-*- coding: utf-8 -*-
"""Project2_Task1.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/14LX2Nlk9_KlXtc8XP3mJJ8Vw4dD3NUsT

Student Names:
Rebecca Moore

1. Task 1 - basic CNN
2. Task 3 - VGG16 transfer Learning

Manish Reddy

1. Task 1 - basic CNN
2. Task 2 - Hash Filter and error function

Setup

Imports
"""

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import numpy as np
import pandas as pd
import os
import glob
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from google.colab import drive

from tensorflow.keras.utils import to_categorical
from PIL import Image
from tensorflow.keras.optimizers import Adam

from keras.models import Model
from tensorflow.keras.layers import BatchNormalization
from keras.layers.convolutional import Conv2D
from keras.layers.convolutional import MaxPooling2D
from keras.layers.core import Activation
from keras.layers.core import Dropout
from keras.layers.core import Lambda
from keras.layers.core import Dense
from keras.layers import Flatten
from keras.layers import Input
import tensorflow as tf
import plotly.graph_objects as go
from keras.callbacks import ModelCheckpoint

"""Mounting google drive because that is where the directory with the images is stored, set up mapping for gender and races"""

```
drive.mount('/content/drive')

data_name = '/content/drive/MyDrive/UTKFace'

image_size = (224, 224)
batch_size = 32

TT_SPLIT = 0.7
IM_WIDTH = 198
IM_HEIGHT = 198

dict_categories = {
    'race_id': {0: 'white', 1: 'black', 2: 'asian', 3: 'indian', 4: 'others'},
    'sex_id': {0: 'male',1: 'female'}}

dict_categories['sex_tmp'] = dict((g, i) for i, g in dict_categories['sex_id'].items())
dict_categories['race_tmp'] = dict((r, i) for i, r in dict_categories['race_id'].items())
```

"""Parse through the directory and image files to extract the labels and create a dataframe

"""

```
def parse(dataset_path, ext='jpg'):
    def extract(path):
        try:
            filename = os.path.split(path)[1]
            filename = os.path.splitext(filename)[0]
            age, gender, race, _ = filename.split('_')

            return int(age), dict_categories['sex_id'][int(gender)], dict_categories['race_id'][int(race)]
        except Exception as ex:
            return None, None, None

    files = glob.glob(os.path.join(dataset_path, "*.%s" % ext))
    records = []
    for file in files:
        info = extract(file)
        records.append(info)
    df = pd.DataFrame(records)
    df['file'] = files
    df.columns = ['age', 'gender', 'race', 'file']
    df = df.dropna()
    return df

df = parse(data_name)
df.head()
```

"""# Generator for CNN"""

```
class UtkGenerator():
    def __init__(self, df):
        self.df = df
    def split(self):
```

```
        p = np.random.permutation(len(self.df))
        spliter = int(len(self.df) * TT_SPLIT)
        train = p[:spliter]
        test = p[spliter:]

        spliter = int(spliter * TT_SPLIT)
        train, valid = train[:spliter], train[spliter:]

        self.df['sex_id'] = self.df['gender'].map(lambda sex: dict_categories['sex_tmp'][sex])
        self.df['race_id'] = self.df['race'].map(lambda race: dict_categories['race_tmp'][race])

        self.max_age = self.df['age'].max()

        return train, valid, test

    def preprocess_image(self, img_path):
        im = Image.open(img_path)
        im = im.resize((IM_WIDTH, IM_HEIGHT))
        im = np.array(im) / 255.0

        return im

    def generate_images(self, image_idx, is_training, batch_size=16):
        images, ages, races, sexs = [], [], [], []
        while True:
            for idx in image_idx:
                f = self.df.iloc[idx]

                age = f['age']
                race = f['race_categorical']
                sex = f['sex_categorical']
                file = f['file']

                im = self.preprocess_image(file)
                ages.append(age / self.max_age)
                races.append(to_categorical(race, len(dict_categories['race_categorical'])))
                sexs.append(to_categorical(sex, len(dict_categories['sex_id'])))
                images.append(im)

                if len(images) >= batch_size:
                    yield np.array(images), [np.array(ages), np.array(races), np.array(sexs)]
                    images, ages, races, sexs = [], [], [], []

            if not is_training:
                break

generator = UtkGenerator(df)
train, valid, test = generator.split()

"""# Set up the new model for the UTK dataframe off of the cats v dog CNN"""

class utkmodel():
    def hidden_layers(self, inputs):
        x = Conv2D(16, (3, 3), padding="same")(inputs)
        x = Activation("relu")(x)
        x = BatchNormalization(axis=-1)(x)
```

```
        x = MaxPooling2D(pool_size=(3, 3))(x)
        x = Dropout(0.25)(x)

        x = Conv2D(32, (3, 3), padding="same")(x)
        x = Activation("relu")(x)
        x = BatchNormalization(axis=-1)(x)
        x = MaxPooling2D(pool_size=(2, 2))(x)
        x = Dropout(0.25)(x)

        x = Conv2D(32, (3, 3), padding="same")(x)
        x = Activation("relu")(x)
        x = BatchNormalization(axis=-1)(x)
        x = MaxPooling2D(pool_size=(2, 2))(x)
        x = Dropout(0.25)(x)
        return x

    def race(self, inputs, num_races):
        x = self.hidden_layers(inputs)
        x = Flatten()(x)
        x = Dense(128)(x)
        x = Activation("relu")(x)
        x = BatchNormalization()(x)
        x = Dropout(0.5)(x)
        x = Dense(num_races)(x)
        x = Activation("softmax", name="race_output")(x)
        return x

    def sex(self, inputs, num_sexs=2):
        x = Lambda(lambda c: tf.image.rgb_to_grayscale(c))(inputs)
        x = self.hidden_layers(inputs)
        x = Flatten()(x)
        x = Dense(128)(x)
        x = Activation("relu")(x)
        x = BatchNormalization()(x)
        x = Dropout(0.5)(x)
        x = Dense(num_sexs)(x)
        x = Activation("sigmoid", name="sex_output")(x)
        return x

    def age(self, inputs):
        x = self.hidden_layers(inputs)
        x = Flatten()(x)
        x = Dense(128)(x)
        x = Activation("relu")(x)
        x = BatchNormalization()(x)
        x = Dropout(0.5)(x)
        x = Dense(1)(x)
        x = Activation("linear", name="age_output")(x)
        return x

    def compile_model(self, width, height, num_races):
        input_shape = (height, width, 3)
        x = Input(shape=input_shape)
        a_branch = self.age(x)
        r_branch = self.race(x, num_races)
        s_branch = self.sex(x)
```

```
        model = Model(inputs=x,
                      outputs = [a_branch, r_branch, s_branch],
                      name="utk_cnn_p2")

        return model

model = utkmodel().compile_model(IM_WIDTH, IM_HEIGHT, num_races= 5)

"""# Configure and Compile the model"""

learning_rate = 1e-4
epochs = 5
opt = Adam(learning_rate=learning_rate, decay=learning_rate / epochs)
model.compile(optimizer=opt,
              loss={
                  'age': 'mse',
                  'race': 'categorical_crossentropy',
                  'sex': 'binary_crossentropy'},
              loss_weights={
                  'age': 4.,
                  'race': 1.5,
                  'sex': 0.1},
              metrics={
                  'age': 'mae',
                  'race': 'accuracy',
                  'sex': 'accuracy'})

"""# Training"""

batch_size = 32
valid_batch_size = 32
train_gen = data_generator.generate_images(train, True, batch_size=batch_size)
valid_gen = data_generator.generate_images(valid, True, batch_size=valid_batch_size)

callbacks = [
    ModelCheckpoint("./model_checkpoint", monitor='val_loss')
]

history = model.fit(train_gen,
                    steps_per_epoch=len(train)//batch_size,
                    epochs=epochs,
                    callbacks=callbacks,
                    validation_data=valid_gen,
                    validation_steps=len(valid)//valid_batch_size)

# -*- coding: utf-8 -*-
"""Project2_Task2_MR.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1CU3r6KHirMgV2VLEsusVHzqmxlh0OQxj
"""

import pandas as pd
```

```
df = pd.read_csv("main_dataset.csv")

df

df.drop('Unnamed: 0', inplace=True, axis=1)
df.drop('Path', inplace=True, axis=1)

df

error_list = [[],[]]

error_list[0] = [[],[],[],[],[]]
error_list[1] = [[],[],[],[],[]]

for i in range(0,5):
    error_list[0][i] = [[],[],[],[],[],[],[],[],[],[]]
    error_list[1][i] = [[],[],[],[],[],[],[],[],[],[]]

for row in df.iterrows():
    for j in range(0,len(row)):
        print(row[j])
    break

for index, row in df.iterrows():
    print(type(row))
    break

import numpy as np
import os
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import cv2

sample_list = os.listdir('UTKFace')

m = []
for i in range(0,len(sample_list)):
    l = list(map(int,sample_list[i][:10].split('_')))
    m_i = [l[1],l[2],l[0]]
    m.append(m_i)

m

m.sort()

for i in m:
    if i[0]>1:
        print(i[0])

len(m)

m[23708-1]

m
```

```
c=0
for i in m:
    if i[0]==0:
        c+=1
    else:
        break
print(c)

for i in m:
    if i[2]>=0 and i[2]<10:
        error_list[i[0]][i[1]][0].append(i[2])
    elif i[2]>=10 and i[2]<20:
        error_list[i[0]][i[1]][1].append(i[2])
    elif i[2]>=20 and i[2]<30:
        error_list[i[0]][i[1]][2].append(i[2])
    elif i[2]>=30 and i[2]<40:
        error_list[i[0]][i[1]][3].append(i[2])
    elif i[2]>=40 and i[2]<50:
        error_list[i[0]][i[1]][4].append(i[2])
    elif i[2]>=50 and i[2]<60:
        error_list[i[0]][i[1]][5].append(i[2])
    elif i[2]>=60 and i[2]<70:
        error_list[i[0]][i[1]][6].append(i[2])
    elif i[2]>=70 and i[2]<80:
        error_list[i[0]][i[1]][7].append(i[2])
    elif i[2]>=80 and i[2]<90:
        error_list[i[0]][i[1]][8].append(i[2])
    elif i[2]>=90:
        error_list[i[0]][i[1]][9].append(i[2])

def fun_generate_age_bin_number(age):
    if age>=0 and age<10:
        return 0
    elif age>=10 and age<20:
        return 1
    elif age>=20 and age<30:
        return 2
    elif age>=30 and age<40:
        return 3
    elif age>=40 and age<50:
        return 4
    elif age>=50 and age<60:
        return 5
    elif age>=60 and age<70:
        return 6
    elif age>=70 and age<80:
        return 7
    elif age>=80 and age<90:
        return 8
    elif age>=90 :
        return 9

#actual_list = ["ACTUAL SEX", "ACTUAL RACE", "ACTUAL AGE"]
#pred_list = ["PREDICTED SEX", "PREDICTED RACE", "PREDICTED AGE"]

def error_fun(pred_list, actual_list):
```

```
    error_num = 0
    if pred_list[0]!=actual_list[0]:
            #error_num += 50
            # moving araound race in actual side pred
            #error_num += actual_list[1] *10
            #error_num += fun_generate_age_bin_number(actual_list[2])
            error_num += 50 + actual_list[1] *10 + fun_generate_age_bin_number(actual_list[2])
    elif pred_list[0] == actual_list[0] and pred_list[1] != actual_list[1]:
            error_num += (abs(pred_list[1] - actual_list[1]) -1) * 10 +
fun_generate_age_bin_number(actual_list[2])
    elif pred_list[0] == actual_list[0] and pred_list[1] == actual_list[1]:
        actual_age_bin = fun_generate_age_bin_number(actual_list[2])
        pred_age_bin = fun_generate_age_bin_number(pred_list[2])
        if actual_age_bin != pred_age_bin:
            error_num += abs(actual_age_bin - pred_age_bin)

    return error_num

pred_list = [0,0,57]
actual_list = [1,1,57]
error = error_fun(pred_list, actual_list)

error


"""Project2_Task3_RM.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1VGEv_CnBZuH-96KAr2rxAjOXclG-bE4L
"""

from google.colab import drive
import tensorflow as tf
import numpy as np

import os
import glob
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

"""# Preprocessing on images """

from tensorflow.keras.utils import to_categorical
import cv2
train_data_path = '/content/drive/MyDrive/UTKFace/'

x = []
y_a = []
y_g = []
y_r = []

root_path, dirs, files = next(os.walk(train_data_path))
```

```
for f in files:
    f_items = str(f).split('_')
    if len(f_items) == 4 and int(f_items[0]) <= 116:
        image = cv2.imread(os.path.join(root_path, f))
        image = cv2.resize(image, (224, 224))
        x.append(image)
        y_a.append(int(f_items[0]) - 1)
        y_g.append(int(f_items[1]))
        y_r.append(int(f_items[2]))

y_a = to_categorical(y_a, 116)
y_g = to_categorical(y_g, 2)
y_r = to_categorical(y_r, 5)

"""Takes the labels for each category and puts them into an array"""

x = np.array(x)
y_a = np.array(y_a)
y_g = np.array(y_g)
y_r = np.array(y_r)

train_index = int(len(x)*(1 - 0.1))

x_train = x[:train_index]
y_train_a = y_a[:train_index]
y_train_g = y_g[:train_index]
y_train_r = y_r[:train_index]

x_test = x[train_index:]
y_test_a = y_a[train_index:]
y_test_g = y_g[train_index:]
y_test_r = y_r[train_index:]

print(x.shape)
print(y_a.shape)
print(y_g.shape)
print(y_r.shape)

drive.mount('/content/drive')

import numpy as np
import pandas as pd
import os
import glob
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from google.colab import drive
data_name = '/content/drive/MyDrive/UTKFace'

TRAIN_TEST_SPLIT = 0.7
IM_WIDTH = IM_HEIGHT = 224

data_dict = {
    'race_id': {
        0: 'white',
```

```
        1: 'black',
        2: 'asian',
        3: 'indian',
        4: 'others'
    },
    'gender_id': {
        0: 'male',
        1: 'female'
    }
}

data_dict['gender_alias'] = dict((g, i) for i, g in data_dict['gender_id'].items())
data_dict['race_alias'] = dict((r, i) for i, r in data_dict['race_id'].items())

def parse_data(dataset_path, ext='jpg'):
    def from_file(path):
        try:
            filename = os.path.split(path)[1]
            filename = os.path.splitext(filename)[0]
            age, gender, race, _ = filename.split('_')

            return int(age), data_dict['gender_id'][int(gender)], data_dict['race_id'][int(race)]
        except Exception as ex:
            return None, None, None

    files = glob.glob(os.path.join(dataset_path, "*.%s" % ext))

    records = []
    for file in files:
        info = from_file(file)
        records.append(info)

    df = pd.DataFrame(records)
    df['file'] = files
    df.columns = ['age', 'gender', 'race', 'file']
    df = df.dropna()

    return df

df = parse_data(data_name)
df.head()

"""# Generates Images and splits up the processing into batches because the dataset is so large it will
crash without being split"""

from tensorflow.keras.utils import to_categorical
from PIL import Image
from tensorflow.keras.optimizers import Adam

class UtkGenerator():
    def __init__(self, df):
        self.df = df

    def generate_split_indexes(self):
        p = np.random.permutation(len(self.df))
        train_up_to = int(len(self.df) * TRAIN_TEST_SPLIT)
```

```
        train_idx = p[:train_up_to]
        test_idx = p[train_up_to:]

        train_up_to = int(train_up_to * TRAIN_TEST_SPLIT)
        train_idx, valid_idx = train_idx[:train_up_to], train_idx[train_up_to:]

        self.df['gender_id'] = self.df['gender'].map(lambda gender: data_dict['gender_alias'][gender])
        self.df['race_id'] = self.df['race'].map(lambda race: data_dict['race_alias'][race])

        self.max_age = self.df['age'].max()

        return train_idx, valid_idx, test_idx

    def preprocess(self, img_path):
        im = Image.open(img_path)
        im = im.resize((IM_WIDTH, IM_HEIGHT))
        im = np.array(im) / 255.0

        return im

    def generate_images(self, image_idx, is_training, batch_size=16):
        images, ages, races, genders = [], [], [], []
        while True:
            for idx in image_idx:
                person = self.df.iloc[idx]

                age = person['age']
                race = person['race_id']
                gender = person['gender_id']
                file = person['file']

                im = self.preprocess(file)

                ages.append(to_categorical(age, 116))
                races.append(to_categorical(race, 5))
                genders.append(to_categorical(gender, 2))
                images.append(im)

                # yielding condition
                if len(images) >= batch_size:
                    yield np.array(images), [np.array(ages), np.array(races), np.array(genders)]
                    images, ages, races, genders = [], [], [], []

            if not is_training:
                break

data_generator = UtkGenerator(df)
train_idx, valid_idx, test_idx = data_generator.generate_split_indexes()

from keras.applications.vgg16 import VGG16
from keras.applications.vgg16 import preprocess_input
from keras.layers import Dense, GlobalAveragePooling2D
from keras import  models

base_model = VGG16(include_top=True,
          input_tensor=None,
```

```
        weights='imagenet',
        pooling= None,
        input_shape = None)

base_model.summary()

from keras.models import Model
from keras.layers import Dense

base_output = base_model.output
output_a = Dense(116, activation='softmax', name='predications_age')(base_output)
output_g = Dense(2, activation='softmax', name='predications_gender')(base_output)
output_r = Dense(5, activation='softmax', name='predications_race')(base_output)

new_model = Model(inputs=base_model.input, outputs=[output_a, output_g, output_r],
name='transfer_learning_vgg16_utk')

new_model.summary()

batch_size=32
valid_batch_size = 32
epochs= 5
initial_lr=0.01
val_split=0.1
test_split=0.1
data_augmentation = True

from keras.callbacks import ModelCheckpoint
callbacks = [
   ModelCheckpoint("./model_checkpoint", monitor='val_loss')
]

from keras.callbacks import EarlyStopping

new_model.compile(
   optimizer='adam',
   loss={
        'predications_age': 'mse',
        'predications_race': 'categorical_crossentropy',
        'predications_gender': 'binary_crossentropy'},
     loss_weights={
        'predications_age': 4.,
        'predications_race': 1.5,
        'predications_gender': 0.1},
     metrics={
        'predications_age': 'mae',
        'predications_race': 'accuracy',
        'predications_gender': 'accuracy'},
   run_eagerly=True
)

train_gen = data_generator.generate_images(train_idx, is_training=True, batch_size=batch_size)
valid_gen = data_generator.generate_images(valid_idx, is_training=True, batch_size=valid_batch_size)

es = EarlyStopping(monitor='val_accuracy', mode='max', patience=5,  restore_best_weights=True)
```

```
history = new_model.fit(x_train, [y_train_a, y_train_g, y_train_r],
                        batch_size=batch_size, epochs= epochs,
                        callbacks=callbacks, validation_data=(x_test, [y_test_a, y_test_g, y_test_r]))
for layer in new_model.layers:
   print('layer_name:{0}=====trainable:{1}'.format(layer.name, layer.trainable))
new_model.summary()
```