The Second Project Report

(Recognize the Fashion-MNIST digits)

Supervised by: Hazem Abbas

Team Members:

Ahmed Salama (6)

Zeyad Tarek Mohamed (55)

Mahmoud Khorshed (29)

Queens' university, school of computing

April 5, 2022

Project objective:

It is a dataset comprised of 60,000 small square 28×28-pixel grayscale images 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 is listed below.

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

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.
- **Challenges ▶**:
    1. Interact With Images
    2. Data Augmentation
    3. Encoding the labels.
    4. Interact with Transfer learning.
    5. choose the best hyperparameters for the network.

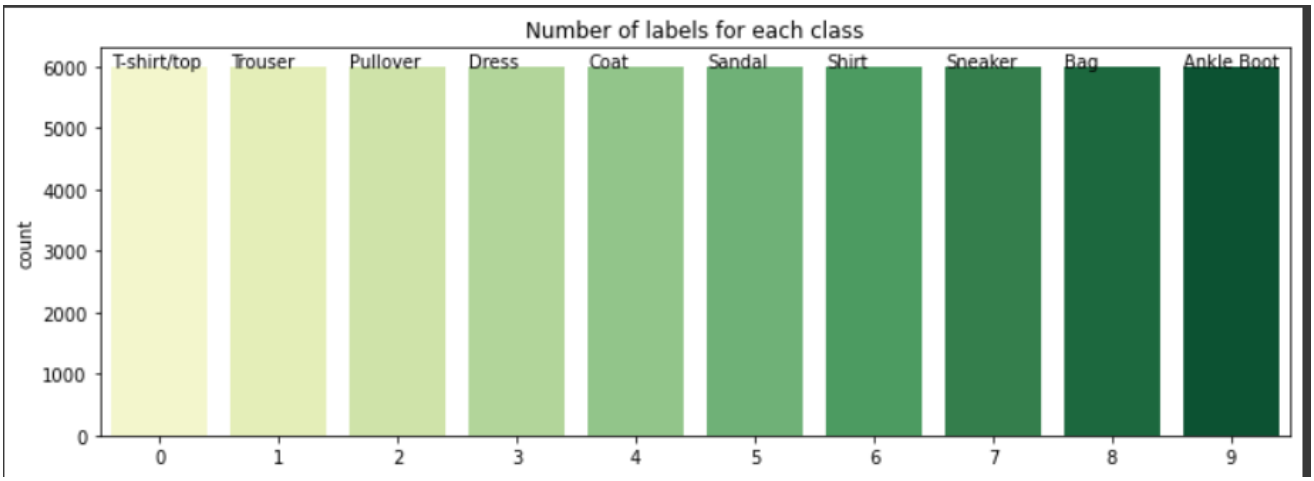
- **Impact ▶**: *Predicting the type of clothes (Bag, Shirt ...etc.).*

Data Description: -

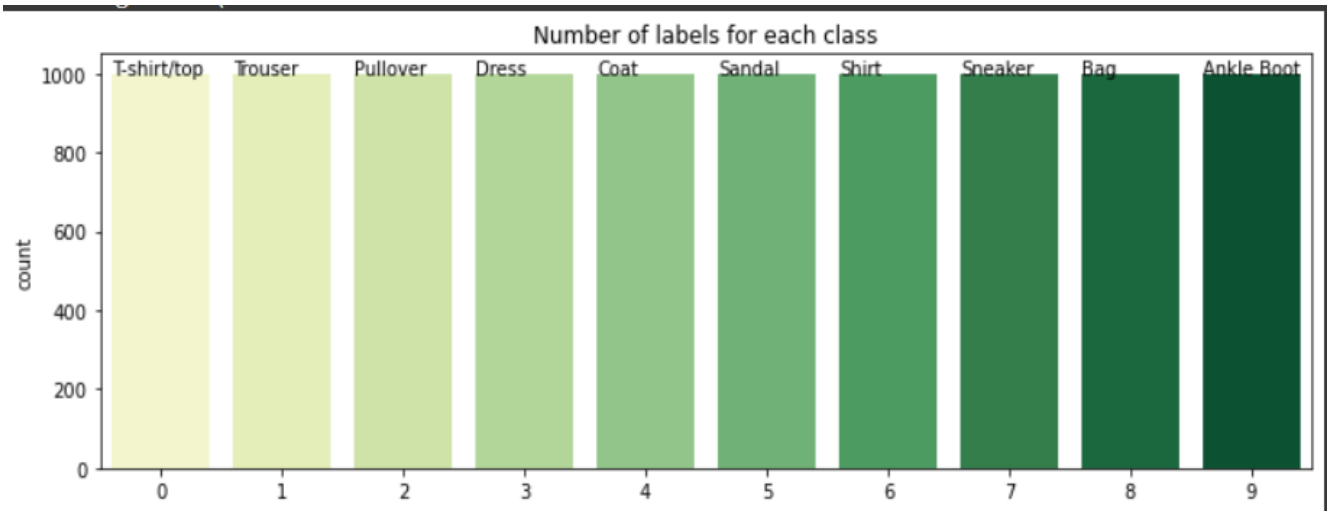
Fashion is a dataset of Zalando's article images consisting of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 grayscale image, associated with a label from 10 classes. It is split in 10,000 as test and 50,000 as train datasets. Although the dataset is relatively 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.

Part I: Data Exploratory:

- *Number of labels for each class on training data*



- *Number of labels for each class on testing data*



1. *Visualize Some of Images*

Here sample of dataset represents all types of clothes

Part 2: Data Preprocessing:

- We have split the data into 60% training, 20% validation ,20 testing
- Encode the labels using factorize

- Normalize Images (0-255)
- Reshape each image according to Architecture we used
- Data Augmentation

Part III: *Training in CNN (**LeNet-5 Architecture**)*:

- ***Implement lenet-5 without Tunning any Hyperparameters***

```
model.summary()
```

- ***We plot graphs for Accuracies (Train Vs Validation set)***

epoch

- ***We plot graphs for losses (Train Vs Validation set)***

epoch

Results

Loss Of Training: ***0.2156***

Loss Of Validation: ***0.2240***

Accuracy Of Training : ***0.9193***

Accuracy Of Validation : ***0.9210***

We used Kera's tuner to choose the best hyperparameters.

- **Learning Rate**
- **Activation Functions**
- **Early Stopping**
- **Padding**
- **Number Of units of each Layer**
- **Number of filters**

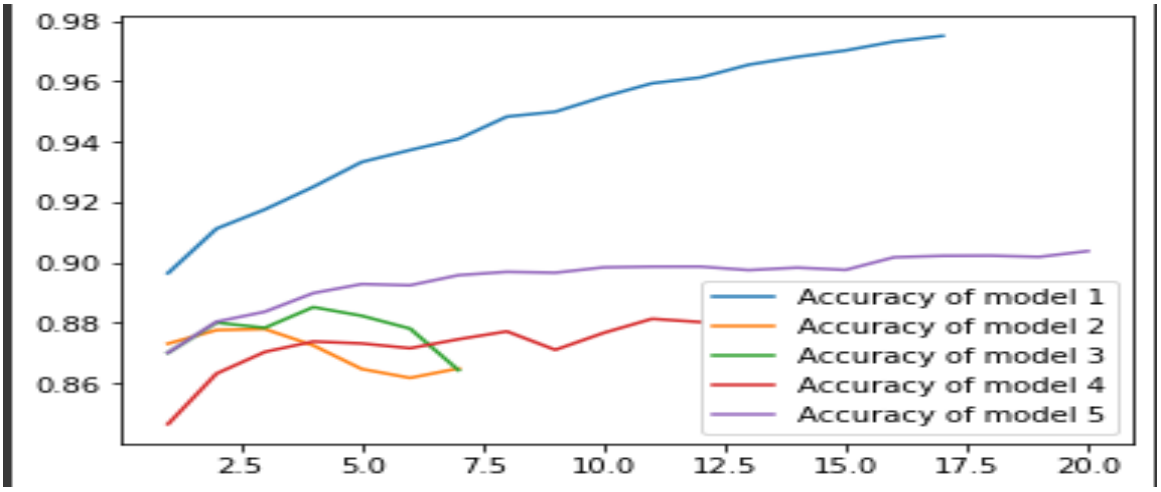
1- The search method used is hyperband (it's faster than Bayesian and better than random search) to get the best hyperparameters combinations.

- Here are the highest five values of learning rates and activation function

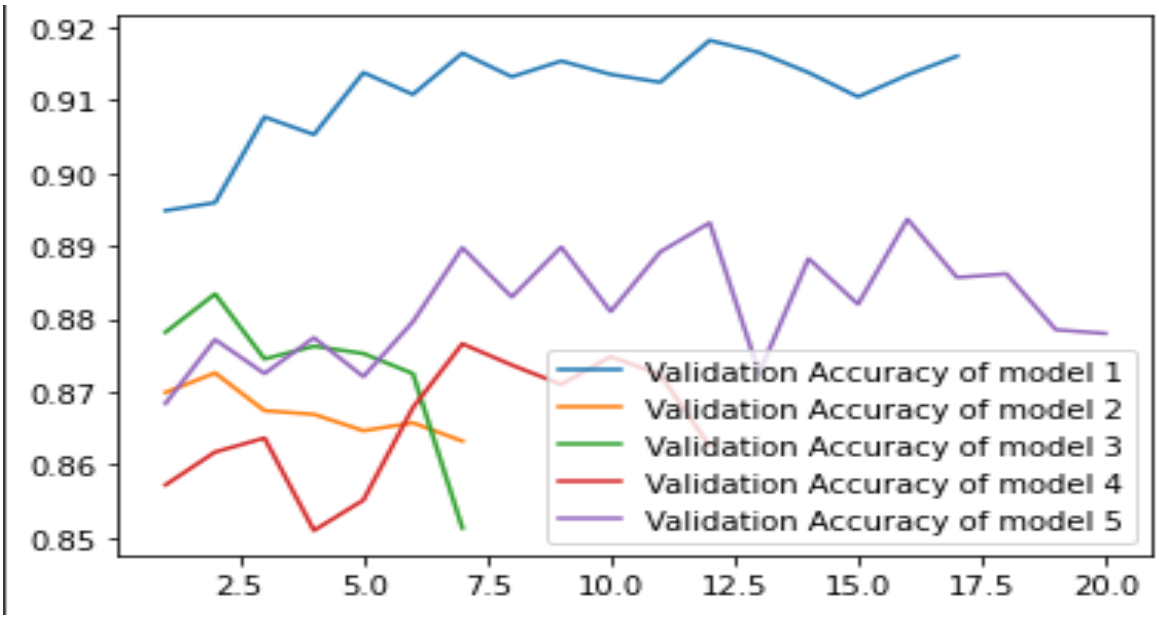
	Learning rate	best activation function used for first convo layer	best activation function used for second convo layer	best activation function used for first hidden layer	best activation function used for second hidden layer	best number of filters used for first convo layer	best number of filters used for second convo layer	best padding value used for first convo layer	best padding value used for second convo layer	best number of neurons used for first hidden layer	best number of neurons used for second hidden layer
0	0.001	relu	relu	tanh	tanh	32	64	valid	valid	64	128
1	0.010	relu	relu	tanh	sigmoid	8	32	same	same	160	320
2	0.010	tanh	tanh	tanh	relu	8	8	same	valid	224	160
3	0.010	relu	tanh	sigmoid	sigmoid	8	8	valid	same	224	128

- *Let's compare the improvement of tuning the parameters trials*
- *Note the models sorted from highest to lowest one*

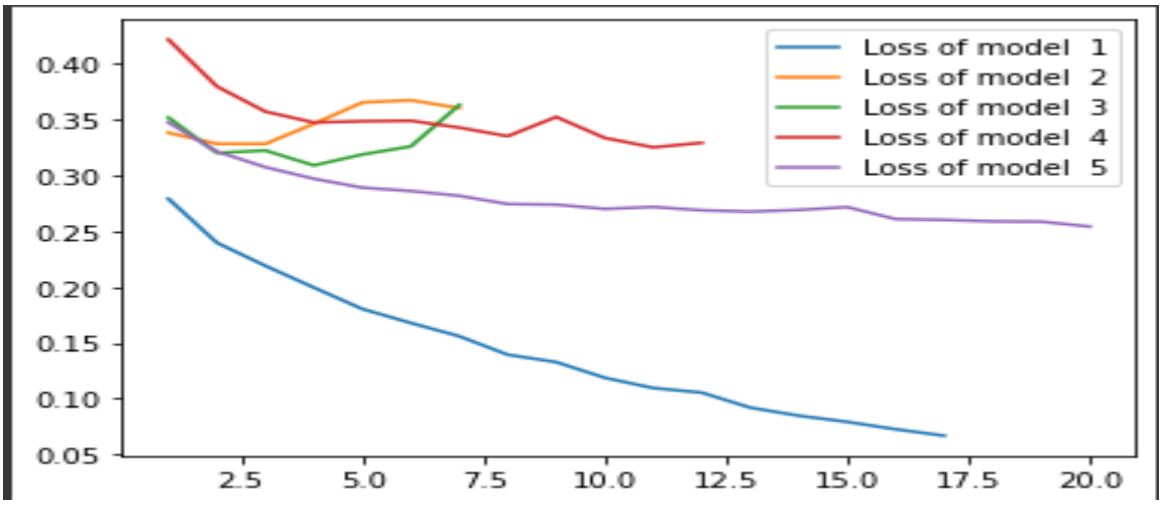
1. ***<u>Compare the Accuracies of the first highest models on Training Data</u>***



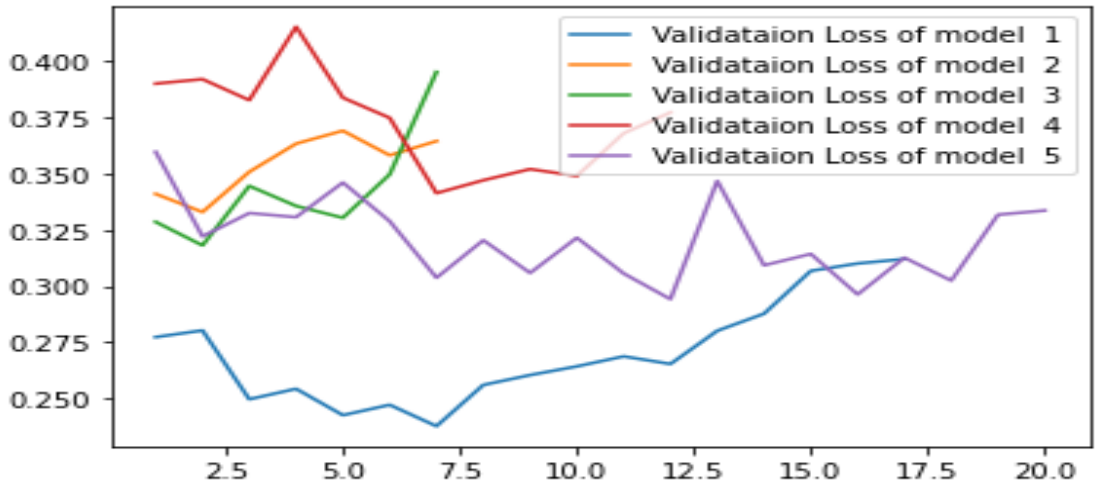
2. ***<u>Compare the Accuracies of the first highest models on validation set</u>***



3. <u>Compare the Losses of the first highest models on Training set</u>



4. <u>Compare the Losses of the first highest models on Validation set</u>



Conclusion

- *We got the first fifth Highest models by trying Different Tunning Hyper Parameters Using Hyper Band method to get them Using Architecture LeNet-5 model, and no overfitting happened.*

- *The best hyper-Parametric are: -*
    1. *learning rate =* ***0.001***
    2. *Activation function used for first convo layer =* **relu**
    3. *Activation function used for the second convo layer =* **relu**
    4. *Activation function used for the first hidden layer =* ***tanh***
    5. *Activation function used for the second hidden layer =* ***tanh***
    6. *Number of filters used for first convo layer =* ***32***
    7. *Number of filters used for the second convo layer =* ***64***
    8. *Padding value used for first convo layer =* ***valid***
    9. *Padding value used for second convo layer =* ***valid***
    10. *Number of neurons used for the first hidden layer =* ***64***

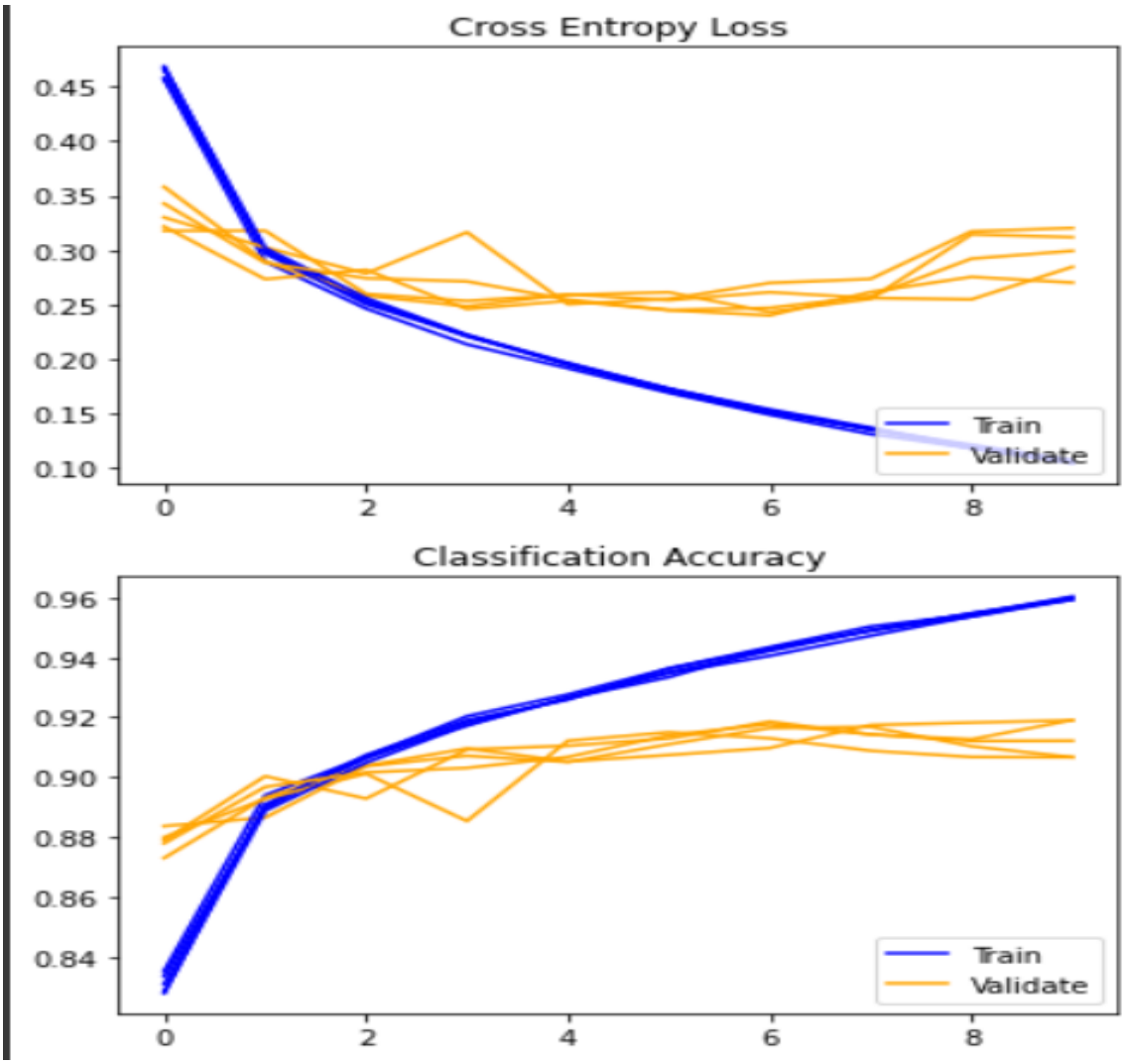
11. Number of neurons used for the second hidden layer= **128**

- According to graphs the Accuracy is Improved by Using the hyperparametric.

- ***LeNet was used in detecting handwritten cheques by banks based on 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.***

Part IIII (Cross Validation Part)

Evaluate the model using 5-fold cross-validation.



We also tried to use another two CNN models (using transfer learning) and we will compare the results with the fully trained LeNet-5.

1- Resnet50: -

ResNet, short for Residual Network is a specific type of neural network that was introduced in 2015 by Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun in their paper “Deep Residual Learning for Image Recognition”.The ResNet models were extremely successful.

At first (Preprocessing)

Resnet 50 with fashion mnist, We need to resize the MNIST data set. Note that minimum size actually depends on the ImageNet model. For example: Inception v3 requires at least 75, where ResNet is asking for 32. Apart from that, the MNIST is a grayscale image, but it may conflict if you're using the pretrained weight of these models. So, good and safe side is to resize and convert grayscale to RGB.

So We will resize **fashion-MNIST** from 28 to **32**. Also, make **3 channels** instead of keeping 1.

```
# expand new axis, channel axis
x_train = np.expand_dims(x_train, axis=-1)
x_test = np.expand_dims(x_test, axis=-1)

# [optional]: we may need 3 channel (instead of 1)
x_train = np.repeat(x_train, 3, axis=-1)
x_test = np.repeat(x_test, 3, axis=-1)

# it's always better to normalize
x_train = x_train.astype('float32') / 255
x_test = x_test.astype('float32') / 255

# resize the input shape , i.e. old shape: 28, new shape: 32
x_train = tf.image.resize(x_train, [32,32]) # if we want to resize
x_test = tf.image.resize(x_test, [32,32]) # if we want to resize
```

1- Here we used **(numpy.expand_dims) to Expand** the shape of an array.

Insert a new axis that will appear at the *axis* position in the expanded array shape.

2- We repeated the 1 channel (Gray scale) to be 3 channels (RGB) using **(NumPy.Repeat)**.

3- Finally, we resize its shape from 28 to 32 to be suitable with resnet50 using **(tf.image.resize).**

```
print(x_train.shape
print(x_test.shape,

(60000, 32, 32, 3)
(10000, 32, 32, 3)
```

To implement ResNet version1 with 50 layers (**ResNet 50**), we simply use the function from Keras

ResNet 50

```
from keras import backend as K

# Do some code, e.g. train and save model

K.clear_session()

input = tf.keras.Input(shape=(32,32,3))
efnet = tf.keras.applications.ResNet50(weights='imagenet',
                                       include_top = False,
                                       input_tensor = input)
# Now that we apply global max pooling.
gap = tf.keras.layers.GlobalMaxPooling2D()(efnet.output)

# Finally, we add a classification layer.
output = tf.keras.layers.Dense(10, activation='softmax', use_bias=True)(gap)

# bind all
RESN_model = tf.keras.Model(efnet.input, output)
RESN_model.summary()
```

- **include_top**: whether to include the fully-connected layer at the top of the network, But here we make it **False** because we have our FCN design.
- **weights**: 'Imagenet' (pre-training on ImageNet).
- **input_tensor**: optional Keras tensor (i.e. output of layers.Input(32,32,3)) to use as image input for the model.
- Output >> Dense(10) because of 10 classes.

Train using Resnet 50

Train

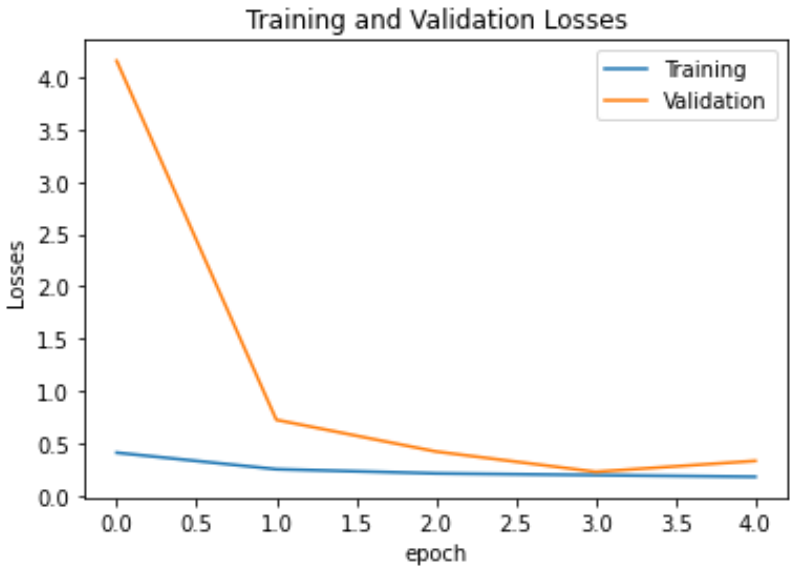
```
RESN_model.compile(
          loss  = tf.keras.losses.CategoricalCrossentropy(),
          metrics = tf.keras.metrics.CategoricalAccuracy(),
          optimizer = tf.keras.optimizers.Adam())
# fit
history=RESN_model.fit(x_train, y_train,validation_split=0.2, batch_size=128, epochs=5, verbose = 1)
```

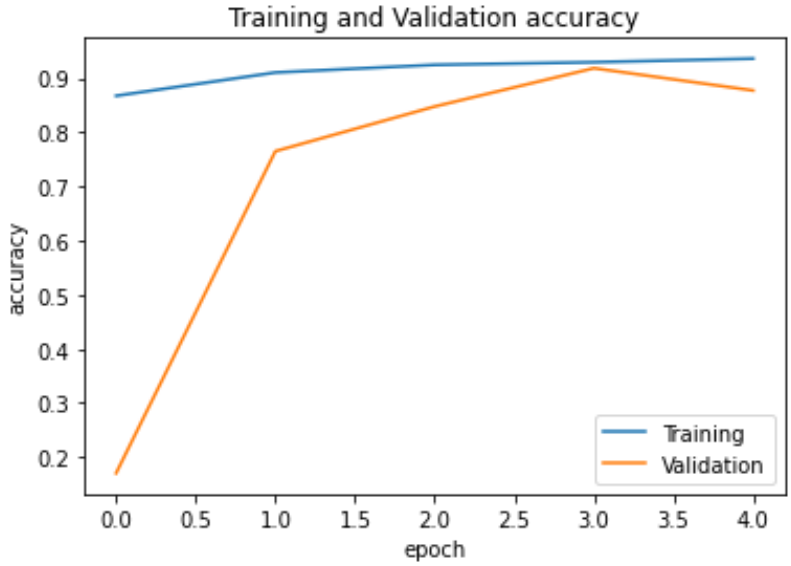
```
Epoch 1/5
375/375 [==============================] - 70s 166ms/step - loss: 0.4085 - categorical_accuracy: 0.8673 - val_loss: 4.1558 - val_categorical_accuracy: 0.1706
Epoch 2/5
375/375 [==============================] - 60s 161ms/step - loss: 0.2504 - categorical_accuracy: 0.9103 - val_loss: 0.7226 - val_categorical_accuracy: 0.7650
Epoch 3/5
375/375 [==============================] - 60s 160ms/step - loss: 0.2104 - categorical_accuracy: 0.9244 - val_loss: 0.4194 - val_categorical_accuracy: 0.8475
Epoch 4/5
375/375 [==============================] - 60s 160ms/step - loss: 0.1943 - categorical_accuracy: 0.9293 - val_loss: 0.2255 - val_categorical_accuracy: 0.9183
Epoch 5/5
375/375 [==============================] - 60s 160ms/step - loss: 0.1778 - categorical_accuracy: 0.9359 - val_loss: 0.3300 - val_categorical_accuracy: 0.8771
```

Train using Le-Net

```
Best val_accuracy So Far: 0.8889999985694885
Total elapsed time: 00h 45m 38s
INFO:tensorflow:Oracle triggered exit
```

We have noticed that Validation Accuracy for resnet (87.7%) has Closed to LeNet (0.88%)





There is a **`well-fitting`** because of no gap between train and validation.

2- Inception V3: -

Inception v3 is a convolutional neural network for assisting in image analysis and object detection and got its start as a module for Google Net. It is the third edition of Google's Inception Convolutional Neural Network, originally introduced during the ImageNet Recognition Challenge. The design of Inceptionv3 was intended to allow deeper networks while also keeping the number of parameters from growing too large: it has "under 25 million parameters", compared against 60 million for Alex Net.

- We would also need to resize images to **75*75*3**

```
img_dims= 75

# resize the input shape , i.e. old shape: 28, new shape: 75
x_train = tf.image.resize(x_train, [img_dims,img_dims]) # if we want to resize
x_test = tf.image.resize(x_test, [img_dims,img_dims]) # if we want to resize


# one hot
y_train = tf.keras.utils.to_categorical(y_train , num_classes=10)
y_test = tf.keras.utils.to_categorical(y_test , num_classes=10)

print(x_train.shape, y_train.shape)
print(x_test.shape, y_test.shape)
```

```
(60000, 75, 75, 3) (60000, 10)
(10000, 75, 75, 3) (10000, 10)
```

Train using Inception v3

Train

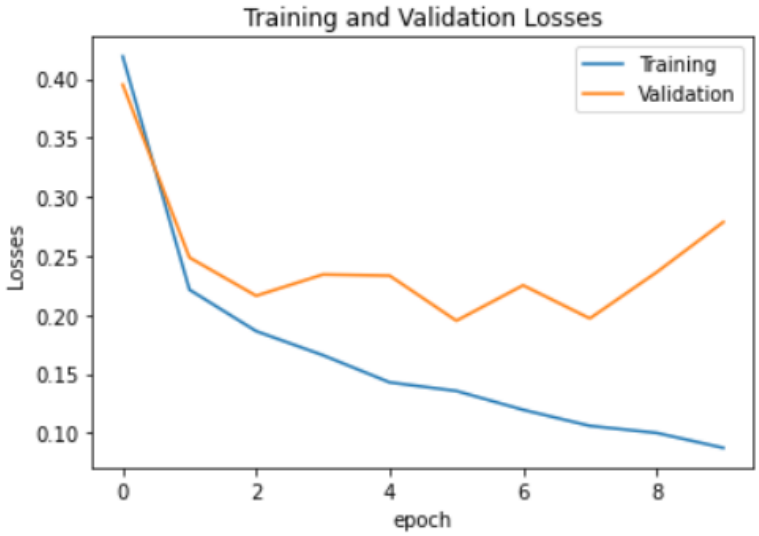
```
Incept_model.compile(
          loss  = tf.keras.losses.CategoricalCrossentropy(),
          metrics = tf.keras.metrics.CategoricalAccuracy(),
          optimizer = tf.keras.optimizers.Adam())
# fit
print("Fit model on training data")
history = Incept_model.fit(x_train, y_train,validation_split=0.2, batch_size=128, epochs=10, verbose = 1)
```

```
Fit model on training data
Epoch 1/10
375/375 [==============================] - 134s 306ms/step - loss: 0.4190 - categorical_accuracy: 0.8589 - val_loss: 0.3952 - val_categorical_accuracy: 0.8785
Epoch 2/10
375/375 [==============================] - 112s 298ms/step - loss: 0.2214 - categorical_accuracy: 0.9245 - val_loss: 0.2486 - val_categorical_accuracy: 0.9121
Epoch 3/10
375/375 [==============================] - 112s 298ms/step - loss: 0.1862 - categorical_accuracy: 0.9361 - val_loss: 0.2162 - val_categorical_accuracy: 0.9264
Epoch 4/10
375/375 [==============================] - 112s 298ms/step - loss: 0.1659 - categorical_accuracy: 0.9431 - val_loss: 0.2342 - val_categorical_accuracy: 0.9197
Epoch 5/10
375/375 [==============================] - 112s 300ms/step - loss: 0.1428 - categorical_accuracy: 0.9510 - val_loss: 0.2333 - val_categorical_accuracy: 0.9247
Epoch 6/10
375/375 [==============================] - 112s 298ms/step - loss: 0.1356 - categorical_accuracy: 0.9532 - val_loss: 0.1951 - val_categorical_accuracy: 0.9345
Epoch 7/10
375/375 [==============================] - 117s 311ms/step - loss: 0.1196 - categorical_accuracy: 0.9594 - val_loss: 0.2251 - val_categorical_accuracy: 0.9267
Epoch 8/10
375/375 [==============================] - 112s 298ms/step - loss: 0.1060 - categorical_accuracy: 0.9638 - val_loss: 0.1970 - val_categorical_accuracy: 0.9334
```

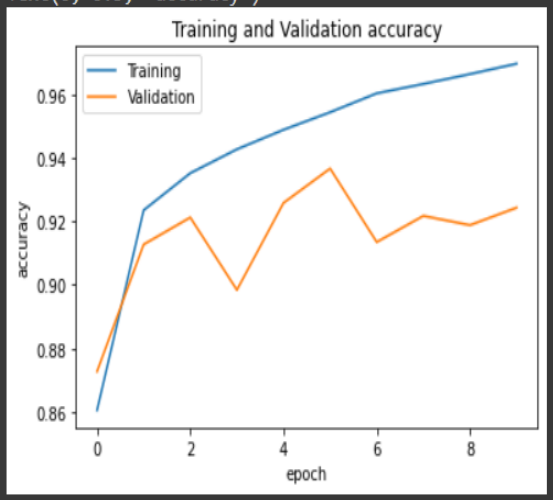
We have noticed that Validation Accuracy for Inception improved from 88% in LeNet to 93.5%

Model	Le-Net	Resnet	Inception
Validation Accuracy	*0.888*	87.7	93.5

```
Text(0, 0.5, 'Losses')
```




```
Text(0, 0.5, 'accuracy')
```



There is an `no overfitting` this trail without freezee any layers and the gap between training and validation little bit small according to losses and accuracies and the Inception module the best than resnet-50 module

References

- *https://www.tensorflow.org/tutorials/keras/keras_tuner*