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-imagedatagenerator-with-flow-from-dataframe-8bd5776e45c1)](https://medium.com/@vijayabhaskar96/tutorial-on-keras-imagedatagenerator-with-flow-from-dataframe-8bd5776e45c1)

[https://medium.com/@vijayabhaskar96/tutorial-on-keras-flow-from-dataframe-1fd4493d237c (https://medium.com/@vijayabhaskar96/tutorial-on-keras-flow-from-dataframe-1fd4493d237c)](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, tr y 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 (https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html)

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 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 (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**
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.

In [1]:

```
import tensorflow as tf
import os
import numpy as np
import pandas as pd
import shutil
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from keras_preprocessing.image import ImageDataGenerator
from keras.layers import Dense, Activation, Flatten, Dropout, BatchNormalizati
on
from keras.layers import Conv2D, MaxPooling2D
from keras import regularizers, optimizers

from PIL import Image
```

In [2]:
```
tf.__version__
```

Out[2]:
```
'2.4.1'
```

Get the Dataset

In [3]:
```
!gdown --id 1Z4TyI7FcFVEx8qdl4jO9qxvxaqLSqoEu -q
```

In [4]:
```
!mkdir dataset
```

In [100]:
```
!unrar e "/content/rvl-cdip.rar" "dataset/"
```

In [6]:
```
shutil.move("./dataset/labels_final.csv","./")
```

Out[6]:
```
'./labels_final.csv'
```

Load Data

In [7]:
```
datadf = pd.read_csv("labels_final.csv")
datadf.tail(5)
```

Out[7]:

	path	label
47995	imagesk/k/q/l/kql82f00/tob07414.87.tif	10
47996	imagesi/i/r/r/irr80c00/2084343690_3692.tif	12
47997	imagesa/a/z/h/azh32d00/2063887153_7176.tif	6
47998	imagesg/g/p/d/gpd45f00/0060075263.tif	8
47999	imagesr/r/o/l/rol45d00/2064701657.tif	1

In [8]:
```
datadf["file_name"] = datadf["path"].apply(lambda x: x.split("/")[-1])
datadf["label"] = datadf["label"].apply(lambda x: str(x))
```

In [9]:
```
datadf.head(3)
```

Out[9]:

	path	label	file_name
0	imagesv/v/o/h/voh71d00/509132755+-2755.tif	3	509132755+-2755.tif
1	imagesl/l/x/t/lxt19d00/502213303.tif	3	502213303.tif
2	imagesx/x/e/d/xed05a00/2075325674.tif	2	2075325674.tif

In [10]:

```
# im = Image.open('dataset/2084343690_3692.tif').convert('RGB')
# print(im)
# im = im.resize((156,256))
# im.size
# # im
```

In [11]:

```
train_df, val_df = train_test_split(datadf,test_size=0.3)
datagen=ImageDataGenerator(rescale=1./255)
```

In [12]:

```
train_generator=datagen.flow_from_dataframe(
  dataframe=train_df,
  directory="./dataset/",
  x_col="file_name",
  y_col="label",
  batch_size=32,
  seed=42,
  shuffle=True,
  class_mode="categorical",
  target_size=(156,256))
```

```
Found 33600 validated image filenames belonging to 16 classes.
```

In [13]:

```
val_generator=datagen.flow_from_dataframe(
  dataframe=val_df,
  directory="./dataset/",
  x_col="file_name",
  y_col="label",
  batch_size=32,
  seed=42,
  shuffle=True,
  class_mode="categorical",
  target_size=(156,256))
```

```
Found 14400 validated image filenames belonging to 16 classes.
```

In [14]:

```
print(train_generator.n)
print(train_generator.batch_size)
```

```
33600
32
```

Model 1

INPUT --> VGG-16 without Top layers(FC) --> Conv Layer --> Maxpool Layer --> 2 FC layers --> Output Layer

In [43]:
```
from tensorflow.keras.applications.vgg16 import VGG16
import keras
from tensorflow.keras.layers import Dense,Input,Conv2D,MaxPool2D,Activation,Dr
opout,Flatten
from tensorflow.keras.models import Model
import datetime
%load_ext tensorboard
```

```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
```

In [61]:
```
!rm -rf logs/*
log_dir="logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir, histogr
am_freq=1, write_graph=True)
```

In [62]:
```
base_model = VGG16(
    weights='imagenet',  # Load weights pre-trained on ImageNet.
    input_shape=(156, 256, 3),
    include_top=False)
base_model.trainable = False
```

In [63]:
```
inputs = keras.Input(shape=(156, 256, 3))

x = base_model(inputs, training=False)

Conv1 = Conv2D(filters=64,kernel_size=(3,3),padding='same',
              activation='relu',kernel_initializer=tf.keras.initializers.he_no
rmal(seed=0),name='Conv1')(x)

Pool1 = MaxPool2D(pool_size=(2,2),strides=(2,2),padding='same',name='Pool1')(C
onv1)

#Flatten
flatten = Flatten(data_format='channels_last',name='Flatten')(Pool1)

FC1 = Dense(units=400,activation='relu',kernel_initializer=tf.keras.initialize
rs.glorot_normal(seed=32),name='FC1')(flatten)

FC2 = Dense(units=200,activation='relu',kernel_initializer=tf.keras.initialize
rs.glorot_normal(seed=33),name='FC2')(FC1)

outputs = Dense(units=16,activation='softmax',kernel_initializer=tf.keras.init
ializers.glorot_normal(seed=3),name='Output')(FC2)
```

In [64]:
```
model = keras.Model(inputs, outputs)
```

In [65]:
```
model.compile(optimizer=keras.optimizers.Adam(),
              loss="categorical_crossentropy",
              metrics=["accuracy"])
```

In [66]:
```
model.summary()
```

```
Model: "model_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_13 (InputLayer)        [(None, 156, 256, 3)]     0         
_________________________________________________________________
vgg16 (Functional)           (None, 4, 8, 512)         14714688  
_________________________________________________________________
Conv1 (Conv2D)               (None, 4, 8, 64)          294976    
_________________________________________________________________
Pool1 (MaxPooling2D)         (None, 2, 4, 64)          0         
_________________________________________________________________
Flatten (Flatten)            (None, 512)               0         
_________________________________________________________________
FC1 (Dense)                  (None, 400)               205200    
_________________________________________________________________
FC2 (Dense)                  (None, 200)               80200     
_________________________________________________________________
Output (Dense)               (None, 16)                3216      
=================================================================
Total params: 15,298,280
Trainable params: 583,592
Non-trainable params: 14,714,688
_________________________________________________________________
```

In [67]:

```
STEP_SIZE_TRAIN=train_generator.n//train_generator.batch_size
STEP_SIZE_VALID=val_generator.n//val_generator.batch_size

model.fit_generator(generator=train_generator,
                    steps_per_epoch=STEP_SIZE_TRAIN,
                    validation_data=val_generator,
                    validation_steps=STEP_SIZE_VALID,
                    epochs=10,
                    callbacks = [tensorboard_callback])
```

```
/usr/local/lib/python3.6/dist-packages/tensorflow/python/keras/engine/trainin
g.py:1844: UserWarning: `Model.fit_generator` is deprecated and will be remov
ed in a future version. Please use `Model.fit`, which supports generators.
  warnings.warn('`Model.fit_generator` is deprecated and '

Epoch 1/10
1050/1050 [==============================] - 191s 182ms/step - loss: 1.7084 -
accuracy: 0.4600 - val_loss: 1.2045 - val_accuracy: 0.6348
Epoch 2/10
1050/1050 [==============================] - 181s 172ms/step - loss: 1.0845 -
accuracy: 0.6658 - val_loss: 1.1227 - val_accuracy: 0.6555
Epoch 3/10
1050/1050 [==============================] - 178s 169ms/step - loss: 0.9254 -
accuracy: 0.7135 - val_loss: 1.0347 - val_accuracy: 0.6892
Epoch 4/10
1050/1050 [==============================] - 178s 170ms/step - loss: 0.8122 -
accuracy: 0.7448 - val_loss: 1.0269 - val_accuracy: 0.6981
Epoch 5/10
1050/1050 [==============================] - 177s 168ms/step - loss: 0.7240 -
accuracy: 0.7741 - val_loss: 1.1051 - val_accuracy: 0.6822
Epoch 6/10
1050/1050 [==============================] - 178s 169ms/step - loss: 0.6367 -
accuracy: 0.7938 - val_loss: 1.0458 - val_accuracy: 0.6976
Epoch 7/10
1050/1050 [==============================] - 176s 168ms/step - loss: 0.5584 -
accuracy: 0.8206 - val_loss: 1.0648 - val_accuracy: 0.7061
Epoch 8/10
1050/1050 [==============================] - 177s 169ms/step - loss: 0.4869 -
accuracy: 0.8404 - val_loss: 1.1052 - val_accuracy: 0.7090
Epoch 9/10
1050/1050 [==============================] - 177s 169ms/step - loss: 0.4232 -
accuracy: 0.8637 - val_loss: 1.1718 - val_accuracy: 0.7073
Epoch 10/10
1050/1050 [==============================] - 178s 170ms/step - loss: 0.3635 -
accuracy: 0.8813 - val_loss: 1.2640 - val_accuracy: 0.7006
```

Out[67]:

```
<tensorflow.python.keras.callbacks.History at 0x7fee7c14c4a8>
```

In [69]:

```
%tensorboard --logdir logs/fit
```

Model 2

INPUT --> VGG-16 without Top layers(FC) --> Conv Layer --> Maxpool Layer --> 2 FC layers --> Output Layer

In [78]:

```
base_model = VGG16(
    weights='imagenet',  # Load weights pre-trained on ImageNet.
    input_shape=(156, 256, 3),
    include_top=False)
base_model.trainable = False
```

In [79]:

```
!rm -rf logs/*
log_dir="logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir, histogr
am_freq=1, write_graph=True)
```

In [80]:

```
inputs = keras.Input(shape=(156, 256, 3))

x = base_model(inputs, training=False)


ConvFC1 = Conv2D(filters=400,kernel_size=(4,8),padding='valid',strides=(1,1),
              activation='relu',kernel_initializer=tf.keras.initializers.he_no
rmal(seed=0),name='ConvFC1')(x)

ConvFC2 = Conv2D(filters=200,kernel_size=(1,1),padding='valid',strides=(1,1),
              activation='relu',kernel_initializer=tf.keras.initializers.he_no
rmal(seed=0),name='ConvFC2')(ConvFC1)

#Train Params : 200*16 = 3200 + 16 (Bias weights) = 3216 params
flatten = Flatten(data_format='channels_last',name='Flatten')(ConvFC2)

outputs = Dense(units=16,activation='softmax',kernel_initializer=tf.keras.init
ializers.glorot_normal(seed=3),name='Output')(flatten)
```

In [81]:

```
model2 = keras.Model(inputs, outputs)
```

In [82]:

```
model2.compile(optimizer=keras.optimizers.Adam(),
              loss="categorical_crossentropy",
              metrics=["accuracy"])
```

In [83]:
```
model2.summary()
```

```
Model: "model_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_18 (InputLayer)        [(None, 156, 256, 3)]     0         
_________________________________________________________________
vgg16 (Functional)           (None, 4, 8, 512)         14714688  
_________________________________________________________________
ConvFC1 (Conv2D)             (None, 1, 1, 400)         6554000   
_________________________________________________________________
ConvFC2 (Conv2D)             (None, 1, 1, 200)         80200     
_________________________________________________________________
Flatten (Flatten)            (None, 200)               0         
_________________________________________________________________
Output (Dense)               (None, 16)                3216      
=================================================================
Total params: 21,352,104
Trainable params: 6,637,416
Non-trainable params: 14,714,688
_________________________________________________________________
```

In [84]:

```
STEP_SIZE_TRAIN=train_generator.n//train_generator.batch_size
STEP_SIZE_VALID=val_generator.n//val_generator.batch_size
model2.fit_generator(generator=train_generator,
                    steps_per_epoch=STEP_SIZE_TRAIN,
                    validation_data=val_generator,
                    validation_steps=STEP_SIZE_VALID,
                    epochs=10,
                     callbacks=[tensorboard_callback])
```

```
/usr/local/lib/python3.6/dist-packages/tensorflow/python/keras/engine/trainin
g.py:1844: UserWarning: `Model.fit_generator` is deprecated and will be remov
ed in a future version. Please use `Model.fit`, which supports generators.
  warnings.warn('`Model.fit_generator` is deprecated and '

Epoch 1/10
1050/1050 [==============================] - 181s 172ms/step - loss: 1.7048 -
accuracy: 0.4965 - val_loss: 1.1781 - val_accuracy: 0.6454
Epoch 2/10
1050/1050 [==============================] - 181s 172ms/step - loss: 1.0498 -
accuracy: 0.6784 - val_loss: 1.0822 - val_accuracy: 0.6787
Epoch 3/10
1050/1050 [==============================] - 178s 170ms/step - loss: 0.8953 -
accuracy: 0.7242 - val_loss: 0.9871 - val_accuracy: 0.7110
Epoch 4/10
1050/1050 [==============================] - 182s 173ms/step - loss: 0.7747 -
accuracy: 0.7621 - val_loss: 0.9891 - val_accuracy: 0.7157
Epoch 5/10
1050/1050 [==============================] - 182s 174ms/step - loss: 0.6844 -
accuracy: 0.7874 - val_loss: 1.0174 - val_accuracy: 0.7072
Epoch 6/10
1050/1050 [==============================] - 182s 173ms/step - loss: 0.6035 -
accuracy: 0.8092 - val_loss: 1.0656 - val_accuracy: 0.7089
Epoch 7/10
1050/1050 [==============================] - 182s 174ms/step - loss: 0.5421 -
accuracy: 0.8278 - val_loss: 1.0484 - val_accuracy: 0.7247
Epoch 8/10
1050/1050 [==============================] - 182s 173ms/step - loss: 0.4730 -
accuracy: 0.8494 - val_loss: 1.0757 - val_accuracy: 0.7199
Epoch 9/10
1050/1050 [==============================] - 181s 173ms/step - loss: 0.4368 -
accuracy: 0.8607 - val_loss: 1.1674 - val_accuracy: 0.7105
Epoch 10/10
1050/1050 [==============================] - 182s 173ms/step - loss: 0.3912 -
accuracy: 0.8737 - val_loss: 1.3354 - val_accuracy: 0.6874
```

Out[84]:

```
<tensorflow.python.keras.callbacks.History at 0x7fee5a4659b0>
```

In [99]:

```
%tensorboard --logdir logs/fit
```

Model 3

In [88]:

```
!rm -rf logs/*
log_dir="logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir, histogr
am_freq=1, write_graph=True)
```

In [89]:

```
base_model = VGG16(
    weights='imagenet',  # Load weights pre-trained on ImageNet.
    input_shape=(156, 256, 3),
    include_top=False)
base_model.trainable = False
```

In [90]:

```
last_6_layers = base_model.layers[-6:]
for layer in last_6_layers:
  layer.trainable = True

for idx, layer in enumerate(base_model.layers):
  print("Layer %d Trainaible : %s"%(idx+1, layer.trainable))
```

```
Layer 1 Trainaible : False
Layer 2 Trainaible : False
Layer 3 Trainaible : False
Layer 4 Trainaible : False
Layer 5 Trainaible : False
Layer 6 Trainaible : False
Layer 7 Trainaible : False
Layer 8 Trainaible : False
Layer 9 Trainaible : False
Layer 10 Trainaible : False
Layer 11 Trainaible : False
Layer 12 Trainaible : False
Layer 13 Trainaible : False
Layer 14 Trainaible : True
Layer 15 Trainaible : True
Layer 16 Trainaible : True
Layer 17 Trainaible : True
Layer 18 Trainaible : True
Layer 19 Trainaible : True
```

In [91]:

```
inputs = keras.Input(shape=(156, 256, 3))

x = base_model(inputs, training=False)

ConvFC1 = Conv2D(filters=400,kernel_size=(4,8),padding='valid',strides=(1,1),
              activation='relu',kernel_initializer=tf.keras.initializers.he_no
rmal(seed=0),name='ConvFC1')(x)

ConvFC2 = Conv2D(filters=200,kernel_size=(1,1),padding='valid',strides=(1,1),
              activation='relu',kernel_initializer=tf.keras.initializers.he_no
rmal(seed=0),name='ConvFC2')(ConvFC1)

flatten = Flatten(data_format='channels_last',name='Flatten')(ConvFC2)

outputs = Dense(units=16,activation='softmax',kernel_initializer=tf.keras.init
ializers.glorot_normal(seed=3),name='Output')(flatten)
```

In [92]:

```
model3 = keras.Model(inputs, outputs)
```

In [93]:

```
model3.compile(optimizer=keras.optimizers.Adam(),
              loss="categorical_crossentropy",
              metrics=["accuracy"])
```

In [94]:

```
model3.summary()
```

```
Model: "model_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_21 (InputLayer)        [(None, 156, 256, 3)]     0         
_________________________________________________________________
vgg16 (Functional)           (None, 4, 8, 512)         14714688  
_________________________________________________________________
ConvFC1 (Conv2D)             (None, 1, 1, 400)         6554000   
_________________________________________________________________
ConvFC2 (Conv2D)             (None, 1, 1, 200)         80200     
_________________________________________________________________
Flatten (Flatten)            (None, 200)               0         
_________________________________________________________________
Output (Dense)               (None, 16)                3216      
=================================================================
Total params: 21,352,104
Trainable params: 6,637,416
Non-trainable params: 14,714,688
_________________________________________________________________
```

In [95]:

```
STEP_SIZE_TRAIN=train_generator.n//train_generator.batch_size
STEP_SIZE_VALID=val_generator.n//val_generator.batch_size
model3.fit_generator(generator=train_generator,
                    steps_per_epoch=STEP_SIZE_TRAIN,
                    validation_data=val_generator,
                    validation_steps=STEP_SIZE_VALID,
                    epochs=10,
                     callbacks=[tensorboard_callback])
```

```
/usr/local/lib/python3.6/dist-packages/tensorflow/python/keras/engine/trainin
g.py:1844: UserWarning: `Model.fit_generator` is deprecated and will be remov
ed in a future version. Please use `Model.fit`, which supports generators.
  warnings.warn('`Model.fit_generator` is deprecated and '

Epoch 1/10
1050/1050 [==============================] - 175s 166ms/step - loss: 1.6905 -
accuracy: 0.4921 - val_loss: 1.1690 - val_accuracy: 0.6485
Epoch 2/10
1050/1050 [==============================] - 178s 170ms/step - loss: 1.0465 -
accuracy: 0.6801 - val_loss: 1.0734 - val_accuracy: 0.6781
Epoch 3/10
1050/1050 [==============================] - 178s 169ms/step - loss: 0.8770 -
accuracy: 0.7304 - val_loss: 1.0044 - val_accuracy: 0.7050
Epoch 4/10
1050/1050 [==============================] - 178s 169ms/step - loss: 0.7709 -
accuracy: 0.7605 - val_loss: 1.0486 - val_accuracy: 0.6879
Epoch 5/10
1050/1050 [==============================] - 178s 169ms/step - loss: 0.6742 -
accuracy: 0.7867 - val_loss: 1.0038 - val_accuracy: 0.7107
Epoch 6/10
1050/1050 [==============================] - 177s 169ms/step - loss: 0.6090 -
accuracy: 0.8066 - val_loss: 1.0802 - val_accuracy: 0.6957
Epoch 7/10
1050/1050 [==============================] - 177s 169ms/step - loss: 0.5520 -
accuracy: 0.8235 - val_loss: 1.0356 - val_accuracy: 0.7145
Epoch 8/10
1050/1050 [==============================] - 178s 170ms/step - loss: 0.4993 -
accuracy: 0.8373 - val_loss: 1.1582 - val_accuracy: 0.7101
Epoch 9/10
1050/1050 [==============================] - 178s 169ms/step - loss: 0.4343 -
accuracy: 0.8619 - val_loss: 1.3375 - val_accuracy: 0.6953
Epoch 10/10
1050/1050 [==============================] - 177s 169ms/step - loss: 0.4023 -
accuracy: 0.8715 - val_loss: 1.2258 - val_accuracy: 0.7108
```

Out[95]:

```
<tensorflow.python.keras.callbacks.History at 0x7fee5a1c0ac8>
```

In [97]:

```
%tensorboard --logdir logs/fit
```

In [ ]:

In [ ]:

In [ ]: