Assignment 2

```
import math, re, os
import numpy as np
import tensorflow as tf



from tensorflow.keras.applications import VGG16
from tensorflow.keras import Sequential
from keras.layers import *
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report,confusion_matrix
import seaborn as sns
tf.random.set_seed(20)
np.random.seed(20)
print("Tensorflow version " + tf.__version__)
try:
  tpu = tf.distribute.cluster_resolver.TPUClusterResolver()
  print('Running on TPU ', tpu.master())
except ValueError:
  tpu = None

if tpu:
  tf.config.experimental_connect_to_cluster(tpu)
  tf.tpu.experimental.initialize_tpu_system(tpu)
  strategy = tf.distribute.experimental.TPUStrategy(tpu)
else:
  strategy = tf.distribute.get_strategy()

print("REPLICAS: ", strategy.num_replicas_in_sync)
```

```
from kaggle_datasets import KaggleDatasets


GCS_DS_PATH = KaggleDatasets().get_gcs_path('tpu getting started')
print(GCS_DS_PATH)
TRAINING_FILENAMES
mydataset = load_dataset(TRAINING_FILENAMES, labeled=True)
mydataset
only10classes = ['pink primrose',
                 'snapdragon',
                 'purple coneflower',
                 'king protea',
                 'wild geranium',
                 'tiger lily',
                'peruvian lily',
                 'bird of paradise',
                 'monkshood',
                 'globe thistle',
          ]
len(only10classes)
def display_20_image(images):

    w = 10
    h = 10
    fig = plt.figure(figsize=(9, 13))
    columns = 4
    rows = 5
    # prep (x,y) for extra plotting
    xs = np.linspace(0, 2*np.pi, 60)  # from 0 to 2pi
    ys = np.abs(np.sin(xs))          # absolute of sine
    ax = []
```

```
    for i in range(columns*rows):
        img = np.random.randint(10, size=(h,w))
        ax.append( fig.add_subplot(rows, columns, i+1) )
        plt.imshow(images[i].reshape(images[i].shape))
    ax[2].plot(xs, 3*ys)
    ax[19].plot(ys**2, xs)

    plt.show()
mydataset
class0images = []
class1images = []
class2images = []
class3images = []
class4images = []
class5images = []
class6images = []
class7images = []
class8images = []
class9images = []

for images, labels in mydataset:  # only take first element of dataset
    #print(images.numpy().shape)
    if int(labels.numpy()) == 0:
        class0images.append(images.numpy())
    if int(labels.numpy()) == 10:
        class1images.append(images.numpy())
    if int(labels.numpy()) == 16:
        class2images.append(images.numpy())
    if int(labels.numpy()) == 12:
        class3images.append(images.numpy())
if int(labels.numpy()) == 4:
```

```
        class4images.append(images.numpy())
    if int(labels.numpy()) == 5:
        class5images.append(images.numpy())
    if int(labels.numpy()) == 17:
        class6images.append(images.numpy())
    if int(labels.numpy()) == 7:
        class7images.append(images.numpy())
    if int(labels.numpy()) == 8:
        class8images.append(images.numpy())
    if int(labels.numpy()) == 9:
        class9images.append(images.numpy())
print("done")
print(len(class0images))
print(len(class1images))
print(len(class2images))
print(len(class3images))
print(len(class4images))
print(len(class5images))
print(len(class6images))
print(len(class7images))
print(len(class8images))
print(len(class9images))
train=np.concatenate([class0images[:50],class1images[:50],class2images[:50],class3images[:50],class
4images[:50],class5images[:50],class6images[:50],class7images[:50],class8images[:50],class9images[
:50]])
train.shape
label0 =(len(class0images)*"pink_primrose ").split(' ')[:-1]
label1 =(len(class1images)*"snapdragon ").split(' ')[:-1]
label2 =(len(class2images)*"purple_coneflower ").split(' ')[:-1]
label3 =(len(class3images)*"king_protea ").split(' ')[:-1]
label4 =(len(class4images)*"wild_geranium ").split(' ')[:-1]
label5 =(len(class5images)*"tiger_lily ").split(' ')[:-1]
```

```
label6 =(len(class6images)*"peruvian_lily ").split(' ')[:-1]

label7 =(len(class7images)*"birdof_p_aradise ").split(' ')[:-1]

label8 =(len(class8images)*"monkshood ").split(' ')[:-1]

label9 =(len(class9images)*"globe_thistle ").split(' ')[:-1]

label0 =(len(class0images)*"0 ").split(' ')[:-1]

label1 =(len(class1images)*"1 ").split(' ')[:-1]

label2 =(len(class2images)*"2 ").split(' ')[:-1]

label3 =(len(class3images)*"3 ").split(' ')[:-1]

label4 =(len(class4images)*"4 ").split(' ')[:-1]

label5 =(len(class5images)*"5 ").split(' ')[:-1]

label6 =(len(class6images)*"6 ").split(' ')[:-1]

label7 =(len(class7images)*"7 ").split(' ')[:-1]

label8 =(len(class8images)*"8 ").split(' ')[:-1]

label9 =(len(class9images)*"9 ").split(' ')[:-1]

labels=np.concatenate([label0[:50],label1[:50],label2[:50],label3[:50],label4[:50],label5[:50],label6[:
50],label7[:50],label8[:50],label9[:50]])

labels

labels.shape

display_20_image(train)

from skimage.exposure import equalize_adapthist

from skimage.transform import resize

new_train=[]

for i in train:

    new_train.append((equalize_adapthist(i)))


display_20_image(new_train)

new_train=np.array(new_train)

mydataset=[]

X_train, X_test, y_train, y_test = train_test_split(new_train, labels, test_size=0.2, random_state=42)

Base_model = VGG16(include_top= False, weights='imagenet',input_shape=(192,192,3),
pooling='avg')

Base_model.trainable = False
```

```
data_augmentation = tf.keras.Sequential([
  RandomFlip("horizontal_and_vertical"),
  RandomRotation(0.1),
])

new_X=[]
new_y=[]
for i in range(len(X_train)):
    for j in range(8):
        new_X.append(data_augmentation(X_train[i]))
        new_y.append(y_train[i])
new_X=np.array(new_X)
new_X.shape
es = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=3)
model = Sequential(Base_model.layers)
model.add(Dense(1000,activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(1000,activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(1000,activation='relu'))


# adding prediction(softmax) layer
model.add(Dense(10,activation="softmax"))
model.summary()


model.compile(optimizer='adam', loss= 'categorical_crossentropy',
metrics=['accuracy',tf.keras.metrics.Recall(),tf.keras.metrics.Precision()])
y_tr=tf.keras.utils.to_categorical(new_y,num_classes=10)
y_ts=tf.keras.utils.to_categorical(y_test,num_classes=10)
```

```
history=model.fit(new_X,y_tr,epochs=20,batch_size=256,validation_split=0.2,callbacks=[es])

y_pred=model.predict(X_test

print(classification_report(y_test.astype('int'),np.argmax(y_pred,axis=1)))
y_pred=model.predict(new_X)

print(classification_report(np.array(new_y).astype('int'),np.argmax(y_pred,axis=1)))
import matplotlib.pyplot as plt

history_dict = history.history
loss_values = history_dict['loss']
val_loss_values = history_dict['val_loss']
accuracy = history_dict['accuracy']
val_accuracy = history_dict['val_accuracy']

epochs = range(1, len(loss_values) + 1)
fig, ax = plt.subplots(1, 2, figsize=(14, 6))
#
# Plot the model accuracy vs Epochs
#
ax[0].plot(epochs, accuracy, 'r', label='Training accuracy')
ax[0].plot(epochs, val_accuracy, 'b', label='Validation accuracy')
ax[0].set_title('Training & Validation Accuracy', fontsize=16)
ax[0].set_xlabel('Epochs', fontsize=16)
ax[0].set_ylabel('Accuracy', fontsize=16)
ax[0].legend()
ax[1].plot(epochs, loss_values, 'r', label='Training loss')
ax[1].plot(epochs, val_loss_values, 'b', label='Validation loss')
ax[1].set_title('Training & Validation Loss', fontsize=16)
ax[1].set_xlabel('Epochs', fontsize=16)
```

```
ax[1].set_ylabel('Loss', fontsize=16)
ax[1].legend()
import matplotlib.pyplot as plt

history_dict = history.history
loss_values = history_dict['recall']
val_loss_values = history_dict['val_recall']
accuracy = history_dict['precision']
val_accuracy = history_dict['val_precision']

epochs = range(1, len(loss_values) + 1)
fig, ax = plt.subplots(1, 2, figsize=(14, 6))
#
# Plot the model accuracy vs Epochs
#
ax[0].plot(epochs, accuracy, 'r', label='Training Recall')
ax[0].plot(epochs, val_accuracy, 'b', label='Validation Recall')
ax[0].set_title('Training & Validation Recall', fontsize=16)
ax[0].set_xlabel('Epochs', fontsize=16)
ax[0].set_ylabel('Recall', fontsize=16)
ax[0].legend()
ax[1].plot(epochs, loss_values, 'r', label='Training precision')
ax[1].plot(epochs, val_loss_values, 'b', label='Validation precision')
ax[1].set_title('Training & Validation precision', fontsize=16)
ax[1].set_xlabel('Epochs', fontsize=16)
ax[1].set_ylabel('precision', fontsize=16)
ax[1].legend()
cm=confusion_matrix(np.array(new_y).astype('int'),np.argmax(y_pred,axis=1))
sns.heatmap(cm,annot=True)
X_train, X_test, y_train, y_test = train_test_split(new_train, labels, test_size=0.2, random_state=42)
```

```
tf.random.set_seed(20)
np.random.seed(20)
model = Sequential(Base_model.layers)


model.add(Dense(1000,activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(1000,activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(1000,activation='relu'))


# adding prediction(softmax) layer
model.add(Dense(10,activation="softmax"))
model.summary()
model.compile(optimizer='adam', loss= 'categorical_crossentropy',
metrics=['accuracy',tf.keras.metrics.Recall(),tf.keras.metrics.Precision()])
y_tr=tf.keras.utils.to_categorical(y_train,num_classes=10)
y_ts=tf.keras.utils.to_categorical(y_test,num_classes=10)

history=model.fit(X_train,y_tr,epochs=20,batch_size=256,validation_split=0.2,callbacks=[es])

y_pred=model.predict(X_test)

print(classification_report(y_test.astype('int'),np.argmax(y_pred,axis=1)))
cm=confusion_matrix(y_test.astype('int'),np.argmax(y_pred,axis=1))
sns.heatmap(cm,annot=True)
X_train, X_test, y_train, y_test = train_test_split(train, labels, test_size=0.2, random_state=42)

tf.random.set_seed(20)
np.random.seed(20)
```

```
model = Sequential(Base_model.layers)



model.add(Dense(1000,activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(1000,activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(1000,activation='relu'))



# adding prediction(softmax) layer
model.add(Dense(10,activation="softmax"))
model.summary()
model.compile(optimizer='adam', loss= 'categorical_crossentropy',
metrics=['accuracy',tf.keras.metrics.Recall(),tf.keras.metrics.Precision()])
y_tr=tf.keras.utils.to_categorical(y_train,num_classes=10)
y_ts=tf.keras.utils.to_categorical(y_test,num_classes=10)

history=model.fit(X_train,y_tr,epochs=20,batch_size=256,validation_split=0.2,callbacks=[es])


y_pred=model.predict(X_test)


print(classification_report(y_test.astype('int'),np.argmax(y_pred,axis=1)))
```