Problem statement

- we are given with image data which belongs to 16 categories...we need to build CNN model to classify the given image into its category using transfer learning with VGG16 model...

Transfer Learning

Download all the data in this rar_file (https://drive.google.com/open?id=1Z4TyI7FcFVEx8qdl4jO9qxvxaqLSqoEu) , it contains all the data required for the project. When you unrar the file you'll get the files in the following format: **path/to/the/image.tif,category**

```
where the categories are numbered 0 to 15, in the following order:
```

```
0 letter
1 form
2 email
3 handwritten
4 advertisement
5 scientific report
6 scientific publication
7 specification
8 file folder
9 news article
10 budget
11 invoice
12 presentation
13 questionnaire
14 resume
15 memo
```

There is a file named as 'labels_final.csv' , it consists of two columns. First column is path which is the required path to the images and second is the class label.

In [ ]:

```
#the dataset that we are dealing with is quite large 3.7 GB and hence there are two methods to import the data to Colab
# Method 1- you can use gdown module to get the data directly from Google drive to Colab
# the syntax is as follows !gdown --id file_id , for ex - running the below cell will import the rvl-cdip.rar dataset
```

In [ ]:

```
#!gdown --id 1Z4TyI7FcFVEx8qdl4jO9qxvxaqLSqoEu
```

In [ ]:

```
# Method -2 you can also import the data using wget function
#https://www.youtube.com/watch?v=BPUfVq7RaY8
```

In [ ]:

```
#unrar the file
#get_ipython().system_raw("unrar x rvl-cdip.rar")
```

On this image data, we are training 3 types of models as given below we have to split the data into Train and Validation data.

Model-1

```
1. Use VGG-16 (https://www.tensorflow.org/api_docs/python/tf/keras/applications/VGG16) pretrained network without Fully
Connected layers and initilize all the weights with Imagenet trained weights.
2. After VGG-16 network without FC layers, add a new Conv block ( 1 Conv layer and 1 Maxpooling ), 2 FC layers and an ou
tput layer to classify 16 classes. You are free to choose any hyperparameters/parameters of conv block, FC layers, outpu
t layer.
3. Final architecture will be INPUT --> VGG-16 without Top layers(FC) --> Conv Layer --> Maxpool Layer --> 2 FC layers -
-> Output Layer
4.Print model.summary() and plot the architecture of the model.
Reference for plotting model (https://www.tensorflow.org/api_docs/python/tf/keras/utils/plot_model)
5. Train only new Conv block, FC layers, output layer. Don't train the VGG-16 network.
```

In [ ]:

```
!curl --header "Host: www.kaggle.com" --header "User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
100 4442M  100 4442M    0     0  20.7M      0  0:03:33  0:03:33 --:--:-- 22.6M
```

In [ ]:

```
#!wget --header='Host: www.kaggle.com' --header='User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
```

In [ ]:

```
#!kaggle datasets download -d brahma0545/aaic-assignment-tl
```

In [ ]:

```
import zipfile

with zipfile.ZipFile('/content/archive.zip', 'r') as zip_ref:
    zip_ref.extractall()
```

In [ ]:

```
import matplotlib.pyplot as plt # importing the libraries
import pandas as pd
import numpy as np
import seaborn as sns
import tensorflow as tf
import datetime, os
from tensorflow import keras
from keras.models import Model
```

In [ ]:

```
#get_ipython().system_raw("unrar x archive.zip") # extracting the uploaded file
```

In [ ]:

```
df=pd.read_csv("labels_final.csv")
```

In [ ]:

```
df.head(5)
```

Out[6]:

	path	label
0	imagesv/v/o/h/voh71d00/509132755+-2755.tif	3
1	imagesl/l/x/t/lxt19d00/502213303.tif	3
2	imagesx/x/e/d/xed05a00/2075325674.tif	2
3	imageso/o/j/b/ojb60d00/517511301+-1301.tif	3
4	imagesq/q/z/k/qzk17e00/2031320195.tif	7

In [ ]:

```
labels_dict={ 0 :"letter",
    1 :"form",
    2 :"email",
    3 :"handwritten",
    4 :"advertisement",
    5 :"scientific report",
    6 :"scientific publication",
    7 :"specification",
    8 :"file folder",
    9 :"news article",
    10 :" budget",
    11 :"invoice",
    12 :" presentation",
    13 :"questionnaire",
    14 :"resume",
    15: "memo"}
```

In [ ]:

```
df['label']=df['label'].apply(lambda x:labels_dict[x])
df.head(5)
```

Out[6]:

	path	label
0	imagesv/v/o/h/voh71d00/509132755+-2755.tif	handwritten
1	imagesl/l/x/t/lxt19d00/502213303.tif	handwritten
2	imagesx/x/e/d/xed05a00/2075325674.tif	email
3	imageso/o/j/b/ojb60d00/517511301+-1301.tif	handwritten
4	imagesq/q/z/k/qzk17e00/2031320195.tif	specification

In [ ]:

```
from keras_preprocessing.image import ImageDataGenerator
datagen = ImageDataGenerator(rescale=1/255., validation_split=0.2) #image generator

print("------TRAIN DATA-------")   # train data
train_generator = datagen.flow_from_dataframe(dataframe=df, directory="/content/data_final",
                                             x_col='path',
                                             y_col='label', # using flow from data frame
                                    target_size=(256,256),
                                             class_mode='categorical',
                                             batch_size=32,
                                             drop_remainder=True,
                                             subset='training',
                                             seed=7)
print("------CROSS VALIDATION DATA-------") # cross validation data

validation_generator = datagen.flow_from_dataframe(dataframe=df, directory="/content/data_final",
                                             x_col='path',
                                             y_col='label',
                                             target_size=(256,256),
                                             class_mode='categorical',
                                             batch_size=32,
                                              drop_remainder=True,
                                             subset='validation',
                                             seed=7)
```

```
------TRAIN DATA-------
Found 38400 validated image filenames belonging to 16 classes.
------CROSS VALIDATION DATA-------
Found 9600 validated image filenames belonging to 16 classes.
```

In [ ]:

```
import tensorflow as tf
from tensorflow.keras.layers import Input, Lambda, Dense, Flatten
from tensorflow.keras.models import Model
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.applications.vgg16 import preprocess_input
from tensorflow.keras.preprocessing import image
from tensorflow.keras.layers import Dense, Conv2D, MaxPool2D , Flatten
from tensorflow.keras.callbacks import Callback
from tensorflow.keras.callbacks import TensorBoard
```

In [ ]:

```
%load_ext tensorboard
```

In [ ]:

```
logdir="logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S") # tensorboard
tensorboard_callback = TensorBoard(log_dir=logdir, histogram_freq=0)
```

In [ ]:

```
IMAGE_SIZE = [256, 256] #pre trained vgg16 model
model = VGG16(input_shape=IMAGE_SIZE + [3], weights='imagenet', include_top=False)

model.summary() #pre trained vgg16 model
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_order
ing_tf_kernels_notop.h5 (https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_or
dering_tf_kernels_notop.h5)
58889256/58889256 [==============================] - 4s 0us/step
Model: "vgg16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_1 (InputLayer)        [(None, 256, 256, 3)]     0         
                                                                 
 block1_conv1 (Conv2D)       (None, 256, 256, 64)      1792      
                                                                 
 block1_conv2 (Conv2D)       (None, 256, 256, 64)      36928     
                                                                 
 block1_pool (MaxPooling2D)  (None, 128, 128, 64)      0         
                                                                 
 block2_conv1 (Conv2D)       (None, 128, 128, 128)     73856     
                                                                 
 block2_conv2 (Conv2D)       (None, 128, 128, 128)     147584    
                                                                 
 block2_pool (MaxPooling2D)  (None, 64, 64, 128)       0         
                                                                 
 block3_conv1 (Conv2D)       (None, 64, 64, 256)       295168    
                                                                 
 block3_conv2 (Conv2D)       (None, 64, 64, 256)       590080    
                                                                 
 block3_conv3 (Conv2D)       (None, 64, 64, 256)       590080    
                                                                 
 block3_pool (MaxPooling2D)  (None, 32, 32, 256)       0         
                                                                 
 block4_conv1 (Conv2D)       (None, 32, 32, 512)       1180160   
                                                                 
 block4_conv2 (Conv2D)       (None, 32, 32, 512)       2359808   
                                                                 
 block4_conv3 (Conv2D)       (None, 32, 32, 512)       2359808   
                                                                 
 block4_pool (MaxPooling2D)  (None, 16, 16, 512)       0         
                                                                 
 block5_conv1 (Conv2D)       (None, 16, 16, 512)       2359808   
                                                                 
 block5_conv2 (Conv2D)       (None, 16, 16, 512)       2359808   
                                                                 
 block5_conv3 (Conv2D)       (None, 16, 16, 512)       2359808   
                                                                 
 block5_pool (MaxPooling2D)  (None, 8, 8, 512)         0         
                                                                 
=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:

```
train_steps = train_generator.n//train_generator.batch_size
validation_steps = validation_generator.n//validation_generator.batch_size
```

In [ ]:

```
#model_1
for layer in model.layers:
  layer.trainable = False
#Adding custom Layers
x = model.output
x = Conv2D(filters=512,kernel_size=(3,3),padding="same", activation="relu")(x)
x = MaxPool2D(2,2)(x)
x = Flatten()(x)
x = Dense(256, activation="relu")(x)
x = Dense(128, activation="relu")(x)
output = Dense(16, activation="softmax")(x)
# creating the final model
model_1 = Model(inputs = model.input, outputs = output)
# compile the model
model_1.compile(loss = "categorical_crossentropy", optimizer ='Adam', metrics=["accuracy"])
```

In [ ]:

```
# summary of the model_1
model_1.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_1 (InputLayer)        [(None, 256, 256, 3)]     0         
                                                                 
 block1_conv1 (Conv2D)       (None, 256, 256, 64)      1792      
                                                                 
 block1_conv2 (Conv2D)       (None, 256, 256, 64)      36928     
                                                                 
 block1_pool (MaxPooling2D)  (None, 128, 128, 64)      0         
                                                                 
 block2_conv1 (Conv2D)       (None, 128, 128, 128)     73856     
                                                                 
 block2_conv2 (Conv2D)       (None, 128, 128, 128)     147584    
                                                                 
 block2_pool (MaxPooling2D)  (None, 64, 64, 128)       0         
                                                                 
 block3_conv1 (Conv2D)       (None, 64, 64, 256)       295168    
                                                                 
 block3_conv2 (Conv2D)       (None, 64, 64, 256)       590080    
                                                                 
 block3_conv3 (Conv2D)       (None, 64, 64, 256)       590080    
                                                                 
 block3_pool (MaxPooling2D)  (None, 32, 32, 256)       0         
                                                                 
 block4_conv1 (Conv2D)       (None, 32, 32, 512)       1180160   
                                                                 
 block4_conv2 (Conv2D)       (None, 32, 32, 512)       2359808   
                                                                 
 block4_conv3 (Conv2D)       (None, 32, 32, 512)       2359808   
                                                                 
 block4_pool (MaxPooling2D)  (None, 16, 16, 512)       0         
                                                                 
 block5_conv1 (Conv2D)       (None, 16, 16, 512)       2359808   
                                                                 
 block5_conv2 (Conv2D)       (None, 16, 16, 512)       2359808   
                                                                 
 block5_conv3 (Conv2D)       (None, 16, 16, 512)       2359808   
                                                                 
 block5_pool (MaxPooling2D)  (None, 8, 8, 512)         0         
                                                                 
 conv2d (Conv2D)             (None, 8, 8, 512)         2359808   
                                                                 
 max_pooling2d (MaxPooling2D  (None, 4, 4, 512)        0         
 )                                                               
                                                                 
 flatten (Flatten)           (None, 8192)              0         
                                                                 
 dense (Dense)               (None, 256)               2097408   
                                                                 
 dense_1 (Dense)             (None, 128)               32896     
                                                                 
 dense_2 (Dense)             (None, 16)                2064      
                                                                 
=================================================================
Total params: 19,206,864
Trainable params: 4,492,176
Non-trainable params: 14,714,688
_________________________________________________________________
```

In [ ]:

```
#fitting the model_1
model_1.fit_generator(train_generator,steps_per_epoch=train_steps, epochs=5,
                              validation_data=validation_generator,validation_steps=validation_steps,callbacks=[tensorboard_callb
```

```
<ipython-input-19-f33c75d26d90>:4: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future
version. Please use `Model.fit`, which supports generators.
  model_1.fit_generator(train_generator,steps_per_epoch=train_steps, epochs=5,

Epoch 1/5
1200/1200 [==============================] - 333s 266ms/step - loss: 1.3061 - accuracy: 0.5939 - val_loss: 1.0037 -
val_accuracy: 0.6965
Epoch 2/5
1200/1200 [==============================] - 352s 294ms/step - loss: 0.8966 - accuracy: 0.7256 - val_loss: 0.8949 -
val_accuracy: 0.7352
Epoch 3/5
1200/1200 [==============================] - 315s 262ms/step - loss: 0.7528 - accuracy: 0.7684 - val_loss: 0.8907 -
val_accuracy: 0.7389
Epoch 4/5
1200/1200 [==============================] - 309s 258ms/step - loss: 0.6509 - accuracy: 0.7988 - val_loss: 0.8663 -
val_accuracy: 0.7506
Epoch 5/5
1200/1200 [==============================] - 314s 262ms/step - loss: 0.5694 - accuracy: 0.8202 - val_loss: 0.8897 -
val_accuracy: 0.7464
```

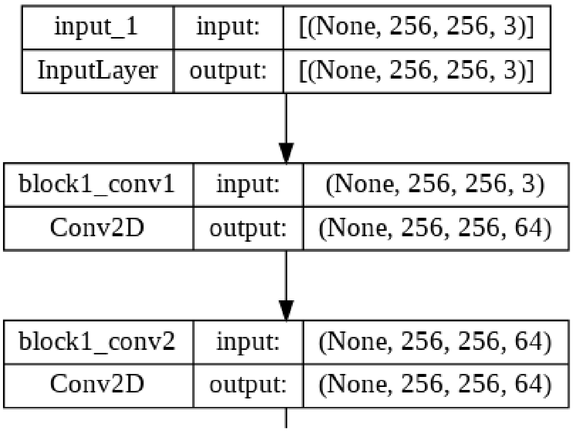
Out[19]:

```
<keras.callbacks.History at 0x7f37f54c7220>
```

In [ ]:

```
# model graphs
tf.keras.utils.plot_model(
    model_1, to_file='model_1.png', show_shapes=True, show_layer_names=True,
    rankdir='TB', expand_nested=False, dpi=96
)
```

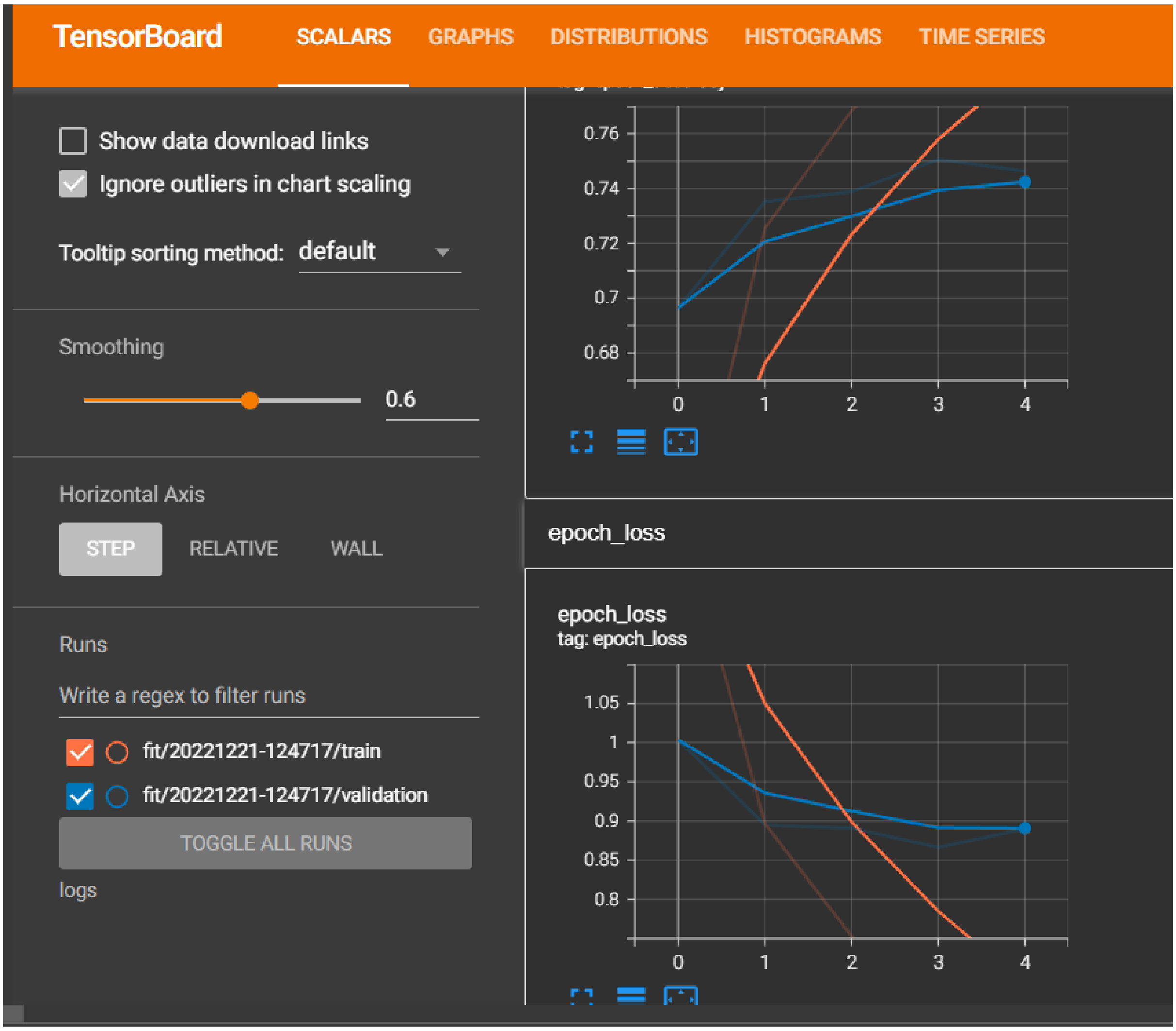
Out[20]:



In [ ]:

```
!rm -rf ./logs/
```

In [ ]:



□ Show data download links
☑ Ignore outliers in chart scaling

Tooltip sorting method: default

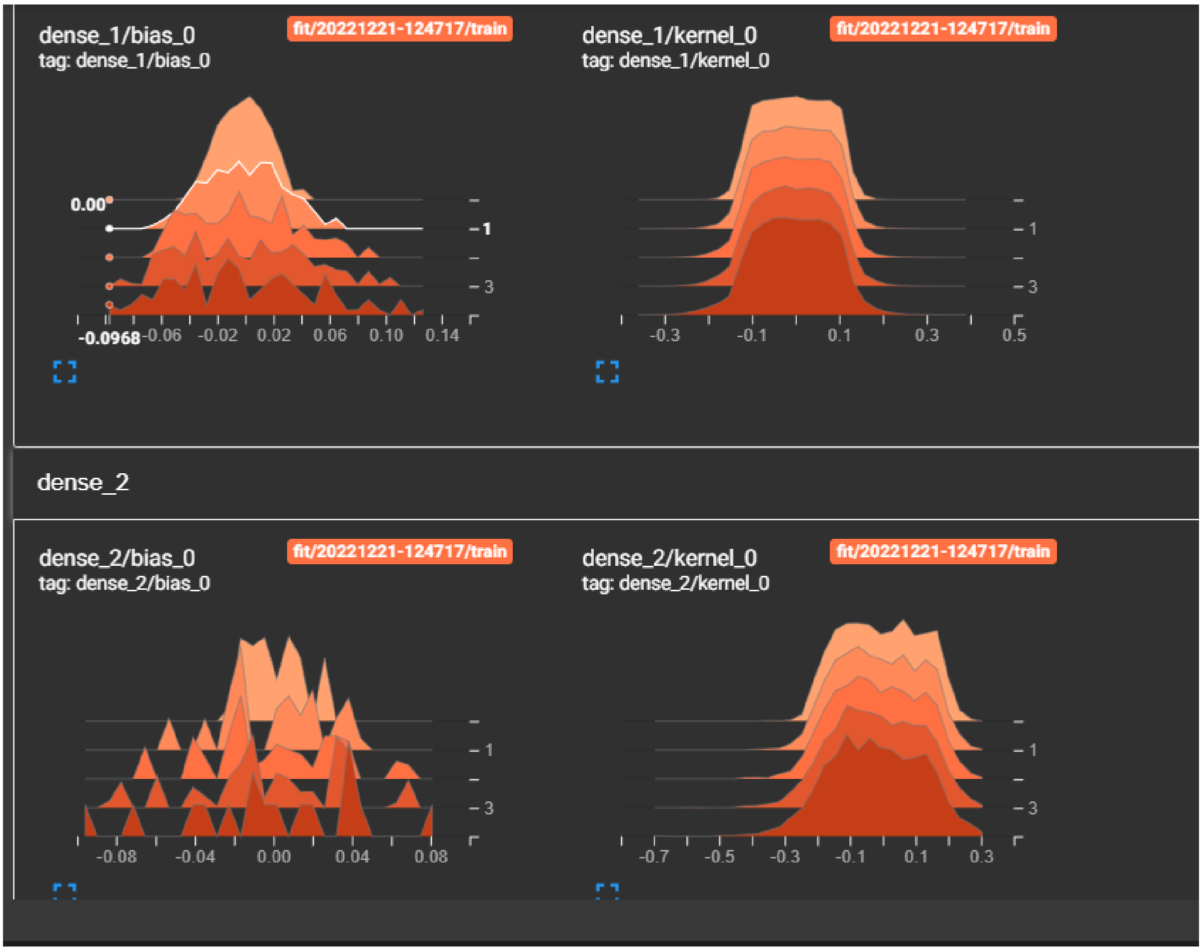
Smoothing 0.6

Horizontal Axis
[✓ STEP] RELATIVE WALL

Runs
Write a regex to filter runs
☑ ○ fit/20221221-124717/train
☑ ○ fit/20221221-124717/validation
TOGGLE ALL RUNS
logs

epoch_loss

epoch_loss
tag: epoch_loss

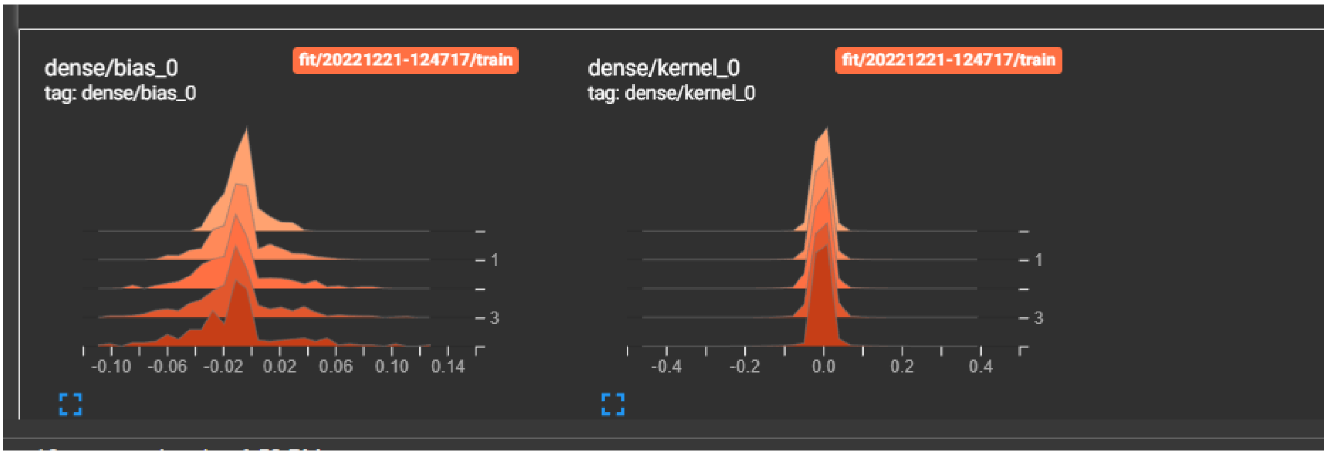

dense_1/bias_0
tag: dense_1/bias_0
fit/20221221-124717/train

dense_1/kernel_0
tag: dense_1/kernel_0
fit/20221221-124717/train

dense_2

dense_2/bias_0
tag: dense_2/bias_0
fit/20221221-124717/train

dense_2/kernel_0
tag: dense_2/kernel_0
fit/20221221-124717/train

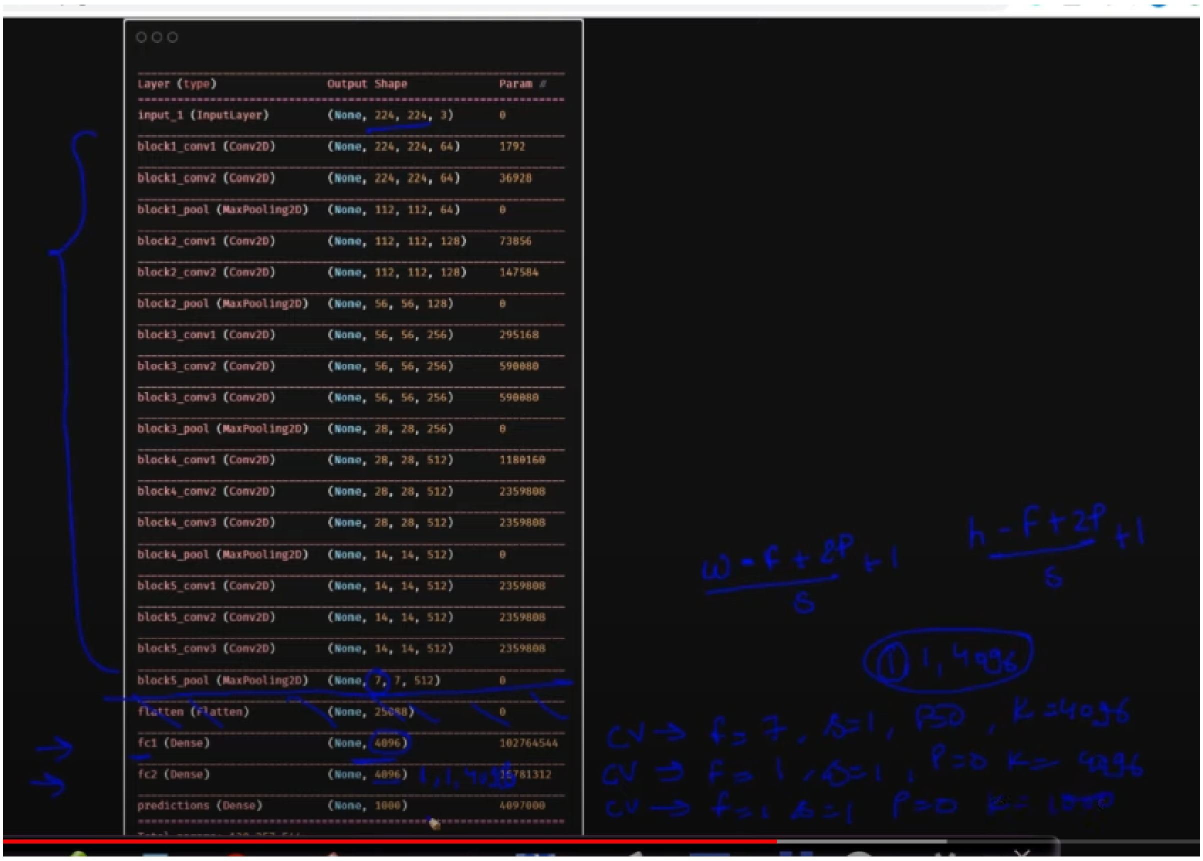


Observation

- we can observe that 1/4th(which is 4,492,176/ 19,206,864) of the parameters are trainable
- after 5 epoch we got the train accuracy of 82% and val_accuracy of 74.6%
- the weights in the last 3rd and 2nd dense layers are well distributed and the last layers wights have many values closer to zero since many of the features are zeroed out from the 128 dense neurons layer to 16 neurons layer for classification.

Model-2

1. Use VGG-16 (https://www.tensorflow.org/api_docs/python/tf/keras/applications/VGG16) pretrained network without Fully Connected layers and initilize all the weights with Imagenet trained weights.
2. After VGG-16 network without FC layers, don't use FC layers, use conv layers only as Fully connected layer.Any FC layer can be converted to a CONV layer. This conversion will reduce the No of Trainable parameters in FC layers.
For example, an FC layer with K=4096 that is looking at some input volume of size 7×7×512 can be equivalently expressed as a CONV layer with F=7,P=0,S=1,K=4096.
In other words, we are setting the filter size to be exactly the size of the input volume, and hence the output will simply be 1×1×4096 since only a single depth column “fits” across the input volume, giving identical result as the initial FC layer. You can refer this (http://cs231n.github.io/convolutional-networks/#convert) link to better understanding of using Conv layer in place of fully connected layers.
3. Final architecture will be VGG-16 without FC layers(without top), 2 Conv layers identical to FC layers, 1 output layer for 16 class classification. **INPUT --> VGG-16 without Top layers(FC) --> 2 Conv Layers identical to FC -->Output Layer**
4. 4.Print model.summary() and plot the architecture of the model.
Reference for plotting model (https://www.tensorflow.org/api_docs/python/tf/keras/utils/plot_model)
5. Train only last 2 Conv layers identical to FC layers, 1 output layer. Don't train the VGG-16 network.



```
Layer (type)                 Output Shape              Param #
==================================================================
input_1 (InputLayer)         (None, 224, 224, 3)       0
__________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
__________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
__________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
__________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
__________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
__________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
__________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
__________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
__________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
__________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
__________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
__________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
__________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
__________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
__________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
__________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
__________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
__________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
__________________________________________________________________
flatten (Flatten)            (None, 25088)             0
__________________________________________________________________
fc1 (Dense)                  (None, 4096)              102764544
__________________________________________________________________
fc2 (Dense)                  (None, 4096)              16781312
__________________________________________________________________
predictions (Dense)          (None, 1000)              4097000
==================================================================
```

In [ ]:

```
#model_2
for layer in model.layers:
  layer.trainable = False
#Adding custom Layers
x = model.output
x = Conv2D(filters=4096,kernel_size=8 ,strides=1,activation="relu")(x)
x = Conv2D(filters=4096,kernel_size=1 ,strides=1,activation="relu")(x)
x = Flatten()(x)
# creating the final model
output= Dense(16, activation="softmax")(x)
model_2 = Model(inputs = model.input, outputs = output)
# compile the model
model_2.compile(loss="categorical_crossentropy",optimizer = 'Adam',metrics=['accuracy'])
```

In [ ]:

```
# summary of the model_2
model_2.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_1 (InputLayer)        [(None, 256, 256, 3)]     0         
                                                                 
 block1_conv1 (Conv2D)       (None, 256, 256, 64)      1792      
                                                                 
 block1_conv2 (Conv2D)       (None, 256, 256, 64)      36928     
                                                                 
 block1_pool (MaxPooling2D)  (None, 128, 128, 64)      0         
                                                                 
 block2_conv1 (Conv2D)       (None, 128, 128, 128)     73856     
                                                                 
 block2_conv2 (Conv2D)       (None, 128, 128, 128)     147584    
                                                                 
 block2_pool (MaxPooling2D)  (None, 64, 64, 128)       0         
                                                                 
 block3_conv1 (Conv2D)       (None, 64, 64, 256)       295168    
                                                                 
 block3_conv2 (Conv2D)       (None, 64, 64, 256)       590080    
                                                                 
 block3_conv3 (Conv2D)       (None, 64, 64, 256)       590080    
                                                                 
 block3_pool (MaxPooling2D)  (None, 32, 32, 256)       0         
                                                                 
 block4_conv1 (Conv2D)       (None, 32, 32, 512)       1180160   
                                                                 
 block4_conv2 (Conv2D)       (None, 32, 32, 512)       2359808   
                                                                 
 block4_conv3 (Conv2D)       (None, 32, 32, 512)       2359808   
                                                                 
 block4_pool (MaxPooling2D)  (None, 16, 16, 512)       0         
                                                                 
 block5_conv1 (Conv2D)       (None, 16, 16, 512)       2359808   
                                                                 
 block5_conv2 (Conv2D)       (None, 16, 16, 512)       2359808   
                                                                 
 block5_conv3 (Conv2D)       (None, 16, 16, 512)       2359808   
                                                                 
 block5_pool (MaxPooling2D)  (None, 8, 8, 512)         0         
                                                                 
 conv2d (Conv2D)             (None, 1, 1, 4096)        134221824 
                                                                 
 conv2d_1 (Conv2D)           (None, 1, 1, 4096)        16781312  
                                                                 
 flatten (Flatten)           (None, 4096)              0         
                                                                 
 dense (Dense)               (None, 16)                65552     
                                                                 
=================================================================
Total params: 165,783,376
Trainable params: 151,068,688
Non-trainable params: 14,714,688
_________________________________________________________________
```

In [ ]:

```
#fitting model_2
model_2.fit_generator(train_generator,steps_per_epoch=train_steps, epochs=5,verbose=1,callbacks=[tensorboard_callback]
                              )
```

```
<ipython-input-20-8d58a9e47380>:2: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future
version. Please use `Model.fit`, which supports generators.
  model_2.fit_generator(train_generator,steps_per_epoch=train_steps, epochs=5,verbose=1,callbacks=[tensorboard_call
back]

Epoch 1/5
1200/1200 [==============================] - 472s 394ms/step - loss: 0.6773 - accuracy: 0.7921
Epoch 2/5
1200/1200 [==============================] - 472s 394ms/step - loss: 0.6032 - accuracy: 0.8122
Epoch 3/5
1200/1200 [==============================] - 472s 393ms/step - loss: 0.5321 - accuracy: 0.8344
Epoch 4/5
1200/1200 [==============================] - 474s 395ms/step - loss: 0.4828 - accuracy: 0.8484
Epoch 5/5
1200/1200 [==============================] - 473s 394ms/step - loss: 0.4370 - accuracy: 0.8649
```

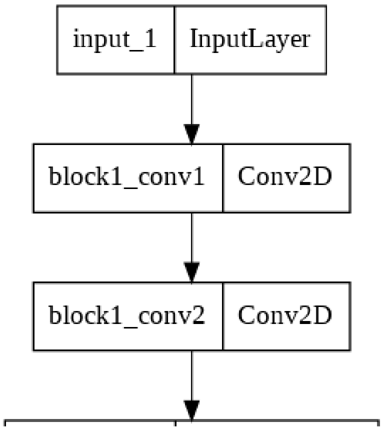
Out[20]:

```
<keras.callbacks.History at 0x7fae405bcdc0>
```

In [ ]:

```
# model graphs
tf.keras.utils.plot_model(
    model_2, to_file='model_2.png', show_shapes=False, show_layer_names=True,
    rankdir='TB', expand_nested=False, dpi=96
)
```

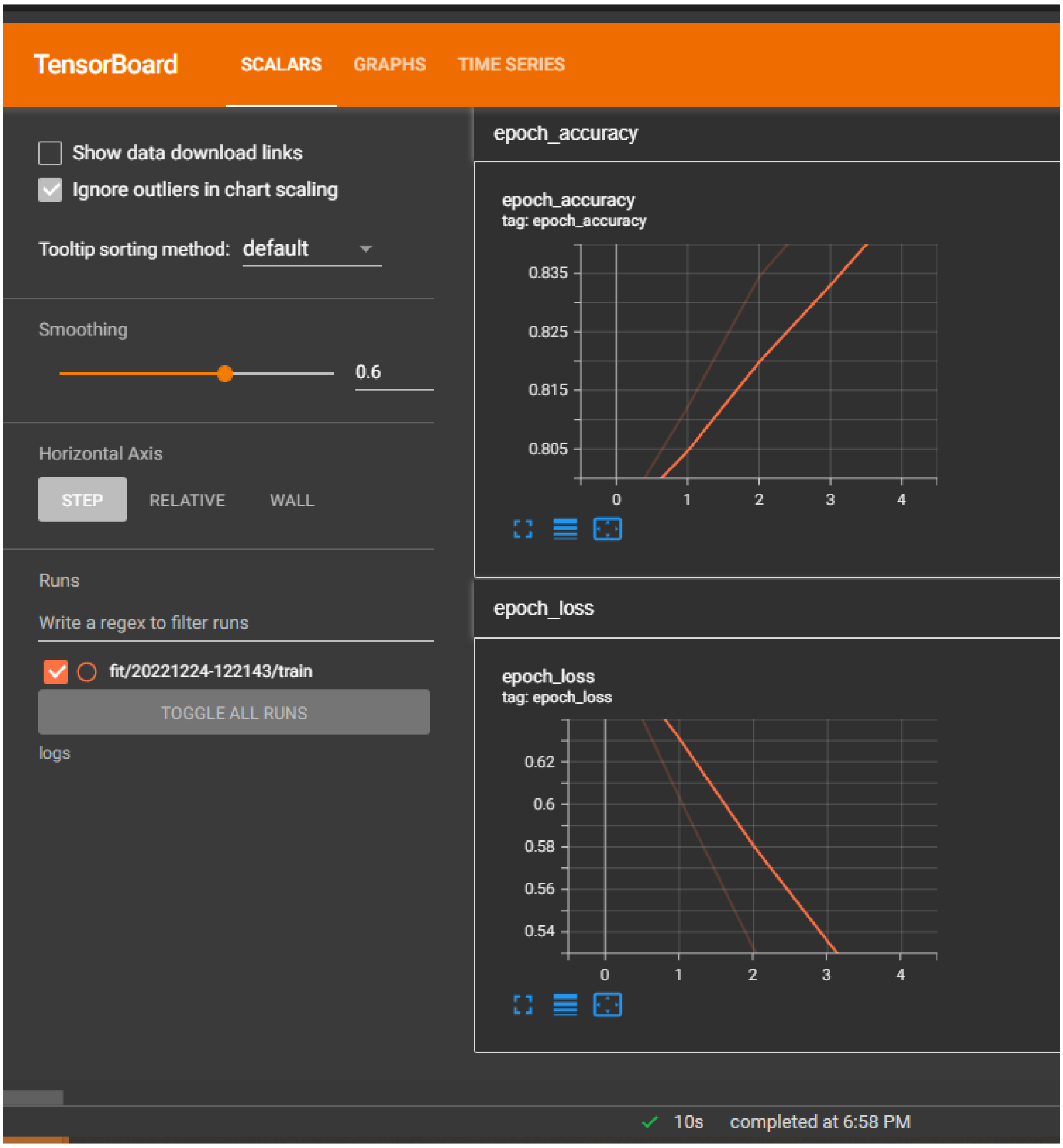
Out[21]:



In [ ]:

```
!rm -rf ./logs/
```

In [ ]:



□ Show data download links

☑ Ignore outliers in chart scaling

Tooltip sorting method: default

Smoothing 0.6

Horizontal Axis: [✓ STEP] RELATIVE WALL

Runs

Write a regex to filter runs

☑ ○ fit/20221224-122143/train

TOGGLE ALL RUNS

logs

epoch_accuracy

epoch_accuracy
tag: epoch_accuracy

epoch_loss

epoch_loss
tag: epoch_loss

✓ 10s completed at 6:58 PM

Observation

- we can observe that 911/1000th (which is 151,068,688/ 165,783,376) of the parameters are trainable
- after 5 epoch we got the train accuracy of 86.49%

Model-3

```
1. Use same network as Model-2 'INPUT --> VGG-16 without Top layers(FC) --> 2 Conv Layers identical to FC --> Output Layer' and train only Last 6 Layers of VGG-16 network, 2 Conv layers identical to FC layers, 1 output layer.
```

In [ ]:

```python
#model_1
for layer in model.layers:
  layer.trainable = False
for layer in model.layers[-6:]: # training last 6 layers of vgg16
    layer.trainable = True
    print("Layer '%s' is trainable" % layer.name)
```

```
Layer 'block4_conv3' is trainable
Layer 'block4_pool' is trainable
Layer 'block5_conv1' is trainable
Layer 'block5_conv2' is trainable
Layer 'block5_conv3' is trainable
Layer 'block5_pool' is trainable
```

In [ ]:

```python
#model_3
#Adding custom Layers
x = model.output
x = Conv2D(filters=4096,kernel_size=8 ,strides=1,activation="relu")(x)
x = Conv2D(filters=4096,kernel_size=1 ,strides=1,activation="relu")(x)
x = Flatten()(x)
# creating the final model
output = Dense(16, activation="softmax")(x)
model_3 = Model(inputs = model.input, outputs = output)
# compile the model
model_3.compile(loss="categorical_crossentropy",optimizer = 'Adam',metrics=['accuracy'])
```

In [ ]:

```
model_3.summary()
```

```
Model: "model_2"

_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_2 (InputLayer)        [(None, 256, 256, 3)]     0         
                                                                 
 block1_conv1 (Conv2D)       (None, 256, 256, 64)      1792      
                                                                 
 block1_conv2 (Conv2D)       (None, 256, 256, 64)      36928     
                                                                 
 block1_pool (MaxPooling2D)  (None, 128, 128, 64)      0         
                                                                 
 block2_conv1 (Conv2D)       (None, 128, 128, 128)     73856     
                                                                 
 block2_conv2 (Conv2D)       (None, 128, 128, 128)     147584    
                                                                 
 block2_pool (MaxPooling2D)  (None, 64, 64, 128)       0         
                                                                 
 block3_conv1 (Conv2D)       (None, 64, 64, 256)       295168    
                                                                 
 block3_conv2 (Conv2D)       (None, 64, 64, 256)       590080    
                                                                 
 block3_conv3 (Conv2D)       (None, 64, 64, 256)       590080    
                                                                 
 block3_pool (MaxPooling2D)  (None, 32, 32, 256)       0         
                                                                 
 block4_conv1 (Conv2D)       (None, 32, 32, 512)       1180160   
                                                                 
 block4_conv2 (Conv2D)       (None, 32, 32, 512)       2359808   
                                                                 
 block4_conv3 (Conv2D)       (None, 32, 32, 512)       2359808   
                                                                 
 block4_pool (MaxPooling2D)  (None, 16, 16, 512)       0         
                                                                 
 block5_conv1 (Conv2D)       (None, 16, 16, 512)       2359808   
                                                                 
 block5_conv2 (Conv2D)       (None, 16, 16, 512)       2359808   
                                                                 
 block5_conv3 (Conv2D)       (None, 16, 16, 512)       2359808   
                                                                 
 block5_pool (MaxPooling2D)  (None, 8, 8, 512)         0         
                                                                 
 conv2d_4 (Conv2D)           (None, 1, 1, 4096)        134221824 
                                                                 
 conv2d_5 (Conv2D)           (None, 1, 1, 4096)        16781312  
                                                                 
 flatten_2 (Flatten)         (None, 4096)              0         
                                                                 
 dense_2 (Dense)             (None, 16)                65552     
                                                                 
=================================================================
Total params: 165,783,376
Trainable params: 160,507,920
Non-trainable params: 5,275,456
_________________________________________________________________
```

In [ ]:

```
#fitting model_3
model_3.fit_generator(train_generator,steps_per_epoch=train_steps, epochs=2,
                             callbacks=[tensorboard_callback])
```

```
Epoch 1/2

<ipython-input-27-0939d2b11be3>:2: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future
version. Please use `Model.fit`, which supports generators.
  model_3.fit_generator(train_generator,steps_per_epoch=train_steps, epochs=2,

1200/1200 [==============================] - 788s 657ms/step - loss: 2.7729 - accuracy: 0.0631
Epoch 2/2
1200/1200 [==============================] - 787s 655ms/step - loss: 2.7728 - accuracy: 0.0616
```

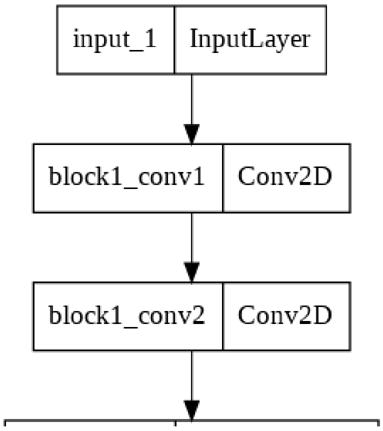
Out[27]:

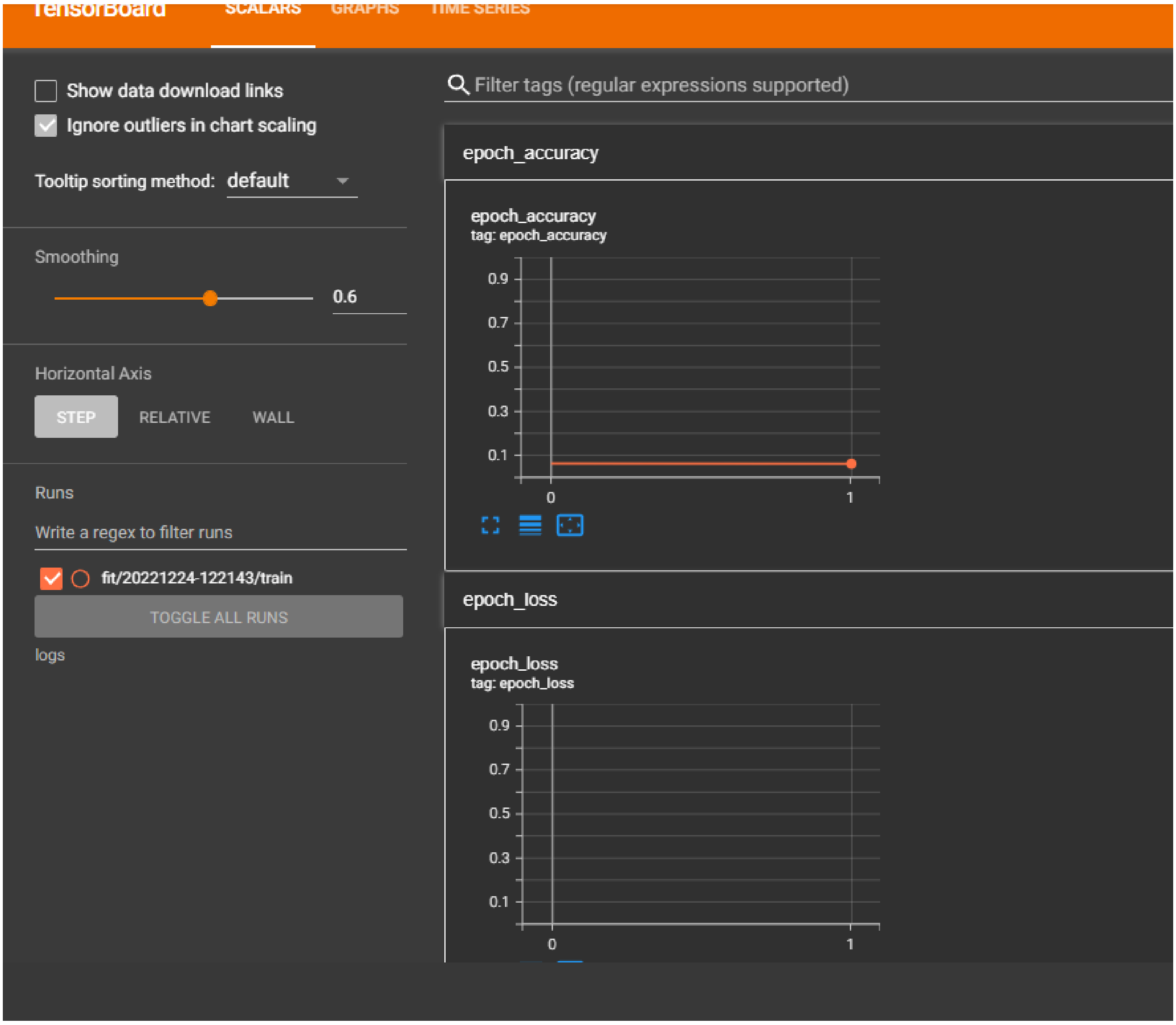
```
<keras.callbacks.History at 0x7fae402a4e80>
```

In [ ]:

```
# model graphs
tf.keras.utils.plot_model(
    model_3, to_file='model_3.png', show_shapes=False, show_layer_names=True,
    rankdir='TB', expand_nested=False, dpi=96
)
```

Out[28]:





□ Show data download links

☑ Ignore outliers in chart scaling

Tooltip sorting method: default

Smoothing 0.6

Horizontal Axis: STEP RELATIVE WALL

Runs

Write a regex to filter runs

☑ ○ fit/20221224-122143/train

TOGGLE ALL RUNS

logs

Filter tags (regular expressions supported)

epoch_accuracy

epoch_accuracy
tag: epoch_accuracy

epoch_loss

epoch_loss
tag: epoch_loss

Observation

- we can observe that 968/1000th(which is 160,507,920/ 165,783,376) of the parameters are trainable
- after 5 epoch we got the train accuracy of 6.16%.

Summary

In [6]:

```
# Please compare all your models using Prettytable library
#http://zetcode.com/python/prettytable/
from prettytable import PrettyTable

0
x = PrettyTable()
x.field_names = ["Features", "Model", "Epochs", "Train accuracy", "CV accuracy"]
x.add_row(["Assignment 1 VGG16 w/o FC + 1 Conv + 1 MP + 2 FC + output", "CNN+Dense+softmax", "5", "82.02%","74.64% "])
x.add_row(["Assignment 2 VGG16 w/o FC + 2 Conv + output", "CNN+softmax","5", "86.49%", "-" ])
x.add_row(["Assignment 3 VGG16 w/o FC + 2 Conv + output", "CNN+softmax","2", "6.16%", "- " ])
print(x)
```

```
+-----------------------------------------------------------+-------------------+--------+----------------+--------
-----+
|                          Features                         |       Model       | Epochs | Train accuracy | CV accu
racy |
+-----------------------------------------------------------+-------------------+--------+----------------+--------
-----+
| Assignment 1 VGG16 w/o FC + 1 Conv + 1 MP + 2 FC + output | CNN+Dense+softmax |   5    |     82.02%     |   74.6
4%    |
|        Assignment 2 VGG16 w/o FC + 2 Conv + output        |    CNN+softmax    |   5    |     86.49%     |      -
|
|        Assignment 3 VGG16 w/o FC + 2 Conv + output        |    CNN+softmax    |   2    |     6.16%      |      -
|
+-----------------------------------------------------------+-------------------+--------+----------------+--------
-----+
```