CISC 867 - Deep Learning

Project 2: Fashion-MNist classification.

Supervised by:

Dr. Hazem Abbas

T.A. Asef Mahfouz

T.A. Reem Abdel-salam

Name: Manar Samy Elghobashy

Id: 20398569

problem definition:

The fashion-MNIST dataset is used for solving a classification problem using Deep Learning. This dataset has a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28 x 28 grayscale image, associated with a label from 10 classes

problem formulation:

- Input: - 28×28-pixel grayscale images of items of 10 types of clothing, such as shoes, t-shirts, dresses, and more
- Output: - Predict the type of clothes (Bag, Shirt ...etc.)
- Deep Learning Function: - Manipulating, analyzing, preprocessing the data, and training the data.

Dataset Description:

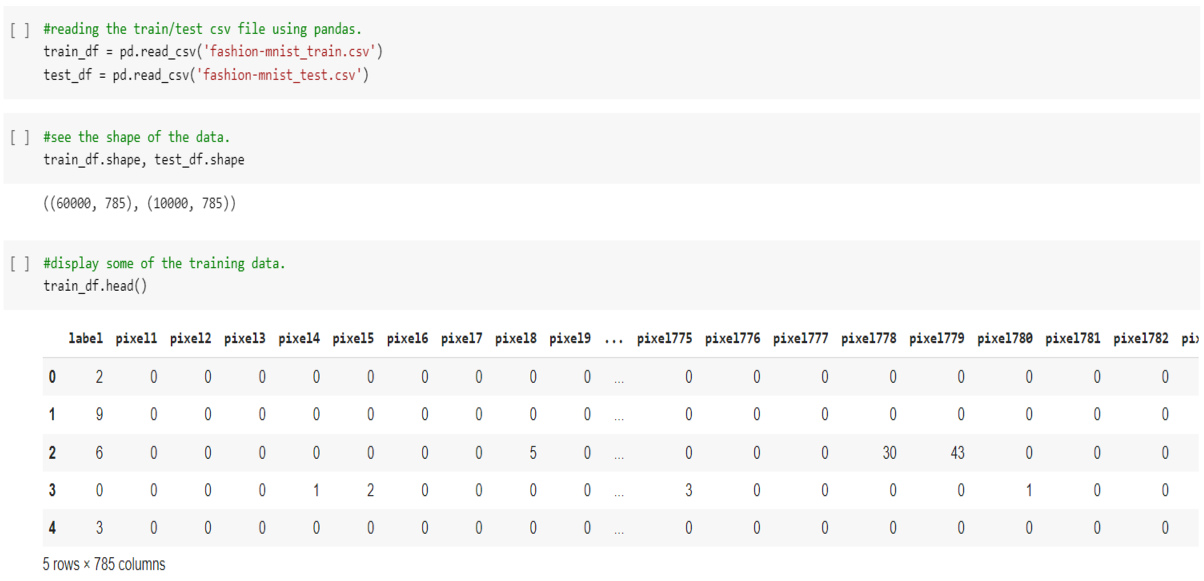
Fashion is a dataset of Zalando's article images consisting of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 grayscale image, associated with a label from 10 classes. It is split into 10,000 as a test and 50,000 as a training dataset. Although the dataset is relatively simple, it can be used as the basis for learning and practicing how to develop, evaluate, and use deep convolutional neural networks for image classification from scratch.

It is a dataset comprised of 60,000 images with a shape 28×28 pixel grayscale of items of 10 types of clothing, such as shoes, t-shirts, dresses, and more. The mapping of all 0-9 integers to class labels defined is as:

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

Part I: Data Preparation.

Reading train/test data:



```
[ ] #reading the train/test csv file using pandas.
    train_df = pd.read_csv('fashion-mnist_train.csv')
    test_df = pd.read_csv('fashion-mnist_test.csv')
```

```
[ ] #see the shape of the data.
    train_df.shape, test_df.shape
```

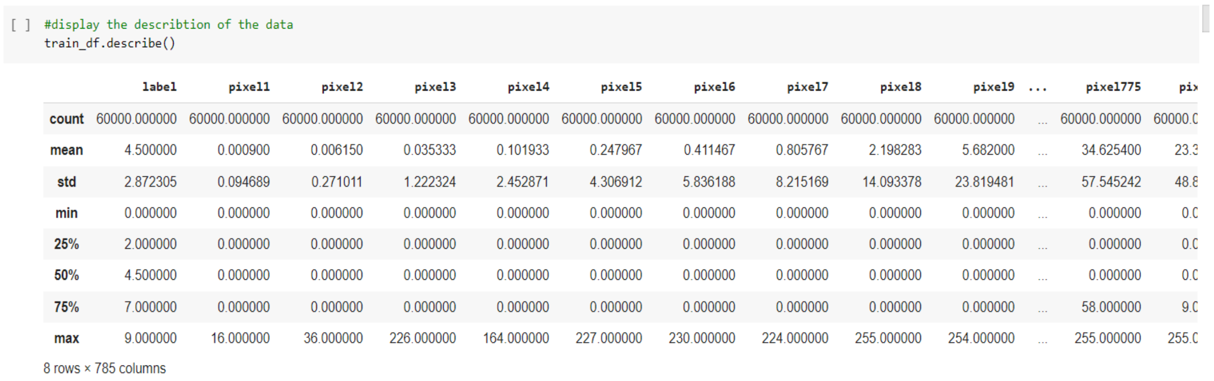
```
((60000, 785), (10000, 785))
```

```
[ ] #display some of the training data.
    train_df.head()
```

	label	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel775	pixel776	pixel777	pixel778	pixel779	pixel780	pixel781	pixel782	pi
0	2	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	
1	9	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	
2	6	0	0	0	0	0	0	0	5	0	...	0	0	0	30	43	0	0	0	
3	0	0	0	0	1	2	0	0	0	0	...	3	0	0	0	0	1	0	0	
4	3	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	

5 rows × 785 columns

1. Describe the data.
   By using describe () function a detailed description of the data like count, mean, std, etc.



```
[ ] #display the describtion of the data
    train_df.describe()
```

	label	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel775	pi
count	60000.000000	60000.000000	60000.000000	60000.000000	60000.000000	60000.000000	60000.000000	60000.000000	60000.000000	60000.000000	...	60000.000000	60000.C
mean	4.500000	0.000900	0.006150	0.035333	0.101933	0.247967	0.411467	0.805767	2.198283	5.682000	...	34.625400	23.3
std	2.872305	0.094689	0.271011	1.222324	2.452871	4.306912	5.836188	8.215169	14.093378	23.819481	...	57.545242	48.8
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.C
25%	2.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.C
50%	4.500000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.C
75%	7.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	58.000000	9.C
max	9.000000	16.000000	36.000000	226.000000	164.000000	227.000000	230.000000	224.000000	255.000000	254.000000	...	255.000000	255.C

8 rows × 785 columns

2. Check null and duplication in the data:
   Using isnull () and duplicated () functions to check if there are any null or missing and duplicated data. Then, dropped the duplicated data from the train data.

```
[ ] #see if there are any null in the data
    train_df.isnull().sum().sum()

    0
```

```
[ ]
    # check if there are any duplications.
    train_df.duplicated().sum()

    43
```

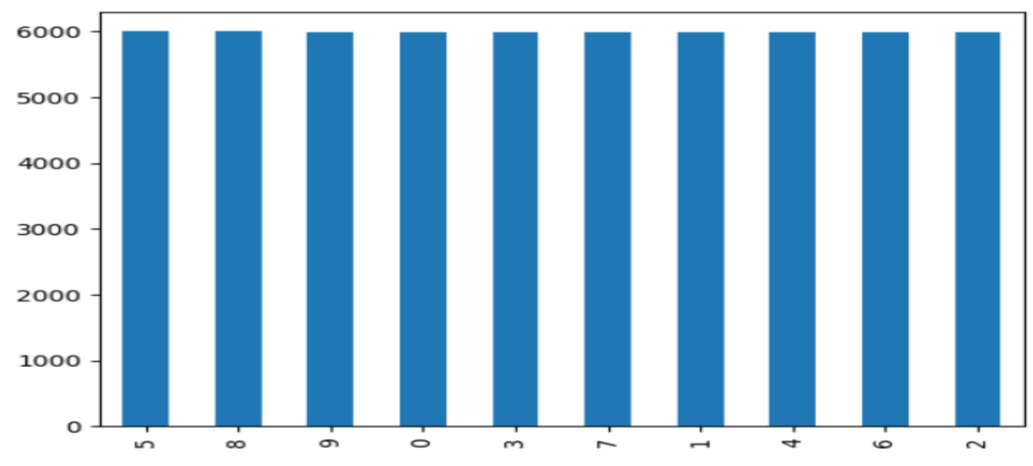
```
[ ] #drop duplicated data
    train_df.drop_duplicates(subset=None, keep="first", inplace=True)
```

```
[ ] train_df.duplicated().sum()

    0
```

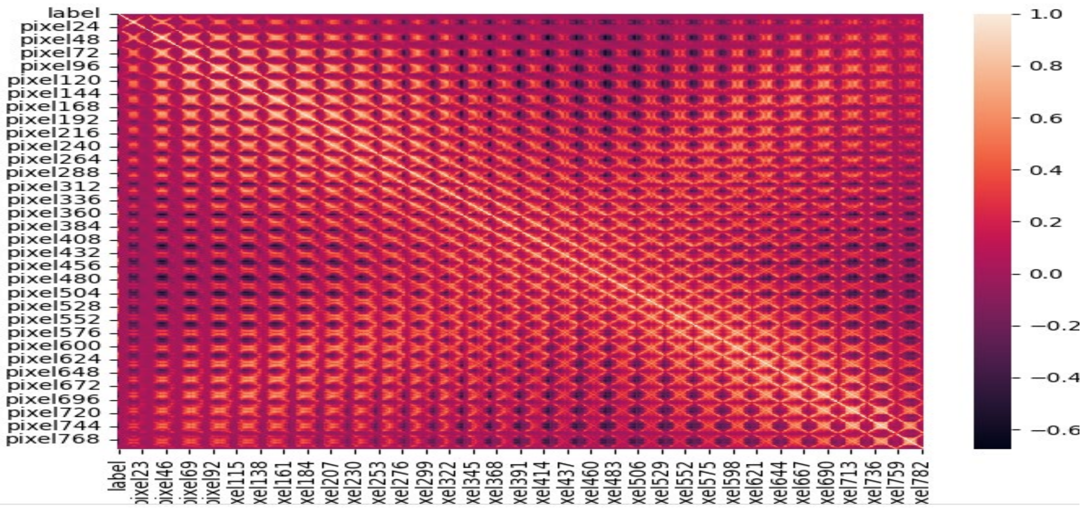
3. Visualize the data using proper visualization methods
   Display some visualization as labels distribution and heatmap to understand the data better.

```
#visualize the label data count.
train_df['label'].value_counts().plot.bar()
```

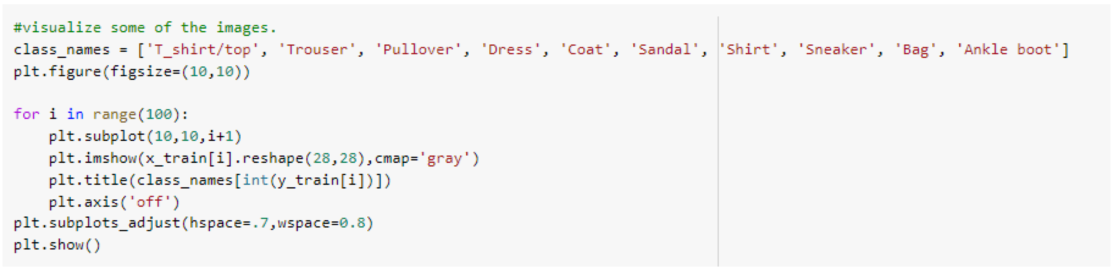
```
<Axes: >
```



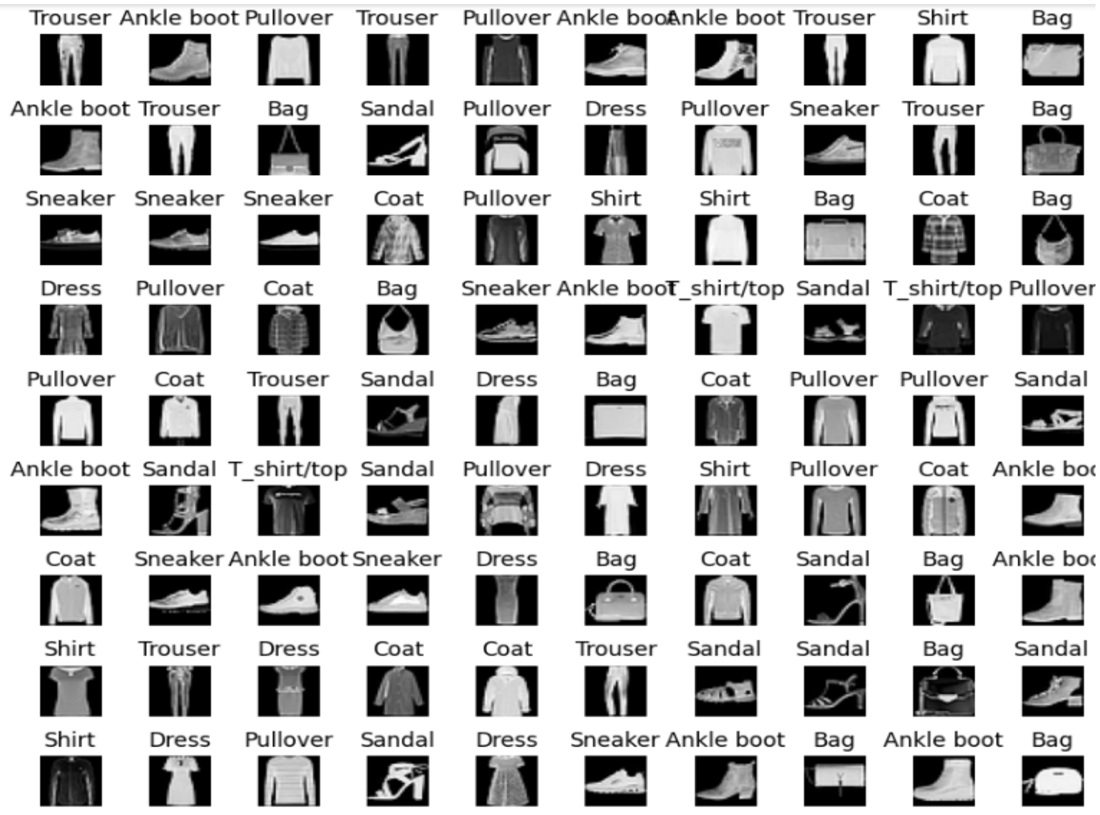
```
plt.figure(figsize=(8, 6))

# plotting correlation heatmap
dataplot = sns.heatmap(train_df.corr())

# displaying heatmap
plt.show()
```



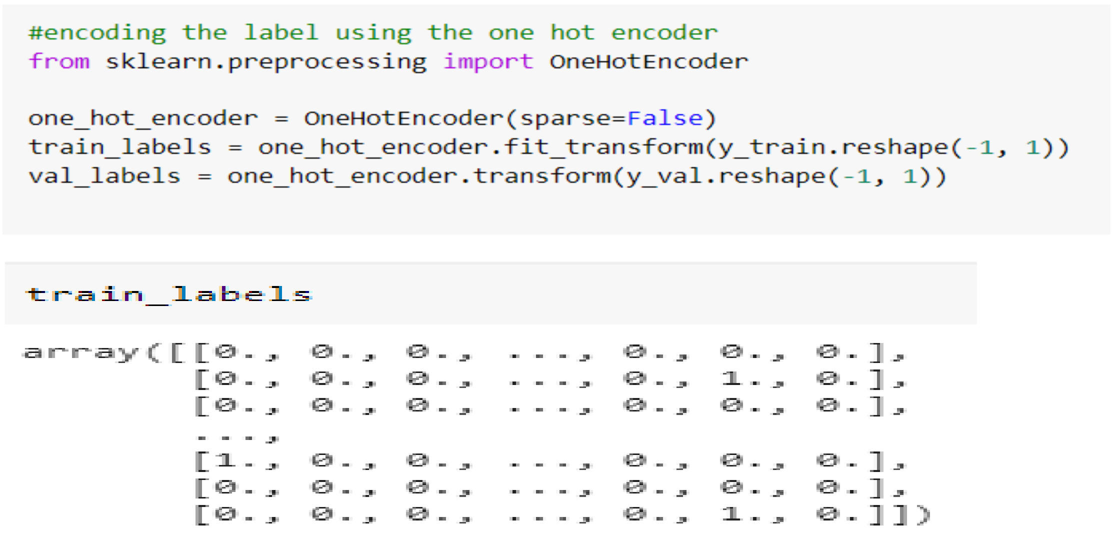
4. Draw some images:



```
#visualize some of the images.
class_names = ['T_shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat', 'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
plt.figure(figsize=(10,10))

for i in range(100):
    plt.subplot(10,10,i+1)
    plt.imshow(x_train[i].reshape(28,28),cmap='gray')
    plt.title(class_names[int(y_train[i])])
    plt.axis('off')
plt.subplots_adjust(hspace=.7,wspace=0.8)
plt.show()
```



5. Label encoder:
   Encoding the label column using one hot encoder

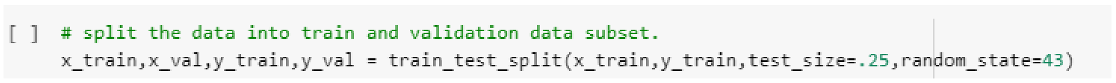


```
#encoding the label using the one hot encoder
from sklearn.preprocessing import OneHotEncoder

one_hot_encoder = OneHotEncoder(sparse=False)
train_labels = one_hot_encoder.fit_transform(y_train.reshape(-1, 1))
val_labels = one_hot_encoder.transform(y_val.reshape(-1, 1))
```

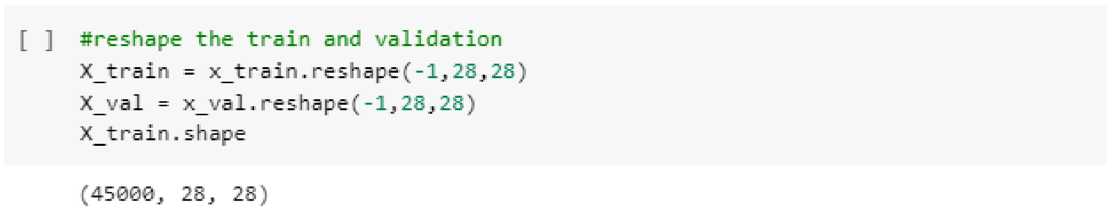
```
train_labels
```

```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 1., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [1., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 1., 0.]])
```

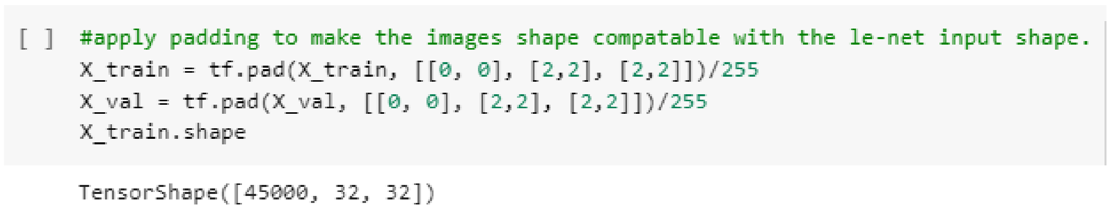
6. Required processes on the data before building the models: Split the data into train and validation sets, normalize the data by dividing it by 255 as the images are grayscale, and reshape it to be compatible with the model input shape.



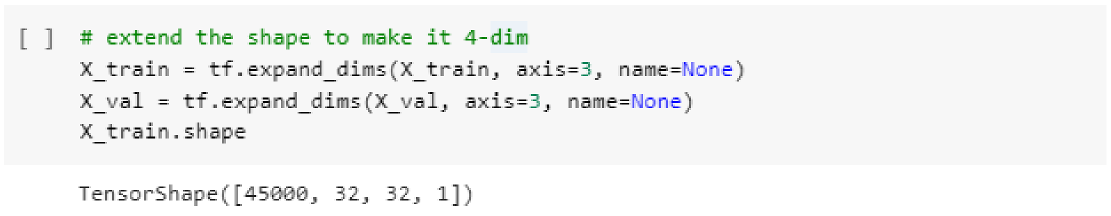
```
[ ]  # split the data into train and validation data subset.
     x_train,x_val,y_train,y_val = train_test_split(x_train,y_train,test_size=.25,random_state=43)
```



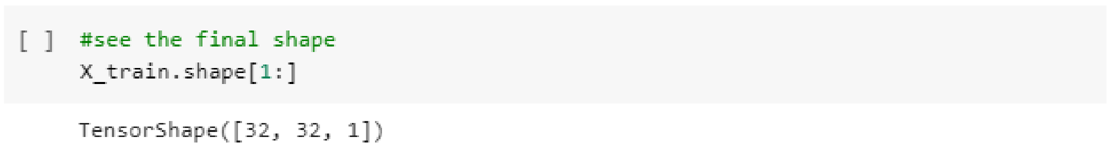
```
[ ]  #reshape the train and validation
     X_train = x_train.reshape(-1,28,28)
     X_val = x_val.reshape(-1,28,28)
     X_train.shape

     (45000, 28, 28)
```



```
[ ]  #apply padding to make the images shape compatable with the le-net input shape.
     X_train = tf.pad(X_train, [[0, 0], [2,2], [2,2]])/255
     X_val = tf.pad(X_val, [[0, 0], [2,2], [2,2]])/255
     X_train.shape

     TensorShape([45000, 32, 32])
```



```
[ ]  # extend the shape to make it 4-dim
     X_train = tf.expand_dims(X_train, axis=3, name=None)
     X_val = tf.expand_dims(X_val, axis=3, name=None)
     X_train.shape

     TensorShape([45000, 32, 32, 1])
```



```
[ ]  #see the final shape
     X_train.shape[1:]

     TensorShape([32, 32, 1])
```

Part II: Training a CNN neural network

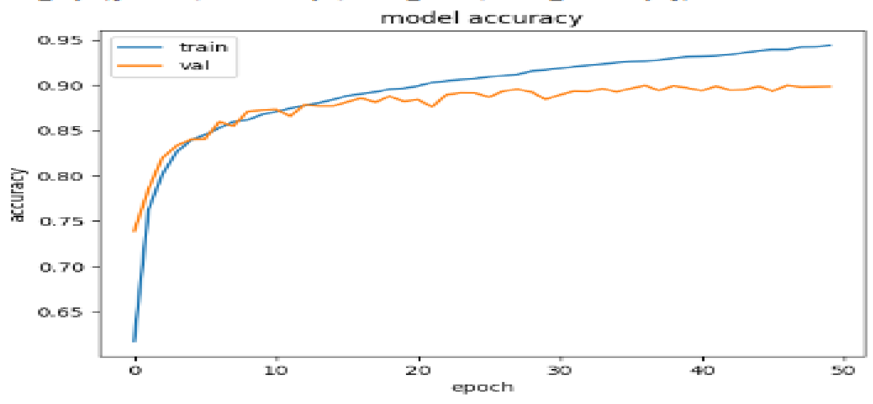
1. Implement a LeNet-5 model:

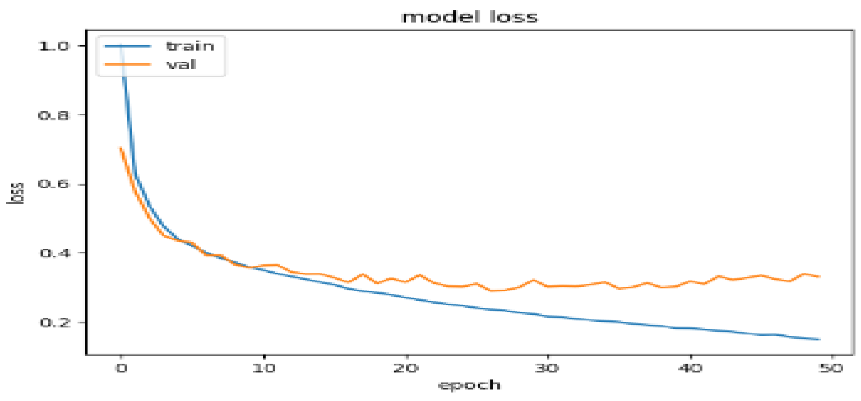
- Lenet-5 is one of the earliest pre-trained models proposed by Yann LeCun and others in the year 1998.
- LeNet-5 CNN architecture is made up of 8 layers. The layer composition consists of 3 convolutional layers, 2 subsampling layers, and 3 fully connected layers.
- Implement the LeNet-5 model using the sequential model that has the following architecture.

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_8 (Conv2D)           (None, 32, 32, 32)        832

 max_pooling2d_8 (MaxPooling  (None, 16, 16, 32)       0
 2D)

 conv2d_9 (Conv2D)           (None, 12, 12, 48)        38448

 max_pooling2d_9 (MaxPooling  (None, 6, 6, 48)         0
 2D)

 flatten_4 (Flatten)         (None, 1728)              0

 dense_12 (Dense)            (None, 256)               442624

 dense_13 (Dense)            (None, 84)                21588

 dense_14 (Dense)            (None, 10)                850

=================================================================
Total params: 504,342
Trainable params: 504,342
Non-trainable params: 0
```

- The model has an accuracy of 0.9439 and a validation accuracy of 0.8985.
- The loss-accuracy curves of the model:

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

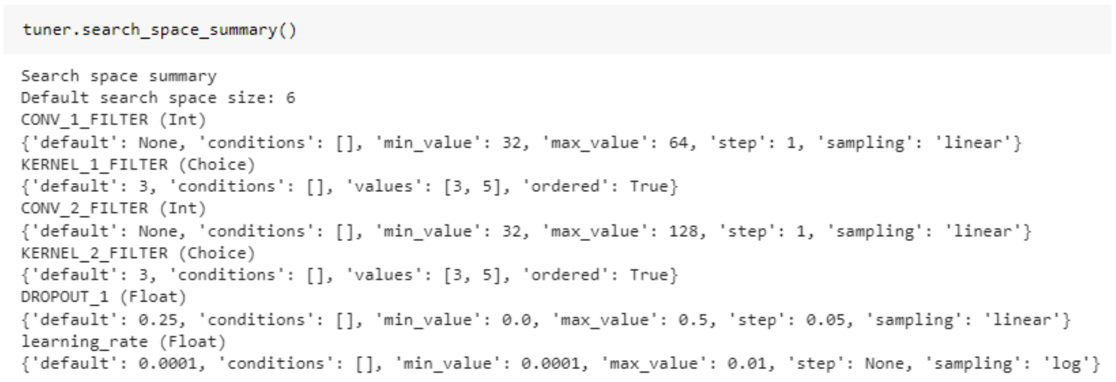




2. Modify hyperparameters to get the best performance:

Using the Keras tuner to find the best hyperparameter combinations and get the best performance.

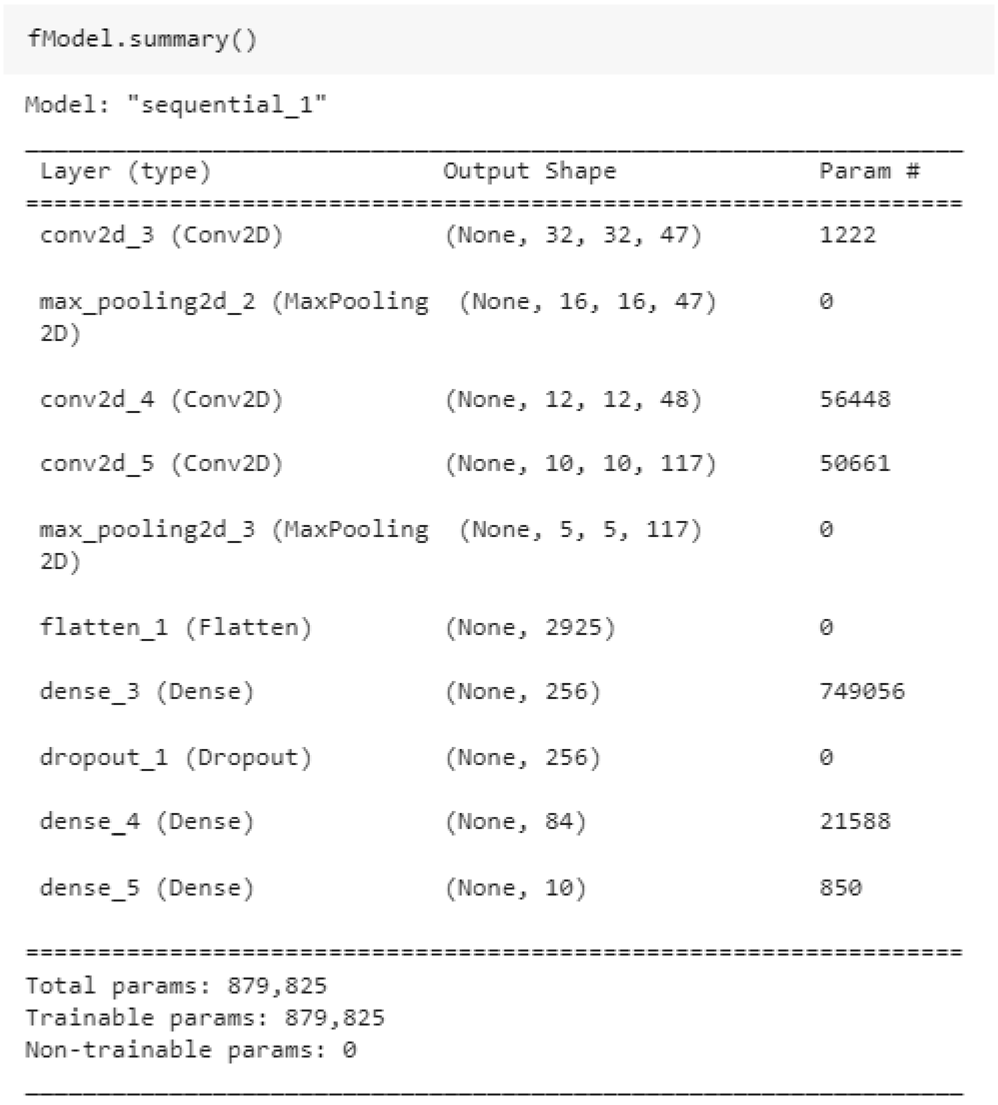
- Model hyperparameters search space:



```
tuner.search_space_summary()
```

```
Search space summary
Default search space size: 6
CONV_1_FILTER (Int)
{'default': None, 'conditions': [], 'min_value': 32, 'max_value': 64, 'step': 1, 'sampling': 'linear'}
KERNEL_1_FILTER (Choice)
{'default': 3, 'conditions': [], 'values': [3, 5], 'ordered': True}
CONV_2_FILTER (Int)
{'default': None, 'conditions': [], 'min_value': 32, 'max_value': 128, 'step': 1, 'sampling': 'linear'}
KERNEL_2_FILTER (Choice)
{'default': 3, 'conditions': [], 'values': [3, 5], 'ordered': True}
DROPOUT_1 (Float)
{'default': 0.25, 'conditions': [], 'min_value': 0.0, 'max_value': 0.5, 'step': 0.05, 'sampling': 'linear'}
learning_rate (Float)
{'default': 0.0001, 'conditions': [], 'min_value': 0.0001, 'max_value': 0.01, 'step': None, 'sampling': 'log'}
```

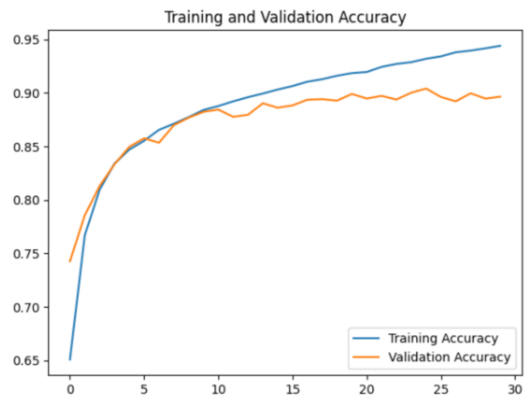
- Best model architecture:

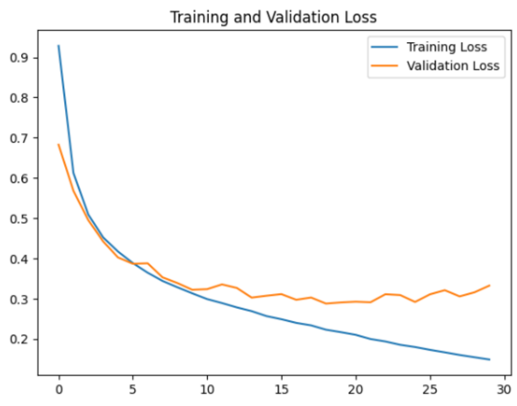


```
fModel.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_3 (Conv2D)           (None, 32, 32, 47)        1222      
                                                                 
 max_pooling2d_2 (MaxPooling  (None, 16, 16, 47)       0         
 2D)                                                             
                                                                 
 conv2d_4 (Conv2D)           (None, 12, 12, 48)        56448     
                                                                 
 conv2d_5 (Conv2D)           (None, 10, 10, 117)       50661     
                                                                 
 max_pooling2d_3 (MaxPooling  (None, 5, 5, 117)        0         
 2D)                                                             
                                                                 
 flatten_1 (Flatten)         (None, 2925)              0         
                                                                 
 dense_3 (Dense)             (None, 256)               749056    
                                                                 
 dropout_1 (Dropout)         (None, 256)               0         
                                                                 
 dense_4 (Dense)             (None, 84)                21588     
                                                                 
 dense_5 (Dense)             (None, 10)                850       
                                                                 
=================================================================
Total params: 879,825
Trainable params: 879,825
Non-trainable params: 0
_________________________________________________________________
```

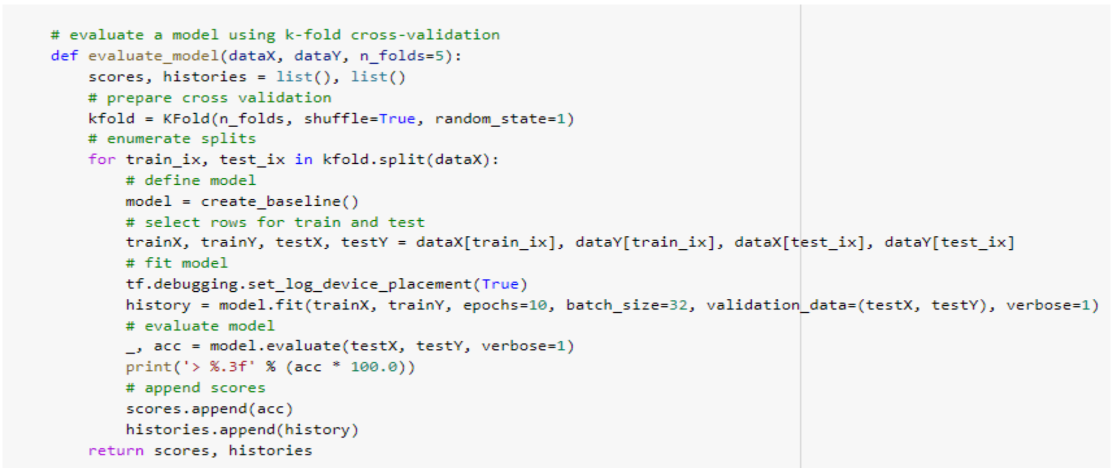
- Accuracy of 0.9440 and a validation accuracy of 0.8964.
- Accuracy-loss curves:





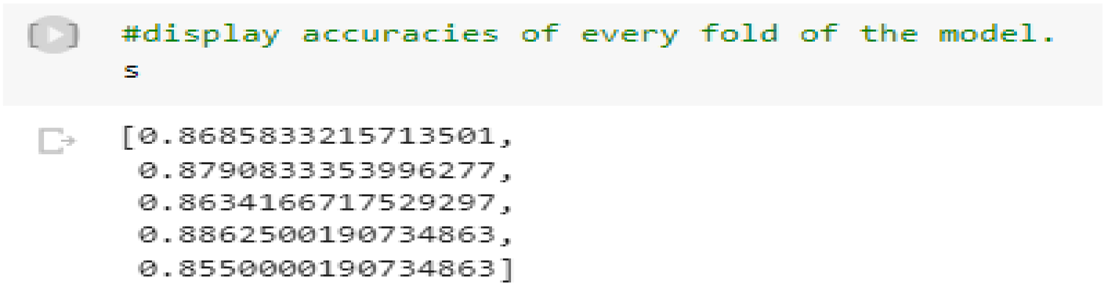
3. Evaluate the model using 5-fold cross-validation:

A function to evaluate the model with 5 k-folds.



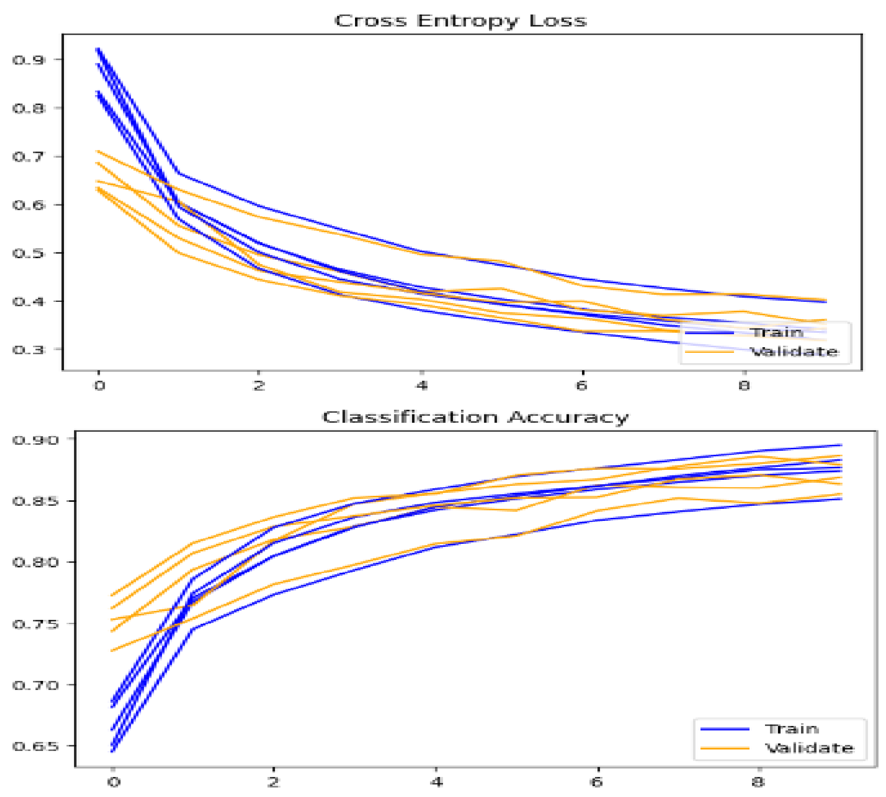
```
# evaluate a model using k-fold cross-validation
def evaluate_model(dataX, dataY, n_folds=5):
    scores, histories = list(), list()
    # prepare cross validation
    kfold = KFold(n_folds, shuffle=True, random_state=1)
    # enumerate splits
    for train_ix, test_ix in kfold.split(dataX):
        # define model
        model = create_baseline()
        # select rows for train and test
        trainX, trainY, testX, testY = dataX[train_ix], dataY[train_ix], dataX[test_ix], dataY[test_ix]
        # fit model
        tf.debugging.set_log_device_placement(True)
        history = model.fit(trainX, trainY, epochs=10, batch_size=32, validation_data=(testX, testY), verbose=1)
        # evaluate model
        _, acc = model.evaluate(testX, testY, verbose=1)
        print('> %.3f' % (acc * 100.0))
        # append scores
        scores.append(acc)
        histories.append(history)
    return scores, histories
```

- Accuracies of the five folds:



```
[ ]  #display accuracies of every fold of the model.
     s

     [0.8685833215713501,
      0.8790833353996277,
      0.8634166717529297,
      0.8862500190734863,
      0.8550000190734863]
```

- Accuracy-loss curves for the five folds:



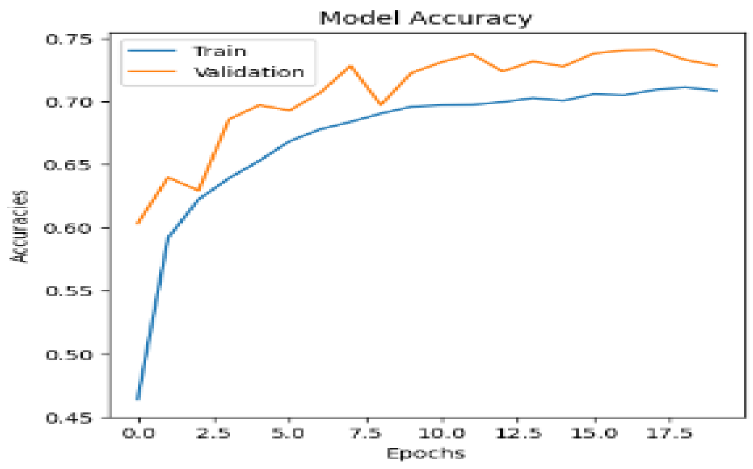
4. why do you think LeNet-5 further improves the accuracy if any at all? And if it doesn't, why not?

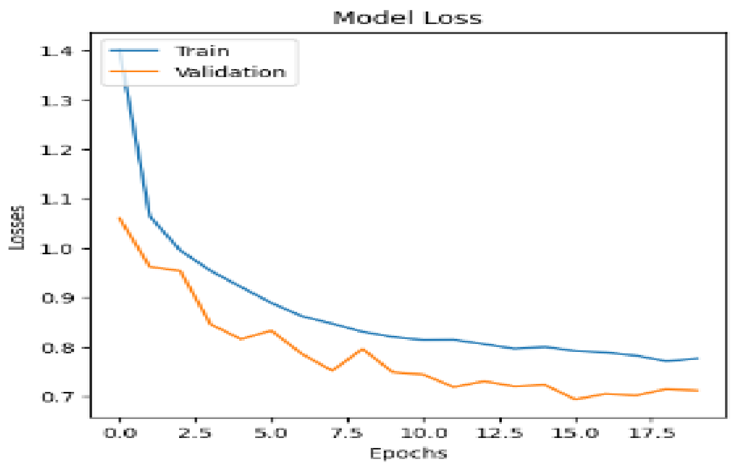
- LeNet was used in detecting handwritten cheques by banks based on the MNIST dataset. Fully connected networks and activation functions were previously known in neural networks. LeNet-5 introduced convolutional and pooling layers. LeNet-5 is believed to be the base for all other ConvNets.
- LeNet is modest and simple to learn, but it produces interesting results. Additionally, the LeNet + MNIST combo may run on the CPU, making it simple for newcomers to get started with Deep Learning and Convolutional Neural Networks.

5. two CNN models (using transfer learning):

5.1. ResNet 152 V2 mode:

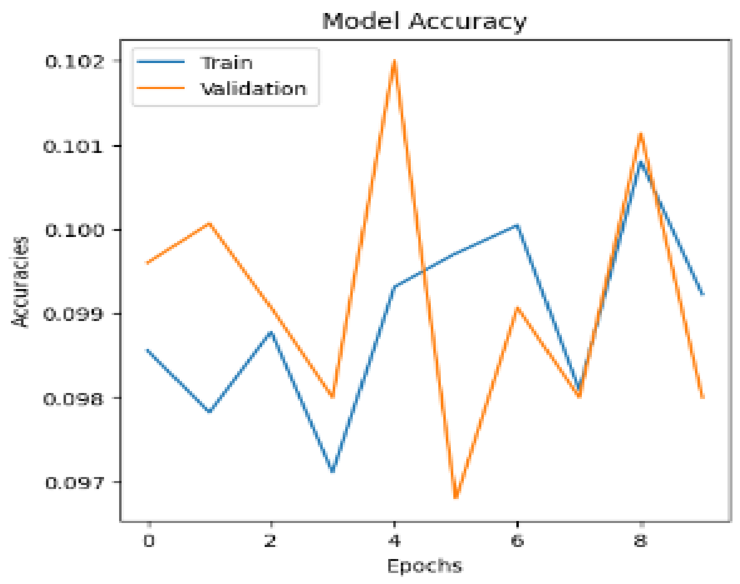
- ResNet model pre-trained on ImageNet-1k at resolution 224x224. It was introduced in the paper Deep Residual Learning for Image Recognition by He et al. Disclaimer. In this model, we convert the shape to our image shape (32,32,1)
- The model has an accuracy of 0.7085 and a validation accuracy of 0.7283 which is less than the LeNet model.
- Accuracy-loss curves:

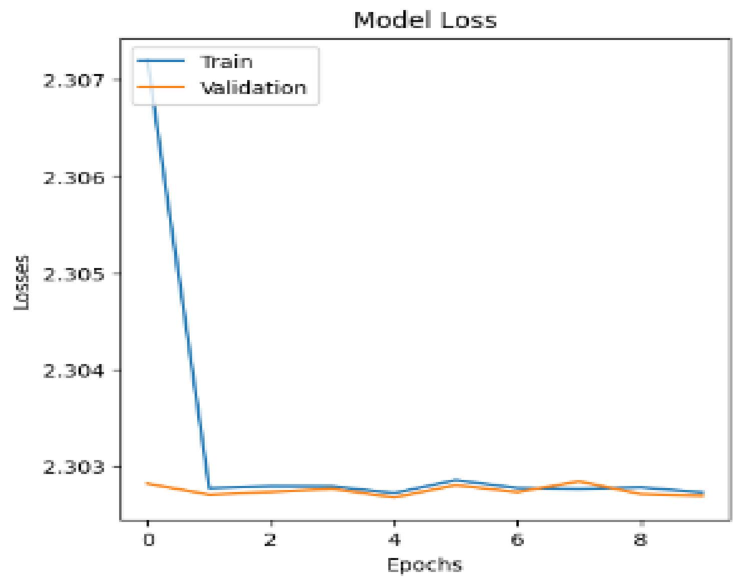




5.2. VGG16 model:

- VGG16 is a convolutional neural network model that's used for im age recognition. It's unique in that it has only 16 layers that have weights, as opposed to relying on a large number of hyperparame ters. It's considered one of the best vision model architectures.
- The model has an accuracy of 0.0992 and a validation accuracy of 0.0980 which is very bad compared with the LeNet model.
- Accuracy-loss curves:





- As you can see, when we applied transfer learning, the accuracy decreased. Transfer learning may fail if there is a domain mismatch between the dataset for pretext tasks and the dataset for downstream tasks. Although the pre-trained models may converge, they will become trapped in a local minimum. As a result, the performance will be no better than starting from scratch.

- The LeNet has the best accuracy compared to the VGG16 model and ResNet 152 v2.

References:

1. https://www.tensorflow.org/tutorials/keras/keras_tuner
2. https://www.analyticsvidhya.com/blog/2021/03/the-architecture-of-lenet-5/
3. MNIST Demos on Yann LeCun's website