Bahria University
Discovering Knowledge

Fall 2022
Department of Computer Science
Bahria University, Karachi Campus

INTRODUCTION TO DATA SCIENCE LAB [CSC-487]

Project Name: Predicting Age/Gender from Facial Images

SEMESTER PROJECT

Maximum Marks: 30

Submission Due Date: 17th January 2023

Sr.no	Name	Enrolment	Semester
01	Muhammad Anas Inam	02-134201-072	6-B
02	Hafiz Muhammad Usama Azlan	02-134201-022	6-B

Name	Designation
Ms. Soomal Fatima	Course Instructor
Ms. Salas Akbar	Lab Engineer

Acknowledgement

I would like to express my special thanks of gratitude to my professor, Ms. Soomal Fatima as well as our lab instructor, Ms. Salas Akbar, who gave me the golden opportunity to do this wonderful project on the topic **Gender/Age Prediction**, which also helped me in doing a lot of Research and I came to know about so many new things, I am thankful to them.

Contents

1. Chapter 1

1.1. Problem Statement

We dealt with the issue of determining a person's age and gender from an image. This app could be used on dating sites and social media, to access content that is restricted to certain ages, and to self-check out at grocery stores when buying alcohol.

The technique for gender and age detection is CNN. The convolution layer is the layer where the filter is applied to our input image to extract or detect its features. A filter is applied to the image multiple times and creates a feature map which helps in classifying the input image.

2. Chapter 2

2.1. Literature Review

Face Recognition Performance: Role of Demographic Information

It was published on 08 October 2012, by a group of researchers under the organization *"IEEE"*. This paper studies the influence of demographics on the performance of face recognition algorithms. The recognition accuracies of six different face recognition algorithms (three commercial, two nontrainable, and one trainable) are computed on a large-scale gallery that is partitioned so that each partition consists entirely of specific demographic cohorts.

Face Recognition and Age Estimation implications of Changes in Facial Features: A Critical Review Study

It was published on 18 May 2018, by *Rasha Ragheb Atallah, Amirrudin Kamsin, Maizatul Akmar Ismail, Sherin Ali Abdelrahman and Saber Zerdoumi*. Facial features are considered as one of the important personal characteristics. This can be used in

many applications, such as face recognition and age estimation. The value of these applications depends in several areas, such as security applications, law enforcement applications, and attendance systems.

Age estimation via face images: A Survey

Facial aging adversely impacts performance of face recognition and face verification and authentication using facial features. This stochastic personalized inevitable process poses dynamic theoretical and practical challenge to the computer vision and pattern recognition community. Age estimation is labelling a face image with exact real age or age group.

Convolutional Neural Networks for Age Classification

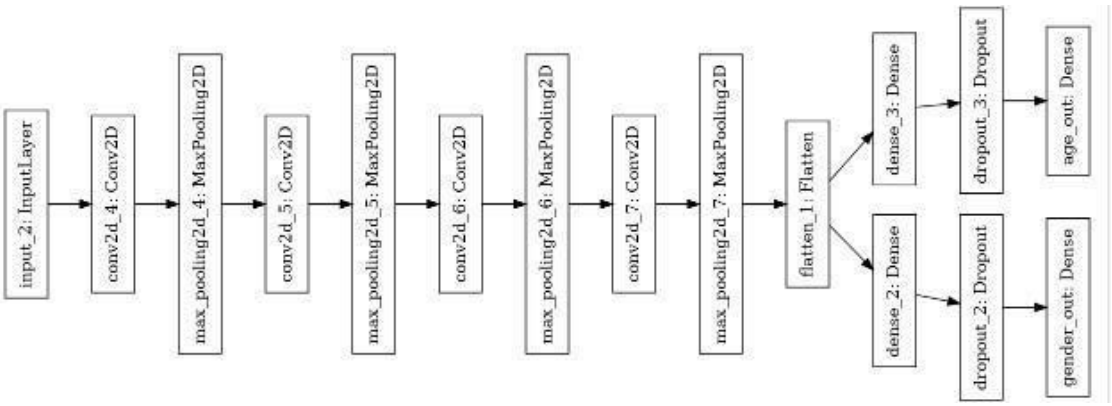
This paper focuses on the problem of gender and age classification for an image. I build off of previous work [12] that has developed efficient, accurate architectures for these tasks and aim to extend their approaches in order to improve results. The first main area of experimentation in this project is modifying some previously published, effective architectures used for gender and age classification [12]. My attempts include reducing the number of parameters (in the style of [19]), increasing the depth of the network, and modifying the level of dropout used. These modifications actually ended up causing system performance to decrease (or at best, stay the same) as compared with the simpler architecture I began with. This verified suspicions I had that the tasks of age and gender classification are more prone to over-fitting than other types of classification.

3. Chapter 3

3.1. Methodology

The architecture that was used for gender/age classification tasks is described below -

- Basic idea was to have a block of convolutional and pooling layers, that help the model learn features and representations, followed by a set of FC layers, which are used to classify. My model took an input image of size (256,256,1). It was a grayscale image.
- To predict both age and gender at once, I used a multi-output classification technique. This involved having a common part of feature extraction for both age and gender, but separate FC layers. The branching point is just after features have been extracted for both. I kept the feature extraction part common for both age and gender classification, because as per researchers, features are along the same lines for predicting age and gender



4. Chapter 4

4.3. Dataset:

https://www.kaggle.com/datasets/jangedoo/utkface-new

4.4. Code Snippet

- **Importing Modules:**

```
import pandas as pd
import numpy as np
import os #for file processing
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
from tqdm.notebook import tqdm #progress bar while loading the images
warnings.filterwarnings('ignore') #this will give us clean result
%matplotlib inline

#For neural network:
import tensorflow as tf
from keras.preprocessing.image import load_img #this will load the image directly into numpy array
from keras.models import Sequential, Model
from keras.layers import Dense, Conv2D, Dropout, Flatten, MaxPooling2D, Input
```

- **Loading Dataset:**

```
BASE_DIR = '/kaggle/input/utkface-new/UTKFace/'
```

```
# labels - age, gender, ethnicity
image_paths = []
age_labels = []
gender_labels = []

for filename in tqdm(os.listdir(BASE_DIR)): #this will iterate all the files inside this particular directory
    image_path = os.path.join(BASE_DIR, filename)
    temp = filename.split('_')
    age = int(temp[0])
    gender = int(temp[1])
    image_paths.append(image_path)
    age_labels.append(age)
    gender_labels.append(gender)
```

```
100%  23708/23708 [00:00<00:00, 142609.88it/s]
```

The os and tqdm libraries to iterate through all the files in a directory specified by the variable BASE_DIR. For each file, it gets the file path by joining the BASE_DIR variable with the file name, and then it splits the file name by '_' and assigns the first part to the variable age and the second part to the variable gender. Then it appends the image path, age, and gender to the respective lists (image_paths, age_labels, gender_labels). The tqdm library is used to provide a progress bar, so that you can see the progress of the loop as it iterates through all the files.

```
# convert to dataframe
df = pd.DataFrame()
df['image'], df['age'], df['gender'] = image_paths, age_labels, gender_labels
df.head()
```

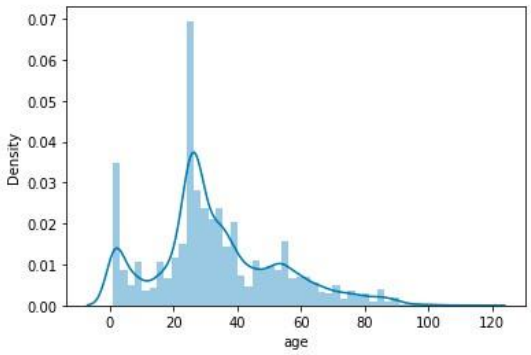
	image	age	gender
0	/kaggle/input/utkface-new/UTKFace/26_0_2_20170...	26	0
1	/kaggle/input/utkface-new/UTKFace/22_1_1_20170...	22	1
2	/kaggle/input/utkface-new/UTKFace/21_1_3_20170...	21	1
3	/kaggle/input/utkface-new/UTKFace/28_0_0_20170...	28	0
4	/kaggle/input/utkface-new/UTKFace/17_1_4_20170...	17	1

- **Exploratory Data Analysis:**

```
from PIL import Image
img = Image.open(df['image'][5])
plt.axis('off')
plt.imshow(img);
```

```
sns.distplot(df['age'])
```

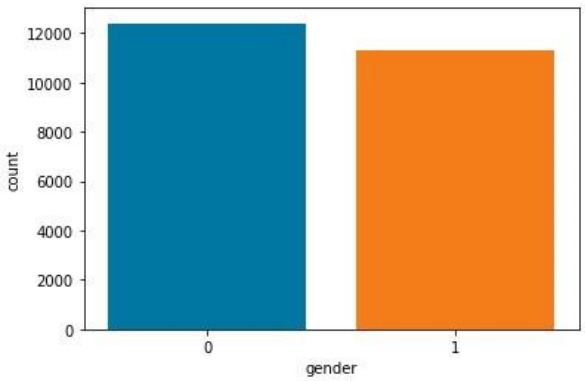
```
<AxesSubplot:xlabel='age', ylabel='Density'>
```



Above code uses the seaborn library to create a histogram and kernel density estimate (KDE) plot for the "age" column of a DataFrame called "df". The sns.distplot() function will show a histogram of the data in the "age" column, with a line over it repres enting the estimated probability density function (PDF) of the data. The plot allows you to see the distribution of the data and get an idea of the

```
sns.countplot(df['gender'])
```

```
<AxesSubplot:xlabel='gender', ylabel='count'>
```



underlying probability distribution of the variable being plotted.

Above code uses the seaborn library to create a bar plot for the "gender" column of a DataFrame called "df". The sns.countplot() function will show a count of the number of occurrences for each unique value in the "gender" column. It is useful to visualize the frequency distribution of categorical data. If the data is binary, it will show the count of occurence of each class. This plot allows you to see how many observations of each category are present in the data.

```
# to display grid of images
plt.figure(figsize=(20, 20))
files = df.iloc[0:25]

for index, file, age, gender in files.itertuples(): #itertuples() method will return an iterator yielding a named tuple for each row
    plt.subplot(5, 5, index+1)
    img = load_img(file)
    img = np.array(img)
    plt.imshow(img)
    plt.title(f"Age: {age} Gender: {gender_dict[gender]}")
    plt.axis('off')
```

Above code uses the matplotlib library to display a grid of images. It uses the iloc method to select the first 25 rows of the DataFrame "df" and then iterates over those rows using the itertuples() method. It loads each image from the file path, converts it to a numpy array, and then displays it using the plt.imshow() function. It also sets the title of each subplot to show the "age" and "gender" value of the image. The plt.axis('off') is used to remove the x and y axis from the subplot.
It is assumed that the DataFrame 'df' has columns 'file', 'age', 'gender' where file column contains the path of the image and 'gender' is a integer value.
It is also assumed that 'gender_dict' is a dictionary which contains the mapping of integers to the actual gender values.

- **Feature Extraction:**

```
def extract_features(images):
    features = []
    for image in tqdm(images):
        img = load_img(image, grayscale=True)
        img = img.resize((128, 128), Image.ANTIALIAS)
        img = np.array(img)
        features.append(img)

    #converting features into numpy array because neural network can handle only numpy array
    features = np.array(features)
    # ignore this step if using RGB
    features = features.reshape(len(features), 128, 128, 1)
    return features
```

Above code defines a function called "extract_features()" that takes a list of image file paths as input, loads each image, resizes it to 128x128 pixels, converts it to grayscale, and appends it to a list called "features". It then converts the "features" list to a NumPy array and reshapes it to have a shape of (number of images, 128, 128, 1). The tqdm library is used to provide a progress bar so that you can see the progress of the loop as it iterates through all the images.

```
X = extract_features(df['image'])
```

```
100% 23708/23708 [03:37<00:00, 118.34it/s]
```

+ Code + Markdown

```
X.shape
```

```
(23708, 128, 128, 1)
```

```
# normalize the images
X = X/255.0
```

```
y_gender = np.array(df['gender'])
y_age = np.array(df['age'])
```

```
input_shape = (128, 128, 1)
```

- **Model Creation:**

```
inputs = Input((input_shape))
# convolutional layers
conv_1 = Conv2D(32, kernel_size=(3, 3), activation='relu') (inputs)
maxp_1 = MaxPooling2D(pool_size=(2, 2)) (conv_1)
conv_2 = Conv2D(64, kernel_size=(3, 3), activation='relu') (maxp_1)
maxp_2 = MaxPooling2D(pool_size=(2, 2)) (conv_2)
conv_3 = Conv2D(128, kernel_size=(3, 3), activation='relu') (maxp_2)
maxp_3 = MaxPooling2D(pool_size=(2, 2)) (conv_3)
conv_4 = Conv2D(256, kernel_size=(3, 3), activation='relu') (maxp_3)
maxp_4 = MaxPooling2D(pool_size=(2, 2)) (conv_4)

flatten = Flatten() (maxp_4)

# fully connected layers
dense_1 = Dense(256, activation='relu') (flatten)
dense_2 = Dense(256, activation='relu') (flatten)

dropout_1 = Dropout(0.3) (dense_1)
dropout_2 = Dropout(0.3) (dense_2)

output_1 = Dense(1, activation='sigmoid', name='gender_out') (dropout_1)
output_2 = Dense(1, activation='relu', name='age_out') (dropout_2)

model = Model(inputs=[inputs], outputs=[output_1, output_2])

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

This code defines a Neural Network model using the Keras library. The model starts with an input layer that has a shape specified by the "input_shape" variable. It then has a series of convolutional layers, max pooling layers, and fully connected layers.

```
# plot the model
from tensorflow.keras.utils import plot_model
plot_model(model)
```

- **Train The model:**

```
history = model.fit(x=X, y=[y_gender, y_age], batch_size=32, epochs=30, validation_split=0.2)
```

This code uses the fit () function from the Keras library to train the model defined previously. The function takes the following arguments:

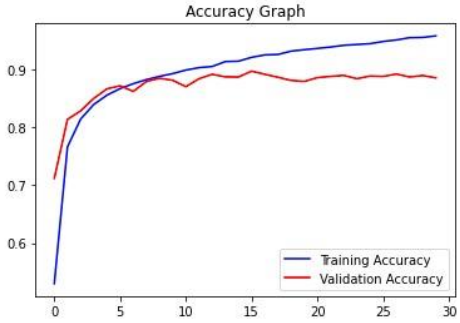
1. x: The input data, in this case, is the variable X which contains the features extracted from images
2. y: The target data, in this case, is a list of two variables, y_gender, and y_age, which contain the corresponding gender and age labels for the images.

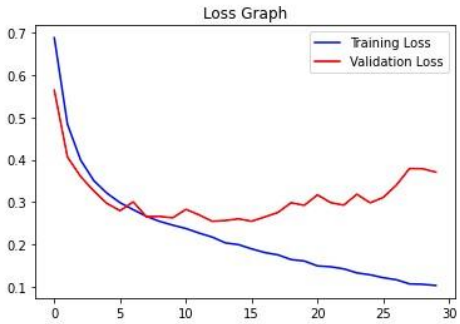
3. batch_size: The number of samples per gradient update. The model will update the weights after processing each batch of 32 images.
4. epochs: The number of times the model will iterate over the entire dataset. The model will be trained for 30 iterations or epochs.
5. validation_split: The fraction of the data that will be used for validation. In this case, 20% of the data will be used for validation and 80% for training.

- **Plotting The Result:**

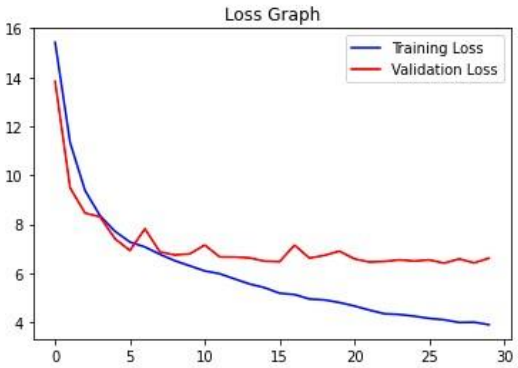
```
# plot results for gender
acc = history.history['gender_out_accuracy']
val_acc = history.history['val_gender_out_accuracy']
epochs = range(len(acc))

plt.plot(epochs, acc, 'b', label='Training Accuracy')
plt.plot(epochs, val_acc, 'r', label='Validation Accuracy')
plt.title('Accuracy Graph')
plt.legend()
plt.figure()

loss = history.history['gender_out_loss']
val_loss = history.history['val_gender_out_loss']

plt.plot(epochs, loss, 'b', label='Training Loss')
plt.plot(epochs, val_loss, 'r', label='Validation Loss')
plt.title('Loss Graph')
plt.legend()
plt.show()
```

This code uses the matplotlib library to create two plots, one for accuracy and one for loss, for the gender output of the model. The accuracy and loss values are taken from the history object returned by the model. fit () function.






```
# plot results for age
loss = history.history['age_out_loss']
val_loss = history.history['val_age_out_loss']
epochs = range(len(loss))

plt.plot(epochs, loss, 'b', label='Training Loss')
plt.plot(epochs, val_loss, 'r', label='Validation Loss')
plt.title('Loss Graph')
plt.legend()
plt.show()
```



This code uses the matplotlib library to create a plot for the loss of the age output of the model. The loss values are taken from the history object returned by the model.fit() function.

- **Prediction With Test Data:**

```
image_index = 3000
print("Original Gender:", gender_dict[y_gender[image_index]], "Original Age:", y_age[image_index])
# predict from model
pred = model.predict(X[image_index].reshape(1, 128, 128, 1))
pred_gender = gender_dict[round(pred[0][0][0])]
pred_age = round(pred[1][0][0])
print("Predicted Gender:", pred_gender, "Predicted Age:", pred_age)
plt.axis('off')
plt.imshow(X[image_index].reshape(128, 128), cmap='gray');
```

```
Original Gender: Male Original Age: 28
Predicted Gender: Male Predicted Age: 32
```

5. Chapter 6

5.1. Conclusion

This project offers a simple age and gender prediction solution. This also applies to other industrial or medical issues. After classification, we may use these images to tackle various classification and detection issues by feeding them into more advanced ML models. Although CNN served this objective, there are even more effective methods that can be employed in this situation.

5.2. Future Work

Following are the implementations of gender and age prediction which can be useful:

- Buying Alcohols
- Dating Sites

6. Chapter 6

6.1. References

https://www.pyimagesearch.com/2018/06/04/keras-multiple-outputs-and- multiplelosses/

https://medium.com/datadriveninvestor/introduction-to-how-cnns-work-77e0e4cde99b

https://blog.keras.io/building-powerful-image-classification-models-using-verylittle-data.html

https://paperswithcode.com/task/age-estimation https://keras.io/api/layers/