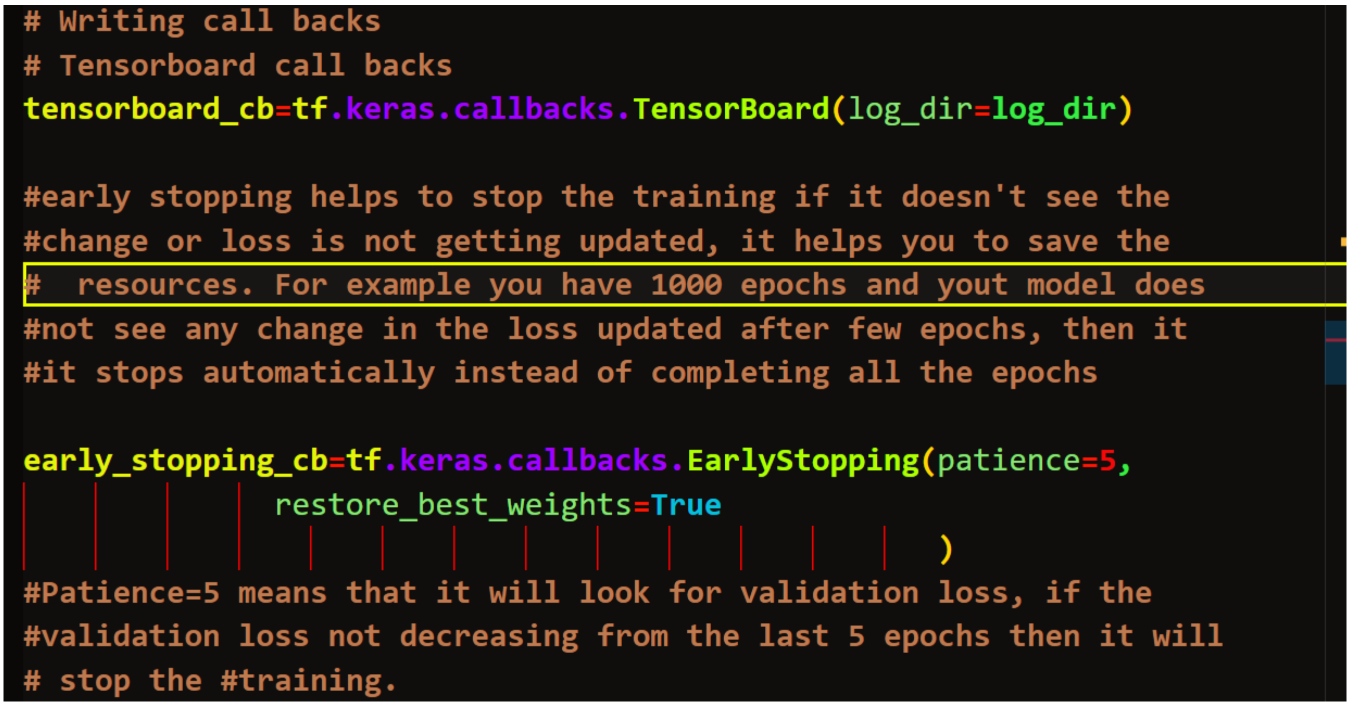
Call back is an extra function given in TF, it will help you to gather some extra information while training.

Tensor board is a utility provided by TF to gather some information to visuallize the logs and all those things.

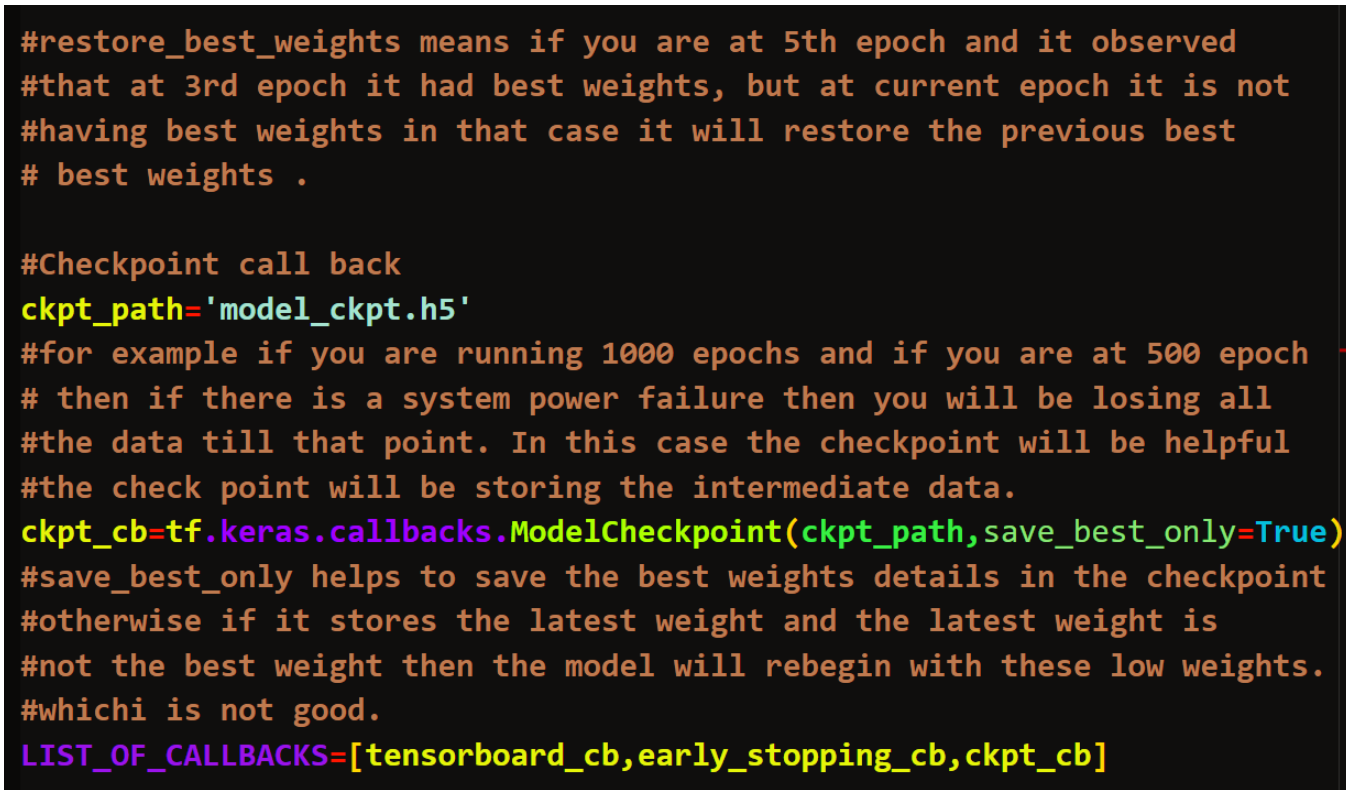
Here we have used the same code as in Deep Learing-3 document, only with the below highlighted changes. Here we are performing below tasks

# Writing information to the Tensorboard
# Defining Tenorboard call back for log files
# Defining the Checkpoint call back (to store the intermediate models as a checkpoint)
# Defying early stop call back (Auto terminating the model training if there is no further reduction in validation loss at any point of epochs.
#Restarting the training at the checkpoint model

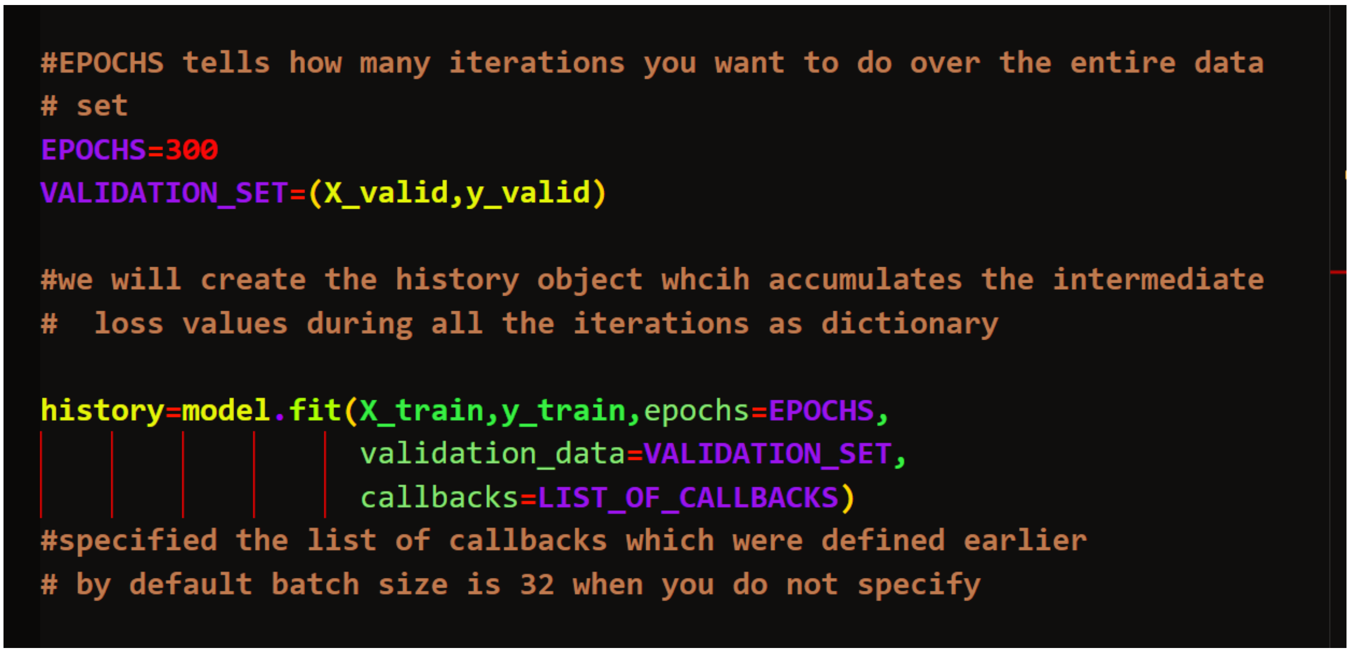
Go through the below comments



```
# Writing call backs
# Tensorboard call backs
tensorboard_cb=tf.keras.callbacks.TensorBoard(log_dir=log_dir)

#early stopping helps to stop the training if it doesn't see the
#change or loss is not getting updated, it helps you to save the
#  resources. For example you have 1000 epochs and yout model does
#not see any change in the loss updated after few epochs, then it
#it stops automatically instead of completing all the epochs

early_stopping_cb=tf.keras.callbacks.EarlyStopping(patience=5,
              restore_best_weights=True
                                                  )
#Patience=5 means that it will look for validation loss, if the
#validation loss not decreasing from the last 5 epochs then it will
# stop the #training.
```

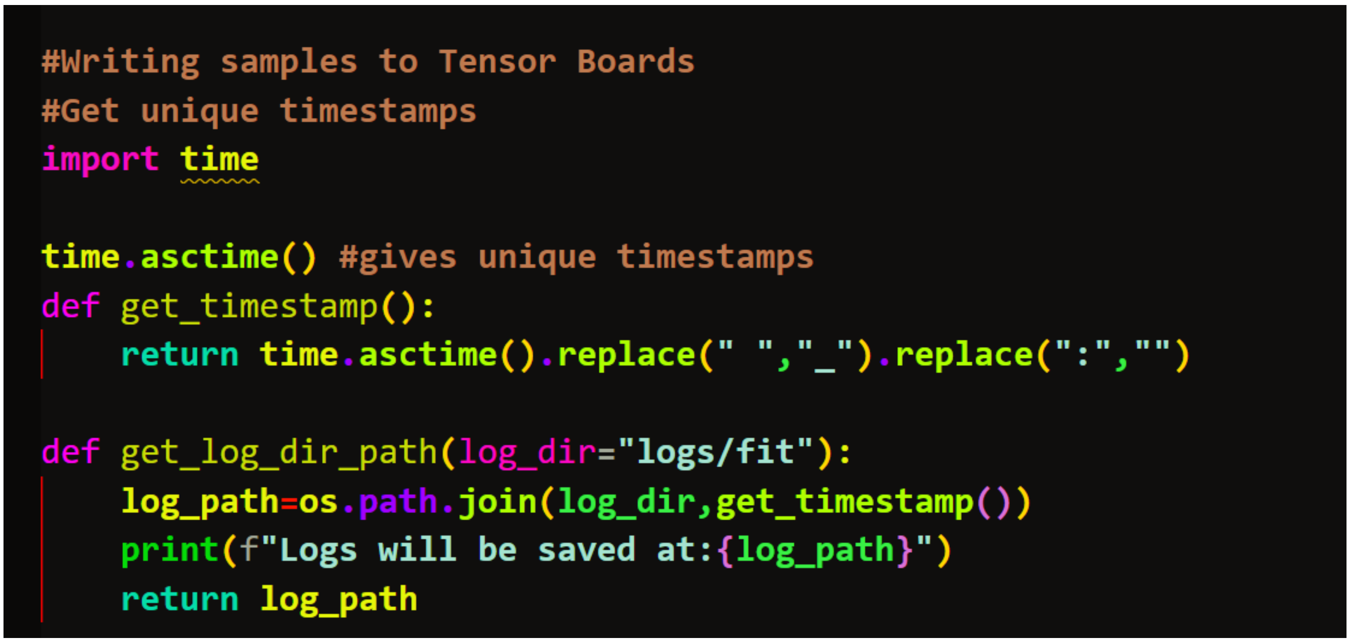


```python
#restore_best_weights means if you are at 5th epoch and it observed
#that at 3rd epoch it had best weights, but at current epoch it is not
#having best weights in that case it will restore the previous best
# best weights .

#Checkpoint call back
ckpt_path='model_ckpt.h5'
#for example if you are running 1000 epochs and if you are at 500 epoch
# then if there is a system power failure then you will be losing all
#the data till that point. In this case the checkpoint will be helpful
#the check point will be storing the intermediate data.
ckpt_cb=tf.keras.callbacks.ModelCheckpoint(ckpt_path,save_best_only=True)
#save_best_only helps to save the best weights details in the checkpoint
#otherwise if it stores the latest weight and the latest weight is
#not the best weight then the model will rebegin with these low weights.
#whichi is not good.
LIST_OF_CALLBACKS=[tensorboard_cb,early_stopping_cb,ckpt_cb]
```

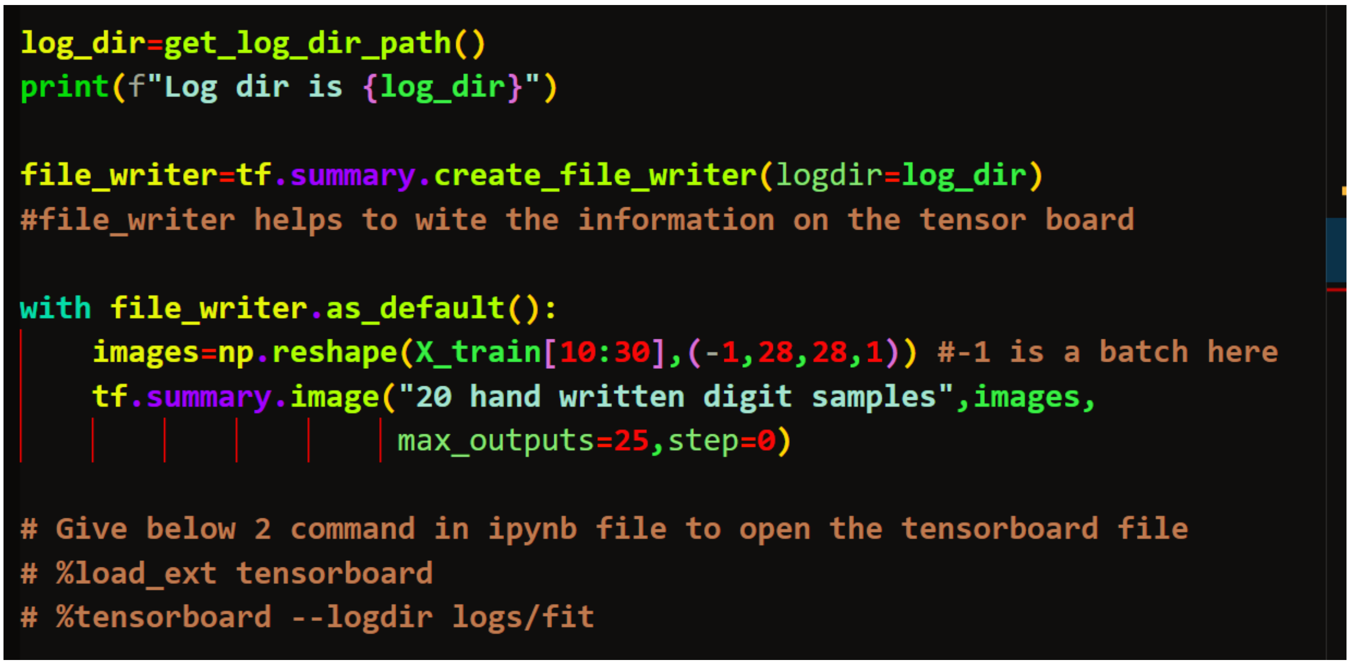


```python
#EPOCHS tells how many iterations you want to do over the entire data
# set
EPOCHS=300
VALIDATION_SET=(X_valid,y_valid)

#we will create the history object whcih accumulates the intermediate
#  loss values during all the iterations as dictionary

history=model.fit(X_train,y_train,epochs=EPOCHS,
                  validation_data=VALIDATION_SET,
                  callbacks=LIST_OF_CALLBACKS)
#specified the list of callbacks which were defined earlier
# by default batch size is 32 when you do not specify
```

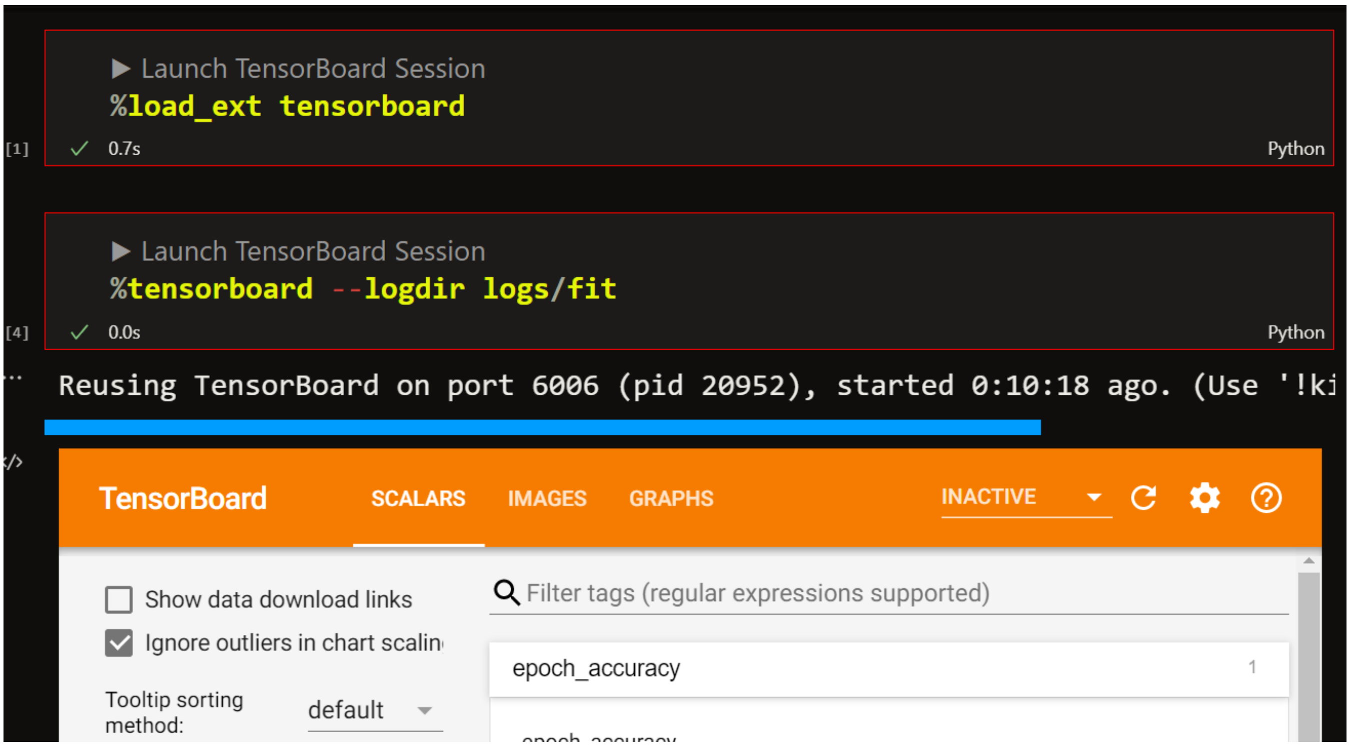
Code starts
Tensor board is a utility provided by TF to gather some information to visuallize the logs and all those things. Through this a board is created in which we can write or pass the information about our model data to visualize.



```python
#Writing samples to Tensor Boards
#Get unique timestamps
import time

time.asctime() #gives unique timestamps
def get_timestamp():
    return time.asctime().replace(" ","_").replace(":","")

def get_log_dir_path(log_dir="logs/fit"):
    log_path=os.path.join(log_dir,get_timestamp())
    print(f"Logs will be saved at:{log_path}")
    return log_path
```

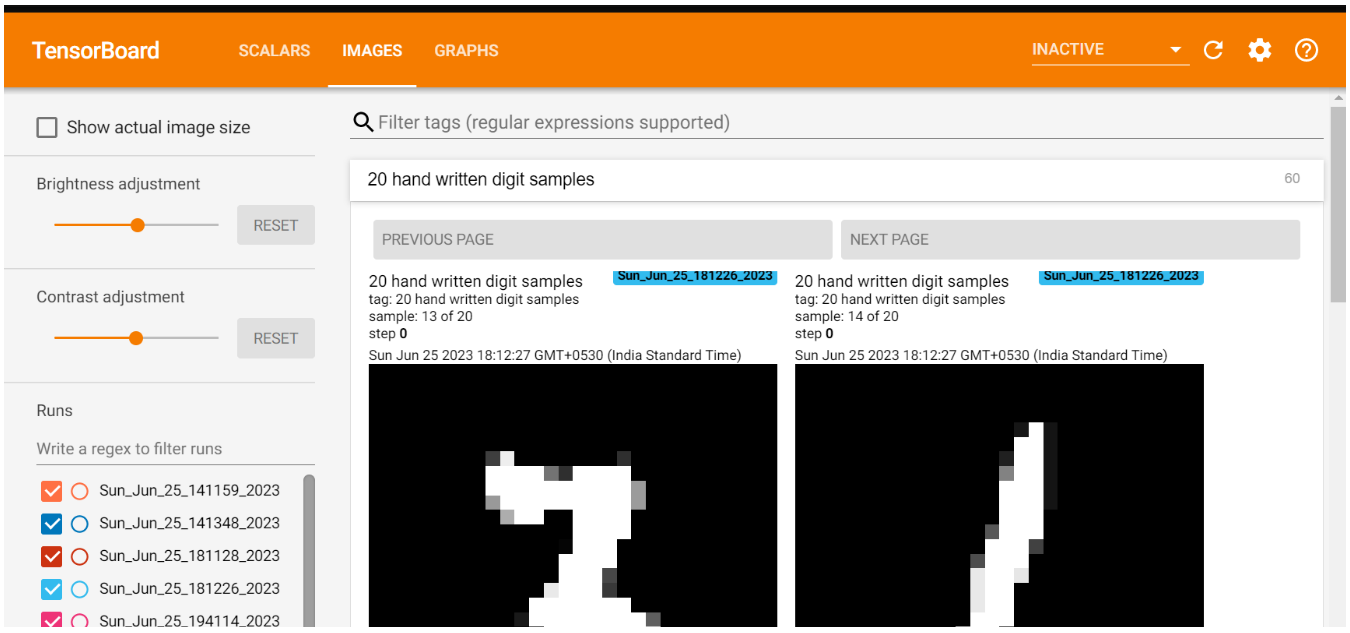


```python
log_dir=get_log_dir_path()
print(f"Log dir is {log_dir}")

file_writer=tf.summary.create_file_writer(logdir=log_dir)
#file_writer helps to wite the information on the tensor board

with file_writer.as_default():
    images=np.reshape(X_train[10:30],(-1,28,28,1)) #-1 is a batch here
    tf.summary.image("20 hand written digit samples",images,
                     max_outputs=25,step=0)

# Give below 2 command in ipynb file to open the tensorboard file
# %load_ext tensorboard
# %tensorboard --logdir logs/fit
```



```
▶ Launch TensorBoard Session
%load_ext tensorboard
[1]  ✓ 0.7s                                                    Python

▶ Launch TensorBoard Session
%tensorboard --logdir logs/fit
[4]  ✓ 0.0s                                                    Python

Reusing TensorBoard on port 6006 (pid 20952), started 0:10:18 ago. (Use '!ki
```

Tensor board allow us to visualize the data as shown below



Show actual image size

Brightness adjustment RESET

Contrast adjustment RESET

Runs

Write a regex to filter runs

- Sun_Jun_25_141159_2023
- Sun_Jun_25_141348_2023
- Sun_Jun_25_181128_2023
- Sun_Jun_25_181226_2023
- Sun_Jun_25_194114_2023

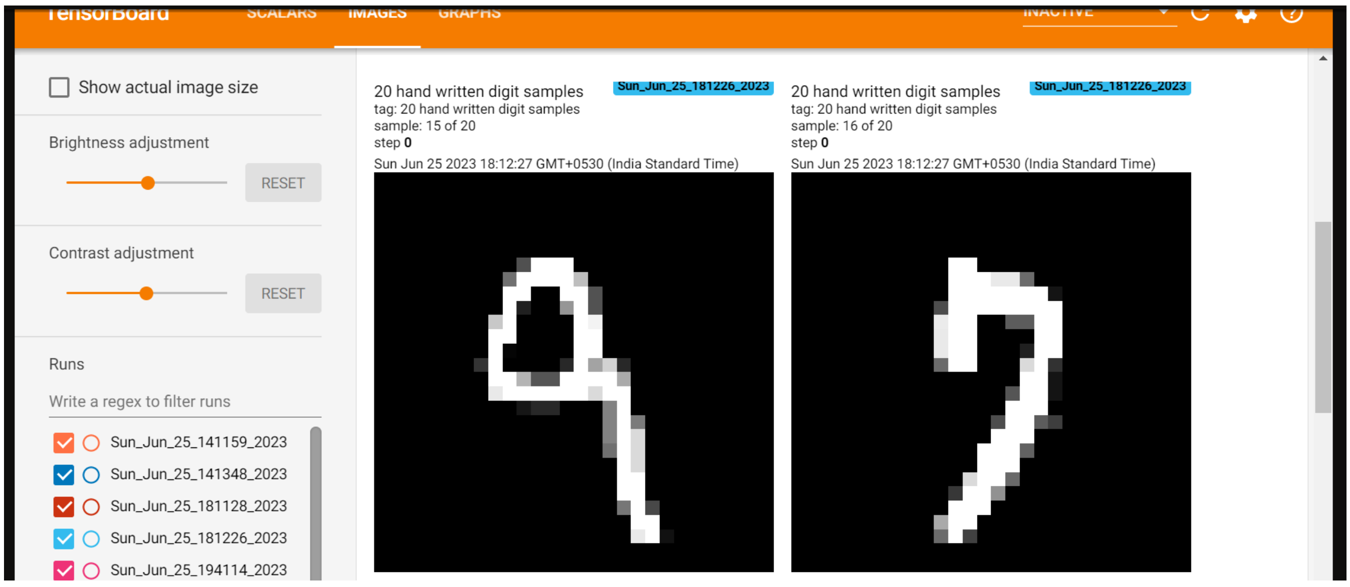
Filter tags (regular expressions supported)

20 hand written digit samples 60

PREVIOUS PAGE NEXT PAGE

20 hand written digit samples Sun_Jun_25_181226_2023
tag: 20 hand written digit samples
sample: 13 of 20
step **0**
Sun Jun 25 2023 18:12:27 GMT+0530 (India Standard Time)

20 hand written digit samples Sun_Jun_25_181226_2023
tag: 20 hand written digit samples
sample: 14 of 20
step **0**
Sun Jun 25 2023 18:12:27 GMT+0530 (India Standard Time)



Show actual image size

Brightness adjustment RESET

Contrast adjustment RESET

Runs

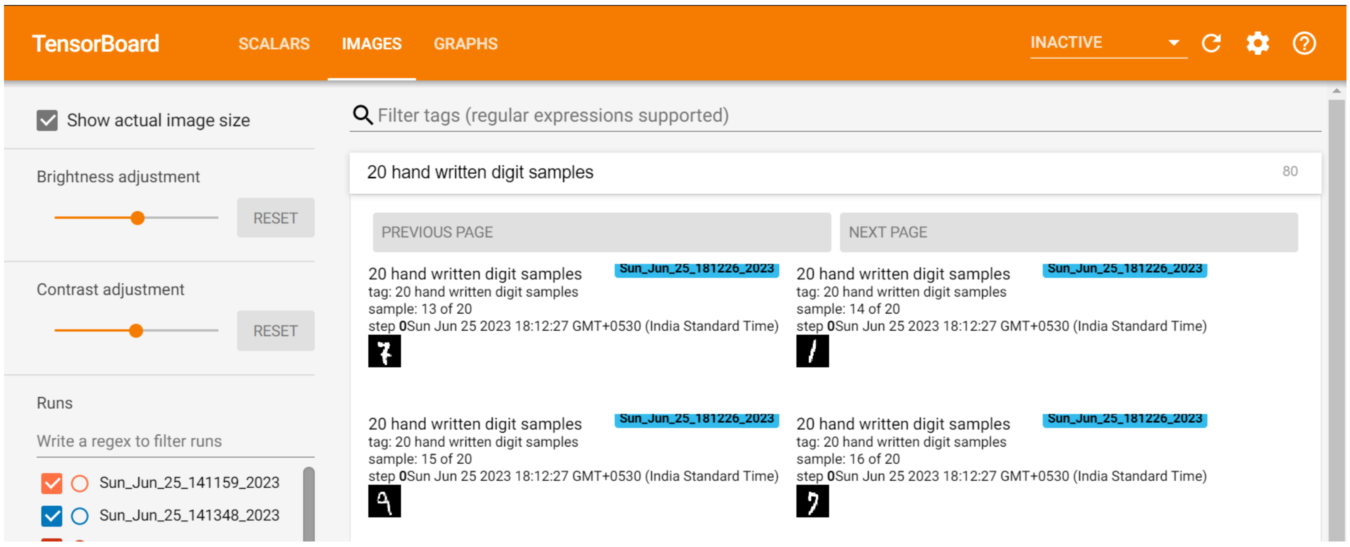
Write a regex to filter runs

- Sun_Jun_25_141159_2023
- Sun_Jun_25_141348_2023
- Sun_Jun_25_181128_2023
- Sun_Jun_25_181226_2023
- Sun_Jun_25_194114_2023

20 hand written digit samples Sun_Jun_25_181226_2023
tag: 20 hand written digit samples
sample: 15 of 20
step **0**
Sun Jun 25 2023 18:12:27 GMT+0530 (India Standard Time)

20 hand written digit samples Sun_Jun_25_181226_2023
tag: 20 hand written digit samples
sample: 16 of 20
step **0**
Sun Jun 25 2023 18:12:27 GMT+0530 (India Standard Time)

Below shows the original size of the image



Show actual image size

Brightness adjustment RESET

Contrast adjustment RESET

Runs

Write a regex to filter runs

- Sun_Jun_25_141159_2023
- Sun_Jun_25_141348_2023

Filter tags (regular expressions supported)

20 hand written digit samples 80

PREVIOUS PAGE NEXT PAGE

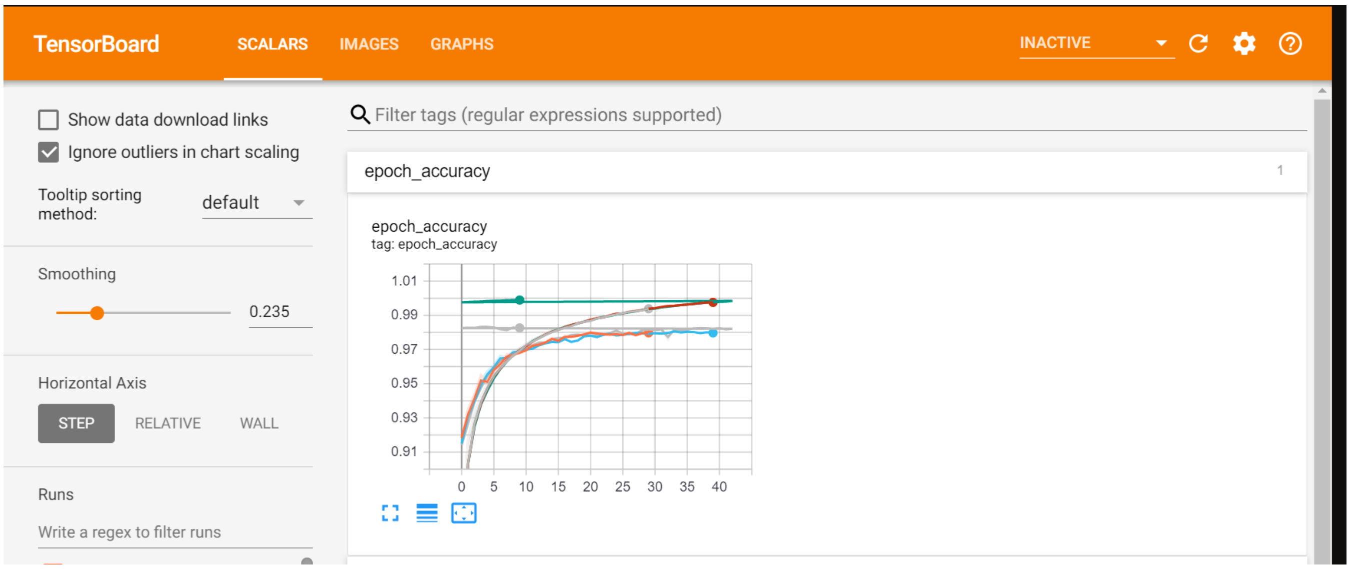
20 hand written digit samples Sun_Jun_25_181226_2023
tag: 20 hand written digit samples
sample: 13 of 20
step **0**Sun Jun 25 2023 18:12:27 GMT+0530 (India Standard Time)

20 hand written digit samples Sun_Jun_25_181226_2023
tag: 20 hand written digit samples
sample: 14 of 20
step **0**Sun Jun 25 2023 18:12:27 GMT+0530 (India Standard Time)

20 hand written digit samples Sun_Jun_25_181226_2023
tag: 20 hand written digit samples
sample: 15 of 20
step **0**Sun Jun 25 2023 18:12:27 GMT+0530 (India Standard Time)

20 hand written digit samples Sun_Jun_25_181226_2023
tag: 20 hand written digit samples
sample: 16 of 20
step **0**Sun Jun 25 2023 18:12:27 GMT+0530 (India Standard Time)

Below screen shows that after defining the Tensorboard call back the tensorfoard showing some more details



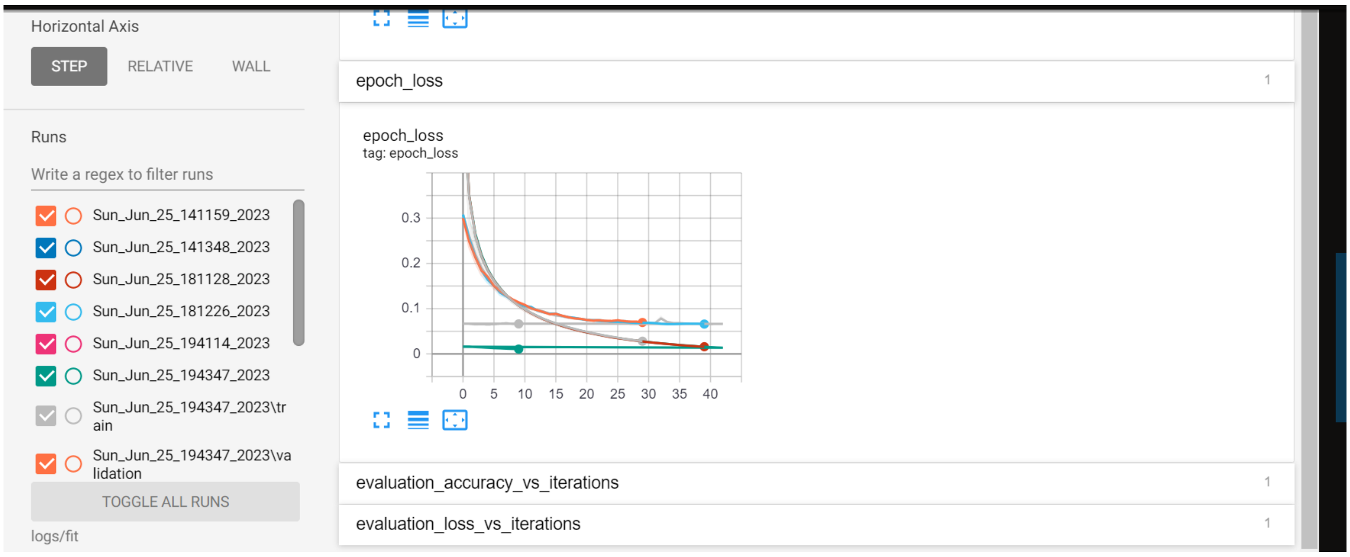
☐ Show data download links
☑ Ignore outliers in chart scaling

Tooltip sorting method: default

Smoothing 0.235

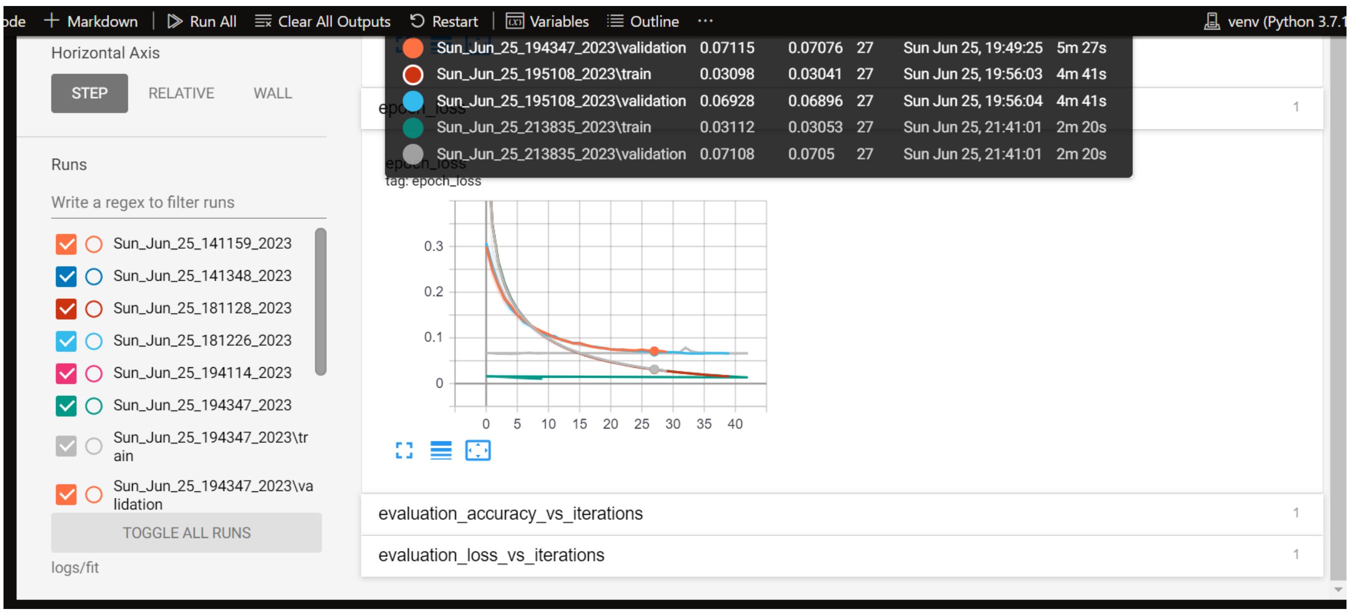
Horizontal Axis: [✓ STEP] RELATIVE WALL

epoch_accuracy 1



☑ Sun_Jun_25_141159_2023
☑ Sun_Jun_25_141348_2023
☑ Sun_Jun_25_181128_2023
☑ Sun_Jun_25_181226_2023
☑ Sun_Jun_25_194114_2023
☑ Sun_Jun_25_194347_2023
☑ Sun_Jun_25_194347_2023\train
☑ Sun_Jun_25_194347_2023\validation

evaluation_accuracy_vs_iterations 1
evaluation_loss_vs_iterations 1

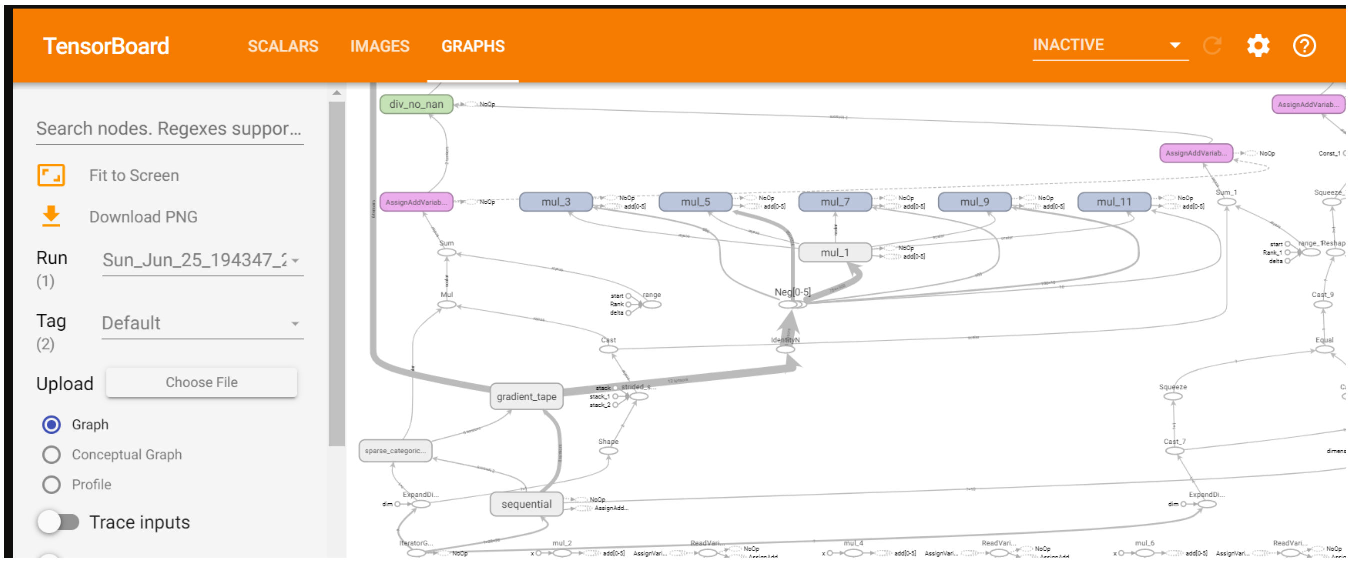


●	Sun_Jun_25_194347_2023\validation	0.07115	0.07076	27	Sun Jun 25, 19:49:25	5m 27s
●	Sun_Jun_25_195108_2023\train	0.03098	0.03041	27	Sun Jun 25, 19:56:03	4m 41s
●	Sun_Jun_25_195108_2023\validation	0.06928	0.06896	27	Sun Jun 25, 19:56:04	4m 41s
●	Sun_Jun_25_213835_2023\train	0.03112	0.03053	27	Sun Jun 25, 21:41:01	2m 20s
●	Sun_Jun_25_213835_2023\validation	0.07108	0.0705	27	Sun Jun 25, 21:41:01	2m 20s

☑ Sun_Jun_25_141159_2023
☑ Sun_Jun_25_141348_2023
☑ Sun_Jun_25_181128_2023
☑ Sun_Jun_25_181226_2023
☑ Sun_Jun_25_194114_2023
☑ Sun_Jun_25_194347_2023
☑ Sun_Jun_25_194347_2023\train
☑ Sun_Jun_25_194347_2023\validation

evaluation_accuracy_vs_iterations 1
evaluation_loss_vs_iterations 1

Below screen shows the training graph

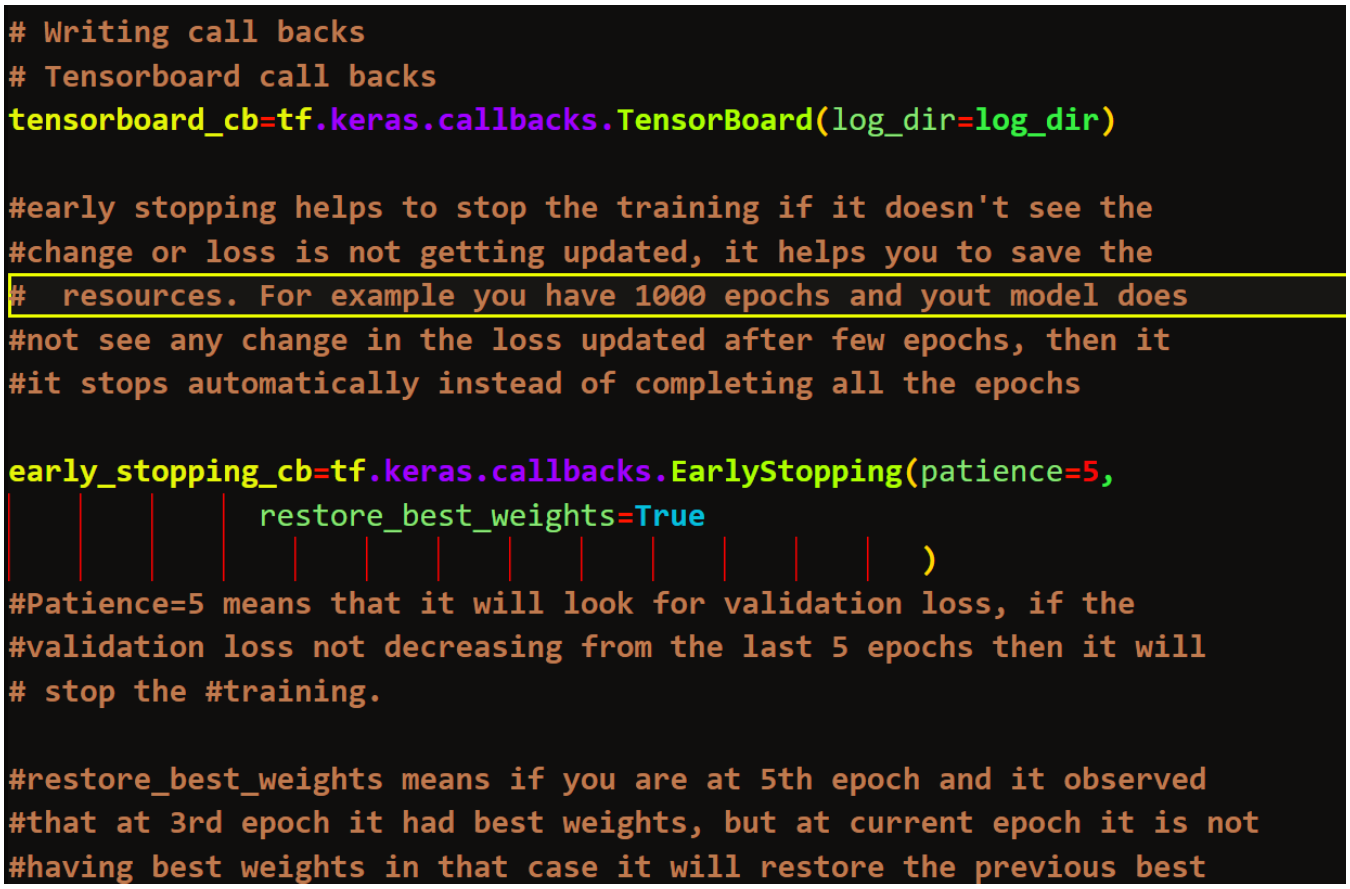


Run (1) Sun_Jun_25_194347_2

Tag (2) Default

Upload

[✓ Graph] Conceptual Graph Profile

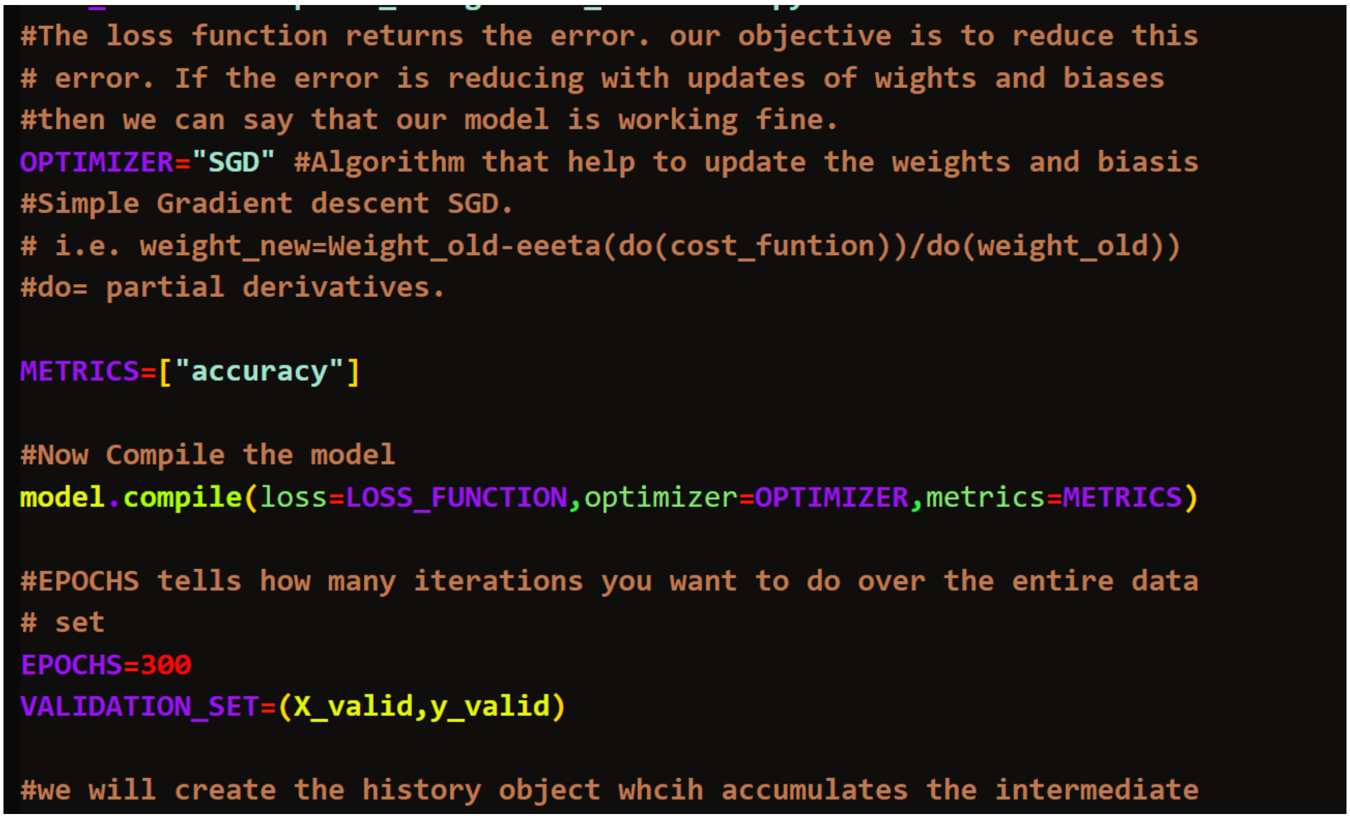


```
# Writing call backs
# Tensorboard call backs
tensorboard_cb=tf.keras.callbacks.TensorBoard(log_dir=log_dir)

#early stopping helps to stop the training if it doesn't see the
#change or loss is not getting updated, it helps you to save the
#  resources. For example you have 1000 epochs and yout model does
#not see any change in the loss updated after few epochs, then it
#it stops automatically instead of completing all the epochs

early_stopping_cb=tf.keras.callbacks.EarlyStopping(patience=5,
                                  restore_best_weights=True
                                                        )
#Patience=5 means that it will look for validation loss, if the
#validation loss not decreasing from the last 5 epochs then it will
# stop the #training.

#restore_best_weights means if you are at 5th epoch and it observed
#that at 3rd epoch it had best weights, but at current epoch it is not
#having best weights in that case it will restore the previous best
```

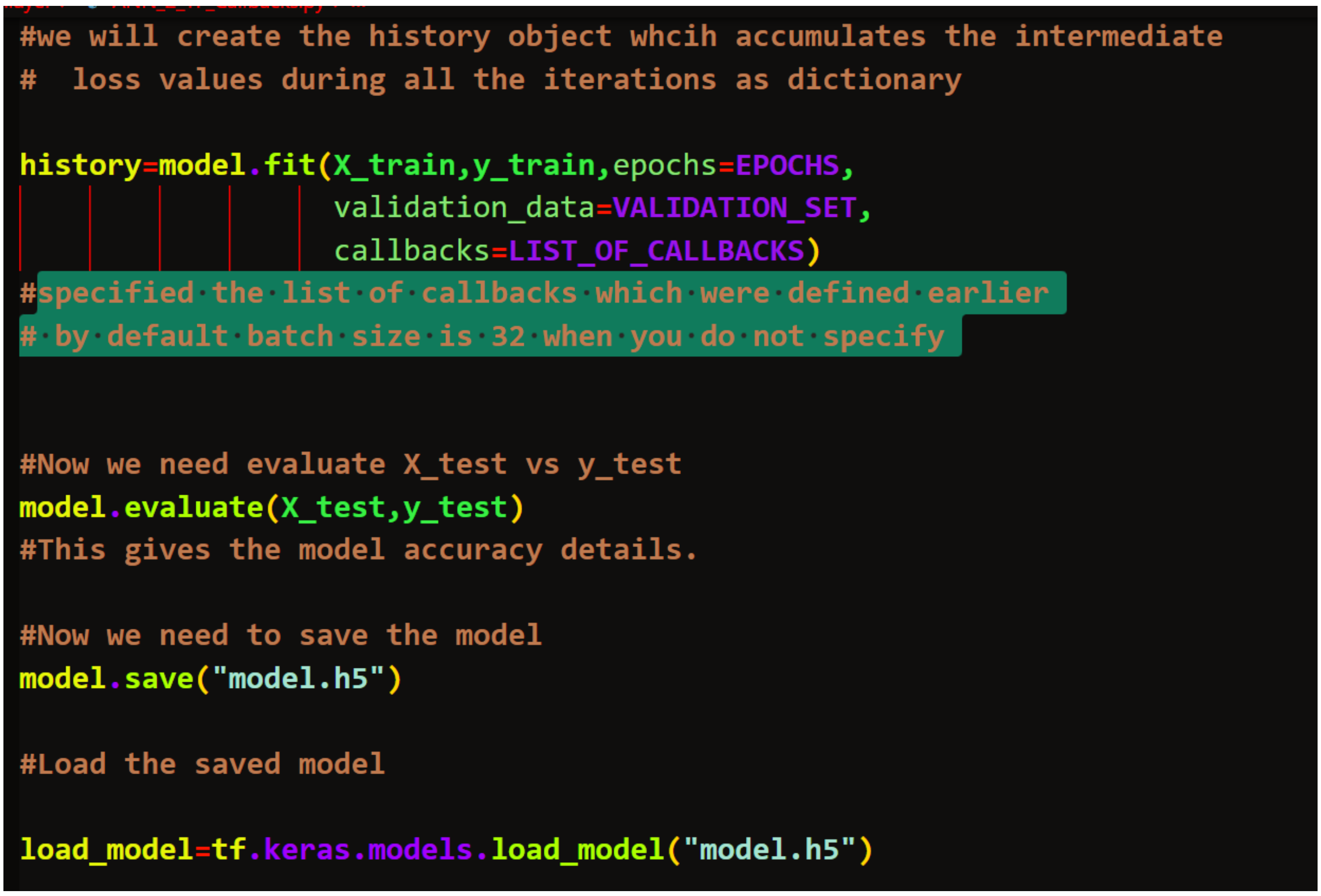
```
# best weights .

#Checkpoint call back
ckpt_path='model_ckpt.h5'
#for example if you are running 1000 epochs and if you are at 500 epoch
# then if there is a system power failure then you will be losing all
#the data till that point. In this case the checkpoint will be helpful
#the check point will be storing the intermediate data.
ckpt_cb=tf.keras.callbacks.ModelCheckpoint(ckpt_path,save_best_only=True)
#save_best_only helps to save the best weights details in the checkpoint
#otherwise if it stores the latest weight and the latest weight is
#not the best weight then the model will rebegin with these low weights.
#whichi is not good.
LIST_OF_CALLBACKS=[tensorboard_cb,early_stopping_cb,ckpt_cb]
```

```
#Building the ANN network
#Step1: Define layers
LAYERS=[
    tf.keras.layers.Flatten(input_shape=[28,28],name='inputlayer'),
    tf.keras.layers.Dense(300,activation='relu',name='hiddenlayer1'),
    tf.keras.layers.Dense(100,activation='relu',name='hiddenlayer2'),
    tf.keras.layers.Dense(10,activation='softmax',name='outputlayer'),
]
#Step2: Define model
model=tf.keras.models.Sequential(LAYERS)

print("model layers", model.layers)
print("model summary", model.summary())

#To get the layers
hidden1=model.layers[1]
print("model hidden layer 1",model.get_layer(hidden1.name))
```

```
#to get the weights of the layer
#This will gives out 2 arrays first array contains the weights of the
# layer, and second array contains the weights of the biases.


print("model layer weights", hidden1.get_weights())
#store the wights and bias weight into variable
weights,biases=hidden1.get_weights()

#Display the shape of the wights, gives as 784,300
print('Shape of layer wieght',weights.shape)
print("shape of the biase", biases.shape)

#Now we need to define the loss function and compile our model
#loss function will takes the actual and prediction value and gives the
#error as difference between them
LOSS_FUNCTION="sparse_categorical_crossentropy"
```

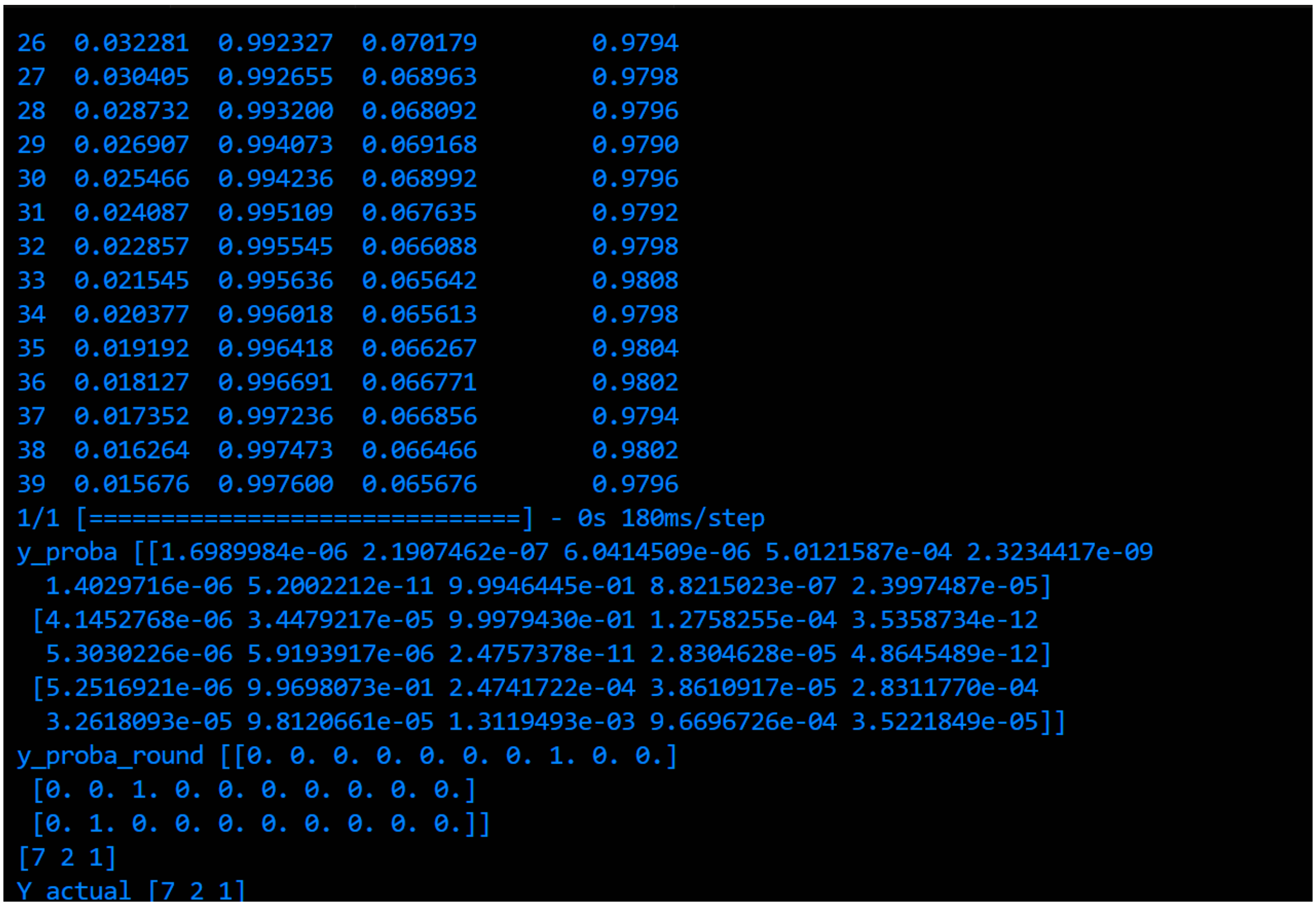


```
#The loss function returns the error. our objective is to reduce this
# error. If the error is reducing with updates of wights and biases
#then we can say that our model is working fine.
OPTIMIZER="SGD" #Algorithm that help to update the weights and biasis
#Simple Gradient descent SGD.
# i.e. weight_new=Weight_old-eeeta(do(cost_funtion))/do(weight_old))
#do= partial derivatives.

METRICS=["accuracy"]

#Now Compile the model
model.compile(loss=LOSS_FUNCTION,optimizer=OPTIMIZER,metrics=METRICS)

#EPOCHS tells how many iterations you want to do over the entire data
# set
EPOCHS=300
VALIDATION_SET=(X_valid,y_valid)

#we will create the history object whcih accumulates the intermediate
```

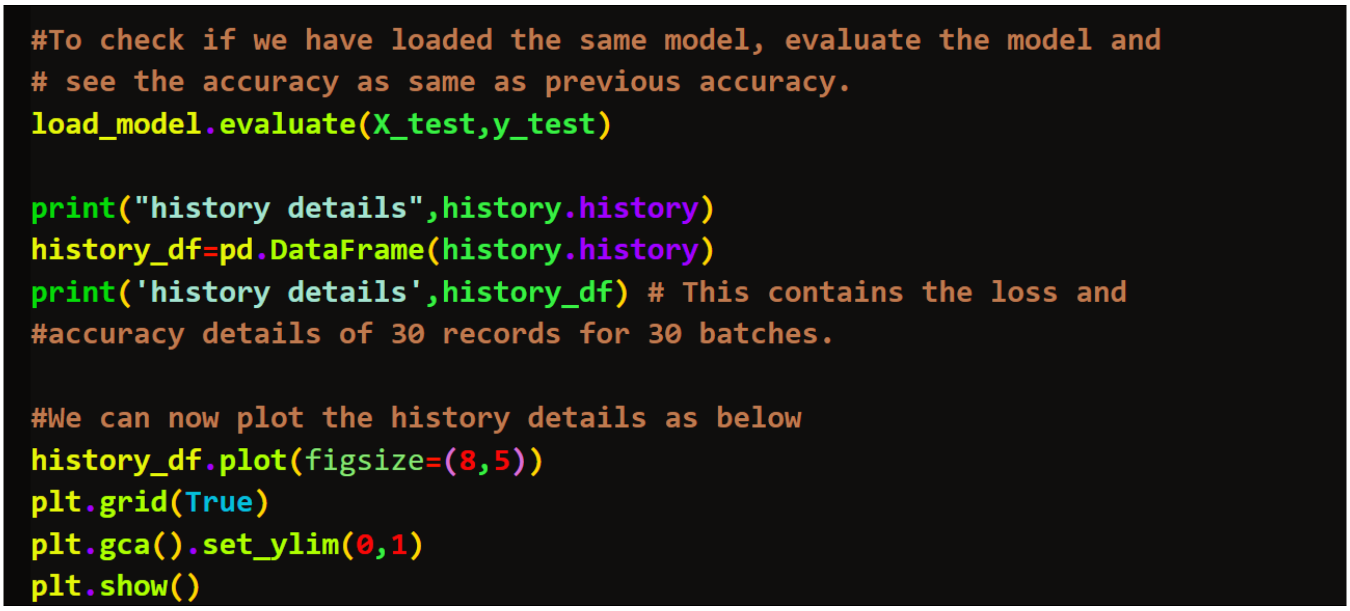


```
#we will create the history object whcih accumulates the intermediate
#  loss values during all the iterations as dictionary

history=model.fit(X_train,y_train,epochs=EPOCHS,
                  validation_data=VALIDATION_SET,
                  callbacks=LIST_OF_CALLBACKS)
#specified the list of callbacks which were defined earlier
# by default batch size is 32 when you do not specify


#Now we need evaluate X_test vs y_test
model.evaluate(X_test,y_test)
#This gives the model accuracy details.

#Now we need to save the model
model.save("model.h5")

#Load the saved model

load_model=tf.keras.models.load_model("model.h5")
```

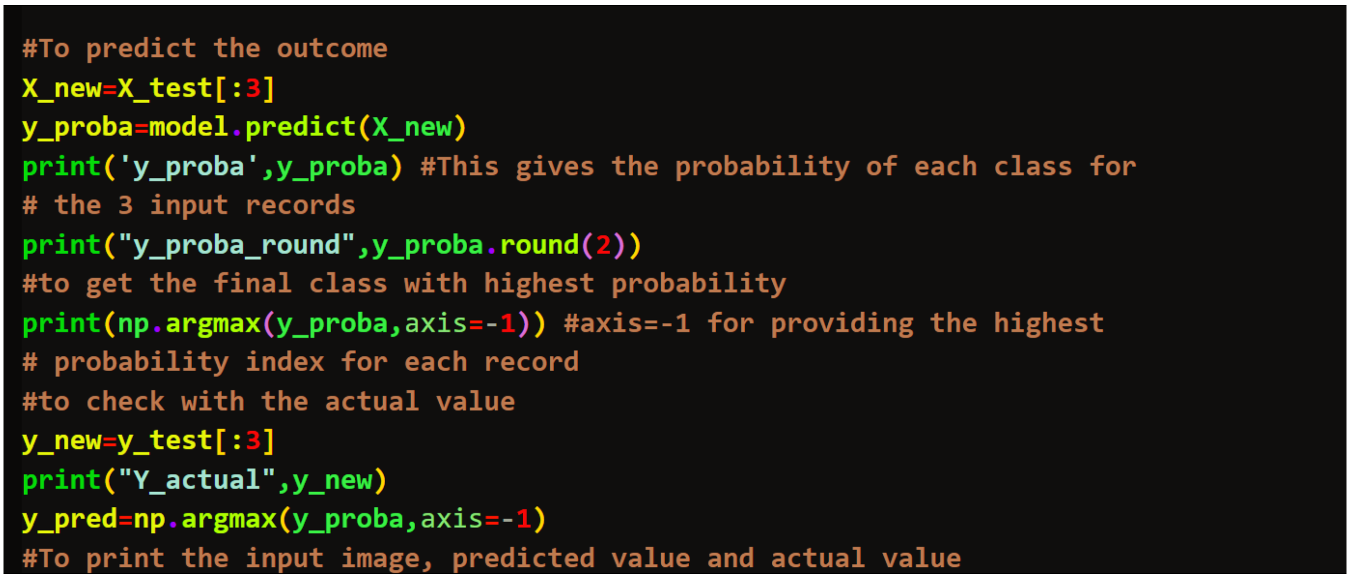
Because of specifying early stop the model training has been stopped at 39 epoch only as shown below



```
26  0.032281  0.992327  0.070179        0.9794
27  0.030405  0.992655  0.068963        0.9798
28  0.028732  0.993200  0.068092        0.9796
29  0.026907  0.994073  0.069168        0.9790
30  0.025466  0.994236  0.068992        0.9796
31  0.024087  0.995109  0.067635        0.9792
32  0.022857  0.995545  0.066088        0.9798
33  0.021545  0.995636  0.065642        0.9808
34  0.020377  0.996018  0.065613        0.9798
35  0.019192  0.996418  0.066267        0.9804
36  0.018127  0.996691  0.066771        0.9802
37  0.017352  0.997236  0.066856        0.9794
38  0.016264  0.997473  0.066466        0.9802
39  0.015676  0.997600  0.065676        0.9796
1/1 [==============================] - 0s 180ms/step
y_proba [[1.6989984e-06 2.1907462e-07 6.0414509e-06 5.0121587e-04 2.3234417e-09
  1.4029716e-06 5.2002212e-11 9.9946445e-01 8.8215023e-07 2.3997487e-05]
 [4.1452768e-06 3.4479217e-05 9.9979430e-01 1.2758255e-04 3.5358734e-12
  5.3030226e-06 5.9193917e-06 2.4757378e-11 2.8304628e-05 4.8645489e-12]
 [5.2516921e-06 9.9698073e-01 2.4741722e-04 3.8610917e-05 2.8311770e-04
  3.2618093e-05 9.8120661e-05 1.3119493e-03 9.6696726e-04 3.5221849e-05]]
y_proba_round [[0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]
 [0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]]
[7 2 1]
Y actual [7 2 1]
```

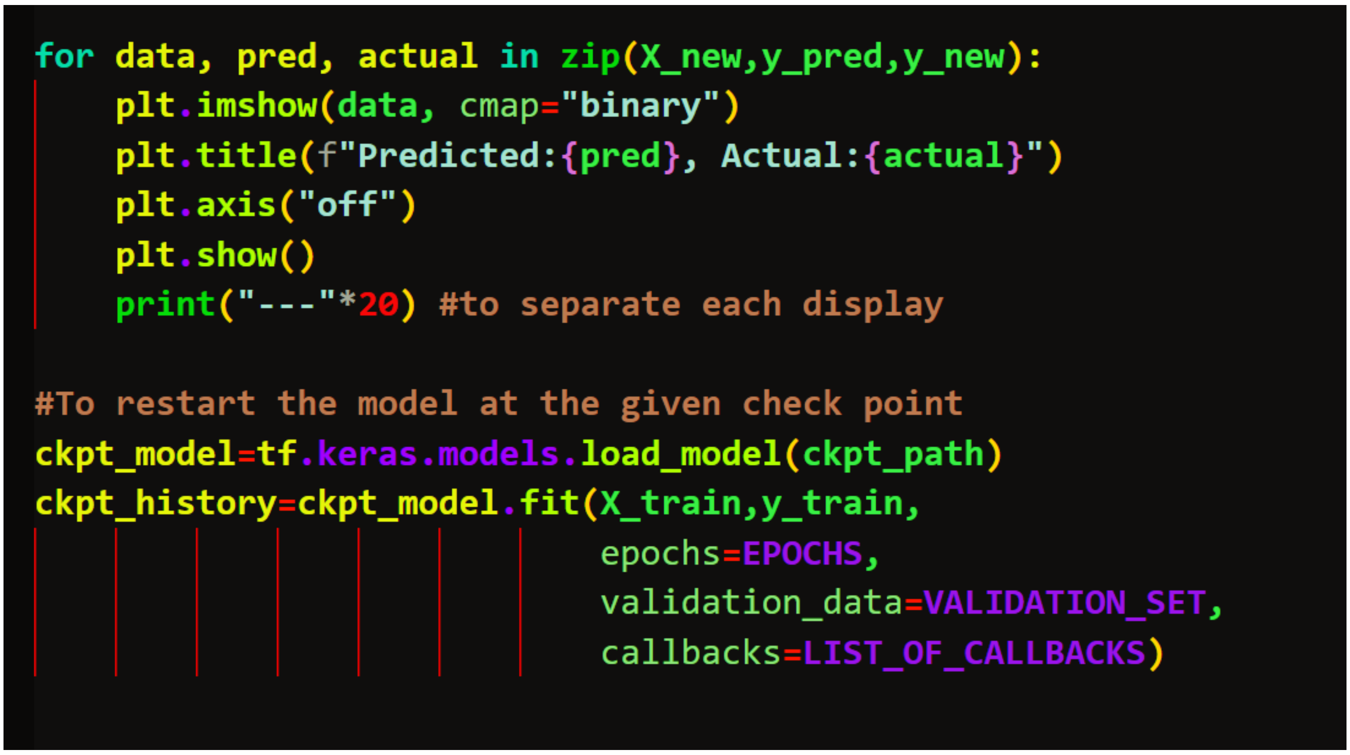


```
#To check if we have loaded the same model, evaluate the model and
# see the accuracy as same as previous accuracy.
load_model.evaluate(X_test,y_test)

print("history details",history.history)
history_df=pd.DataFrame(history.history)
print('history details',history_df) # This contains the loss and
#accuracy details of 30 records for 30 batches.

#We can now plot the history details as below
history_df.plot(figsize=(8,5))
plt.grid(True)
plt.gca().set_ylim(0,1)
plt.show()
```

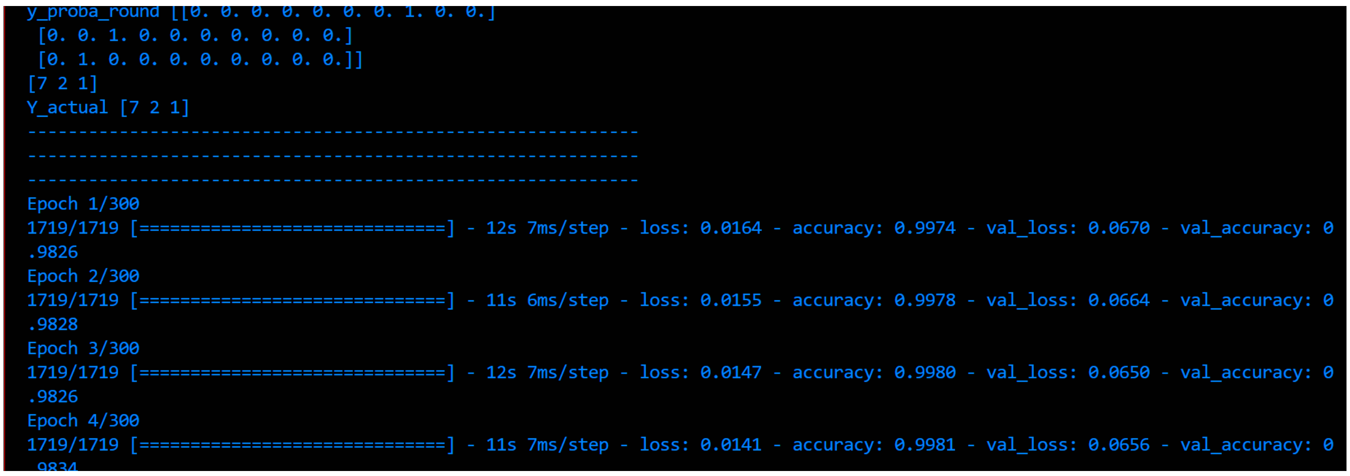


```
#To predict the outcome
X_new=X_test[:3]
y_proba=model.predict(X_new)
print('y_proba',y_proba) #This gives the probability of each class for
# the 3 input records
print("y_proba_round",y_proba.round(2))
#to get the final class with highest probability
print(np.argmax(y_proba,axis=-1)) #axis=-1 for providing the highest
# probability index for each record
#to check with the actual value
y_new=y_test[:3]
print("Y_actual",y_new)
y_pred=np.argmax(y_proba,axis=-1)
#To print the input image, predicted value and actual value
```

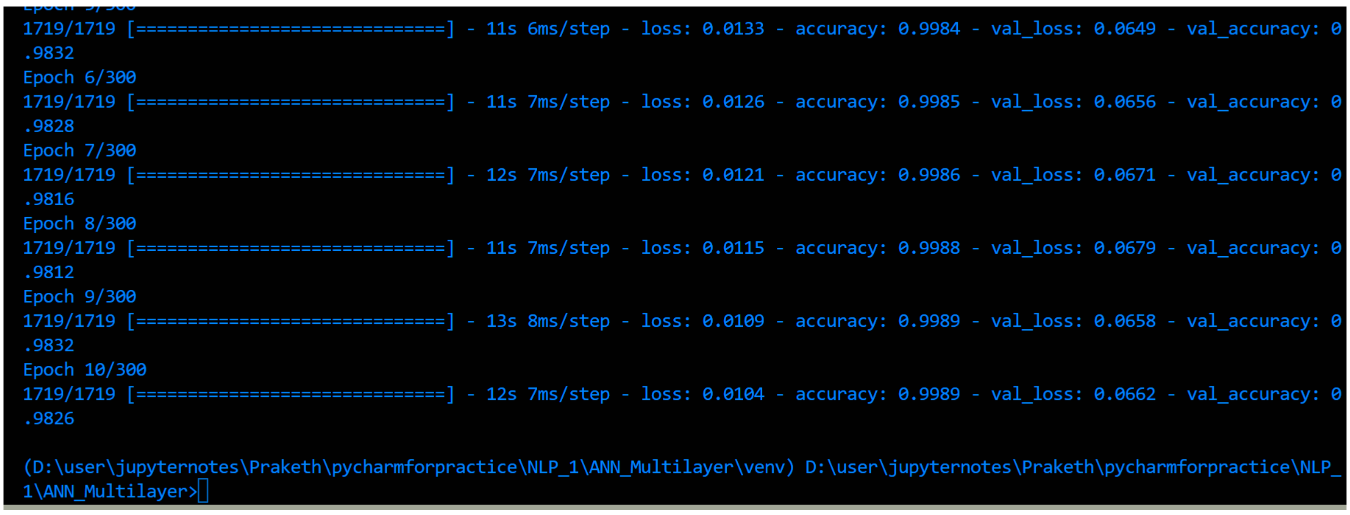
Here in the below code the checkpoint restart has been done where the model retraining started from loading the checkpoint restored model with starting at the best weights.



```
for data, pred, actual in zip(X_new,y_pred,y_new):
    plt.imshow(data, cmap="binary")
    plt.title(f"Predicted:{pred}, Actual:{actual}")
    plt.axis("off")
    plt.show()
    print("---"*20) #to separate each display

#To restart the model at the given check point
ckpt_model=tf.keras.models.load_model(ckpt_path)
ckpt_history=ckpt_model.fit(X_train,y_train,
                            epochs=EPOCHS,
                            validation_data=VALIDATION_SET,
                            callbacks=LIST_OF_CALLBACKS)
```



```
y_proba_round [[0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]
 [0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]]
[7 2 1]
Y_actual [7 2 1]
------------------------------------------------------------
------------------------------------------------------------
------------------------------------------------------------
Epoch 1/300
1719/1719 [==============================] - 12s 7ms/step - loss: 0.0164 - accuracy: 0.9974 - val_loss: 0.0670 - val_accuracy: 0
.9826
Epoch 2/300
1719/1719 [==============================] - 11s 6ms/step - loss: 0.0155 - accuracy: 0.9978 - val_loss: 0.0664 - val_accuracy: 0
.9828
Epoch 3/300
1719/1719 [==============================] - 12s 7ms/step - loss: 0.0147 - accuracy: 0.9980 - val_loss: 0.0650 - val_accuracy: 0
.9826
Epoch 4/300
1719/1719 [==============================] - 11s 7ms/step - loss: 0.0141 - accuracy: 0.9981 - val_loss: 0.0656 - val_accuracy: 0
.9834
```



```
1719/1719 [==============================] - 11s 6ms/step - loss: 0.0133 - accuracy: 0.9984 - val_loss: 0.0649 - val_accuracy: 0
.9832
Epoch 6/300
1719/1719 [==============================] - 11s 7ms/step - loss: 0.0126 - accuracy: 0.9985 - val_loss: 0.0656 - val_accuracy: 0
.9828
Epoch 7/300
1719/1719 [==============================] - 12s 7ms/step - loss: 0.0121 - accuracy: 0.9986 - val_loss: 0.0671 - val_accuracy: 0
.9816
Epoch 8/300
1719/1719 [==============================] - 11s 7ms/step - loss: 0.0115 - accuracy: 0.9988 - val_loss: 0.0679 - val_accuracy: 0
.9812
Epoch 9/300
1719/1719 [==============================] - 13s 8ms/step - loss: 0.0109 - accuracy: 0.9989 - val_loss: 0.0658 - val_accuracy: 0
.9832
Epoch 10/300
1719/1719 [==============================] - 12s 7ms/step - loss: 0.0104 - accuracy: 0.9989 - val_loss: 0.0662 - val_accuracy: 0
.9826

(D:\user\jupyternotes\Praketh\pycharmforpractice\NLP_1\ANN_Multilayer\venv) D:\user\jupyternotes\Praketh\pycharmforpractice\NLP_
1\ANN_Multilayer>
```