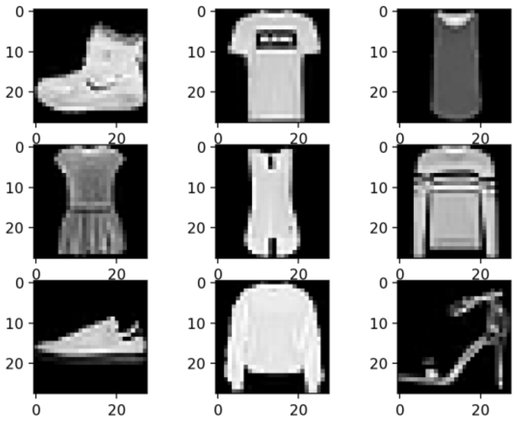
END-Course Test

Scenario – 1

Description

The dataset is similar to MNIST but includes images of certain clothing and accessory. The objective is to classify images into specific classes using CNN.



Dataset:

Total Images: - 70,000

Train Images: - 60,000

Test Images:- 10,000

Image Size:- 28 X 28

Different Classes:

- Classes: 'T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat', 'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle Boot'

Note: Please use google colab to work on this project Also, make sure to select GPU backend while selecting a runtime.

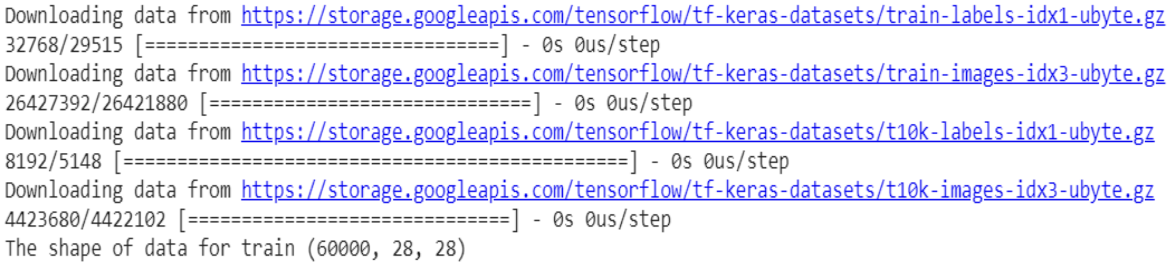
https://medium.com/deep-learning-turkey/google-colab-free-gpu-tutorial-e113627b9f5d

Problem Statement

Question 1

A. Load Fashion data from Keras Library and Split the same into Train and Test.

Output :-



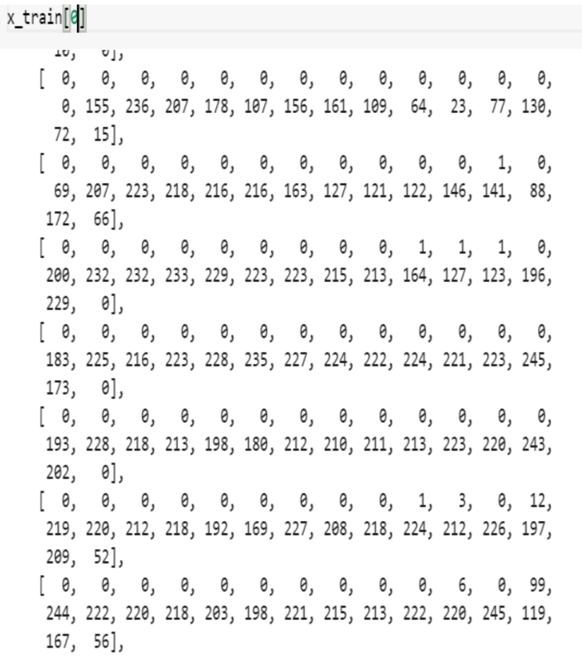
```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz
32768/29515 [=================================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz
26427392/26421880 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz
8192/5148 [===============================================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz
4423680/4422102 [==============================] - 0s 0us/step
The shape of data for train (60000, 28, 28)
```

B. Scale the values of train and test between 0 & 1 by dividing train & test by 255

Output:

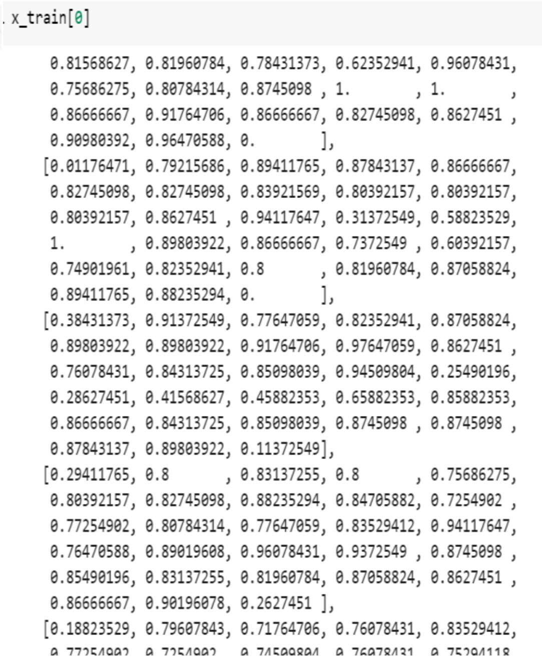
Original X_train

Scaled X_Train



```
x_train[0]
```

```
        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,   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,
        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,
        193, 228, 218, 213, 198, 180, 212, 210, 211, 213, 223, 220, 243,
        202,   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,  99,
        244, 222, 220, 218, 203, 198, 221, 215, 213, 222, 220, 245, 119,
        167,  56],
```



```
x_train[0]
```

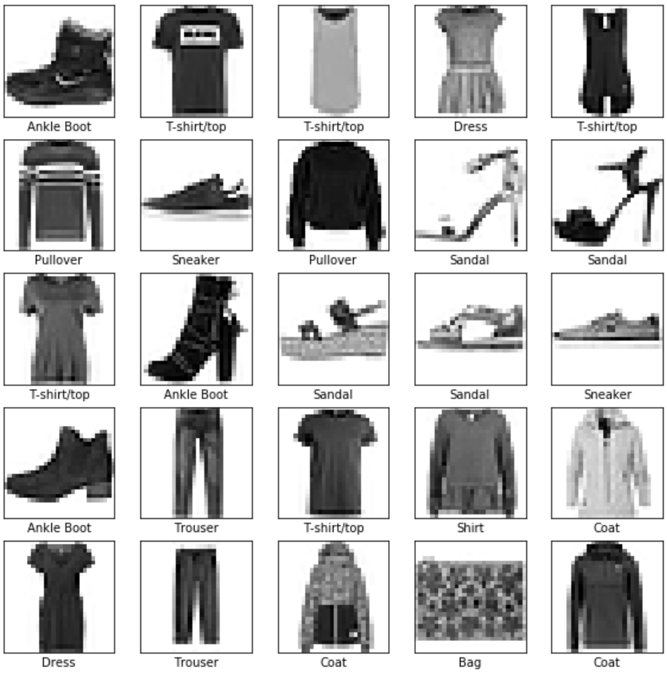
```
        0.81568627, 0.81960784, 0.78431373, 0.62352941, 0.96078431,
        0.75686275, 0.80784314, 0.8745098 , 1.        , 1.        ,
        0.86666667, 0.91764706, 0.86666667, 0.82745098, 0.8627451 ,
        0.90980392, 0.96470588, 0.        ],
       [0.01176471, 0.79215686, 0.89411765, 0.87843137, 0.86666667,
        0.82745098, 0.82745098, 0.83921569, 0.80392157, 0.80392157,
        0.80392157, 0.8627451 , 0.94117647, 0.31372549, 0.58823529,
        1.        , 0.89803922, 0.86666667, 0.7372549 , 0.60392157,
        0.74901961, 0.82352941, 0.8       , 0.81960784, 0.87058824,
        0.89411765, 0.88235294, 0.        ],
       [0.38431373, 0.91372549, 0.77647059, 0.82352941, 0.87058824,
        0.89803922, 0.89803922, 0.91764706, 0.97647059, 0.8627451 ,
        0.76078431, 0.84313725, 0.85098039, 0.94509804, 0.25490196,
        0.28627451, 0.41568627, 0.45882353, 0.65882353, 0.85882353,
        0.86666667, 0.84313725, 0.85098039, 0.8745098 , 0.8745098 ,
        0.87843137, 0.89803922, 0.11372549],
       [0.29411765, 0.8       , 0.83137255, 0.8       , 0.75686275,
        0.80392157, 0.82745098, 0.88235294, 0.84705882, 0.7254902 ,
        0.77254902, 0.80784314, 0.77647059, 0.83529412, 0.94117647,
        0.76470588, 0.89019608, 0.96078431, 0.9372549 , 0.8745098 ,
        0.85490196, 0.83137255, 0.81960784, 0.87058824, 0.8627451 ,
        0.86666667, 0.90196078, 0.2627451 ],
       [0.18823529, 0.79607843, 0.71764706, 0.76078431, 0.83529412,
        0.77254902, 0.7254902 , 0.74509804, 0.76078431, 0.75294118
```

Scenario – 2

Problem Statement

Question 2: Display first 25 images from the training dataset and display the labels along with them

Output:-



Then the output should be:

QUESTION 3:

Load the data (again, important)

1. Reshape the data to (28,28,1). The actual data is in (28,28) format and we need to add a single channel, 1 to it. Do the reshape for both train and test.

Output:

Train image before reshaping → (60000, 28, 28)

Train image after reshaping → (60000, 28, 28, 1)

Question 4:

A. Build basic CNN on the fashion MNIST Data.

HINT:-

- Reshape/flatten the data

- Conv2D with 32 Neuron; Filter 3,3 ; Activation: Relu ; Stride (1,1)

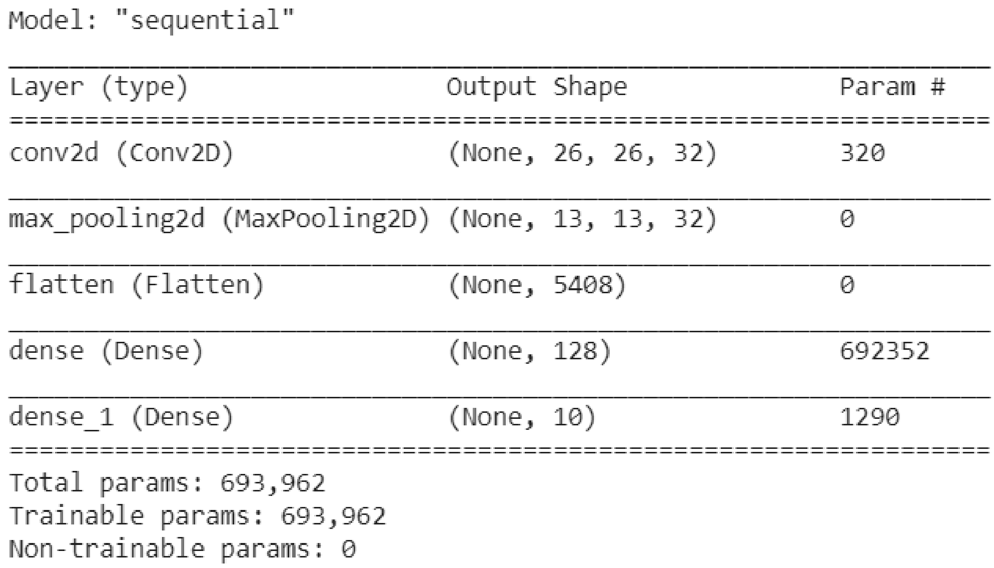
- MaxPool2D ; Pool Size (2,2)

- Flatten the data again to send to dense layer

- 128 Neuron single Dense Layer with relu

- 10 neuron single dense layer with SoftMax as output layer

Output:-



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 13, 13, 32)        0
_________________________________________________________________
flatten (Flatten)            (None, 5408)              0
_________________________________________________________________
dense (Dense)                (None, 128)               692352
_________________________________________________________________
dense_1 (Dense)              (None, 10)                1290
=================================================================
Total params: 693,962
Trainable params: 693,962
Non-trainable params: 0
```

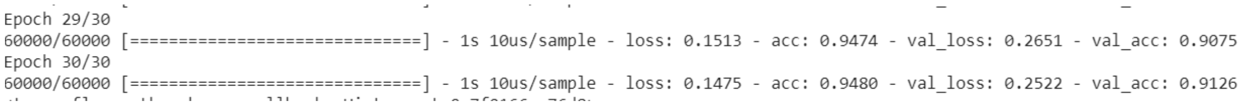
B. Compile the model with either ADAM or GradientDescent with Loss as sparse_categorical_crossentropy and metrics 'accuracy'

HINT:-

- Optimizer = ‘Adam’
- Loss = sparse categorical cross entropy

C. Fit a model with 30 Epochs and 1000 Batch Size

Output:-



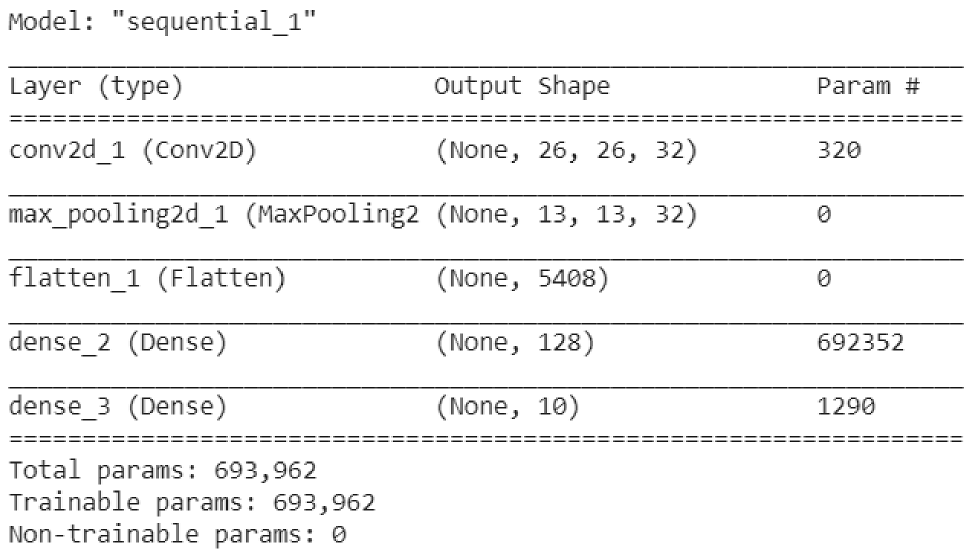
```
Epoch 29/30
60000/60000 [==============================] - 1s 10us/sample - loss: 0.1513 - acc: 0.9474 - val_loss: 0.2651 - val_acc: 0.9075
Epoch 30/30
60000/60000 [==============================] - 1s 10us/sample - loss: 0.1475 - acc: 0.9480 - val_loss: 0.2522 - val_acc: 0.9126
```

Question 5:

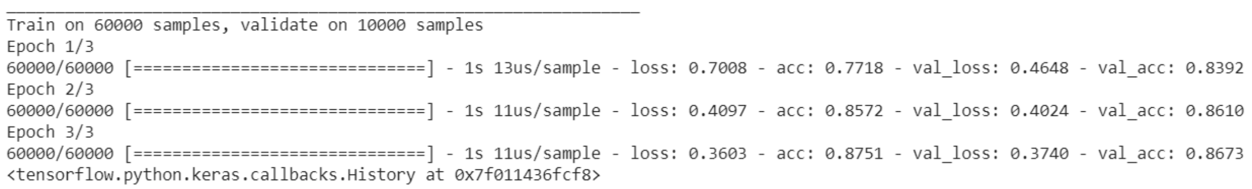
Now, let's have the same model, but this time using simple "categorical_crossentropy" as loss. For this, you would have to convert "y" or labels to dummy encoding or categorical encoding. You can use to_categorical function from Keras.

HINT:-

- Optimizer = ‘Adam’
- Loss = ‘categorical_crossentropy
- https://www.tensorflow.org/api_docs/python/tf/keras/utils/to_categorical



```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_1 (Conv2D)            (None, 26, 26, 32)        320       
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 13, 13, 32)        0         
_________________________________________________________________
flatten_1 (Flatten)          (None, 5408)              0         
_________________________________________________________________
dense_2 (Dense)              (None, 128)               692352    
_________________________________________________________________
dense_3 (Dense)              (None, 10)                1290      
=================================================================
Total params: 693,962
Trainable params: 693,962
Non-trainable params: 0
```

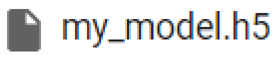


```
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/3
60000/60000 [==============================] - 1s 13us/sample - loss: 0.7008 - acc: 0.7718 - val_loss: 0.4648 - val_acc: 0.8392
Epoch 2/3
60000/60000 [==============================] - 1s 11us/sample - loss: 0.4097 - acc: 0.8572 - val_loss: 0.4024 - val_acc: 0.8610
Epoch 3/3
60000/60000 [==============================] - 1s 11us/sample - loss: 0.3603 - acc: 0.8751 - val_loss: 0.3740 - val_acc: 0.8673
<tensorflow.python.keras.callbacks.History at 0x7f011436fcf8>
```

Question 6:

A. Save the model as .H5 file, as “my_model”.h5. In colab, make sure your files are stored in “files” section (left section of notebook, there’s a tab called files)

Output :-

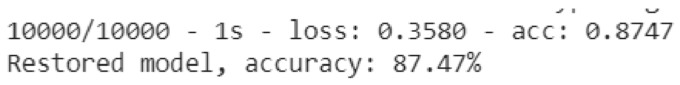


my_model.h5

B. Now load the "my_model.h5" and evaluate the test_image and test_label with new_model to check accuracy. Print the accuracy.

Hint: https://www.tensorflow.org/api_docs/python/tf/keras/models/load_model

Output:-

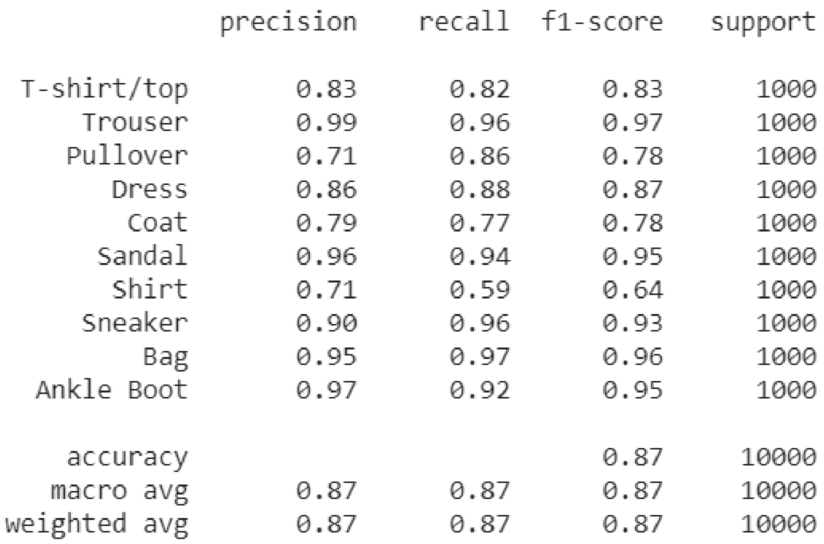


```
10000/10000 - 1s - loss: 0.3580 - acc: 0.8747
Restored model, accuracy: 87.47%
```

Question 7:

Print the Confusion Matrix for predicted classes of test_images and test_label.

Output:-



```
              precision    recall  f1-score   support

 T-shirt/top       0.83      0.82      0.83      1000
     Trouser       0.99      0.96      0.97      1000
    Pullover       0.71      0.86      0.78      1000
       Dress       0.86      0.88      0.87      1000
        Coat       0.79      0.77      0.78      1000
      Sandal       0.96      0.94      0.95      1000
       Shirt       0.71      0.59      0.64      1000
     Sneaker       0.90      0.96      0.93      1000
         Bag       0.95      0.97      0.96      1000
  Ankle Boot       0.97      0.92      0.95      1000

    accuracy                           0.87     10000
   macro avg       0.87      0.87      0.87     10000
weighted avg       0.87      0.87      0.87     10000
```

Question 8:

Print the count of total misclassification that occurred using the saved model.

Output :- 1327

Question 9:

Write an experiment that can perform multiple parameters training. Save the intermediate output of each experiment in the dictionary. The key for each experiment will be string combination of (optimizer+Epoch+BatchSize).

Hint:-

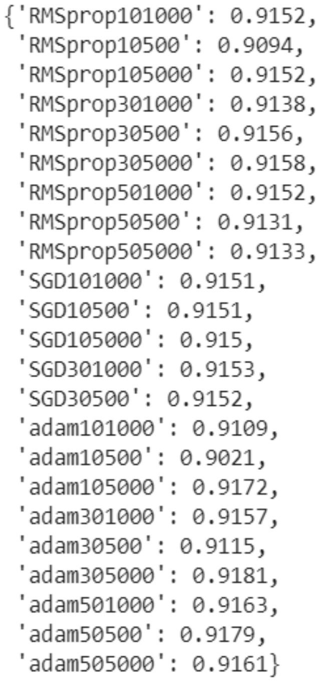
- Epochs = [10,30,50]

- Batch Size = [500,1000, 5000]

- Optimizer = [Adam, RmsProp, SGD]

Use the standard CNN architecture using above parameters and check the accuracy and performance of various models.

HINT:- Result O/P:-



```
{'RMSprop101000': 0.9152,
 'RMSprop10500': 0.9094,
 'RMSprop105000': 0.9152,
 'RMSprop301000': 0.9138,
 'RMSprop30500': 0.9156,
 'RMSprop305000': 0.9158,
 'RMSprop501000': 0.9152,
 'RMSprop50500': 0.9131,
 'RMSprop505000': 0.9133,
 'SGD101000': 0.9151,
 'SGD10500': 0.9151,
 'SGD105000': 0.915,
 'SGD301000': 0.9153,
 'SGD30500': 0.9152,
 'adam101000': 0.9109,
 'adam10500': 0.9021,
 'adam105000': 0.9172,
 'adam301000': 0.9157,
 'adam30500': 0.9115,
 'adam305000': 0.9181,
 'adam501000': 0.9163,
 'adam50500': 0.9179,
 'adam505000': 0.9161}
```