1. Download all the data in this folder https://drive.google.com/open?id=1Z4TyI7FcFVEx8qdl4jO9qxvxaqLSqoEu. it contains two file both images and labels. The label file list the images and their categories 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

2. On this image data, you have to train 3 types of models as given below. You have to split the data into Train and Validation data.

3. Try not to load all the images into memory, use the gernarators that we have given the reference notebooks to load the batch of images only during the train data.
or you can use this method also
https://medium.com/@vijayabhaskar96/tutorial-on-keras-imagedatagenerator-with-flow-from-dataframe-8bd5776e45c1

https://medium.com/@vijayabhaskar96/tutorial-on-keras-flow-from-dataframe-1fd4493d237c

4. You are free to choose Learning rate, optimizer, loss function, image augmentation, any hyperparameters. but you have to use the same architechture what we are asking below.

5. Use tensorboard for every model and analyse your gradients. (you need to upload the screenshots for each model for evaluation)

Note: fit_genarator() method will have problems with the tensorboard histograms, try to debug it, if you could not do use histgrams=0 i.e don't include histograms, check the documentation of tensorboard for more information.

6. You can check about Transfer Learning in this link - https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html

Model-1

1. Use VGG-16 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 a output layer to classify 16 classes. You are free to choose any hyperparameters/parameters of conv block, FC layers, output layer.
3. Final architecture will be **INPUT --> VGG-16 without Top layers(FC) --> Conv Layer --> Maxpool Layer --> 2 FC layers --> Output Layer**
4. Train only new Conv block, FC layers, output layer. Don't train the VGG-16 network.

Model-2

1. Use VGG-16 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 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**
3. Train only last 2 Conv layers identical to FC layers, 1 output layer. Don't train the VGG-16 network.

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.

Solution

In [1]:

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

```
Downloading...
From: https://drive.google.com/uc?id=1Z4TyI7FcFVEx8qdl4jO9qxvxaqLSqoEu
To: /content/rvl-cdip.rar
4.66GB [01:23, 55.5MB/s]
```

In [2]:

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

In [3]:

```
%load_ext tensorboard
```

In [4]:

```
import tensorflow as tf
import os
import numpy as np
import pandas as pd
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.layers import Dense, Conv2D, Input, MaxPooling2D, GlobalAveragePooling2D, Flatten,
from tensorflow.keras.callbacks import Callback, TensorBoard
import datetime
```

In [5]:

```
path_y_labels = pd.read_csv('labels_final.csv')
print(path_y_labels.shape)
path_y_labels.head()
```

```
(48000, 2)
```

Out[5]:

	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 [6]:

```
path_y_labels['label'] = path_y_labels['label'].astype(str)
```

In [7]:

```
# Create ImageDataGenerator for train and validation

datagen = ImageDataGenerator(rescale=1./255, validation_split=0.2)
```

```
train_datagen = datagen.flow_from_dataframe(
        dataframe=path_y_labels,
        directory="./data_final/",
        x_col="path",
        y_col="label",
        subset="training",
        batch_size=32,
        seed=42,
        shuffle=True,
        class_mode="categorical",
        target_size=(224,224))

val_datagen = datagen.flow_from_dataframe(
        dataframe=path_y_labels,
        directory="./data_final/",
        x_col="path",
        y_col="label",
        subset="validation",
        batch_size=32,
        seed=42,
        shuffle=True,
        class_mode="categorical",
        target_size=(224,224))
```

```
Found 38400 validated image filenames belonging to 16 classes.
Found 9600 validated image filenames belonging to 16 classes.
```

Model 1

(Input-> VGG16-> Conv1-> Pool1-> Flatten-> FC_Dense1-> FC_Dense2-> Output) = 60% accuracy needed

In [20]:

```
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.layers import Dense, Conv2D, Input, MaxPooling2D, GlobalAveragePooling2D, Flatten,

# Use VGG16
vgg_model = VGG16(weights='imagenet', include_top=False)
# Dont train weights in VGG16
for layer in vgg_model.layers:
    layer.trainable = False

#Input layer
input_x = Input(shape = (224, 224, 3), name='input_layer')
# VGG
vgg_output = vgg_model(input_x)
#Conv Layer
Conv1 = Conv2D(filters=32,kernel_size=(3,3),strides=(1,1),padding='valid', activation='relu', data_format:
               kernel_initializer=tf.keras.initializers.he_normal(seed=0),name='Conv1')(vgg_output)
#MaxPool Layer
pool1 = MaxPooling2D(pool_size=(2,2),strides=(2,2),padding='valid',name='Pool1')(Conv1)
#Flatten
flatten1 = Flatten()(pool1)
#FC layer
fc11 = Dense(units=512,activation='relu',kernel_initializer=tf.keras.initializers.he_normal(seed=9),name=
#FC layer
fc12 = Dense(units=256,activation='relu',kernel_initializer=tf.keras.initializers.he_normal(seed=9),name=
#Output layer
out1 = Dense(units=16,activation='softmax',kernel_initializer=tf.keras.initializers.glorot_normal(seed=3)
#Creating a model
model1 = Model(inputs=input_x,outputs=out1)
model1.summary()
```

```
Model: "model_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_layer (InputLayer)     [(None, 224, 224, 3)]     0         
_________________________________________________________________
vgg16 (Functional)           (None, None, None, 512)   14714688  
_________________________________________________________________
Conv1 (Conv2D)               (None, 5, 5, 32)          147488    
_________________________________________________________________
Pool1 (MaxPooling2D)         (None, 2, 2, 32)          0         
_________________________________________________________________
flatten_2 (Flatten)          (None, 128)               0         
_________________________________________________________________
FC1 (Dense)                  (None, 512)               66048     
_________________________________________________________________
FC2 (Dense)                  (None, 256)               131328    
_________________________________________________________________
Output (Dense)               (None, 16)                4112      
=================================================================
Total params: 15,063,664
Trainable params: 348,976
Non-trainable params: 14,714,688
_________________________________________________________________
```

In [21]:

```
#compiling
model1.compile(optimizer=tf.keras.optimizers.Adam(lr=0.001),loss='categorical_crossentropy',metrics=['acc
```

In [24]:

```
class Callback_Custom(Callback):
    def on_train_begin(self, logs={}):
        ## on begin of training, we are creating a instance varible called history
        ## it is a dict with keys [loss, acc, val_loss, val_acc]
        self.history={'acc': [], 'val_acc': []}

    def on_epoch_end(self, epoch, logs={}):
        self.history['acc'].append(logs.get('accuracy'))
        if logs.get('val_accuracy', -1) != -1:
            self.history['val_acc'].append(logs.get('val_accuracy'))

        loss = logs.get('loss')
        if logs.get('val_loss', -1) != -1:
            val_loss = logs.get('val_loss')
            if val_loss - loss > 0.25:
               print("Val-loss is much more than train loss and terminated at epoch {}".format(epoch+1))
               self.model.stop_training = True

        # Stop the training if your validation accuracy is not increased in last 3 epochs.
        if len(self.history['val_acc']) >= 4:
            if ((self.history['val_acc'][-1] < self.history['val_acc'][-4]) and
               (self.history['val_acc'][-1] < self.history['val_acc'][-3]) and
               (self.history['val_acc'][-1] < self.history['val_acc'][-2])):
                    print("Validation accuracy not improving and terminated at epoch {}".format(epoch+1))
                    self.model.stop_training = True
```

In [25]:

```
##fitting generator
import datetime
logdir1 = os.path.join("logs_model1", datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
tensorboard_callback1 = TensorBoard(log_dir=logdir1,histogram_freq=1, write_graph=True)
callback1 = Callback_Custom()

model1.fit(
        train_datagen,
        epochs=10,
        validation_data=val_datagen,
        callbacks=[tensorboard_callback1, callback1])
```

```
Epoch 1/10
1200/1200 [==============================] - 237s 197ms/step - loss: 1.6796 - accuracy: 0.4753 -
val_loss: 1.1953 - val_accuracy: 0.6404
Epoch 2/10
1200/1200 [==============================] - 244s 203ms/step - loss: 1.0897 - accuracy: 0.6641 -
val_loss: 1.0671 - val_accuracy: 0.6762
Epoch 3/10
1200/1200 [==============================] - 243s 202ms/step - loss: 0.9499 - accuracy: 0.7039 -
val_loss: 1.0659 - val_accuracy: 0.6848
Epoch 4/10
1200/1200 [==============================] - 244s 203ms/step - loss: 0.8493 - accuracy: 0.7352 -
val_loss: 1.0580 - val_accuracy: 0.6914
Epoch 5/10
1200/1200 [==============================] - 238s 198ms/step - loss: 0.7692 - accuracy: 0.7594 -
val_loss: 1.0473 - val_accuracy: 0.6980
Val-loss is much more than train loss and terminated at epoch 5
```

Out[25]:

```
<tensorflow.python.keras.callbacks.History at 0x7fa5b86ee5c0>
```

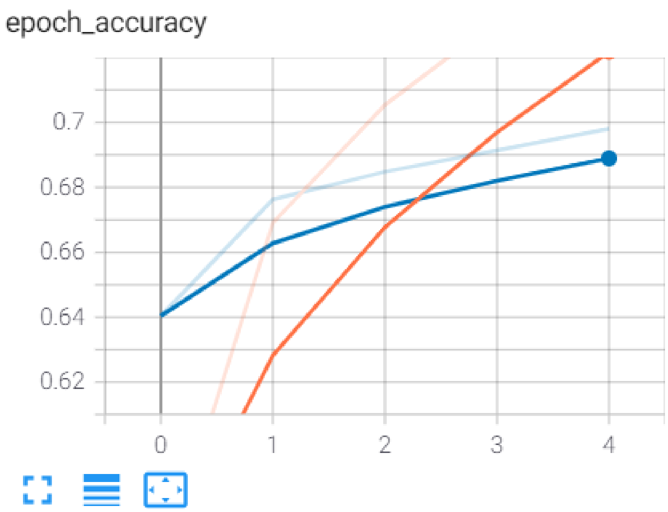
In [26]:

```
%tensorboard --logdir logs_model1
```

```
Output hidden; open in https://colab.research.google.com to view.
```

Conv and dense layers are used with VGG16 with non-trainable VGG16. 32 conv filters and 512,256 units of dense layers are used to get accuracy more than 60%. Training got terminated as val_loss was more than train_loss by 25% (just a threshold).

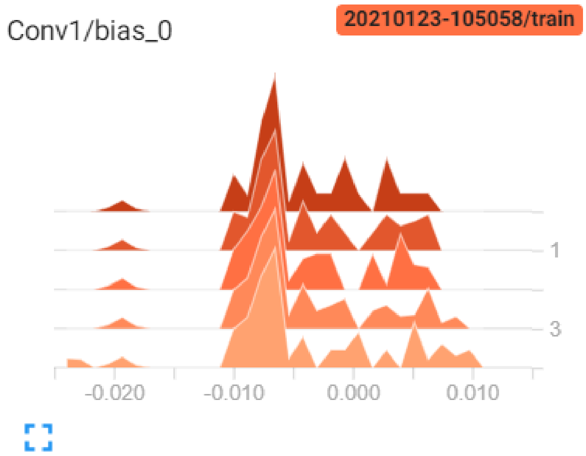
Accuracy Vs Epoch and Loss Vs Epoch

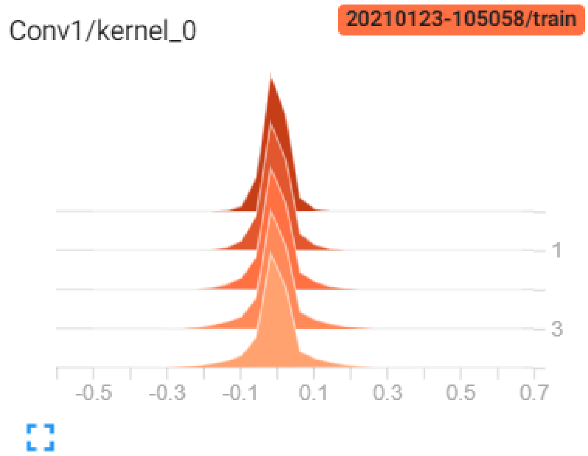


epoch_loss

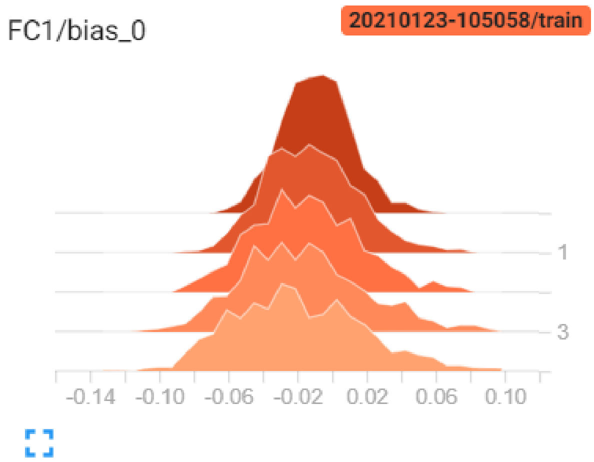
Histograms of Conv, two FC-dense and output layers

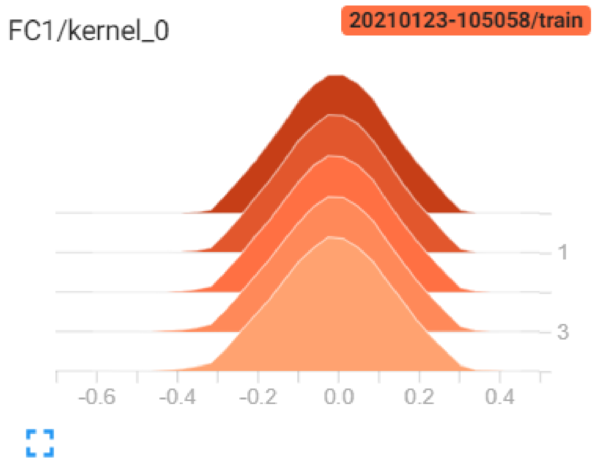
Conv1



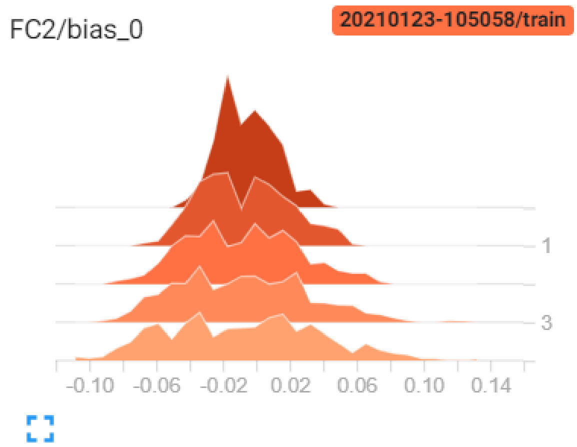


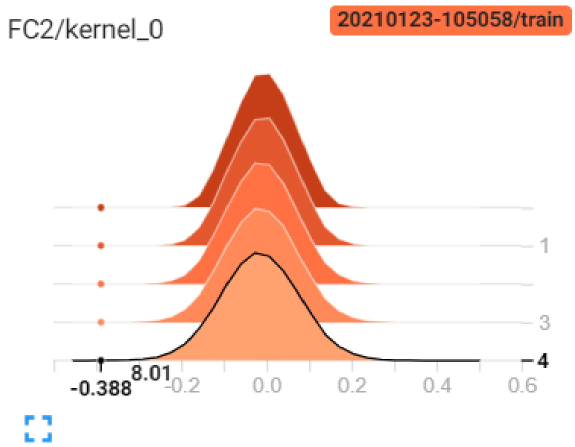
FC1



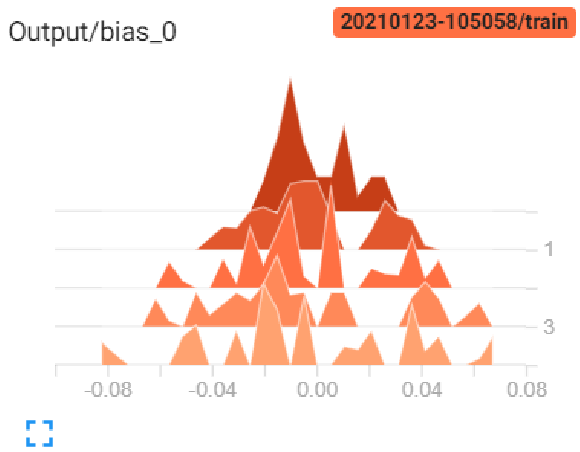


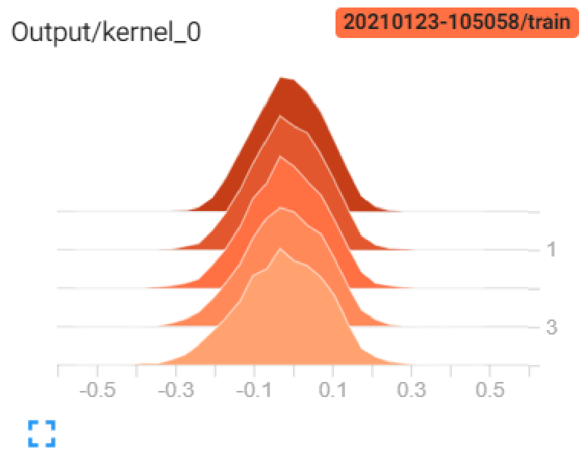
FC2





Output





We got the updated optimal weights which are intializsed as normal distributions (he, glorot)

Model 2

In [27]:

```
# Use VGG16
vgg_model = VGG16(weights='imagenet', include_top=False)
# Dont train weights in VGG16
for layer in vgg_model.layers:
    layer.trainable = False

#Input layer
input_x = Input(shape = (224, 224, 3), name='input_layer')

# VGG
vgg_output = vgg_model(input_x)

filter_size = (int)(224 / (pow(2,5)))
# FC with Conv
#Conv Layer1
Conv1 = Conv2D(filters=128,kernel_size=(filter_size, filter_size),strides=(1,1),padding='valid', activati
               kernel_initializer=tf.keras.initializers.he_normal(seed=0),name='Conv1')(vgg_output)
#Conv Layer2
Conv2 = Conv2D(filters=64,kernel_size=(1, 1),strides=(1,1),padding='valid', activation='relu', data_forma
               kernel_initializer=tf.keras.initializers.he_normal(seed=0),name='Conv2')(Conv1)
#Flatten
flatten1 = Flatten()(Conv2)
#Output layer
out = Dense(units=16,activation='softmax',kernel_initializer=tf.keras.initializers.glorot_normal(seed=3),
#Creating a model
model2 = Model(inputs=input_x,outputs=out)
model2.summary()
```

```
Model: "model_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_layer (InputLayer)     [(None, 224, 224, 3)]     0
_________________________________________________________________
vgg16 (Functional)           (None, None, None, 512)   14714688
_________________________________________________________________
Conv1 (Conv2D)               (None, 1, 1, 128)         3211392
_________________________________________________________________
Conv2 (Conv2D)               (None, 1, 1, 64)          8256
_________________________________________________________________
flatten_3 (Flatten)          (None, 64)                0
_________________________________________________________________
Output (Dense)               (None, 16)                1040
=================================================================
Total params: 17,935,376
Trainable params: 3,220,688
Non-trainable params: 14,714,688
_________________________________________________________________
```

In [28]:

```
#compiling
model2.compile(optimizer=tf.keras.optimizers.Adam(lr=0.001),loss='categorical_crossentropy',metrics=['acc
```

In [29]:

```
##fitting generator
import datetime
logdir2 = os.path.join("logs_model2", datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
tensorboard_callback2 = TensorBoard(log_dir=logdir2,histogram_freq=1, write_graph=True)
callback2 = Callback_Custom()
model2.fit(
        train_datagen,
        epochs=10,
        validation_data=val_datagen,
        callbacks=[tensorboard_callback2, callback2])
```

```
Epoch 1/10
1200/1200 [==============================] - 257s 213ms/step - loss: 1.6120 - accuracy: 0.5172 -
val_loss: 1.0383 - val_accuracy: 0.6949
Epoch 2/10
1200/1200 [==============================] - 252s 210ms/step - loss: 0.9530 - accuracy: 0.7083 -
val_loss: 1.0402 - val_accuracy: 0.6933
Epoch 3/10
1200/1200 [==============================] - 254s 212ms/step - loss: 0.7854 - accuracy: 0.7571 -
val_loss: 0.9571 - val_accuracy: 0.7210
Epoch 4/10
1200/1200 [==============================] - 250s 208ms/step - loss: 0.6720 - accuracy: 0.7893 -
val_loss: 0.9181 - val_accuracy: 0.7357
Epoch 5/10
1200/1200 [==============================] - 249s 208ms/step - loss: 0.5793 - accuracy: 0.8178 -
val_loss: 1.0274 - val_accuracy: 0.7095
Val-loss is much more than train loss and terminated at epoch 5
```

Out[29]:

```
<tensorflow.python.keras.callbacks.History at 0x7fa5b85ff1d0>
```

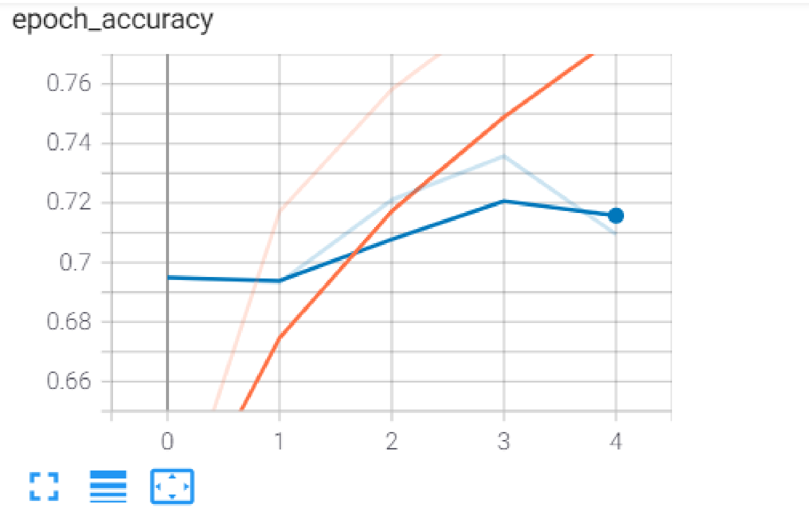
In [30]:

```
%tensorboard --logdir logs_model2
```

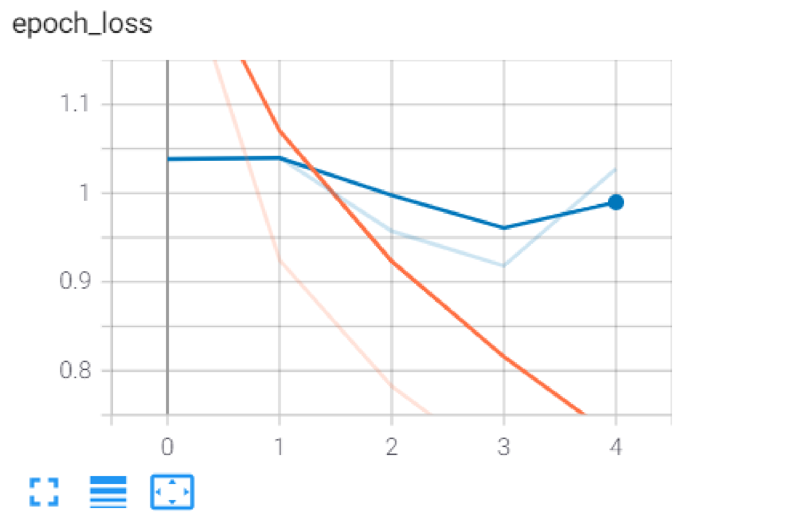
```
Output hidden; open in https://colab.research.google.com to view.
```

Two Conv layers are used in place of dense layers alng with VGG16 with non-trainable VGG16. 128 and 64 conv filters are used to get accuracy more than 60%. Training got terminated as val_loss was more than train_loss by 25% (just a threshold).

Accuracy vs Epoch and Loss Vs Epoch

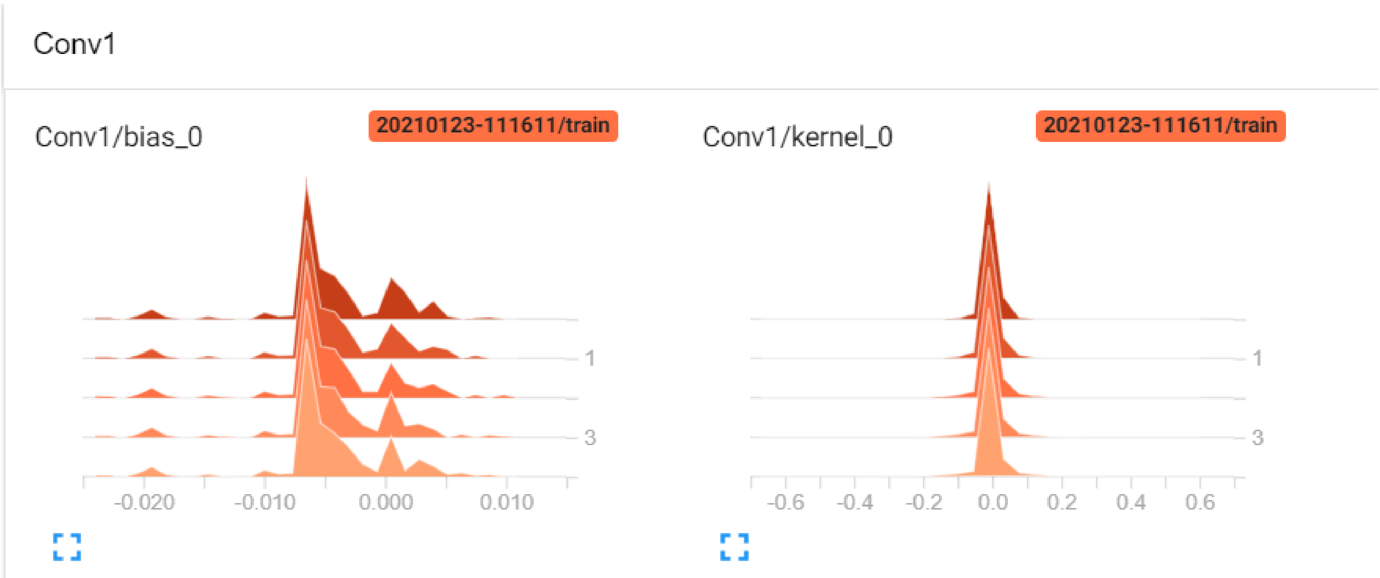


epoch_loss

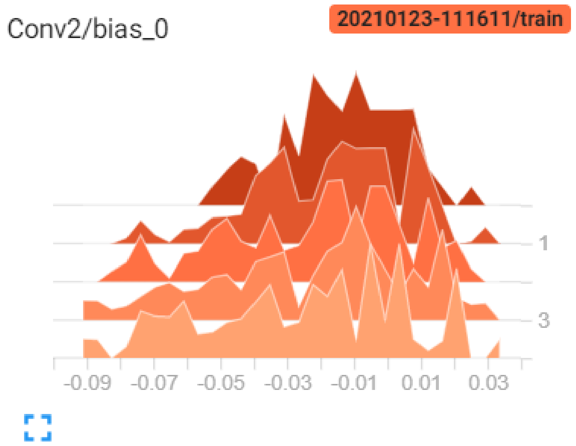


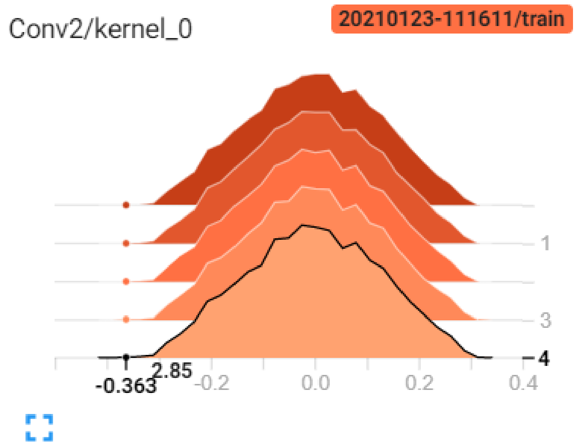
We got minimum loss and higher acucracy in epoch 4 after that the difference got increased between train and validation.

Histogram of two conv layers and output layer



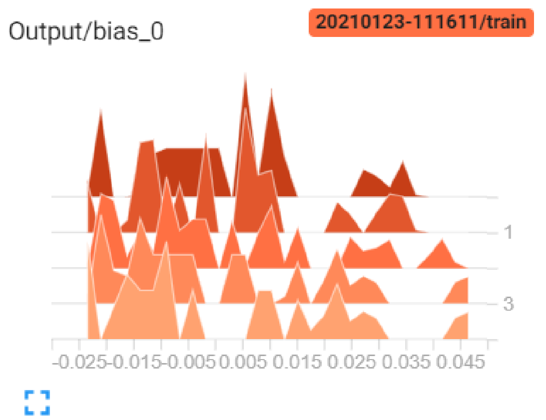
Conv2

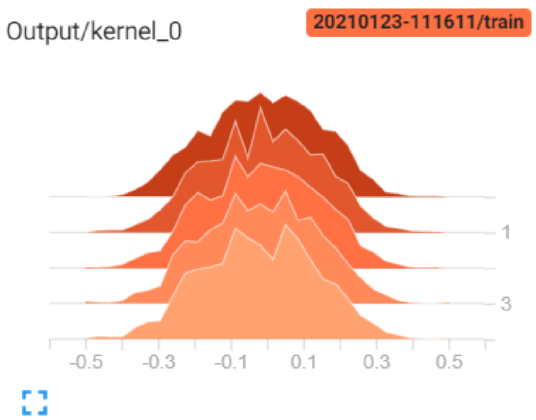




Output

2





We got the updated optimal weights which are intializsed as normal distributions (he, glorot)

Model 3

In [31]:

```
# Use VGG16
vgg_model = VGG16(weights='imagenet', include_top=False)
# Dont train weights in VGG16
for layer in vgg_model.layers[:-6]:
    layer.trainable = False

#Input layer
input_x = Input(shape = (224, 224, 3), name='input_layer')
# VGG
vgg_output = vgg_model(input_x)
filter_size = (int)(224 / (pow(2,5)))
# FC with Conv
#Conv Layer1
Conv1 = Conv2D(filters=128,kernel_size=(filter_size, filter_size),strides=(1,1),padding='valid', activati
               kernel_initializer=tf.keras.initializers.he_normal(seed=0),name='Conv1')(vgg_output)
#Conv Layer2
Conv2 = Conv2D(filters=64,kernel_size=(1, 1),strides=(1,1),padding='valid', activation='relu', data_forma
               kernel_initializer=tf.keras.initializers.he_normal(seed=0),name='Conv2')(Conv1)
#Flatten
flatten1 = Flatten()(Conv2)
#Output layer
out = Dense(units=16,activation='softmax',kernel_initializer=tf.keras.initializers.glorot_normal(seed=3),
#Creating a model
model3 = Model(inputs=input_x,outputs=out)

model3.summary()
```

```
Model: "model_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_layer (InputLayer)     [(None, 224, 224, 3)]     0         
_________________________________________________________________
vgg16 (Functional)           (None, None, None, 512)   14714688  
_________________________________________________________________
Conv1 (Conv2D)               (None, 1, 1, 128)         3211392   
_________________________________________________________________
Conv2 (Conv2D)               (None, 1, 1, 64)          8256      
_________________________________________________________________
flatten_4 (Flatten)          (None, 64)                0         
_________________________________________________________________
Output (Dense)               (None, 16)                1040      
=================================================================
Total params: 17,935,376
Trainable params: 12,659,920
Non-trainable params: 5,275,456
_________________________________________________________________
```

In [32]:

```
#compiling
model3.compile(optimizer=tf.keras.optimizers.Adam(lr=0.001),loss='categorical_crossentropy',metrics=['acc
```

In [33]:

```
##fitting generator
import datetime
logdir3 = os.path.join("logs_model3", datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
tensorboard_callback3 = TensorBoard(log_dir=logdir3,histogram_freq=1, write_graph=True)
callback3 = Callback_Custom()

model3.fit(
        train_datagen,
        epochs=10,
        validation_data=val_datagen,
        callbacks=[tensorboard_callback3, callback3])
```

```
Epoch 1/10
1200/1200 [==============================] - 275s 228ms/step - loss: 2.8477 - accuracy: 0.0611 -
val_loss: 2.7729 - val_accuracy: 0.0614
Epoch 2/10
1200/1200 [==============================] - 269s 224ms/step - loss: 2.7728 - accuracy: 0.0575 -
val_loss: 2.7729 - val_accuracy: 0.0584
Epoch 3/10
1200/1200 [==============================] - 267s 222ms/step - loss: 2.7728 - accuracy: 0.0606 -
val_loss: 2.7730 - val_accuracy: 0.0584
Epoch 4/10
1200/1200 [==============================] - 266s 222ms/step - loss: 2.7727 - accuracy: 0.0619 -
val_loss: 2.7730 - val_accuracy: 0.0601
Epoch 5/10
1200/1200 [==============================] - 268s 223ms/step - loss: 2.7727 - accuracy: 0.0624 -
val_loss: 2.7729 - val_accuracy: 0.0613
Epoch 6/10
1200/1200 [==============================] - 267s 222ms/step - loss: 2.7728 - accuracy: 0.0616 -
val_loss: 2.7729 - val_accuracy: 0.0616
Epoch 7/10
1200/1200 [==============================] - 265s 221ms/step - loss: 2.7727 - accuracy: 0.0630 -
val_loss: 2.7729 - val_accuracy: 0.0584
Validation accuracy not improving and terminated at epoch 7
```

Out[33]:

```
<tensorflow.python.keras.callbacks.History at 0x7fa5b867b550>
```

In [34]:

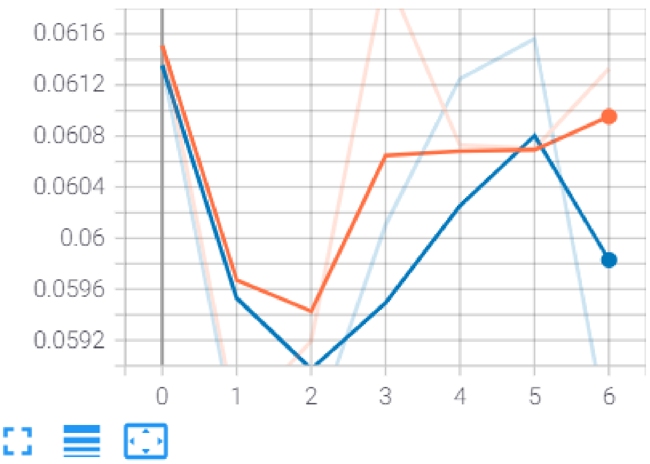
```
%tensorboard --logdir logs_model3
```

```
Output hidden; open in https://colab.research.google.com to view.
```

Two Conv layers are used in place of dense layers alng with VGG16 with trainable VGG16 by last 6 layers. 128 and 64 conv filters are used to get accuracy more than 6%. Training got terminated as val_loss was more than train_loss by 25% (just a threshold). Also we got less accuracy in 7th epoch. So we can use the weights obtained till 6th epoch.

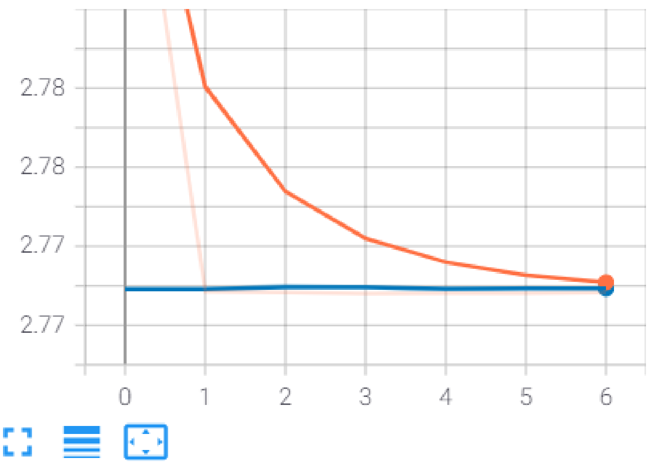
Accuracy Vs Epoch and Loss Vs Epoch

epoch_accuracy



epoch_loss

epoch_loss

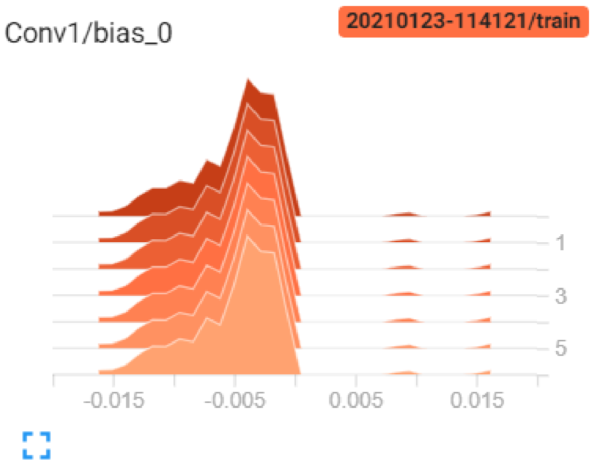


Validation loss was almost constant through out each epoch, but accuracy got changed and we got best accuracy in 1st and 6th epoch. We can choose weights of 6th epoch.

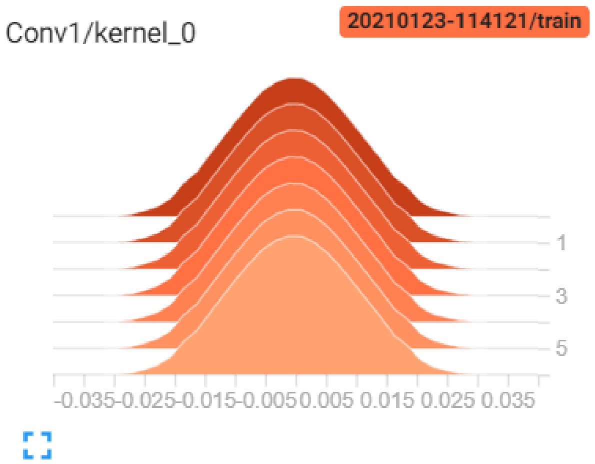
Histogram for two conv layers and output layer

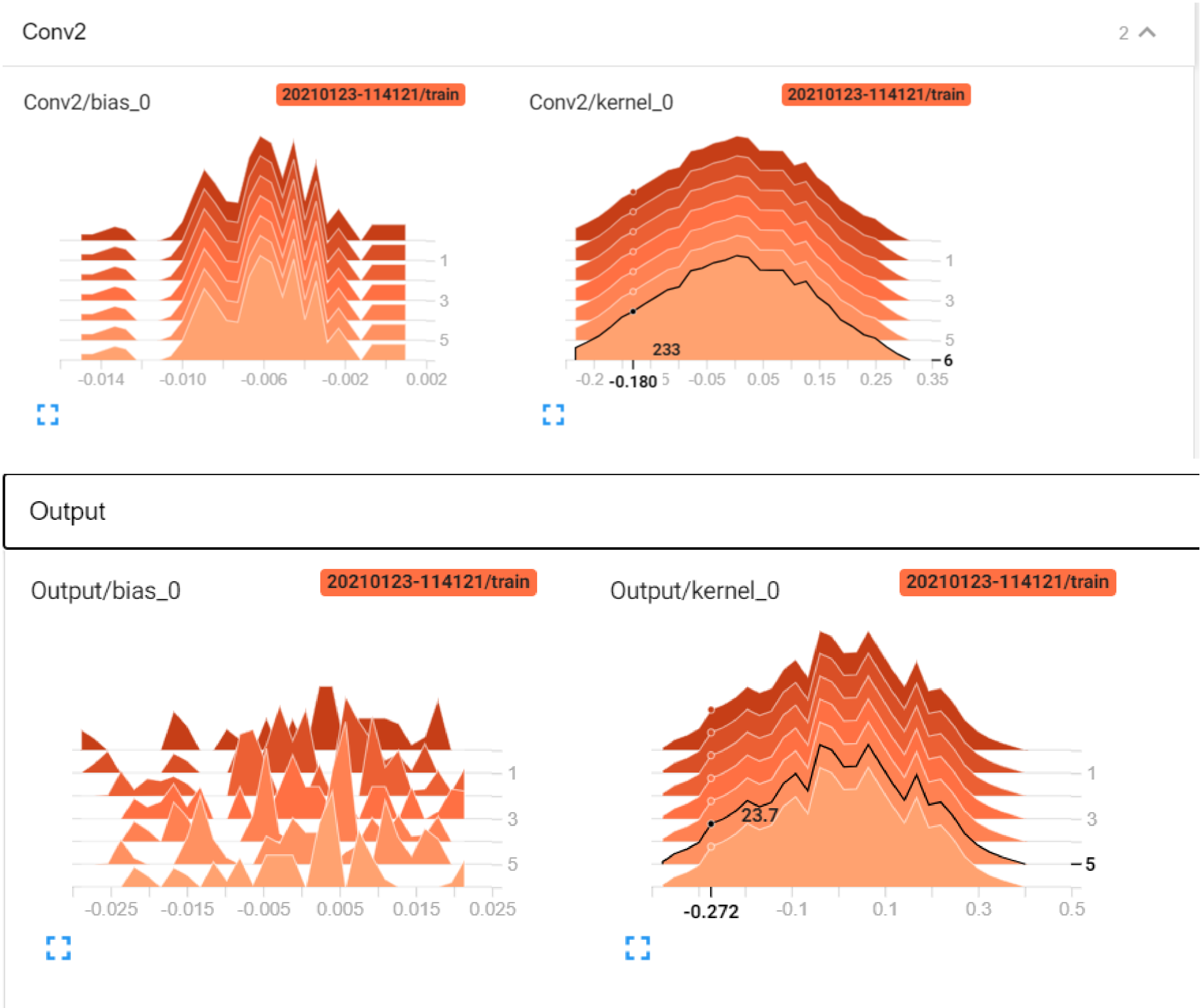
Conv1

Conv1/bias_0



Conv1/kernel_0





We got the updated optimal weights which are intializsed as normal distributions (he, glorot)