Theophilus Braimoh (tob2zd)

3/12/2023

Recognizing UVA landmarks with neural nets (70 pts)

The UVA Grounds is known for its Jeffersonian architecture and place in U.S. history as a model for college and university campuses throughout the country. Throughout its history, the University of Virginia has won praises for its unique Jeffersonian architecture.

In this assignment, you will attempt the build an image recognition system to classify different buildlings/landmarks on Grounds. You will earn 70 points for this assignment plus 10 bonus points if (1) your classifier performs exceed 94% accuracy.

To make it easier for you, some codes have been provided to help you process the data, you may modify it to fit your needs. You must submit the .ipynb file via UVA Collab with the following format: yourcomputingID_assignment_2.ipynb

Best of luck, and have fun!

Load Packages

```
import sys
import sklearn
import os
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
from functools import partial
import PIL
import PIL.Image
```

```
import tensorflow as tf

%tensorflow_version 2.x
import tensorflow as tf
from tensorflow import keras

np.random.seed(42) # note that you must use the same seed to ensure consistentcy in your training/validation/testing
tf.random.set_seed(42)
```

```
    Colab only includes TensorFlow 2.x; %tensorflow_version has no effect.
```

Import Dataset

The full dataset is huge (+37GB) with +13K images of 18 classes. So it will take a while to download, extract, and process. To save you time and effort, a subset of the data has been resized and compressed to only 379Mb and stored in my Firebase server. This dataset will be the one you will benchmark for your grade. If you are up for a challenge (and perhaps bonus points), contact the instructor for the full dataset!

```
# Download dataset from FirebaseStorage
!pip install wget
!wget https://firebasestorage.googleapis.com/v0/b/uva-landmark-images.appspot.com/o/dataset.zip?alt=media&token=e1403951-30d6-42b8-ba4e-394af
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting wget
  Downloading wget-3.2.zip (10 kB)
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: wget
  Building wheel for wget (setup.py) ... done
  Created wheel for wget: filename=wget-3.2-py3-none-any.whl size=9674 sha256=b2f102052a58a948c3094094b7ec7400d90ccc7d998f1e6ba8afdc74c9
  Stored in directory: /root/.cache/pip/wheels/04/5f/3e/46cc37c5d698415694d83f607f833f83f0149e49b3af9d0f38
Successfully built wget
Installing collected packages: wget
Successfully installed wget-3.2
--2023-03-13 03:03:25--  https://firebasestorage.googleapis.com/v0/b/uva-landmark-images.appspot.com/o/dataset.zip?alt=media
Resolving firebasestorage.googleapis.com (firebasestorage.googleapis.com)... 142.251.10.95, 142.251.12.95, 172.217.194.95, ...
Connecting to firebasestorage.googleapis.com (firebasestorage.googleapis.com)|142.251.10.95|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 397174132 (379M) [application/zip]
Saving to: ‘dataset.zip?alt=media’

dataset.zip?alt=med 100%[===================>] 378.77M  24.0MB/s    in 17s

2023-03-13 03:03:43 (21.9 MB/s) - ‘dataset.zip?alt=media’ saved [397174132/397174132]
```

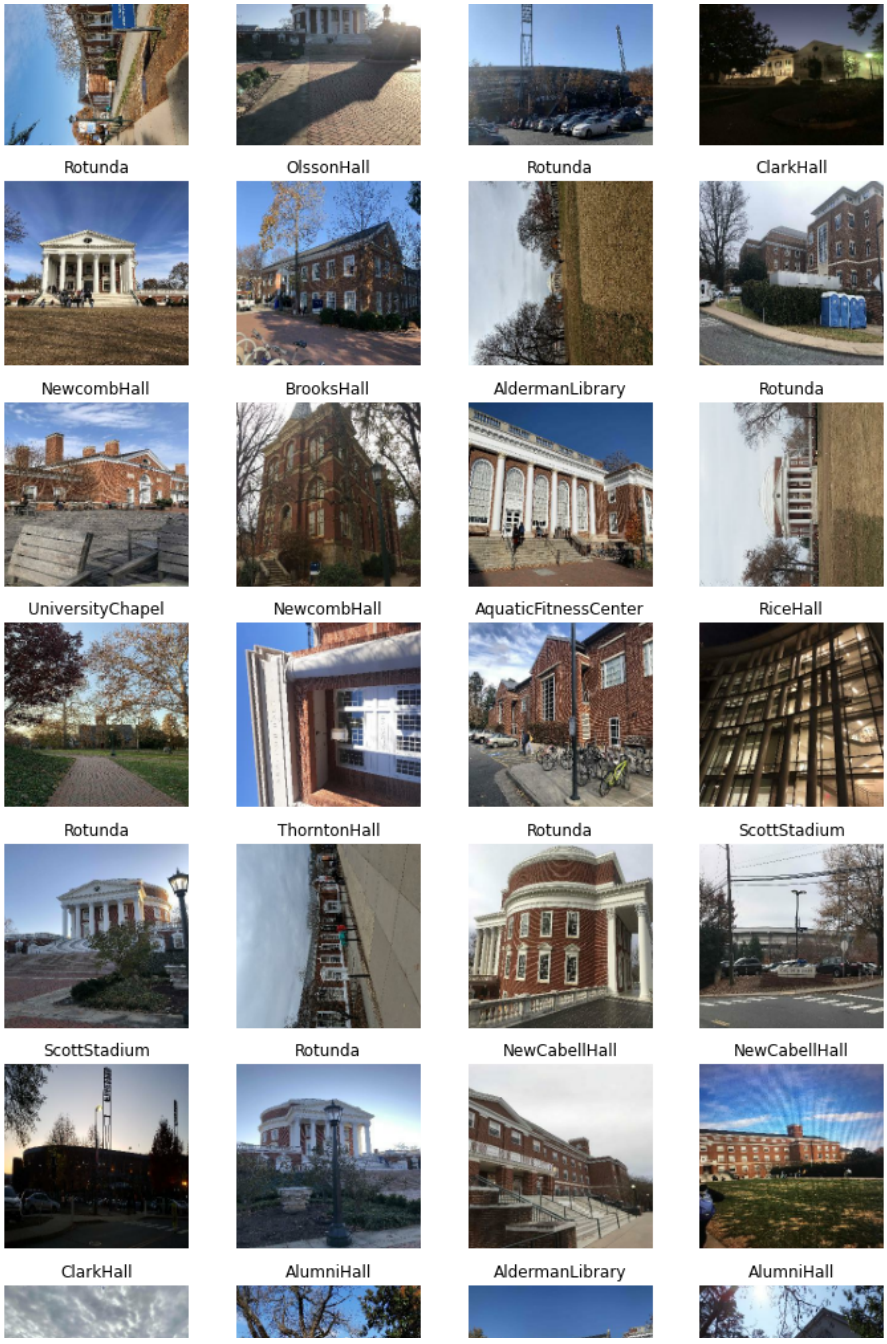
```
# Extract content
!unzip "/content/dataset.zip?alt=media"
```

```
        inflating: __MACOSX/dataset/MinorHall/._0000004263.jpg
        inflating: dataset/MinorHall/0000004539.jpg
        inflating: __MACOSX/dataset/MinorHall/._0000004539.jpg
        inflating: dataset/MinorHall/0000004713.jpg
        inflating: __MACOSX/dataset/MinorHall/._0000004713.jpg
        inflating: dataset/MinorHall/0000004075.jpg
        inflating: __MACOSX/dataset/MinorHall/._0000004075.jpg
        inflating: dataset/MinorHall/0000004061.jpg
        inflating: __MACOSX/dataset/MinorHall/._0000004061.jpg
        inflating: dataset/MinorHall/0000004707.jpg
        inflating: __MACOSX/dataset/MinorHall/._0000004707.jpg
        inflating: dataset/MinorHall/0000004049.jpg
        inflating:   MACOSX/dataset/MinorHall/._0000004049.jpg
```

```
from sklearn.datasets import load_files
from keras.utils import np_utils

from keras.preprocessing import image
from tqdm import tqdm # progress bar

data_dir = "/content/dataset/"
batch_size = 32;
# IMPORTANT: Depends on what pre-trained model you choose, you will need to change these dimensions accordingly
img_height = 150;
img_width = 150;

# Training Dataset
train_ds = tf.keras.preprocessing.image_dataset_from_directory(
    data_dir,
    validation_split = 0.2,
    subset = "training",
    seed = 42,
    image_size= (img_height, img_width),
    batch_size = batch_size
)

# Validation Dataset
validation_ds = tf.keras.preprocessing.image_dataset_from_directory(
    data_dir,
    validation_split = 0.2,
    subset = "validation",
    seed = 42,
    image_size = (img_height, img_width),
    batch_size = batch_size
)
```

```
     Found 14286 files belonging to 18 classes.
     Using 11429 files for training.
     Found 14286 files belonging to 18 classes.
     Using 2857 files for validation.
```

```
# Visualize some of the train samples of one batch
# Make sure you create the class names that match the order of their appearances in the "files" variable
class_names = ['AcademicalVillage', 'AldermanLibrary', 'AlumniHall', 'AquaticFitnessCenter',
  'BavaroHall', 'BrooksHall', 'ClarkHall', 'MadisonHall', 'MinorHall', 'NewCabellHall',
  'NewcombHall', 'OldCabellHall', 'OlssonHall', 'RiceHall', 'Rotunda', 'ScottStadium',
  'ThorntonHall', 'UniversityChapel']

# Rows and columns are set to fit one training batch (32)
n_rows = 8
n_cols = 4
plt.figure(figsize=(n_cols * 3, n_rows * 3))
for images, labels in train_ds.take(1):
    for i in range (n_rows*n_cols):
        plt.subplot(n_rows, n_cols, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))
        plt.axis('off')
        plt.title(class_names[labels[i]], fontsize=12)
plt.subplots_adjust(wspace=.2, hspace=.2)
```



It's your turn: Building a classifier for UVA Landmark Dataset

You may design your own architecture AND re-use any of the exising frameworks.

Best of luck!

```
# YOUR CODE STARTS HERE
```

```
num_classes = len(train_ds.class_names)
print(f"Number of classes: {num_classes}") #18 classes as tated in problem title
```

```
Number of classes: 18
```

Model 1

```
# Using pretrained ResNet50 from the Keras API: https://keras.io/api/applications/
# set include_top to false to remove the top layer with 1000 classes. Our dataset has 18 classes.
model_1_no_top = keras.applications.resnet50.ResNet50(weights="imagenet", include_top=False)
model_1_with_top = keras.applications.resnet50.ResNet50(weights="imagenet") # you may also set include_top=True
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels_no
94765736/94765736 [==============================] - 5s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels.h5
102967424/102967424 [==============================] - 5s 0us/step
```

```
# Print the model architecture with the top layer included
model_1_with_top.summary()
```

```
 conv5_block2_1_relu (Activatio  (None, 7, 7, 512)   0           ['conv5_block2_1_bn[0][0]']
 n)

 conv5_block2_2_conv (Conv2D)   (None, 7, 7, 512)    2359808     ['conv5_block2_1_relu[0][0]']

 conv5_block2_2_bn (BatchNormal  (None, 7, 7, 512)   2048        ['conv5_block2_2_conv[0][0]']
 ization)

 conv5_block2_2_relu (Activatio  (None, 7, 7, 512)   0           ['conv5_block2_2_bn[0][0]']
 n)

 conv5_block2_3_conv (Conv2D)   (None, 7, 7, 2048)   1050624     ['conv5_block2_2_relu[0][0]']

 conv5_block2_3_bn (BatchNormal  (None, 7, 7, 2048)  8192        ['conv5_block2_3_conv[0][0]']
 ization)

 conv5_block2_add (Add)         (None, 7, 7, 2048)   0           ['conv5_block1_out[0][0]',
                                                                  'conv5_block2_3_bn[0][0]']

 conv5_block2_out (Activation)  (None, 7, 7, 2048)   0           ['conv5_block2_add[0][0]']

 conv5_block3_1_conv (Conv2D)   (None, 7, 7, 512)    1049088     ['conv5_block2_out[0][0]']

 conv5_block3_1_bn (BatchNormal  (None, 7, 7, 512)   2048        ['conv5_block3_1_conv[0][0]']
 ization)

 conv5_block3_1_relu (Activatio  (None, 7, 7, 512)   0           ['conv5_block3_1_bn[0][0]']
 n)

 conv5_block3_2_conv (Conv2D)   (None, 7, 7, 512)    2359808     ['conv5_block3_1_relu[0][0]']

 conv5_block3_2_bn (BatchNormal  (None, 7, 7, 512)   2048        ['conv5_block3_2_conv[0][0]']
 ization)

 conv5_block3_2_relu (Activatio  (None, 7, 7, 512)   0           ['conv5_block3_2_bn[0][0]']
 n)

 conv5_block3_3_conv (Conv2D)   (None, 7, 7, 2048)   1050624     ['conv5_block3_2_relu[0][0]']

 conv5_block3_3_bn (BatchNormal  (None, 7, 7, 2048)  8192        ['conv5_block3_3_conv[0][0]']
 ization)

 conv5_block3_add (Add)         (None, 7, 7, 2048)   0           ['conv5_block2_out[0][0]',
                                                                  'conv5_block3_3_bn[0][0]']

 conv5_block3_out (Activation)  (None, 7, 7, 2048)   0           ['conv5_block3_add[0][0]']

 avg_pool (GlobalAveragePooling  (None, 2048)        0           ['conv5_block3_out[0][0]']
 2D)

 predictions (Dense)            (None, 1000)         2049000     ['avg_pool[0][0]']

==================================================================================================
Total params: 25,636,712
Trainable params: 25,583,592
Non-trainable params: 53,120
__________________________________________________________________________________________________
```

```
# Print the model architecture without top layer
model_1_no_top.summary()
```

```
n)                              512)

conv5_block2_2_conv (Conv2D)   (None, None, None,   2359808     ['conv5_block2_1_relu[0][0]']
                                512)

conv5_block2_2_bn (BatchNormal  (None, None, None,   2048       ['conv5_block2_2_conv[0][0]']
ization)                        512)

conv5_block2_2_relu (Activatio  (None, None, None,   0          ['conv5_block2_2_bn[0][0]']
n)                              512)

conv5_block2_3_conv (Conv2D)   (None, None, None,   1050624     ['conv5_block2_2_relu[0][0]']
                                2048)

conv5_block2_3_bn (BatchNormal  (None, None, None,   8192       ['conv5_block2_3_conv[0][0]']
ization)                        2048)

conv5_block2_add (Add)         (None, None, None,   0           ['conv5_block1_out[0][0]',
                                2048)                            'conv5_block2_3_bn[0][0]']

conv5_block2_out (Activation)  (None, None, None,   0           ['conv5_block2_add[0][0]']
                                2048)

conv5_block3_1_conv (Conv2D)   (None, None, None,   1049088     ['conv5_block2_out[0][0]']
                                512)

conv5_block3_1_bn (BatchNormal  (None, None, None,   2048       ['conv5_block3_1_conv[0][0]']
ization)                        512)

conv5_block3_1_relu (Activatio  (None, None, None,   0          ['conv5_block3_1_bn[0][0]']
n)                              512)

conv5_block3_2_conv (Conv2D)   (None, None, None,   2359808     ['conv5_block3_1_relu[0][0]']
                                512)

conv5_block3_2_bn (BatchNormal  (None, None, None,   2048       ['conv5_block3_2_conv[0][0]']
ization)                        512)

conv5_block3_2_relu (Activatio  (None, None, None,   0          ['conv5_block3_2_bn[0][0]']
n)                              512)

conv5_block3_3_conv (Conv2D)   (None, None, None,   1050624     ['conv5_block3_2_relu[0][0]']
                                2048)

conv5_block3_3_bn (BatchNormal  (None, None, None,   8192       ['conv5_block3_3_conv[0][0]']
ization)                        2048)

conv5_block3_add (Add)         (None, None, None,   0           ['conv5_block2_out[0][0]',
                                2048)                            'conv5_block3_3_bn[0][0]']

conv5_block3_out (Activation)  (None, None, None,   0           ['conv5_block3_add[0][0]']
                                2048)

==================================================================================================
Total params: 23,587,712
Trainable params: 23,534,592
Non-trainable params: 53,120
__________________________________________________________________________________________________
```

```
# Add new avg_pool_layer for aggregation of the "conv5_block3_out" layer. You can name it just "avg_pool", if you want.
avg_pool_layer = keras.layers.GlobalAveragePooling2D()(model_1_no_top.output)
```

```
# Verify the shape of the weights of the new avg_pool_layer conforms with the output of the "conv5_block3_out" layer
avg_pool_layer.shape
```

```
TensorShape([None, 2048])
```

```
# Add Fully connected output layer with the number of classes (18) in our dataset. Here we named it predictions, you may call it something el
output_layer = keras.layers.Dense(num_classes, activation="softmax", name="predictions")(avg_pool_layer)
output_layer.shape
```

```
TensorShape([None, 18])
```

```
model_1 = keras.models.Model(inputs=model_1_no_top.input, outputs=output_layer)
```

```
# Print the modified architecture
model_1.summary()
```

```
ization)                       512)

conv5_block2_2_relu (Activatio  (None, None, None,   0          ['conv5_block2_2_bn[0][0]']
n)                             512)

conv5_block2_3_conv (Conv2D)   (None, None, None,   1050624     ['conv5_block2_2_relu[0][0]']
                               2048)

conv5_block2_3_bn (BatchNormal  (None, None, None,   8192       ['conv5_block2_3_conv[0][0]']
ization)                       2048)

conv5_block2_add (Add)         (None, None, None,   0           ['conv5_block1_out[0][0]',
                               2048)                             'conv5_block2_3_bn[0][0]']

conv5_block2_out (Activation)  (None, None, None,   0           ['conv5_block2_add[0][0]']
                               2048)

conv5_block3_1_conv (Conv2D)   (None, None, None,   1049088     ['conv5_block2_out[0][0]']
                               512)

conv5_block3_1_bn (BatchNormal  (None, None, None,   2048       ['conv5_block3_1_conv[0][0]']
ization)                       512)

conv5_block3_1_relu (Activatio  (None, None, None,   0          ['conv5_block3_1_bn[0][0]']
n)                             512)

conv5_block3_2_conv (Conv2D)   (None, None, None,   2359808     ['conv5_block3_1_relu[0][0]']
                               512)

conv5_block3_2_bn (BatchNormal  (None, None, None,   2048       ['conv5_block3_2_conv[0][0]']
ization)                       512)

conv5_block3_2_relu (Activatio  (None, None, None,   0          ['conv5_block3_2_bn[0][0]']
n)                             512)

conv5_block3_3_conv (Conv2D)   (None, None, None,   1050624     ['conv5_block3_2_relu[0][0]']
                               2048)

conv5_block3_3_bn (BatchNormal  (None, None, None,   8192       ['conv5_block3_3_conv[0][0]']
ization)                       2048)

conv5_block3_add (Add)         (None, None, None,   0           ['conv5_block2_out[0][0]',
                               2048)                             'conv5_block3_3_bn[0][0]']

conv5_block3_out (Activation)  (None, None, None,   0           ['conv5_block3_add[0][0]']
                               2048)

global_average_pooling2d (Glob  (None, 2048)        0           ['conv5_block3_out[0][0]']
alAveragePooling2D)

predictions (Dense)            (None, 18)           36882       ['global_average_pooling2d[0][0]'
                                                                ]

==================================================================================================
Total params: 23,624,594
Trainable params: 23,571,474
Non-trainable params: 53,120
__________________________________________________________________________________________________
```

```
# Show the full index of the layers and choose where to start training from. You can see the convolution operations are in blocks. You want t
for index, layer in enumerate(model_1.layers):
  print(index, layer.name)
```

```
136 conv4_block6_2_conv
137 conv4_block6_2_bn
138 conv4_block6_2_relu
139 conv4_block6_3_conv
140 conv4_block6_3_bn
141 conv4_block6_add
142 conv4_block6_out
143 conv5_block1_1_conv
144 conv5_block1_1_bn
145 conv5_block1_1_relu
146 conv5_block1_2_conv
147 conv5_block1_2_bn
148 conv5_block1_2_relu
149 conv5_block1_0_conv
150 conv5_block1_3_conv
151 conv5_block1_0_bn
152 conv5_block1_3_bn
153 conv5_block1_add
154 conv5_block1_out
155 conv5_block2_1_conv
156 conv5_block2_1_bn
157 conv5_block2_1_relu
158 conv5_block2_2_conv
159 conv5_block2_2_bn
160 conv5_block2_2_relu
161 conv5_block2_3_conv
162 conv5_block2_3_bn
163 conv5_block2_add
164 conv5_block2_out
165 conv5_block3_1_conv
166 conv5_block3_1_bn
167 conv5_block3_1_relu
168 conv5_block3_2_conv
169 conv5_block3_2_bn
170 conv5_block3_2_relu
171 conv5_block3_3_conv
172 conv5_block3_3_bn
173 conv5_block3_add
174 conv5_block3_out
175 global_average_pooling2d
176 predictions
```

Fine-tuning the entire model on our dataset. This sometimes help to adapt it to our domain.

```
# Train the model
num_epochs = 5

# You may fine-tune the entire model, if you have time and compute resource. For that we don't need to freeze any layer.
for layer in model_1.layers: # These 2 lines are not really required since we are not indexing into the list of layers.
    layer.trainable = True

# Define the optimizer and set hyper-parameters
optimizer = tf.keras.optimizers.SGD(learning_rate=0.3, momentum=0.9)

# Specify loss based on the type of task. Here, we are dealing with a classification problem
model_1.compile(loss="sparse_categorical_crossentropy", optimizer=optimizer,
              metrics=["accuracy"])

history = model_1.fit(train_ds, validation_data = validation_ds,
                      epochs = num_epochs)
```

```
Epoch 1/5
358/358 [==============================] - 44s 57ms/step - loss: 3.5222 - accuracy: 0.1102 - val_loss: 2.9178 - val_accuracy: 0.1306
Epoch 2/5
358/358 [==============================] - 19s 52ms/step - loss: 2.7528 - accuracy: 0.1238 - val_loss: 2.7884 - val_accuracy: 0.1222
Epoch 3/5
358/358 [==============================] - 19s 52ms/step - loss: 2.7281 - accuracy: 0.1318 - val_loss: 2.7866 - val_accuracy: 0.1414
Epoch 4/5
358/358 [==============================] - 19s 53ms/step - loss: 2.6972 - accuracy: 0.1422 - val_loss: 2.7220 - val_accuracy: 0.1197
Epoch 5/5
358/358 [==============================] - 19s 53ms/step - loss: 2.6597 - accuracy: 0.1424 - val_loss: 2.7006 - val_accuracy: 0.1348
```

Train only the final dense layer by indexing and freezing the other layers.

```
# Train the model
num_epochs = 5

# You may only We want to train the final layer that we just created. For this we freeze the other layers.
for layer in model_1.layers[:-1]:
```

```
    layer.trainable = False

# Define the optimizer and set hyper-parameters
optimizer = tf.keras.optimizers.SGD(learning_rate=0.3, momentum=0.9)

# Specify loss based on the type of task. Here, we are dealing with a classification problem
model_1.compile(loss="sparse_categorical_crossentropy", optimizer=optimizer,
              metrics=["accuracy"])

history = model_1.fit(train_ds, validation_data = validation_ds,
                      epochs = num_epochs)
```

```
Epoch 1/5
358/358 [==============================] - 11s 23ms/step - loss: 2.5914 - accuracy: 0.1578 - val_loss: 2.5856 - val_accuracy: 0.1498
Epoch 2/5
358/358 [==============================] - 7s 20ms/step - loss: 2.5709 - accuracy: 0.1646 - val_loss: 2.6265 - val_accuracy: 0.1547
Epoch 3/5
358/358 [==============================] - 7s 19ms/step - loss: 2.5634 - accuracy: 0.1675 - val_loss: 2.5953 - val_accuracy: 0.1624
Epoch 4/5
358/358 [==============================] - 7s 20ms/step - loss: 2.5619 - accuracy: 0.1708 - val_loss: 2.5831 - val_accuracy: 0.1631
Epoch 5/5
358/358 [==============================] - 7s 19ms/step - loss: 2.5657 - accuracy: 0.1676 - val_loss: 2.5683 - val_accuracy: 0.1768
```

```
# Repeat for any other architecture that you choose from the API or create a custom architecture of your own.
```

Model 2

```
# Using pretrained ResNet152V2 from the Keras API: https://keras.io/api/applications/
# set include_top to false to remove the top layer with 1000 classes. Our dataset has 18 classes.
model_2_no_top = keras.applications.ResNet152V2(weights="imagenet", include_top=False)
model_2_with_top = keras.applications.ResNet152V2(weights="imagenet") # you may also set include_top=True
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet152v2_weights_tf_dim_ordering_tf_kernels
234545216/234545216 [==============================] - 13s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet152v2_weights_tf_dim_ordering_tf_kernels
242745792/242745792 [==============================] - 12s 0us/step
```

```
# Print the model architecture with the top layer included
model_2_with_top.summary()
```

```
 conv5_block3_2_relu (Activatio  (None, 7, 7, 512)   0           ['conv5_block3_2_bn[0][0]']
 n)

 conv5_block3_3_conv (Conv2D)   (None, 7, 7, 2048)   1050624     ['conv5_block3_2_relu[0][0]']

 conv5_block3_out (Add)         (None, 7, 7, 2048)   0           ['conv5_block2_out[0][0]',
                                                                  'conv5_block3_3_conv[0][0]']

 post_bn (BatchNormalization)   (None, 7, 7, 2048)   8192        ['conv5_block3_out[0][0]']

 post_relu (Activation)         (None, 7, 7, 2048)   0           ['post_bn[0][0]']

 avg_pool (GlobalAveragePooling  (None, 2048)        0           ['post_relu[0][0]']
 2D)

 predictions (Dense)            (None, 1000)         2049000     ['avg_pool[0][0]']

==================================================================================================
Total params: 60,380,648
Trainable params: 60,236,904
Non-trainable params: 143,744
```

```
# Print the model architecture without top layer
model_2_no_top.summary()
```

```
# Add new avg_pool_layer for aggregation of the "conv5_block3_out" layer. You can name it just "avg_pool", if you want.
avg_pool_layer = keras.layers.GlobalAveragePooling2D()(model_2_no_top.output)


# Verify the shape of the weights of the new avg_pool_layer conforms with the output of the "conv5_block3_out" layer
avg_pool_layer.shape
```

```
TensorShape([None, 2048])
```

```
# Add Fully connected output layer with the number of classes (18) in our dataset. Here we named it predictions, you may call it something el
output_layer = keras.layers.Dense(num_classes, activation="softmax", name="predictions")(avg_pool_layer)
output_layer.shape
```

```
TensorShape([None, 18])
```

```
model_2 = keras.models.Model(inputs=model_2_no_top.input, outputs=output_layer)


# Train the model
num_epochs = 5

# You may fine-tune the entire model, if you have time and compute resource. For that we don't need to freeze any layer.
for layer in model_2.layers: # These 2 lines are not really required since we are not indexing into the list of layers.
    layer.trainable = True

# Define the optimizer and set hyper-parameters
optimizer = tf.keras.optimizers.Adam(learning_rate=0.2, use_ema=True, ema_momentum=0.8)

# Specify loss based on the type of task. Here, we are dealing with a classification problem
model_2.compile(loss="sparse_categorical_crossentropy", optimizer=optimizer,
              metrics=["accuracy"])

history = model_2.fit(train_ds, validation_data = validation_ds,
                      epochs = num_epochs)
```

```
Epoch 1/5
358/358 [==============================] - 152s 171ms/step - loss: 5.4780 - accuracy: 0.1026 - val_loss: 41.6907 - val_accuracy: 0.1019
Epoch 2/5
358/358 [==============================] - 59s 164ms/step - loss: 2.8157 - accuracy: 0.1038 - val_loss: 3.9095 - val_accuracy: 0.1064
Epoch 3/5
358/358 [==============================] - 59s 164ms/step - loss: 2.8197 - accuracy: 0.1028 - val_loss: 3.5223 - val_accuracy: 0.1089
Epoch 4/5
358/358 [==============================] - 59s 163ms/step - loss: 2.8284 - accuracy: 0.0991 - val_loss: 2.9935 - val_accuracy: 0.1071
Epoch 5/5
358/358 [==============================] - 59s 164ms/step - loss: 2.8127 - accuracy: 0.1065 - val_loss: 5.4184 - val_accuracy: 0.0732
```

```
# Train the model
num_epochs = 5

# You may only We want to train the final layer that we just created. For this we freeze the other layers.
for layer in model_2.layers[:-1]:
    layer.trainable = False

# Define the optimizer and set hyper-parameters
optimizer = tf.keras.optimizers.Adam(learning_rate=0.2, use_ema=True, ema_momentum=0.8)

# Specify loss based on the type of task. Here, we are dealing with a classification problem
model_2.compile(loss="sparse_categorical_crossentropy", optimizer=optimizer,
              metrics=["accuracy"])

history = model_2.fit(train_ds, validation_data = validation_ds,
                      epochs = num_epochs)
```

```
Epoch 1/5
358/358 [==============================] - 28s 54ms/step - loss: 5.2733 - accuracy: 0.1030 - val_loss: 4.7947 - val_accuracy: 0.1113
Epoch 2/5
358/358 [==============================] - 17s 47ms/step - loss: 5.7284 - accuracy: 0.1071 - val_loss: 5.2256 - val_accuracy: 0.1103
Epoch 3/5
358/358 [==============================] - 17s 46ms/step - loss: 4.8148 - accuracy: 0.1130 - val_loss: 6.7038 - val_accuracy: 0.1103
Epoch 4/5
358/358 [==============================] - 17s 46ms/step - loss: 5.3419 - accuracy: 0.1063 - val_loss: 5.2191 - val_accuracy: 0.1148
Epoch 5/5
358/358 [==============================] - 17s 46ms/step - loss: 5.1645 - accuracy: 0.1138 - val_loss: 5.2425 - val_accuracy: 0.1169
```

Model 3

```
# Using pretrained EfficientNetB4 from the Keras API: https://keras.io/api/applications/
# set include_top to false to remove the top layer with 1000 classes. Our dataset has 18 classes.
model_3_no_top = keras.applications.EfficientNetB4(weights="imagenet", include_top=False)
model_3_with_top = keras.applications.EfficientNetB4(weights="imagenet") # you may also set include_top=True
```

```
Downloading data from https://storage.googleapis.com/keras-applications/efficientnetb4_notop.h5
71686520/71686520 [==============================] - 4s 0us/step
Downloading data from https://storage.googleapis.com/keras-applications/efficientnetb4.h5
78864416/78864416 [==============================] - 5s 0us/step
```

```
# Print the model architecture with the top layer included
model_3_with_top.summary()
```

```
 block7b_expand_bn (BatchNormal  (None, 12, 12, 2688  10752      ['block7b_expand_conv[0][0]']
 ization)                       )

 block7b_expand_activation (Act  (None, 12, 12, 2688  0          ['block7b_expand_bn[0][0]']
 ivation)                       )

 block7b_dwconv (DepthwiseConv2  (None, 12, 12, 2688  24192      ['block7b_expand_activation[0][0]
 D)                             )                                 ']

 block7b_bn (BatchNormalization  (None, 12, 12, 2688  10752      ['block7b_dwconv[0][0]']
 )                              )

 block7b_activation (Activation  (None, 12, 12, 2688  0          ['block7b_bn[0][0]']
 )                              )

 block7b_se_squeeze (GlobalAver  (None, 2688)        0           ['block7b_activation[0][0]']
 agePooling2D)

 block7b_se_reshape (Reshape)   (None, 1, 1, 2688)   0           ['block7b_se_squeeze[0][0]']

 block7b_se_reduce (Conv2D)     (None, 1, 1, 112)    301168      ['block7b_se_reshape[0][0]']

 block7b_se_expand (Conv2D)     (None, 1, 1, 2688)   303744      ['block7b_se_reduce[0][0]']

 block7b_se_excite (Multiply)   (None, 12, 12, 2688  0           ['block7b_activation[0][0]',
                                )                                 'block7b_se_expand[0][0]']

 block7b_project_conv (Conv2D)  (None, 12, 12, 448)  1204224     ['block7b_se_excite[0][0]']

 block7b_project_bn (BatchNorma  (None, 12, 12, 448)  1792       ['block7b_project_conv[0][0]']
 lization)

 block7b_drop (Dropout)         (None, 12, 12, 448)  0           ['block7b_project_bn[0][0]']

 block7b_add (Add)              (None, 12, 12, 448)  0           ['block7b_drop[0][0]',
                                                                  'block7a_project_bn[0][0]']

 top_conv (Conv2D)              (None, 12, 12, 1792  802816      ['block7b_add[0][0]']
                                )

 top_bn (BatchNormalization)    (None, 12, 12, 1792  7168        ['top_conv[0][0]']
                                )

 top_activation (Activation)    (None, 12, 12, 1792  0           ['top_bn[0][0]']
                                )

 avg_pool (GlobalAveragePooling  (None, 1792)        0           ['top_activation[0][0]']
 2D)

 top_dropout (Dropout)          (None, 1792)         0           ['avg_pool[0][0]']

 predictions (Dense)            (None, 1000)         1793000     ['top_dropout[0][0]']

==================================================================================================
Total params: 19,466,823
Trainable params: 19,341,616
Non-trainable params: 125,207
__________________________________________________________________________________________________
```

```
# Print the model architecture without top layer
model_3_no_top.summary()
```

```
Model: "efficientnetb4"
__________________________________________________________________________________________________
```

```
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_5 (InputLayer)           [(None, None, None,  0           []
                                 3)]

 rescaling (Rescaling)          (None, None, None,   0           ['input_5[0][0]']
                                3)

 normalization (Normalization)  (None, None, None,   7           ['rescaling[0][0]']
                                3)

 rescaling_1 (Rescaling)        (None, None, None,   0           ['normalization[0][0]']
                                3)

 stem_conv_pad (ZeroPadding2D)  (None, None, None,   0           ['rescaling_1[0][0]']
                                3)

 stem_conv (Conv2D)             (None, None, None,   1296        ['stem_conv_pad[0][0]']
                                48)

 stem_bn (BatchNormalization)   (None, None, None,   192         ['stem_conv[0][0]']
                                48)

 stem_activation (Activation)   (None, None, None,   0           ['stem_bn[0][0]']
                                48)

 block1a_dwconv (DepthwiseConv2  (None, None, None,   432        ['stem_activation[0][0]']
 D)                             48)

 block1a_bn (BatchNormalization  (None, None, None,   192        ['block1a_dwconv[0][0]']
 )                              48)

 block1a_activation (Activation  (None, None, None,   0          ['block1a_bn[0][0]']
 )                              48)

 block1a_se_squeeze (GlobalAver  (None, 48)          0           ['block1a_activation[0][0]']
 agePooling2D)

 block1a_se_reshape (Reshape)   (None, 1, 1, 48)     0           ['block1a_se_squeeze[0][0]']

 block1a_se_reduce (Conv2D)     (None, 1, 1, 12)     588         ['block1a_se_reshape[0][0]']

 block1a_se_expand (Conv2D)     (None, 1, 1, 48)     624         ['block1a_se_reduce[0][0]']

 block1a_se_excite (Multiply)   (None, None, None,   0           ['block1a_activation[0][0]',
                                48)                               'block1a_se_expand[0][0]']

 block1a_project_conv (Conv2D)  (None, None, None,   1152        ['block1a_se_excite[0][0]']
                                24)

 block1a_project_bn (BatchNorma  (None, None, None,   96         ['block1a_project_conv[0][0]']
 lization)                      24)

 block1b_dwconv (DepthwiseConv2  (None, None, None,   216        ['block1a_project_bn[0][0]']
 D)                             24)
```

```
# Add new avg_pool_layer for aggregation of the "conv5_block3_out" layer. You can name it just "avg_pool", if you want.
avg_pool_layer = keras.layers.GlobalAveragePooling2D()(model_3_no_top.output)


# Verify the shape of the weights of the new avg_pool_layer conforms with the output of the "conv5_block3_out" layer
avg_pool_layer.shape
```

```
TensorShape([None, 1792])
```

```
# Add Fully connected output layer with the number of classes (18) in our dataset. Here we named it predictions, you may call it something el
output_layer = keras.layers.Dense(num_classes, activation="softmax", name="predictions")(avg_pool_layer)
output_layer.shape
```

```
TensorShape([None, 18])
```

```
model_3 = keras.models.Model(inputs=model_3_no_top.input, outputs=output_layer)


# Train the model
num_epochs = 10

# You may fine-tune the entire model, if you have time and compute resource. For that we don't need to freeze any layer.
for layer in model_3.layers: # These 2 lines are not really required since we are not indexing into the list of layers.
    layer.trainable = True
```

```
# Define the optimizer and set hyper-parameters
optimizer = keras.optimizers.legacy.SGD(learning_rate=0.3, momentum=0.9, decay=0.01)

# Specify loss based on the type of task. Here, we are dealing with a classification problem
model_3.compile(loss="sparse_categorical_crossentropy", optimizer=optimizer,
              metrics=["accuracy"])

history = model_3.fit(train_ds, validation_data = validation_ds,
                      epochs = num_epochs)
```

```
Epoch 1/10
358/358 [==============================] - 50s 95ms/step - loss: 2.3894 - accuracy: 0.2862 - val_loss: 2.3015 - val_accuracy: 0.3119
Epoch 2/10
358/358 [==============================] - 31s 85ms/step - loss: 1.3510 - accuracy: 0.5931 - val_loss: 1.3132 - val_accuracy: 0.6069
Epoch 3/10
358/358 [==============================] - 31s 85ms/step - loss: 0.8637 - accuracy: 0.7479 - val_loss: 1.0549 - val_accuracy: 0.6955
Epoch 4/10
358/358 [==============================] - 31s 85ms/step - loss: 0.5927 - accuracy: 0.8266 - val_loss: 0.6623 - val_accuracy: 0.8232
Epoch 5/10
358/358 [==============================] - 31s 85ms/step - loss: 0.4046 - accuracy: 0.8801 - val_loss: 0.6232 - val_accuracy: 0.8341
Epoch 6/10
358/358 [==============================] - 31s 86ms/step - loss: 0.2756 - accuracy: 0.9179 - val_loss: 0.6099 - val_accuracy: 0.8484
Epoch 7/10
358/358 [==============================] - 31s 86ms/step - loss: 0.2043 - accuracy: 0.9361 - val_loss: 0.6662 - val_accuracy: 0.8460
Epoch 8/10
358/358 [==============================] - 31s 86ms/step - loss: 0.1404 - accuracy: 0.9567 - val_loss: 0.6917 - val_accuracy: 0.8523
Epoch 9/10
358/358 [==============================] - 31s 86ms/step - loss: 0.1031 - accuracy: 0.9680 - val_loss: 0.6494 - val_accuracy: 0.8663
Epoch 10/10
358/358 [==============================] - 30s 85ms/step - loss: 0.0854 - accuracy: 0.9738 - val_loss: 0.6579 - val_accuracy: 0.8694
```

```
# Train the model
num_epochs = 10

# You may only We want to train the final layer that we just created. For this we freeze the other layers.
for layer in model_3.layers[:-1]:
    layer.trainable = False

# Define the optimizer and set hyper-parameters
optimizer = keras.optimizers.legacy.SGD(learning_rate=0.3, momentum=0.9, decay=0.01)

# Specify loss based on the type of task. Here, we are dealing with a classification problem
model_3.compile(loss="sparse_categorical_crossentropy", optimizer=optimizer,
              metrics=["accuracy"])

history = model_3.fit(train_ds, validation_data = validation_ds,
                      epochs = num_epochs)
```

```
Epoch 1/10
358/358 [==============================] - 24s 36ms/step - loss: 0.0317 - accuracy: 0.9917 - val_loss: 0.6620 - val_accuracy: 0.8733
Epoch 2/10
358/358 [==============================] - 10s 28ms/step - loss: 0.0241 - accuracy: 0.9928 - val_loss: 0.6690 - val_accuracy: 0.8715
Epoch 3/10
358/358 [==============================] - 10s 28ms/step - loss: 0.0242 - accuracy: 0.9923 - val_loss: 0.6690 - val_accuracy: 0.8757
Epoch 4/10
358/358 [==============================] - 10s 27ms/step - loss: 0.0233 - accuracy: 0.9937 - val_loss: 0.6719 - val_accuracy: 0.8761
Epoch 5/10
358/358 [==============================] - 10s 28ms/step - loss: 0.0192 - accuracy: 0.9948 - val_loss: 0.6737 - val_accuracy: 0.8761
Epoch 6/10
358/358 [==============================] - 10s 28ms/step - loss: 0.0197 - accuracy: 0.9948 - val_loss: 0.6759 - val_accuracy: 0.8747
Epoch 7/10
358/358 [==============================] - 10s 28ms/step - loss: 0.0227 - accuracy: 0.9938 - val_loss: 0.6772 - val_accuracy: 0.8750
Epoch 8/10
358/358 [==============================] - 10s 28ms/step - loss: 0.0210 - accuracy: 0.9949 - val_loss: 0.6777 - val_accuracy: 0.8754
Epoch 9/10
358/358 [==============================] - 10s 28ms/step - loss: 0.0204 - accuracy: 0.9943 - val_loss: 0.6786 - val_accuracy: 0.8754
Epoch 10/10
358/358 [==============================] - 10s 28ms/step - loss: 0.0210 - accuracy: 0.9935 - val_loss: 0.6798 - val_accuracy: 0.8757
```

Conclusion:

I made use of the EfficientNetB4 pre-trained framework supplied by the API to achieve an accuracy of 0.99. By visiting the link 'https://kera applications/' and reading the chart, I found that this architecture i achieve ~96% accuracy using its pre-trained weights, with about 15.1ms inference step.

Conclusion:

I made use of the EfficientNetB4 pre-trained framework supplied by the Keras API to achieve an accuracy of 0.99. By visiting the link 'https://keras.io/api/applications/' and reading the chart, I found that this

```
In order to make the first layer fit the class-size, I took the top of
pre-trained model. SGD was the most efficient optimizer with softmax
activation. I found that increasing the epochs to 10 made for higher a
than using 5, which meant training the model for longer as I was only
~88% accuracy with 5 epochs.
```

architecture is able to achieve ~96% accuracy using its pre-trained weights, with about 15.1ms per inference step.

In order to make the first layer fit the class-size, I took the top off the the pre-trained model. SGD was the most efficient optimizer with softmax activation. I found that increasing the epochs to 10 made for higher accuracy than using 5, which meant training the model for longer as I wa

✓ 1m 54s completed at 11:23 PM