Final project appendix

Rayna liu, Zheyue Wang

April 2021

1 Appendix

```
# I. ====================Load Libraries
    ======================================================
import os
import cv2
import keras
import random
import numpy as np
import pandas as pd
import seaborn as sn
import sklearn.metrics
import tensorflow as tf
import matplotlib.pyplot as plt
from keras.models import load_model
from keras.models import Sequential
from keras.optimizers import Adam, SGD
from keras.utils import to_categorical
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import ModelCheckpoint, EarlyStopping,
    ReduceLROnPlateau
from keras.layers import Dense, Dropout, BatchNormalization, Conv2D,
     Flatten, MaxPooling2D, AveragePooling2D, InputLayer
from datetime import datetime

# The time of program's execution
start_time = datetime.now()          # Duration: 1:30:21.060016


# II. ====================Load Dataset
    ======================================================
```

```
data = pd.read_csv('age_gender.csv')
data.info()     # Check the type of data and NaN
data.head()     # Show the first 5 row of dataset


# III. ====================Data Preprocessing (for Exploratory Data Analysis)==============
# Convert the type of pixels into np.array
data['pixels'] = data['pixels'].apply(lambda x: np.array(x.split(), dtype='float32'))
print('The pixels of each image is', len(data['pixels'][0]))
        # The length of pixels of each image: 2304 (48x48)
print('The image width is', int(np.sqrt(len(data['pixels'][0]))))
        # 48
print('The image height is', int(np.sqrt(len(data['pixels'][0]))))
        # 48


# IV. ==================================Description of dataset ==============================
data['Number of Image'] = 1

# Age
# Age 1-116, missing some age
Age_group = data.groupby(by='age')['Number of Image'].sum().reset_index(name='Number of Image')
# Plot the distribution of Age Group
plt.figure()
plt.bar(Age_group['age'],Age_group['Number of Image'])
plt.title('The Distribution of Age Group')
plt.xlabel('Age')
plt.ylabel('Number of Image')
plt.show()

# Ethnicity
# 0: White, 1:Black, 2:Asian, 3:Indian, 4:Others(like Hispanic, Latino,Middle Eastern)
Ethnicity_group = data.groupby(by='ethnicity')['Number of Image'].sum().reset_index(name='Number of Image')
# Plot the distribution of Ethnicity Group
label_ethnicity = ['White', 'Black', 'Asian', 'Indian', 'Others']
plt.figure()
plt.bar(Ethnicity_group['ethnicity'],Ethnicity_group['Number of Image'])
plt.title('The Distribution of Ethnicity Group')
plt.xlabel('Ethnicity')
```

```
plt.ylabel('Number of Image')
plt.xticks(Ethnicity_group['ethnicity'], label_ethnicity)
plt.show()

# Gender
# 0: Male, 1:Female
Gender_group = data.groupby(by='gender')['Number of Image'].sum().
    reset_index(name='Number of Image')
# Plot the distribution of Gender Group
label_gender = ['Male', 'Female']
plt.figure()
plt.bar(Gender_group['gender'],Gender_group['Number of Image'])
plt.title('The Distribution of Gender Group')
plt.xlabel('Gender')
plt.ylabel('Number of Image')
plt.xticks(Gender_group['gender'],label_gender)
plt.show()

# V. ----Show some sample images randomly------
def sample_images(data):
  fig, axs = plt.subplots(4,4,figsize=(16,16))
  df = data.sample(n=16).reset_index(drop=True)
  axs = axs.ravel()
  j = 0
  for i in range(len(df)):
    if j < 16:
      pixels = df['pixels'][i].reshape(48,48)
      axs[j].imshow(pixels, cmap='gray')
      axs[j].get_xaxis().set_ticks([])
      axs[j].get_yaxis().set_ticks([])
      axs[j].set_xlabel(f"Age: {df['age'].iloc[i]}, Ethnicity: {df['
    ethnicity'].iloc[i]}, Gender: {'F' if df['gender'].iloc[i]==1
    else 'M'}", fontsize=14)
      j += 1
    else:
      break
  fig.suptitle('Sample Images',fontsize=40)
  plt.show()

## Plot 16 images with label of age, enthnicity, and gender randomly
sample_images(data)


# VI. ================================ Two Models
    ================================================
# Setting up
```

```
SEED = 42
os.environ['PYTHONHASHSEED'] = str(SEED)
random.seed(SEED)
np.random.seed(SEED)
tf.random.set_seed(SEED)

# ---------------------------------- 1. Age Predict Model ----------------------------------------
# Hyper Parameters
LR_age = 0.001
N_EPOCHS_age = 200
BATCH_SIZE_age = 64
DROPOUT_age = 0.5

# Data Preparation
Image = np.array(data['pixels'].tolist())
Image = Image.reshape(Image.shape[0],48,48,1)      # Reshapes to (n_examples, n_channels, height_pixels, width_pixels)
Age_target = np.array(data['age']).reshape(-1,1)
Image = Image/255                                  # Normalizing pixels data
print('Input Image shape:',Image.shape)            # (23705, 48, 48, 1)
print('Age Target shape:',Age_target.shape)        # (23705, 1)
print()

# Split Data into training set (70%), validation set (15%), and testing set (15%)
age_x_train, age_x_test, age_y_train, age_y_test = train_test_split(Image, Age_target, test_size=0.3, random_state=SEED, shuffle=True)  # stratify=Age_target (ValueError: The least populated class in y has only 1 member, which is too few. The minimum number of groups for any class cannot be less than 2.)
age_x_val, age_x_test, age_y_val, age_y_test = train_test_split(age_x_test, age_y_test, test_size=0.5, random_state=SEED, shuffle=True)
print('Age train samples:',age_x_train.shape[0])       # 16593
print('Age validation samples:',age_x_val.shape[0])    # 3556
print()

# Image Augmentation  (It didn't do well in modeling)
## A technique to increase the diversity of your training set by applying random (but realistic) transformations such as image rotation and normalizing
#train_data_gen = ImageDataGenerator(rotation_range=40, width_shift_range=1,brightness_range=[0.8,1.2],zoom_range
```

```
    =[0.8,1.2], horizontal_flip=True, rescale=1/255)
#test_data_gen = ImageDataGenerator(rescale=1/255)
#age_train = train_data_gen.flow(age_x_train, age_y_train, seed=SEED
    , shuffle=False, batch_size=BATCH_SIZE_age)
#age_test = test_data_gen.flow(age_x_test, age_y_test, seed=SEED,
    shuffle=False, batch_size=BATCH_SIZE_age)

# Training Preparation
model_age = tf.keras.Sequential([
                                InputLayer(input_shape=(48,48,1)),
                                # output feature map size = ((
    image_size+2*pad)-(kernel_size-1)) / (max*stride)
                                Conv2D(64, (3, 3), strides=(1,1),
    padding='valid', activation="relu"),    # output(n_examples, 64,
    48, 48)
                                BatchNormalization(),
                                MaxPooling2D((1, 1)),
                                        # output(n_examples, 64,
    48, 48)

                                Conv2D(128, (2, 2), activation="
    relu"),                                 # output(n_examples,
    128, 46, 46)
                                BatchNormalization(),
                                MaxPooling2D((2, 2)),
                                        # output(n_examples, 128,
    23, 23)

                                Conv2D(256, (3, 3), activation="
    relu"),                                 # output(n_examples,
    256, 21, 21)
                                BatchNormalization(),
                                MaxPooling2D((2, 2)),
                                        # output(n_examples, 256,
    10, 10)

                                Conv2D(512, (3, 3), activation="
    relu"),                                 # output(n_examples,
    512, 8, 8)
                                BatchNormalization(),
                                MaxPooling2D((2, 2)),
                                        # output(n_examples, 512,
    4, 4)

                                Conv2D(512, (3, 3),strides=(1, 1),
    padding='same', activation="relu"),    # output(n_examples, 512,
```

```
4, 4)
                                Conv2D(512, (3, 3), strides=(1, 1), padding='same', activation="relu"),   # output(n_examples, 512, 4, 4)
                                Conv2D(512, (3, 3),strides=(1, 1), padding='same', activation="relu"),    # output(n_examples, 512, 4, 4)
                                Conv2D(512, (3, 3),strides=(1, 1), padding='same', activation="relu"),    # output(n_examples, 512, 4, 4)
                                Conv2D(512, (3, 3),strides=(1, 1), padding='same', activation="relu"),    # output(n_examples, 512, 4, 4)
                                Conv2D(512, (3, 3),strides=(1, 1), padding='same', activation="relu"),    # output(n_examples, 512, 4, 4)
                                Conv2D(512, (3, 3),strides=(1, 1), padding='same', activation="relu"),    # output(n_examples, 512, 4, 4)
                                MaxPooling2D((2, 2)),                                                     # output(n_examples, 512, 2, 2)

                                Flatten(),
                                Dense(512, activation="relu"),
                                Dense(4096, activation="relu"),
                                Dropout(DROPOUT_age),
                                Dense(5000, activation="relu"),
                                Dropout(DROPOUT_age),
                                BatchNormalization(),
                                Dense(1, activation="relu")
                               ])
# Age Model Summary
model_age.summary()

# Compiling the model
model_age.compile(optimizer=Adam(lr=LR_age), loss="mean_squared_error", metrics=["mae"])

# Setting callbacks
callbacks_age = [ModelCheckpoint("age_model_best.hdf5", monitor="val_loss", save_best_only=True),
                 EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=25, restore_best_weights=True),
                 ReduceLROnPlateau(factor=0.105, patience=15)]

```

```
# Training Loop
history_age = model_age.fit(age_x_train, age_y_train, batch_size=
    BATCH_SIZE_age, epochs=N_EPOCHS_age, validation_data=(age_x_val,
    age_y_val), callbacks=callbacks_age)
print()

# Evaluate Training History
plt.figure()
plt.plot(history_age.history['mae'])
plt.plot(history_age.history['val_mae'])
plt.title('Age model accuracy')
plt.ylabel('Mean Absolute Error')
plt.xlabel('Epoch')
plt.legend(['train mae', 'validation mae'], loc='upper right')
plt.show()

# Test MAE in age model
print("Test MAE of age:", model_age.evaluate(age_x_test,age_y_test)
    [1])
print()

#  ---------------------------------- 2. Ethnicity & Gender Predict
    Model-----------------------------
# Hyper Parameters
LR_eg = 0.0001
N_EPOCHS_eg = 60
BATCH_SIZE_eg = 512
DROPOUT_eg = 0.3
num_classes_eg = 7

# Data Preparation
Ethnicity_target = np.array(data['ethnicity']).reshape(-1,1)
Gender_target = np.array(data['gender']).reshape(-1,1)
Ethnicity_class = keras.utils.to_categorical(Ethnicity_target,5)
Gender_class = keras.utils.to_categorical(Gender_target,2)
# Concatenate Ethnicity and Gender target in one-hot-encoded (eg.
    White:0,Female:1 is [1,0,0,0,0,0,1])
Ethnicity_Gender_target = np.concatenate((Ethnicity_class,
    Gender_class),axis=1)     # one-hot-encoded
print('Input Image shape:',Image.shape)
                    # (23705, 48, 48, 1)
print('Ethnicity & Gender Target shape:',Ethnicity_Gender_target.
    shape)              # (23705, 7)

# Split Data into training set (70%) and validation set (15%)
    testing set (15%)
```

```
eg_x_train, eg_x_test, eg_y_train, eg_y_test = train_test_split(Image, Ethnicity_Gender_target, test_size=0.3, random_state=SEED, stratify=Ethnicity_Gender_target)
eg_x_val, eg_x_test, eg_y_val, eg_y_test = train_test_split(eg_x_test, eg_y_test, test_size=0.5, random_state=SEED, stratify=eg_y_test)
print('Ethnicity & Gender train sample:',eg_x_train.shape[0])
                    # 16593
print('Ethnicity & Gender validation sample:',eg_x_val.shape[0])
                    # 3556

# Image Augmentation (It didn't do well in modeling)
## A technique to increase the diversity of your training set by applying random (but realistic) transformations such as image rotation and normalizing
#train_data_gen = ImageDataGenerator(rotation_range=40, width_shift_range=0.2,brightness_range=[0.8,1.2],zoom_range=[0.8,1.2],rescale=1/255)
#test_data_gen = ImageDataGenerator(rescale=1/255)
#eg_train = train_data_gen.flow(eg_x_train, eg_y_train,batch_size=BATCH_SIZE_eg)
#eg_test = test_data_gen.flow(eg_x_test, eg_y_test,batch_size=BATCH_SIZE_eg)

# Training Preparation
model_eg = tf.keras.Sequential([
                                InputLayer(input_shape=(48,48,1)),
                                Conv2D(32,(3,3),activation="relu"),
                # output(n_examples, 32, 46, 46)
                                BatchNormalization(),
                                MaxPooling2D((2,2)),
                # output(n_examples, 32, 23, 23)

                                Conv2D(64,(3,3), activation="relu"),
                # output(n_examples, 64, 21, 21)
                                BatchNormalization(),
                                AveragePooling2D((2,2)),
                # output(n_examples, 64, 10, 10)

                                Conv2D(128,(3,3), activation="relu"),
                # output(n_examples, 128, 8, 8)
                                BatchNormalization(),

                                Conv2D(200,(3,3), activation="relu"),
                # output(n_examples, 200, 6, 6)
                                BatchNormalization(),
```

```

                                Conv2D(400,(3,3),activation='relu'), # output(n_examples, 400, 4, 4)
                                BatchNormalization(),

                                Conv2D(500,(3,3),activation="relu"), # output(n_examples, 500, 2, 2)

                                Flatten(),
                                Dense(500, activation="relu"),
                                Dropout(DROPOUT_eg),
                                BatchNormalization(),
                                Dense(4096,activation='relu'),
                                Dropout(DROPOUT_eg),
                                BatchNormalization(),
                                Dense(4096,activation="relu"),
                                Dropout(0.2),
                                BatchNormalization(),
                                Dense(4096,activation='relu'),
                                Dropout(0.2),
                                BatchNormalization(),
                                Dense(4096,activation='relu'),
                                Dropout(0.2),
                                BatchNormalization(),
                                Dense(4096,activation='relu'),
                                Dropout(0.2),
                                BatchNormalization(),
                                Dense(4096,activation='relu'),
                                Dropout(0.2),
                                BatchNormalization(),
                                Dense(4096,activation='relu'),
                                Dropout(0.2),
                                BatchNormalization(),
                                Dense(4096,activation='relu'),
                                Dropout(0.2),
                                BatchNormalization(),
                                Dense(4096,activation='relu'),
                                Dropout(0.2),
                                BatchNormalization(),
                                Dense(7, activation="sigmoid")
                              ])

# Ethnicity and Gender Model Summary
model_eg.summary()

# Compiling the model
```

```
model_eg.compile(optimizer='adam', loss=tf.keras.losses.
    BinaryCrossentropy(), metrics=["accuracy"])

# Setting callbacks
callbacks = [ModelCheckpoint("Eg_model.hdf5", monitor="val_accuracy"
    , save_best_only=True),
             EarlyStopping(monitor='val_accuracy', mode='max',
    verbose=1, patience=30, restore_best_weights=True),
             ReduceLROnPlateau(factor=0.105, patience=20)]

# Training Loop
history_eg = model_eg.fit(eg_x_train, eg_y_train, batch_size=
    BATCH_SIZE_eg, epochs=N_EPOCHS_eg, validation_data=(eg_x_test,
    eg_y_test), callbacks=callbacks)

# Evaluate Training History of Ethnicity & Gender Model
plt.figure()
plt.plot(history_eg.history['loss'])
plt.plot(history_eg.history['val_loss'])
plt.title('Ethnicity & Gender model accuracy')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['train loss', 'validation loss'], loc='upper right')
plt.show()

# Test accuracy in ethnicity_gender model
print("Test accuracy on ethnicity and gender:", 100*model_eg.
    evaluate(eg_x_test, eg_y_test)[1], "%")


# VII. ==============Age, Ethnicity, and Gender Prediction
    ================================================

# Predict age, ethnicity, and gender of all images
age_prediction = np.round(model_age.predict(Image)).astype(int)
ethnicity_gender_prediction = np.round(model_eg.predict(Image))
ethnicity_prediction = ethnicity_gender_prediction[:,:-2].argmax(
    axis=1)
gender_prediction = ethnicity_gender_prediction[:,-2:].argmax(axis
    =1)

# Create a DataFrame contain the prediction and predict error
data_pred = data.copy()
data_pred['age pred'] = age_prediction
data_pred['ethnicity pred'] = ethnicity_prediction
data_pred['gender pred'] = gender_prediction
```

```
data_pred['age error'] = data_pred['age']-data_pred['age pred']
data_pred['ethnicity error'] = data_pred['ethnicity']-data_pred['ethnicity pred']
data_pred['gender error'] = data_pred['gender']-data_pred['gender pred']

## Plot some images with real label and predict label randomly
def predict_images(data):
  fig, axs = plt.subplots(4,4,figsize=(16,16))
  df = data_pred.sample(n=16).reset_index(drop=True)
  axs = axs.ravel()
  j = 0
  for i in range(len(df)):
    if j < 16:
      pixels = df['pixels'][i].reshape(48,48)
      axs[j].imshow(pixels, cmap='gray')
      axs[j].get_xaxis().set_ticks([])
      axs[j].get_yaxis().set_ticks([])
      axs[j].set_xlabel('[Real]: Age:'+str(df['age'].iloc[i])+
                        ' Ethnicity:'+str(df['ethnicity'].iloc[i])+
                        ' Gender:'+str(df['gender'].iloc[i])+
                        '\n[Pred]: Age:'+str(df['age pred'].iloc[i])+
                        ' Ethnicity:'+str(df['ethnicity pred'].iloc[i])+
                        ' Gender:'+str(df['gender pred'].iloc[i]))
      j += 1
    else:
      break
  fig.suptitle('Sample Prediction Images',fontsize=40)
  plt.show()

predict_images(data_pred)

# VII. ====================Prediction Accuracy Analysis ==================================

# --------------------------------1. Age ------------------------------------------------
Age_pred_group = data_pred[data_pred['age']==data_pred['age pred']]
Age_correct_group = Age_pred_group.groupby(by='age')['Number of Image'].sum().reset_index(name='Sum of Correct Image')
age_result = pd.merge(Age_group, Age_correct_group, on='age')
age_result['Accuracy'] = 100*age_result['Sum of Correct Image']/age_result['Number of Image']
conditions = [(age_result['age'] <= 10),
```

```
                (age_result['age'] <= 20) & (age_result['age'] > 10),
                (age_result['age'] <= 30) & (age_result['age'] > 20),
                (age_result['age'] <= 40) & (age_result['age'] > 30),
                (age_result['age'] <= 50) & (age_result['age'] > 40),
                (age_result['age'] <= 60) & (age_result['age'] > 50),
                (age_result['age'] <= 70) & (age_result['age'] > 60),
                (age_result['age'] <= 80) & (age_result['age'] > 70),
                (age_result['age'] <= 90) & (age_result['age'] > 80),
                (age_result['age'] <= 100) & (age_result['age'] > 90),
                (age_result['age'] > 100)]
values = ['0-10', '11-20', '21-30', '31-40', '41-50', '51-60','61-70
    ','71-80','81-90','91-100','100+']
age_result['Age Groups'] = np.select(conditions, values)
Age_Group_avg = age_result.groupby(by='Age Groups')['Accuracy'].mean
    ().reset_index(name='Average of Accuracy')

# Plot the Distribution of Correctly Prediction of Whole Age Group
plt.figure()
plt.bar(Age_Group_avg['Age Groups'],Age_Group_avg['Average of
    Accuracy'],color=['palevioletred'])            # change x to
    age_result['age'] could see the distribution of correctly
    prediction of age
plt.title('The Distribution of Correctly Prediction of Whole Age
    Group')
plt.xlabel('Age Groups')
plt.ylabel('Percentage of Correct Prediction Image (%)')
for a,b in zip(Age_Group_avg['Age Groups'],Age_Group_avg['Average of
     Accuracy']):
    plt.text(a, b+0.05, '%.3f' % b, ha='center', va= 'bottom',
    fontsize=7)
plt.show()

# Histogram of prediction error of age
plt.figure()
plt.hist(data_pred['age error'],bins=40)
plt.title('The Histogram of Prediction Error of Age')
plt.xlabel('Error')
plt.ylabel('Number of Images')
plt.xlim(-20,20)
plt.show()


# -------------------- 2. Ethnicity
    ----------------------------------
Ethnicity_pred_group = data_pred[data_pred['ethnicity']==data_pred['
    ethnicity pred']]
```

```
Ethnicity_correct_group = Ethnicity_pred_group.groupby(by='ethnicity')['Number of Image'].sum().reset_index(name='Sum of Correct Image')
ethnicity_result = pd.merge(Ethnicity_group, Ethnicity_correct_group, on='ethnicity')
ethnicity_result['Accuracy'] = 100*ethnicity_result['Sum of Correct Image']/ethnicity_result['Number of Image']

# Plot the Distribution of Correctly Prediction of Ethnicity Group
label_ethnicity = ['White', 'Black', 'Asian', 'Indian', 'Others']
plt.figure()
plt.bar(ethnicity_result['ethnicity'],ethnicity_result['Accuracy'])
plt.title('The Distribution of Correctly Prediction of Ethnicity Group')
plt.xlabel('Ethnicity')
plt.ylabel('Percentage of Correct Prediction Image (%)')
plt.xticks(ethnicity_result['ethnicity'], label_ethnicity)
plt.show()

# Plot Confusion Matrix of Ethnicity
cm = sklearn.metrics.confusion_matrix(data_pred['ethnicity'], data_pred['ethnicity pred'])
df_cm = pd.DataFrame(cm, range(5), range(5))
plt.figure(figsize=(10,7))
sn.set(font_scale=1.4)          # for label size
sn.heatmap(df_cm, annot=True, annot_kws={"size": 16},cmap="Blues")
     # font size
plt.xlabel("Predicted Ethnicity")
plt.ylabel("True Ethnicity")
plt.title('Confusion Matrix of Ethnicity')
plt.show()


# -------------------------3. Gender
    ------------------------------------
Gender_pred_group = data_pred[data_pred['gender']==data_pred['gender
     pred']]
Gender_correct_group = Gender_pred_group.groupby(by='gender')['
    Number of Image'].sum().reset_index(name='Sum of Correct Image')
gender_result = pd.merge(Gender_group, Gender_correct_group, on='
    gender')
gender_result['Accuracy'] = 100*gender_result['Sum of Correct Image'
    ]/gender_result['Number of Image']

# Plot the Distribution of Correctly Prediction of Gender Group
label_gender = ['Male', 'Female']
```

```
plt.figure(figsize=(10,7))
plt.bar(gender_result['gender'],gender_result['Accuracy'])
plt.title('The Distribution of Correctly Prediction of Gender Group')
plt.xlabel('Gender')
plt.ylabel('Percentage of Correct Prediction Image (%)')
plt.xticks(gender_result['gender'],label_gender)
plt.show()

# Plot Confusion Matrix of Gender
cm = sklearn.metrics.confusion_matrix(data_pred['gender'],data_pred['gender pred'])
df_cm = pd.DataFrame(cm, range(2), range(2))
plt.figure(figsize=(10,7))
sn.set(font_scale=1.4)          # for label size
sn.heatmap(df_cm, annot=True, annot_kws={"size": 16},cmap="Blues")      # font size
plt.xlabel("Predicted Gender")
plt.ylabel("True Gender")
plt.title('Confusion Matrix of Gender')
plt.show()

# VIII.================================ Error Analysis ==============================

# Show Incorrectly Age Images
def Incorrectly_images_age(x_test,y_test):
    fig, axs = plt.subplots(4,4,figsize=(16,16))
    age_y_pred = np.round(model_age.predict(x_test))
    axs = axs.ravel()
    x_test = x_test * 255
    j = 0
    for i in range(len(x_test)):
        if j < 16:
            if age_y_pred[i] != y_test[i]:
                pixels = x_test[i].reshape((48,48))
                axs[j].imshow(pixels, cmap='gray')
                axs[j].get_xaxis().set_ticks([])
                axs[j].get_yaxis().set_ticks([])
                axs[j].set_xlabel('Real Age:'+str(y_test[i])+
                                  ' Pred Age:'+str(age_y_pred[i]), fontsize=14)
                j += 1
        else:
            break
    fig.suptitle('Incorrectly Age Images',fontsize=40)
```

```
    plt.show()

Incorrectly_images_age(age_x_test,age_y_test)

# Show Incorrectly Ethnicity and Gender Images
eg_y_pred = model_eg.predict(eg_x_test)
eg_y_pred = np.round(eg_y_pred)
ethnicity_pred = eg_y_pred[:,:-2].argmax(axis=1)
gender_pred = eg_y_pred[:,-2:].argmax(axis=1)

ethnicity_y_test = eg_y_test[:,:-2].argmax(axis=1)
gender_y_test = eg_y_test[:,-2:].argmax(axis=1)

def Incorrectly_images_eg(x_test,y_test):
    fig, axs = plt.subplots(4,4,figsize=(16,16))
    axs = axs.ravel()
    x_test = x_test * 255
    j = 0
    for i in range(len(x_test)):
        if j < 16:
            if ethnicity_pred[i] != ethnicity_y_test[i] and
   gender_pred[i] != gender_y_test[i]:
                pixels = x_test[i].reshape((48,48))
                axs[j].imshow(pixels, cmap='gray')
                axs[j].get_xaxis().set_ticks([])
                axs[j].get_yaxis().set_ticks([])
                axs[j].set_xlabel('[Real] Ethnicity:'+str(
   ethnicity_y_test[i])+
                                  ' Gender:'+str(gender_y_test[i])+
                                  '\n [Pred] Ethnicity:'+str(
   ethnicity_pred[i])+
                                  ' Gender:'+str(gender_pred[i]),
   fontsize=14)
                j += 1
        else:
            break
    fig.suptitle('Incorrectly Images',fontsize=40)
    plt.show()

Incorrectly_images_eg(eg_x_test,eg_y_test)

# IX.================================ Application
   ======================
# Predict the age,ethnicity, and gender of the image which randomly
   choose from Google
# Load the Test_Images Folder
```

```
direction = os.getcwd() + "/Test_Images/"         # get the image
    path
image_path = []
for im in [f for f in os.listdir(direction)]:
    image_path.append(direction+im)

# Reshape the Images
RESIZE_TO = 48
x = []
pixels = []
for png in image_path:
    pixel = cv2.imread(png)
    pixels.append(pixel)
    image = cv2.resize(pixel, (RESIZE_TO, RESIZE_TO))
    image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    x.append(image)
x = np.array(x)
x = x.reshape(len(x), RESIZE_TO, RESIZE_TO, 1)
x = x / 255

# Age Prediction
age_pred_apply = np.round(model_age.predict(x))

# Ethnicity & Gender Prediction
ethnicity_gender_pred_apply = np.round(model_eg.predict(x))
ethnicity_pred_apply = ethnicity_gender_pred_apply[:,:-2].argmax(
    axis=1)
gender_pred_apply = ethnicity_gender_pred_apply[:,-2:].argmax(axis
    =1)

ethnicity_pred_label = []
for i in range(len(ethnicity_pred_apply)):
    if ethnicity_prediction[i] == 0:
        ethnicity_pred_label.append('White')
    elif ethnicity_prediction[i] == 1:
        ethnicity_pred_label.append('Black')
    elif ethnicity_prediction[i] == 2:
        ethnicity_pred_label.append('Asian')
    elif ethnicity_prediction[i] == 3:
        ethnicity_pred_label.append('Indian')
    else:
        ethnicity_pred_label.append('Other')

gender_pred_label = []
for i in range(len(gender_pred_apply)):
    if gender_prediction[i] == 0:
```

```
        gender_pred_label.append('M')
    else:
        gender_pred_label.append('F')

# Plot the Prediction
fig, axs = plt.subplots(3,2,figsize=(16,16))
axs = axs.ravel()
for i in range(len(image_path)):
    image = pixels[i]
    axs[i].imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
    axs[i].get_xaxis().set_ticks([])
    axs[i].get_yaxis().set_ticks([])
    axs[i].set_xlabel('[Pred]: Age:'+str(age_pred_apply[i])+
                      ' Ethnicity:'+str([ethnicity_pred_label[i]])+
                      ' Gender:'+str([gender_pred_label[i]]), fontsize
   =18)
fig.suptitle('Prediction Images from Google Image',fontsize=40)
plt.show()

# End time
end_time = datetime.now()
print('Duration: {}'.format(end_time - start_time))
```