Queens' University

Deep Learning

CISC 867 Project_2: Fashion-MNIST

Supervised by:

Prof. Hazem Abbas

T.A. Ahmed Nagib

Team members:

Sara Ahmed Mohamed Elfetiany

Mahmoud Mohamed Abdelhamied Abdin

Mostafa Mohamed Abdelhamid Azazi

CONTENTS

1. Introduction

Fashion classification encompasses the identification of clothing items in an image. Clothing in many cultures reflects characteristics such as age, social status, lifestyle and gender. Apparel is also an important descriptor in identifying humans, e.g. ”the man wearing an orange jacket” or ”the woman in red high heels.” Given the role of clothing apparel in society, ”fashion classification” has many applications.

In order to solve this issue, we will use the Fashion MNIST dataset from tensorflow.keras.datasets and apply different deep learning models on it as well as some pre-trained models using Transfer Learning

So, Our objectives represent in accurately identify the 10 categories of clothes that the dataset contains using the different neural network architectures

2. Data Description

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.

Each image is 28 pixels in height and 28 pixels in width, for a total of 784 pixels in total. Each pixel has a single pixel-value associated with it, indicating the lightness or darkness of that pixel, with higher numbers meaning darker. This pixel-value is an integer between 0 and 255.

Each training and test example is assigned to one of the following labels:

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

3. Data Preprocessing

3.1. Check the missing data

The first step is to check if there is any missing data in the dataset to clean it and replace it with appropriate value.
And by checking our dataset we didn't find any missing data.

3.2. Encoding the data

The only column that contains categorical data and needs to be encoded is our target and we encode it using '`tf.keras.utils.to_categorical`' that converts a class vector (integers) to a binary class matrix.

3.3. Normalization

Normalize the pixel values of the train and test images to be between 0 and 1 instead of being between 0 and 255

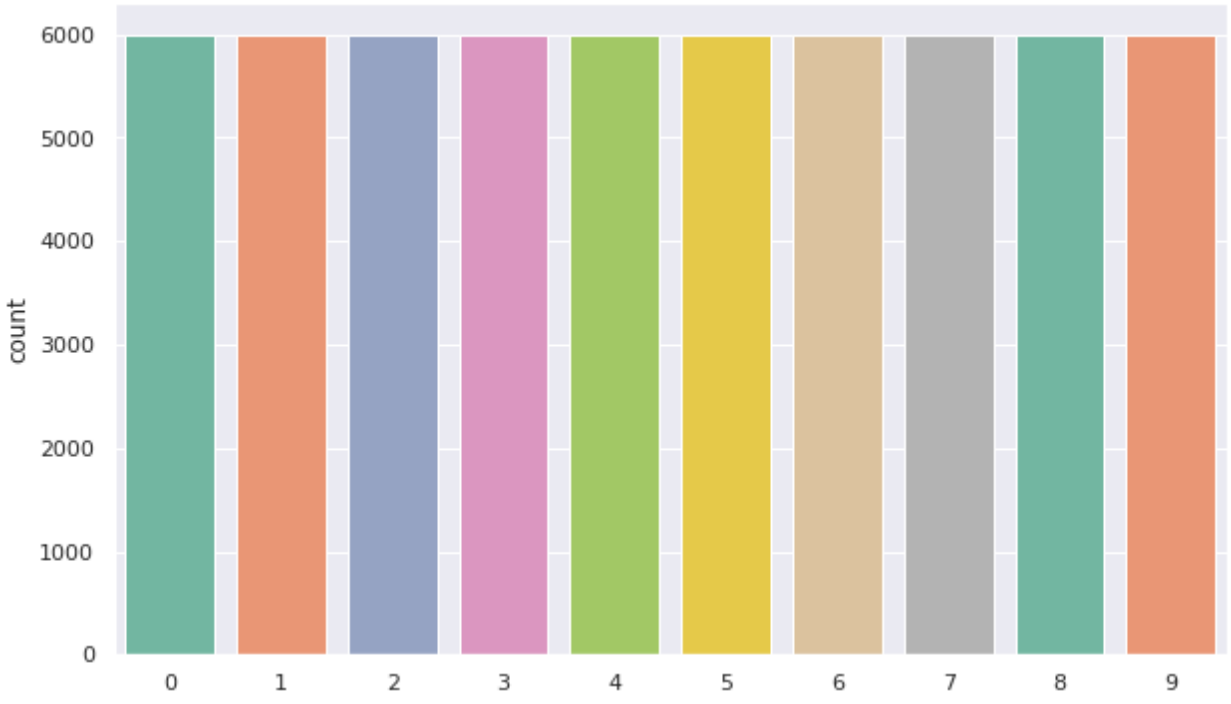
3.4. Reshaping

Reshaping the train and test images as per the tensor format required by tensorflow (28, 28, 1) instead of (28, 28)
The shape (28, 28, 1) represents the dimension of the image and the number of channels. We specify the number of channels equal to one as our dataset contains grayscale images. In case of RGB images the number of channels will be three

4. Data visualization

4.1. Classes count

Plotting the count of each category and we can see that the dataset is balanced as the data contain the same number of data points for each target category.



4.2. Visualizing images

4.2.1. IMAGE PIXEL ARRAY

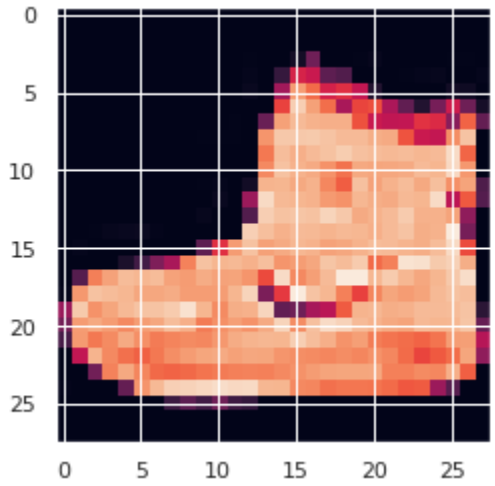
Print the image pixel array of the first image which has label 9 that represent Ankle boot

```
LABEL: 9

IMAGE PIXEL ARRAY:
 [[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   1   0   0  13  73   0   0   1   4   0   0   0   0   1   1   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   3   0  36 136 127  62  54   0   0   0   1   3   4   0   0   3]
 [  0   0   0   0   0   0   0   0   0   0   0   0   6   0 102 204 176 134 144 123  23   0   0   0   0  12  10   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0 155 236 207 178 107 156 161 109  64  23  77 130  72  15]
 [  0   0   0   0   0   0   0   0   0   0   0   1   0  69 207 223 218 216 216 163 127 121 122 146 141  88 172  66]
 [  0   0   0   0   0   0   0   0   0   0   1   1   1   0 200 232 232 233 229 223 223 215 213 164 127 123 196 229   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0 183 225 216 223 228 235 227 224 222 224 221 223 245 173   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0 193 228 218 213 198 180 212 210 211 213 223 220 243 202   0]
 [  0   0   0   0   0   0   0   0   0   0   1   3   0  12 219 220 212 218 192 169 227 208 218 224 212 226 197 209  52]
 [  0   0   0   0   0   0   0   0   0   0   6   0   0  99 244 222 220 218 203 198 221 215 213 222 220 245 119 167  56]
 [  0   0   0   0   0   0   0   0   0   4   0   0  55 236 228 230 228 240 232 213 218 223 234 217 217 209  92   0]
 [  0   0   1   4   6   7   2   0   0   0   0   0 237 226 217 223 222 219 222 221 216 223 229 215 218 255  77   0]
 [  0   3   0   0   0   0   0   0   0  62 145 204 228 207 213 221 218 208 211 218 224 223 219 215 224 244 159   0]
 [  0   0   0   0  18  44  82 107 189 228 220 222 217 226 200 205 211 230 224 234 176 188 250 248 233 238 215   0]
 [  0  57 187 208 224 221 224 208 204 214 208 209 200 159 245 193 206 223 255 255 221 234 221 211 220 232 246   0]
 [  3 202 228 224 221 211 211 214 205 205 205 220 240  80 150 255 229 221 188 154 191 210 204 209 222 228 225   0]
 [ 98 233 198 210 222 229 229 234 249 220 194 215 217 241  65  73 106 117 168 219 221 215 217 223 223 224 229  29]
 [ 75 204 212 204 193 205 211 225 216 185 197 206 198 213 240 195 227 245 239 223 218 212 209 222 220 221 230  67]
 [ 48 203 183 194 213 197 185 190 194 192 202 214 219 221 220 236 225 216 199 206 186 181 177 172 181 205 206 115]
 [  0 122 219 193 179 171 183 196 204 210 213 207 211 210 200 196 194 191 195 191 198 192 176 156 167 177 210  92]
 [  0   0  74 189 212 191 175 172 175 181 185 188 189 188 193 198 204 209 210 210 211 188 188 194 192 216 170   0]
 [  2   0   0   0  66 200 222 237 239 242 246 243 244 221 220 193 191 179 182 182 181 176 166 168  99  58   0   0]
 [  0   0   0   0   0   0   0  40  61  44  72  41  35   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]]
```

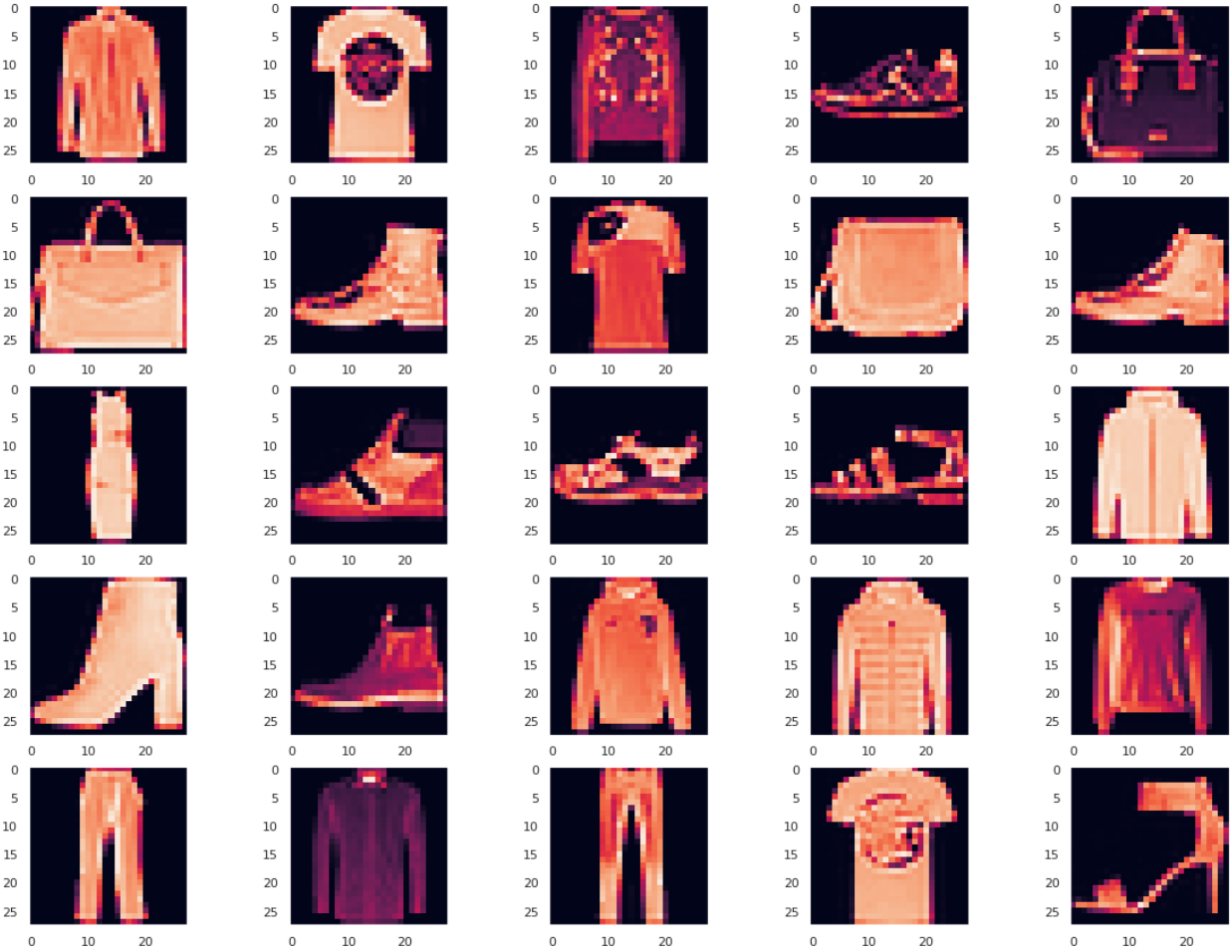
4.2.2. Plotting image

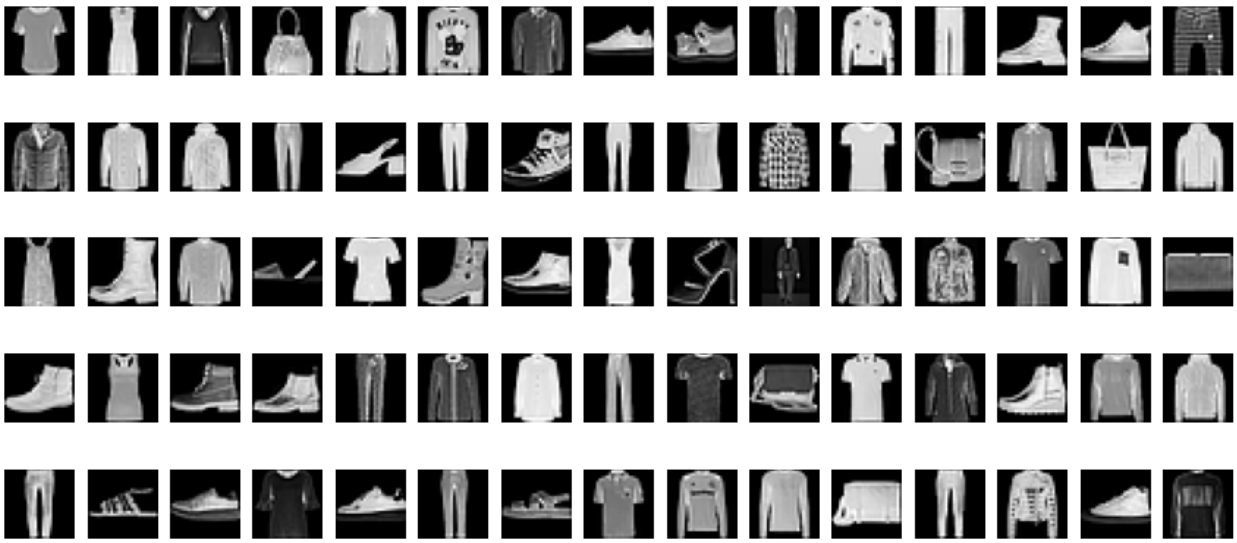
Plot the previous image using plt.imshow()



4.3. Random sample from the data

Plotting some random image from our dataset using two different ways



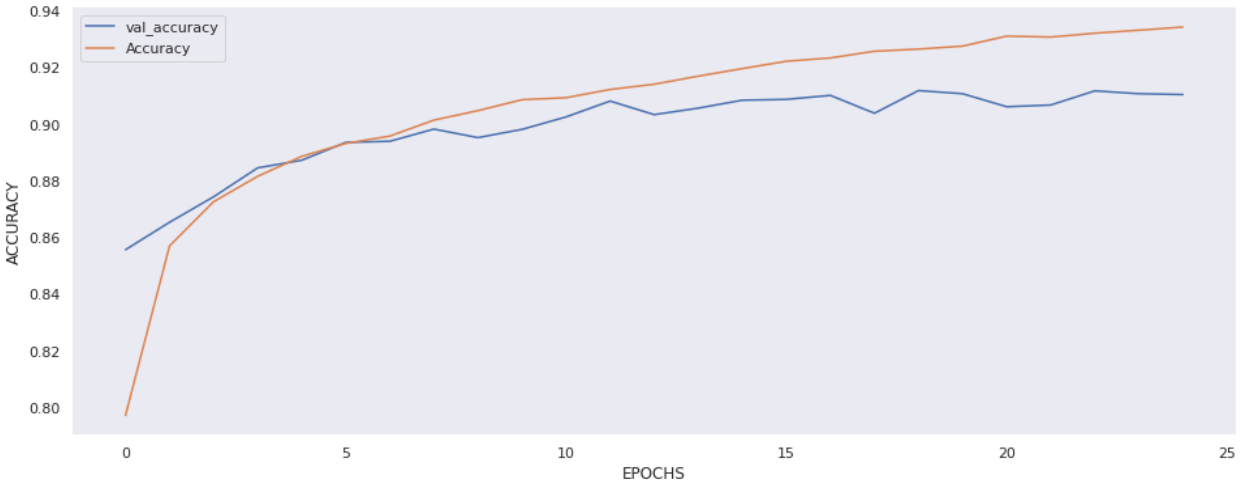


5. Modeling

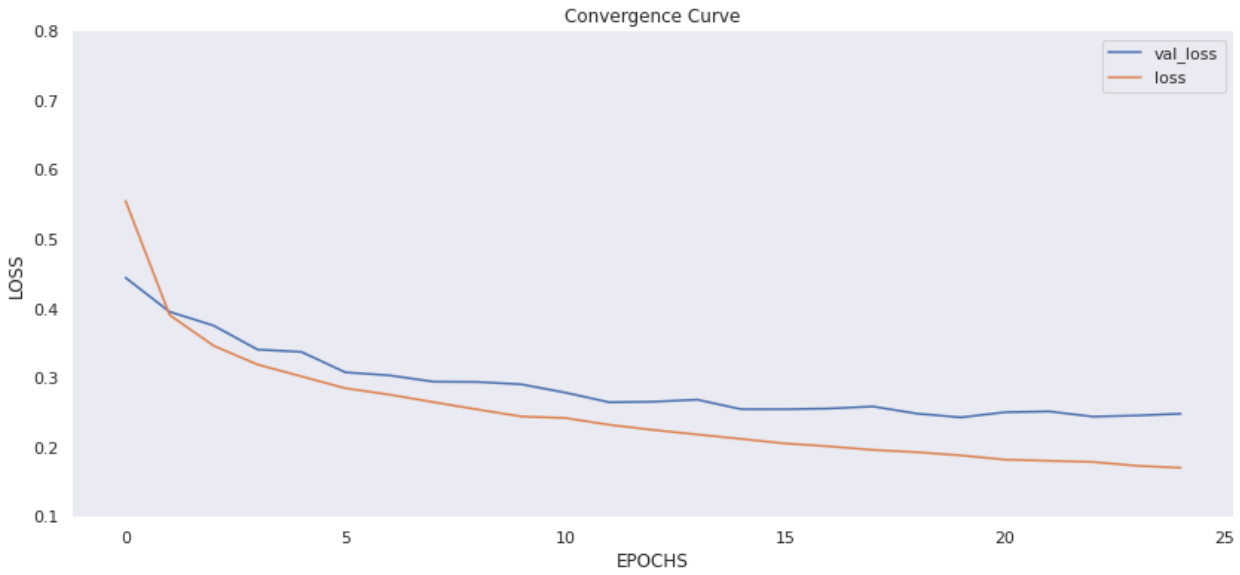
5.1. First trial: Building LeNet-5 network

For the first trial we implement the LeNet-5 network which contain Conv2D layer followed by MaxPooling2D layer and then another Conv2D layer with dropout followed by MaxPooling2D layer and finally there is Conv2D layer followed by two Dense layers
We used for train this simple network 60 epochs with EarlyStopping callback that monitor the validation loss with patience value equal to 5

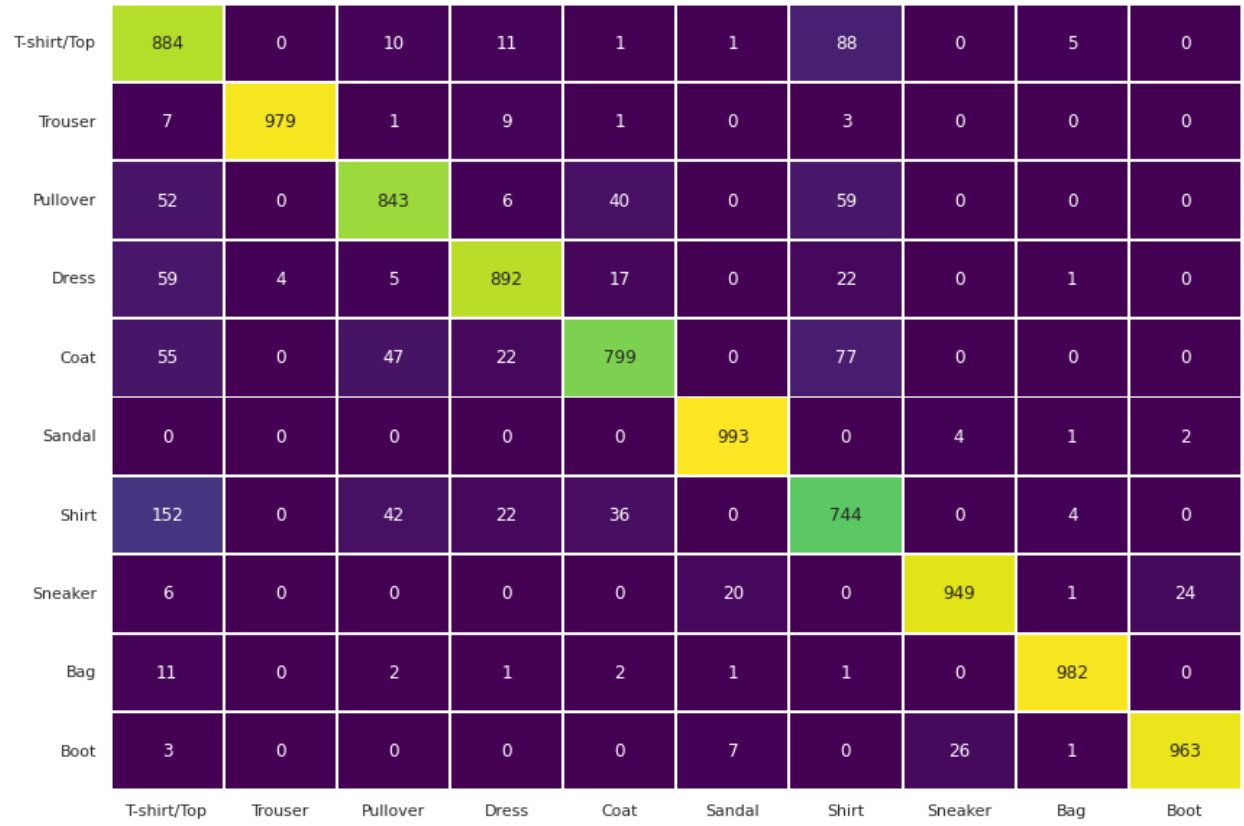
To evaluate this network, we use 5-Fold cross-validation which gives us 90.9% accuracy as shown in the following figure



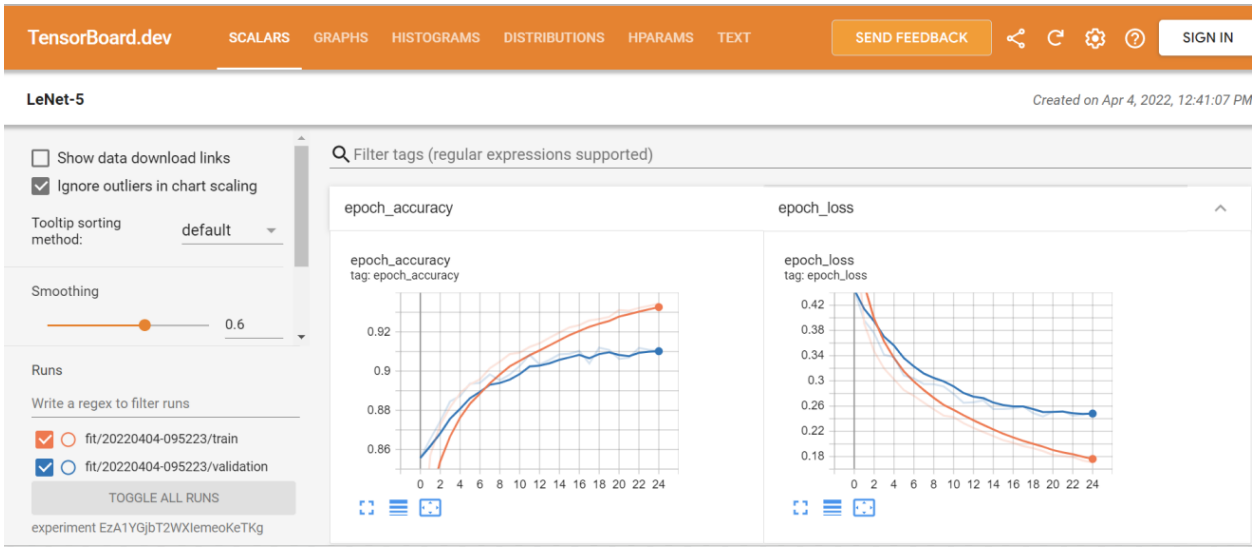
And the `Convergence Curve` of this trial represented in this figure



And here is the `confusion_matrix` of this trial



tensorboard of trial 1 :
https://tensorboard.dev/experiment/EzA1YGjbT2WXIemeoKeTKg/

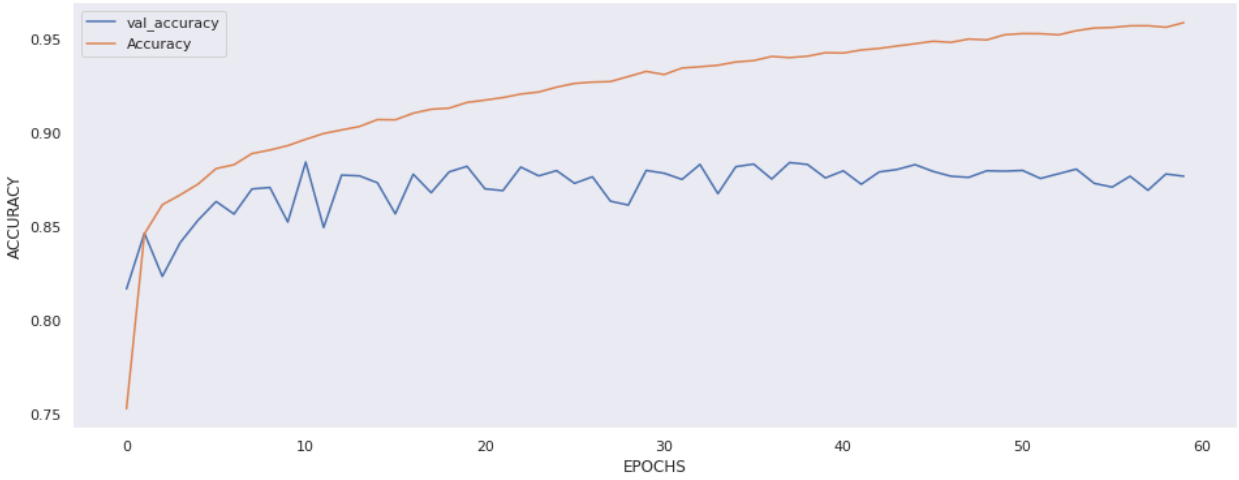


5.2. Second trial: Building AlexNet network

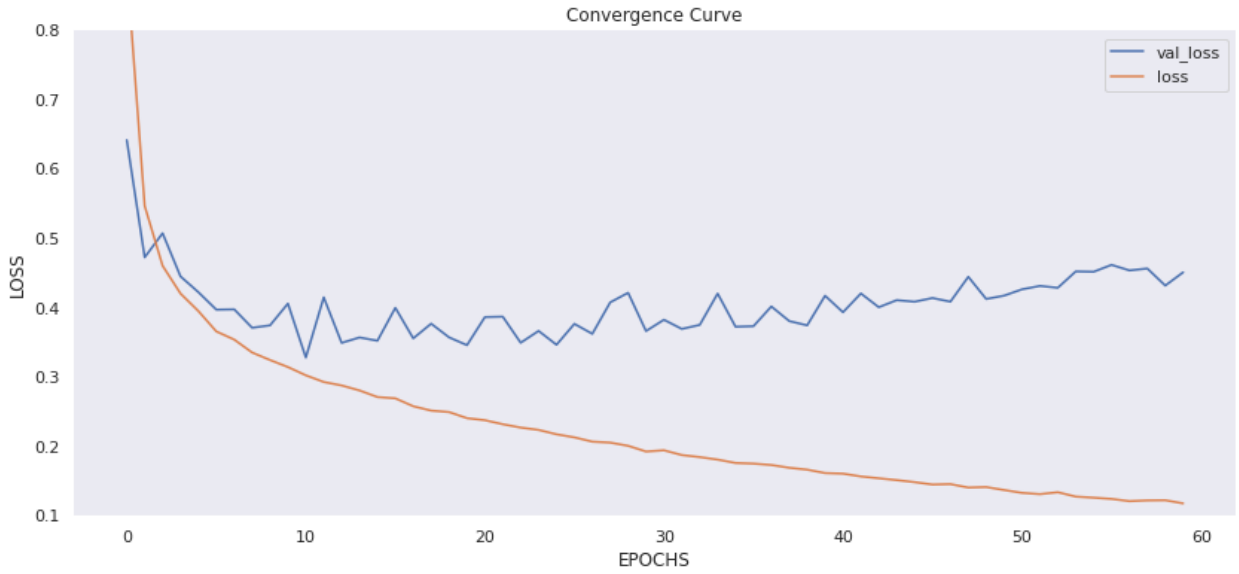
For the second trial we implement another network architecture which is the AlexNet model, to compare it with built LeNet-5 network

Alexnet architecture is used to extract features for each sample in the dataset. The architecture of AlexNet contains 60,000 total parameters within 8 total layers: five convolutional layers, and three fully connected layers.
We used for train this network 60 epochs with EarlyStopping callback that monitor the validation loss with patience value equal to 5

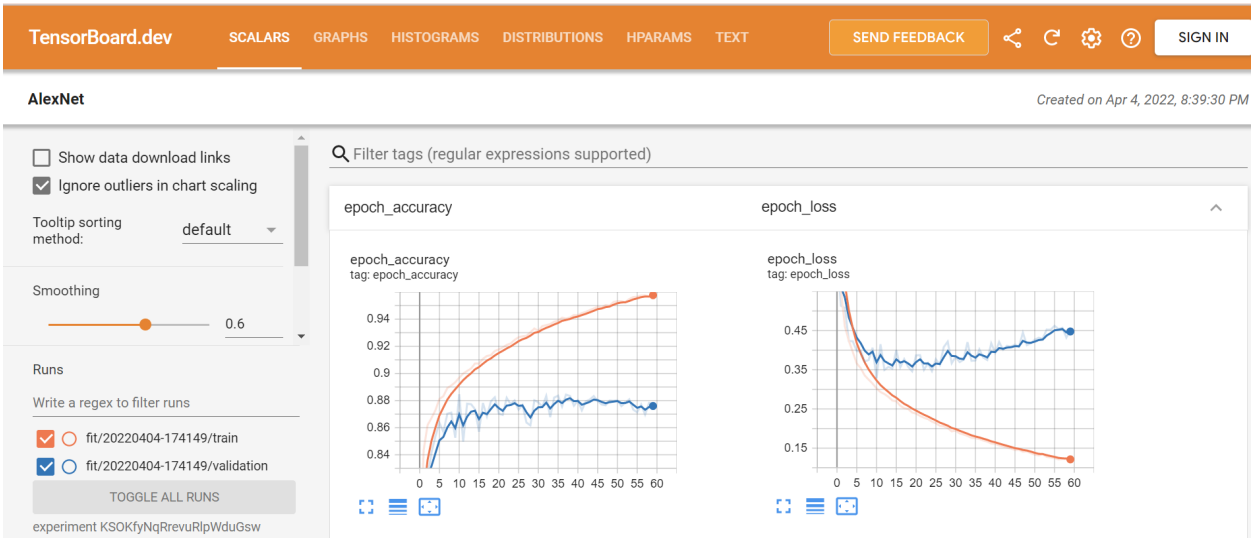
By using the 5-Fold cross-validation to measure the model accuracy we get 87.94% as shown in the following figure



Training loss vs testing loss `'Convergence Curve'`



Tensorboard of trial 2 :
https://tensorboard.dev/experiment/KSOKfyNqRrevuRlpWduGsw/



5.3. Third trial: Modifying hyperparameters for (LeNet-5)

To get the best performance the model hyperparameters will be tuned.
we will tune the following hyperparameters:

- activation function
- weight initialization
- kernel size
- number of filters per layer
- optimizer
- learning rate
- dropout
- batch normalization

Before we start it's important to mention that the fashion mnist validation accuracy of LeNet model without modification is 0.8669999837875366 with (tanh as activation)

5.3.1. Tuning

5.3.1.1. Activation and weight initialization

We tried 3 different activation functions relu , tanh and LeakyReLU with alpha 0.1 Along with 4 different weight initialization ways he_uniform , he_normal, glorot_normal and glorot_uniform.

Keras tuner is used in a way that the search space covers all possible combinations.

The best validation accuracy achieved was : 0.9157500267028809
With relu as activation function and glorot_uniform as kernel_initializer.
And nearly 5% improvement from the original model.

View TensorBoard of this step at
https://tensorboard.dev/experiment/eTQt6aO0QEmwbSak2WHreA/

5.3.1.2. Kernel size

We only tried 2 different kinds of kernels (3x3 and 5x5)
5x5 is the default value in lenet original model.

The best validation accuracy achieved was: 0.9181666374206543
With kernel_size: 5 compared to 0.9127500057220459 with kernel = 3
The difference is small so we will use a kernel of size 5x5.
View your TensorBoard at
https://tensorboard.dev/experiment/WDGVyUmfQGWGROPZu3niDQ/

5.3.1.3. Batch size

We tried a range of batch sizes that is defined as min_value = 64 max_value 256 step=32 (max value is included) . We didn't include batch size of 32 as it was the default value for the previous trial.

We ran the keras search and the results as follows:
The best score is: 0.918666660785675 which is nearly the same as a batch of size 32. So we continue with the batch of size 32.

View your TensorBoard at
https://tensorboard.dev/experiment/9yvKH7TmRUGC7riAkwysAA/

5.3.1.4. number of filters

In the LeNet-5 model there are 3 convolution layers for each one we tried different values from 2^2 to 2^8 the step is in power of 2.

Running more that 160 trials we go this result:
The best validation accuracy achieved was: 0.9190000295639038
Which is 0.1 percent higher than previous trial and the number of filters for each layer was as follows:

- conv1 2^7
- conv2 2^7
- conv3 2^5

View your TensorBoard at
https://tensorboard.dev/experiment/iop4UWUGTo6DmxXXZQbzNg/

5.3.1.5. optimizers and learning rate

For optimizers, 4 different optimizers were used with 3 different learning rates
Optimizers: ['Adagrad','Adadelta','RMSprop','Adam']
Learning rates : [1e-1, 1e-2, 1e-4]

Keras tuner tried every combination and the results as follows:
The best validation accuracy achieved was: 0.9164166450500488 with optimizer: Adagrad and learning_rate: 0.1, but this accuracy is lower than the previous trials in which we used adam optimizer with learning rate of 0.001. So, the chosen optimizer is Adam

View your TensorBoard at
https://tensorboard.dev/experiment/mNy1Pm4lQQyUmpynu4ZvuA/

5.3.1.6. Dropout

We tried to add SpatialDropout after each convolution layer with different rates [0.2,0.3,0.4,0.5]

After trying all rate value the best result was 0.9211666584014893 with dropout of 0.2
There is improvement compared to the last accuracy achieved.

View your TensorBoard at
https://tensorboard.dev/experiment/NyJMtwubTQGaI3marVLs6g/

5.3.1.7. batch norm

Batch normalization layer is added before the input of each layers in lenet model

After training the model achieved the following result:
loss: 0.1472 - accuracy: 0.9465 - val_loss: 0.2174 - val_accuracy: 0.9243
Which is higher than the previous result by 0.3% and this the best accuracy achieved so far.

View your TensorBoard at
https://tensorboard.dev/experiment/7IzsQ5U2R76jjUKar4SarQ/

Next we will perform cross validation and evaluate the model with the test set.

5.3.2. Evaluate the model

To evaluate the model we will use 5-fold cross-validation
the best model will be used and will be trained for 20 epochs with the following parameters:

- epochs=20,
- batch_size=32,
- activation = 'relu',
- kernel_initializer = 'glorot_uniform',
- kernel_size = 5,
- conv1_filters = 2**7,
- conv2_filters = 2**7,
- conv3_filters = 2**5,
- optimizer = 'Adam',
- lr = 0.001,
- drop_out = True,
- drop_out_rate = 0.2,
- batch_norm = True,

KerasClassifier wrapper form scikeras will be used to wrap the keras model so that we can use cross-validation score from sklearn

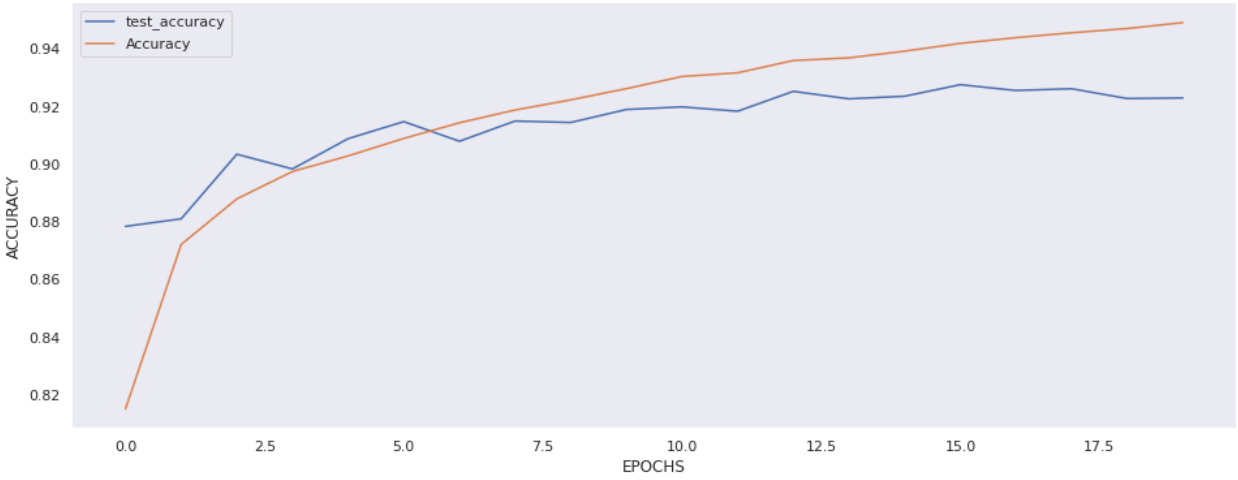
the mean validation accuracy is : 92.6%
and this is a great improvement from the original LeNet-5 architecture

Now let's see how the model performs on the test set.

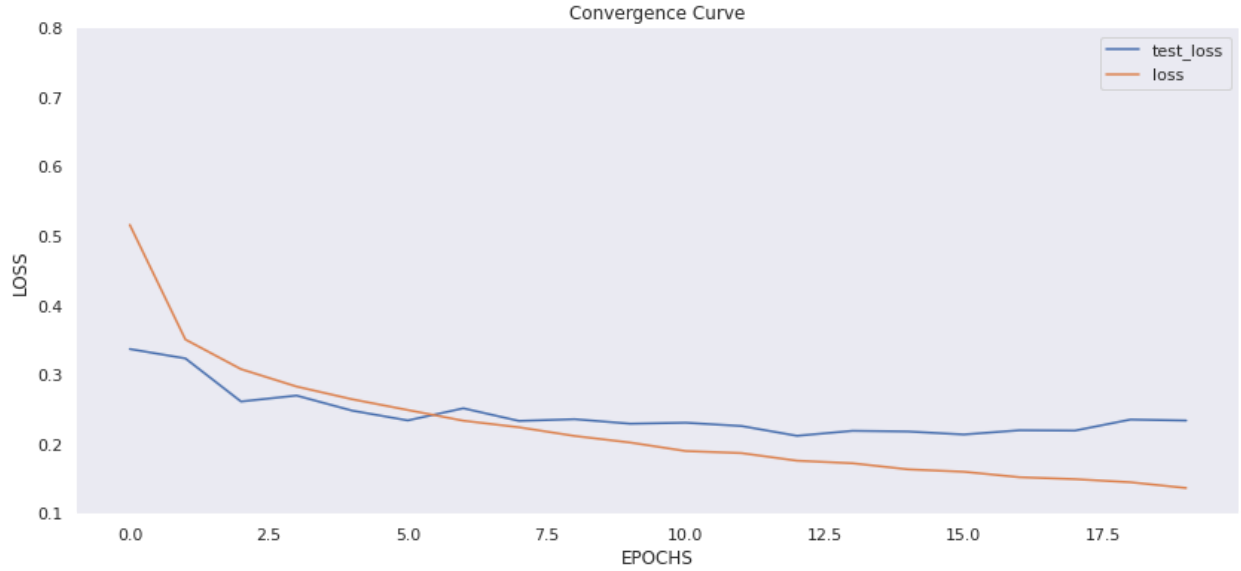
After training for 20 epochs with the test set as validation we got the following results:

loss: 0.1374 - accuracy: 0.9491 - val_loss: 0.2347 - val_accuracy: 0.9230

Train set accuracy vs test set accuracy



Train set loss vs test set loss



And the confusion matrix:

View TensorBoard at
https://tensorboard.dev/experiment/H4wmdBt7SzS2w1R25XBZMA/

- The accuracy improved because the lenet5 architecture is too old
- Now relu activation function is far better tanh
- Adding dropout helps improve the learning process and reduce overfitting
- Batch normalization also reduce overfitting
- Fashion data set is different from digit dataset which lenet5 is made for
- Fashion dataset is more complex so we need a complex model so using bigger filters helped to improve the accuracy.

5.4. Transfer Learning

Transfer learning generally refers to a process where a model trained on one problem is used in some way on a second related problem.

There are perhaps a dozen or more top-performing models for image recognition that can be downloaded and used as the basis for image recognition and related computer vision tasks.

Perhaps three of the more popular models are as follows:

- VGG (e.g. VGG16 or VGG19).
- GoogLeNet (e.g. InceptionV3).
- Residual Network (e.g. ResNet50).

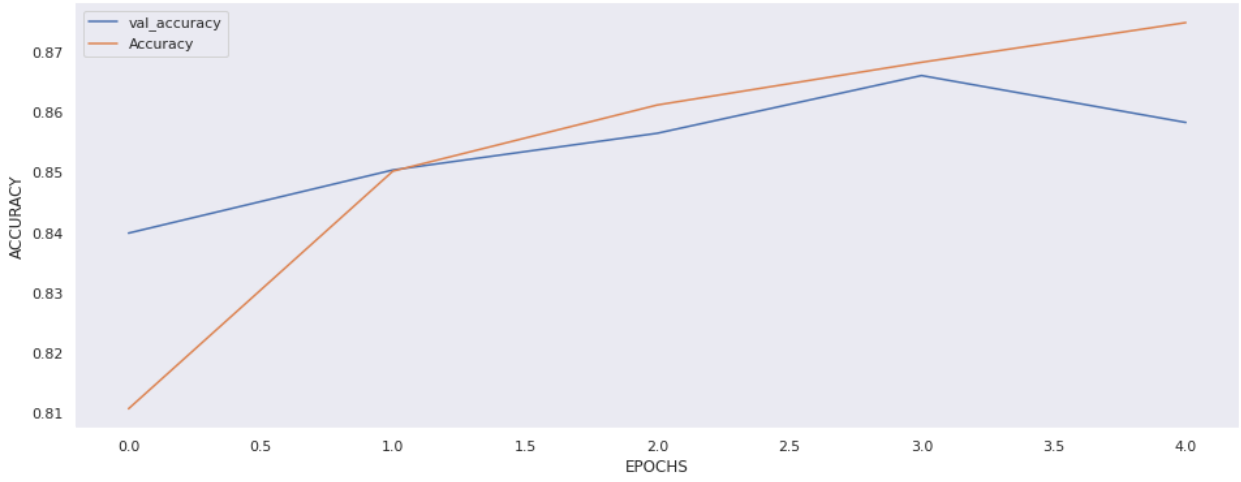
These models are both widely used for transfer learning both because of their performance, but also because they were examples that introduced specific architectural innovations, namely consistent and repeating structures (VGG), inception modules (GoogLeNet), and residual modules (ResNet)

In this part we choose two pre-trained model using the transfer learning (VGG16 and ResNet50) to compare the best LeNet model result with others pre-trained models

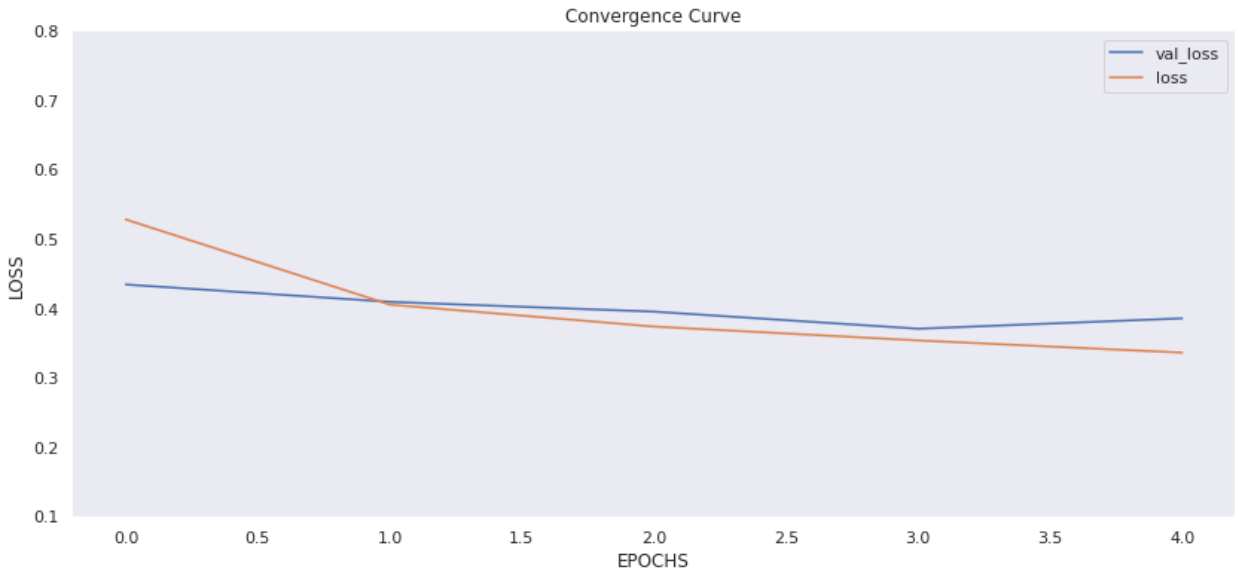
5.4.1. Fourth trial: Transfer Learning using VGG16

For the fourth trial, we trained this model using 5 epochs with EarlyStopping callback that monitor the validation loss with patience value equal to 3

By measuring the model accuracy we get 85.43% as shown in the following figure



Training loss vs testing loss `'Convergence Curve'`



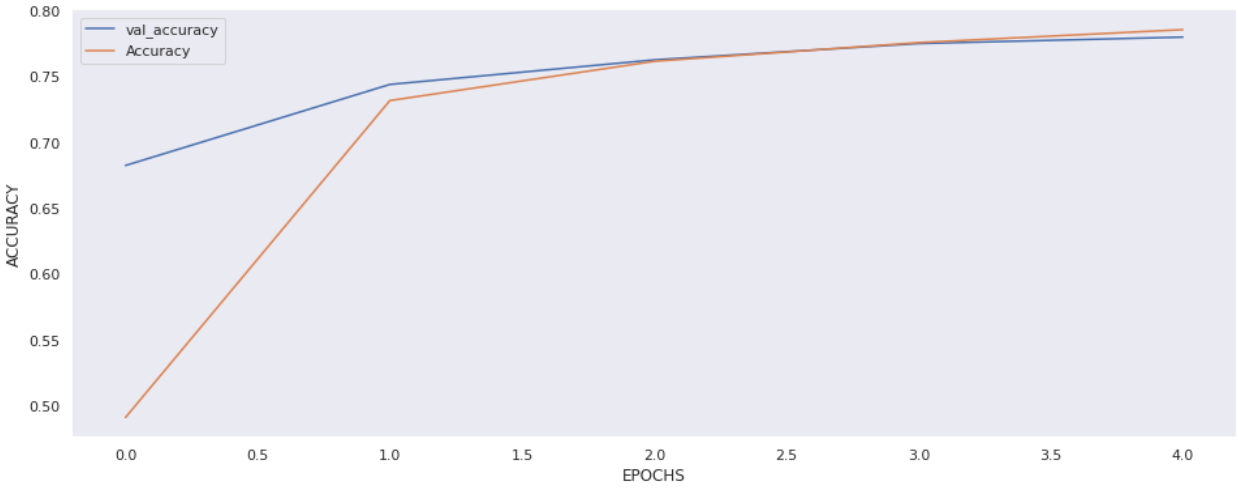
Confusion matrix

From these results, we can say that Modified LeNet is better than VGG with transfer learning

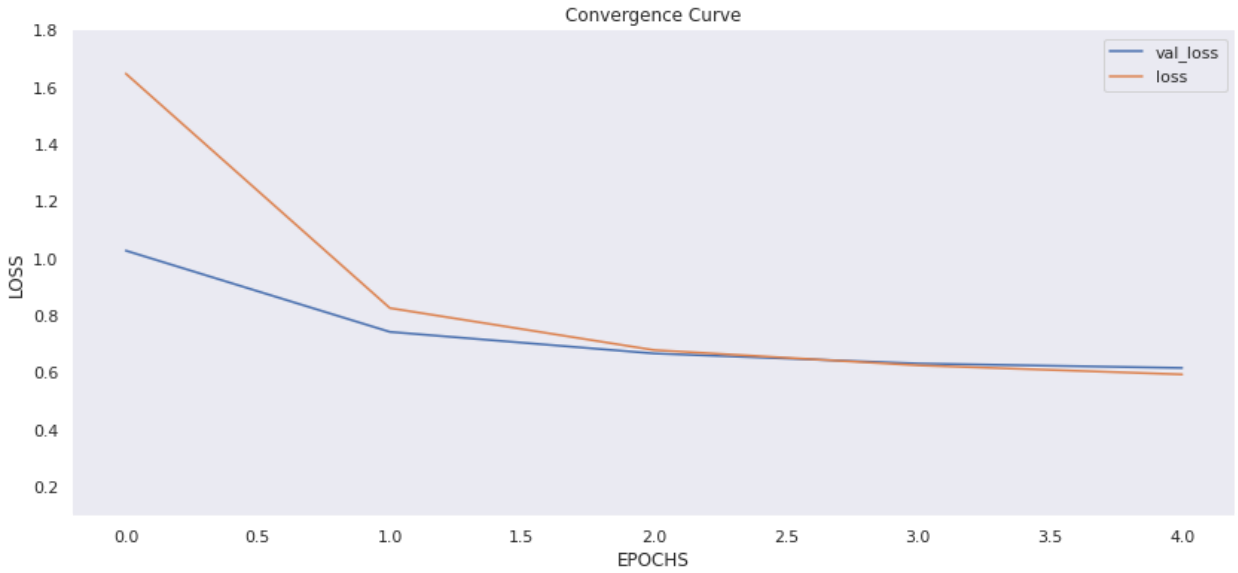
5.4.2. Fifth trial: Transfer Learning using ResNet50

For the fifth trial, we trained this model using 5 epochs with EarlyStopping callback that monitor the validation loss with patience value equal to 3

By measuring the model accuracy we get 74.92% as shown in the following figure



Training loss vs testing loss `'Convergence Curve'`



Tensorboard of trial 5 :
https://tensorboard.dev/experiment/4eVjzh4DTA6zE5OWI9pyNw/

6. Conclusion

Out of all the trials, we can notice that the best accuracy was 92.6% which obtained by the third trial (Modifying hyperparameters for LeNet-5) and this is a slightly improvement from the original LeNet-5 architecture that achieve 90.9%

Also we notice that the accuracy of transfer learning models is consider far away from the one achieved by LeNet-5 architecture especially when we used ResNet50 model that produce only 74.92%

All models are evaluated using 5-Fold cross-validation
KerasClassifier wrapper form scikeras will be used to wrap the keras model so that we can use cross-validation score from sklearn