Note: For easy navigation, please use the Table of contents

Section 0: Name

Pamela Sin Hui

Section 1: Project Title

Product Detection Image Classification - Using Kerastuner, Transfer Learning, Finetuning

Section 2: Goals, Dataset, Tasks

1. Goals

In Shopee Product Detection Dataset, there are more than 100k images directly from E-commercial industry field. You will be able to explore the real-world images which is noisy and long-tailed, and let your model predict the correct categories for the images. There contains 42 most popular categories product at Shopee.

https://www.kaggle.com/c/shopee-product-detection-open/overview

2. Dataset

- **train folder** - Contains 42 folders (each labelled '00', '01', ..., '41'), each containing images of 42 popular product categories at Shoppee.
- **test folder** - Contains images where the goal is to give labels (from the 42 categories) to each image.
- **train.csv** - 2 Columns (filename and category)
- **test.csv** - 2 Columns (filename and dummy category)

3. Tasks

Note: We re-write certain blocks of code again in some cells due to the possibility of Colab-run timeouts

- Data exploration
- Create train and validation datasets from smaller sample subset of 3000 images - For more rapid coding and testing
- Transfer learning (mobilenet and inception v3 featurizers) + Kerastuner randomsearch
- Transfer learning + Hyperparameter search with tensorboard visualisation
- Perform further training on best Transfer Learning featurizer + hyperparameters
- Check model performance and metrics
- Train and validate on global dataset
- Ideas for improvement

```
import datetime, os
import shutil
import zipfile
import pickle

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.metrics import classification_report

import tensorflow as tf
from keras.layers import Dropout
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
from tensorflow.keras.applications.inception_v3 import InceptionV3
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras import layers
from tensorflow.keras.layers import Input, Conv2D, LeakyReLU, Flatten, Dense,
from tensorflow.keras import Model
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.models import load_model
from tensorflow.keras.optimizers import RMSprop
from tensorflow.keras.optimizers import SGD
```

```
#DO NOT RUN THIS CELL AFTER FIRST RUN
#Uploading the zip file to Google Drive took ~3 hours
#The below unzip operation took ~ 45 min, but did not unzip folders 15 to 41
#Manual unzip locally was done ~45 min, and manual drag and drop upload of fo
!unzip "/content/drive/My Drive/shopee-product-detection-open.zip" -d "/conte
```

```
Streaming output truncated to the last 5000 lines.
  inflating: /content/drive/My Drive/train/train/train/13/31c2dc64a0007d1f3
  inflating: /content/drive/My Drive/train/train/train/13/31c9a82cf3f38dcfd
  inflating: /content/drive/My Drive/train/train/train/13/31dc15cb2eb3f2c91
  inflating: /content/drive/My Drive/train/train/train/13/31e4d096df72eaaca
  inflating: /content/drive/My Drive/train/train/train/13/31ebeaaa50685c435
  inflating: /content/drive/My Drive/train/train/train/13/320ab3dfa81199c85
  inflating: /content/drive/My Drive/train/train/train/13/32174db1e4b57b5c4
  inflating: /content/drive/My Drive/train/train/train/13/32209e1ac67fb78f7
  inflating: /content/drive/My Drive/train/train/train/13/322143750fb1e4717
  inflating: /content/drive/My Drive/train/train/train/13/322df0ed3b7ffe60f
  inflating: /content/drive/My Drive/train/train/train/13/3230da734295bebf5
  inflating: /content/drive/My Drive/train/train/train/13/3232afc5ae1762af9
  inflating: /content/drive/My Drive/train/train/train/13/325572f3ffc7950da
  inflating: /content/drive/My Drive/train/train/train/13/325a78e4243ec15ee
  inflating: /content/drive/My Drive/train/train/train/13/3267f4caf8bc4823f
  inflating: /content/drive/My Drive/train/train/train/13/327250586f75d4ba5
  inflating: /content/drive/My Drive/train/train/train/13/32d22a2cdc1b0eb15
  inflating: /content/drive/My Drive/train/train/train/13/3364558dda36d0ab0
  inflating: /content/drive/My Drive/train/train/train/13/336fffd7e5eab0feb
  inflating: /content/drive/My Drive/train/train/train/13/337fac602c56672ff
  inflating: /content/drive/My Drive/train/train/train/13/338a44430a849d446
```

```
inflating: /content/drive/My Drive/train/train/train/13/33bb74713f53595ff
inflating: /content/drive/My Drive/train/train/train/13/33c71d4f438084112
inflating: /content/drive/My Drive/train/train/train/13/33d3862ec9c022e60
inflating: /content/drive/My Drive/train/train/train/13/34217c80a8c4a7dcc
inflating: /content/drive/My Drive/train/train/train/13/342de12b5f9f4f80c
inflating: /content/drive/My Drive/train/train/train/13/3447112e0247f4830
inflating: /content/drive/My Drive/train/train/train/13/34c42328e2389febc
inflating: /content/drive/My Drive/train/train/train/13/34effad42d6fc0e4e
inflating: /content/drive/My Drive/train/train/train/13/34f2cf349431d551c
inflating: /content/drive/My Drive/train/train/train/13/351b96269366a3d93
inflating: /content/drive/My Drive/train/train/train/13/35212edd9057a9a0f
inflating: /content/drive/My Drive/train/train/train/13/3525f1758798a6884
inflating: /content/drive/My Drive/train/train/train/13/3526ba6a256d1f159
inflating: /content/drive/My Drive/train/train/train/13/3548a1b532690c665
inflating: /content/drive/My Drive/train/train/train/13/359fc4987220dc8e9
inflating: /content/drive/My Drive/train/train/train/13/35a4530e0d2339373
inflating: /content/drive/My Drive/train/train/train/13/364b1db2c553d4868
inflating: /content/drive/My Drive/train/train/train/13/3684088dd79bd46a0
inflating: /content/drive/My Drive/train/train/train/13/36985e9089e0d7ca4
inflating: /content/drive/My Drive/train/train/train/13/36fc3178b3dff0b4e
inflating: /content/drive/My Drive/train/train/train/13/37063530fb8b4559c
inflating: /content/drive/My Drive/train/train/train/13/370b537f474a48d78
inflating: /content/drive/My Drive/train/train/train/13/3714fb099b5dcbfab
inflating: /content/drive/My Drive/train/train/train/13/37452f745da28286e
inflating: /content/drive/My Drive/train/train/train/13/3756c1aa3374bdf37
inflating: /content/drive/My Drive/train/train/train/13/37614f5f974d4bb8b
inflating: /content/drive/My Drive/train/train/train/13/377db5d2f8bf82e29
inflating: /content/drive/My Drive/train/train/train/13/37850a979ca9bf98b
inflating: /content/drive/My Drive/train/train/train/13/37ca866e2e6bbc079
inflating: /content/drive/My Drive/train/train/train/13/37cb389558ba72529
inflating: /content/drive/My Drive/train/train/train/13/37d1afc9e17fa4b36
inflating: /content/drive/My Drive/train/train/train/13/37e320be999fcffbd
inflating: /content/drive/My Drive/train/train/train/13/381eaf2910c4fb54a
inflating: /content/drive/My Drive/train/train/train/13/383c62550c4f6eca8
inflating: /content/drive/My Drive/train/train/train/13/387a3a90749478e3c
inflating: /content/drive/My Drive/train/train/train/13/387ca76cb70a51509
inflating: /content/drive/My Drive/train/train/train/13/389558be25a294f72
```

Section 3: Data Engineering

- Data exploration
- Prepare smaller sample subset of 3000 images

```
df_traincsv = pd.read_csv('/content/drive/My Drive/train.csv')
df_traincsv.head()
```

	filename	category
0	45e2d0c97f7bdf8cbf3594beb6fdcda0.jpg	3
1	f74d1a5fc2498bbbfa045c74e3cc333e.jpg	3
2	f6c172096818c5fab10ecae722840798.jpg	3
3	251ffd610399ac00fea7709c642676ee.jpg	3
4	73c7328b8eda399199fdedec6e4badaf.jpg	3

```
df_traincsv.shape
```

```
(105390, 2)
```

```
df_traincsv.dtypes
```

```
filename    object
category     int64
dtype: object
```

```
df_check_files = df_traincsv.groupby(['category']).count()
df_check_files.head()
```

category	filename
0	2683
1	2702
2	2687
3	2703
4	2703

```
#Sanity check - Count number of files in the original train dataset path uplo
import os
df_check_files['directory_count'] = 0
for i in range(42):
  img_folder_path = "/content/drive/My Drive/train/train/train/" + folder_nam
  dirListing = os.listdir(img_folder_path)
  df_check_files.loc[i,'directory_count'] = len(dirListing)
```

```
#Due to large size of 100K images in original train dataset comparison of th
#This cell was accidently run after running another cell that moved 15% of th
df_check_files['difference'] = df_check_files.filename - df_check_files.direc
df_check_files
```

category	filename	directory_count	difference
0	2683	2281	402
1	2702	2298	404
2	2687	2284	403
3	2703	2298	405
4	2703	2298	405
5	2641	2245	396
6	2641	2245	396
7	2660	2262	398
8	2700	2295	405
9	2698	2293	405
10	2672	2272	400
11	1843	1567	276
12	2691	2287	404
13	2682	2280	402
14	2684	2282	402
15	2632	2237	395
16	2665	2265	400
17	1553	1320	233
18	2103	1789	314
19	2679	2337	342
20	2653	2256	397
21	2598	2208	390
22	2623	2230	393
23	2540	2159	381
24	2705	2299	406
25	2692	2288	404
26	2684	2281	403

27	2702	2297	405
28	2561	2177	384
29	2138	1817	321

```
#Creating a list of folder names from '00' to '41'
folder_names = []
for i in range(0,42,1):
  if len(str(i)) == 1:
    folder_names.append(str(0) + str(i))
  else:
    folder_names.append(str(i))
print(folder_names)
```

```
['00', '01', '02', '03', '04', '05', '06', '07', '08', '09', '10', '11', '1
```

```
#Create new_train_df dataframe by checking which image filenames are in each
#We had split the data into train and validation sets, so we can get the file
import os
from os import listdir
from os.path import isfile, join
the_index = 0
new_train_df = pd.DataFrame(columns = ['filename', 'category'])
for i in range(42):
  img_folder_path = "/content/drive/My Drive/train/train/train/" + folder_nam
  onlyfiles = [f for f in listdir(img_folder_path) if isfile(join(img_folder_
  temp_df = pd.DataFrame(columns = ['filename', 'category'])
  temp_df.filename = onlyfiles
  temp_df.category = i
  new_train_df = new_train_df.append(temp_df, ignore_index=True)
```

```
new_train_df.head()
```

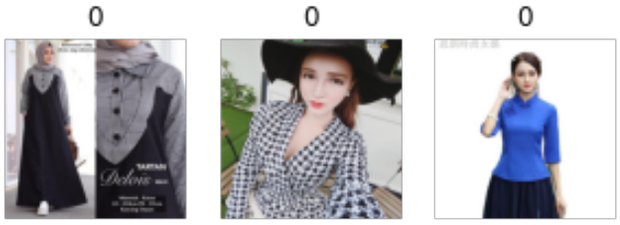
	filename	category
0	d089953699c05da67da914cceb991d5e.jpg	0
1	d0018d2c844645f14c0e33de7415348d.jpg	0
2	d2178eab8cb870b65a396cd1808f6528.jpg	0
3	c6187c65a9883ffae35c9e3301a32417.jpg	0
4	ce3a9f520751cc48de872d500ecd541a.jpg	0

```
new_train_df.shape
```

```
(89646, 2)
```

```
#Show random 3 images from each category
import matplotlib.image as mpimg
for i in range(42):
  fig=plt.figure(figsize=(5, 5))
  img_folder_path = "/content/drive/My Drive/train/train/train/" + folder_nam
  temp_list = list(new_train_df.loc[new_train_df.category == i].sample(n=3).f
  print('Category ' + str(i) + ': ')
  for j in range(3):
    img = mpimg.imread(img_folder_path + '/' + temp_list[j])
    fig.add_subplot(1, 3, j+1)
    plt.axis('off')
    plt.title(str(i))
    plt.imshow(img)
  plt.show()
```

Category 0:



Category 1:

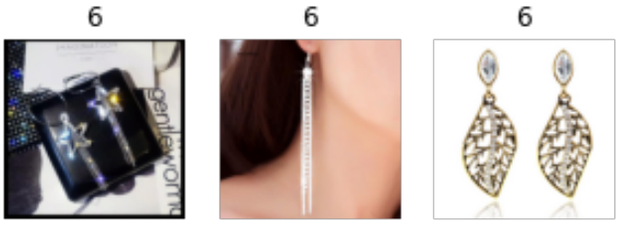
Category 2:

Category 3:

Category 4:

Category 5:

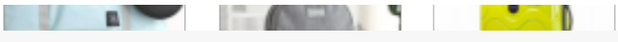
Category 6:



Category 7:

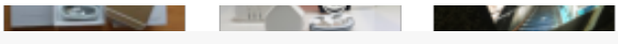

```
1 df_traincsv.head()
```

	filename	category
0	45e2d0c97f7bdf8cbf3594beb6fdcda0.jpg	3
1	f74d1a5fc2498bbbfa045c74e3cc333e.jpg	3
2	f6c172096818c5fab10ecae722840798.jpg	3
3	251ffd610399ac00fea7709c642676ee.jpg	3
4	73c7328b8eda399199fdedec6e4badaf.jpg	3



```
1 df_testcsv = pd.read_csv('/content/drive/My Drive/test.csv')
2 df_testcsv.head()
```

	filename	category
0	fd663cf2b6e1d7b02938c6aaae0a32d2.jpg	43
1	c7fd77508a8c355eaab0d4e10efd6b15.jpg	43
2	127f3e6d6e3491b2459812353f33a913.jpg	43
3	5ca4f2da11eda083064e6c36f37eeb81.jpg	43
4	46d681a542f2c71be017eef6aae23313.jpg	43



```
1 df_testcsv.shape
```

```
(12186, 2)
```

```
img_folder_path = "/content/drive/My Drive/test/test/test"
dirListing = os.listdir(img_folder_path)
len(dirListing)
```

```
12192
```

Can consider making this sampling logic into a function and adjust N=3000

```
#DO NOT RUN THIS CELL AFTER FIRST RUN
#We will take a sample of 3000 images from the train dataset to build initial
#From the sample of 3000 images, we will also split into train and validation
#We create a df_traincsv_3000marker dataframe to later note which photos have
df_traincsv_3000marker = df_traincsv
df_traincsv_3000marker['selected_for_3000'] = 0
df_traincsv_3000marker['selected_for_3000_validation'] = 0
df_traincsv_3000marker.head()
```

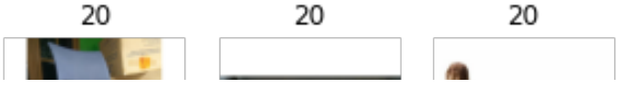
	filename	category	selected_for_3000	selected_fc
0	45e2d0c97f7bdf8cbf3594beb6fdcda0.jpg	3	0	
1	f74d1a5fc2498bbbfa045c74e3cc333e.jpg	3	0	
2	f6c172096818c5fab10ecae722840798.jpg	3	0	
3	251ffd610399ac00fea7709c642676ee.jpg	3	0	
4	73c7328b8eda399199fdedec6e4badaf.jpg	3	0	

```
#DO NOT RUN THIS CELL AFTER FIRST RUN
#Taking a sample of 3000 from the train dataset
#Creating folders '00' to '41', and sampling 3% of each category
#Images are COPIED, leaving the original test dataset as is
rolling = 0
for two_digit in folder_names:
  path = "/content/drive/My Drive/sample_3000/train/" + two_digit
  os.mkdir(path)

  temp_df = df_traincsv.loc[df_traincsv.category == rolling,:].sample(frac=0.
  temp_index = temp_df.index
  df_traincsv_3000marker.loc[temp_index, 'selected_for_3000'] = 1
  temp_df.reset_index(inplace=True)

  for i in range(len(temp_df)):
    f = temp_df.loc[i,'filename']
    shutil.copy("/content/drive/My Drive/train/train/train/" + two_digit + '/
  rolling+=1
```

```
Category 20:
```



```
#DO NOT RUN THIS CELL AFTER FIRST RUN
#TO BE REPEATED (WITH SOME CHANGE) DURING RUN ON MORE IMAGES
#From the sample of 3000, we MOVE 20% of images in each category to the valid
rolling = 0
for two_digit in folder_names:
  path = "/content/drive/My Drive/sample_3000/validation/" + two_digit
  os.mkdir(path)

  df_traincsv_3000marker
  temp_df = df_traincsv_3000marker.loc[(df_traincsv_3000marker.category==roll
  temp_index = temp_df.index
  df_traincsv_3000marker.loc[temp_index, 'selected_for_3000_validation'] = 1
  temp_df.reset_index(inplace=True)

  for i in range(len(temp_df)):
    f = temp_df.loc[i,'filename']
    shutil.move("/content/drive/My Drive/sample_3000/train/" + two_digit + '/

  rolling+=1
```

Section 4: Feature Engineering

- Use ImageDataGenerator to generate augmented images as the sample 3000 dataset is small

Section 5: Model Engineering

- Transfer learning - Featurizers from mobilenet and inceptionv3 pre-trained CNNs
- Kerastuner RandomSearch - No. of hidden layers, no. units in hidden layers, Optimizer
- Hyperparameter search with tensorboard visualisation - No. units in hidden layer1, no. units in hidden layer2, Optimizer, % dropout

```
featurizer_mobilenet = MobileNetV2(input_shape=(224, 224, 3), include_top=Fal
featurizer_inception = InceptionV3(input_shape = (299, 299, 3), include_top =
#featurizer_mobilenet.summary()
#featurizer_inception.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-appli
9412608/9406464 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/keras-appli
87916544/87910968 [==============================] - 1s 0us/step
```

--- Kerastuner - RandomSearch ---

```
#mobilenet featurizer and kerastuner RandomSearch on 1. Hidden layers (2 or 3
#Running for just 2 iterations - We will later use searching across another s
!pip install keras-tuner
import kerastuner as kt
from kerastuner import tuners

def MyHyperModel(hp):

  image_size = (224, 224)

  featurizer_mobilenet.trainable = False
  model_input_mobilenet = featurizer_mobilenet.input
  x = Flatten()(featurizer_mobilenet.output)
  for i in range(hp.Int('num_layers_rnn', 2, 3)):
    x = Dense(hp.Int('units', min_value=64, max_value=128, step=16), activati
  #x = Dropout(0.2)(x)
  x = Dense(42, activation='softmax')(x)

  model_mobilenet = Model(model_input_mobilenet, x)
  #model_mobilenet.summary()
  model_mobilenet.compile(loss='categorical_crossentropy',
                          optimizer=hp.Choice('optimizer', values= ['Adam', '
                          metrics=['acc', tf.keras.metrics.TopKCategoricalAcc

  return model_mobilenet

"""tuner = kt.Hyperband(MyHyperModel,
                     objective = 'acc',
                     max_epochs = 2,
                     factor = 3,
                     directory = '/content/drive/My Drive/sample_3000',
                     project_name = 'mobilenet_hyper')"""

tuner = kt.RandomSearch(
    MyHyperModel,
    objective='val_acc',
    max_trials = 2, #Change this for more iterations
    directory='/content/drive/My Drive/sample_3000',
    project_name='mobilenet_hyper3'
)

image_size = (240,240)
train_datagen = ImageDataGenerator(
      rotation_range=45.0,
      horizontal_flip=True,
      shear_range=30.0,
      zoom_range=0.5,
      preprocessing_function=preprocess_input)

X_train = train_datagen.flow_from_directory('/content/drive/My Drive/sample_3
```

← the version later on dropout tuning, and can supercede this version

```
val_datagen = ImageDataGenerator(preprocessing_function=preprocess_input)
X_val = val_datagen.flow_from_directory('/content/drive/My Drive/sample_3000/
X_val_batch, y_val_batch = next(X_val)

logdir_mobilenet = os.path.join("/content/drive/My Drive/sample_3000/logs/mob
tensorboard_callback_mobilenet = tf.keras.callbacks.TensorBoard(logdir_mobile

tuner.search(X_train, epochs=1, validation_data=X_val, callbacks=[tensorboard
```

```
Found 2532 images belonging to 42 classes.
Found 630 images belonging to 42 classes.
 1/80 [..............................] - ETA: 0s - loss: 4.3508 - acc: 0.00
Instructions for updating:
use `tf.profiler.experimental.stop` instead.
80/80 [==============================] - 2057s 26s/step - loss: 4.0510 - ac
```

Trial complete

Trial summary

|-Trial ID: 57634774b2d83a6cbcd3ec088809258b
|-Score: 0.03650793805718422
|-Best step: 0

Hyperparameters:

|-num_layers_rnn: 2
|-optimizer: Adadelta
|-units: 80

```
80/80 [==============================] - 62s 781ms/step - loss: 4.2620 - ac
```

Trial complete

Trial summary

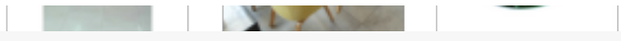
|-Trial ID: df3ffeb1bd27ff63f09e911f785ef980
|-Score: 0.1190476194024086
|-Best step: 0

Hyperparameters:

|-num_layers_rnn: 2
|-optimizer: Adam
|-units: 80

```
INFO:tensorflow:Oracle triggered exit
```

--- Tensorboard - hyperparameter search ---



```
#mobilenet featurizer + kerastunr + tensorboard visualisation setup
#Get a sense of which hyperparameters make training accuracy faster
!pip install keras-tuner
import kerastuner as kt
from kerastuner import tuners
from tensorboard.plugins.hparams import api as hpp

```

```
#HP_NUM_LAYERS = hpp.HParam('num_layers', hpp.Discrete([2,3]))
HP_NUM_UNITS1 = hpp.HParam('num_units1', hpp.Discrete([96,128])) #CHANGE THIS
HP_NUM_UNITS2 = hpp.HParam('num_units2', hpp.Discrete([64,96])) #CHANGE THIS
HP_DROPOUT = hpp.HParam('dropout', hpp.RealInterval(0.1, 0.3))
HP_OPTIMIZER = hpp.HParam('optimizer', hpp.Discrete(['adam', 'sgd']))
METRIC_ACCURACY = 'accuracy'

with tf.summary.create_file_writer('logs4/hparam_tuning').as_default():
  hpp.hparams_config(
    hparams=[HP_NUM_UNITS1, HP_NUM_UNITS2, HP_DROPOUT, HP_OPTIMIZER],
    metrics=[hpp.Metric(METRIC_ACCURACY)]
  )

image_size = (240,240)
train_datagen = ImageDataGenerator(
      rotation_range=45.0,
      horizontal_flip=True,
      shear_range=30.0,
      zoom_range=0.5,
      preprocessing_function=preprocess_input)

X_train = train_datagen.flow_from_directory('/content/drive/My Drive/sample_3
val_datagen = ImageDataGenerator(preprocessing_function=preprocess_input)
X_val = val_datagen.flow_from_directory('/content/drive/My Drive/sample_3000/
X_val_batch, y_val_batch = next(X_val)

def MyHyperModel(hp):

  featurizer_mobilenet.trainable = False
  model_input_mobilenet = featurizer_mobilenet.input
  x = Flatten()(featurizer_mobilenet.output)
  #for i in range(hparams[HP_NUM_LAYERS]):
  x = Dense(hparams[HP_NUM_UNITS1], activation='relu')(x)
  x = Dense(hparams[HP_NUM_UNITS2], activation='relu')(x)
  x = Dropout(hparams[HP_DROPOUT])(x)
  x = Dense(42, activation='softmax')(x)

  model_mobilenet = Model(model_input_mobilenet, x)
  #model_mobilenet.summary()
  model_mobilenet.compile(loss='categorical_crossentropy',
                          optimizer=hparams[HP_OPTIMIZER],
                          metrics=['accuracy', tf.keras.metrics.TopKCategoric
  ###################################CHANGE HOW MANY EPOCHS FOR EACH MODEL RUN
  model_mobilenet.fit(
    X_train, epochs = 10,
    callbacks=[
        tf.keras.callbacks.TensorBoard(logdir),  # log metrics
        hpp.KerasCallback(logdir, hparams),  # log hparams
    ],
  )

  print(model_mobilenet.evaluate(X_val))
```

hope this one used and not the above Hypermodel ?

```
  a, validationaccuracy, d = model_mobilenet.evaluate(X_val)

  return validationaccuracy

def run(run_dir, hparams):
  with tf.summary.create_file_writer(run_dir).as_default():
    hpp.hparams(hparams)
    validationaccuracy = MyHyperModel(hparams)
    tf.summary.scalar(METRIC_ACCURACY, validationaccuracy, step=1)

"""tuner = kt.Hyperband(MyHyperModel,
                     objective = 'acc',
                     max_epochs = 2,
                     factor = 3,
                     directory = '/content/drive/My Drive/sample_3000',
                     project_name = 'mobilenet_hyper')"""

"""tuner = kt.RandomSearch(
    MyHyperModel,
    objective='val_acc',
    max_trials = 2,
    directory='/content/drive/My Drive/sample_3000',
    project_name='mobilenet_hyper3'
)"""

logdir = os.path.join("/content/drive/My Drive/sample_3000/logs/metrics", dat

#logdir_mobilenet = os.path.join("/content/drive/My Drive/sample_3000/logs/mo
#tensorboard_callback_mobilenet = tf.keras.callbacks.TensorBoard(logdir_mobil

#tuner.search(X_train, epochs=1, validation_data = X_val, callbacks=[tensorbo

session_num = 0

#for num_layers in HP_NUM_LAYERS.domain.values:
for num_units1 in HP_NUM_UNITS1.domain.values:
  for num_units2 in HP_NUM_UNITS2.domain.values:
    for dropout_rate in (HP_DROPOUT.domain.min_value, HP_DROPOUT.domain.max_v
      for optimizer in HP_OPTIMIZER.domain.values:
        hparams = {
            HP_NUM_UNITS1: num_units1,
            HP_NUM_UNITS2: num_units2,
            HP_DROPOUT: dropout_rate,
            HP_OPTIMIZER: optimizer,
        }
        run_name = "run-%d" % session_num
        print('--- Starting trial: %s' % run_name)
        print({h.name: hparams[h] for h in hparams})
        run('logs4/hparam_tuning/' + run_name, hparams)
        session_num += 1
```

Can make these constants

Requirement already satisfied: keras-tuner in /usr/local/lib/python3.6/dist

```
Requirement already satisfied: keras-tuner in /usr/local/lib/python3.6/dist
Requirement already satisfied: colorama in /usr/local/lib/python3.6/dist-pa
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.6/dis
Requirement already satisfied: tqdm in /usr/local/lib/python3.6/dist-packag
Requirement already satisfied: requests in /usr/local/lib/python3.6/dist-pa
Requirement already satisfied: terminaltables in /usr/local/lib/python3.6/d
Requirement already satisfied: future in /usr/local/lib/python3.6/dist-pack
Requirement already satisfied: numpy in /usr/local/lib/python3.6/dist-packa
Requirement already satisfied: tabulate in /usr/local/lib/python3.6/dist-pa
Requirement already satisfied: scipy in /usr/local/lib/python3.6/dist-packa
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.6/dis
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.6/dis
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.
Found 2532 images belonging to 42 classes.
Found 630 images belonging to 42 classes.
--- Starting trial: run-0
{'num_units1': 96, 'num_units2': 64, 'dropout': 0.1, 'optimizer': 'adam'}
Epoch 1/10
80/80 [==============================] - 51s 632ms/step - loss: 4.0186 - ac
Epoch 2/10
80/80 [==============================] - 49s 618ms/step - loss: 3.6537 - ac
Epoch 3/10
80/80 [==============================] - 50s 625ms/step - loss: 3.5845 - ac
Epoch 4/10
80/80 [==============================] - 50s 627ms/step - loss: 3.4908 - ac
Epoch 5/10
80/80 [==============================] - 49s 614ms/step - loss: 3.3367 - ac
Epoch 6/10
80/80 [==============================] - 49s 608ms/step - loss: 3.3002 - ac
Epoch 7/10
80/80 [==============================] - 49s 609ms/step - loss: 3.1785 - ac
Epoch 8/10
80/80 [==============================] - 49s 607ms/step - loss: 3.0675 - ac
Epoch 9/10
80/80 [==============================] - 50s 623ms/step - loss: 3.0706 - ac
Epoch 10/10
80/80 [==============================] - 50s 627ms/step - loss: 2.9912 - ac
20/20 [==============================] - 6s 295ms/step - loss: 2.9766 - acc
[2.976576566696167, 0.17301587760448456, 0.5142857432365417]
20/20 [==============================] - 6s 285ms/step - loss: 2.9766 - acc
--- Starting trial: run-1
{'num_units1': 96, 'num_units2': 64, 'dropout': 0.1, 'optimizer': 'sgd'}
Epoch 1/10
80/80 [==============================] - 50s 630ms/step - loss: 3.6143 - ac
Epoch 2/10
80/80 [==============================] - 51s 638ms/step - loss: 3.0925 - ac
Epoch 3/10
80/80 [==============================] - 50s 629ms/step - loss: 2.7200 - ac
Epoch 4/10
80/80 [==============================] - 50s 625ms/step - loss: 2.4918 - ac
Epoch 5/10
80/80 [==============================] - 50s 628ms/step - loss: 2.2647 - ac
Epoch 6/10
80/80 [==============================] - 50s 630ms/step - loss: 2.1491 - ac
Epoch 7/10
80/80 [==============================] - 50s 624ms/step - loss: 1.9777 - ac
```

```
Epoch 8/10
80/80 [==============================] - 49s 617ms/step - loss: 1.8620 - ac
```

```
#Run the below individually to view and understand input shapes
X_train_batch, y_train_batch = next(X_train)

X_train_batch.shape
y_train_batch.shape
X_train_batch[0].shape
X_train_batch[0]
y_train_batch.shape
y_train_batch
features_batch = featurizer.predict(X_train_batch)
features_batch
features_batch[0]
features_batch[0].shape
```

```
(8, 224, 224, 3)
```

```
#Sorting by descending accuracy, we see that best hyperparameters that can be
#All the optimizer sgd models worked better than using adam
%load_ext tensorboard
%tensorboard --logdir logs4/hparam_tuning
```

Note: This picture shows the results from running the above cell. The above is not showing output now as the file was saved only in a temporary storage.

[Click for larger image] (https://drive.google.com/uc?id=1rvR11t28QqWNO5Y-hiaSzKLglU7A37M4)

```
%tensorboard --logdir "/content/drive/My Drive/sample_3000/logs"
```

```
#inception featurizer + kerastunr + tensorboard visualisation setup
#Get a sense of which hyperparameters make training accuracy faster
#!pip install keras-tuner
import kerastuner as kt
from kerastuner import tuners
from tensorboard.plugins.hparams import api as hpp

#HP_NUM_LAYERS = hpp.HParam('num_layers', hpp.Discrete([2,3]))
HP_NUM_UNITS1 = hpp.HParam('num_units1', hpp.Discrete([96, 128])) #CHANGE THI
HP_NUM_UNITS2 = hpp.HParam('num_units2', hpp.Discrete([64, 96])) #CHANGE THIS
HP_DROPOUT = hpp.HParam('dropout', hpp.RealInterval(0.1, 0.3))
```

```
HP_OPTIMIZER = hp.HParam('optimizer', hp.Discrete(['adam', 'sgd']))
METRIC_ACCURACY = 'accuracy'

with tf.summary.create_file_writer('logs5/hparam_tuning').as_default():
  hp.hparams_config(
    hparams=[HP_NUM_UNITS1, HP_NUM_UNITS2, HP_DROPOUT, HP_OPTIMIZER],
    metrics=[hp.Metric(METRIC_ACCURACY)]
  )

image_size = (299,299)
train_datagen = ImageDataGenerator(
      rotation_range=45.0,
      horizontal_flip=True,
      shear_range=30.0,
      zoom_range=0.5,
      preprocessing_function=preprocess_input)

X_train = train_datagen.flow_from_directory('/content/drive/My Drive/sample_3
val_datagen = ImageDataGenerator(preprocessing_function=preprocess_input)
X_val = val_datagen.flow_from_directory('/content/drive/My Drive/sample_3000/
X_val_batch, y_val_batch = next(X_val)

def MyHyperModel(hp):

  featurizer_inception.trainable = False
  model_input_inception = featurizer_inception.input
  x = Flatten()(featurizer_inception.output)
  #for i in range(hparams[HP_NUM_LAYERS]):
  x = Dense(hparams[HP_NUM_UNITS1], activation='relu')(x)
  x = Dense(hparams[HP_NUM_UNITS2], activation='relu')(x)
  x = Dropout(hparams[HP_DROPOUT])(x)
  x = Dense(42, activation='softmax')(x)

  model_inception = Model(model_input_inception, x)
  #model_inception.summary()
  model_inception.compile(loss='categorical_crossentropy',
                          optimizer=hparams[HP_OPTIMIZER],
                          metrics=['accuracy', tf.keras.metrics.TopKCategoric
  ###################################CHANGE HOW MANY EPOCHS FOR EACH MODEL RUN
  model_inception.fit(
    X_train, epochs = 10,
    callbacks=[
        tf.keras.callbacks.TensorBoard(logdir),  # log metrics
        hpp.KerasCallback(logdir, hparams),  # log hparams
    ],
  )

  print(model_inception.evaluate(X_val))
  a, validationaccuracy, d = model_inception.evaluate(X_val)

  return validationaccuracy

```

```
def run(run_dir, hparams):
  with tf.summary.create_file_writer(run_dir).as_default():
    hpp.hparams(hparams)
    validationaccuracy = MyHyperModel(hparams)
    tf.summary.scalar(METRIC_ACCURACY, validationaccuracy, step=1)

"""tuner = kt.Hyperband(MyHyperModel,
                     objective = 'acc',
                     max_epochs = 2,
                     factor = 3,
                     directory = '/content/drive/My Drive/sample_3000',
                     project_name = 'inception_hyper')"""

"""tuner = kt.RandomSearch(
    MyHyperModel,
    objective='val_acc',
    max_trials = 2,
    directory='/content/drive/My Drive/sample_3000',
    project_name='inception_hyper3'
)"""

logdir = os.path.join("/content/drive/My Drive/sample_3000/logs5/metrics", da

#logdir_inception = os.path.join("/content/drive/My Drive/sample_3000/logs/in
#tensorboard_callback_inception = tf.keras.callbacks.TensorBoard(logdir_incep

#tuner.search(X_train, epochs=1, validation_data = X_val, callbacks=[tensorbo

session_num = 0

#for num_layers in HP_NUM_LAYERS.domain.values:
for num_units1 in HP_NUM_UNITS1.domain.values:
  for num_units2 in HP_NUM_UNITS2.domain.values:
    for dropout_rate in (HP_DROPOUT.domain.min_value, HP_DROPOUT.domain.max_v
      for optimizer in HP_OPTIMIZER.domain.values:
        hparams = {
            HP_NUM_UNITS1: num_units1,
            HP_NUM_UNITS2: num_units2,
            HP_DROPOUT: dropout_rate,
            HP_OPTIMIZER: optimizer,
        }
        run_name = "run-%d" % session_num
        print('--- Starting trial: %s' % run_name)
        print({h.name: hparams[h] for h in hparams})
        run('logs5/hparam_tuning/' + run_name, hparams)
        session_num += 1
```

```
Found 2532 images belonging to 42 classes.
Found 630 images belonging to 42 classes.
--- Starting trial: run-0
{'num_units1': 96, 'num_units2': 64, 'dropout': 0.1, 'optimizer': 'adam'}
Epoch 1/10
 2/80 [..............................] - ETA: 41s - loss: 6.0829 - accuracy
```

```
 2/80 [..............................] - ETA: 41s - loss: 6.0829 - accuracy
80/80 [==============================] - 69s 867ms/step - loss: 5.2451 - ac
Epoch 2/10
80/80 [==============================] - 68s 856ms/step - loss: 3.5861 - ac
Epoch 3/10
80/80 [==============================] - 67s 840ms/step - loss: 3.4501 - ac
Epoch 4/10
80/80 [==============================] - 68s 845ms/step - loss: 3.3192 - ac
Epoch 5/10
80/80 [==============================] - 67s 839ms/step - loss: 3.1631 - ac
Epoch 6/10
80/80 [==============================] - 67s 836ms/step - loss: 3.2075 - ac
Epoch 7/10
80/80 [==============================] - 67s 837ms/step - loss: 3.0683 - ac
Epoch 8/10
80/80 [==============================] - 67s 834ms/step - loss: 2.9981 - ac
Epoch 9/10
80/80 [==============================] - 67s 831ms/step - loss: 2.8827 - ac
Epoch 10/10
80/80 [==============================] - 66s 821ms/step - loss: 2.8202 - ac
20/20 [==============================] - 6s 312ms/step - loss: 2.6579 - acc
[2.6578850746154785, 0.2730158865451813, 0.5476190447807312]
20/20 [==============================] - 6s 294ms/step - loss: 2.6579 - acc
--- Starting trial: run-1
{'num_units1': 96, 'num_units2': 64, 'dropout': 0.1, 'optimizer': 'sgd'}
Epoch 1/10
80/80 [==============================] - 66s 831ms/step - loss: 3.8136 - ac
Epoch 2/10
80/80 [==============================] - 66s 830ms/step - loss: 3.4532 - ac
Epoch 3/10
80/80 [==============================] - 66s 827ms/step - loss: 3.0611 - ac
Epoch 4/10
80/80 [==============================] - 66s 828ms/step - loss: 2.7727 - ac
Epoch 5/10
80/80 [==============================] - 67s 833ms/step - loss: 2.4935 - ac
Epoch 6/10
80/80 [==============================] - 67s 835ms/step - loss: 2.4025 - ac
Epoch 7/10
80/80 [==============================] - 66s 831ms/step - loss: 2.1892 - ac
Epoch 8/10
80/80 [==============================] - 66s 831ms/step - loss: 2.1046 - ac
Epoch 9/10
80/80 [==============================] - 67s 833ms/step - loss: 2.0000 - ac
Epoch 10/10
80/80 [==============================] - 68s 847ms/step - loss: 1.8843 - ac
20/20 [==============================] - 6s 302ms/step - loss: 1.9215 - acc
[1.9214684963226318, 0.5111111402511597, 0.7714285850524902]
20/20 [==============================] - 6s 299ms/step - loss: 1.9215 - acc
--- Starting trial: run-2
{'num_units1': 96, 'num_units2': 64, 'dropout': 0.3, 'optimizer': 'adam'}
Epoch 1/10
80/80 [==============================] - 68s 851ms/step - loss: 4.8350 - ac
Epoch 2/10
80/80 [==============================] - 67s 842ms/step - loss: 3.7359 - ac
Epoch 3/10
```

```
#%load_ext tensorboard
%tensorboard --logdir logs5/hparam_tuning
```

Note: This picture shows the results from running the above cell. The above is not showing output now as the file was saved only in a temporary storage.

[Click for larger image] (https://drive.google.com/uc?id=1rvR11t28QqWNO5Y-hiaSzKLglU7A37M4)

```
%tensorboard --logdir "/content/drive/My Drive/sample_3000/logs5"
```

--- 3000 sample train on best model + hyperparameters ---

```
#Based on the results for mobilenet and inception based on the above tensorbo
#We will train a model using INCEPTION: We perform training on more epochs, u
#It did not make it to the full 200 epochs, only up to epoch 140

#adam = tf.keras.optimizers.Adam(learning_rate=1.0)

#Step 1: Load existing saved model or if there is not, create a new model
image_size = (299, 299)
model_file_inception = '/content/drive/My Drive/sample_3000/featurizer_incept
if os.path.isfile(model_file_inception):
  model_inception = load_model(model_file_inception)
else:
  featurizer_inception.trainable = False
  model_input_inception = featurizer_inception.input
  x = Flatten()(featurizer_inception.output)

  x = Dense(128)(x)
  x = LeakyReLU()(x)
  x = Dense(96)(x)
  x = LeakyReLU()(x)
  x = Dropout(0.1)(x)
  x = Dense(42, activation='softmax')(x)

  model_inception = Model(model_input_inception, x)
  model_inception.summary()
  model_inception.compile(loss='categorical_crossentropy', optimizer='sgd', m

#Step 2: ImageDataGenerator and defining X_train, X_val
```

Hyperversion didn't include LeakyReLU ...

```
batch_size = 64
train_datagen = ImageDataGenerator(
    rotation_range=45.0,
    horizontal_flip=True,
    shear_range=30.0,
    zoom_range=0.5,
    preprocessing_function=preprocess_input)
X_train = train_datagen.flow_from_directory('/content/drive/My Drive/sample_3
val_datagen = ImageDataGenerator(preprocessing_function=preprocess_input)
X_val = val_datagen.flow_from_directory('/content/drive/My Drive/sample_3000/
X_val_batch, y_val_batch = next(X_val)

#Step 3: Run the model
mc_inception = ModelCheckpoint('/content/drive/My Drive/sample_3000/featurize
logdir_inception = os.path.join("/content/drive/My Drive/sample_3000/final_lo
tensorboard_callback_inception = tf.keras.callbacks.TensorBoard(logdir_incept
history_inception = model_inception.fit(X_train, epochs=200,
                                        validation_data=(X_val_batch, y_val_b
                                        callbacks=[mc_inception, tensorboard_
```

```
Model: "functional_75"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_4 (InputLayer)         [(None, 299, 299, 3)      0         
_________________________________________________________________
conv2d_94 (Conv2D)           (None, 149, 149, 32)      864       
_________________________________________________________________
batch_normalization_94 (BatchNo (None, 149, 149, 32)      96        
_________________________________________________________________
activation_94 (Activation)   (None, 149, 149, 32)      0         
_________________________________________________________________
conv2d_95 (Conv2D)           (None, 147, 147, 32)      9216      
_________________________________________________________________
batch_normalization_95 (BatchNo (None, 147, 147, 32)      96        
_________________________________________________________________
activation_95 (Activation)   (None, 147, 147, 32)      0         
_________________________________________________________________
conv2d_96 (Conv2D)           (None, 147, 147, 64)      18432     
_________________________________________________________________
batch_normalization_96 (BatchNo (None, 147, 147, 64)      192       
_________________________________________________________________
activation_96 (Activation)   (None, 147, 147, 64)      0         
_________________________________________________________________
max_pooling2d_4 (MaxPooling2D)  (None, 73, 73, 64)       0         
_________________________________________________________________
conv2d_97 (Conv2D)           (None, 73, 73, 80)        5120      
_________________________________________________________________
batch_normalization_97 (BatchNo (None, 73, 73, 80)        240       
_________________________________________________________________
activation_97 (Activation)   (None, 73, 73, 80)        0         
_________________________________________________________________
conv2d_98 (Conv2D)           (None, 71, 71, 192)       138240    
_________________________________________________________________
batch_normalization_98 (BatchNo (None, 71, 71, 192)       576       
```

```
activation_98 (Activation)      (None, 71, 71, 192)  0           batch_norm
max_pooling2d_5 (MaxPooling2D)  (None, 35, 35, 192)  0           activation
conv2d_102 (Conv2D)             (None, 35, 35, 64)   12288       max_poolin
batch_normalization_102 (BatchN (None, 35, 35, 64)   192         conv2d_102
activation_102 (Activation)     (None, 35, 35, 64)   0           batch_norm
conv2d_100 (Conv2D)             (None, 35, 35, 48)   9216        max_poolin
conv2d_103 (Conv2D)             (None, 35, 35, 96)   55296       activation
batch_normalization_100 (BatchN (None, 35, 35, 48)   144         conv2d_100
batch_normalization_103 (BatchN (None, 35, 35, 96)   288         conv2d_103
activation_100 (Activation)     (None, 35, 35, 48)   0           batch_norm
activation_103 (Activation)     (None, 35, 35, 96)   0           batch_norm
average_pooling2d_9 (AveragePoo (None, 35, 35, 192)  0           max_poolin
```

```
#We notice that after the first 15 epochs, there is divergence between the tr
%load_ext tensorboard
%tensorboard --logdir "/content/drive/My Drive/sample_3000/final_logs"
```

```
#Run classification_report on the best model
batch_size = 630
image_size = (299, 299)

val_datagen = ImageDataGenerator(preprocessing_function=preprocess_input)
X_val = val_datagen.flow_from_directory('/content/drive/My Drive/sample_3000/
X_val_batch, y_val_batch = next(X_val)

best_model = load_model('/content/drive/My Drive/sample_3000/featurizer_incep
pred_test = best_model.predict(X_val_batch)
print(classification_report(y_val_batch.argmax(axis=1), pred_test.argmax(axis
```

```
Found 630 images belonging to 42 classes.
              precision    recall  f1-score   support

           0       0.65      0.69      0.67        16
           1       0.54      0.88      0.67        16
           2       0.62      0.50      0.55        16
           3       0.60      0.38      0.46        16
           4       0.76      0.81      0.79        16
           5       0.67      0.88      0.76        16
           6       0.58      0.44      0.50        16
           7       0.50      0.69      0.58        16
           8       0.65      0.69      0.67        16
           9       0.60      0.75      0.67        16
          10       0.64      0.44      0.52        16
          11       1.00      0.55      0.71        11
          12       0.41      0.88      0.56        16
          13       0.82      0.88      0.85        16
          14       0.91      0.62      0.74        16
          15       0.73      0.50      0.59        16
          16       0.78      0.44      0.56        16
          17       0.73      0.89      0.80         9
          18       0.29      0.38      0.33        13
          19       0.57      0.25      0.35        16
          20       0.54      0.44      0.48        16
          21       0.67      0.75      0.71        16
          22       0.79      0.69      0.73        16
          23       0.62      0.33      0.43        15
          24       0.80      0.75      0.77        16
          25       0.92      0.69      0.79        16
          26       0.31      0.69      0.42        16
          27       0.48      0.81      0.60        16
          28       0.70      0.93      0.80        15
          29       0.80      0.62      0.70        13
          30       0.71      0.94      0.81        16
          31       0.35      0.75      0.48        16
          32       1.00      0.23      0.38        13
          33       1.00      0.33      0.50         3
          34       0.58      0.88      0.70        16
          35       0.88      0.44      0.58        16
          36       0.25      0.12      0.17        16
          37       0.75      0.60      0.67        10
          38       0.80      0.25      0.38        16
          39       0.38      0.50      0.43        16
          40       0.75      0.38      0.50        16
          41       0.43      0.19      0.26        16

    accuracy                           0.60       630
   macro avg       0.66      0.59      0.59       630
weighted avg       0.64      0.60      0.59       630
```

--- Finetuning featurizer ---

```
1 #Finetuning - Inspect which layers to finetune
```

```
featurizer_inception = InceptionV3(input_shape = (299, 299, 3), include_top =
featurizer_inception.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-appli
87916544/87910968 [==============================] - 2s 0us/step
Model: "inception_v3"
```

Layer (type)	Output Shape	Param #	Connected
input_1 (InputLayer)	[(None, 299, 299, 3)	0	
conv2d (Conv2D)	(None, 149, 149, 32)	864	input_1[0]
batch_normalization (BatchNorma	(None, 149, 149, 32)	96	conv2d[0][
activation (Activation)	(None, 149, 149, 32)	0	batch_norm
conv2d_1 (Conv2D)	(None, 147, 147, 32)	9216	activation
batch_normalization_1 (BatchNor	(None, 147, 147, 32)	96	conv2d_1[0
activation_1 (Activation)	(None, 147, 147, 32)	0	batch_norm
conv2d_2 (Conv2D)	(None, 147, 147, 64)	18432	activation
batch_normalization_2 (BatchNor	(None, 147, 147, 64)	192	conv2d_2[0
activation_2 (Activation)	(None, 147, 147, 64)	0	batch_norm
max_pooling2d (MaxPooling2D)	(None, 73, 73, 64)	0	activation
conv2d_3 (Conv2D)	(None, 73, 73, 80)	5120	max_poolin
batch_normalization_3 (BatchNor	(None, 73, 73, 80)	240	conv2d_3[0
activation_3 (Activation)	(None, 73, 73, 80)	0	batch_norm
conv2d_4 (Conv2D)	(None, 71, 71, 192)	138240	activation
batch_normalization_4 (BatchNor	(None, 71, 71, 192)	576	conv2d_4[0
activation_4 (Activation)	(None, 71, 71, 192)	0	batch_norm
max_pooling2d_1 (MaxPooling2D)	(None, 35, 35, 192)	0	activation
conv2d_8 (Conv2D)	(None, 35, 35, 64)	12288	max_poolin
batch_normalization_8 (BatchNor	(None, 35, 35, 64)	192	conv2d_8[0
activation_8 (Activation)	(None, 35, 35, 64)	0	batch_norm
conv2d_6 (Conv2D)	(None, 35, 35, 48)	9216	max_poolin
conv2d_9 (Conv2D)	(None, 35, 35, 96)	55296	activation
batch_normalization_6 (BatchNor	(None, 35, 35, 48)	144	conv2d_6[0
batch_normalization_9 (BatchNor	(None, 35, 35, 96)	288	conv2d_9[0

```
batch_normalization_9 (BatchNor (None, 35, 35, 96)   288         conv2d_9[0
__________________________________________________________________________
activation_6 (Activation)       (None, 35, 35, 48)   0           batch_norm
__________________________________________________________________________
activation_9 (Activation)       (None, 35, 35, 96)   0           batch_norm
```

```
#Unfreeze all models after "mixed6"
featurizer_inception.trainable = False
unfreeze = False
for l in featurizer_inception.layers:
  print(l.name + ' unfrozen: ' + str(unfreeze))
  if unfreeze:
    l.trainable = True
  if l.name == 'mixed6':
    unfreeze = True
```

```
input_1 unfrozen: False
conv2d unfrozen: False
batch_normalization unfrozen: False
activation unfrozen: False
conv2d_1 unfrozen: False
batch_normalization_1 unfrozen: False
activation_1 unfrozen: False
conv2d_2 unfrozen: False
batch_normalization_2 unfrozen: False
activation_2 unfrozen: False
max_pooling2d unfrozen: False
conv2d_3 unfrozen: False
batch_normalization_3 unfrozen: False
activation_3 unfrozen: False
conv2d_4 unfrozen: False
batch_normalization_4 unfrozen: False
activation_4 unfrozen: False
max_pooling2d_1 unfrozen: False
conv2d_8 unfrozen: False
batch_normalization_8 unfrozen: False
activation_8 unfrozen: False
conv2d_6 unfrozen: False
conv2d_9 unfrozen: False
batch_normalization_6 unfrozen: False
batch_normalization_9 unfrozen: False
activation_6 unfrozen: False
activation_9 unfrozen: False
average_pooling2d unfrozen: False
conv2d_5 unfrozen: False
conv2d_7 unfrozen: False
conv2d_10 unfrozen: False
conv2d_11 unfrozen: False
batch_normalization_5 unfrozen: False
batch_normalization_7 unfrozen: False
batch_normalization_10 unfrozen: False
batch_normalization_11 unfrozen: False
activation_5 unfrozen: False
activation_7 unfrozen: False
activation_10 unfrozen: False
activation_11 unfrozen: False
```

```
mixed0 unfrozen: False
conv2d_15 unfrozen: False
batch_normalization_15 unfrozen: False
activation_15 unfrozen: False
conv2d_13 unfrozen: False
conv2d_16 unfrozen: False
batch_normalization_13 unfrozen: False
batch_normalization_16 unfrozen: False
activation_13 unfrozen: False
activation_16 unfrozen: False
average_pooling2d_1 unfrozen: False
conv2d_12 unfrozen: False
conv2d_14 unfrozen: False
conv2d_17 unfrozen: False
conv2d_18 unfrozen: False
batch_normalization_12 unfrozen: False
batch_normalization_14 unfrozen: False
batch_normalization_17 unfrozen: False
batch_normalization_18 unfrozen: False
activation_12 unfrozen: False
```

```
#Finetuning for 10 epochs at alow learning rate
#Somehow, the tensorboard command produced an error, so we will comment it ou
from tensorflow.keras.optimizers import SGD

model_file_inception = '/content/drive/My Drive/sample_3000/featurizer_incept
model_inception = load_model(model_file_inception)

unfreeze = False
for l in featurizer_inception.layers:
  if unfreeze:
    l.trainable = True
  if l.name == 'mixed6':
    unfreeze = True

"""model_input_inception = featurizer_inception.input
x = Flatten()(featurizer_inception.output)

x = Dense(128)(x)
x = LeakyReLU()(x)
x = Dense(96)(x)
x = LeakyReLU()(x)
x = Dropout(0.1)(x)
x = Dense(42, activation='softmax')(x)

model_inception = Model(model_input_inception, x)
model_inception.compile(loss='categorical_crossentropy', optimizer=SGD(learni

batch_size = 64
image_size = (299, 299)
train_datagen = ImageDataGenerator(
    rotation_range=45.0,
    horizontal_flip=True,
    shear_range=30.0,
```

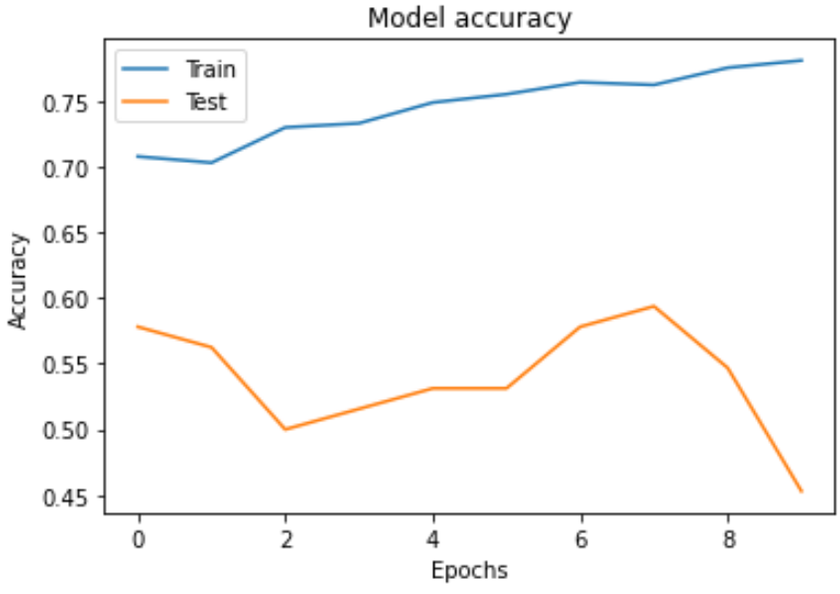
```
    zoom_range=0.5,
    preprocessing_function=preprocess_input)
X_train = train_datagen.flow_from_directory('/content/drive/My Drive/sample_3
val_datagen = ImageDataGenerator(preprocessing_function=preprocess_input)
X_val = val_datagen.flow_from_directory('/content/drive/My Drive/sample_3000/
X_val_batch, y_val_batch = next(X_val)

mc = ModelCheckpoint('/content/drive/My Drive/sample_3000/featurizer_inceptio
#logdir_inception = os.path.join("/content/drive/My Drive/sample_3000/final_l
#tensorboard_callback_inception = tf.keras.callbacks.TensorBoard(logdir_incep
#,tensorboard_callback_inception
history = model_inception.fit(X_train, epochs=10,
                    validation_data=(X_val_batch, y_val_batch),
                    callbacks=[mc])
```

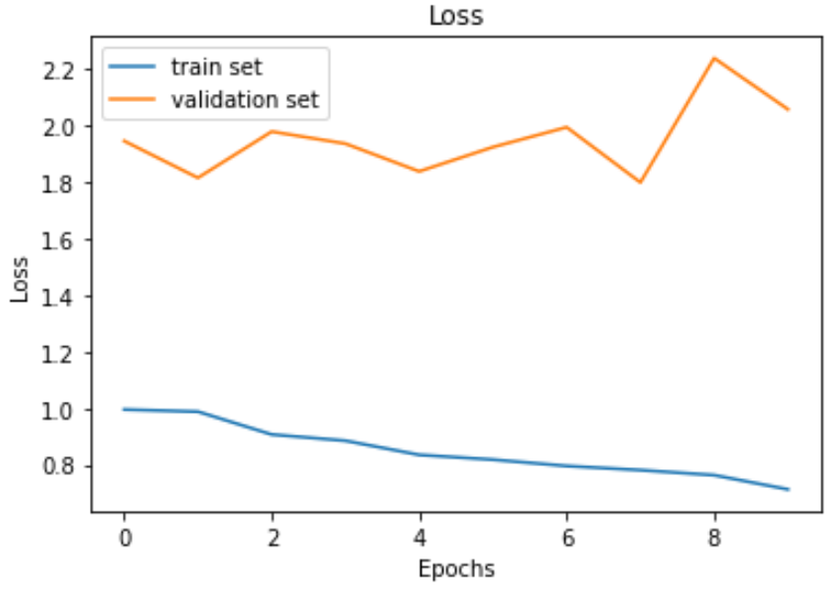
```
Found 2532 images belonging to 42 classes.
Found 630 images belonging to 42 classes.
Epoch 1/10
40/40 [==============================] - ETA: 0s - loss: 0.9954 - acc: 0.70
40/40 [==============================] - 423s 11s/step - loss: 0.9954 - acc
Epoch 2/10
40/40 [==============================] - 64s 2s/step - loss: 0.9879 - acc:
Epoch 3/10
40/40 [==============================] - 64s 2s/step - loss: 0.9070 - acc:
Epoch 4/10
40/40 [==============================] - 64s 2s/step - loss: 0.8851 - acc:
Epoch 5/10
40/40 [==============================] - 62s 2s/step - loss: 0.8351 - acc:
Epoch 6/10
40/40 [==============================] - 62s 2s/step - loss: 0.8182 - acc:
Epoch 7/10
40/40 [==============================] - 64s 2s/step - loss: 0.7958 - acc:
Epoch 8/10
40/40 [==============================] - 66s 2s/step - loss: 0.7811 - acc:
Epoch 9/10
40/40 [==============================] - 63s 2s/step - loss: 0.7633 - acc:
Epoch 10/10
40/40 [==============================] - 62s 2s/step - loss: 0.7131 - acc:
```

```
#The below can be run if there were no time-out that occurred. But we can see
plt.plot(history.history['acc'], label='train set')
plt.plot(history.history['val_acc'], label='validation set')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Model accuracy')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()

plt.plot(history.history['loss'], label='train set')
plt.plot(history.history['val_loss'], label='validation set')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Loss')
plt.legend()
plt.show()
```



Starting to overfit



--- Global dataset ---

```
#So far, we have run on the dataset of 3000 images
#Now we run on the global dataset of 100K images
#Split global dataset into train and validation sets - Creating CSV file to n
df_traincsv_marker = df_traincsv
df_traincsv_marker['selected_for_validation'] = 0
df_traincsv_marker.head()
```

	filename	category	selected_for_validation
0	45e2d0c97f7bdf8cbf3594beb6fdcda0.jpg	3	0
1	f74d1a5fc2498bbbfa045c74e3cc333e.jpg	3	0
2	f6c172096818c5fab10ecae722840798.jpg	3	0
3	251ffd610399ac00fea7709c642676ee.jpg	3	0
4	73c7328b8eda399199fdedec6e4badaf.jpg	3	0

```
#DO NOT RUN THIS CELL AFTER FIRST RUN
rolling = 20
for two_digit in folder_names[20:]:
  path = "/content/drive/My Drive/validation/" + two_digit
  os.mkdir(path)

  temp_df = df_traincsv_marker.loc[df_traincsv_marker.category==rolling,:].sa
  temp_index = temp_df.index
  df_traincsv_marker.loc[temp_index, 'selected_for_validation'] = 1
  temp_df.reset_index(inplace=True)

  for i in range(len(temp_df)):
    f = temp_df.loc[i,'filename']
    shutil.move("/content/drive/My Drive/train/train/train/" + two_digit + '/

  rolling+=1
```

```
featurizer_mobilenet = MobileNetV2(input_shape=(224, 224, 3), include_top=Fal
featurizer_inception = InceptionV3(input_shape = (299, 299, 3), include_top =
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-appli
9412608/9406464 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/keras-appli
87916544/87910968 [==============================] - 2s 0us/step
```

```
#As the global dataset is large, we expect that the colab session will run ou
#We create a function that can re-used after the colab session runs out
#RUN THIS CELL AFTER COLAB 6-HOUR RUNTIME
#optimizer=SGD(learning_rate=0.9)
adam = tf.keras.optimizers.Adam(learning_rate=1.0)

```

consider making a function with parameter for dataset size

Sample set

You can also from larger and fine tune

```
def load_and_run_100k(the_model):
  #Step 1: Load existing saved model or if there is not, create a new model
  if the_model == 'mobilenet':
    image_size = (224, 224)
    #model_file_mobilenet_savebest = '/content/drive/My Drive/featurizer_mobi
    #model_file_mobilenet_savefreq = '/content/drive/My Drive/featurizer_mobi
    model_file_mobilenet_savefreq = '/content/drive/My Drive/featurizer_mobil

    if os.path.isfile(model_file_mobilenet_savefreq):
      model_mobilenet = load_model(model_file_mobilenet_savefreq)

    else:
      featurizer_mobilenet.trainable = False
      model_input_mobilenet = featurizer_mobilenet.input
      x = Flatten()(featurizer_mobilenet.output)

      x = Dense(128)(x)
      x = LeakyReLU()(x)
      x = Dense(64)(x)
      x = LeakyReLU()(x)
      x = Dense(42, activation='softmax')(x)

      model_mobilenet = Model(model_input_mobilenet, x)
      model_mobilenet.summary()
      model_mobilenet.compile(loss='categorical_crossentropy', optimizer=adam


  elif the_model == 'inception':
    image_size = (299, 299)
    #model_file_inception_savebest = '/content/drive/My Drive/featurizer_ince
    model_file_inception_savefreq = '/content/drive/My Drive/featurizer_incep

    if os.path.isfile(model_file_inception_savefreq):
      print('yes')
      model_inception = load_model(model_file_inception_savefreq)
    else:
      featurizer_inception.trainable = False
      model_input_inception = featurizer_inception.input
      x = Flatten()(featurizer_inception.output)

      x = Dense(128)(x)
      x = LeakyReLU()(x)
      x = Dense(64)(x)
      x = LeakyReLU()(x)
      x = Dense(42, activation='softmax')(x)

      model_inception = Model(model_input_inception, x)
      model_inception.summary()
      model_inception.compile(loss='categorical_crossentropy', optimizer=adam


  #Step 2: ImageDataGenerator and defining X_train, X_val
```

```
batch_size = 64

train_datagen = ImageDataGenerator(
    rotation_range=45.0,
    horizontal_flip=True,
    shear_range=30.0,
    zoom_range=0.5,
    preprocessing_function=preprocess_input)
X_train = train_datagen.flow_from_directory('/content/drive/My Drive/train/

val_datagen = ImageDataGenerator(preprocessing_function=preprocess_input)
X_val = val_datagen.flow_from_directory('/content/drive/My Drive/validation
X_val_batch, y_val_batch = next(X_val)

#Step 3: Run the model
if the_model == 'mobilenet':
  mc_mobilenet_savebest = ModelCheckpoint('/content/drive/My Drive/featuriz
  mc_mobilenet_savefreq = ModelCheckpoint('/content/drive/My Drive/featuriz
  logdir_mobilenet = os.path.join("/content/drive/My Drive/logs/mobilenet",
  tensorboard_callback_mobilenet = tf.keras.callbacks.TensorBoard(logdir_mo
  history_mobilenet = model_mobilenet.fit(X_train, epochs=10,
                  validation_data=(X_val_batch, y_val_batch),
                  callbacks=[mc_mobilenet_savebest, mc_mobilenet_savefreq,
  file_history_mobilenet = open('/content/drive/My Drive/file_history_mobil
  pickle.dump(history_mobilenet, file_history_mobilenet)
  file_history_mobilenet.close()

elif the_model == 'inception':
  mc_inception_savebest = ModelCheckpoint('/content/drive/My Drive/featuriz
  mc_inception_savefreq = ModelCheckpoint('/content/drive/My Drive/featuriz
  logdir_inception = os.path.join("/content/drive/My Drive/logs/inception",
  tensorboard_callback_inception = tf.keras.callbacks.TensorBoard(logdir_in
  history_inception = model_inception.fit(X_train, epochs=10,
                  validation_data=(X_val_batch, y_val_batch),
                  callbacks=[mc_inception_savebest, mc_inception_savefreq,
  file_history_inception = open('/content/drive/My Drive/file_history_incep
  pickle.dump(history_inception, file_history_inception)
  file_history_inception.close()
```

```
load_and_run_100k("mobilenet")
```

```
yes
Found 89640 images belonging to 42 classes.
Found 15742 images belonging to 42 classes.
Epoch 1/100
   1/1401 [..............................] - ETA: 0s - loss: 313250.5625 -
Instructions for updating:
use `tf.profiler.experimental.stop` instead.
 377/1401 [=======>......................] - ETA: 17:11:14 - loss: 290402.4
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-5-e938c1895d8d> in <module>()
----> 1 load_and_run_100k("mobilenet")

                                 9 frames
/usr/local/lib/python3.6/dist-packages/tensorflow/python/eager/execute.py
in quick_execute(op_name, num_outputs, inputs, attrs, ctx, name)
     58     ctx.ensure_initialized()
     59     tensors = pywrap_tfe.TFE_Py_Execute(ctx._handle, device_name,
op_name,
---> 60                                         inputs, attrs, num_outputs)
     61   except core._NotOkStatusException as e:
     62     if name is not None:

KeyboardInterrupt:
```

```
%reload_ext tensorboard
%tensorboard --logdir "/content/drive/My Drive/logs"
```

```
#As the dataset is large and during training it did not train through 1 epoch
#This cell gets the validation accuracy if you specify which saved .h5 model
#We notice that on some classes, the model was not able to get any prediction
#And we note that the model did not even see one full epoch, much less get tr
#'/content/drive/My Drive/featurizer_mobilenet_savefreq20200929-113939-291276
#'/content/drive/My Drive/featurizer_inception_savefreq20200929-231315-323220
val_datagen = ImageDataGenerator(preprocessing_function=preprocess_input)
batch_size = int(input('Batch size: '))
if input('Type m for mobilenet: ') == "m":
  image_size = (224, 224)
else:
  image_size = (299, 299)
X_val = val_datagen.flow_from_directory('/content/drive/My Drive/validation',
X_val_batch, y_val_batch = next(X_val)
loaded_model = input('Trained model file path: ')
try_model = load_model(loaded_model)
pred_test = try_model.predict(X_val_batch)
print(classification_report(y_val_batch.argmax(axis=1), pred_test.argmax(axis
```

```
Batch size: 256
Type m for mobilenet: m
Found 15742 images belonging to 42 classes.
Trained model file path: /content/drive/My Drive/featurizer_mobilenet_savef
/usr/local/lib/python3.6/dist-packages/sklearn/metrics/_classification.py:1
  _warn_prf(average, modifier, msg_start, len(result))
```

	precision	recall	f1-score	support
0	0.00	0.00	0.00	10
1	0.54	0.88	0.67	8
2	0.40	0.22	0.29	9
3	0.25	0.12	0.17	8
4	1.00	0.83	0.91	6
5	0.50	0.43	0.46	7
6	0.67	0.25	0.36	8
7	0.75	0.60	0.67	5
8	0.25	0.33	0.29	6
9	0.00	0.00	0.00	7
10	0.40	0.57	0.47	7
11	0.00	0.00	0.00	2
12	0.15	0.60	0.24	5
13	0.50	0.86	0.63	7
14	0.67	0.67	0.67	6
15	0.50	0.43	0.46	7
16	1.00	0.20	0.33	10
17	1.00	0.12	0.22	8
18	0.00	0.00	0.00	2
19	0.33	0.50	0.40	4
20	0.00	0.00	0.00	5
21	0.00	0.00	0.00	3
22	0.40	0.67	0.50	3
23	0.38	0.60	0.46	5
24	0.57	0.57	0.57	7
25	0.50	0.50	0.50	10
26	0.00	0.00	0.00	5
27	0.00	0.00	0.00	10
28	0.67	0.67	0.67	6
29	1.00	0.67	0.80	6
30	0.50	0.38	0.43	8
31	0.00	0.00	0.00	8
32	0.40	0.57	0.47	7
33	0.12	0.50	0.20	2
34	0.25	0.67	0.36	6
35	1.00	0.54	0.70	13
36	0.18	0.67	0.29	3
37	0.25	0.25	0.25	4
38	0.00	0.00	0.00	5
39	1.00	0.33	0.50	3
40	0.00	0.00	0.00	1
41	0.00	0.00	0.00	4
accuracy			0.37	256
macro avg	0.38	0.36	0.33	256
weighted avg	0.44	0.37	0.36	256

--- Trying model trained on 3000 samples on global validation set ---

```
#We try using the best model trained using the sample of 3000, using it to pr
#The results was surprisingly good, achieving an accuracy of 0.61

#/content/drive/My Drive/sample_3000/featurizer_inception_finetune.h5
val_datagen = ImageDataGenerator(preprocessing_function=preprocess_input)
batch_size = int(input('Batch size: '))
if input('Type m for mobilenet: ') == "m":
  image_size = (224, 224)
else:
  image_size = (299, 299)
X_val = val_datagen.flow_from_directory('/content/drive/My Drive/validation',
X_val_batch, y_val_batch = next(X_val)
loaded_model = input('Trained model file path: ')
try_model = load_model(loaded_model)
pred_test = try_model.predict(X_val_batch)
print(classification_report(y_val_batch.argmax(axis=1), pred_test.argmax(axis
```

```
Batch size: 256
Type m for mobilenet: i
Found 15742 images belonging to 42 classes.
Trained model file path: /content/drive/My Drive/sample_3000/featurizer_inc
/usr/local/lib/python3.6/dist-packages/sklearn/metrics/_classification.py:1
  _warn_prf(average, modifier, msg_start, len(result))
              precision    recall  f1-score   support

           0       0.50      0.33      0.40         9
           1       0.70      0.70      0.70        10
           2       0.50      0.33      0.40         6
           3       0.54      0.64      0.58        11
           4       1.00      0.83      0.91         6
           5       0.67      0.67      0.67         6
           6       0.67      0.57      0.62         7
           7       0.86      0.67      0.75         9
           8       1.00      0.62      0.77         8
           9       0.50      0.67      0.57         3
          10       0.45      0.83      0.59         6
          11       1.00      0.50      0.67         4
          12       0.75      0.60      0.67        10
          13       0.86      1.00      0.92         6
          14       0.86      0.86      0.86         7
          15       0.86      0.86      0.86         7
          16       0.67      0.80      0.73         5
          17       0.67      0.50      0.57         4
          18       0.50      0.60      0.55         5
          19       0.33      0.67      0.44         3
          20       0.33      0.40      0.36         5
          21       0.80      0.80      0.80         5
          22       1.00      0.40      0.57         5
          23       0.50      0.80      0.62         5
          24       0.50      0.67      0.57         3
          25       0.71      0.83      0.77         6
          26       0.40      0.33      0.36         6
          27       0.67      0.80      0.73         5
          28       1.00      0.71      0.83         7
          29       0.71      1.00      0.83         5
          30       0.90      1.00      0.95         9
          31       0.50      0.25      0.33         4
          32       0.67      0.67      0.67         3
          33       0.00      0.00      0.00         0
          34       0.33      0.50      0.40         8
          35       0.67      0.29      0.40         7
          36       0.57      0.25      0.35        16
          37       0.00      0.00      0.00         3
          38       0.22      0.50      0.31         4
          39       0.60      0.75      0.67         4
          40       0.43      0.60      0.50         5
          41       0.25      0.33      0.29         9

    accuracy                           0.61       256
   macro avg       0.61      0.60      0.58       256
weighted avg       0.64      0.61      0.60       256
```

Section 6: Evaluate Metrics

The cells above have a mixture of Section 5 and 6, as there are a few rounds of model creation and performance evaluation. Please see the above comments in green for more comments

Preliminary testing of hyperparameters

- Worst combination (accuracy 0.025397): Inception + 96 units + 96 units + adam + dropout(0.3)
- Best combination (accuracy 0.53125): Inception + 128 units + 96 units + sgd + dropout(0.1)

General evaluation of metrics:

- Train on 3000 sample + Inception featurizer (trainable = False) + best hyperparameters = **0.60** accuracy (on 630 images from 3000 sample validation set)
- Train on global dataset (less than 1 epoch) + Inception featurizer (trainable = False) + hyperparameters (128 units + 64 units + leaky relu + adam) = **0.37** accuracy (on 256 images from global validation set)
- Train on 3000 sample + Inception featurizer (trainable = True from mixed 7 onwards) + best hyperparameters = **0.61** accuracy (on 256 images from global validation set)

Good attempt and summary!
Some suggestions to consider
- Choose 1 version of HyperModel and make it reflect your actual model (e.g LeakyRelu...)
- Make the dataset logic into a function so you can easily change sizes without rewriting.
- If dataset too large to load at once, try sampling it many times & update model progressively.
Use code to throw out what is

Section 7: Observations and analysis

1. Conclusion from metrics

- As there are 42 categories for this problem / dataset, we note that an accuracy of 0.0238 (1/42) is a random guess on which category an image belongs to. To have an accruracy of 0.61 could be considered to be pretty good in this context.
- We note also that the images in the dataset are not all clean - There are some pictures that are not taken too well, and this simulates what an actual image taken by a person might be like. Per the description from kaggle, there are also some mislabelled images.
- There is some degree of imbalance in the dataset, but not wildly so. So in the sampling of data, we have ignored this degree of imbalance.
- The Tensorboard hyperparameter search has been extremely useful in giving us a rough guage on which hyperparameters would do well on our dataset. For example, all the models that had SGD as the optimizer performed better than those using adam. We can then be more confident in selecting hyperparameters to perform longer training on (more epochs).

2. Proposed improvements

- We had decided to unfreeze inceptionv3 featurizer layers from mixed7 onwards. More tests could be run to determine the results of gradually unfreezing layers infinetuning.
- We had run training and validation on less than one epoch of the global dataset, due to shortage of time. The ideal scenario is to perform training and validation on perhaps 15 or so epochs of the global dataset.
- The model trained on 3000 samples (inceptionv3, best hyperparameters, finetuning featurizer layers from from mixed7 onwards) did surprisingly well on a batch of 256 on the global validation set. This shows, we can probably train on a bigger model of about 20,000 samples using the same methodology and probably get better results. We may not need to run training on the entire global training images we had defined, and still obtain decent results.

```
1
```