PROJECT REPORT

GENDER DETECTION AND AGE PREDICTION

SUBMITTED IN COMPLETE FULFILLMENT OF THE
REQUIREMENTS FOR THE AWARD OF THE DEGREE
OF
BACHELOR OF TECHNOLOGY
IN

COMPUTER ENGINEERING

SUBMITTED BY

PRAKARSH PANWAR

2K21/CO/334

PRITESH DAS

2K21/CO/348

Under the supervision of

PROF. MS. DIPIKA JAIN

COMPILER DESIGN(CO302)

DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Delhi-110042

GENDER DETECTION AND AGE PREDICTION

Table of Contents

1. Introduction

The last decade or two has witnessed a boom of images. With the increasing omnipresence of cameras and with the advent of selfies, the number of facial images available in the world has increased exponentially. Consequently, there has been a growing interest in automatic age prediction and gender detection of a person using facial images. We in our project focus on this challenging problem.

1.1 Background and Significance

Gender Detection & Age Prediction using Deep Learning could be used as a useful tool, to estimate age of individual humans quickly through use of this application. Age and gender prediction has become one of the more recognized fields in deep learning, due to the increased rate of image uploads on the internet in today's data driven world. Humans are inherently good at determining one's gender, recognizing each other and making judgements about ethnicity but age estimation still remains a formidable problem.

1.2 Methodology and Application

This software project leverages use of Deep Learning concepts such as Conventional Neural Networks (CNN), & Machine Learning concepts such as Logistic Regression. Such applications can be used for social media targeted advertisements for every age group, businesses can identify demographics of their consumers, and also for security identification purposes.

2. Data Exploration

2.1 Data Source

Dataset used in this project is the UTKFace dataset which is a large-scale face dataset with long age span (range from 0 to 116 years old). The dataset consists of over 20,000 face images with annotations of age, gender, and ethnicity. The images cover large variation in: -

- **Pose**
- **Facial Expression**
- **Illumination**
- **Occlusion**
- **Resolution**

The link for the dataset is: https://www.kaggle.com/datasets/jangedoo/utkface-new

This dataset could be used on a variety of tasks, such as: -

- **Face Detection**
- **Age Estimation** (From 0 to 116 years)
- **Age Progression/Regression**
- **Landmark**
- **Localization**
- **Gender Detection** (0: Male, 1: Female)

2.2 Data Preprocessing

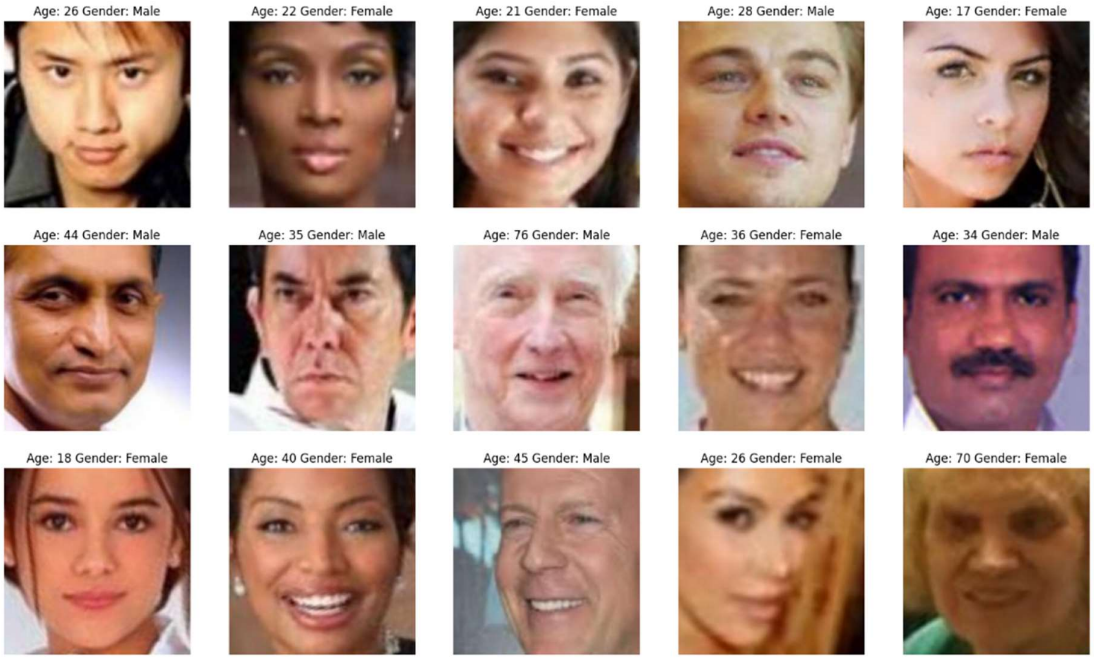
These are the steps followed to preprocess the data: -

- Determining Gender and Age using File Name since file format for all images are as follows: **{AGE}_{GENDER}_{XYZ}**

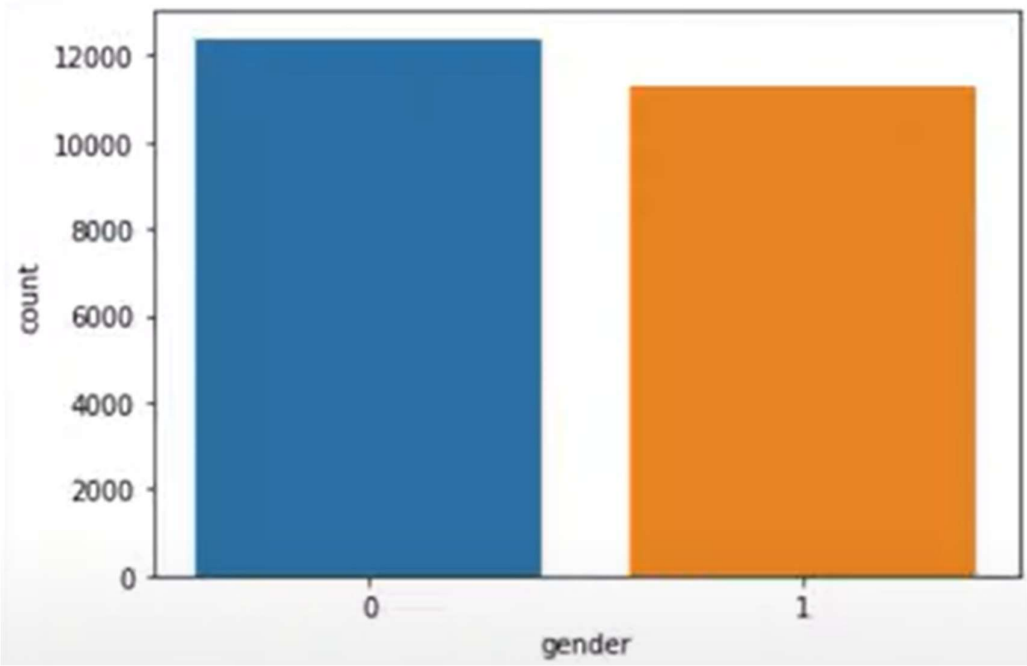
3. Exploratory Data Analysis

3.1 Data Distribution Visualizations

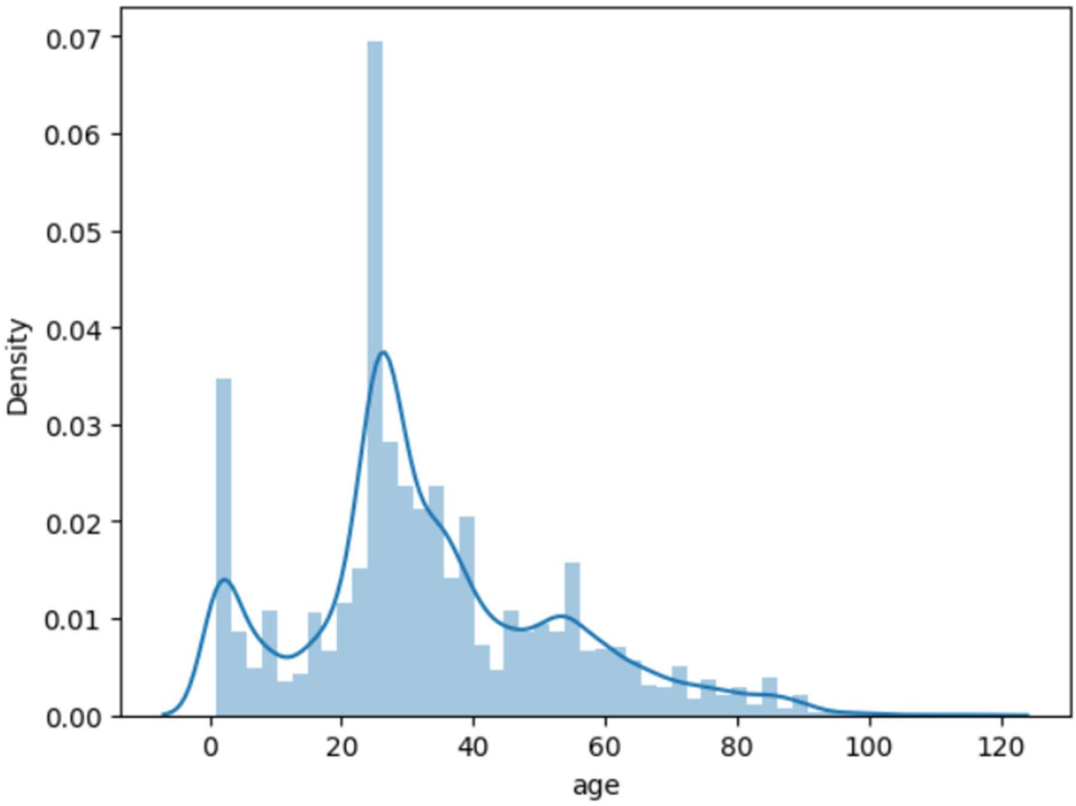
- Checking whether image data is proper and uncorrupted for testing by displaying grid of sample images



- Testing whether sample data is gender biased or not using simple graph



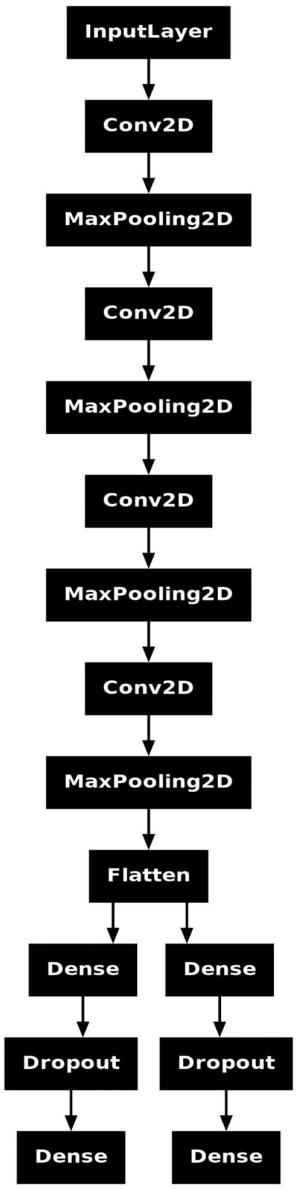
- Created histogram to visualize age bias in sample data



3.2 Feature Extraction

Features from the given dataset of images extracted using neural networks using CNN in which we create Keras Sequential Model. Once feature extraction is done, we can flatten the data into a single vector and feed them to hidden dense layers. The soft-max activation is used at the output layer to make sure these outputs are of categorical data type which is helpful for Image Classification.

Max Pooling down samples the input along its spatial dimensions by taking the maximum value over an input window for each channel of the input. The window is shifted by strides along each dimension.



4. Model Creation

4.1 Conventional Neural Networks (CNN)

It is a type deep learning algorithm that is particularly well-suited for image recognition and processing tasks. It is made up of many layers, including convolutional layers, pooling layers, and fully connected layers. The convolutional layers are the key component of a CNN, where filters are applied to the input image to extract features such as edges, textures, and shapes. The output of the convolutional layers is then passed through pooling layers, which are used to down-sample the feature maps, reducing the spatial dimensions while retaining the most important information. The output of the pooling layers is then passed through one or more fully connected layers, which are used to make a prediction or classify the image.

4.2 Logistic Regression

It is a supervised machine learning algorithm used for classification tasks where the goal is to predict the probability that an instance belongs to a given class or not. Logistic regression is a statistical algorithm which analyse the relationship between two data factors. It involves the usage of dependent & independent variables along with sigmoid function. The dependent variable is categorical & binary (0 or 1). It is used for detection problems such as: fraud detection, emergency detection, spam detection (‘spam’ or ‘no spam’) and also disease diagnosis.

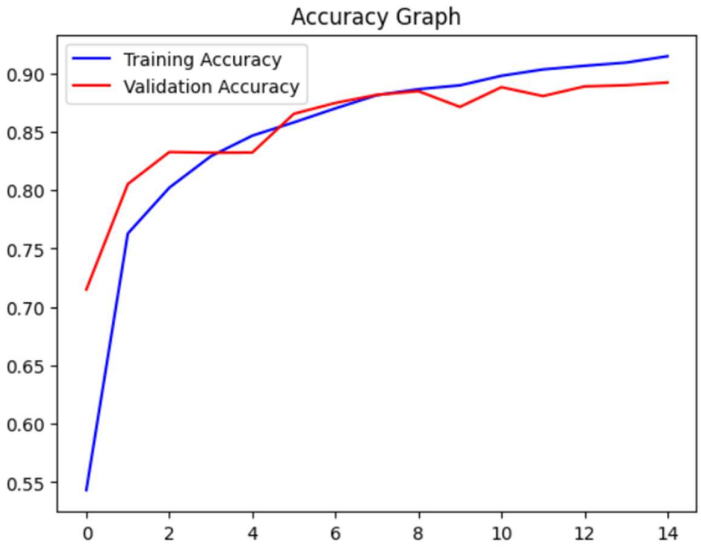
4.3 Mean Absolute Error (MAE)

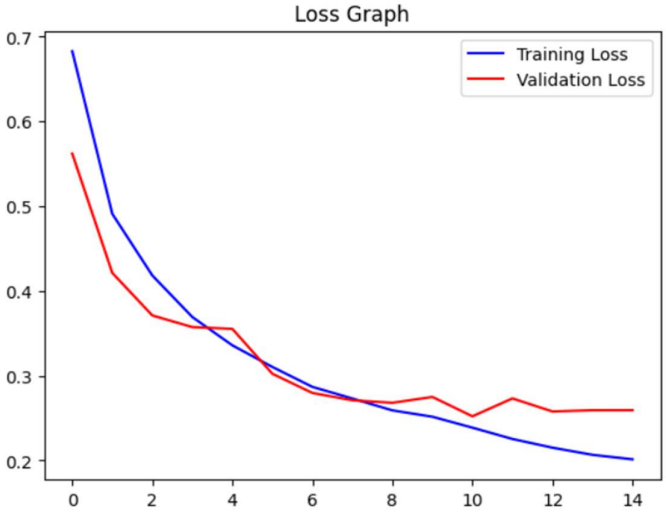
In simpler terms, MAE calculates the average of the absolute differences between each predicted value and its corresponding true value.

It provides a measure of the model's accuracy in terms of how far off its predictions

are from the actual values, regardless of the direction of the error.

4.4 Training Results





As we can see in both of the graphs, given dataset was trained on 30 epochs which lead to underfitting. Possible solution for this is to train at 15 epochs for better validation.

5. Model Evaluation

- **Gender Accuracy:** 89.20 %
- **Age Mean Absolute Error (MAE):** 6.78 years

6. Conclusion

In conclusion, our deep learning project utilizing Convolutional Neural Networks (CNNs) & Logistic Regression successfully achieved gender detection and age prediction tasks with a reasonably high degree of accuracy. By leveraging CNNs & Logistic Regressions, we effectively extracted features from facial images, enabling accurate classification & detection of gender and prediction of age. This project demonstrates the efficacy of CNNs in addressing complex tasks in computer vision, offering promising applications in various domains such as security, marketing, and personalized user experiences.

7. Future Directions

In this study focused on gender detection and age prediction, a crucial consideration was the management of overfitting, addressed by limiting the number of training epochs to 15 from an initial 30 epochs. In pursuit of accuracy enhancement, the necessity for additional epochs becomes apparent. However, it's imperative to tread cautiously, as extending the number of training epochs beyond certain threshold risks exacerbating overfitting. To strike a balance between maximizing accuracy and mitigating overfitting, a systematic approach is warranted. Techniques such as early stopping and model regularization can be employed to prevent the model from memorizing the training data excessively. Overfitting poses significant dangers to the reliability and generalization ability of machine learning models. By excessively fitting to the intricacies of the training data, models risk failing to accurately predict on new, unseen data. The observed increase in validation loss during initial training was a clear indication to us about the model's struggle to generalize beyond the training set, highlighting the importance of vigilant monitoring during model training.

In [1]:

```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the
input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output
when you create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the curren
t session
```

Import Modules

In [2]:

```
import pandas as pd
import numpy as np
import os
os.listdir('/kaggle/input')
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 keras.preprocessing.image import load_img
from keras.models import Sequential, Model
from keras.layers import Dense, Conv2D, Dropout, Flatten, MaxPooling2D, Input
```

In [3]:

```
BASE_DIR = '/kaggle/input/cd-project/UTKFace'
```

In [4]:

```
# 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)
```

```
100% 23708/23708 [00:00<00:00, 272040.14it/s]
```

In [5]:

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

Out[5]:

	image	age	gender
0	/kaggle/input/cd-project/UTKFace/26_0_2_201701...	26	0
1	/kaggle/input/cd-project/UTKFace/22_1_1_201701...	22	1
2	/kaggle/input/cd-project/UTKFace/21_1_3_201701...	21	1
3	/kaggle/input/cd-project/UTKFace/28_0_0_201701...	28	0
4	/kaggle/input/cd-project/UTKFace/17_1_4_201701...	17	1

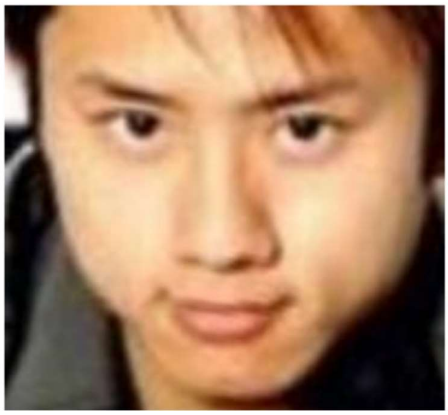
In [6]:

```
# map labels for gender
gender_dict = {0:'Male', 1:'Female'}
```

Exploratory Data Analysis

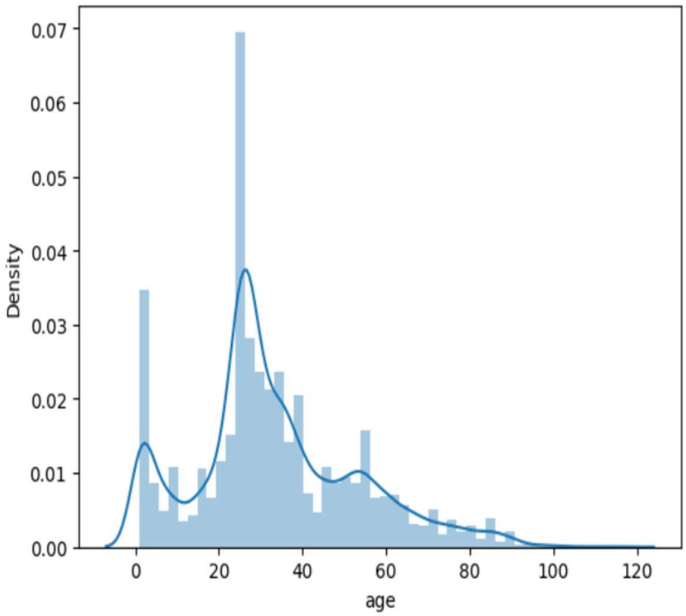
In [7]:

```
from PIL import Image
img = Image.open(df['image'][0])
plt.axis('off')
plt.imshow(img);
```



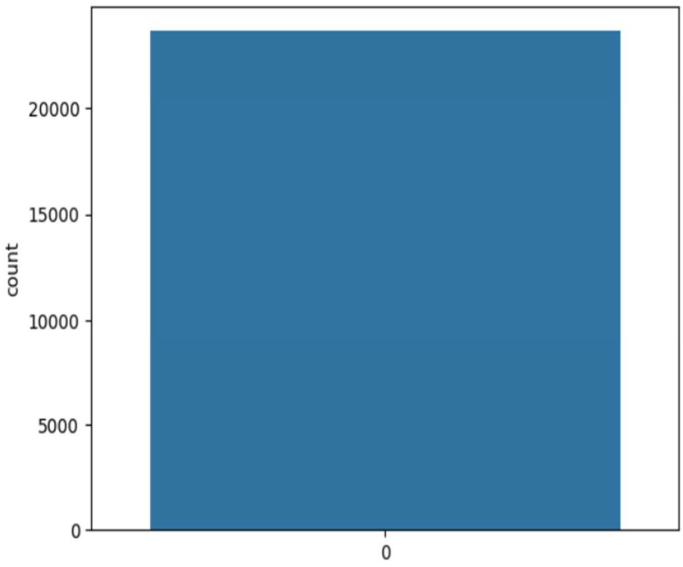
In [8]:
```
sns.distplot(df['age'])
```

Out[8]: <Axes: xlabel='age', ylabel='Density'>



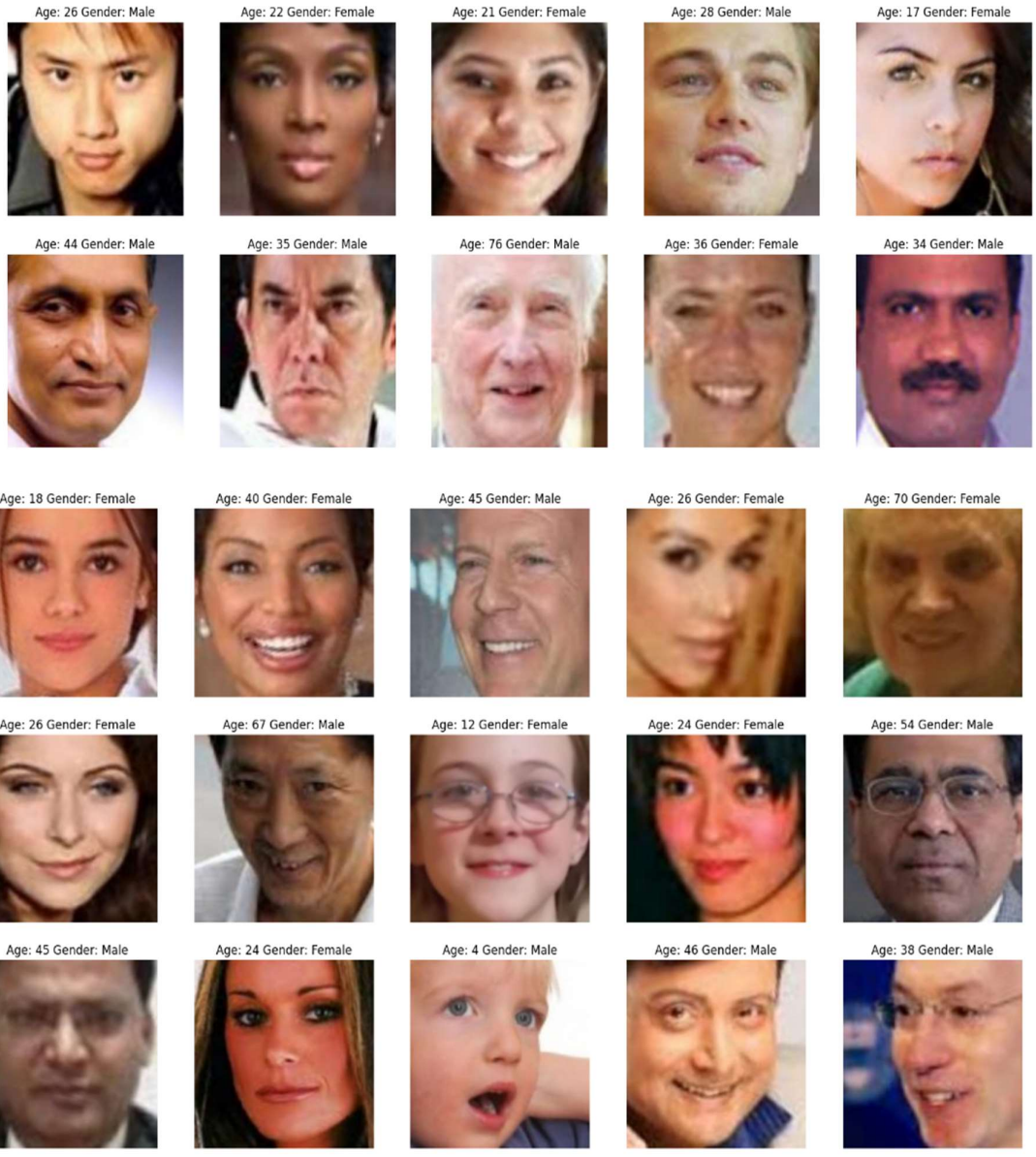
In [9]:
```
sns.countplot(df['gender'])
```

Out[9]: <Axes: ylabel='count'>



In [10]:

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

for index, file, age, gender in files.itertuples():
    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')
```



Feature Extraction

In [11]:
```
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
```

In [12]:
```
X = extract_features(df['image'])
```

```
100% |████████████████████████████████| 23708/23708 [02:43<00:00, 123.47it/s]
```

In [13]:
```
X.shape
```

Out[13]:
```
(23708, 128, 128, 1)
```

In [14]:
```
# normalize the images
X = X/255.0
```

In [15]:
```
y_gender = np.array(df['gender'])
y_age = np.array(df['age'])
```

In [16]:
```
input_shape = (128, 128, 1)
```

Model Creation

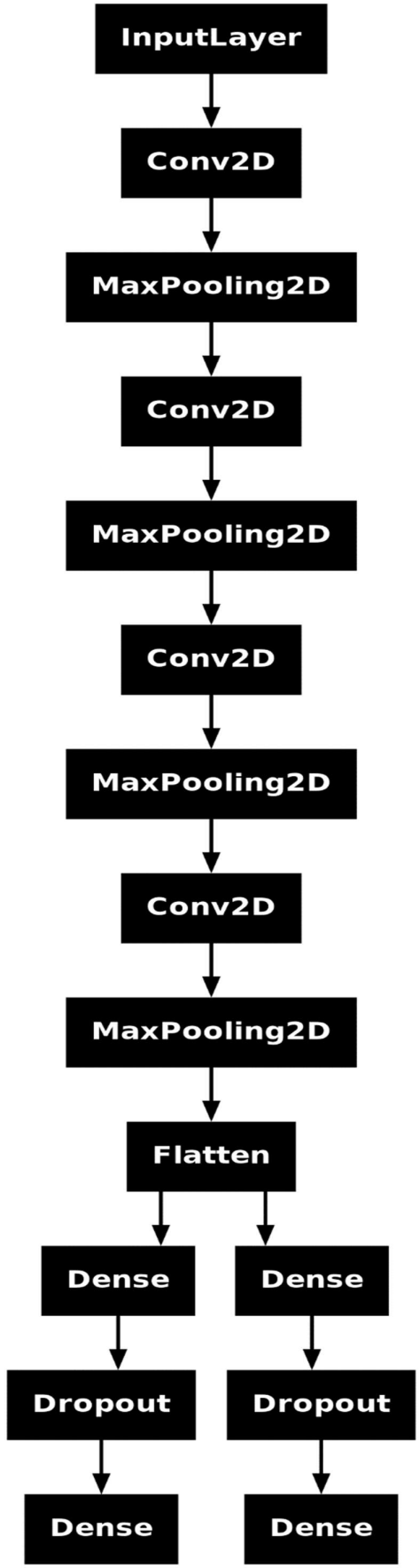
In [17]:

```
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'])
```

In [18]:

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


InputLayer
Conv2D
MaxPooling2D
Conv2D
MaxPooling2D
Conv2D
MaxPooling2D
Conv2D
MaxPooling2D
Flatten
Dense
Dense
Dropout
Dropout
Dense
Dense

In [19]:

```
# train model
history = model.fit(x=X, y=[y_gender, y_age], batch_size=32, epochs=15, validation_split=0.2)
```

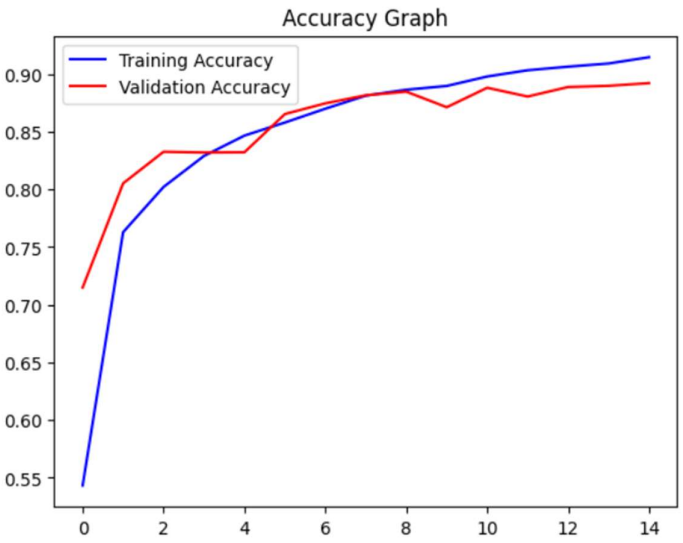
```
Epoch 1/15
593/593 [==============================] - 337s 565ms/step - loss: 16.1278 - gender_out_los
s: 0.6827 - age_out_loss: 15.4451 - gender_out_accuracy: 0.5433 - age_out_accuracy: 0.0474
- val_loss: 13.2087 - val_gender_out_loss: 0.5620 - val_age_out_loss: 12.6467 - val_gender_
out_accuracy: 0.7149 - val_age_out_accuracy: 0.0428
Epoch 2/15
593/593 [==============================] - 352s 593ms/step - loss: 11.7583 - gender_out_los
s: 0.4910 - age_out_loss: 11.2673 - gender_out_accuracy: 0.7628 - age_out_accuracy: 0.0327
- val_loss: 11.0566 - val_gender_out_loss: 0.4213 - val_age_out_loss: 10.6352 - val_gender_
out_accuracy: 0.8051 - val_age_out_accuracy: 0.0164
Epoch 3/15
593/593 [==============================] - 321s 541ms/step - loss: 10.1151 - gender_out_los
s: 0.4183 - age_out_loss: 9.6968 - gender_out_accuracy: 0.8021 - age_out_accuracy: 0.0178 -
val_loss: 8.7165 - val_gender_out_loss: 0.3711 - val_age_out_loss: 8.3453 - val_gender_out_
accuracy: 0.8326 - val_age_out_accuracy: 0.0084
Epoch 4/15
593/593 [==============================] - 322s 542ms/step - loss: 8.8342 - gender_out_los
s: 0.3693 - age_out_loss: 8.4649 - gender_out_accuracy: 0.8290 - age_out_accuracy: 0.0115 -
val_loss: 7.9507 - val_gender_out_loss: 0.3574 - val_age_out_loss: 7.5933 - val_gender_out_
accuracy: 0.8319 - val_age_out_accuracy: 0.0101

Epoch 5/15
593/593 [==============================] - 322s 543ms/step - loss: 8.1193 - gender_out_los
s: 0.3359 - age_out_loss: 7.7834 - gender_out_accuracy: 0.8466 - age_out_accuracy: 0.0099 -
val_loss: 7.9655 - val_gender_out_loss: 0.3553 - val_age_out_loss: 7.6102 - val_gender_out_
accuracy: 0.8321 - val_age_out_accuracy: 0.0044
Epoch 6/15
593/593 [==============================] - 320s 540ms/step - loss: 7.6616 - gender_out_los
s: 0.3104 - age_out_loss: 7.3512 - gender_out_accuracy: 0.8579 - age_out_accuracy: 0.0092 -
val_loss: 7.9376 - val_gender_out_loss: 0.3024 - val_age_out_loss: 7.6352 - val_gender_out_
accuracy: 0.8652 - val_age_out_accuracy: 0.0044
Epoch 7/15
593/593 [==============================] - 319s 537ms/step - loss: 7.2902 - gender_out_los
s: 0.2869 - age_out_loss: 7.0033 - gender_out_accuracy: 0.8699 - age_out_accuracy: 0.0079 -
val_loss: 7.3570 - val_gender_out_loss: 0.2797 - val_age_out_loss: 7.0773 - val_gender_out_
accuracy: 0.8745 - val_age_out_accuracy: 0.0063
Epoch 8/15
593/593 [==============================] - 318s 537ms/step - loss: 7.0370 - gender_out_los
s: 0.2732 - age_out_loss: 6.7638 - gender_out_accuracy: 0.8812 - age_out_accuracy: 0.0085 -
val_loss: 7.1445 - val_gender_out_loss: 0.2711 - val_age_out_loss: 6.8734 - val_gender_out_
accuracy: 0.8815 - val_age_out_accuracy: 0.0061
```

```
Epoch 9/15
593/593 [==============================] - 318s 536ms/step - loss: 6.6850 - gender_out_los
s: 0.2592 - age_out_loss: 6.4258 - gender_out_accuracy: 0.8863 - age_out_accuracy: 0.0080 -
val_loss: 7.4938 - val_gender_out_loss: 0.2681 - val_age_out_loss: 7.2257 - val_gender_out_
accuracy: 0.8846 - val_age_out_accuracy: 0.0036
Epoch 10/15
593/593 [==============================] - 339s 571ms/step - loss: 6.5820 - gender_out_los
s: 0.2517 - age_out_loss: 6.3303 - gender_out_accuracy: 0.8896 - age_out_accuracy: 0.0076 -
val_loss: 7.0044 - val_gender_out_loss: 0.2750 - val_age_out_loss: 6.7294 - val_gender_out_
accuracy: 0.8712 - val_age_out_accuracy: 0.0049
Epoch 11/15
593/593 [==============================] - 318s 535ms/step - loss: 6.2329 - gender_out_los
s: 0.2388 - age_out_loss: 5.9941 - gender_out_accuracy: 0.8978 - age_out_accuracy: 0.0065 -
val_loss: 7.2047 - val_gender_out_loss: 0.2522 - val_age_out_loss: 6.9525 - val_gender_out_
accuracy: 0.8880 - val_age_out_accuracy: 0.0046
Epoch 12/15
593/593 [==============================] - 317s 534ms/step - loss: 6.1126 - gender_out_los
s: 0.2254 - age_out_loss: 5.8872 - gender_out_accuracy: 0.9032 - age_out_accuracy: 0.0073 -
val_loss: 6.8209 - val_gender_out_loss: 0.2733 - val_age_out_loss: 6.5476 - val_gender_out_
accuracy: 0.8804 - val_age_out_accuracy: 0.0049

Epoch 13/15
593/593 [==============================] - 317s 535ms/step - loss: 5.8431 - gender_out_los
s: 0.2152 - age_out_loss: 5.6279 - gender_out_accuracy: 0.9063 - age_out_accuracy: 0.0057 -
val_loss: 6.6959 - val_gender_out_loss: 0.2579 - val_age_out_loss: 6.4380 - val_gender_out_
accuracy: 0.8887 - val_age_out_accuracy: 0.0053
Epoch 14/15
593/593 [==============================] - 320s 540ms/step - loss: 5.6534 - gender_out_los
s: 0.2068 - age_out_loss: 5.4467 - gender_out_accuracy: 0.9090 - age_out_accuracy: 0.0060 -
val_loss: 6.7902 - val_gender_out_loss: 0.2594 - val_age_out_loss: 6.5309 - val_gender_out_
accuracy: 0.8897 - val_age_out_accuracy: 0.0059
Epoch 15/15
593/593 [==============================] - 319s 537ms/step - loss: 5.4796 - gender_out_los
s: 0.2015 - age_out_loss: 5.2781 - gender_out_accuracy: 0.9145 - age_out_accuracy: 0.0063 -
val_loss: 7.0394 - val_gender_out_loss: 0.2594 - val_age_out_loss: 6.7800 - val_gender_out_
accuracy: 0.8920 - val_age_out_accuracy: 0.0046
```

Plot the results

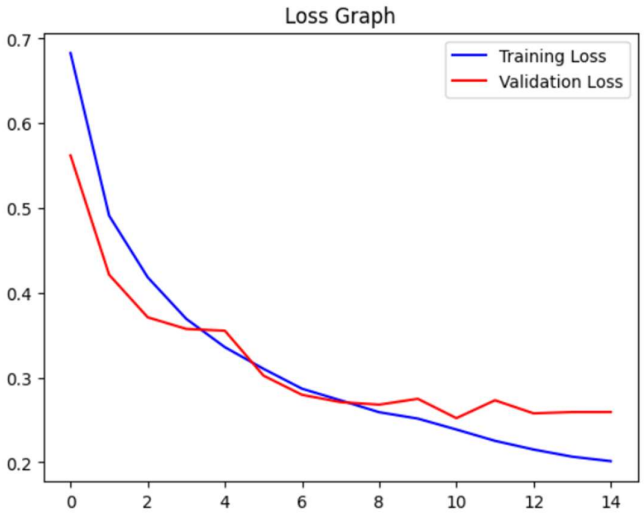
In [20]:

```
# 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()
```



In [21]:

```
# 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()
```



Prediction with Test Data

In [22]:

```
image_index = 100
print("Original Gender:", gender_dict[y_gender[image_index]], "Original Age:", y_age[image_ind
ex])
# 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: Female Original Age: 3
1/1 [==============================] - 0s 184ms/step
Predicted Gender: Female Predicted Age: 0
```

In [23]:

```
image_index = 3000
print("Original Gender:", gender_dict[y_gender[image_index]], "Original Age:", y_age[image_ind
ex])
# 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
1/1 [==============================] - 0s 28ms/step
Predicted Gender: Male Predicted Age: 27
```

In [24]:

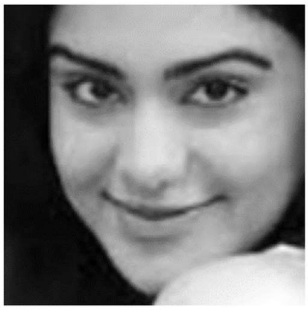
```
image_index = 10000
print("Original Gender:", gender_dict[y_gender[image_index]], "Original Age:", y_age[image_ind
ex])
# predict from model
pred = model.predict(X[image_index].reshape(1, 128, 128, 1))
pred_gender = gender_dict[round(pred[0][0][0])]
pred_age = round(pred[1][0][0])
print("Predicted 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: 42
1/1 [==============================] - 0s 30ms/step
Predicted Gender: Male Predicted Age: 37
```

In [25]:

```
image_index = 420
print("Original Gender:", gender_dict[y_gender[image_index]], "Original Age:", y_age[image_ind
ex])
# 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: Female Original Age: 22
1/1 [==============================] - 0s 28ms/step
Predicted Gender: Female Predicted Age: 25
```



In [26]:

```
image_index = 23000
print("Original Gender:", gender_dict[y_gender[image_index]], "Original Age:", y_age[image_ind
ex])
# 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: Female Original Age: 26
1/1 [==============================] - 0s 29ms/step
Predicted Gender: Female Predicted Age: 26
```

In [27]:

```
image_index = 5000
print("Original Gender:", gender_dict[y_gender[image_index]], "Original Age:", y_age[image_ind
ex])
# 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: Female Original Age: 26
1/1 [==============================] - 0s 28ms/step
Predicted Gender: Female Predicted Age: 24
```

In [28]:

```
image_index = 87
print("Original Gender:", gender_dict[y_gender[image_index]], "Original Age:", y_age[image_ind
ex])
# 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: Female Original Age: 35
1/1 [==============================] - 0s 34ms/step
Predicted Gender: Female Predicted Age: 33
```

In [29]:
```
image_index = 455
print("Original Gender:", gender_dict[y_gender[image_index]], "Original Age:", y_age[image_ind
ex])
# 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: 27
1/1 [==============================] - 0s 28ms/step
Predicted Gender: Male Predicted Age: 30
```

In [30]:
```
image_index = 666
print("Original Gender:", gender_dict[y_gender[image_index]], "Original Age:", y_age[image_ind
ex])
# 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: Female Original Age: 26
1/1 [==============================] - 0s 30ms/step
Predicted Gender: Female Predicted Age: 27
```

In [31]:

```
image_index = 151
print("Original Gender:", gender_dict[y_gender[image_index]], "Original Age:", y_age[image_ind
ex])
# 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: 32
1/1 [==============================] - 0s 28ms/step
Predicted Gender: Male Predicted Age: 27
```