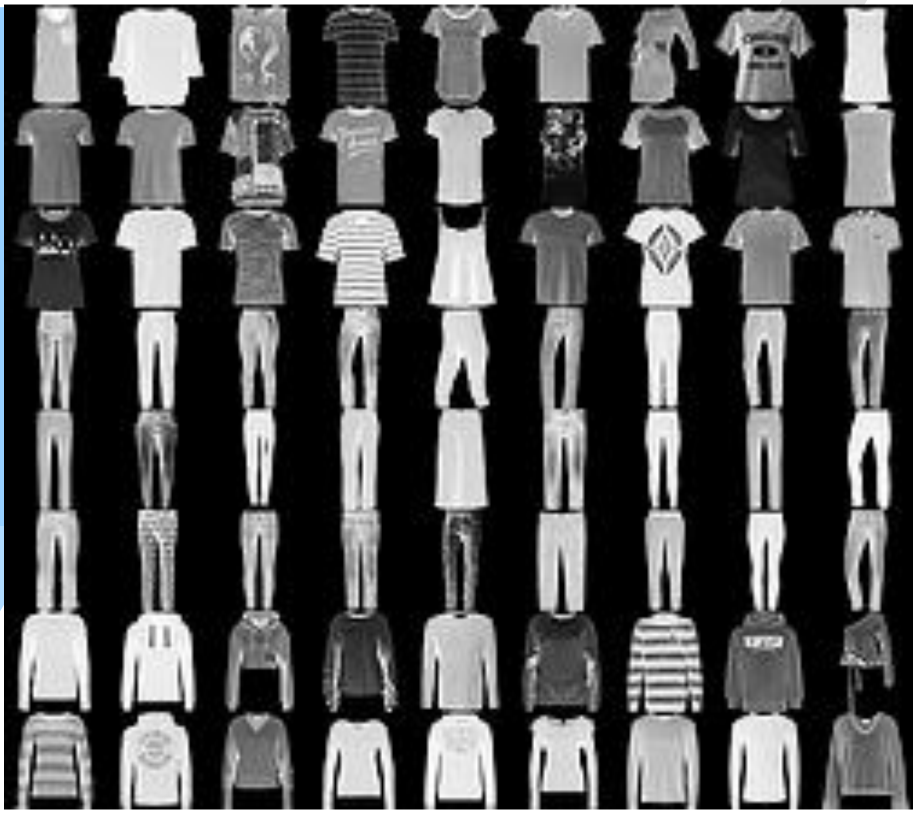
Deep Learning Project I

Email:
[gchalkiopoulos@aueb.gr - p3352124, ntsoukalelis@aueb.gr - p3352123]

MLPs and CNN for image classification on fashion-mnist dataset



Team members: Georgios Chalkiopoulos, Nikolaos Tsoukalelis

ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

CONTENTS

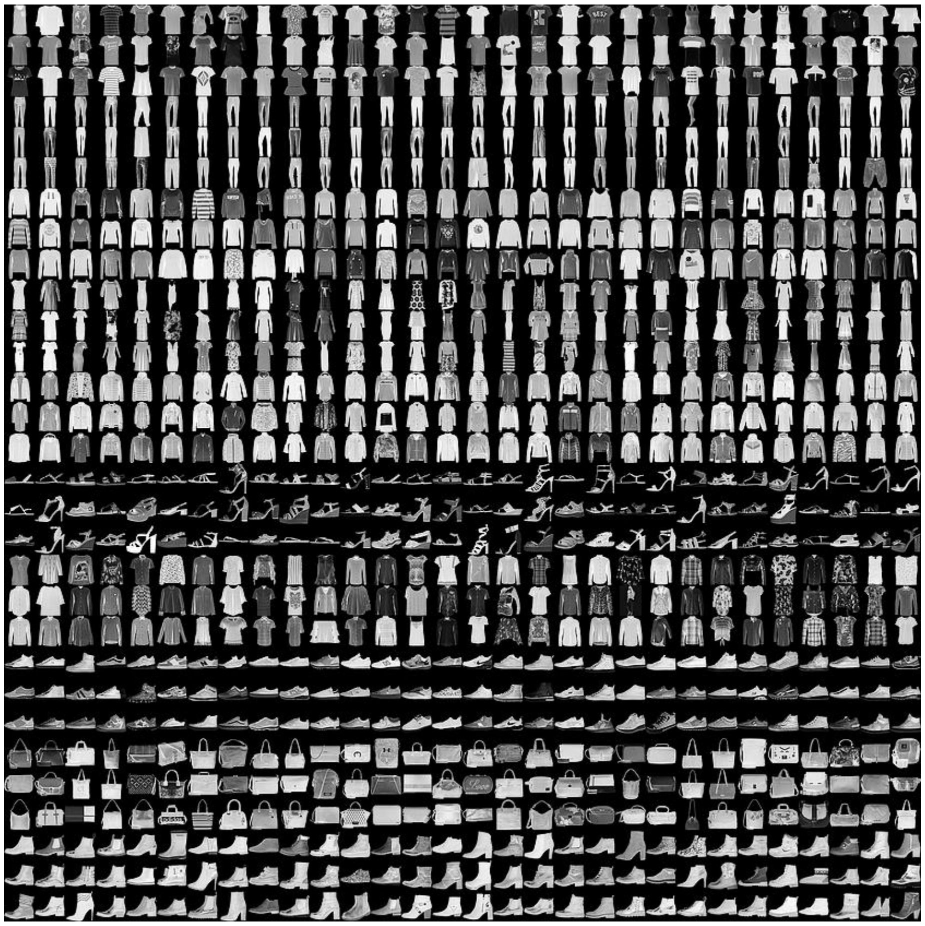
Problem Description

Assignment and Dataset description

Given an image of a fashion item, we had to build a deep learning model that recognizes the fashion item. We had to use at least 2 different architectures, one with MLPs and one with CNNs. We used the Fashion-MNIST dataset to train and evaluate our models. More information about the task and the dataset can be found at https://github.com/zalandoresearch/fashion-mnist. The dataset is also available from Tensorflow.

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. Each example is a 28x28 grayscale image, associated with a label from 10 classes. It shares the same image size and structure of training and testing splits.

Here's an example of how the data looks (each class takes three-rows):

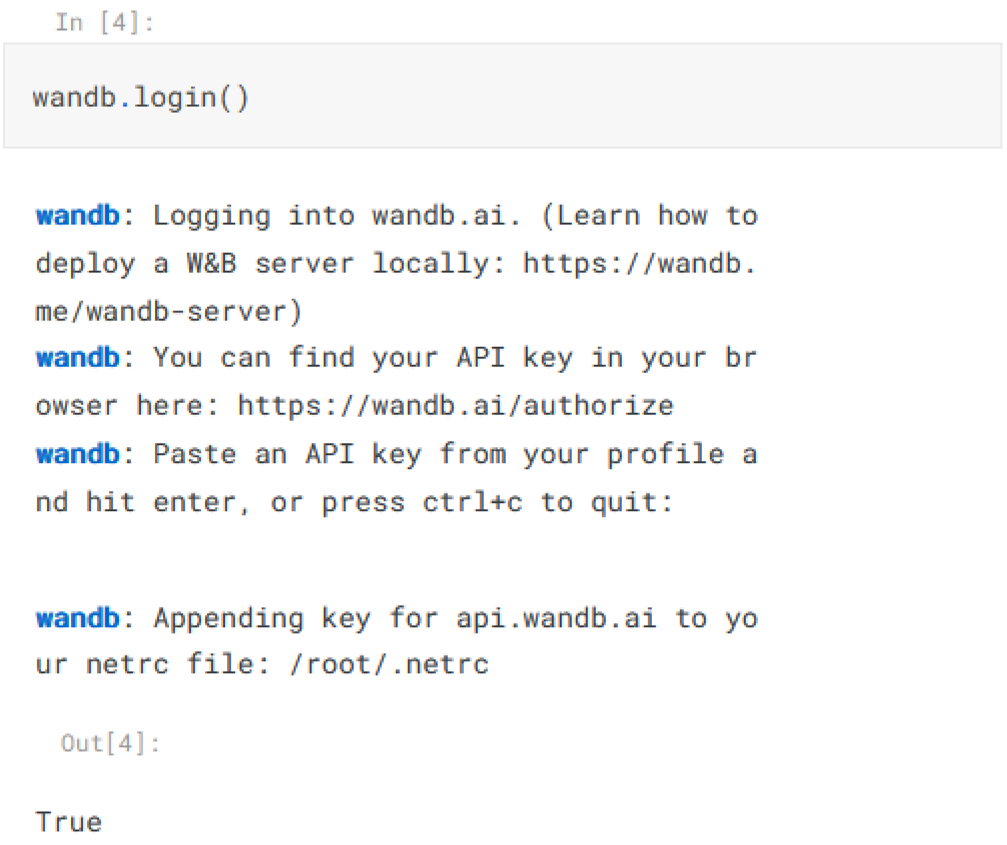


Weights & Biases

Weights & Biases API description

In order to make experiment tracking easier we will use Weights & Biases, which offers a free lisence for academic purposes. For the sake of this assignment a team has been created https://wandb.ai/aueb . Access can be granted by contacting the authors.

Note that this code assumes that you have already set up a Wandb account and API key. If you haven't done so yet, you will need to sign up for a free account at https://wandb.ai/ and follow the instructions there to obtain your API key.



```
In [4]:
wandb.login()
```

```
wandb: Logging into wandb.ai. (Learn how to
deploy a W&B server locally: https://wandb.
me/wandb-server)
wandb: You can find your API key in your br
owser here: https://wandb.ai/authorize
wandb: Paste an API key from your profile a
nd hit enter, or press ctrl+c to quit:


wandb: Appending key for api.wandb.ai to yo
ur netrc file: /root/.netrc
```

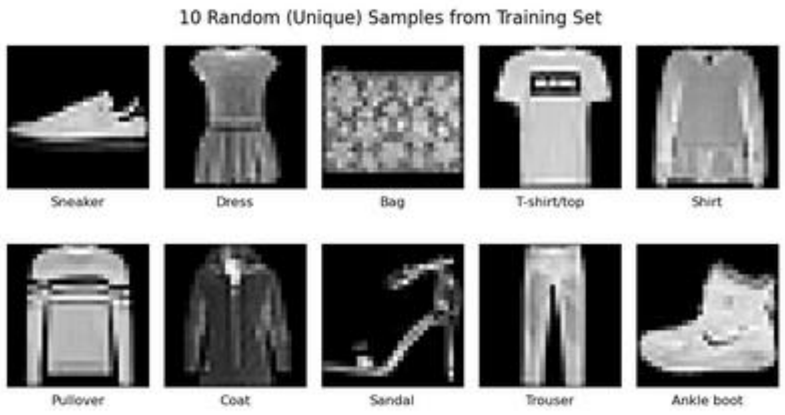
```
Out[4]:
True
```

Visualize

Training Dataset

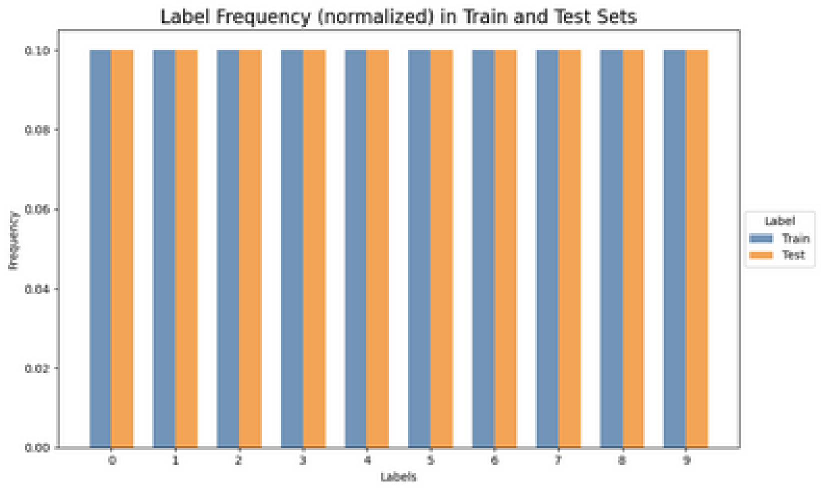
Before proceeding in the implementation of the model and further work on the assignment we produced some visualizations regarding the dataset.

Firstly we produce random samples from Training Set.



Imbalance

Then we also checked the imbalance of our data and produced the labels' frequency.

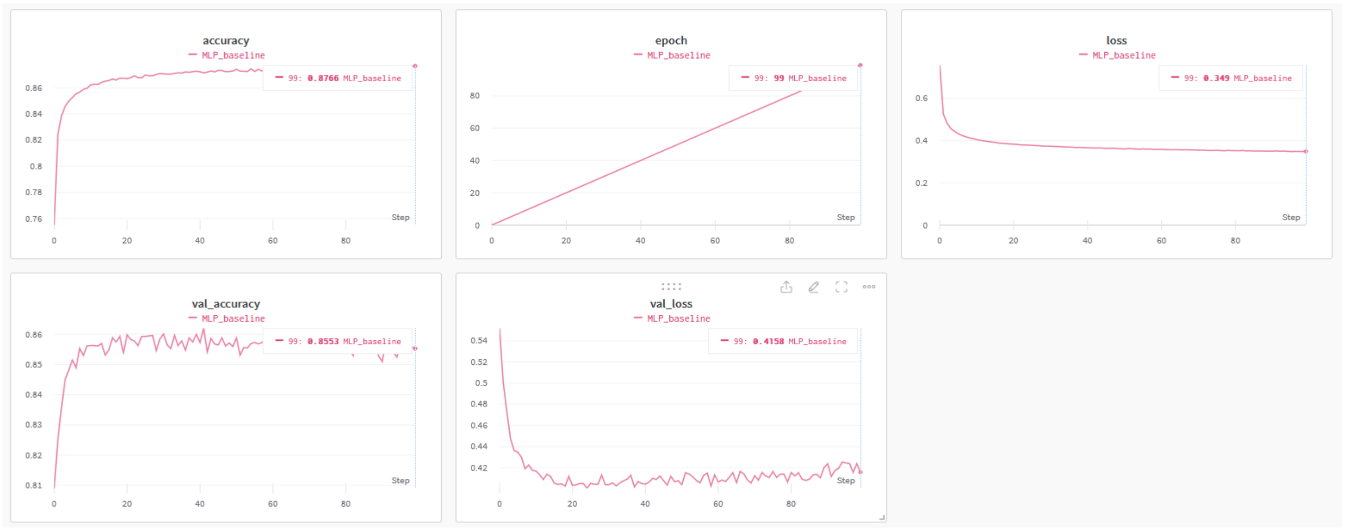


Multilayer Perceptron (MLP)

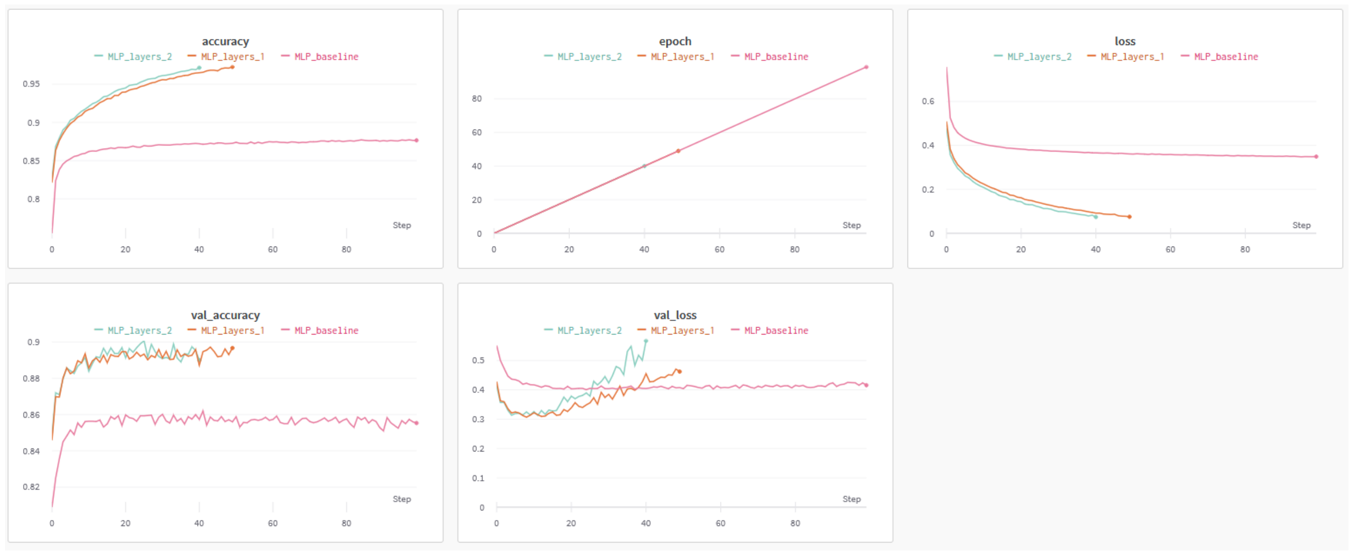
Baseline MLP model

In this part of the assignment we will evaluate the performance of an MLP architecture. We, firstly, built a Logistic Regression model, which served as a baseline for our process.

Furthermore with the use of Weights and Biases we managed to visualize the progress for any model implementation we applied. The implementation of our baseline model:



We configured the base model by adding 1 (with dimensions [256]) and 2 (with dimensions [512] [256]) hidden layers respectively to see its behavior. The results in comparison with the base model:



Multilayer Perceptron (MLP)

Model optimization

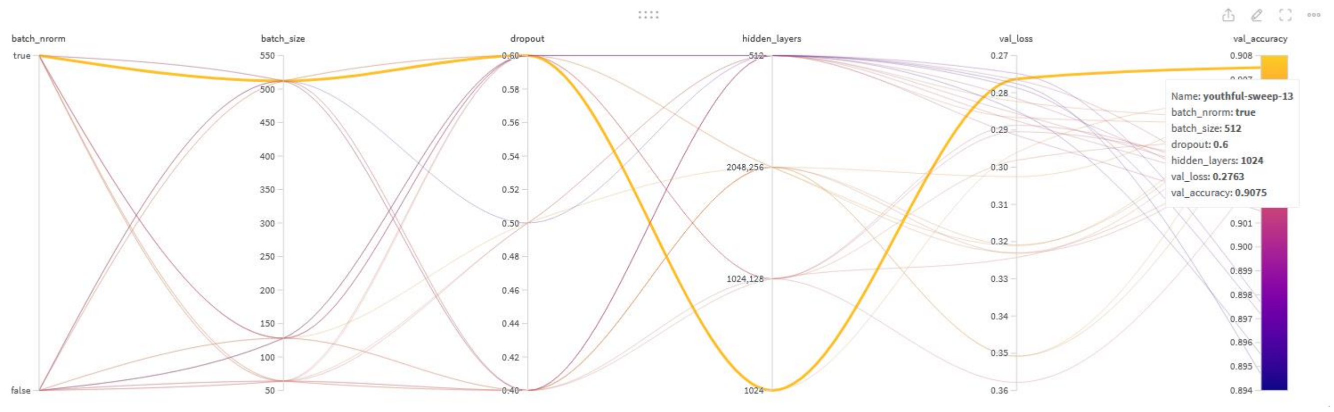
It looks that by adding a hidden layer the model suffers from overfitting. This can be seen by comparing the training loss (loss in Weights and Biases) to the validation loss (val_loss in Weights and Biases). Moreover, it seems that there isn't much of an advantage when adding two hidden layers, compared to one.

We tried to fix these issues, and improve the performance, by performing various optimizations, like:

- Using regularization techniques
- Introduce early stopping
- Add dropout and/or batch normalization
- Add data augmentation
- Reduce the size of the hidden layer
- Optimize hyper-parameters

Following on we applied some sweeps using the Weights and Biases. Actually our approach was to test different combinations of configurations that can be found in our python code, and by showcasing the results in Weights and Biases, to find and highlight the best model. Then we loaded the parameters of this model in our code and used it as final model.

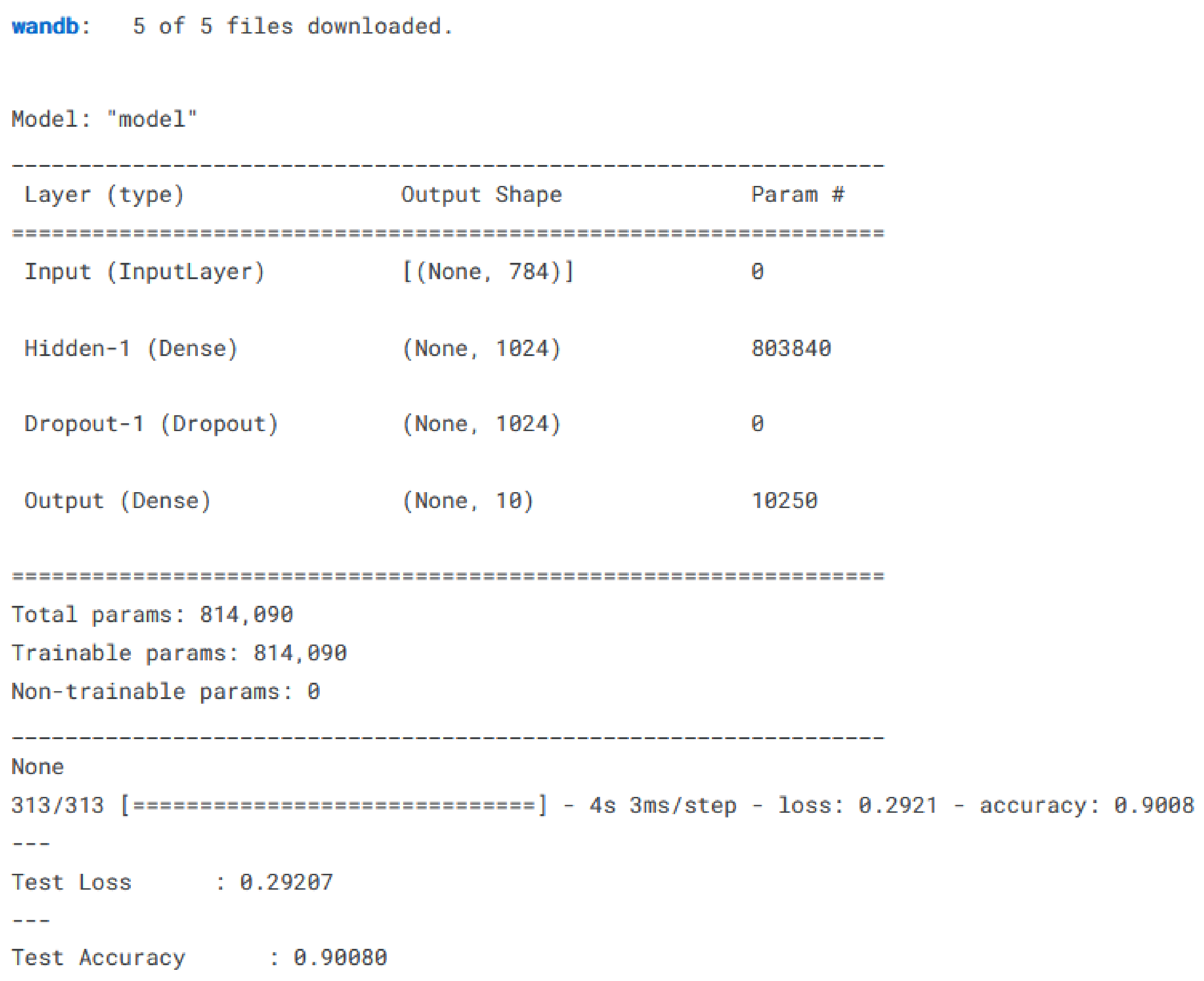
Below we can see, after the aforementioned implementations and having performed a large number of runs, we will stop further exploration, since the results don't seem to improve dramatically. the best configuration and its respective parameters that we eventually used for our MLP model:



Name: youthful-sweep-13
batch_nrorm: true
batch_size: 512
dropout: 0.6
hidden_layers: 1024
val_loss: 0.2763
val_accuracy: 0.9075

As can be seen, above, the best configuration is ***youthful-sweep-13***.

We then loaded this best model-configuration and evaluated the results of the model on the test data:

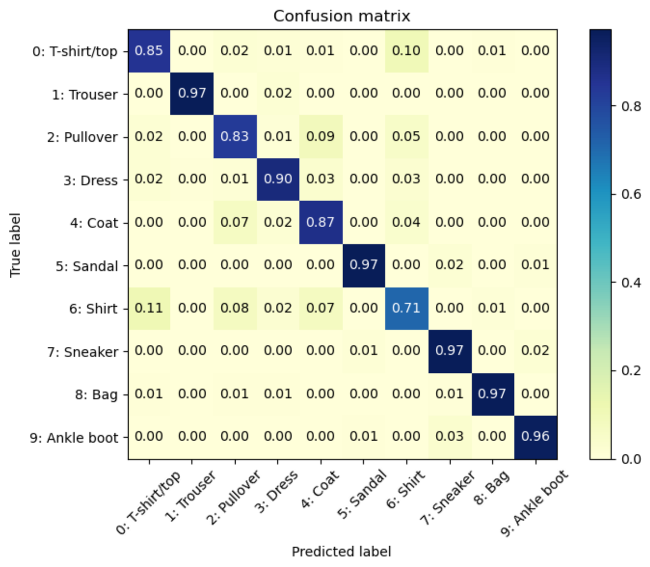


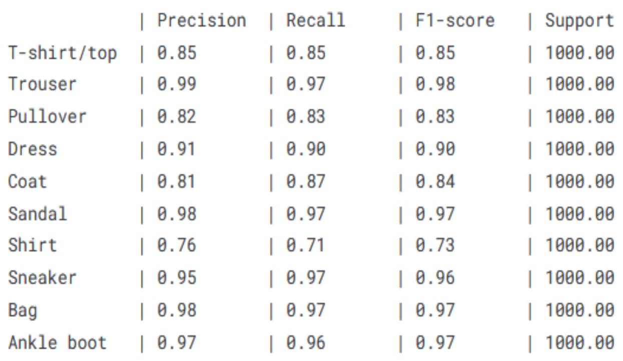
```
wandb:   5 of 5 files downloaded.

Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 Input (InputLayer)          [(None, 784)]             0

 Hidden-1 (Dense)            (None, 1024)              803840

 Dropout-1 (Dropout)         (None, 1024)              0

 Output (Dense)              (None, 10)                10250

=================================================================
Total params: 814,090
Trainable params: 814,090
Non-trainable params: 0
_________________________________________________________________
None
313/313 [==============================] - 4s 3ms/step - loss: 0.2921 - accuracy: 0.9008
---
Test Loss      : 0.29207
---
Test Accuracy      : 0.90080
```

We have reached to a satisfactory score after our optimization process. Next step was to evaluate the results per item as well. From the images provided, we expect some items to be difficult to distinguish.

Below we show the confusion matrix and the full classification report per item:

Results' evaluation

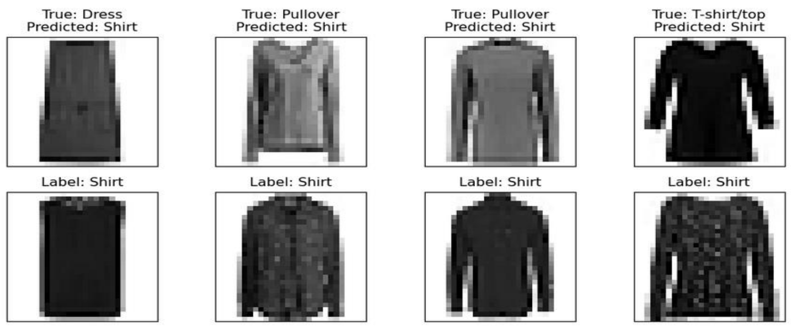



```
            | Precision  | Recall     | F1-score   | Support
T-shirt/top | 0.85       | 0.85       | 0.85       | 1000.00
Trouser     | 0.99       | 0.97       | 0.98       | 1000.00
Pullover    | 0.82       | 0.83       | 0.83       | 1000.00
Dress       | 0.91       | 0.90       | 0.90       | 1000.00
Coat        | 0.81       | 0.87       | 0.84       | 1000.00
Sandal      | 0.98       | 0.97       | 0.97       | 1000.00
Shirt       | 0.76       | 0.71       | 0.73       | 1000.00
Sneaker     | 0.95       | 0.97       | 0.96       | 1000.00
Bag         | 0.98       | 0.97       | 0.97       | 1000.00
Ankle boot  | 0.97       | 0.96       | 0.97       | 1000.00
```

Our intuition was right, and we see that some specific items are difficult to be matched correctly. This is why we proceeded in analyzing the incorrect classifications.

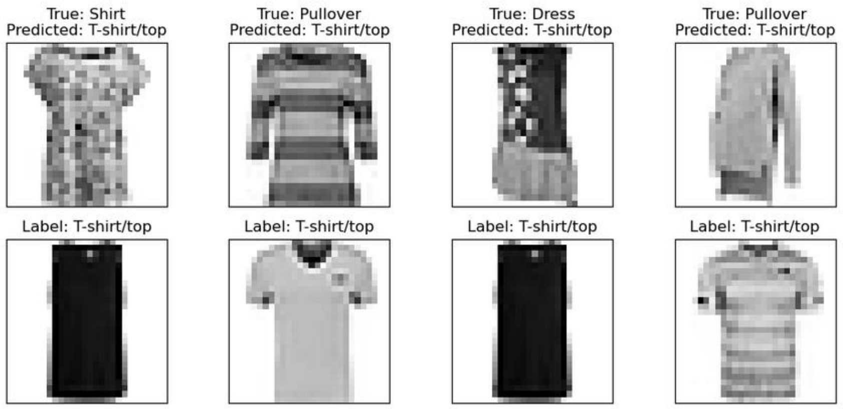
As seen from the confusion matrix, the errors happen most often on specific categories. We will explore these classses to get a better understanding of the behaviour of our model.

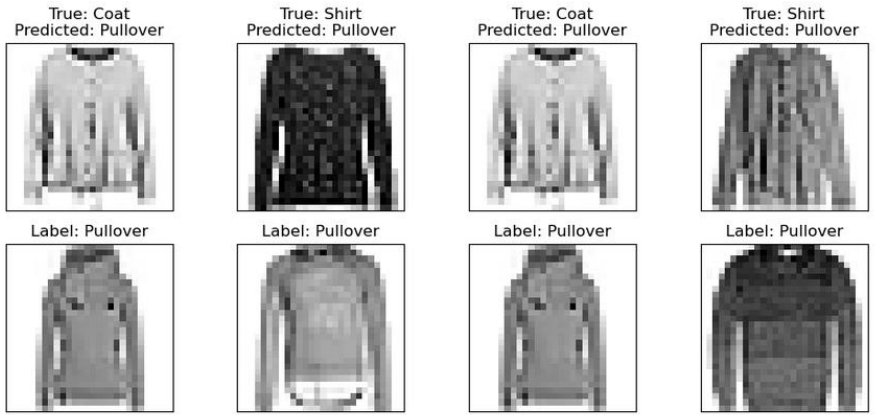
We created a plot containing 4 pictures of incorrect classification and 4 pictures of clothes corresponding to the predicted label. This will help us get a better understanding on the errors.



As seen from the confusion matrix, the Shirt category has the lowest recall and it is confused with the category T-Shirt/top, as well as the Pullover/Dress and coat. Looking at the plot above, we can see that this is expected, as many items of the top row (incorrect classifications) are similar to those of the second (pedicted label prictures).

Similarly, the T-shirt/top category is often confused with the Shirt and Pullovers are confused with Coats and Shirts, which can be expected, as seen from the images below.





Convolutional Neural Network full implementation

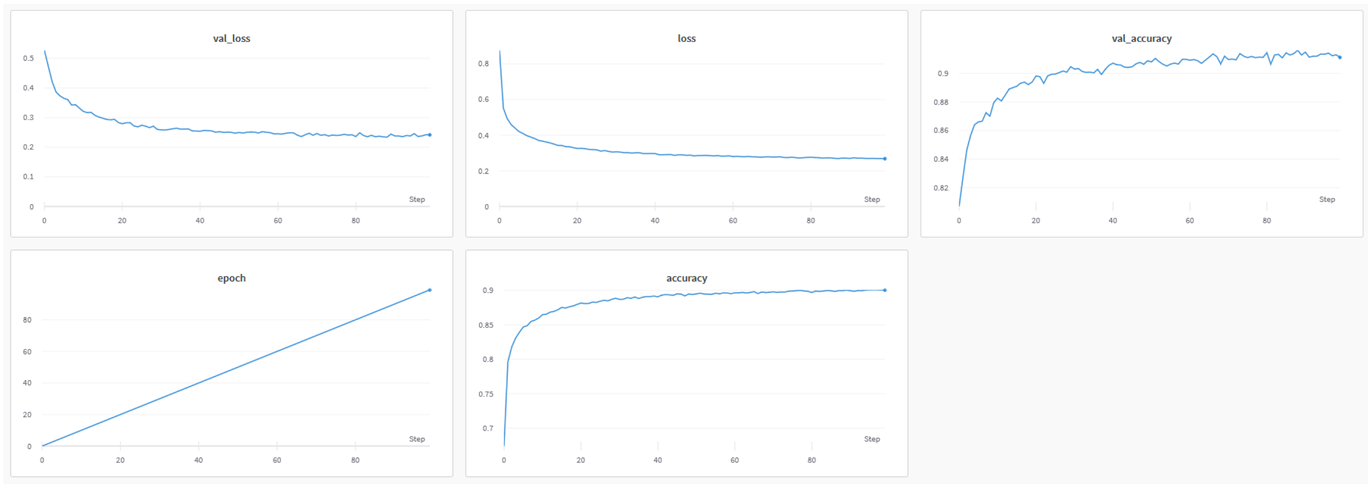
In this part of the assignment we evaluated the performance of a CNN architecture.

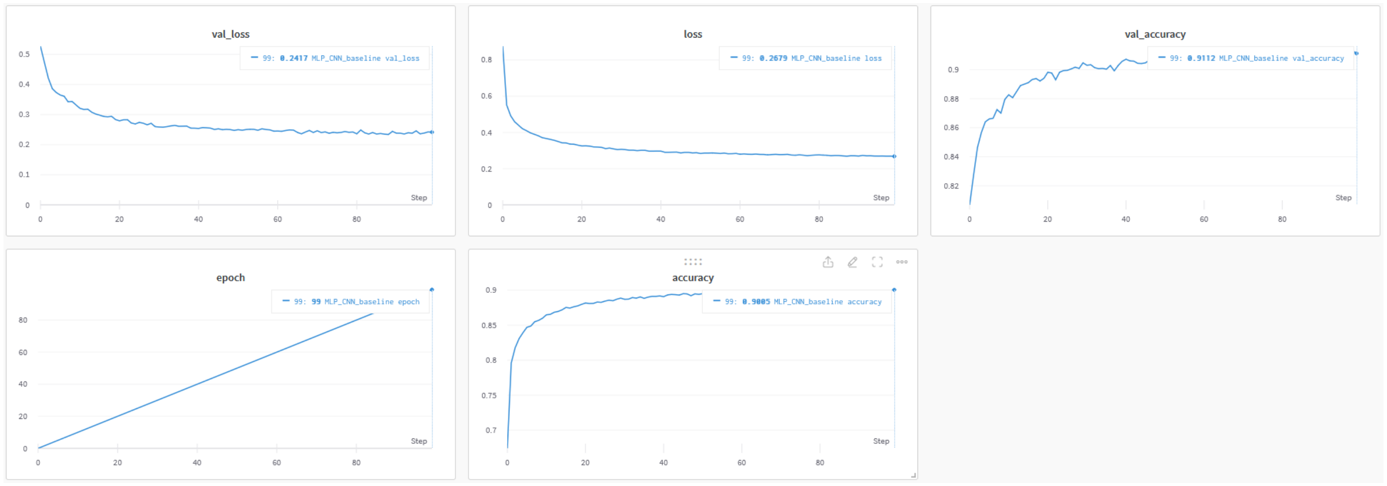
- We first built a base model with 3 conv_layers, which were used as a baseline
- Based on the results we got we created a sweep configuration to optimize hypterparameters

The base model with 3 conv_layers:

```
for i in range(conv_layers):
    x = Conv2D(
        filters=8*(2**i),
        kernel_size=(3, 3),
        strides=(1, 1),
        padding='same',
        dilation_rate=(1, 1),
        activation=conv_activation,
        name='Conv2D-{0:d}'.format(i + 1)
    )(x)
    x = MaxPool2D(
        pool_size=(2, 2),
        strides=(2, 2),
        padding='same',
        name='MaxPool2D-{0:d}'.format(i + 1)
    )(x)
    if dropout:
        x = Dropout(
            rate=0.2,
            name='Dropout-{0:d}'.format(i + 1)
        )(x)
# Flatten the convolved images so as to input them to a Dense Layer
x = Flatten(name='Flatten')(x)
```

The results for Train Loss, Validation Loss, Test Loss, Train Accuracy, Validation Accuracy and Test Accuracy can be visualized effectively using once again Weights and Biases:



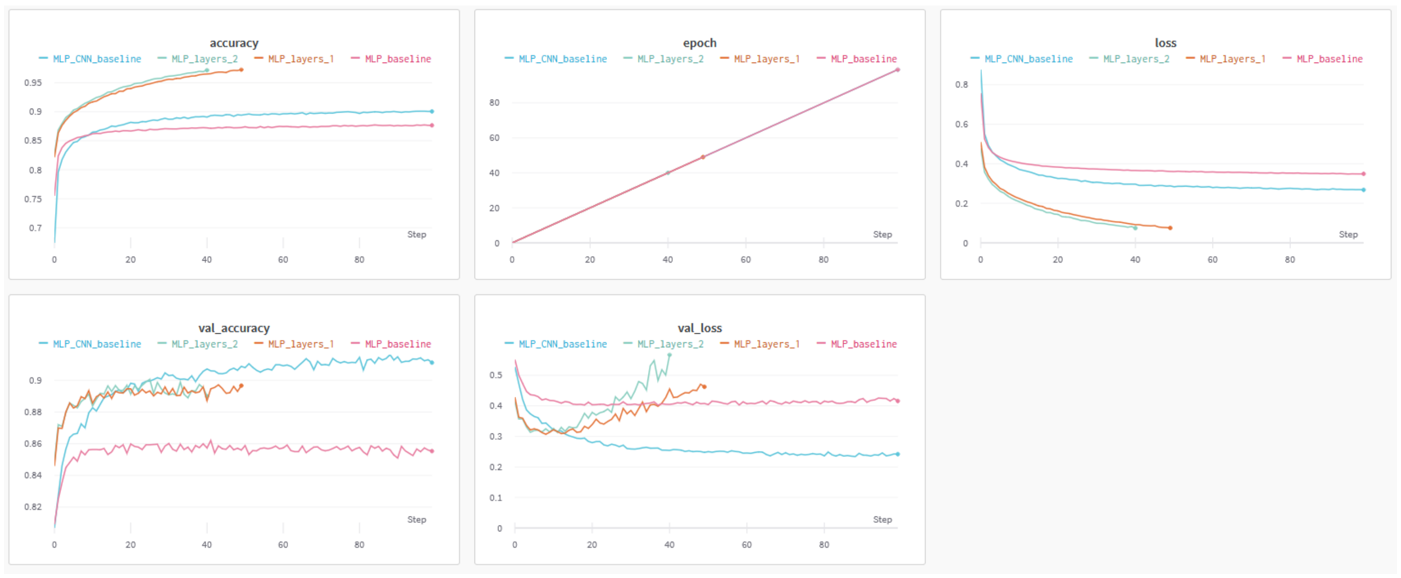


As we can see above, the model works surprisingly well, compared to the MLP one. We managed to match the accuracy of the fully optimized MLP model, with a relatively simple architecture. Nevertheless, we moved forward on fully optimizing the model.

We can also see, that the model reaches a plateau, whithout overfitting. This indicates that we should slightly increase the complexity of the model. Moreover, curves are not very smooth, thus we tried to decrease the learning rate. We built a new, more complex function, in order to create a new sweep which can be found in more detail in our code.

Convolutional Neural Network (CNN)

We can also see our baseline CNN model compared with the MLP baseline models:



We can also see our baseline CNN model compared with the MLP baseline models:For the implementation of the more complex model, mentioned before, we implemented a hyper parameter tuning with the help of Weights and Biases once more.

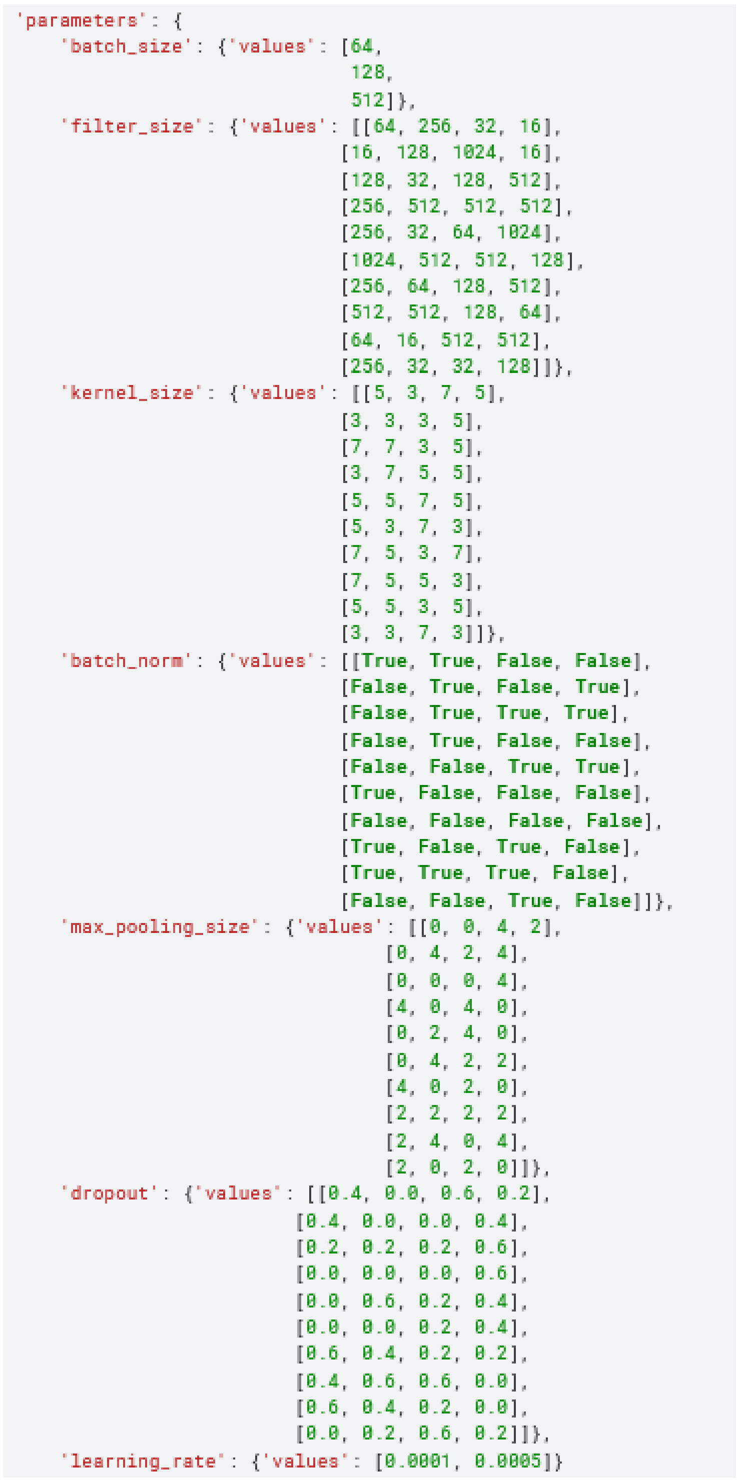
We built a a function named train_model which can build a model with the following blocks:

- Conv2D
- Batch Normalization
- Dropout
- 2D Max Polling

Each layer was built based on a list input, and depending on the value the Batch Normalization, Dropout and max polling layers could be skipped.

We continued with the step of generating random combinations of inputs. For each parameter we created a pool of inputs and generated random lists (of length 4) which were used as an input in the Weights and Biases sweep configuration.

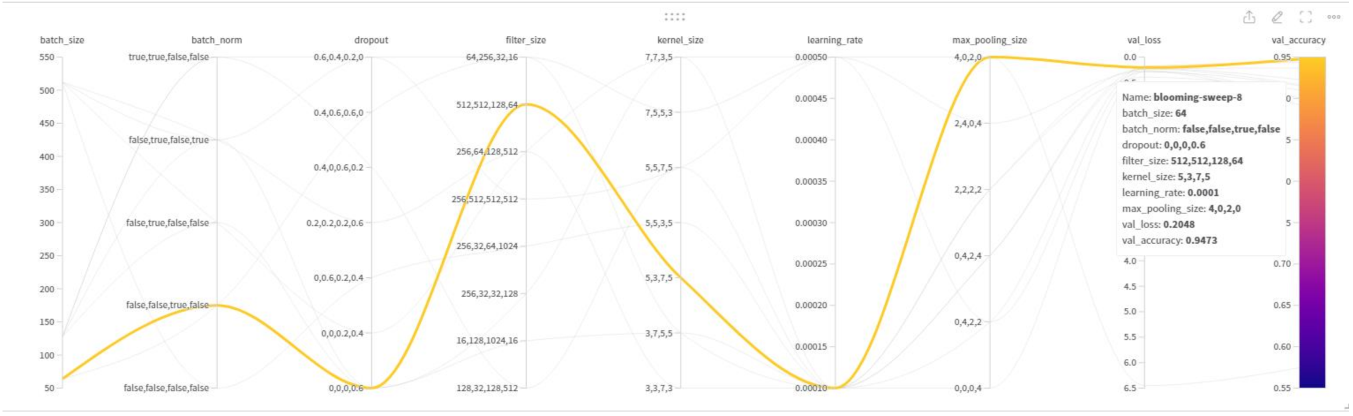
Below you can see the the lists of parameters created for the configuration:



```
'parameters': {
    'batch_size': {'values': [64,
                              128,
                              512]},
    'filter_size': {'values': [[64, 256, 32, 16],
                             [16, 128, 1024, 16],
                             [128, 32, 128, 512],
                             [256, 512, 512, 512],
                             [256, 32, 64, 1024],
                             [1024, 512, 512, 128],
                             [256, 64, 128, 512],
                             [512, 512, 128, 64],
                             [64, 16, 512, 512],
                             [256, 32, 32, 128]]},
    'kernel_size': {'values': [[5, 3, 7, 5],
                             [3, 3, 3, 5],
                             [7, 7, 3, 5],
                             [3, 7, 5, 5],
                             [5, 5, 7, 5],
                             [5, 3, 7, 3],
                             [7, 5, 3, 7],
                             [7, 5, 5, 3],
                             [5, 5, 3, 5],
                             [3, 3, 7, 3]]},
    'batch_norm': {'values': [[True, True, False, False],
                             [False, True, False, True],
                             [False, True, True, True],
                             [False, True, False, False],
                             [False, False, True, True],
                             [True, False, False, False],
                             [False, False, False, False],
                             [True, False, True, False],
                             [True, True, True, False],
                             [False, False, True, False]]},
    'max_pooling_size': {'values': [[0, 0, 4, 2],
                                 [0, 4, 2, 4],
                                 [0, 0, 0, 4],
                                 [4, 0, 4, 0],
                                 [0, 2, 4, 0],
                                 [0, 4, 2, 2],
                                 [4, 0, 2, 0],
                                 [2, 2, 2, 2],
                                 [2, 4, 0, 4],
                                 [2, 0, 2, 0]]},
    'dropout': {'values': [[0.4, 0.0, 0.6, 0.2],
                         [0.4, 0.0, 0.0, 0.4],
                         [0.2, 0.2, 0.2, 0.6],
                         [0.0, 0.0, 0.0, 0.6],
                         [0.0, 0.6, 0.2, 0.4],
                         [0.0, 0.0, 0.2, 0.4],
                         [0.6, 0.4, 0.2, 0.2],
                         [0.4, 0.6, 0.6, 0.0],
                         [0.6, 0.4, 0.2, 0.0],
                         [0.0, 0.2, 0.6, 0.2]]},
    'learning_rate': {'values': [0.0001, 0.0005]}
```

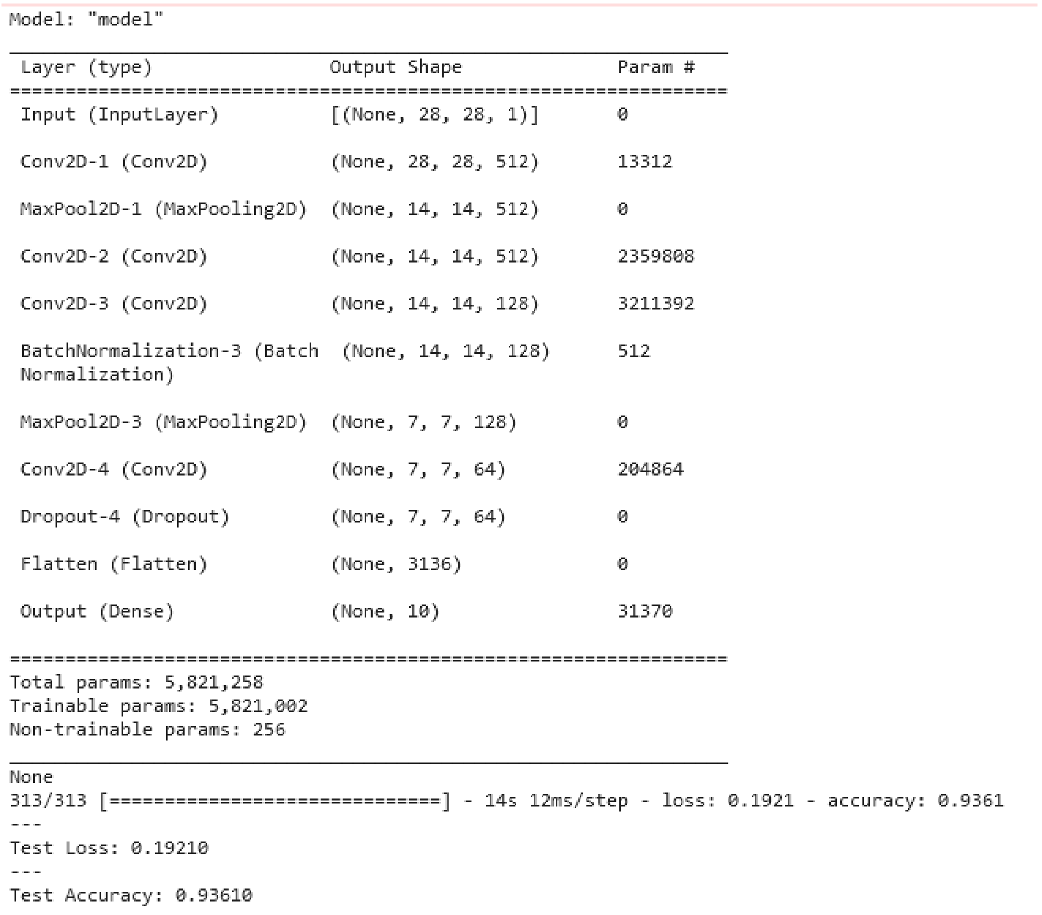
Convolutional Neural Network (CNN)

Having performed a large number of runs, we stopped further exploration, since the results did not seem to improve dramatically. Below you see the best parameter configuration for our model:



Name: **blooming-sweep-8**
batch_size: **64**
batch_norm: **false,false,true,false**
dropout: **0,0,0,0.6**
filter_size: **512,512,128,64**
kernel_size: **5,3,7,5**
learning_rate: **0.0001**
max_pooling_size: **4,0,2,0**
val_loss: **0.2048**
val_accuracy: **0.9473**

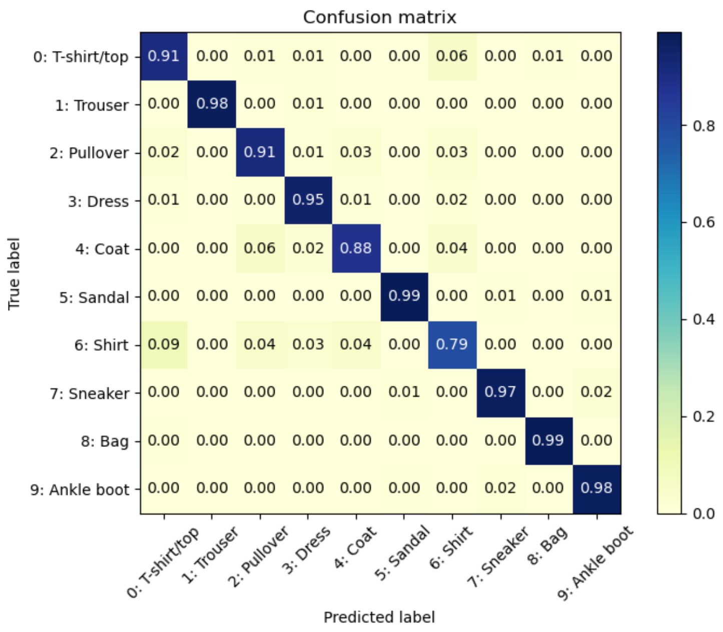
And running the predictions on the test data:



```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 Input (InputLayer)          [(None, 28, 28, 1)]       0

 Conv2D-1 (Conv2D)           (None, 28, 28, 512)       13312

 MaxPool2D-1 (MaxPooling2D)  (None, 14, 14, 512)       0

 Conv2D-2 (Conv2D)           (None, 14, 14, 512)       2359808

 Conv2D-3 (Conv2D)           (None, 14, 14, 128)       3211392

 BatchNormalization-3 (Batch  (None, 14, 14, 128)      512
 Normalization)

 MaxPool2D-3 (MaxPooling2D)  (None, 7, 7, 128)         0

 Conv2D-4 (Conv2D)           (None, 7, 7, 64)          204864

 Dropout-4 (Dropout)         (None, 7, 7, 64)          0

 Flatten (Flatten)           (None, 3136)              0

 Output (Dense)              (None, 10)                31370

=================================================================
Total params: 5,821,258
Trainable params: 5,821,002
Non-trainable params: 256
_________________________________________________________________
None
313/313 [==============================] - 14s 12ms/step - loss: 0.1921 - accuracy: 0.9361
---
Test Loss: 0.19210
---
Test Accuracy: 0.93610
```

Convolutional Neural Network (CNN)

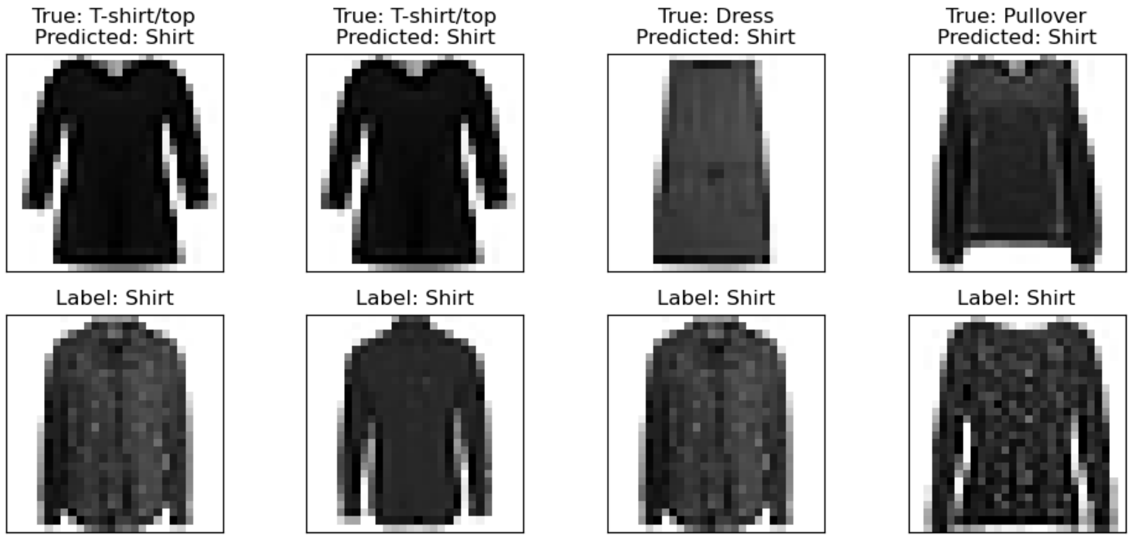
Below we see the confusion matrix of our CNN and the full classification report per item:

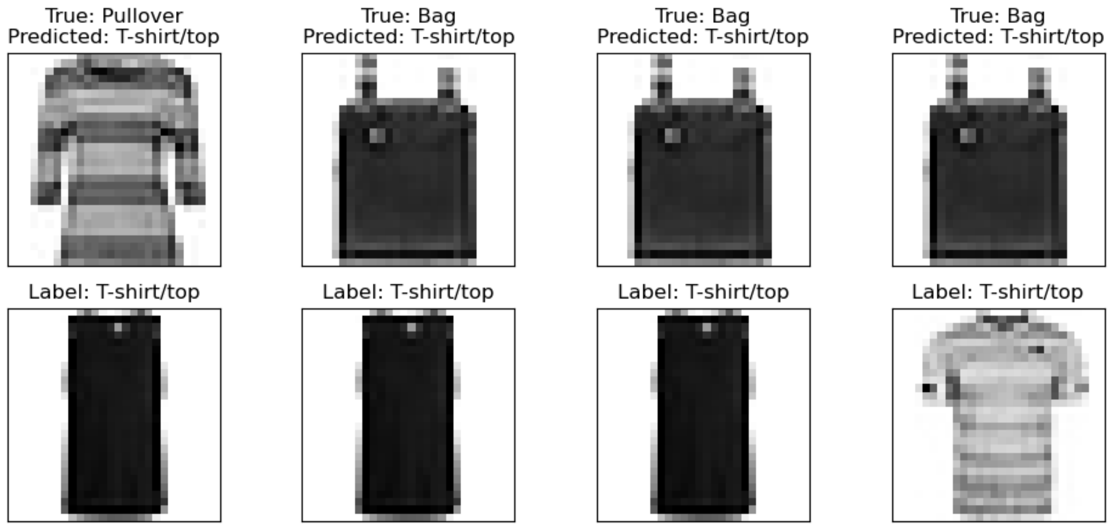


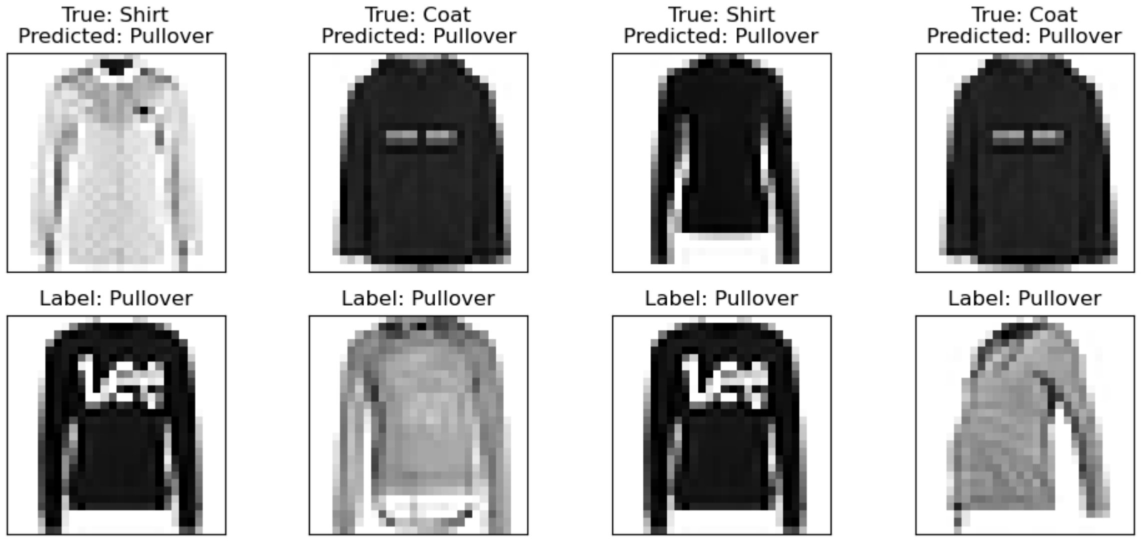
```
             | Precision  | Recall     | F1-score   | Support
T-shirt/top  | 0.87       | 0.91       | 0.89       | 1000.00
Trouser      | 1.00       | 0.98       | 0.99       | 1000.00
Pullover     | 0.88       | 0.91       | 0.90       | 1000.00
Dress        | 0.92       | 0.95       | 0.94       | 1000.00
Coat         | 0.91       | 0.88       | 0.90       | 1000.00
Sandal       | 0.99       | 0.99       | 0.99       | 1000.00
Shirt        | 0.85       | 0.79       | 0.82       | 1000.00
Sneaker      | 0.97       | 0.97       | 0.97       | 1000.00
Bag          | 0.99       | 0.99       | 0.99       | 1000.00
Ankle boot   | 0.97       | 0.98       | 0.98       | 1000.00
```

Even though our model works pretty well we see (as in our MLP implementation), that some items probably are not correctly recognized by the CNN.

So below we see the incorrectly classified images in order to confirm the difficulties our model faced.




True: Pullover
Predicted: T-shirt/top
True: Bag
Predicted: T-shirt/top
True: Bag
Predicted: T-shirt/top
True: Bag
Predicted: T-shirt/top
Label: T-shirt/top
Label: T-shirt/top
Label: T-shirt/top
Label: T-shirt/top


True: Shirt
Predicted: Pullover
True: Coat
Predicted: Pullover
True: Shirt
Predicted: Pullover
True: Coat
Predicted: Pullover
Label: Pullover
Label: Pullover
Label: Pullover
Label: Pullover