**DEEP LEARNING**

**MINI PROJECT**

AGE AND GENDER DETECTION USING CNN (Adience Dataset)

**AIM:**

The aim of this project is to develop a deep learning model using CNNs to accurately detect and estimate the age and gender of individuals from facial images. The model will be trained and evaluated using the Adience Dataset, a large dataset containing facial images with associated age and gender labels.

**ABSTRACT:**

In this project, we propose a deep learning-based approach for age and gender detection using the Adience Dataset. We utilize Convolutional Neural Networks (CNNs) to extract features from facial images and predict the age and gender of individuals. The project involves data collection, preprocessing, model development, training, and evaluation. Our goal is to achieve high accuracy in age and gender prediction, which has various potential applications in fields like demographics analysis, marketing, and user profiling.

**PROCEDURE**

1. **DATA COLLECTION:**

Acquire the Adience Dataset, which contains a diverse set of facial images with age and gender labels. Split the dataset into training, validation, and test sets for model development and evaluation.

1. **DATA PREPROCESSING**:

Resize and standardize all facial images to a consistent resolution (e.g., 128x128 pixels). Normalize pixel values to the range [0, 1].Augment the training data with techniques like rotation, flipping, and brightness adjustments to improve model generalization.

1. **MODEL DEVELOPMENT:**

Design a CNN architecture for age and gender prediction.Create two output layers in the model: one for age estimation (regression) and another for gender classification (binary classification).Train the model on the training data using appropriate loss functions and optimizers.

1. **MODEL PREDICTION:**

Use the trained model to make predictions on the test dataset.For age estimation, predict a continuous value representing the estimated age.For gender classification, predict binary labels (e.g., Male or Female).

1. **EVALUATION:**

Evaluate the model's performance using appropriate evaluation metrics:Age Estimation: Use metrics like Mean Absolute Error (MAE) to measure the difference between predicted and actual ages.Gender Classification: Calculate accuracy, precision, recall, and F1-score.Generate visualizations (e.g., confusion matrices) to assess model performance.

1. **ESTIMATION:**

Utilize the trained model to estimate the age and gender of new, unseen facial images.Provide results in a user-friendly format, such as displaying the estimated age and gender alongside the input image.

**IMPLEMENTATION**

PROGRAM:

from tensorflow import keras

import pandas as pd

import numpy as np

import os

import matplotlib.pyplot as plt

import seaborn as sns

import warnings

from tqdm.notebook import tqdm

warnings.filterwarnings('ignore')

%matplotlib inline

import tensorflow as tf

from tensorflow.keras.preprocessing.image import load\_img

from keras.models import Sequential, Model

from keras.layers import Dense, Conv2D, Dropout, Flatten, MaxPooling2D, Input

BASE\_DIR = '/content/drive/MyDrive/ageGender/crop\_part1

**# labels - age, gender, ethnicity**

image\_paths = []

age\_labels = []

gender\_labels = []

for filename in tqdm(os.listdir(BASE\_DIR)):

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)

OUTPUT:



PROGRAM:

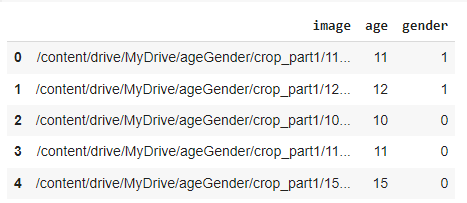
**# convert to dataframe**

df = pd.DataFrame()

df['image'], df['age'], df['gender'] = image\_paths, age\_labels, gender\_labels

df.head()

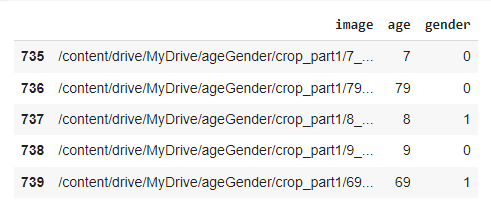
OUTPUT:



PROGRAM:

df.tail()

OUTPUT:



PROGRAM:

**# map labels for gender**

gender\_dict = {0:'Male', 1:'Female'}

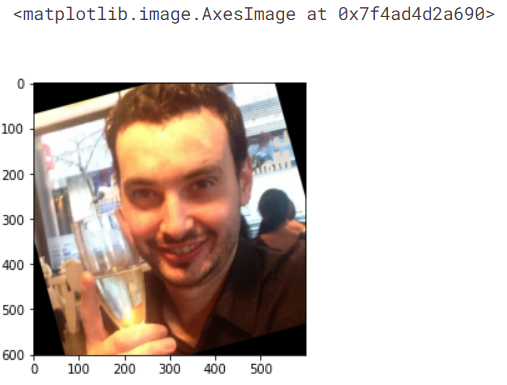
from PIL import Image

img = Image.open(df['image'][2])

plt.axis('off')

plt.imshow(img);

OUTPUT:



PROGRAM:

images = []

for \_ in range(16):

sample\_num = random.randint(0, len(total\_data))

im\_name = total\_data.iloc[sample\_num].original\_image

im\_path = os.path.join(data\_parent, 'faces',str(total\_data.iloc[sample\_num].user\_id), 'coarse\_tilt\_aligned\_face.' + str(total\_data.iloc[sample\_num].face\_id) + '.' + im\_name)

image = cv2.imread(im\_path)

age = total\_data.iloc[sample\_num].age

gender = total\_data.iloc[sample\_num].gender

n\_col = 4

n\_rows = 4

images.append((image, age, gender))

fig, axs = plt.subplots(ncols=n\_col, nrows=n\_rows, figsize=(30,30))

count = 0

for i in range(n\_rows):

for j in range(n\_col):

axs[i][j].imshow(cv2.cvtColor(images[count][0], cv2.COLOR\_BGR2RGB))

axs[i][j].set\_title(f'Age: {images[count][1]}, Gender: {images[count][2]}')

count+=1

plt.show()

OUTPUT:



PROGRAM:

age\_mapping = [('(0, 2)', '0-2'), ('2', '0-2'), ('3', '0-2'), ('(4, 6)', '4-6'), ('(8, 12)', '8-13'), ('13', '8-13'), ('22', '15-20'), ('(8, 23)','15-20'), ('23', '25-32'), ('(15, 20)', '15-20'), ('(25, 32)', '25-32'), ('(27, 32)', '25-32'), ('32', '25-32'), ('34', '25-32'), ('29', '25-32'), ('(38, 42)', '38-43'), ('35', '38-43'), ('36', '38-43'), ('42', '48-53'), ('45', '38-43'), ('(38, 43)', '38-43'), ('(38, 42)', '38-43'), ('(38, 48)', '48-53'), ('46', '48-53'), ('(48, 53)', '48-53'), ('55', '48-53'), ('56', '48-53'), ('(60, 100)', '60+'), ('57', '60+'), ('58', '60+')]

age\_mapping\_dict = {each[0]: each[1] for each in age\_mapping}

drop\_labels = []#contains the indexes that has None (None labels)

for idx, each in enumerate(total\_data.age):

if each == 'None':

drop\_labels.append(idx)

else:

total\_data.age.loc[idx] = age\_mapping\_dict[each]

total\_data = total\_data.drop(labels=drop\_labels, axis=0) #droped None values (axis = 0 , removing rows)

total\_data.age.value\_counts(dropna=False)

OUTPUT:

/opt/conda/lib/python3.7/site-packages/pandas/core/indexing.py:1637: SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy

self.\_setitem\_single\_block(indexer, value, name)

25-32 5296

38-43 2776

0-2 2509

8-13 2292

4-6 2140

15-20 1792

48-53 916

60+ 901

Name: age, dtype: int64

PROGRAM:

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)

features = np.array(features)

**# ignore this step if using RGB**

features = features.reshape(len(features), 128, 128, 1)

return features

X = extract\_features(df['image'])

OUTPUT:



X.shape



**# normalize the images**

X = X/255.0

y\_gender = np.array(df['gender'])

y\_age = np.array(df['age'])

input\_shape = (128, 128, 1)

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)

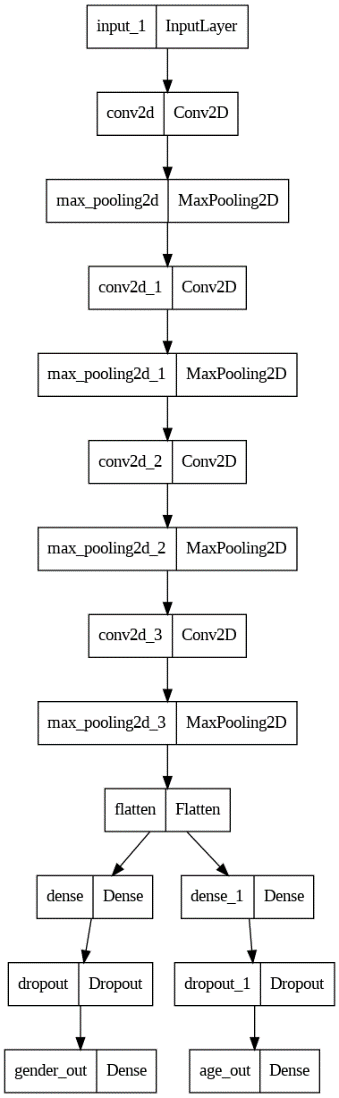
model = Model(inputs=[inputs], outputs=[output\_1, output\_2])

model.compile(loss=['binary\_crossentropy', 'mae'], optimizer='adam', metrics=['accuracy'])

# plot the model

from tensorflow.keras.utils import plot\_model

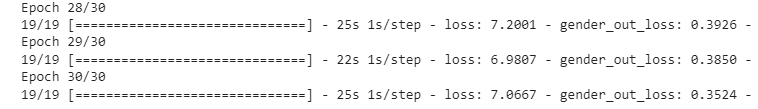
plot\_model(model)



**# train model**

history = model.fit(x=X, y=[y\_gender, y\_age], batch\_size=32, epochs=30, validation\_split=0.2)

OUTPUT:



PROGRAM:

**# 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']

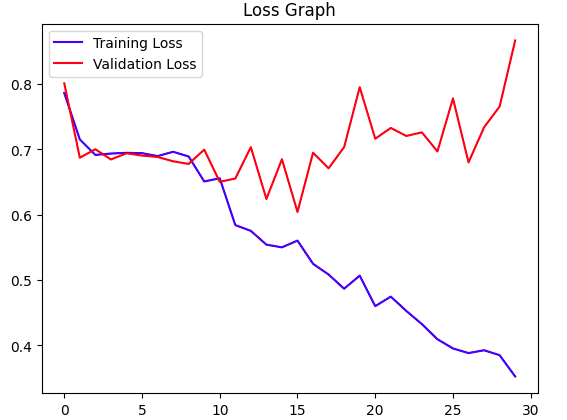
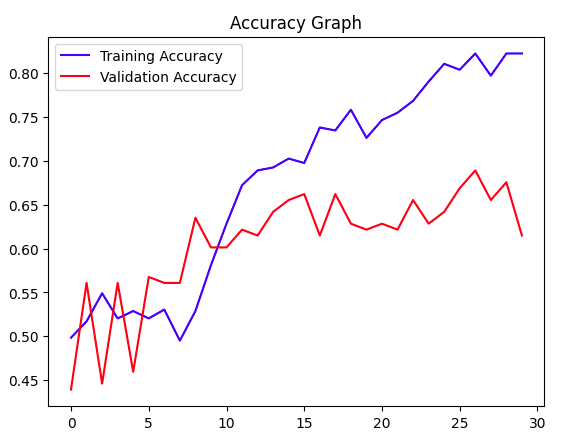
plt.plot(epochs, loss, 'b', label='Training Loss')

plt.plot(epochs, val\_loss, 'r', label='Validation Loss')

plt.title('Loss Graph')

plt.legend()

plt.show()



PROGRAM:

**# 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')

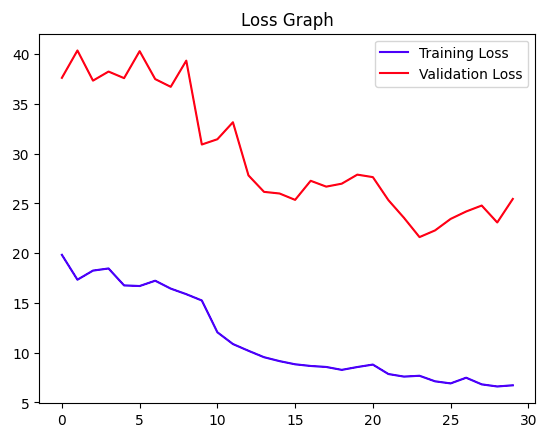
plt.plot(epochs, val\_loss, 'r', label='Validation Loss')

plt.title('Loss Graph')

plt.legend()

plt.show()

OUTPUT:



PROGRAM:

image\_index = 100

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])]

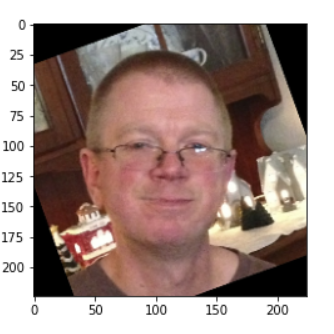
pred\_age = round(pred[1][0][0])

print("Predicted Gender:", pred\_gender, "Predicted Age:", pred\_age)

plt.axis('off')

plt.imshow(X[image\_index].reshape(128, 128), cmap='gray');

OUTPUT:



PROGRAM:

image\_index = 30

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');

OUTPUT:

