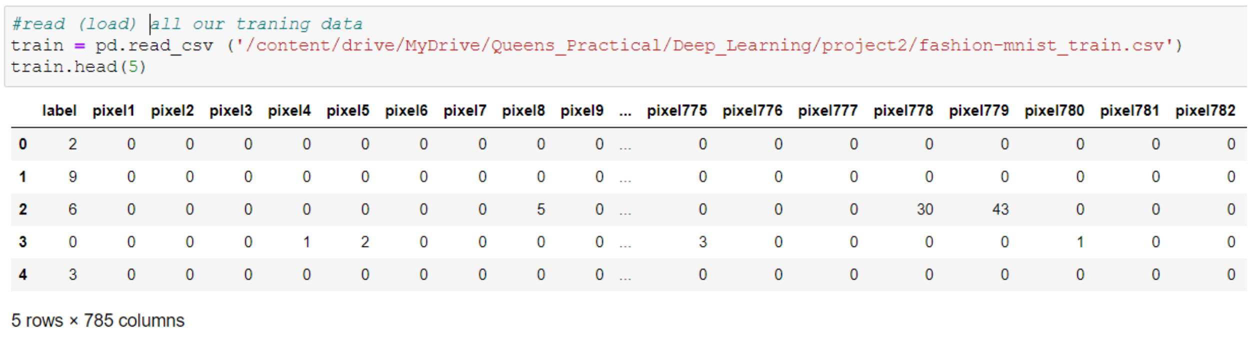
FASHION_MINIST PROJECT

Members:

1- Marim Ashraf Elsayed Mahmoud Amer.
2- Mawada Ashraf Elsayed Mahmoud Amer.
3- Youmna Alsayed Abdalatty Mohamed.

Data Preparation:

1. Download the data file, load it:



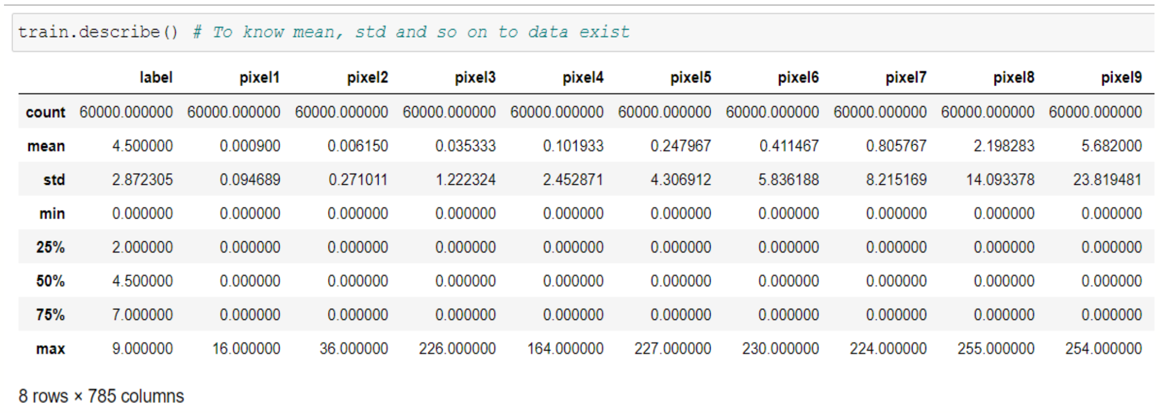
```
#read (load) all our traning data
train = pd.read_csv ('/content/drive/MyDrive/Queens_Practical/Deep_Learning/project2/fashion-mnist_train.csv')
train.head(5)
```

	label	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel775	pixel776	pixel777	pixel778	pixel779	pixel780	pixel781	pixel782
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

2. Describe Data:

   Fashion-MNIST is a dataset that includes a **60,000-item training** set and a **10,000-item test** set. Each image is a ** 28x28 grayscale** graphic with a label from **one of ten categories**. The training and testing splits have the same picture size and structure. Each image has a height of 28 pixels and a width of 28 pixels, for a total of 784 pixels. Each pixel has a single pixel-value that indicates its lightness or darkness, with larger numbers representing darker pixels. This **pixel value** is an integer ranging from **0 to 255**. There are **785 columns** in both the training and test data sets.

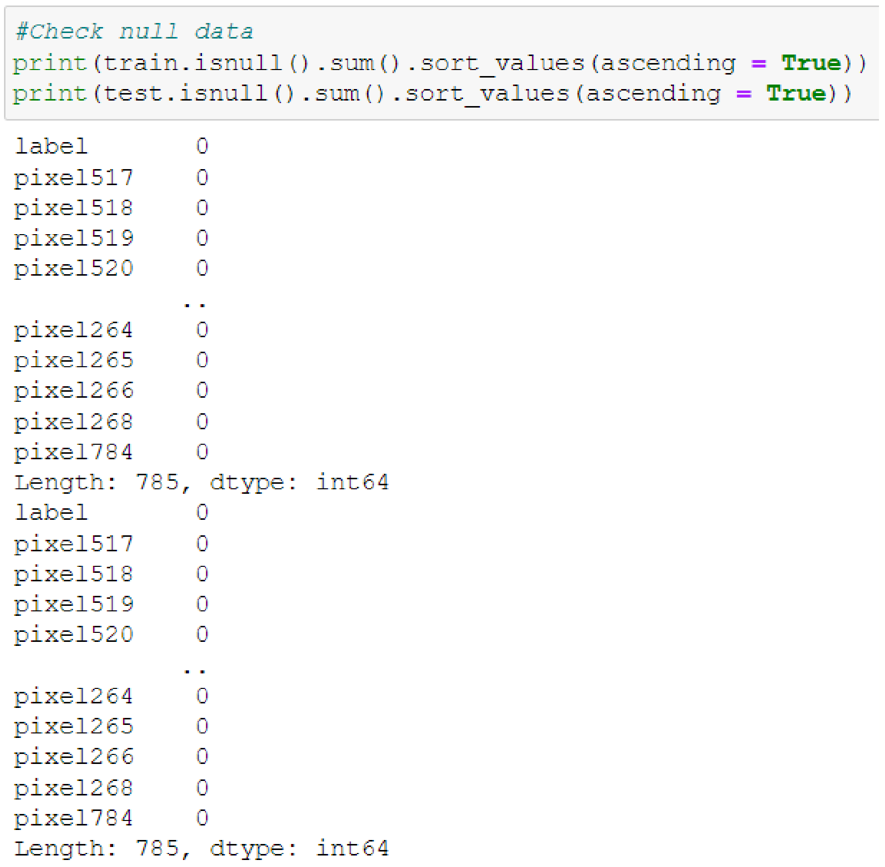


```
train.describe() # To know mean, std and so on to data exist
```

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

8 rows × 785 columns

3. Clean Data
   a. *Check the null values*

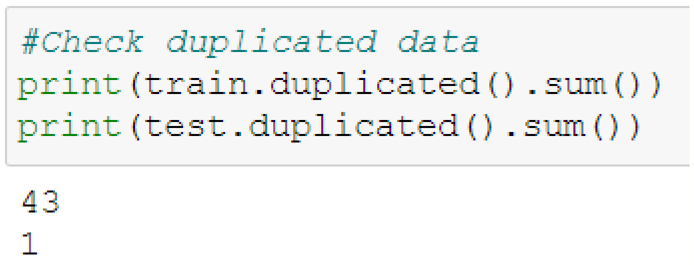


```
#Check null data
print(train.isnull().sum().sort_values(ascending = True))
print(test.isnull().sum().sort_values(ascending = True))
```

```
label       0
pixel517    0
pixel518    0
pixel519    0
pixel520    0
           ..
pixel264    0
pixel265    0
pixel266    0
pixel268    0
pixel784    0
Length: 785, dtype: int64
label       0
pixel517    0
pixel518    0
pixel519    0
pixel520    0
           ..
pixel264    0
pixel265    0
pixel266    0
pixel268    0
pixel784    0
Length: 785, dtype: int64
```

      i. There are no null values in the data.

4. Check the data for missing values or duplicates and carry out proper correction methods
   a. *Check the duplicated data*

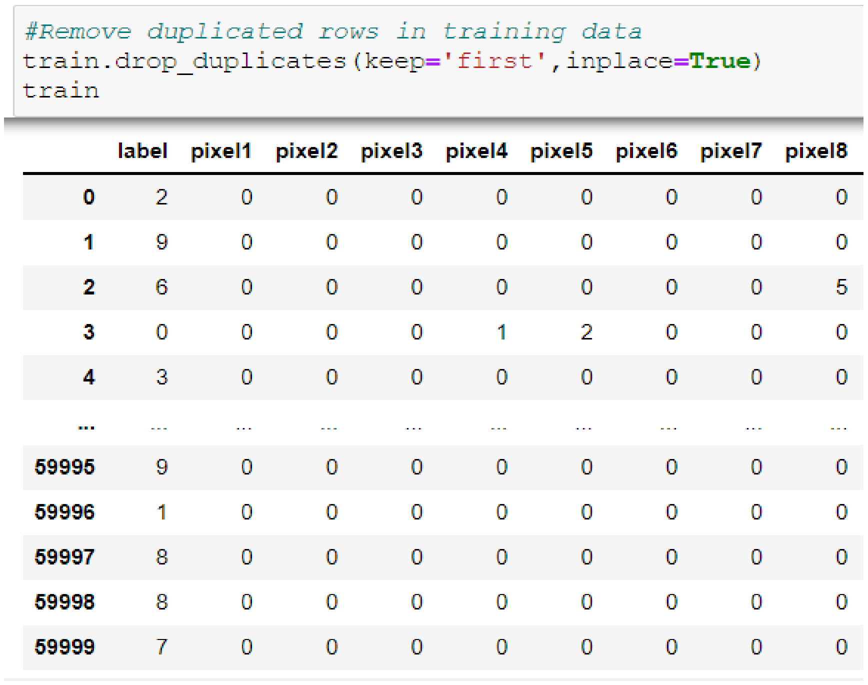


```
#Check duplicated data
print(train.duplicated().sum())
print(test.duplicated().sum())
```

```
43
1
```

      ii. There is 43 duplicated in train data and 1 duplicated in test data.

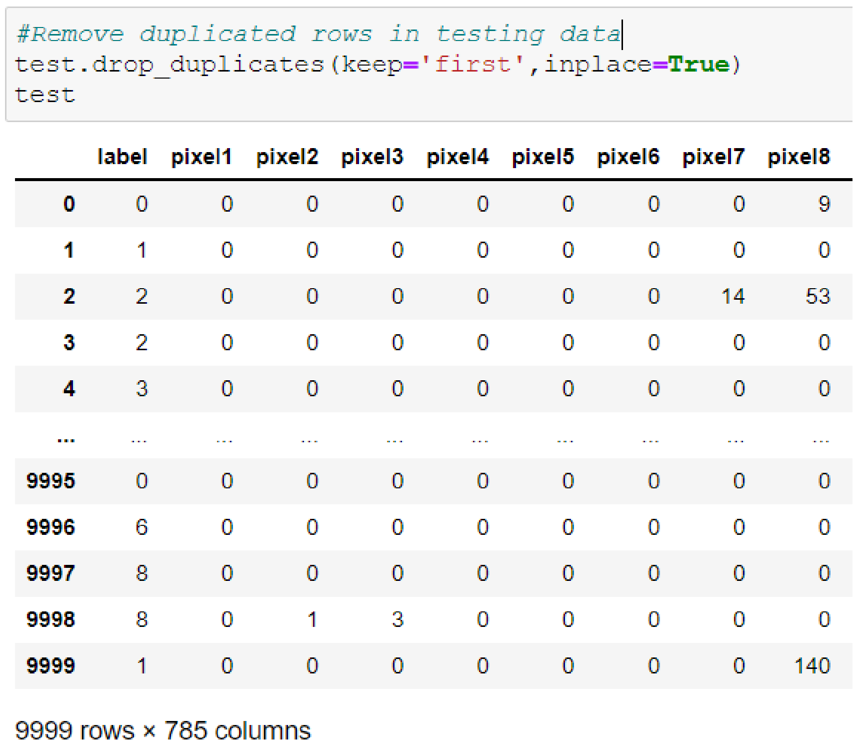
b. Remove duplicated rows in training data



```
#Remove duplicated rows in training data
train.drop_duplicates(keep='first',inplace=True)
train
```

	label	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8
0	2	0	0	0	0	0	0	0	0
1	9	0	0	0	0	0	0	0	0
2	6	0	0	0	0	0	0	0	5
3	0	0	0	0	1	2	0	0	0
4	3	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...
59995	9	0	0	0	0	0	0	0	0
59996	1	0	0	0	0	0	0	0	0
59997	8	0	0	0	0	0	0	0	0
59998	8	0	0	0	0	0	0	0	0
59999	7	0	0	0	0	0	0	0	0

c. Remove duplicated rows in testing data



```
#Remove duplicated rows in testing data
test.drop_duplicates(keep='first',inplace=True)
test
```

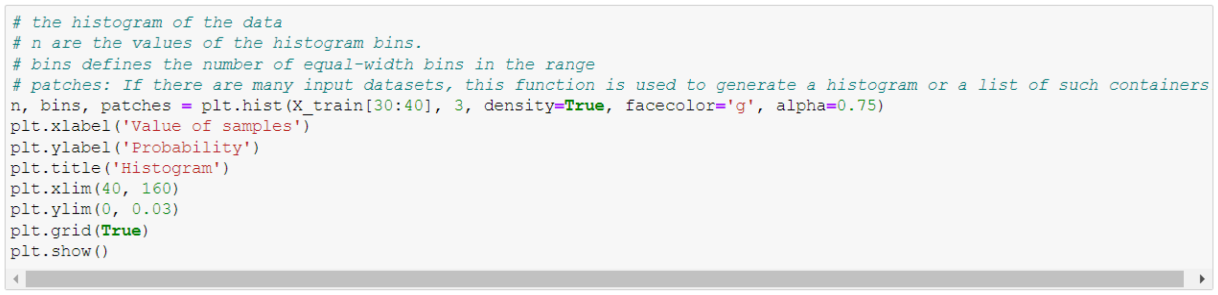
	label	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8
0	0	0	0	0	0	0	0	0	9
1	1	0	0	0	0	0	0	0	0
2	2	0	0	0	0	0	0	14	53
3	2	0	0	0	0	0	0	0	0
4	3	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...
9995	0	0	0	0	0	0	0	0	0
9996	6	0	0	0	0	0	0	0	0
9997	8	0	0	0	0	0	0	0	0
9998	8	0	1	3	0	0	0	0	0
9999	1	0	0	0	0	0	0	0	140

9999 rows × 785 columns

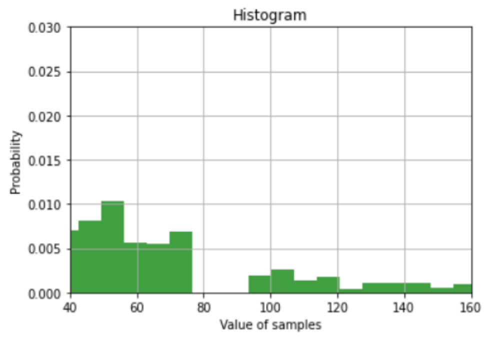
5. Visualize Data

Visualize some of the features: we display 5 plots in a notebook

a. Histogram

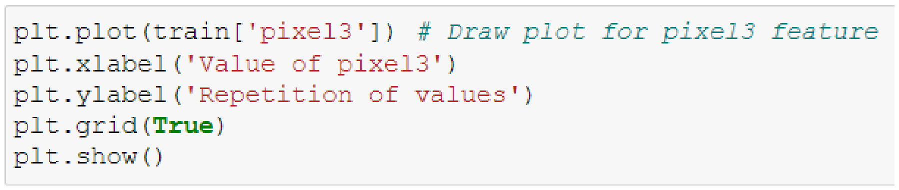


```
# the histogram of the data
# n are the values of the histogram bins.
# bins defines the number of equal-width bins in the range
# patches: If there are many input datasets, this function is used to generate a histogram or a list of such containers
n, bins, patches = plt.hist(X_train[30:40], 3, density=True, facecolor='g', alpha=0.75)
plt.xlabel('Value of samples')
plt.ylabel('Probability')
plt.title('Histogram')
plt.xlim(40, 160)
plt.ylim(0, 0.03)
plt.grid(True)
plt.show()
```

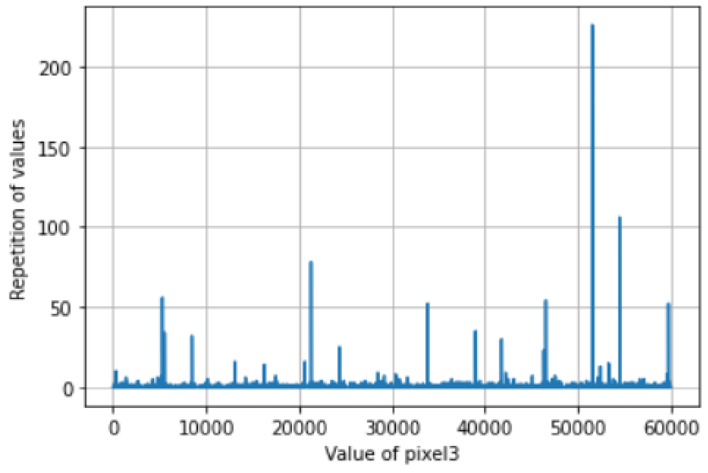


i. From the graph, value of samples range from 40 to 150.
ii. The most value found in this column is 0.010.

b. Plot diagram



```
plt.plot(train['pixel3']) # Draw plot for pixel3 feature
plt.xlabel('Value of pixel3')
plt.ylabel('Repetition of values')
plt.grid(True)
plt.show()
```



The most frequent value is 50600

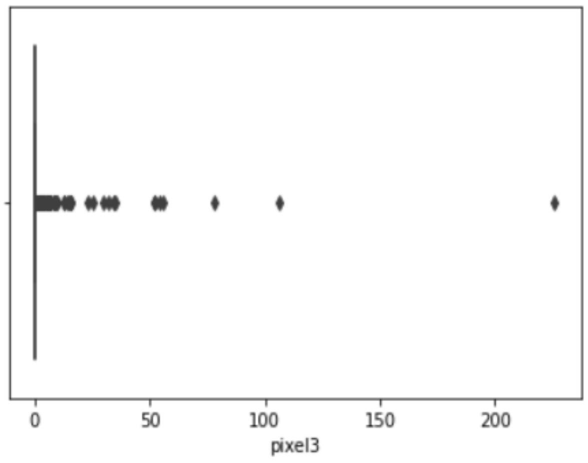
i. The graph shows pixel3's rows and rows' values range from 0 to 60000.
ii. The most value that repeated in this column is 50600 that happened 260 times.

c. *Boxplot diagram for pixel3 feature*

```
# Draw Boxplot for pixel3 feature to determine outliers
sns.boxplot('pixel3', data=X_train)
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators
word arg: x. From version 0.12, the only valid positional
t an explicit keyword will result in an error or misinterp
  FutureWarning
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe5734620d0>
```



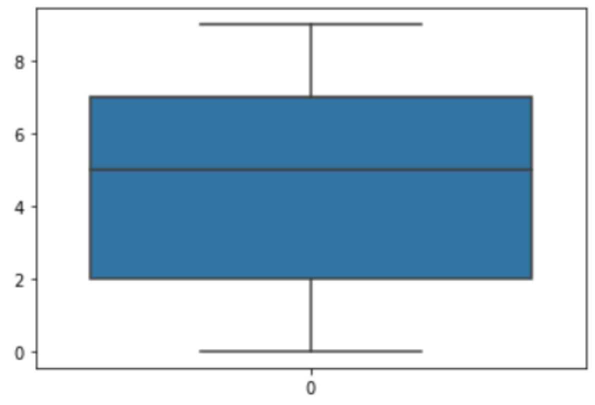
There are outliers that are more than 80

i. The graph shows pixel3's outliers that start from 80.

d. Boxplot diagram for y_train feature

```
# Draw Boxplot for y_train features to determine outliers
sns.boxplot(data=y_train)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe573424910>
```



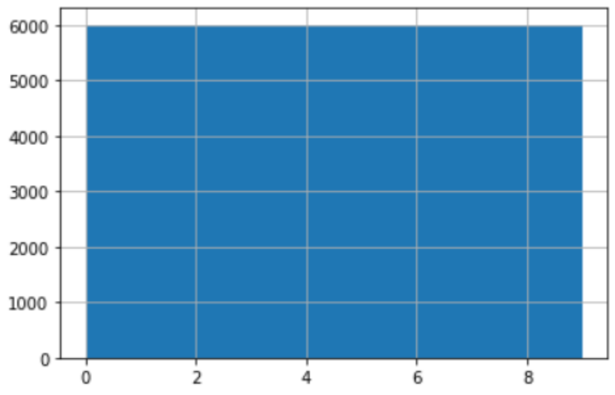
There is no outlier

i. There is no outlier.

e. Histogram for Label

```
y_train.hist(bins=10) # Draw histogram for label column
```

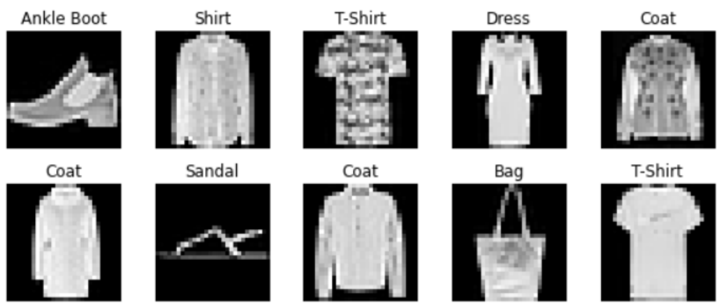
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe4fab65790>
```



6. Draw some of the images:

a. Draw some of the fashion minist using imshow function

```python
labels_map = {0: 'T-Shirt', 1: 'Trouser', 2: 'Pullover', 3: 'Dress', 4: 'Coat',
              5: 'Sandal', 6: 'Shirt', 7: 'Sneaker', 8: 'Bag', 9: 'Ankle Boot'} # Put a number on each label
# figure used to create a figure object
fig = plt.figure(figsize=(8, 8))

for i in range(1, 5 * 5+1):
    img = X_train_arr[i].reshape(28, 28) # respahe X_trian
    label = labels_map[y_train_arr[i]]  # To get each image with its label,

    fig.add_subplot(5, 5, i)
    plt.title(label)  # title of each image
    plt.imshow(img, cmap='gray')
    plt.axis('off') # remove axis numbers
plt.tight_layout()  # Padding between and surrounding subplots can be adjusted.
plt.show()
```



7. Correlation analysis

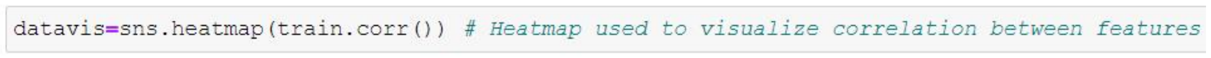
a. .corr() function used to know the relation between features.

```python
# Display the correlation between features by using corr function
X_train.corr()
```

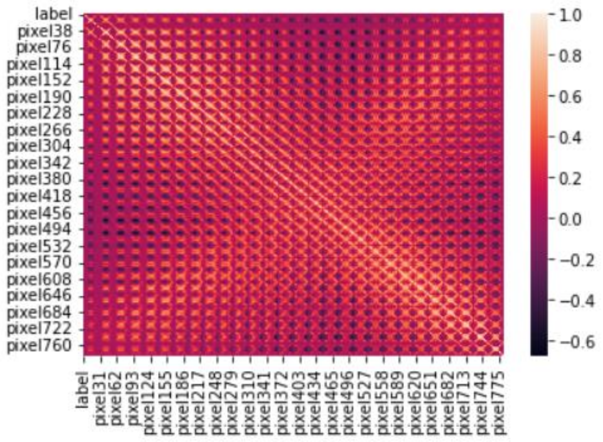
	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	pixel10	...	pixel775	pixel776
pixel1	1.000000	0.297899	0.067551	0.046607	0.026630	0.026172	0.012096	0.012225	0.009644	0.000056	...	-0.000641	0.004628
pixel2	0.297899	1.000000	0.575033	0.138709	0.054353	0.033184	0.022766	0.017138	0.016821	0.010920	...	0.000494	0.004861
pixel3	0.067551	0.575033	1.000000	0.387468	0.118136	0.087300	0.060937	0.035942	0.029674	0.021493	...	0.010095	0.016706
pixel4	0.046607	0.138709	0.387468	1.000000	0.573172	0.325683	0.242987	0.141033	0.085302	0.051147	...	0.009690	0.018710
pixel5	0.026630	0.054353	0.118136	0.573172	1.000000	0.692892	0.423635	0.230693	0.136391	0.075677	...	0.017999	0.031159
...	...	...	...	...	...	...	...	...	...	...	...	...	...
pixel780	-0.002442	-0.002341	-0.001494	0.010101	0.021514	0.032318	0.027065	0.019507	0.024986	0.029775	...	-0.074218	-0.008964
pixel781	-0.000109	0.004271	0.006861	0.023940	0.025971	0.038560	0.030478	0.026278	0.034245	0.039906	...	-0.042288	0.008640
pixel782	0.008764	0.014215	0.013151	0.012388	0.028859	0.044113	0.029461	0.022747	0.020991	0.016768	...	-0.002795	0.034103
pixel783	0.026388	0.021296	0.009946	0.003072	0.022939	0.030802	0.016128	0.005947	0.000090	-0.011062	...	0.001906	0.026552
pixel784	0.041581	0.022161	0.015657	0.008422	0.007124	0.004948	0.001792	0.000594	0.000285	-0.002939	...	0.016611	0.028407

784 rows × 784 columns

b. Heatmap to draw .corr() function



```
datavis=sns.heatmap(train.corr()) # Heatmap used to visualize correlation between features
```



i. The correlation between a feature and itself equals 1.

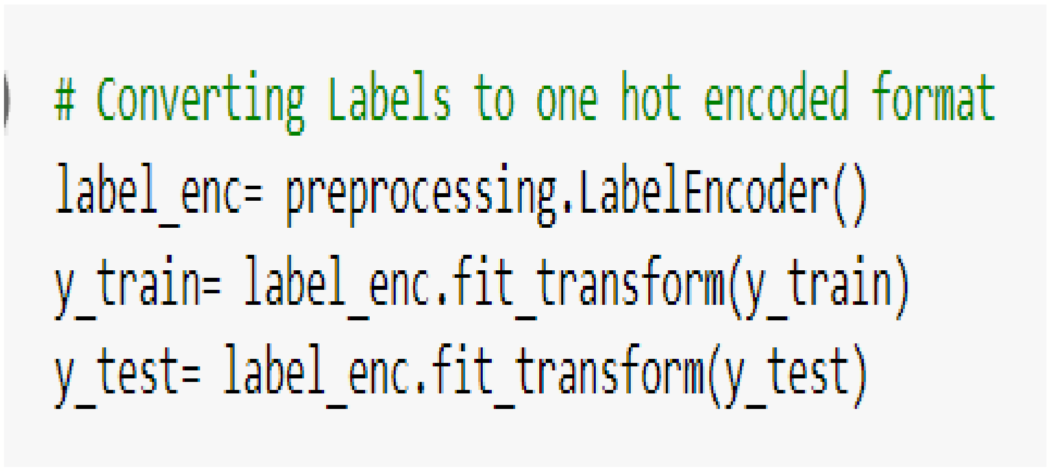
ii. The correlation between features and each other is a negative coefficient, which tells us that the relationship is negative, and a positive coefficient, which tells us that the relationship is positive.

iii. There are large correlation between pixel 781 and pixel 780, and between pixel 782 and pixel 783.

iv. There are large correlation between pixel 4 and pixel 783, and between pixel 9 and pixel 783.

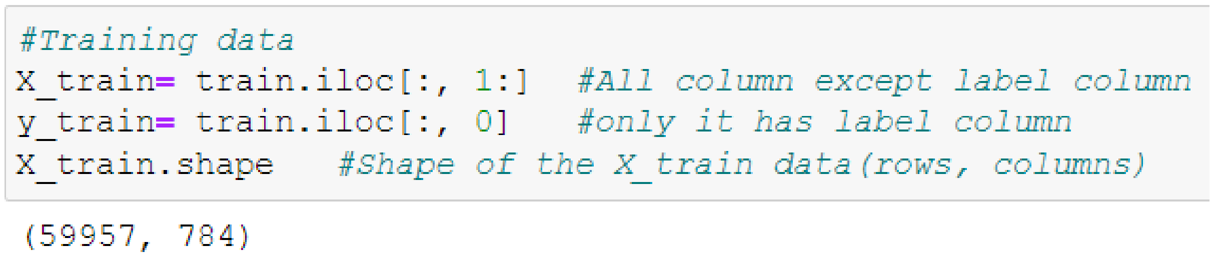
8. Label Encoder

a. We do label encoder to the data label column.



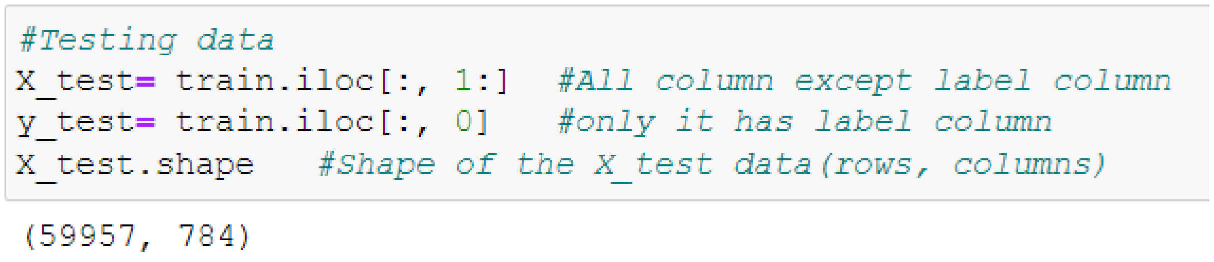
```
# Converting Labels to one hot encoded format
label_enc= preprocessing.LabelEncoder()
y_train= label_enc.fit_transform(y_train)
y_test= label_enc.fit_transform(y_test)
```

9. Split data
    a. Split data into training and testing data by using iloc
    b. X will contain all features except label column.
    c. y will contain label column.



```
#Training data
X_train= train.iloc[:, 1:]   #All column except label column
y_train= train.iloc[:, 0]    #only it has label column
X_train.shape    #Shape of the X_train data(rows, columns)
```

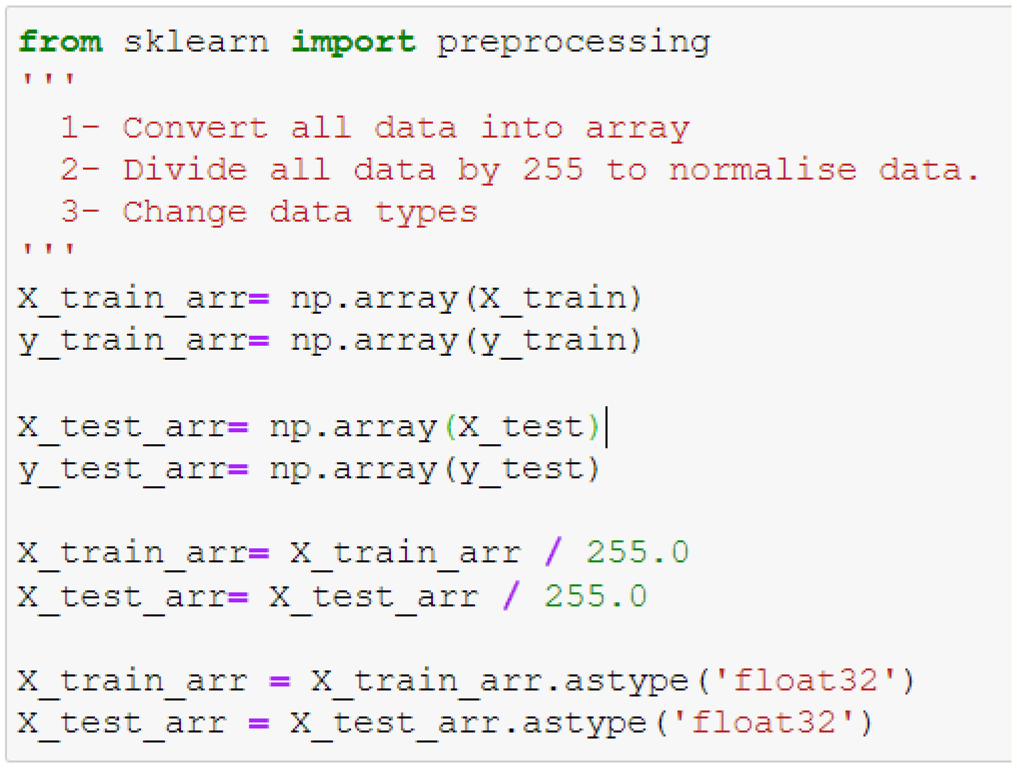
```
(59957, 784)
```



```
#Testing data
X_test= train.iloc[:, 1:]   #All column except label column
y_test= train.iloc[:, 0]    #only it has label column
X_test.shape    #Shape of the X_test data(rows, columns)
```

```
(59957, 784)
```

10. Normalizing data
    a. Normalize data to ensure all data have the same range of values.

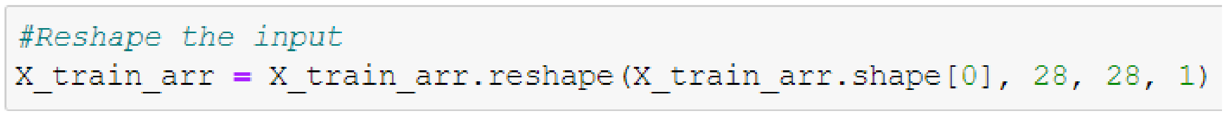


```
from sklearn import preprocessing
'''
  1- Convert all data into array
  2- Divide all data by 255 to normalise data.
  3- Change data types
'''
X_train_arr= np.array(X_train)
y_train_arr= np.array(y_train)

X_test_arr= np.array(X_test)
y_test_arr= np.array(y_test)

X_train_arr= X_train_arr / 255.0
X_test_arr= X_test_arr / 255.0

X_train_arr = X_train_arr.astype('float32')
X_test_arr = X_test_arr.astype('float32')
```

Training a CNN neural network

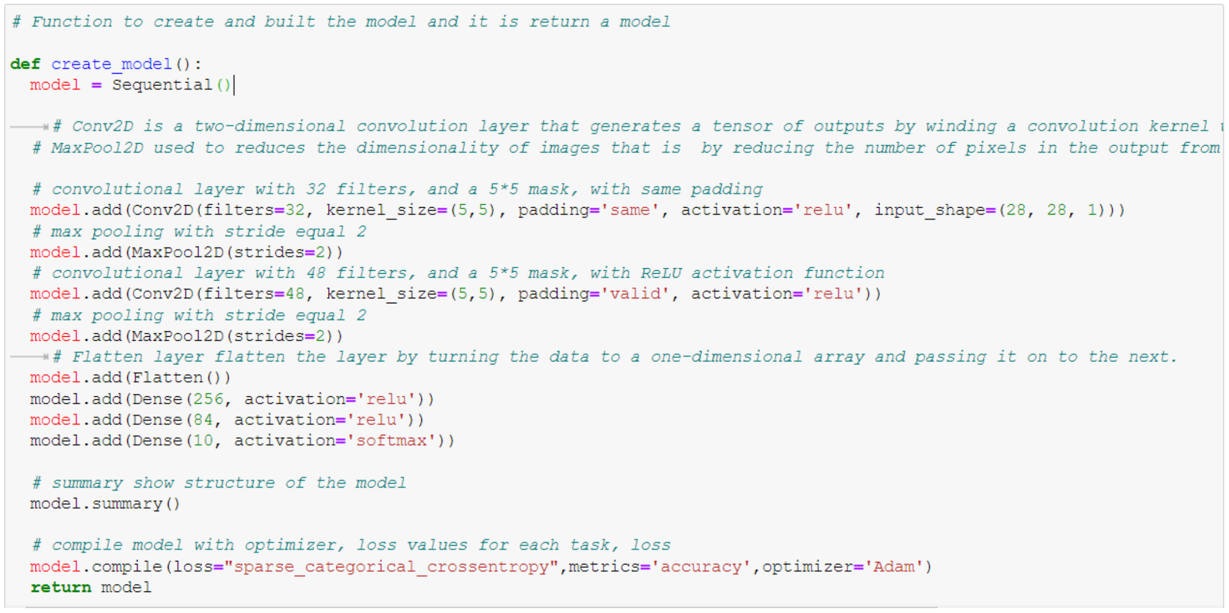
- In this section, we will try different hyper parameters, then we will do many trials to Show if the model improves the accuracy or not.

- Reshape the input



```
#Reshape the input
X_train_arr = X_train_arr.reshape(X_train_arr.shape[0], 28, 28, 1)
```

- Function to create and built the model and it is return a model



```
# Function to create and built the model and it is return a model

def create_model():
  model = Sequential()

    # Conv2D is a two-dimensional convolution layer that generates a tensor of outputs by winding a convolution kernel w
  # MaxPool2D used to reduces the dimensionality of images that is  by reducing the number of pixels in the output from

  # convolutional layer with 32 filters, and a 5*5 mask, with same padding
  model.add(Conv2D(filters=32, kernel_size=(5,5), padding='same', activation='relu', input_shape=(28, 28, 1)))
  # max pooling with stride equal 2
  model.add(MaxPool2D(strides=2))
  # convolutional layer with 48 filters, and a 5*5 mask, with ReLU activation function
  model.add(Conv2D(filters=48, kernel_size=(5,5), padding='valid', activation='relu'))
  # max pooling with stride equal 2
  model.add(MaxPool2D(strides=2))
    # Flatten layer flatten the layer by turning the data to a one-dimensional array and passing it on to the next.
  model.add(Flatten())
  model.add(Dense(256, activation='relu'))
  model.add(Dense(84, activation='relu'))
  model.add(Dense(10, activation='softmax'))

  # summary show structure of the model
  model.summary()

  # compile model with optimizer, loss values for each task, loss
  model.compile(loss="sparse_categorical_crossentropy",metrics='accuracy',optimizer='Adam')
  return model
```

- Function to evaluate a model using k-fold cross-validation:

  - Inputs:

    1- X data

    2- y data

    3- Number of k folds

    4- Number of batch size

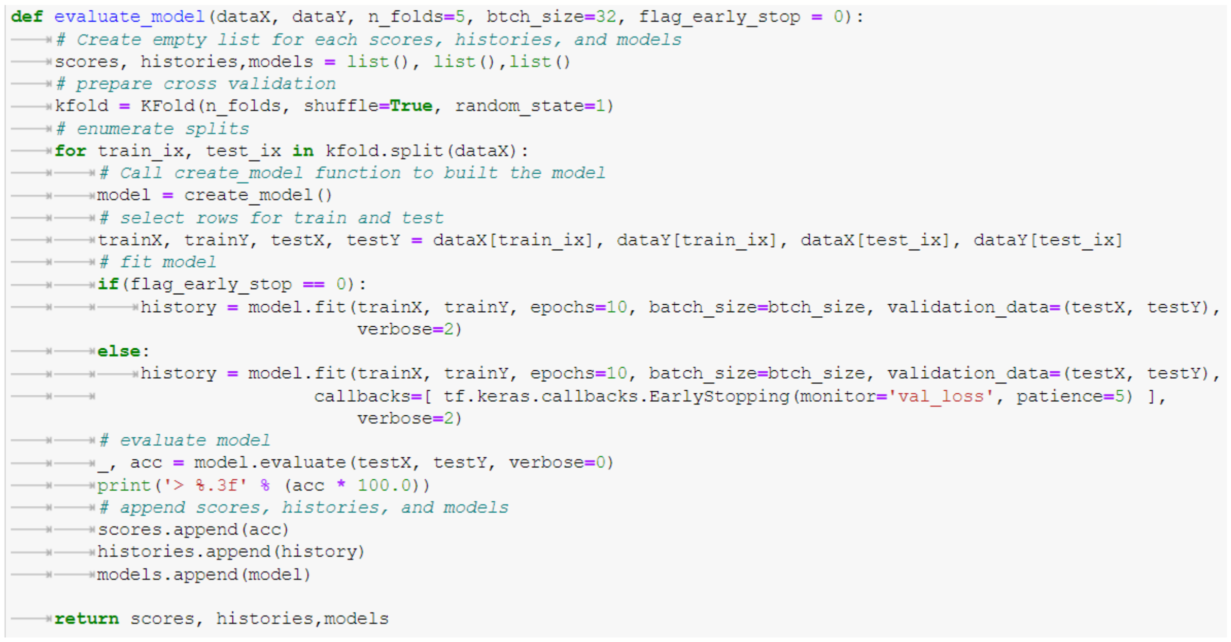
    5- Booling variable for early stopping(if 1 means the model have early stopping of zero means the model have not early stopping)

  - Return (Outputs): lists of all k folds

    1- Scores

    2- Histories

    3- Models



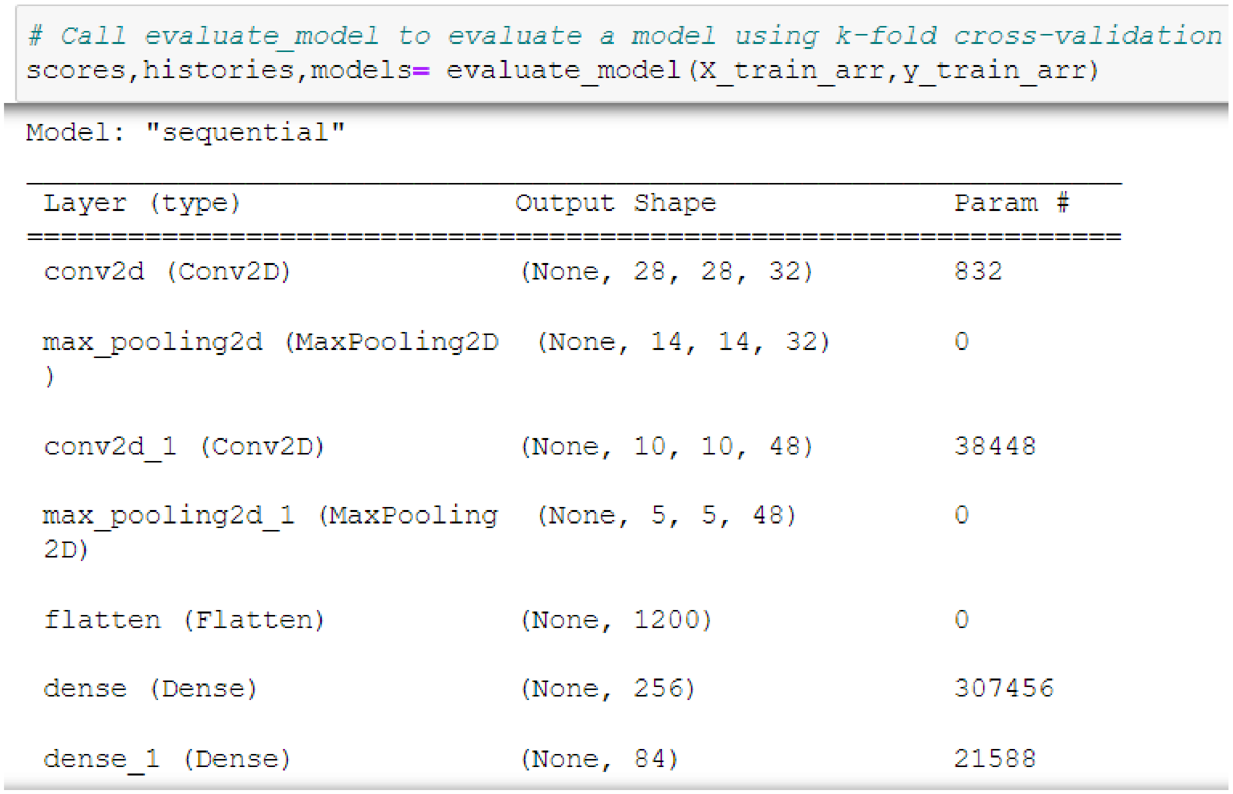
```python
def evaluate_model(dataX, dataY, n_folds=5, btch_size=32, flag_early_stop = 0):
    # Create empty list for each scores, histories, and models
    scores, histories,models = list(), list(),list()
    # prepare cross validation
    kfold = KFold(n_folds, shuffle=True, random_state=1)
    # enumerate splits
    for train_ix, test_ix in kfold.split(dataX):
        # Call create_model function to built the model
        model = create_model()
        # select rows for train and test
        trainX, trainY, testX, testY = dataX[train_ix], dataY[train_ix], dataX[test_ix], dataY[test_ix]
        # fit model
        if(flag_early_stop == 0):
            history = model.fit(trainX, trainY, epochs=10, batch_size=btch_size, validation_data=(testX, testY),
                                verbose=2)
        else:
            history = model.fit(trainX, trainY, epochs=10, batch_size=btch_size, validation_data=(testX, testY),
                            callbacks=[ tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5) ],
                                verbose=2)
        # evaluate model
        _, acc = model.evaluate(testX, testY, verbose=0)
        print('> %.3f' % (acc * 100.0))
        # append scores, histories, and models
        scores.append(acc)
        histories.append(history)
        models.append(model)

    return scores, histories,models
```

- Before these trials, we tried different hyper parameters, but it was these hyper parameters that gave us the best accuracy.

  The following trials Show if the model improves the accuracy or not.
- The first trial we will call the evaluate_model function, but without early stopping.

1- Trial_1

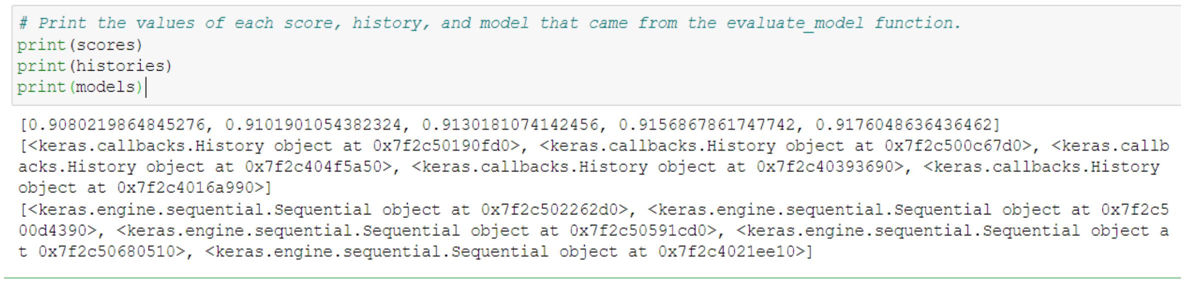
- Call evaluate_model to evaluate a model using k-fold cross-validation



```
# Call evaluate_model to evaluate a model using k-fold cross-validation
scores,histories,models= evaluate_model(X_train_arr,y_train_arr)
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 28, 28, 32)        832

 max_pooling2d (MaxPooling2D  (None, 14, 14, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 10, 10, 48)        38448

 max_pooling2d_1 (MaxPooling  (None, 5, 5, 48)         0
 2D)

 flatten (Flatten)           (None, 1200)              0

 dense (Dense)               (None, 256)               307456

 dense_1 (Dense)             (None, 84)                21588
```

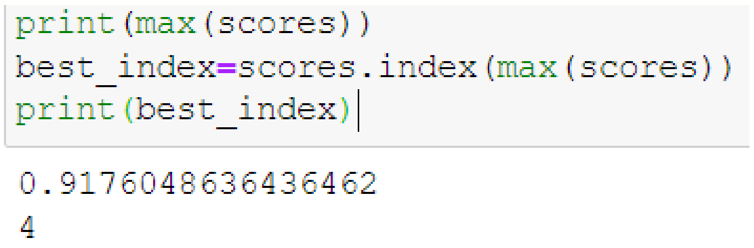
- Print the values of each score, history, and model that came from the evaluate_model function.



```
# Print the values of each score, history, and model that came from the evaluate_model function.
print(scores)
print(histories)
print(models)
```

```
[0.9080219864845276, 0.9101901054382324, 0.9130181074142456, 0.9156867861747742, 0.9176048636436462]
[<keras.callbacks.History object at 0x7f2c50190fd0>, <keras.callbacks.History object at 0x7f2c500c67d0>, <keras.callb
acks.History object at 0x7f2c404f5a50>, <keras.callbacks.History object at 0x7f2c40393690>, <keras.callbacks.History
object at 0x7f2c4016a990>]
[<keras.engine.sequential.Sequential object at 0x7f2c502262d0>, <keras.engine.sequential.Sequential object at 0x7f2c5
00d4390>, <keras.engine.sequential.Sequential object at 0x7f2c50591cd0>, <keras.engine.sequential.Sequential object a
t 0x7f2c50680510>, <keras.engine.sequential.Sequential object at 0x7f2c4021ee10>]
```

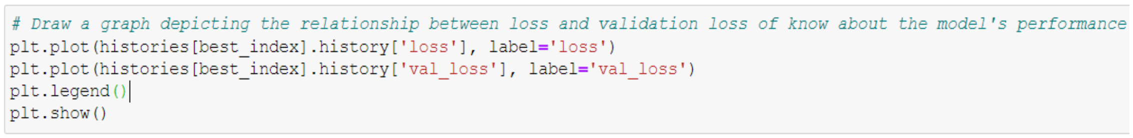
- We need the index of the best model
- Use max to get the best model then we get the index of the best model.



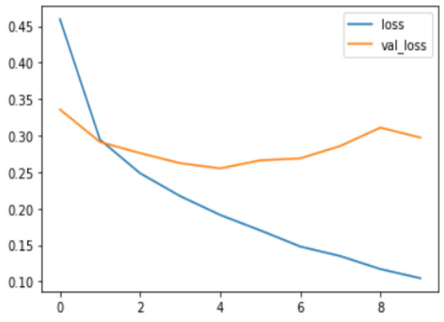
```
print(max(scores))
best_index=scores.index(max(scores))
print(best_index)
```

```
0.9176048636436462
4
```

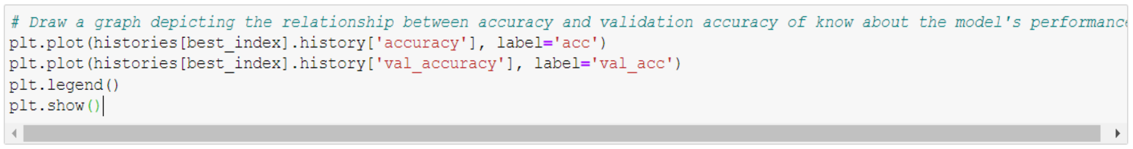
- Draw a graph depicting the relationship between loss and validation loss of know about the model's performance



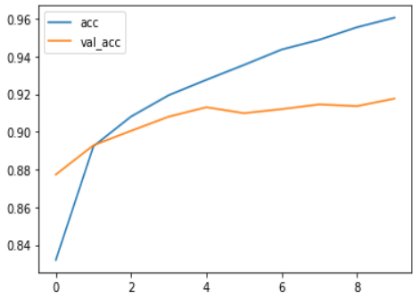
```
# Draw a graph depicting the relationship between loss and validation loss of know about the model's performance
plt.plot(histories[best_index].history['loss'], label='loss')
plt.plot(histories[best_index].history['val_loss'], label='val_loss')
plt.legend()
plt.show()
```



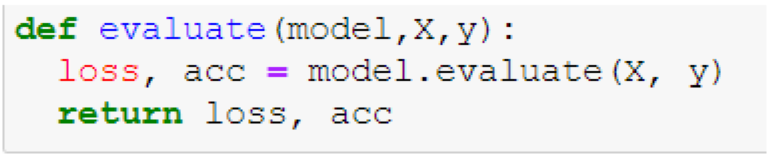
- Draw a graph depicting the relationship between accuracy and validation accuracy of know about the model's performance



```
# Draw a graph depicting the relationship between accuracy and validation accuracy of know about the model's performanc
plt.plot(histories[best_index].history['accuracy'], label='acc')
plt.plot(histories[best_index].history['val_accuracy'], label='val_acc')
plt.legend()
plt.show()
```

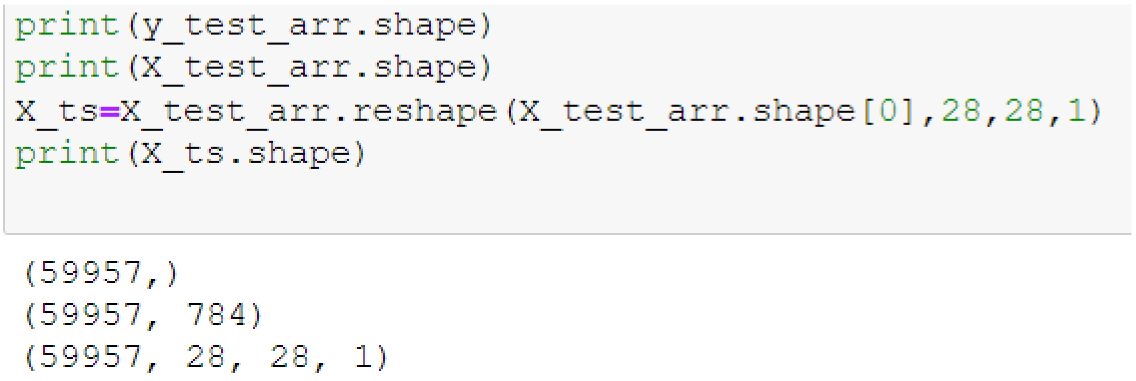


- Function to evaluate the model then return the value of loss and accuracy



```
def evaluate(model,X,y):
    loss, acc = model.evaluate(X, y)
    return loss, acc
```

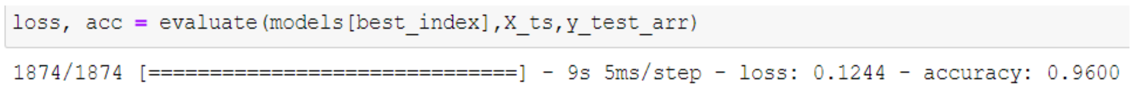
- Reshape x test to have the same shape as input_shape



```
print(y_test_arr.shape)
print(X_test_arr.shape)
X_ts=X_test_arr.reshape(X_test_arr.shape[0],28,28,1)
print(X_ts.shape)
```

```
(59957,)
(59957, 784)
(59957, 28, 28, 1)
```

- Call evaluate to evaluate the model

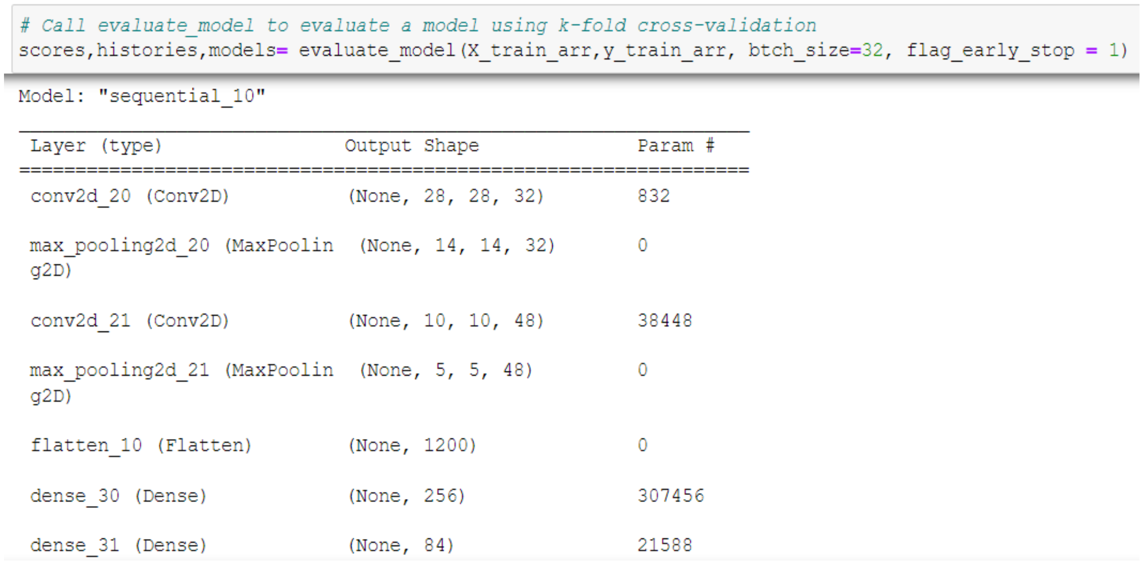


```
loss, acc = evaluate(models[best_index],X_ts,y_test_arr)
```

```
1874/1874 [==============================] - 9s 5ms/step - loss: 0.1244 - accuracy: 0.9600
```

2- Trial_2

- The second trial we will call the evaluate_model function, but we will use early stopping to reduce over fitting that exists in the model.



```
# Call evaluate_model to evaluate a model using k-fold cross-validation
scores,histories,models= evaluate_model(X_train_arr,y_train_arr, btch_size=32, flag_early_stop = 1)
```

```
Model: "sequential_10"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_20 (Conv2D)          (None, 28, 28, 32)        832

 max_pooling2d_20 (MaxPoolin  (None, 14, 14, 32)       0
 g2D)

 conv2d_21 (Conv2D)          (None, 10, 10, 48)        38448

 max_pooling2d_21 (MaxPoolin  (None, 5, 5, 48)         0
 g2D)

 flatten_10 (Flatten)        (None, 1200)              0

 dense_30 (Dense)            (None, 256)               307456

 dense_31 (Dense)            (None, 84)                21588
```

- Print the values of each score, history, and model that came from the evaluate_model function.

```
print(scores)
print(histories)
print(models)
```

```
[0.9148598909378052, 0.9191127419471741, 0.9136852622032166, 0.9104328155517578, 0.9142690300941467]
[<keras.callbacks.History object at 0x7f2baa0e0d50>, <keras.callbacks.History object at 0x7f2b23702050>, <keras.callb
acks.History object at 0x7f2b23473290>, <keras.callbacks.History object at 0x7f2b232f5b50>, <keras.callbacks.History
object at 0x7f2b230e1bd0>]
[<keras.engine.sequential.Sequential object at 0x7f2baa12b690>, <keras.engine.sequential.Sequential object at 0x7f2b2
36dca50>, <keras.engine.sequential.Sequential object at 0x7f2b236287d0>, <keras.engine.sequential.Sequential object a
t 0x7f2b2337b8d0>, <keras.engine.sequential.Sequential object at 0x7f2b232f56d0>]
```

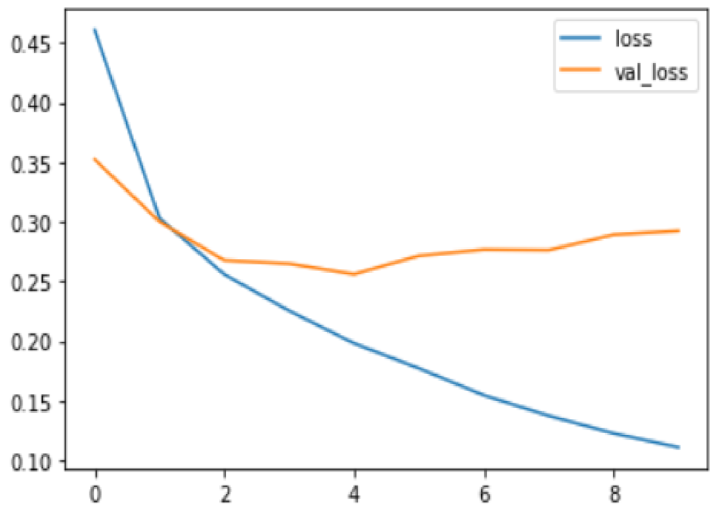
- We need the index of the best model
- Use max to get the best model then we get the index of the best model.

```
print(max(scores))
best_index=scores.index(max(scores))
print(best_index)
```

```
0.9191127419471741
1
```

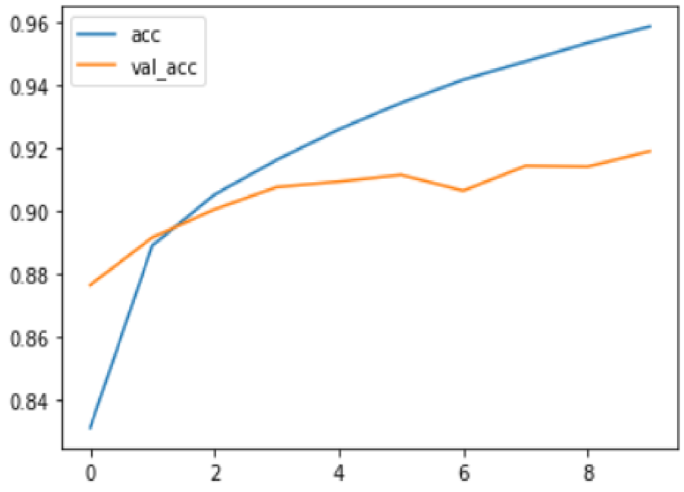
- Draw a graph depicting the relationship between loss and validation loss of know about the model's performance

```
plt.plot(histories[best_index].history['loss'], label='loss')
plt.plot(histories[best_index].history['val_loss'], label='val_loss')
plt.legend()
plt.show()
```



- Draw a graph depicting the relationship between accuracy and validation accuracy of know about the model's performance

```
plt.plot(histories[best_index].history['accuracy'], label='acc')
plt.plot(histories[best_index].history['val_accuracy'], label='val_acc')
plt.legend()
plt.show()
```



- Function to evaluate the model then return the value of loss and accuracy

```
def evaluate(model,X,y):
  loss, acc = model.evaluate(X, y)
  return loss, acc
```

- Reshape x test to have the same shape as input_shape

```
print(y_test_arr.shape)
print(X_test_arr.shape)
X_ts=X_test_arr.reshape(X_test_arr.shape[0],28,28,1)
print(X_ts.shape)
```

```
(59957,)
(59957, 784)
(59957, 28, 28, 1)
```

- Call evaluate to evaluate the model

```
loss, acc =evaluate(models[best_index],X_ts,y_test_arr)
```

```
1874/1874 [==============================] - 5s 3ms/step - loss: 0.1165 - accuracy: 0.9631
```

➢ From previous trials:

Trial 1: loss: 0.1244 - accuracy: 0.9600

Trial 2 : loss: 0.1165 - accuracy: 0.9631

So LeNet-5 improves the accuracy as we reduce over fitting and the changing of hyper parameters means improved accuracy.

Transfer Learning

General code for all model that we will use it

- Converting Labels to categorical encoded format

```
# Converting Labels to categorical encoded format
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
```

- dstack is a command that stacks arrays in depth order.
- Used to convert image from gray scale into RGB

```
x_train_model=np.dstack([X_train_arr] * 3)
x_test_model=np.dstack([X_test_arr]*3)
```

- Show the shape of the x train

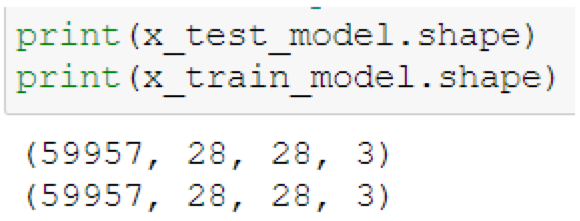
```
x_train_model.shape
```

```
(59957, 784, 3)
```

- Reshape x train and x test into 28*28

```
x_train_model = x_train_model.reshape(-1, 28,28,3)
x_test_model= x_test_model.reshape (-1,28,28,3)
```

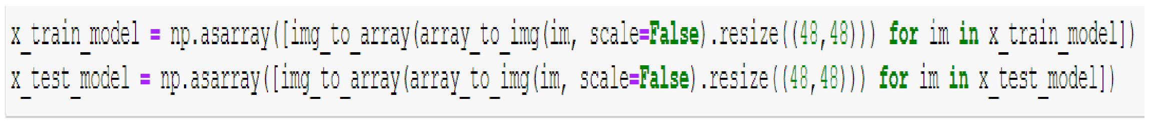
- Show the shape of the x test and x train



```
print(x_test_model.shape)
print(x_train_model.shape)
```

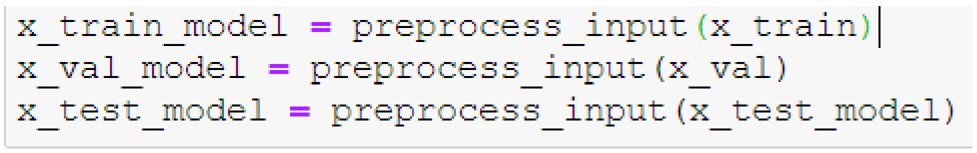
```
(59957, 28, 28, 3)
(59957, 28, 28, 3)
```

- Resize the training and testing images into 48 * 48



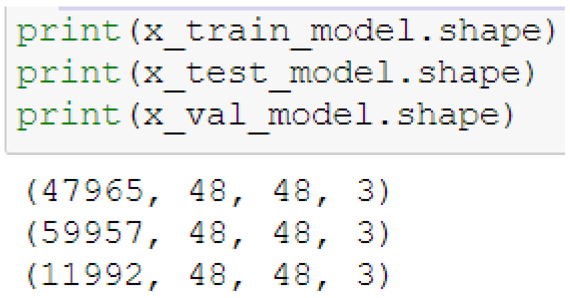
```
x_train_model = np.asarray([img_to_array(array_to_img(im, scale=False).resize((48,48))) for im in x_train_model])
x_test_model = np.asarray([img_to_array(array_to_img(im, scale=False).resize((48,48))) for im in x_test_model])
```

- Convert the image to a model-appropriate format



```
x_train_model = preprocess_input(x_train)
x_val_model = preprocess_input(x_val)
x_test_model = preprocess_input(x_test_model)
```

- Current shape of features



```
print(x_train_model.shape)
print(x_test_model.shape)
print(x_val_model.shape)
```

```
(47965, 48, 48, 3)
(59957, 48, 48, 3)
(11992, 48, 48, 3)
```

1- VGG Model

- Create the base model of VGG16

```
#Create base model of VGG16
model_vgg = VGG16(weights="imagenet", include_top=False, input_shape=(48, 48, 3))
model_vgg.summary()
```

```
Model: "vgg16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 48, 48, 3)]       0

 block1_conv1 (Conv2D)       (None, 48, 48, 64)        1792

 block1_conv2 (Conv2D)       (None, 48, 48, 64)        36928

 block1_pool (MaxPooling2D)  (None, 24, 24, 64)        0

 block2_conv1 (Conv2D)       (None, 24, 24, 128)       73856

 block2_conv2 (Conv2D)       (None, 24, 24, 128)       147584

 block2_pool (MaxPooling2D)  (None, 12, 12, 128)       0

 block3_conv1 (Conv2D)       (None, 12, 12, 256)       295168

 block3_conv2 (Conv2D)       (None, 12, 12, 256)       590080

 block3_conv3 (Conv2D)       (None, 12, 12, 256)       590080

 block3_pool (MaxPooling2D)  (None, 6, 6, 256)         0

 block4_conv1 (Conv2D)       (None, 6, 6, 512)         1180160

 block4_conv2 (Conv2D)       (None, 6, 6, 512)         2359808
```

- Extracting important features from x train, x test, and x validation image

```
x_train_vgg = model_vgg.predict(np.array(x_train_model), batch_size=16, verbose=1)
x_test_vgg = model_vgg.predict(np.array(x_test_model), batch_size=16, verbose=1)
x_val_vgg = model_vgg.predict(np.array(x_val_model), batch_size=16, verbose=1)
```

```
2998/2998 [==============================] - 72s 21ms/step
3748/3748 [==============================] - 80s 21ms/step
750/750 [==============================] - 17s 22ms/step
```

- Current shape of features

```
# Current shape of features
print(x_train_vgg.shape)
print(x_test_vgg.shape)
print(x_val_vgg.shape)
```

```
(47965, 1, 1, 512)
(59957, 1, 1, 512)
(11992, 1, 1, 512)
```

- Reshape extracted features

```
x_train_vgg = np.reshape(x_train_vgg, (47965, 1*1*512))
x_test_vgg = np.reshape(x_test_vgg, (59957, 1*1*512))
x_val_vgg = np.reshape(x_val_vgg, (11992, 1*1*512))
```

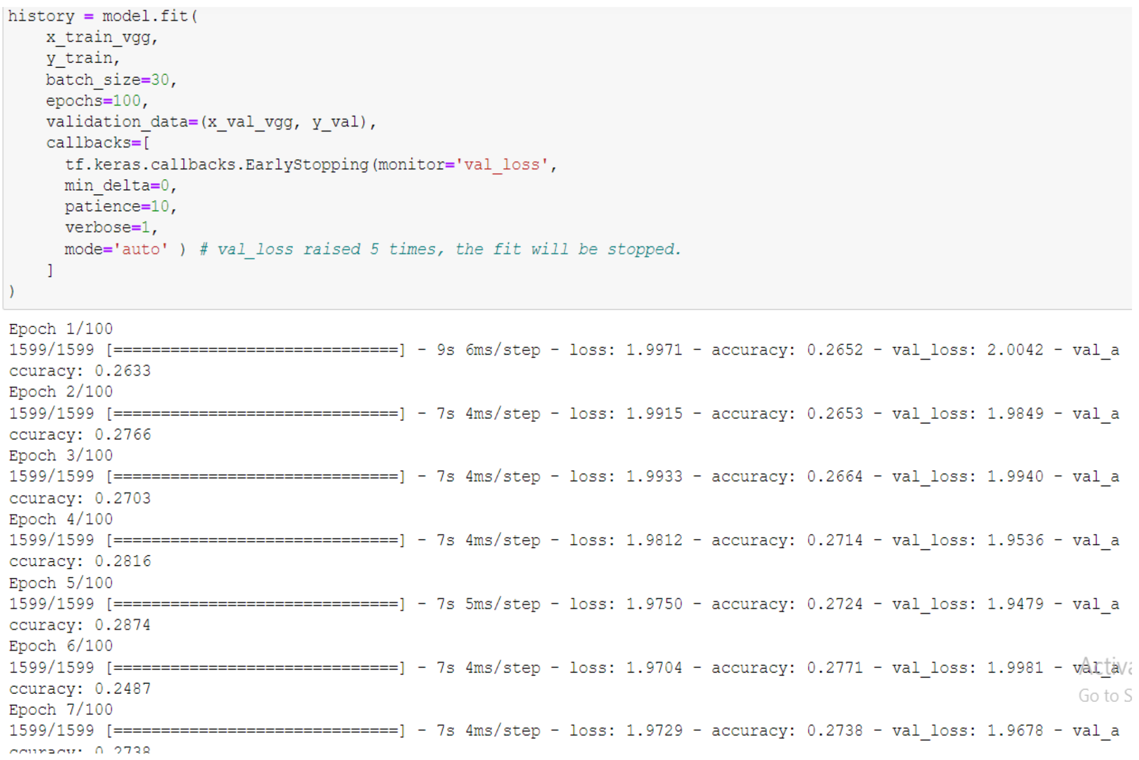
- Do freeze the weight that moves all the layer's weights from trainable to non-trainable.

```
for layer in model_vgg.layers:
      layer.trainable = False

model = Sequential()
# Add new layers
model.add(Dense(256, activation='relu', input_dim=(1*1*512)))
model.add(Dense(128, activation='relu'))
model.add(Dense(10, activation='softmax'))
```

- Compile the model.

```
# Compile the model.
model.compile(loss='categorical_crossentropy', optimizer='Adam',
    metrics=['accuracy'])
```

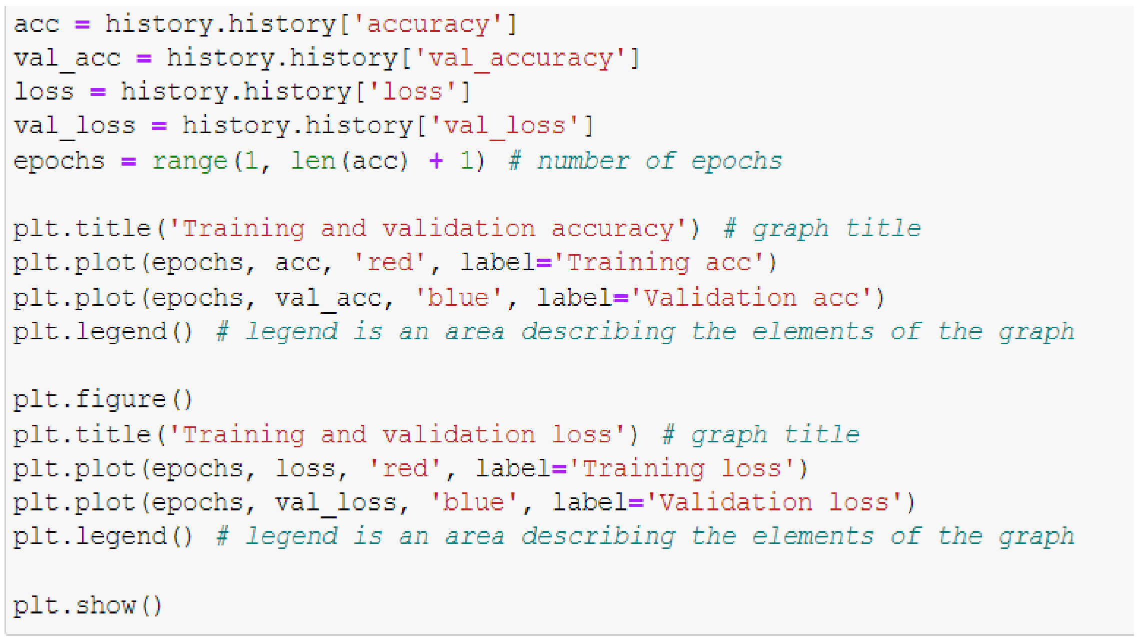
- Train the model



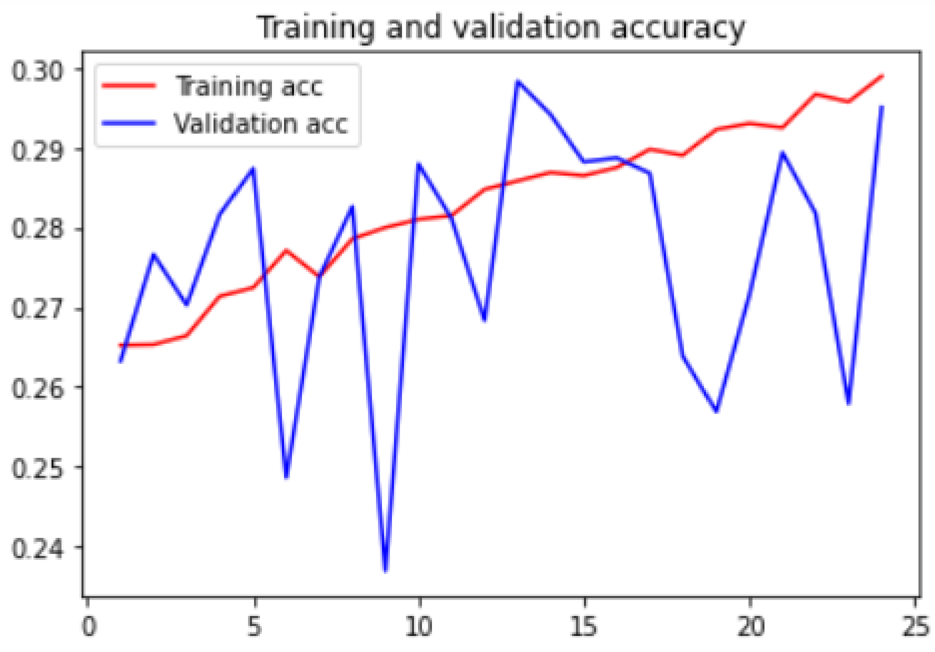
```
history = model.fit(
    x_train_vgg,
    y_train,
    batch_size=30,
    epochs=100,
    validation_data=(x_val_vgg, y_val),
    callbacks=[
      tf.keras.callbacks.EarlyStopping(monitor='val_loss',
      min_delta=0,
      patience=10,
      verbose=1,
      mode='auto' ) # val_loss raised 5 times, the fit will be stopped.
    ]
)
```

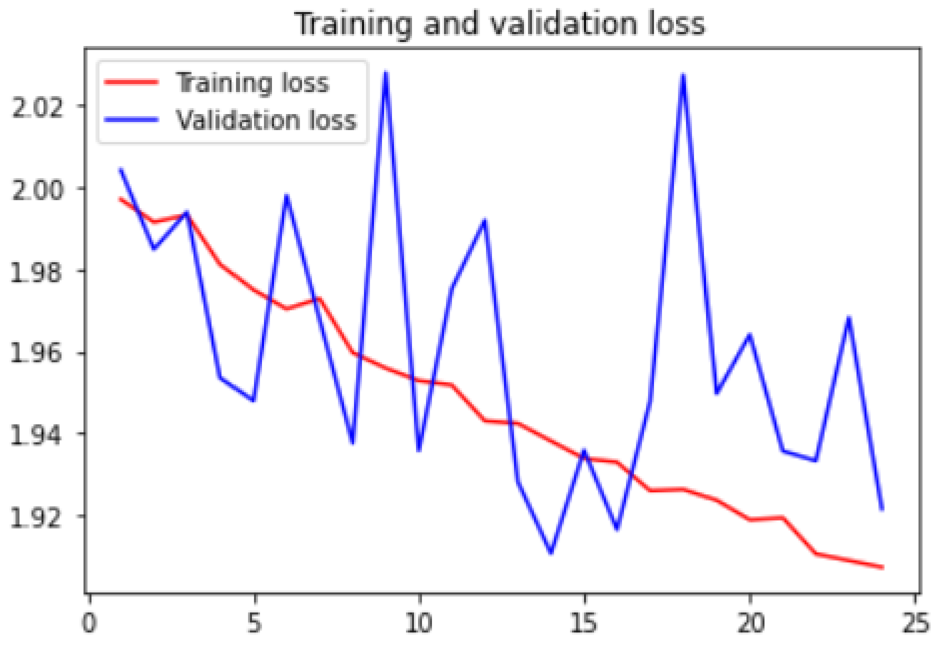
```
Epoch 1/100
1599/1599 [==============================] - 9s 6ms/step - loss: 1.9971 - accuracy: 0.2652 - val_loss: 2.0042 - val_a
ccuracy: 0.2633
Epoch 2/100
1599/1599 [==============================] - 7s 4ms/step - loss: 1.9915 - accuracy: 0.2653 - val_loss: 1.9849 - val_a
ccuracy: 0.2766
Epoch 3/100
1599/1599 [==============================] - 7s 4ms/step - loss: 1.9933 - accuracy: 0.2664 - val_loss: 1.9940 - val_a
ccuracy: 0.2703
Epoch 4/100
1599/1599 [==============================] - 7s 4ms/step - loss: 1.9812 - accuracy: 0.2714 - val_loss: 1.9536 - val_a
ccuracy: 0.2816
Epoch 5/100
1599/1599 [==============================] - 7s 5ms/step - loss: 1.9750 - accuracy: 0.2724 - val_loss: 1.9479 - val_a
ccuracy: 0.2874
Epoch 6/100
1599/1599 [==============================] - 7s 4ms/step - loss: 1.9704 - accuracy: 0.2771 - val_loss: 1.9981 - val_a
ccuracy: 0.2487
Epoch 7/100
1599/1599 [==============================] - 7s 4ms/step - loss: 1.9729 - accuracy: 0.2738 - val_loss: 1.9678 - val_a
ccuracy: 0.2738
```

- plot the loss and accuracy



```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1) # number of epochs

plt.title('Training and validation accuracy') # graph title
plt.plot(epochs, acc, 'red', label='Training acc')
plt.plot(epochs, val_acc, 'blue', label='Validation acc')
plt.legend() # legend is an area describing the elements of the graph

plt.figure()
plt.title('Training and validation loss') # graph title
plt.plot(epochs, loss, 'red', label='Training loss')
plt.plot(epochs, val_loss, 'blue', label='Validation loss')
plt.legend() # legend is an area describing the elements of the graph

plt.show()
```





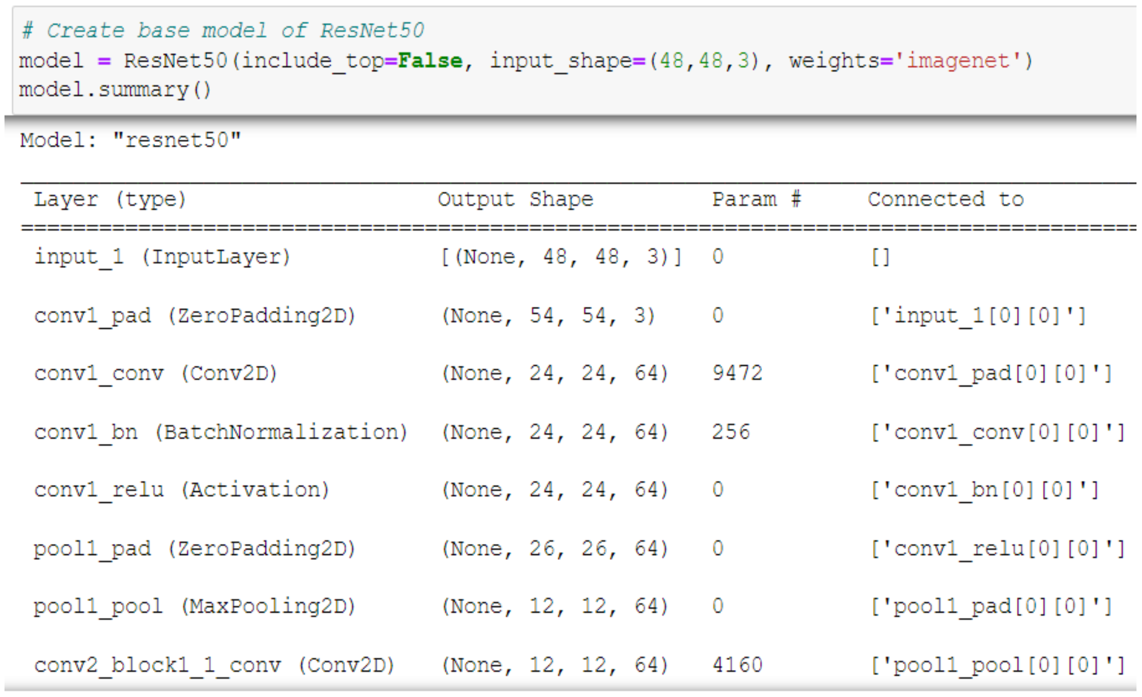
- Evaluate the model

```
# Evaluate the model
score = model.evaluate(x_test_vgg, y_test)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
1874/1874 [==============================] - 6s 3ms/step - loss: 1.9051 - accuracy: 0.2983
Test loss: 1.905113935470581
Test accuracy: 0.29831379652023315
```

2- ResNet Model

- Create base model of ResNet50

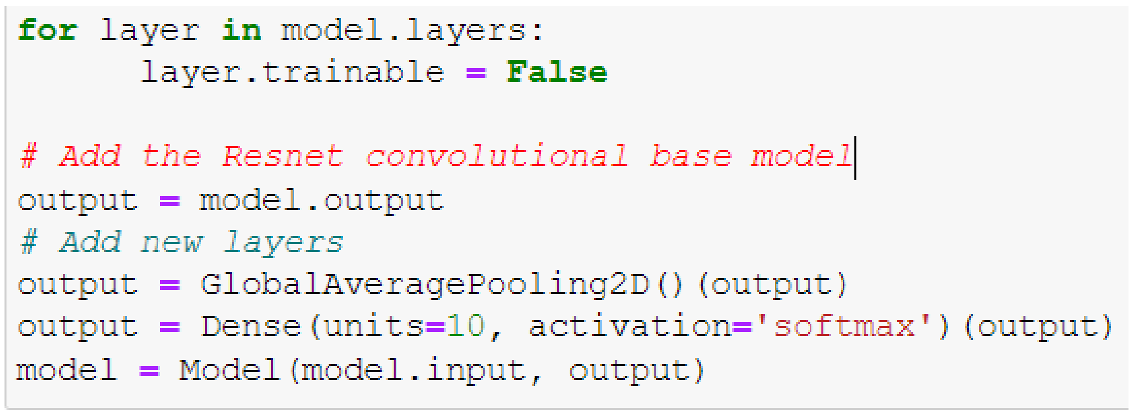


```
# Create base model of ResNet50
model = ResNet50(include_top=False, input_shape=(48,48,3), weights='imagenet')
model.summary()
```

```
Model: "resnet50"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_1 (InputLayer)           [(None, 48, 48, 3)]  0           []

 conv1_pad (ZeroPadding2D)      (None, 54, 54, 3)    0           ['input_1[0][0]']

 conv1_conv (Conv2D)            (None, 24, 24, 64)   9472        ['conv1_pad[0][0]']

 conv1_bn (BatchNormalization)  (None, 24, 24, 64)   256         ['conv1_conv[0][0]']

 conv1_relu (Activation)        (None, 24, 24, 64)   0           ['conv1_bn[0][0]']

 pool1_pad (ZeroPadding2D)      (None, 26, 26, 64)   0           ['conv1_relu[0][0]']

 pool1_pool (MaxPooling2D)      (None, 12, 12, 64)   0           ['pool1_pad[0][0]']

 conv2_block1_1_conv (Conv2D)   (None, 12, 12, 64)   4160        ['pool1_pool[0][0]']
```

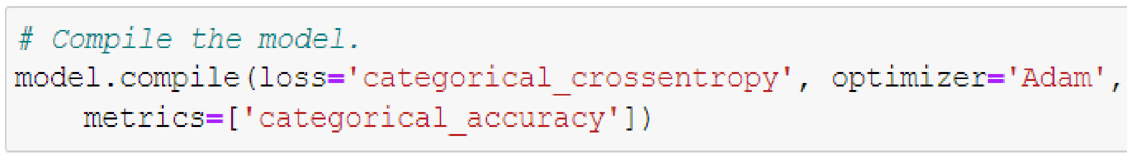
`

- Do freeze the weight that moves all the layer's weights from trainable to non-trainable.



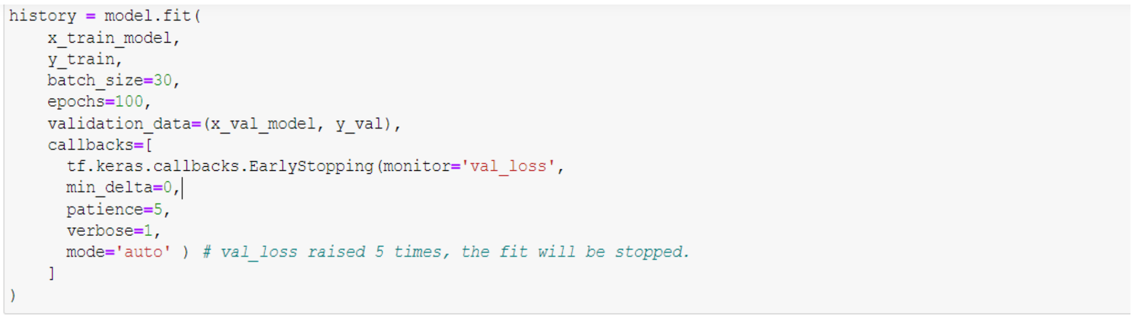
```
for layer in model.layers:
      layer.trainable = False

# Add the Resnet convolutional base model
output = model.output
# Add new layers
output = GlobalAveragePooling2D()(output)
output = Dense(units=10, activation='softmax')(output)
model = Model(model.input, output)
```

- Compile the model.

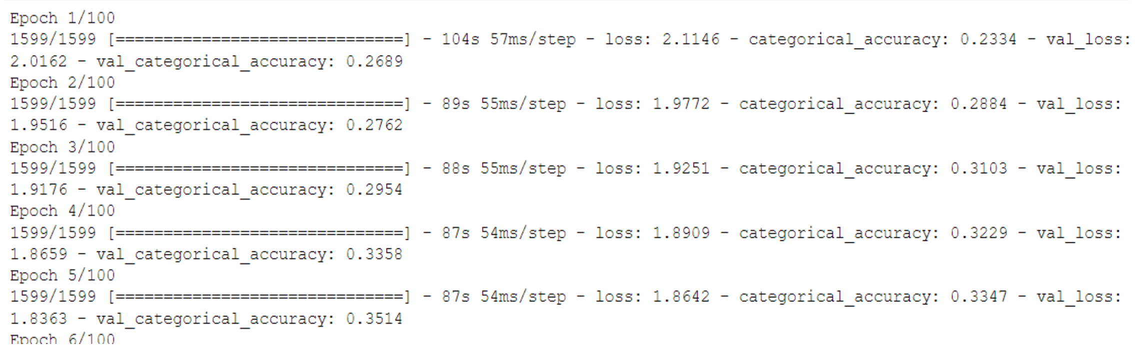


```
# Compile the model.
model.compile(loss='categorical_crossentropy', optimizer='Adam',
    metrics=['categorical_accuracy'])
```

- Train the model

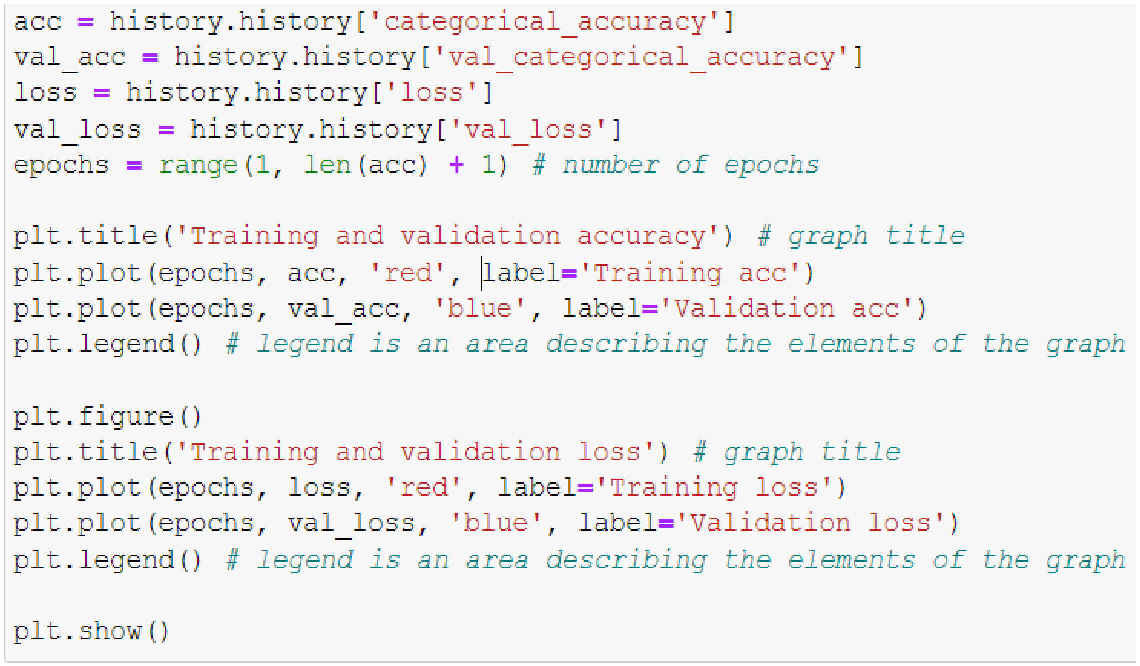


```
history = model.fit(
    x_train_model,
    y_train,
    batch_size=30,
    epochs=100,
    validation_data=(x_val_model, y_val),
    callbacks=[
      tf.keras.callbacks.EarlyStopping(monitor='val_loss',
      min_delta=0,
      patience=5,
      verbose=1,
      mode='auto' ) # val_loss raised 5 times, the fit will be stopped.
    ]
)
```

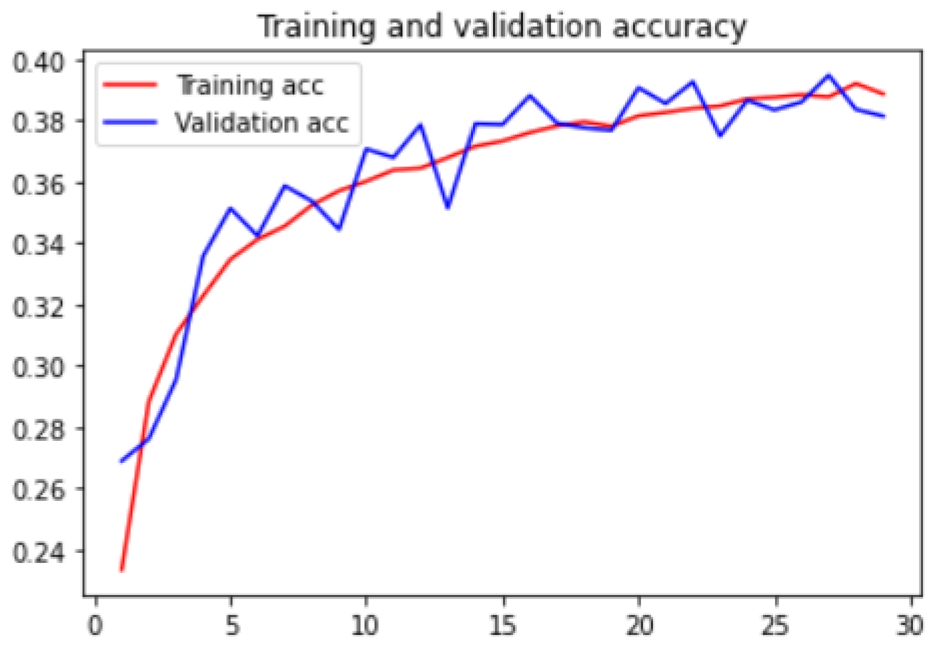


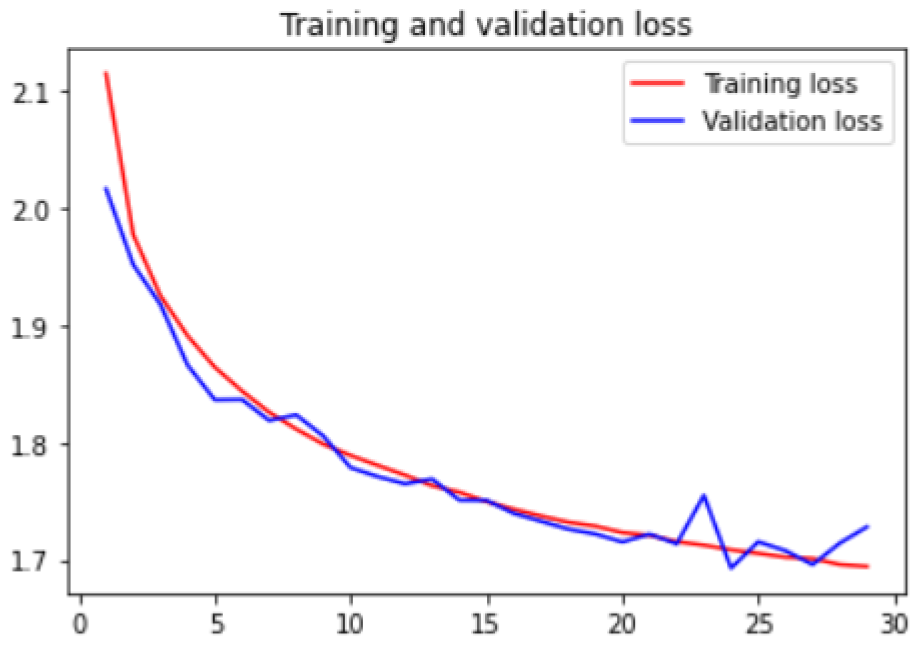
```
Epoch 1/100
1599/1599 [==============================] - 104s 57ms/step - loss: 2.1146 - categorical_accuracy: 0.2334 - val_loss:
2.0162 - val_categorical_accuracy: 0.2689
Epoch 2/100
1599/1599 [==============================] - 89s 55ms/step - loss: 1.9772 - categorical_accuracy: 0.2884 - val_loss:
1.9516 - val_categorical_accuracy: 0.2762
Epoch 3/100
1599/1599 [==============================] - 88s 55ms/step - loss: 1.9251 - categorical_accuracy: 0.3103 - val_loss:
1.9176 - val_categorical_accuracy: 0.2954
Epoch 4/100
1599/1599 [==============================] - 87s 54ms/step - loss: 1.8909 - categorical_accuracy: 0.3229 - val_loss:
1.8659 - val_categorical_accuracy: 0.3358
Epoch 5/100
1599/1599 [==============================] - 87s 54ms/step - loss: 1.8642 - categorical_accuracy: 0.3347 - val_loss:
1.8363 - val_categorical_accuracy: 0.3514
Epoch 6/100
```

- plot the loss and accuracy

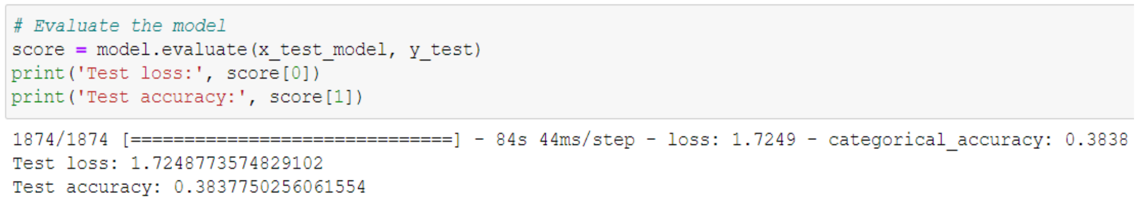


```
acc = history.history['categorical_accuracy']
val_acc = history.history['val_categorical_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1) # number of epochs

plt.title('Training and validation accuracy') # graph title
plt.plot(epochs, acc, 'red', label='Training acc')
plt.plot(epochs, val_acc, 'blue', label='Validation acc')
plt.legend() # legend is an area describing the elements of the graph

plt.figure()
plt.title('Training and validation loss') # graph title
plt.plot(epochs, loss, 'red', label='Training loss')
plt.plot(epochs, val_loss, 'blue', label='Validation loss')
plt.legend() # legend is an area describing the elements of the graph

plt.show()
```





- Evaluate the model



```
# Evaluate the model
score = model.evaluate(x_test_model, y_test)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
1874/1874 [==============================] - 84s 44ms/step - loss: 1.7249 - categorical_accuracy: 0.3838
Test loss: 1.7248773574829102
Test accuracy: 0.3837750256061554
```

3- Result of Transfer Learning

- VGG Model
    1. Test loss: 1.905113935470581
    2. Test accuracy: 0.29831379652023315
- Resnet Model:
    1. Test loss: 1.7248773574829102
    2. Test accuracy: 0.3837750256061554
- ✓ As a result of transfer learning, the LeNet-5 is superior to transfer learning models because we built the model from scratch, which means that the model was trained using only the dataset that I had, rather than another dataset like transfer learning (where I took weights from another dataset to train our model), so the model trained weights that were compatible with our dataset, resulting in higher accuracy than transfer learning models.