In [ ]:

In [ ]:

Image Classification

"Atmin Sheth" and Abigail Solomon

1. Import Libraries

In [1]:

```
import pandas as pd
import numpy as np
import cv2
import seaborn as sns
import matplotlib.pyplot as plt
import keras
import os
import tensorflow as tf
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras import layers
from keras.layers import Conv2D, MaxPooling2D
from keras.layers import Activation, Dropout, Flatten, Dense
from keras.callbacks import EarlyStopping
from tensorflow.keras.optimizers import RMSprop
from sklearn.metrics import accuracy_score
```

2. Import dataset

In [2]:

```
# Use google.colab to directly import dataset from Kaggle to colab with out do
wnloading to pc.
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

In [3]:

```
# Directory to dataset in drive
os.environ['KAGGLE_CONFIG_DIR'] = "/content/gdrive/My Drive/Kaggle"
```

In [4]:

```
# Change directory
%cd /content/gdrive/My Drive/Kaggle
```

```
/content/gdrive/My Drive/Kaggle
```

In [5]:

```
# Kaggle Dataset API
!kaggle datasets download -d hasibalmuzdadid/shoe-vs-sandal-vs-boot-dataset-15k-images
```

```
shoe-vs-sandal-vs-boot-dataset-15k-images.zip: Skipping, found more recently
modified local copy (use --force to force download)
```

In [ ]:

```
#Unzip folder and remove the zipped one
!unzip \*.zip  && rm *.zip
```

3. Data Exploration

In [6]:

```
from PIL import Image
rootDIR = "/content/gdrive/MyDrive/Kaggle/Shoe vs Sandal vs Boot Dataset"
ImageDIR =  os.path.join(rootDIR)
ImageNames = os.listdir(ImageDIR)
TargetImageCount = len(ImageNames)
print('There are',TargetImageCount, 'image targets, and their names are the following:\n',ImageNames)
```

```
There are 3 image targets, and their names are the following:
 ['Boot', 'Sandal', 'Shoe']
```

In [7]:

```
BootDIR = "/content/gdrive/MyDrive/Kaggle/Shoe vs Sandal vs Boot Dataset/Boot"
ImageDIR =  os.path.join(BootDIR)
ImageFiles = os.listdir(ImageDIR)
ImageCount = len(ImageFiles)
print('There are',ImageCount, 'Boot images ')
```

```
There are 5000 Boot images
```

In [ ]:

```
SandalDIR = "/content/gdrive/MyDrive/Kaggle/Shoe vs Sandal vs Boot Dataset/Sandal"
ImageDIR =  os.path.join(SandalDIR)
ImageFiles = os.listdir(ImageDIR)
ImageCount = len(ImageFiles)
print('There are',ImageCount, 'Sandal images ')
```

```
There are 5000 Sandal images
```

In [8]:

```
ShoeDIR = "/content/gdrive/MyDrive/Kaggle/Shoe vs Sandal vs Boot Dataset/Shoe"
ImageDIR =  os.path.join(ShoeDIR)
ImageFiles = os.listdir(ImageDIR)
ImageCount = len(ImageFiles)
print('There are',ImageCount, 'Shoe images ')
```

```
There are 5000 Shoe images
```

In [9]:

```
import tensorflow as tf
from tensorflow import keras

image_size = 256
batch_size = 10
epochs = 10

target_size = (image_size, image_size)
input_shape = (image_size, image_size, 3)
```

In [10]:

```
# Loading sample images to visualize data

img = cv2.imread('/content/gdrive/MyDrive/Kaggle/Shoe vs Sandal vs Boot Dataset/Boot/boot (100).jpg')
# converting to the RGB ordering that matplotlib wants
img_show1 = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

fig, ax = plt.subplots(1,2, figsize=(12,6))
ax[0].imshow(img) # RGB order from imread()
ax[0].axis('off')
ax[1].imshow(img_show1) # expected color to the image
ax[1].axis('off')
plt.show()

img2 = cv2.imread('/content/gdrive/MyDrive/Kaggle/Shoe vs Sandal vs Boot Dataset/Sandal/Sandal (1003).jpg')
# converting to the RGB ordering that matplotlib wants
img_show2 = cv2.cvtColor(img2, cv2.COLOR_BGR2RGB)

fig, ax = plt.subplots(1,2, figsize=(12,6))
ax[0].imshow(img2) # RGB order from imread()
ax[0].axis('off')
ax[1].imshow(img_show2) # expected color to the image
ax[1].axis('off')
plt.show()

img3 = cv2.imread('/content/gdrive/MyDrive/Kaggle/Shoe vs Sandal vs Boot Dataset/Shoe/Shoe (1004).jpg')
# converting to the RGB ordering that matplotlib wants
img_show3 = cv2.cvtColor(img3, cv2.COLOR_BGR2RGB)

fig, ax = plt.subplots(1,2, figsize=(12,6))
ax[0].imshow(img3) # RGB order from imread()
ax[0].axis('off')
ax[1].imshow(img_show3) # expected color to the image
ax[1].axis('off')
plt.show()
```

4. Split data into train and test

In [11]:

```
# By setting validation_split to 0.2, 20% of the original data set will be generated for test and the remaining for train
# Training in TensorFlow using ImageDataGenerator, will automatically label images based on their parent directory.

datagen = ImageDataGenerator(rescale=1./255,rotation_range=45, horizontal_flip=True,
                             vertical_flip=True, fill_mode='reflect',validation_split=.20)

train = datagen.flow_from_directory(rootDIR, batch_size = 10,target_size=(256,256),
        classes = [ 'Boot','Sandal', 'Shoe'],
        class_mode='categorical', subset="training")

test = datagen.flow_from_directory(rootDIR, batch_size = 10,target_size=(256,256),
        classes = [ 'Boot','Sandal', 'Shoe'],
        class_mode='categorical', subset="validation")
```

```
Found 12000 images belonging to 3 classes.
Found 3000 images belonging to 3 classes.
```

In [12]:

```
# The Classes are represented in numbers, since computers perform better with numbers, and also to avoid language bias
print (train.class_indices)
```

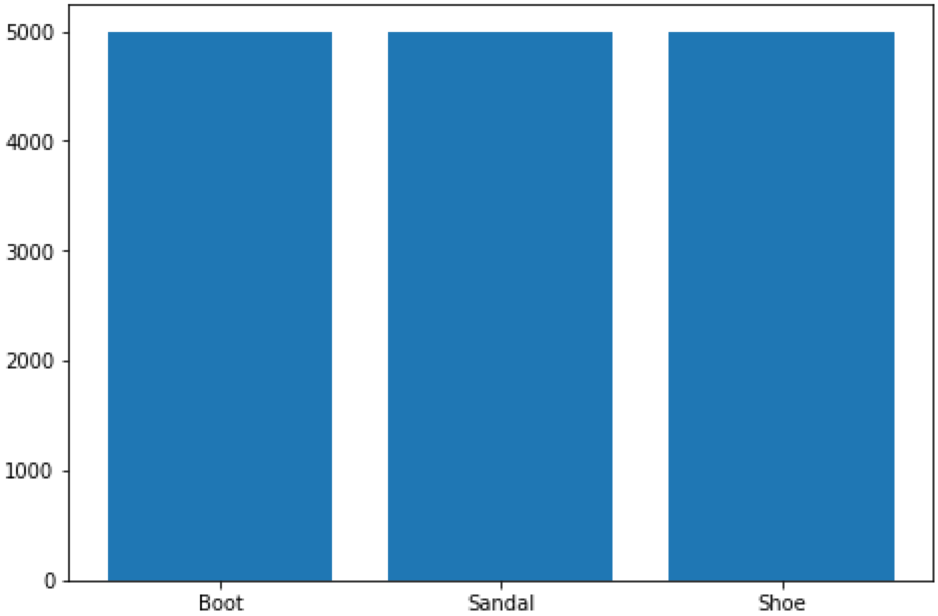
```
{'Boot': 0, 'Sandal': 1, 'Shoe': 2}
```

5. Bar graph

The dataset has 15,000 observations, and the tree target classes has 5000 observations each, the bar graph is displaying the distibution of the target classes visually.

In [13]:

```
import matplotlib.pyplot as plt
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
ShoeImages = ['Boot', 'Sandal', 'Shoe']
ImageCount = [5000,5000,5000]
ax.bar(ShoeImages,ImageCount)
plt.show()
```



6. Describtion of the dataset

The dataset has a total of 15,000 images, labeled into three subclasses, named Boot,Sanadal,and Shoe. Just like a human being, able to recognize anything after observing or using something manytimes, a machine can be trained to detect an image by feeding large number of data to observe. Now, we will train a neural network with 80% of the data to learn whether the given image is Boot, Sandal, or Shoe, by first detecting the patterns in raw pixels to classify images, and then detect features using convolutions trained to spot particular feature that make up a shoe.Then the trained model would be able to predict on the rest 20% of the data, it will classify the new data into Boot,Sandal or Shoe class in the same pattern.

In [14]:

```
#Avoid overfit by making the process to terminate early
callback = keras.callbacks.EarlyStopping(monitor='accuracy', patience=10)
```

7. Create sequential model

In [15]:

```
# Image recognition by processing pixel data
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(256, 256)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.2),
    # Three output neurons for classes
    tf.keras.layers.Dense(3, activation='softmax')
])
```

In [16]:

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 flatten (Flatten)           (None, 65536)             0         
                                                                 
 dense (Dense)               (None, 128)               8388736   
                                                                 
 dropout (Dropout)           (None, 128)               0         
                                                                 
 dense_1 (Dense)             (None, 128)               16512     
                                                                 
 dropout_1 (Dropout)         (None, 128)               0         
                                                                 
 dense_2 (Dense)             (None, 3)                 387       
                                                                 
=================================================================
Total params: 8,405,635
Trainable params: 8,405,635
Non-trainable params: 0
_________________________________________________________________
```

7. Evaluate on the test data

Compile and fit.

In [ ]:

```
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

In [22]:

```
total_sample = train.n
n_epochs = 10
```

In [23]:

```
history = model.fit(
    x= test,
    batch_size=10,
    steps_per_epoch = int(total_sample/batch_size),
    epochs=n_epochs,
    verbose=1
)
```

```
Epoch 1/10

WARNING:tensorflow:Model was constructed with shape (None, 256, 256) for inpu
t KerasTensor(type_spec=TensorSpec(shape=(None, 256, 256), dtype=tf.float32,
name='flatten_input'), name='flatten_input', description="created by layer 'f
latten_input'"), but it was called on an input with incompatible shape (None,
None, None, None).
WARNING:tensorflow:Model was constructed with shape (None, 256, 256) for inpu
t KerasTensor(type_spec=TensorSpec(shape=(None, 256, 256), dtype=tf.float32,
name='flatten_input'), name='flatten_input', description="created by layer 'f
latten_input'"), but it was called on an input with incompatible shape (None,
None, None, None).
```

```
---------------------------------------------------------------------------
InvalidArgumentError                      Traceback (most recent call last)
<ipython-input-23-751ede4a5913> in <module>
----> 1 history = model.fit(
      2     x= test,
      3     batch_size=10,
      4     steps_per_epoch = int(total_sample/batch_size),
      5     epochs=n_epochs,

/usr/local/lib/python3.8/dist-packages/keras/utils/traceback_utils.py in erro
r_handler(*args, **kwargs)
     65     except Exception as e:  # pylint: disable=broad-except
     66       filtered_tb = _process_traceback_frames(e.__traceback__)
---> 67       raise e.with_traceback(filtered_tb) from None
     68     finally:
     69       del filtered_tb

/usr/local/lib/python3.8/dist-packages/tensorflow/python/eager/execute.py in
quick_execute(op_name, num_outputs, inputs, attrs, ctx, name)
     52   try:
     53     ctx.ensure_initialized()
---> 54     tensors = pywrap_tfe.TFE_Py_Execute(ctx._handle, device_name, op_
name,
     55                                         inputs, attrs, num_outputs)
     56   except core._NotOkStatusException as e:

InvalidArgumentError: Graph execution error:

Detected at node 'sequential/dense/Relu' defined at (most recent call last):
    File "/usr/lib/python3.8/runpy.py", line 194, in _run_module_as_main
      return _run_code(code, main_globals, None,
    File "/usr/lib/python3.8/runpy.py", line 87, in _run_code
      exec(code, run_globals)
    File "/usr/local/lib/python3.8/dist-packages/ipykernel_launcher.py", line
16, in <module>
      app.launch_new_instance()
    File "/usr/local/lib/python3.8/dist-packages/traitlets/config/applicatio
n.py", line 985, in launch_instance
      app.start()
    File "/usr/local/lib/python3.8/dist-packages/ipykernel/kernelapp.py", lin
e 612, in start
      self.io_loop.start()
    File "/usr/local/lib/python3.8/dist-packages/tornado/platform/asyncio.p
y", line 149, in start
      self.asyncio_loop.run_forever()
    File "/usr/lib/python3.8/asyncio/base_events.py", line 570, in run_foreve
r
      self._run_once()
    File "/usr/lib/python3.8/asyncio/base_events.py", line 1859, in _run_once
      handle._run()
    File "/usr/lib/python3.8/asyncio/events.py", line 81, in _run
      self._context.run(self._callback, *self._args)
    File "/usr/local/lib/python3.8/dist-packages/tornado/ioloop.py", line 69
0, in <lambda>
      lambda f: self._run_callback(functools.partial(callback, future))
    File "/usr/local/lib/python3.8/dist-packages/tornado/ioloop.py", line 74
3, in _run_callback
```

```
      ret = callback()
    File "/usr/local/lib/python3.8/dist-packages/tornado/gen.py", line 787, i
n inner
      self.run()
    File "/usr/local/lib/python3.8/dist-packages/tornado/gen.py", line 748, i
n run
      yielded = self.gen.send(value)
    File "/usr/local/lib/python3.8/dist-packages/ipykernel/kernelbase.py", li
ne 365, in process_one
      yield gen.maybe_future(dispatch(*args))
    File "/usr/local/lib/python3.8/dist-packages/tornado/gen.py", line 209, i
n wrapper
      yielded = next(result)
    File "/usr/local/lib/python3.8/dist-packages/ipykernel/kernelbase.py", li
ne 268, in dispatch_shell
      yield gen.maybe_future(handler(stream, idents, msg))
    File "/usr/local/lib/python3.8/dist-packages/tornado/gen.py", line 209, i
n wrapper
      yielded = next(result)
    File "/usr/local/lib/python3.8/dist-packages/ipykernel/kernelbase.py", li
ne 543, in execute_request
      self.do_execute(
    File "/usr/local/lib/python3.8/dist-packages/tornado/gen.py", line 209, i
n wrapper
      yielded = next(result)
    File "/usr/local/lib/python3.8/dist-packages/ipykernel/ipkernel.py", line
306, in do_execute
      res = shell.run_cell(code, store_history=store_history, silent=silent)
    File "/usr/local/lib/python3.8/dist-packages/ipykernel/zmqshell.py", line
536, in run_cell
      return super(ZMQInteractiveShell, self).run_cell(*args, **kwargs)
    File "/usr/local/lib/python3.8/dist-packages/IPython/core/interactiveshel
l.py", line 2854, in run_cell
      result = self._run_cell(
    File "/usr/local/lib/python3.8/dist-packages/IPython/core/interactiveshel
l.py", line 2881, in _run_cell
      return runner(coro)
    File "/usr/local/lib/python3.8/dist-packages/IPython/core/async_helpers.p
y", line 68, in _pseudo_sync_runner
      coro.send(None)
    File "/usr/local/lib/python3.8/dist-packages/IPython/core/interactiveshel
l.py", line 3057, in run_cell_async
      has_raised = await self.run_ast_nodes(code_ast.body, cell_name,
    File "/usr/local/lib/python3.8/dist-packages/IPython/core/interactiveshel
l.py", line 3249, in run_ast_nodes
      if (await self.run_code(code, result,  async_=asy)):
    File "/usr/local/lib/python3.8/dist-packages/IPython/core/interactiveshel
l.py", line 3326, in run_code
      exec(code_obj, self.user_global_ns, self.user_ns)
    File "<ipython-input-23-751ede4a5913>", line 1, in <module>
      history = model.fit(
    File "/usr/local/lib/python3.8/dist-packages/keras/utils/traceback_utils.
py", line 64, in error_handler
      return fn(*args, **kwargs)
    File "/usr/local/lib/python3.8/dist-packages/keras/engine/training.py", l
ine 1409, in fit
      tmp_logs = self.train_function(iterator)
    File "/usr/local/lib/python3.8/dist-packages/keras/engine/training.py", l
ine 1051, in train_function
      return step_function(self, iterator)
    File "/usr/local/lib/python3.8/dist-packages/keras/engine/training.py", l
ine 1040, in step_function
      outputs = model.distribute_strategy.run(run_step, args=(data,))
    File "/usr/local/lib/python3.8/dist-packages/keras/engine/training.py", l
ine 1030, in run_step
      outputs = model.train_step(data)
    File "/usr/local/lib/python3.8/dist-packages/keras/engine/training.py", l
ine 889, in train_step
      y_pred = self(x, training=True)
    File "/usr/local/lib/python3.8/dist-packages/keras/utils/traceback_utils.
py", line 64, in error_handler
      return fn(*args, **kwargs)
    File "/usr/local/lib/python3.8/dist-packages/keras/engine/training.py", l
ine 490, in __call__
      return super().__call__(*args, **kwargs)
    File "/usr/local/lib/python3.8/dist-packages/keras/utils/traceback_utils.
py", line 64, in error_handler
      return fn(*args, **kwargs)
    File "/usr/local/lib/python3.8/dist-packages/keras/engine/base_layer.py",
line 1014, in __call__
      outputs = call_fn(inputs, *args, **kwargs)
    File "/usr/local/lib/python3.8/dist-packages/keras/utils/traceback_utils.
py", line 92, in error_handler
      return fn(*args, **kwargs)
    File "/usr/local/lib/python3.8/dist-packages/keras/engine/sequential.py",
line 374, in call
      return super(Sequential, self).call(inputs, training=training, mask=mas
k)
    File "/usr/local/lib/python3.8/dist-packages/keras/engine/functional.py",
line 458, in call
      return self._run_internal_graph(
    File "/usr/local/lib/python3.8/dist-packages/keras/engine/functional.py",
line 596, in _run_internal_graph
      outputs = node.layer(*args, **kwargs)
    File "/usr/local/lib/python3.8/dist-packages/keras/utils/traceback_utils.
py", line 64, in error_handler
      return fn(*args, **kwargs)
    File "/usr/local/lib/python3.8/dist-packages/keras/engine/base_layer.py",
line 1014, in __call__
      outputs = call_fn(inputs, *args, **kwargs)
    File "/usr/local/lib/python3.8/dist-packages/keras/utils/traceback_utils.
py", line 92, in error_handler
      return fn(*args, **kwargs)
    File "/usr/local/lib/python3.8/dist-packages/keras/layers/core/dense.py",
line 235, in call
      outputs = self.activation(outputs)
    File "/usr/local/lib/python3.8/dist-packages/keras/activations.py", line
311, in relu
      return backend.relu(x, alpha=alpha, max_value=max_value, threshold=thre
shold)
    File "/usr/local/lib/python3.8/dist-packages/keras/backend.py", line 499
2, in relu
      x = tf.nn.relu(x)
Node: 'sequential/dense/Relu'
```

```
Matrix size-incompatible: In[0]: [10,196608], In[1]: [65536,128]
         [[{{node sequential/dense/Relu}}]] [Op:__inference_train_function_15
34]
```

8. Accuracy

Plot accuracy value for train and validation

In [ ]:
```
history.history.keys()
```

In [ ]:
```
plt.plot(history.history['val_acc'])
plt.plot(history.history['acc'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```

In [ ]:
```
score = model.evaluate(test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

9. RNN Architecture

The RNN uses the genereic model , to show the accuracy of the data

In [24]:
```
num_epochs = 2
model=tf.keras.models.Sequential()

model.add(layers.Input(shape=[256,256,3]))
model.add(layers.Rescaling(1/255))
model.add(layers.ConvLSTM1D(128,3,activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.Flatten())
model.add(layers.Dense(3,activation='softmax'))
```

In [25]:

```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 rescaling (Rescaling)       (None, 256, 256, 3)       0         
                                                                 
 conv_lstm1d (ConvLSTM1D)    (None, 254, 128)          201728    
                                                                 
 batch_normalization (BatchN  (None, 254, 128)         512       
 ormalization)                                                   
                                                                 
 flatten_1 (Flatten)         (None, 32512)             0         
                                                                 
 dense_3 (Dense)             (None, 3)                 97539     
                                                                 
=================================================================
Total params: 299,779
Trainable params: 299,523
Non-trainable params: 256
_________________________________________________________________
```

In [26]:

```
model.compile(
    optimizer = 'adam',
    loss = 'categorical_crossentropy',
    metrics = ['accuracy']
)

model.fit(
    x = train,
    batch_size=1,
    epochs = 2,
    validation_data = (test),
    validation_steps = None,
    shuffle = False
)

pred_rnn = model.predict(test)
pred_rnn = np.argmax(pred_rnn, axis = 1)
print('\naccuracy: ', accuracy_score(test.labels, pred_rnn))
```

```
Epoch 1/2
   7/1200 [..............................] - ETA: 6:58:51 - loss: 1.1096 - ac
curacy: 0.2714
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-26-a19101845148> in <module>
      5 )
      6 
----> 7 model.fit(
      8     x = train,
      9     batch_size=1,

/usr/local/lib/python3.8/dist-packages/keras/utils/traceback_utils.py in erro
r_handler(*args, **kwargs)
     62     filtered_tb = None
     63     try:
---> 64       return fn(*args, **kwargs)
     65     except Exception as e:  # pylint: disable=broad-except
     66       filtered_tb = _process_traceback_frames(e.__traceback__)

/usr/local/lib/python3.8/dist-packages/keras/engine/training.py in fit(self,
x, y, batch_size, epochs, verbose, callbacks, validation_split, validation_da
ta, shuffle, class_weight, sample_weight, initial_epoch, steps_per_epoch, val
idation_steps, validation_batch_size, validation_freq, max_queue_size, worker
s, use_multiprocessing)
   1407                 _r=1):
   1408               callbacks.on_train_batch_begin(step)
-> 1409               tmp_logs = self.train_function(iterator)
   1410               if data_handler.should_sync:
   1411                 context.async_wait()

/usr/local/lib/python3.8/dist-packages/tensorflow/python/util/traceback_util
s.py in error_handler(*args, **kwargs)
    148     filtered_tb = None
    149     try:
--> 150       return fn(*args, **kwargs)
    151     except Exception as e:
    152       filtered_tb = _process_traceback_frames(e.__traceback__)

/usr/local/lib/python3.8/dist-packages/tensorflow/python/eager/def_function.p
y in __call__(self, *args, **kwds)
    913 
    914       with OptionalXlaContext(self._jit_compile):
--> 915         result = self._call(*args, **kwds)
    916 
    917       new_tracing_count = self.experimental_get_tracing_count()

/usr/local/lib/python3.8/dist-packages/tensorflow/python/eager/def_function.p
y in _call(self, *args, **kwds)
    945       # In this case we have created variables on the first call, so
we run the
    946       # defunned version which is guaranteed to never create variable
s.
--> 947       return self._stateless_fn(*args, **kwds)  # pylint: disable=not
-callable
    948     elif self._stateful_fn is not None:
    949       # Release the lock early so that multiple threads can perform t
he call

/usr/local/lib/python3.8/dist-packages/tensorflow/python/eager/function.py in
```

```
__call__(self, *args, **kwargs)
   2451       (graph_function,
   2452        filtered_flat_args) = self._maybe_define_function(args, kwarg
s)
-> 2453     return graph_function._call_flat(
   2454         filtered_flat_args, captured_inputs=graph_function.captured_i
nputs)  # pylint: disable=protected-access
   2455

/usr/local/lib/python3.8/dist-packages/tensorflow/python/eager/function.py in
_call_flat(self, args, captured_inputs, cancellation_manager)
   1858         and executing_eagerly):
   1859       # No tape is watching; skip to running the function.
-> 1860       return self._build_call_outputs(self._inference_function.call(
   1861           ctx, args, cancellation_manager=cancellation_manager))
   1862     forward_backward = self._select_forward_and_backward_functions(

/usr/local/lib/python3.8/dist-packages/tensorflow/python/eager/function.py in
call(self, ctx, args, cancellation_manager)
    495       with _InterpolateFunctionError(self):
    496         if cancellation_manager is None:
--> 497           outputs = execute.execute(
    498               str(self.signature.name),
    499               num_outputs=self._num_outputs,

/usr/local/lib/python3.8/dist-packages/tensorflow/python/eager/execute.py in
quick_execute(op_name, num_outputs, inputs, attrs, ctx, name)
     52   try:
     53     ctx.ensure_initialized()
---> 54     tensors = pywrap_tfe.TFE_Py_Execute(ctx._handle, device_name, op_
name,
     55                                         inputs, attrs, num_outputs)
     56   except core._NotOkStatusException as e:

KeyboardInterrupt:
```

10. CNN Architecture

In [ ]:

```
from keras.optimizers.optimizer_v1 import Optimizer

model=tf.keras.models.Sequential()

#First convolution layer
model.add(layers.Conv2D(16,(3,3),input_shape=[256,256,3],activation='relu'))
model.add(layers.MaxPooling2D(2,2))
#Second convolution layer
model.add(layers.Conv2D(32,(3,3),activation='relu'))
model.add(layers.MaxPooling2D(2,2))
#Third convolution layer
model.add(layers.Conv2D(64,(3,3),activation='relu'))
#Forth convolution layer
model.add(layers.Conv2D(64,(3,3),activation='relu'))
model.add(layers.MaxPooling2D(2,2))
#Forth convolution layer
model.add(layers.Conv2D(256,(3,3),activation='relu'))
model.add(layers.MaxPooling2D(2,2))
#Fifth convolution layer
model.add(layers.Conv2D(256,(3,3),activation='relu'))
model.add(layers.MaxPooling2D(2,2))
#Flatten the resilts to feed into dense layer
model.add(layers.Flatten())
#Dropouts some neurons
model.add(layers.Dropout(0.5))
#64 neuron in fully connected layer
model.add(layers.Dense(64, activation='relu'))
#Three output neurons for three classes with softmax activation
model.add(layers.Dense(3, activation='softmax'))
```

In [ ]:

```
# Summary
model.summary()
```

11. Evaluate on the test data

Compile and fit.

In [ ]:

```
model.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy'])
```

In [ ]:

```
model.fit(train,
          epochs=10,batch_size=10,
          validation_data=(test),
          callbacks=[callback],)
```

In [ ]:

```
pred = model.predict(test)
pred_pre = np.argmax(pred, axis=1)
print('\naccuracy: ', accuracy_score(test.labels, pred_pre))
```

```
94/94 [==============================] - 44s 465ms/step

accuracy:  0.3256666666666666
```

In [ ]:

```
plt.plot(accuracy_score(test.labels,pred_pre),test)
```

12. Transfer learning

In [ ]:

```
from tensorflow.keras.applications.efficientnet import EfficientNetB3
base_model = EfficientNetB3(include_top=False, input_shape=[256,256,3], weights="imagenet", pooling='max')
img_shape =[256,256,3]
num_epochs = 10

# init base model

fine_tune = 100
for layer in base_model.layers[:fine_tune]:
  layer.trainable =False


model_pre = tf.keras.models.Sequential()

model_pre.add(layers.Input(shape=img_shape))
model_pre.add(base_model)
model_pre.add(layers.BatchNormalization())
model_pre.add(layers.Dense(256, activation='relu'))
model_pre.add(layers.Dropout(rate=.2, seed=1234))
model_pre.add(layers.Dense(256, activation='relu'))
model_pre.add(layers.Dropout(rate=.2, seed=1234))
model_pre.add(layers.Dense(3, activation='softmax'))
```

In [ ]:

```
model_pre.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 efficientnetb3 (Functional)  (None, 1536)             10783535  
                                                                 
 batch_normalization (BatchN  (None, 1536)             6144      
 ormalization)                                                   
                                                                 
 dense_3 (Dense)             (None, 256)               393472    
                                                                 
 dropout_2 (Dropout)         (None, 256)               0         
                                                                 
 dense_4 (Dense)             (None, 256)               65792     
                                                                 
 dropout_3 (Dropout)         (None, 256)               0         
                                                                 
 dense_5 (Dense)             (None, 3)                 771       
                                                                 
=================================================================
Total params: 11,249,714
Trainable params: 11,039,049
Non-trainable params: 210,665
_________________________________________________________________
```

In [ ]:

```
model_pre.compile(
    optimizer = 'adam',
    loss = 'categorical_crossentropy',
    metrics = ['accuracy']
)

model_pre.fit(
    x = train,
    batch_size=10,
    epochs = 30,
    validation_data = (test),
    validation_steps = None,
    shuffle = False
)

pred_pre = model_pre.predict(test)
pred_pre = np.argmax(pred_pre, axis=1)
print('\naccuracy: ', accuracy_score(test.labels, pred_pre))
```

```
Epoch 1/30
375/375 [==============================] - 289s 729ms/step - loss: 0.6167 - a
ccuracy: 0.7593 - val_loss: 0.4757 - val_accuracy: 0.8463
Epoch 2/30
375/375 [==============================] - 266s 710ms/step - loss: 0.3483 - a
ccuracy: 0.8722 - val_loss: 0.1962 - val_accuracy: 0.9313
Epoch 3/30
375/375 [==============================] - 265s 707ms/step - loss: 0.2356 - a
ccuracy: 0.9157 - val_loss: 0.1300 - val_accuracy: 0.9513
Epoch 4/30
375/375 [==============================] - 264s 705ms/step - loss: 0.2121 - a
ccuracy: 0.9262 - val_loss: 0.1765 - val_accuracy: 0.9367
Epoch 5/30
375/375 [==============================] - 264s 705ms/step - loss: 0.1831 - a
ccuracy: 0.9356 - val_loss: 0.1360 - val_accuracy: 0.9527
Epoch 6/30
375/375 [==============================] - 263s 701ms/step - loss: 0.1681 - a
ccuracy: 0.9440 - val_loss: 0.1904 - val_accuracy: 0.9287
Epoch 7/30
375/375 [==============================] - 263s 701ms/step - loss: 0.1503 - a
ccuracy: 0.9492 - val_loss: 0.1169 - val_accuracy: 0.9550
Epoch 8/30
375/375 [==============================] - 263s 702ms/step - loss: 0.1487 - a
ccuracy: 0.9513 - val_loss: 0.1299 - val_accuracy: 0.9527
Epoch 9/30
375/375 [==============================] - 263s 701ms/step - loss: 0.1557 - a
ccuracy: 0.9477 - val_loss: 0.1314 - val_accuracy: 0.9590
Epoch 10/30
375/375 [==============================] - 263s 700ms/step - loss: 0.1910 - a
ccuracy: 0.9338 - val_loss: 2.7454 - val_accuracy: 0.6023
Epoch 11/30
375/375 [==============================] - 263s 700ms/step - loss: 0.1753 - a
ccuracy: 0.9420 - val_loss: 0.0912 - val_accuracy: 0.9667
Epoch 12/30
375/375 [==============================] - 263s 701ms/step - loss: 0.1295 - a
ccuracy: 0.9573 - val_loss: 0.1028 - val_accuracy: 0.9587
Epoch 13/30
375/375 [==============================] - 263s 701ms/step - loss: 0.1292 - a
ccuracy: 0.9574 - val_loss: 0.0789 - val_accuracy: 0.9733
Epoch 14/30
375/375 [==============================] - 263s 700ms/step - loss: 0.1075 - a
ccuracy: 0.9642 - val_loss: 0.0629 - val_accuracy: 0.9803
Epoch 15/30
375/375 [==============================] - 263s 700ms/step - loss: 0.1246 - a
ccuracy: 0.9602 - val_loss: 0.0782 - val_accuracy: 0.9697
Epoch 16/30
375/375 [==============================] - 262s 699ms/step - loss: 0.1114 - a
ccuracy: 0.9634 - val_loss: 0.1410 - val_accuracy: 0.9553
Epoch 17/30
375/375 [==============================] - 262s 699ms/step - loss: 0.1175 - a
ccuracy: 0.9616 - val_loss: 0.1017 - val_accuracy: 0.9663
Epoch 18/30
375/375 [==============================] - 262s 698ms/step - loss: 0.1099 - a
ccuracy: 0.9615 - val_loss: 0.1110 - val_accuracy: 0.9627
Epoch 19/30
375/375 [==============================] - 261s 697ms/step - loss: 0.0989 - a
ccuracy: 0.9678 - val_loss: 0.1201 - val_accuracy: 0.9623
```

```
Epoch 20/30
375/375 [==============================] - 261s 697ms/step - loss: 0.0999 - a
ccuracy: 0.9667 - val_loss: 0.1223 - val_accuracy: 0.9603
Epoch 21/30
375/375 [==============================] - 261s 695ms/step - loss: 0.0985 - a
ccuracy: 0.9677 - val_loss: 0.1191 - val_accuracy: 0.9610
Epoch 22/30
375/375 [==============================] - 261s 696ms/step - loss: 0.1088 - a
ccuracy: 0.9651 - val_loss: 0.1318 - val_accuracy: 0.9543
Epoch 23/30
375/375 [==============================] - 261s 696ms/step - loss: 0.0887 - a
ccuracy: 0.9702 - val_loss: 0.1228 - val_accuracy: 0.9583
Epoch 24/30
375/375 [==============================] - 261s 697ms/step - loss: 0.1406 - a
ccuracy: 0.9532 - val_loss: 0.0705 - val_accuracy: 0.9750
Epoch 25/30
375/375 [==============================] - 261s 697ms/step - loss: 0.0964 - a
ccuracy: 0.9697 - val_loss: 0.0725 - val_accuracy: 0.9797
Epoch 26/30
375/375 [==============================] - 261s 696ms/step - loss: 0.0780 - a
ccuracy: 0.9746 - val_loss: 0.0606 - val_accuracy: 0.9763
Epoch 27/30
375/375 [==============================] - 261s 695ms/step - loss: 0.0971 - a
ccuracy: 0.9678 - val_loss: 0.0814 - val_accuracy: 0.9773
Epoch 28/30
375/375 [==============================] - 261s 696ms/step - loss: 0.0972 - a
ccuracy: 0.9692 - val_loss: 0.0993 - val_accuracy: 0.9723
Epoch 29/30
375/375 [==============================] - 261s 696ms/step - loss: 0.0766 - a
ccuracy: 0.9752 - val_loss: 0.0802 - val_accuracy: 0.9770
Epoch 30/30
375/375 [==============================] - 261s 694ms/step - loss: 0.0786 - a
ccuracy: 0.9743 - val_loss: 0.0871 - val_accuracy: 0.9760
94/94 [==============================] - 49s 501ms/step

accuracy:  0.33633333333333333
```

```
In [ ]: base_model.summary()
```

13. Analysis

The data is to classify between three kinds of footware: boots, shoes and sandle. The model CNN and RNN got the same tesable summary shows the tesability of the data is possible. CNN is giving slight better accuracy compare to RNN. Thoguh I tried to run RNN with increse epoch but the RAM was used up so could not execute or taking too long. The best should be the pre_model from the transfer leaning. It is giving the best accuracy at the moment with epoch of 30. Though it is still a low number it is high than CNN and RNN . The low batch_size did help but due to pre-runed the RAM & GPU was already low. The transfer learning tunes the data which is expected to give more accurate but for some reason it still low of .335 but greater than RNN and CNN. The model are functioning the issue is lack of RAM and GPU limitation.