non-trainable params: 2,304
_________________________________________________________________
```

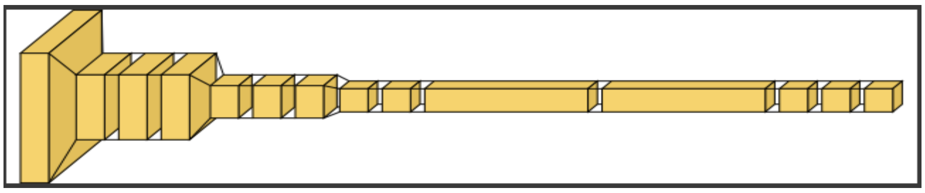
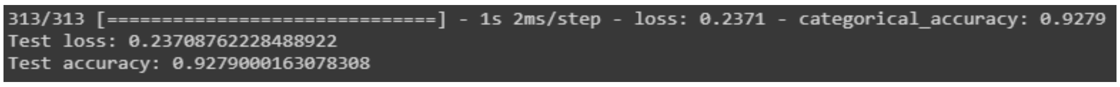


Figure 8: Model 3 Architecture

The results improved but not by much, as the test accuracy from multiple tests hovers around 93%.



```
313/313 [==============================] - 1s 2ms/step - loss: 0.2371 - categorical_accuracy: 0.9279
Test loss: 0.23708762228488922
Test accuracy: 0.9279000163078308
```

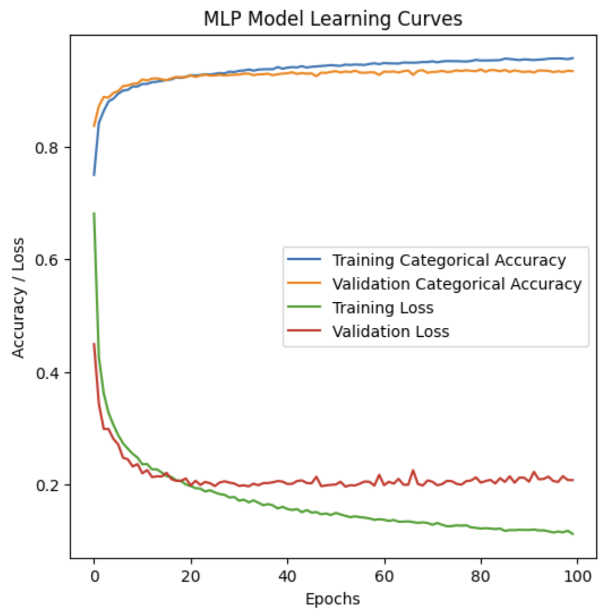
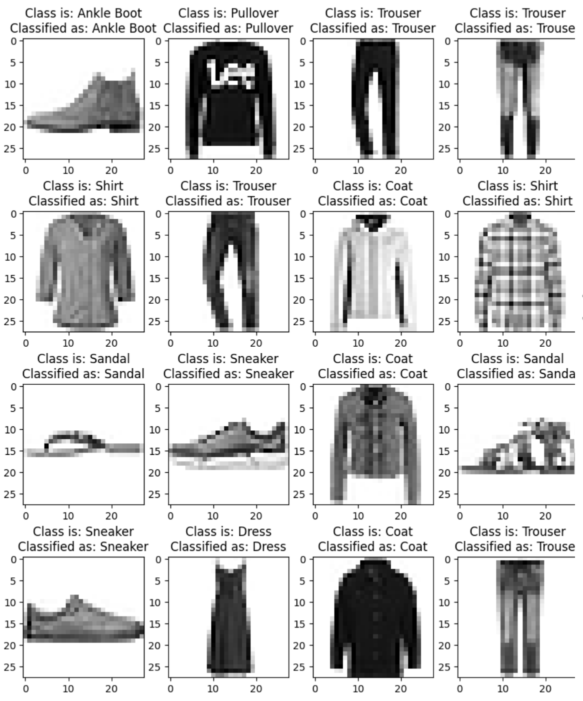
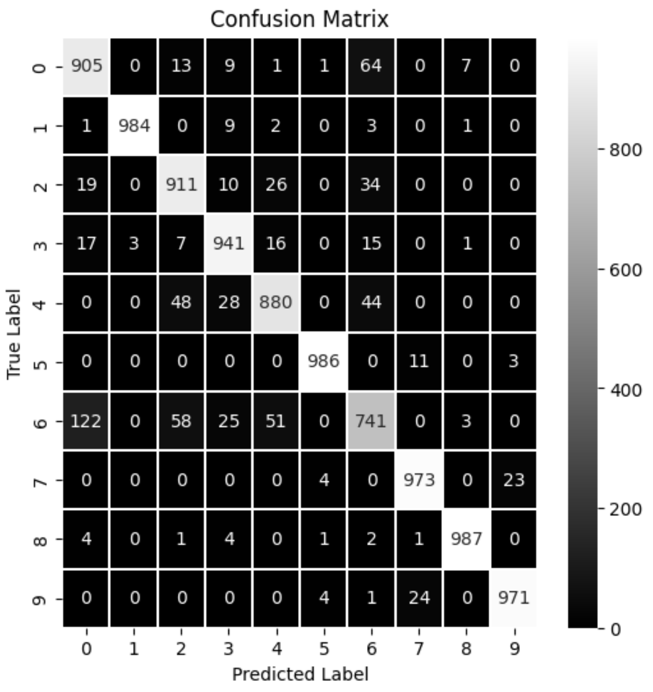


Figure 9: Model 3 Evaluation

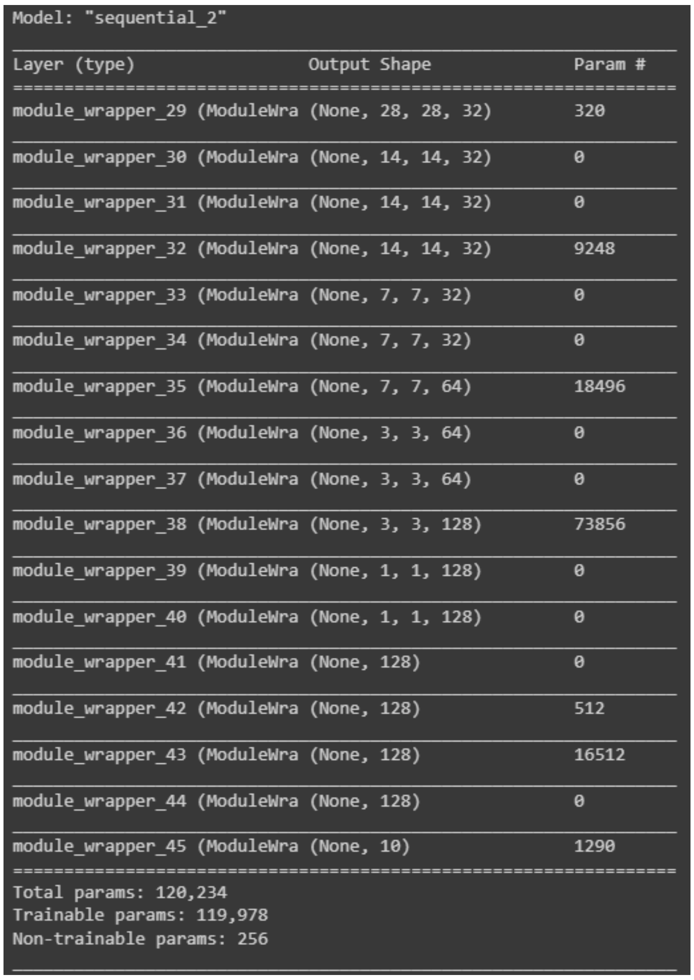




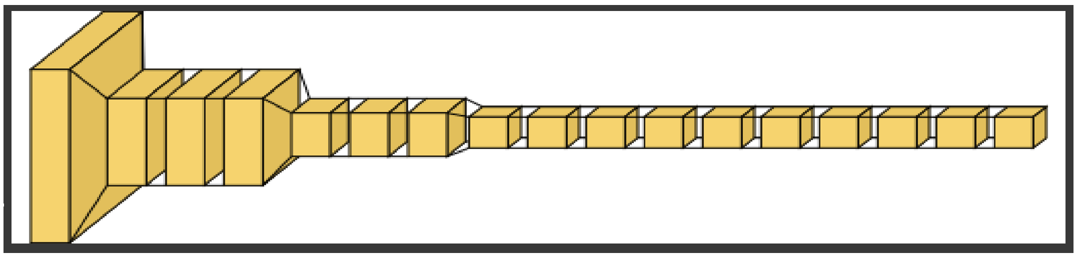
True Label \ Predicted Label	0	1	2	3	4	5	6	7	8	9
0	905	0	13	9	1	1	64	0	7	0
1	1	984	0	9	2	0	3	0	1	0
2	19	0	911	10	26	0	34	0	0	0
3	17	3	7	941	16	0	15	0	1	0
4	0	0	48	28	880	0	44	0	0	0
5	0	0	0	0	0	986	0	11	0	3
6	122	0	58	25	51	0	741	0	3	0
7	0	0	0	0	0	4	0	973	0	23
8	4	0	1	4	0	1	2	1	987	0
9	0	0	0	0	0	4	1	24	0	971

FORTH ATTEMPT

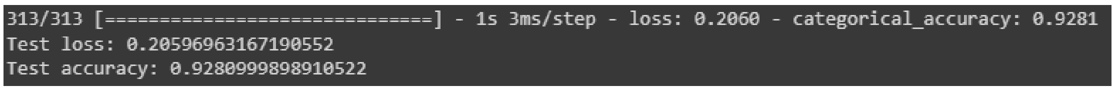
After dropping the number of parameters to a quarter of a million we asked the question if the same results could be replicated to half of those. So we reduced the size of the first convolution layers and decreased the dropout rate a bit.



```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                Output Shape              Param #
=================================================================
module_wrapper_29 (ModuleWra (None, 28, 28, 32)       320
_________________________________________________________________
module_wrapper_30 (ModuleWra (None, 14, 14, 32)       0
_________________________________________________________________
module_wrapper_31 (ModuleWra (None, 14, 14, 32)       0
_________________________________________________________________
module_wrapper_32 (ModuleWra (None, 14, 14, 32)       9248
_________________________________________________________________
module_wrapper_33 (ModuleWra (None, 7, 7, 32)         0
_________________________________________________________________
module_wrapper_34 (ModuleWra (None, 7, 7, 32)         0
_________________________________________________________________
module_wrapper_35 (ModuleWra (None, 7, 7, 64)         18496
_________________________________________________________________
module_wrapper_36 (ModuleWra (None, 3, 3, 64)         0
_________________________________________________________________
module_wrapper_37 (ModuleWra (None, 3, 3, 64)         0
_________________________________________________________________
module_wrapper_38 (ModuleWra (None, 3, 3, 128)        73856
_________________________________________________________________
module_wrapper_39 (ModuleWra (None, 1, 1, 128)        0
_________________________________________________________________
module_wrapper_40 (ModuleWra (None, 1, 1, 128)        0
_________________________________________________________________
module_wrapper_41 (ModuleWra (None, 128)              0
_________________________________________________________________
module_wrapper_42 (ModuleWra (None, 128)              512
_________________________________________________________________
module_wrapper_43 (ModuleWra (None, 128)              16512
_________________________________________________________________
module_wrapper_44 (ModuleWra (None, 128)              0
_________________________________________________________________
module_wrapper_45 (ModuleWra (None, 10)               1290
=================================================================
Total params: 120,234
Trainable params: 119,978
Non-trainable params: 256
_________________________________________________________________
```



The results were a small improvement upon Model 3 with only half the number of weights.



```
313/313 [==============================] - 1s 3ms/step - loss: 0.2060 - categorical_accuracy: 0.9281
Test loss: 0.20596963167190552
Test accuracy: 0.9280999898910522
```

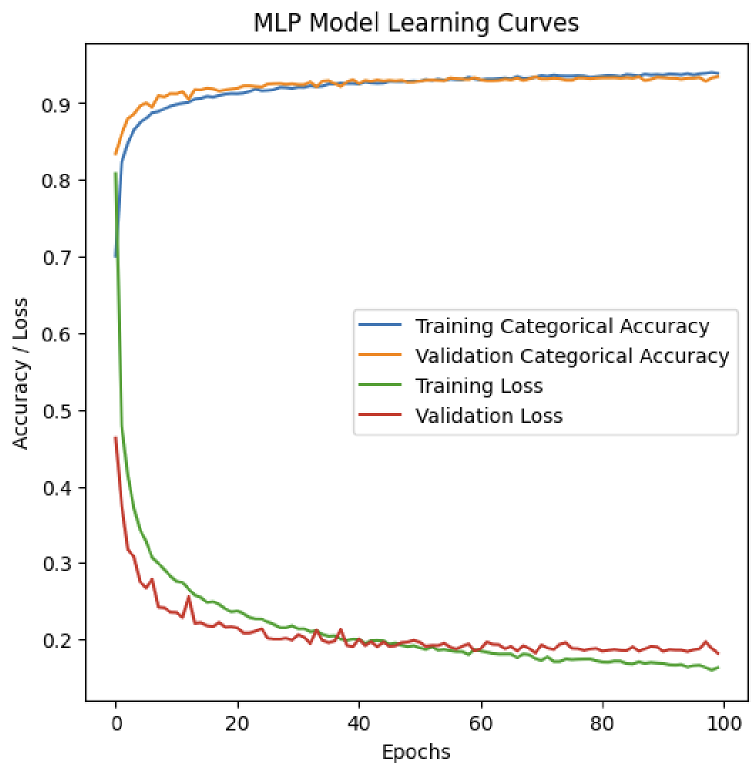
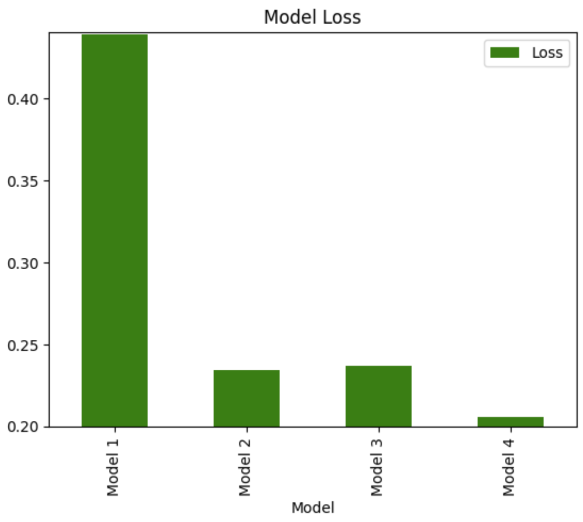


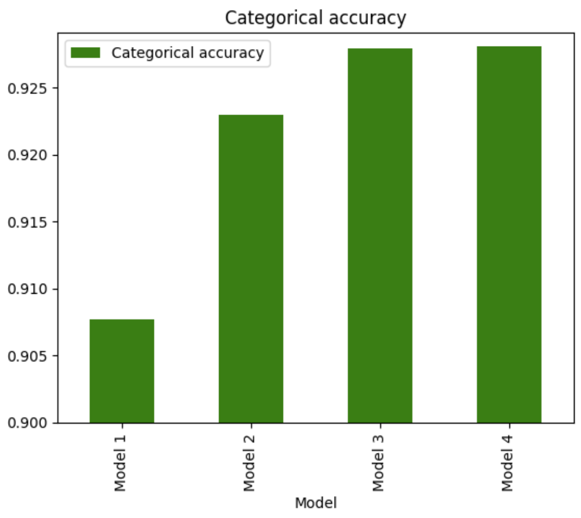
Figure 11: Model 4 Evaluation

OTHER TESTS

There were attempts to decrease the loss further by simplifying further and trying different batch sizes, but the results were inconclusive failing to show meaningful results. As from all experiment accuracy never moved up or down more than 0.4%.

CNN CONCLUSION





Model 4 proved to be the lighter, more accurate with the least losses and with almost no overfitting. Unfortunately, it is likely that another model might exist that archives a higher categorical test accuracy, but it will likely be more complex than Model 4.