Gender&Age_Prediction

April 5, 2024

Age and Gender Prediction. In this project, we will be performing both classification and regression to predict both gender and age respectively.

[3]:
```
# Ignore warnings for cleaner output
import warnings
warnings.filterwarnings('ignore')

# Data manipulation libraries
import pandas as pd
import numpy as np

# Operating system module
import os

# Visualization libraries
import matplotlib.pyplot as plt
import seaborn as sns

# Progress bar module
from tqdm.notebook import tqdm

# Tensorflow and Keras modules for building and training the model
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
```

Load the dataset

[ ]:
```
# Mount Google Drive to access files
from google.colab import drive
drive.mount('/content/drive')

# Unzip the dataset file located in Google Drive
!unzip '/content/drive/MyDrive/Projects/AGE & GENDER/UTKFace.zip'
```

[5]:
```
# Directory path where the dataset is located
BASE_DIR = "/content/UTKFace"
```

[6]:
```python
# List to store paths of images
image_paths = []

# List to store age labels
age_labels = []

# List to store gender labels
gender_labels = []
```

[7]:
```python
# Iterate over each file in the dataset directory
for filename in tqdm(os.listdir(BASE_DIR)):
    # Create the full path of the image file
    image_path = os.path.join(BASE_DIR, filename)

    # Split the filename to extract age and gender information
    temp = filename.split('_')

    # Extract age from the filename and convert it to integer
    age = int(temp[0])

    # Extract gender from the filename and convert it to integer
    gender = int(temp[1])

    # Append the image path to the list of image paths
    image_paths.append(image_path)

    # Append the age label to the list of age labels
    age_labels.append(age)

    # Append the gender label to the list of gender labels
    gender_labels.append(gender)
```

```
  0%|          | 0/23708 [00:00<?, ?it/s]
```

[8]:
```python
print(f'Number of age_labels: {len(age_labels)}, Number of gender_labels:␣
 ↪{len(gender_labels)}, Number of image_paths: {len(image_paths)}')
```

```
Number of age_labels: 23708, Number of gender_labels: 23708, Number of
image_paths: 23708
```

[9]:
```python
# Create a Pandas DataFrame to store image paths, age labels, and gender labels
df = pd.DataFrame()

# Assign image paths, age labels, and gender labels to DataFrame columns
df['image'] = image_paths
df['age'] = age_labels
df['gender'] = gender_labels
```

[10]:

```
# Display the first few rows of the DataFrame to inspect its structure and␣
↪contents
df.head()
```

[10]:

	image	age	gender
0	/content/UTKFace/24_1_0_20170116222814643.jpg…	24	1
1	/content/UTKFace/25_0_1_20170116205335757.jpg…	25	0
2	/content/UTKFace/4_0_4_20161221195021183.jpg.c…	4	0
3	/content/UTKFace/28_0_0_20170117134849833.jpg…	28	0
4	/content/UTKFace/35_0_1_20170113152731945.jpg…	35	0

[11]:

```
# Dictionary mapping gender labels to gender names
gender_dict = {0: 'Male', 1: 'Female'}
```

Exploratory Data Analysis (EDA)

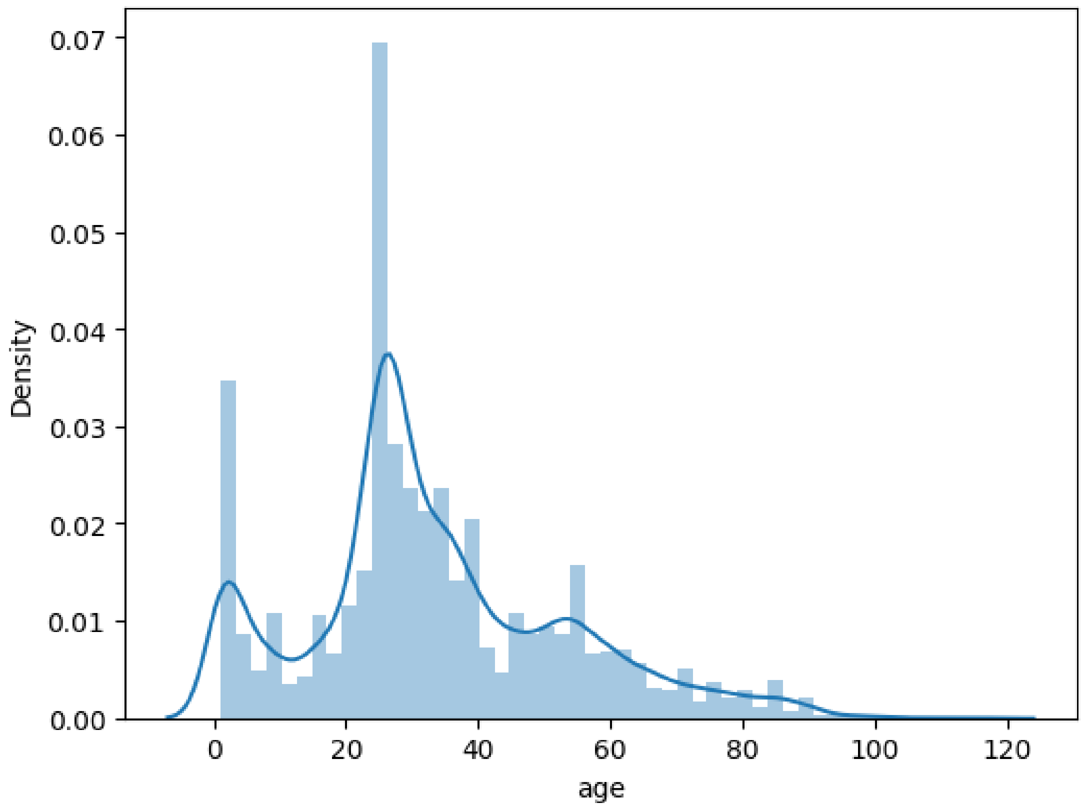
[12]:

```
# Import necessary module from PIL library
from PIL import Image

# Get the age and gender labels for the first image in the DataFrame
age = df['age'][0]
gender = df['gender'][0]

# Open the first image using its path stored in the DataFrame
img = Image.open(df['image'][0])

# Set title with age and gender information
plt.title(f'Age = {age} & Gender = {gender_dict[gender]}')

# Turn off axis
plt.axis('off')

# Display the image
plt.imshow(img);
```

Age = 24 & Gender = Female

[13]:
```python
# Plot a distribution plot for age labels using Seaborn
sns.distplot(df['age'])
```

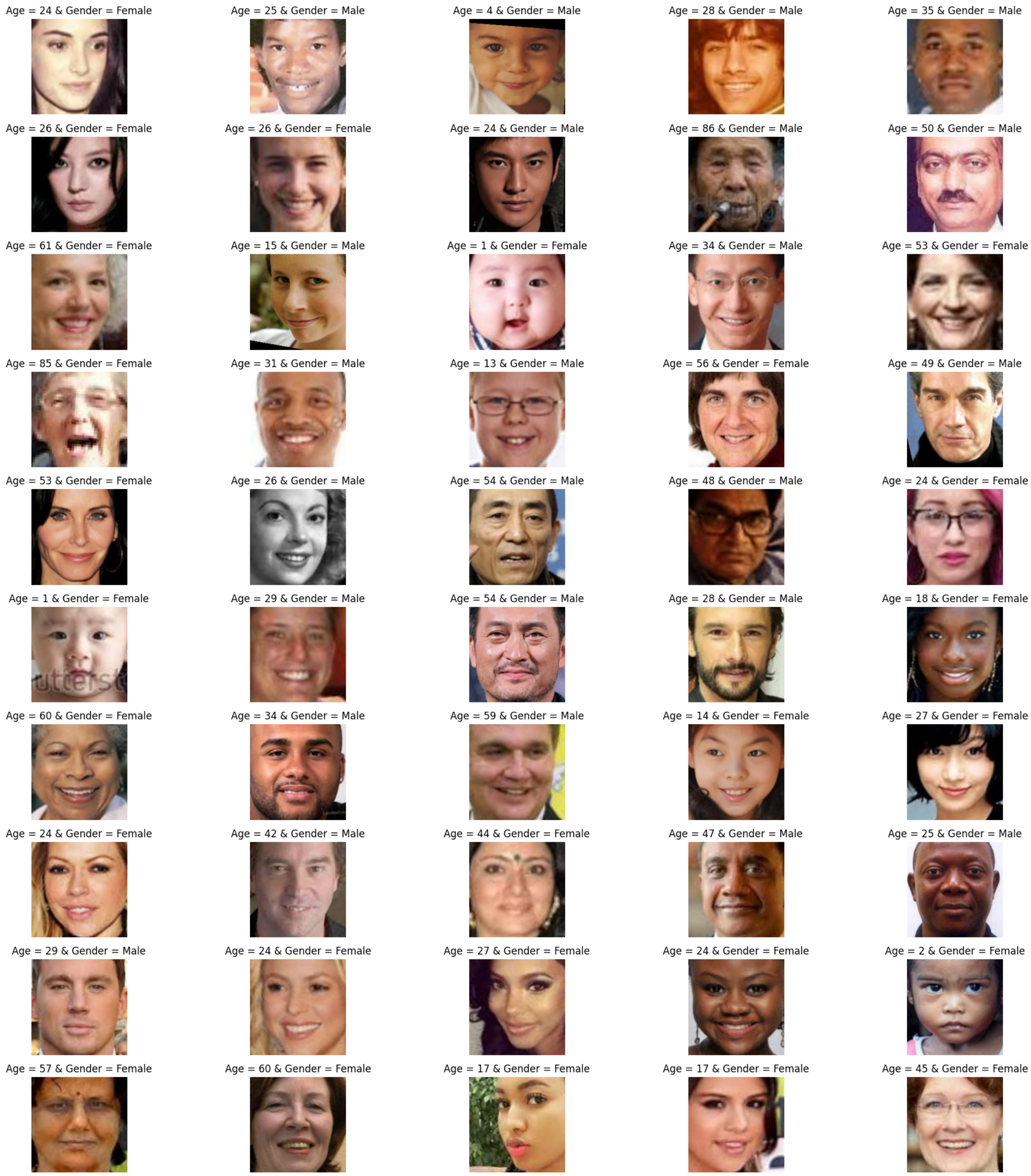
[13]: `<Axes: xlabel='age', ylabel='Density'>`



The distribution roughly follows a normal distribution that is slightly skewed to the right with a median of around 27 years. The majority are in between ages 25 to 30 years old.The range is from 0 to 120 years. There are some outliers at the higher end of the distribution.

[14]:
```
# Set figure size
plt.figure(figsize=(20, 20))

# Select the first 50 rows from the DataFrame
files = df.iloc[0:50]

# Iterate over the selected rows
for index, file, age, gender in files.itertuples():
    # Create subplots in a grid of 10x5
    plt.subplot(10, 5, index+1)

    # Load image
    img = load_img(file)

    # Convert image to numpy array
    img = np.array(img)
```

```python
    # Display the image
    plt.imshow(img)

    # Set title with age and gender information
    plt.title(f'Age = {age} & Gender = {gender_dict[gender]}')

    # Turn off axis
    plt.axis('off')

# Adjust layout for better spacing
plt.tight_layout()

# Show the plot
plt.show()
```



FEATURE EXTRACTION

[15]:
```python
# Define a function to extract features from images
def extract_features(images):
    # Initialize an empty list to store features
    features = []

    # Iterate over each image
    for image in tqdm(images):
```

```
        # Load image in grayscale mode and resize it to 128x128
        img = load_img(image, color_mode='grayscale')
        img = img.resize((128, 128), Image.LANCZOS)

        # Convert image to numpy array
        img = np.array(img)

        # Append the image to the features list
        features.append(img)

    # Convert the list of features to a numpy array
    features = np.array(features)

    # Reshape the features array to the required format for the model
    features = features.reshape(len(features), 128, 128, 1)

    return features
```

[16]:
```
# Extract features from the images in the DataFrame
X = extract_features(df['image'])
```

```
0%|          | 0/23708 [00:00<?, ?it/s]
```

[17]:
```
# Get the shape of the array representing extracted features
X.shape
```

[17]:
```
(23708, 128, 128, 1)
```

[18]:
```
# Normalize the pixel values of the images
X = X / 255.0
```

[19]:
```
# Convert gender labels to numpy array
y_gender = np.array(df['gender'])

# Convert age labels to numpy array
y_age = np.array(df['age'])
```

[20]:
```
# Define the input shape for the model
input_shape = (128, 128, 1)
```

[21]:
```
from sklearn.model_selection import train_test_split

# Split the dataset into training and testing sets
X_train, X_test, y_gender_train, y_gender_test, y_age_train, y_age_test = train_test_split(X, y_gender, y_age, test_size=0.2, random_state=42)
```

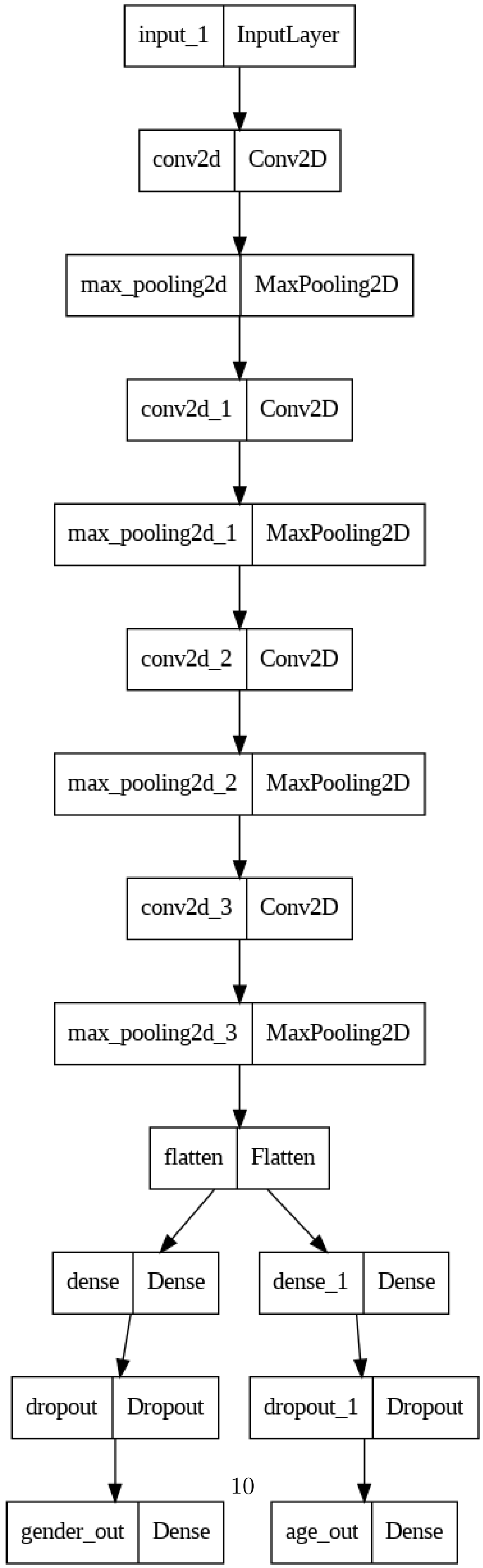
[22]:
```
# Define the input layer
inputs = Input((input_shape))
```

```
# 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 layer
flatten = Flatten()(maxp_4)

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

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

# Output layers for gender and age predictions
output_1 = Dense(1, activation='sigmoid', name='gender_out')(dropout_1)
output_2 = Dense(1, activation='relu', name='age_out')(dropout_2)

# Define the model with input and output layers
model = Model(inputs=[inputs], outputs=[output_1, output_2])
```

[23]:
```
# Compile the model with appropriate loss functions, optimizer, and metrics
model.compile(loss=['binary_crossentropy', 'mae'],
              optimizer='adam',
              metrics=['accuracy', 'mae'])
```

[24]:
```
# Import the necessary module to plot the model
from tensorflow.keras.utils import plot_model

# Plot the model architecture
plot_model(model)
```

[24]:


input_1 InputLayer
conv2d Conv2D
max_pooling2d MaxPooling2D
conv2d_1 Conv2D
max_pooling2d_1 MaxPooling2D
conv2d_2 Conv2D
max_pooling2d_2 MaxPooling2D
conv2d_3 Conv2D
max_pooling2d_3 MaxPooling2D
flatten Flatten
dense Dense
dense_1 Dense
dropout Dropout
dropout_1 Dropout
gender_out Dense
age_out Dense

[25]:
```
# Train the model with the training data, using both gender and age labels as
↪targets
history = model.fit(x=X_train, y=[y_gender_train, y_age_train], batch_size=32,
↪epochs=30, validation_split=0.2)
```

```
Epoch 1/30
475/475 [==============================] - 22s 27ms/step - loss: 16.5882 -
gender_out_loss: 0.7081 - age_out_loss: 15.8801 - gender_out_accuracy: 0.5108 -
gender_out_mae: 0.4985 - age_out_accuracy: 0.0475 - age_out_mae: 15.8801 -
val_loss: 16.0832 - val_gender_out_loss: 0.6925 - val_age_out_loss: 15.3907 -
val_gender_out_accuracy: 0.5374 - val_gender_out_mae: 0.4996 -
val_age_out_accuracy: 0.0477 - val_age_out_mae: 15.3907
Epoch 2/30
475/475 [==============================] - 10s 22ms/step - loss: 14.6125 -
gender_out_loss: 0.6225 - age_out_loss: 13.9900 - gender_out_accuracy: 0.6298 -
gender_out_mae: 0.4358 - age_out_accuracy: 0.0473 - age_out_mae: 13.9900 -
val_loss: 12.5151 - val_gender_out_loss: 0.4788 - val_age_out_loss: 12.0362 -
val_gender_out_accuracy: 0.7746 - val_gender_out_mae: 0.3353 -
val_age_out_accuracy: 0.0477 - val_age_out_mae: 12.0362
Epoch 3/30
475/475 [==============================] - 11s 24ms/step - loss: 11.4935 -
gender_out_loss: 0.4650 - age_out_loss: 11.0284 - gender_out_accuracy: 0.7810 -
gender_out_mae: 0.3054 - age_out_accuracy: 0.0408 - age_out_mae: 11.0284 -
val_loss: 9.9371 - val_gender_out_loss: 0.3998 - val_age_out_loss: 9.5373 -
val_gender_out_accuracy: 0.8229 - val_gender_out_mae: 0.2766 -
val_age_out_accuracy: 0.0314 - val_age_out_mae: 9.5373
Epoch 4/30
475/475 [==============================] - 11s 23ms/step - loss: 9.8988 -
gender_out_loss: 0.3972 - age_out_loss: 9.5016 - gender_out_accuracy: 0.8177 -
gender_out_mae: 0.2587 - age_out_accuracy: 0.0273 - age_out_mae: 9.5016 -
val_loss: 9.1260 - val_gender_out_loss: 0.3657 - val_age_out_loss: 8.7602 -
val_gender_out_accuracy: 0.8334 - val_gender_out_mae: 0.2336 -
val_age_out_accuracy: 0.0148 - val_age_out_mae: 8.7602
Epoch 5/30
475/475 [==============================] - 11s 24ms/step - loss: 9.0532 -
gender_out_loss: 0.3633 - age_out_loss: 8.6899 - gender_out_accuracy: 0.8350 -
gender_out_mae: 0.2359 - age_out_accuracy: 0.0220 - age_out_mae: 8.6899 -
val_loss: 10.8331 - val_gender_out_loss: 0.3430 - val_age_out_loss: 10.4902 -
val_gender_out_accuracy: 0.8416 - val_gender_out_mae: 0.2240 -
val_age_out_accuracy: 0.0074 - val_age_out_mae: 10.4902
Epoch 6/30
475/475 [==============================] - 11s 23ms/step - loss: 8.4760 -
gender_out_loss: 0.3348 - age_out_loss: 8.1412 - gender_out_accuracy: 0.8469 -
gender_out_mae: 0.2170 - age_out_accuracy: 0.0176 - age_out_mae: 8.1412 -
val_loss: 7.9622 - val_gender_out_loss: 0.3226 - val_age_out_loss: 7.6396 -
```

```
val_gender_out_accuracy: 0.8527 - val_gender_out_mae: 0.1999 -
val_age_out_accuracy: 0.0098 - val_age_out_mae: 7.6396
Epoch 7/30
475/475 [==============================] - 11s 23ms/step - loss: 7.9371 -
gender_out_loss: 0.3101 - age_out_loss: 7.6270 - gender_out_accuracy: 0.8622 -
gender_out_mae: 0.1987 - age_out_accuracy: 0.0190 - age_out_mae: 7.6270 -
val_loss: 7.6369 - val_gender_out_loss: 0.3217 - val_age_out_loss: 7.3152 -
val_gender_out_accuracy: 0.8503 - val_gender_out_mae: 0.1910 -
val_age_out_accuracy: 0.0074 - val_age_out_mae: 7.3152
Epoch 8/30
475/475 [==============================] - 11s 23ms/step - loss: 7.5689 -
gender_out_loss: 0.2930 - age_out_loss: 7.2758 - gender_out_accuracy: 0.8704 -
gender_out_mae: 0.1867 - age_out_accuracy: 0.0173 - age_out_mae: 7.2758 -
val_loss: 7.2670 - val_gender_out_loss: 0.2904 - val_age_out_loss: 6.9765 -
val_gender_out_accuracy: 0.8730 - val_gender_out_mae: 0.1776 -
val_age_out_accuracy: 0.0074 - val_age_out_mae: 6.9765
Epoch 9/30
475/475 [==============================] - 10s 21ms/step - loss: 7.1821 -
gender_out_loss: 0.2732 - age_out_loss: 6.9089 - gender_out_accuracy: 0.8797 -
gender_out_mae: 0.1734 - age_out_accuracy: 0.0180 - age_out_mae: 6.9089 -
val_loss: 7.4780 - val_gender_out_loss: 0.2809 - val_age_out_loss: 7.1970 -
val_gender_out_accuracy: 0.8793 - val_gender_out_mae: 0.1770 -
val_age_out_accuracy: 0.0092 - val_age_out_mae: 7.1970
Epoch 10/30
475/475 [==============================] - 12s 24ms/step - loss: 6.9139 -
gender_out_loss: 0.2596 - age_out_loss: 6.6543 - gender_out_accuracy: 0.8851 -
gender_out_mae: 0.1629 - age_out_accuracy: 0.0167 - age_out_mae: 6.6543 -
val_loss: 7.5915 - val_gender_out_loss: 0.2817 - val_age_out_loss: 7.3098 -
val_gender_out_accuracy: 0.8819 - val_gender_out_mae: 0.1785 -
val_age_out_accuracy: 0.0132 - val_age_out_mae: 7.3098
Epoch 11/30
475/475 [==============================] - 11s 22ms/step - loss: 6.8010 -
gender_out_loss: 0.2516 - age_out_loss: 6.5494 - gender_out_accuracy: 0.8885 -
gender_out_mae: 0.1598 - age_out_accuracy: 0.0177 - age_out_mae: 6.5494 -
val_loss: 7.2568 - val_gender_out_loss: 0.2729 - val_age_out_loss: 6.9839 -
val_gender_out_accuracy: 0.8795 - val_gender_out_mae: 0.1638 -
val_age_out_accuracy: 0.0090 - val_age_out_mae: 6.9839
Epoch 12/30
475/475 [==============================] - 12s 25ms/step - loss: 6.4481 -
gender_out_loss: 0.2365 - age_out_loss: 6.2116 - gender_out_accuracy: 0.8961 -
gender_out_mae: 0.1499 - age_out_accuracy: 0.0182 - age_out_mae: 6.2116 -
val_loss: 7.2439 - val_gender_out_loss: 0.2732 - val_age_out_loss: 6.9707 -
val_gender_out_accuracy: 0.8793 - val_gender_out_mae: 0.1722 -
val_age_out_accuracy: 0.0055 - val_age_out_mae: 6.9707
Epoch 13/30
475/475 [==============================] - 11s 24ms/step - loss: 6.2267 -
gender_out_loss: 0.2243 - age_out_loss: 6.0024 - gender_out_accuracy: 0.9032 -
gender_out_mae: 0.1409 - age_out_accuracy: 0.0168 - age_out_mae: 6.0024 -
```

```
val_loss: 7.9993 - val_gender_out_loss: 0.2688 - val_age_out_loss: 7.7305 -
val_gender_out_accuracy: 0.8888 - val_gender_out_mae: 0.1590 -
val_age_out_accuracy: 0.0069 - val_age_out_mae: 7.7305
Epoch 14/30
475/475 [==============================] - 11s 23ms/step - loss: 5.9611 -
gender_out_loss: 0.2161 - age_out_loss: 5.7450 - gender_out_accuracy: 0.9053 -
gender_out_mae: 0.1377 - age_out_accuracy: 0.0173 - age_out_mae: 5.7450 -
val_loss: 7.0924 - val_gender_out_loss: 0.2684 - val_age_out_loss: 6.8240 -
val_gender_out_accuracy: 0.8898 - val_gender_out_mae: 0.1437 -
val_age_out_accuracy: 0.0090 - val_age_out_mae: 6.8240
Epoch 15/30
475/475 [==============================] - 11s 24ms/step - loss: 5.7146 -
gender_out_loss: 0.2016 - age_out_loss: 5.5130 - gender_out_accuracy: 0.9145 -
gender_out_mae: 0.1270 - age_out_accuracy: 0.0225 - age_out_mae: 5.5130 -
val_loss: 7.1868 - val_gender_out_loss: 0.2820 - val_age_out_loss: 6.9048 -
val_gender_out_accuracy: 0.8788 - val_gender_out_mae: 0.1472 -
val_age_out_accuracy: 0.0245 - val_age_out_mae: 6.9048
Epoch 16/30
475/475 [==============================] - 10s 22ms/step - loss: 5.4836 -
gender_out_loss: 0.1938 - age_out_loss: 5.2898 - gender_out_accuracy: 0.9170 -
gender_out_mae: 0.1229 - age_out_accuracy: 0.0432 - age_out_mae: 5.2898 -
val_loss: 6.7811 - val_gender_out_loss: 0.2661 - val_age_out_loss: 6.5151 -
val_gender_out_accuracy: 0.8962 - val_gender_out_mae: 0.1505 -
val_age_out_accuracy: 0.0414 - val_age_out_mae: 6.5151
Epoch 17/30
475/475 [==============================] - 12s 25ms/step - loss: 5.2717 -
gender_out_loss: 0.1844 - age_out_loss: 5.0873 - gender_out_accuracy: 0.9218 -
gender_out_mae: 0.1168 - age_out_accuracy: 0.0432 - age_out_mae: 5.0873 -
val_loss: 7.1845 - val_gender_out_loss: 0.2894 - val_age_out_loss: 6.8951 -
val_gender_out_accuracy: 0.8859 - val_gender_out_mae: 0.1358 -
val_age_out_accuracy: 0.0414 - val_age_out_mae: 6.8951
Epoch 18/30
475/475 [==============================] - 11s 23ms/step - loss: 5.1785 -
gender_out_loss: 0.1717 - age_out_loss: 5.0068 - gender_out_accuracy: 0.9268 -
gender_out_mae: 0.1103 - age_out_accuracy: 0.0447 - age_out_mae: 5.0068 -
val_loss: 6.9792 - val_gender_out_loss: 0.2791 - val_age_out_loss: 6.7000 -
val_gender_out_accuracy: 0.8890 - val_gender_out_mae: 0.1419 -
val_age_out_accuracy: 0.0456 - val_age_out_mae: 6.7000
Epoch 19/30
475/475 [==============================] - 11s 24ms/step - loss: 4.9589 -
gender_out_loss: 0.1633 - age_out_loss: 4.7956 - gender_out_accuracy: 0.9316 -
gender_out_mae: 0.1043 - age_out_accuracy: 0.0461 - age_out_mae: 4.7956 -
val_loss: 7.1478 - val_gender_out_loss: 0.3038 - val_age_out_loss: 6.8439 -
val_gender_out_accuracy: 0.8853 - val_gender_out_mae: 0.1355 -
val_age_out_accuracy: 0.0461 - val_age_out_mae: 6.8439
Epoch 20/30
475/475 [==============================] - 11s 23ms/step - loss: 4.8330 -
gender_out_loss: 0.1534 - age_out_loss: 4.6797 - gender_out_accuracy: 0.9340 -
```

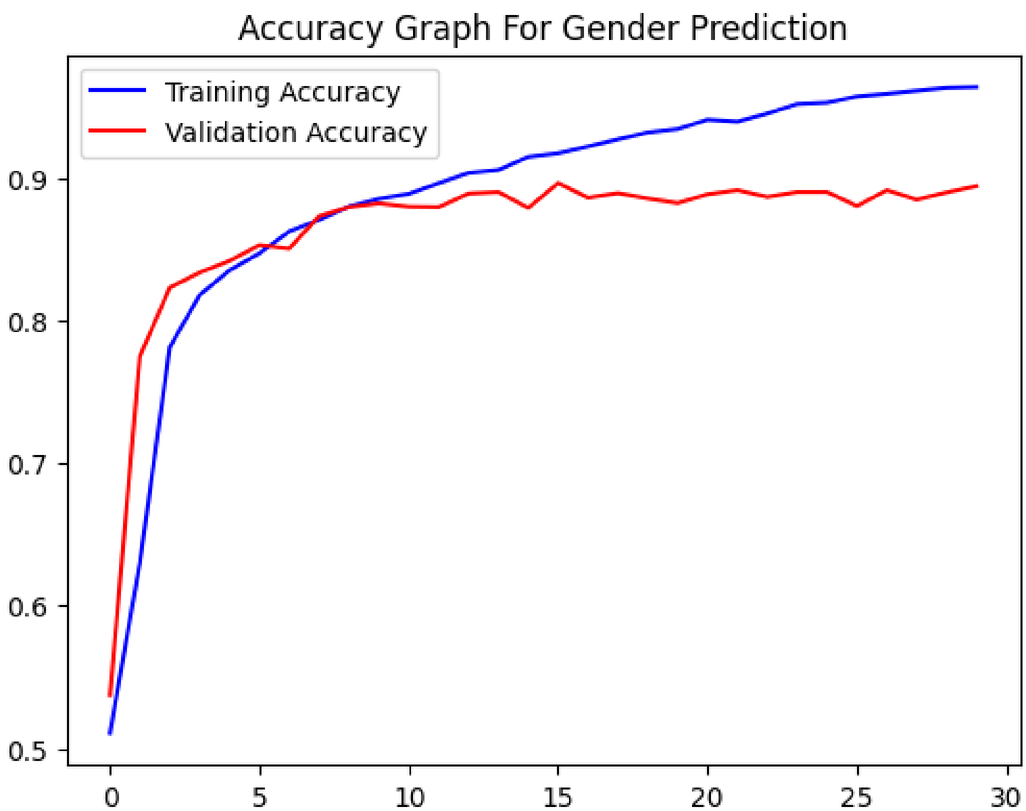
```
gender_out_mae: 0.0986 - age_out_accuracy: 0.0461 - age_out_mae: 4.6797 -
val_loss: 6.8887 - val_gender_out_loss: 0.2969 - val_age_out_loss: 6.5918 -
val_gender_out_accuracy: 0.8822 - val_gender_out_mae: 0.1361 -
val_age_out_accuracy: 0.0469 - val_age_out_mae: 6.5918
Epoch 21/30
475/475 [==============================] - 11s 24ms/step - loss: 4.7012 -
gender_out_loss: 0.1450 - age_out_loss: 4.5562 - gender_out_accuracy: 0.9405 -
gender_out_mae: 0.0926 - age_out_accuracy: 0.0456 - age_out_mae: 4.5562 -
val_loss: 6.9768 - val_gender_out_loss: 0.3125 - val_age_out_loss: 6.6643 -
val_gender_out_accuracy: 0.8882 - val_gender_out_mae: 0.1323 -
val_age_out_accuracy: 0.0456 - val_age_out_mae: 6.6643
Epoch 22/30
475/475 [==============================] - 11s 24ms/step - loss: 4.6172 -
gender_out_loss: 0.1427 - age_out_loss: 4.4745 - gender_out_accuracy: 0.9392 -
gender_out_mae: 0.0933 - age_out_accuracy: 0.0453 - age_out_mae: 4.4745 -
val_loss: 6.9775 - val_gender_out_loss: 0.2953 - val_age_out_loss: 6.6822 -
val_gender_out_accuracy: 0.8911 - val_gender_out_mae: 0.1301 -
val_age_out_accuracy: 0.0472 - val_age_out_mae: 6.6822
Epoch 23/30
475/475 [==============================] - 11s 23ms/step - loss: 4.5018 -
gender_out_loss: 0.1282 - age_out_loss: 4.3736 - gender_out_accuracy: 0.9448 -
gender_out_mae: 0.0831 - age_out_accuracy: 0.0456 - age_out_mae: 4.3736 -
val_loss: 7.2195 - val_gender_out_loss: 0.3085 - val_age_out_loss: 6.9110 -
val_gender_out_accuracy: 0.8864 - val_gender_out_mae: 0.1340 -
val_age_out_accuracy: 0.0469 - val_age_out_mae: 6.9110
Epoch 24/30
475/475 [==============================] - 13s 28ms/step - loss: 4.3616 -
gender_out_loss: 0.1161 - age_out_loss: 4.2456 - gender_out_accuracy: 0.9515 -
gender_out_mae: 0.0756 - age_out_accuracy: 0.0459 - age_out_mae: 4.2456 -
val_loss: 7.0612 - val_gender_out_loss: 0.3299 - val_age_out_loss: 6.7313 -
val_gender_out_accuracy: 0.8898 - val_gender_out_mae: 0.1339 -
val_age_out_accuracy: 0.0453 - val_age_out_mae: 6.7313
Epoch 25/30
475/475 [==============================] - 11s 22ms/step - loss: 4.2698 -
gender_out_loss: 0.1152 - age_out_loss: 4.1546 - gender_out_accuracy: 0.9525 -
gender_out_mae: 0.0752 - age_out_accuracy: 0.0456 - age_out_mae: 4.1546 -
val_loss: 6.9373 - val_gender_out_loss: 0.3486 - val_age_out_loss: 6.5887 -
val_gender_out_accuracy: 0.8898 - val_gender_out_mae: 0.1233 -
val_age_out_accuracy: 0.0456 - val_age_out_mae: 6.5887
Epoch 26/30
475/475 [==============================] - 11s 24ms/step - loss: 4.2428 -
gender_out_loss: 0.1051 - age_out_loss: 4.1378 - gender_out_accuracy: 0.9568 -
gender_out_mae: 0.0683 - age_out_accuracy: 0.0455 - age_out_mae: 4.1378 -
val_loss: 7.0968 - val_gender_out_loss: 0.3610 - val_age_out_loss: 6.7358 -
val_gender_out_accuracy: 0.8801 - val_gender_out_mae: 0.1343 -
val_age_out_accuracy: 0.0472 - val_age_out_mae: 6.7358
Epoch 27/30
475/475 [==============================] - 11s 22ms/step - loss: 4.0552 -
```

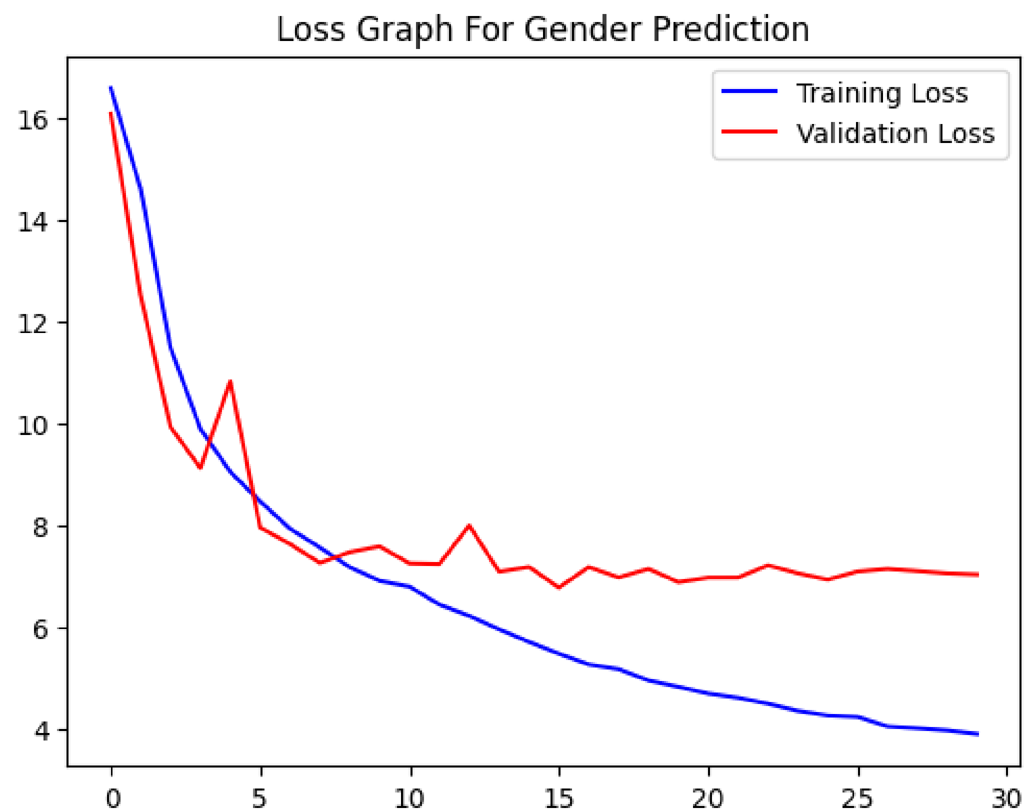
```
gender_out_loss: 0.0985 - age_out_loss: 3.9567 - gender_out_accuracy: 0.9587 -
gender_out_mae: 0.0652 - age_out_accuracy: 0.0453 - age_out_mae: 3.9567 -
val_loss: 7.1485 - val_gender_out_loss: 0.3728 - val_age_out_loss: 6.7757 -
val_gender_out_accuracy: 0.8911 - val_gender_out_mae: 0.1248 -
val_age_out_accuracy: 0.0456 - val_age_out_mae: 6.7757
Epoch 28/30
475/475 [==============================] - 11s 23ms/step - loss: 4.0191 -
gender_out_loss: 0.0928 - age_out_loss: 3.9263 - gender_out_accuracy: 0.9608 -
gender_out_mae: 0.0608 - age_out_accuracy: 0.0463 - age_out_mae: 3.9263 -
val_loss: 7.1056 - val_gender_out_loss: 0.4062 - val_age_out_loss: 6.6993 -
val_gender_out_accuracy: 0.8846 - val_gender_out_mae: 0.1272 -
val_age_out_accuracy: 0.0469 - val_age_out_mae: 6.6993
Epoch 29/30
475/475 [==============================] - 11s 23ms/step - loss: 3.9765 -
gender_out_loss: 0.0875 - age_out_loss: 3.8890 - gender_out_accuracy: 0.9628 -
gender_out_mae: 0.0574 - age_out_accuracy: 0.0453 - age_out_mae: 3.8890 -
val_loss: 7.0601 - val_gender_out_loss: 0.3844 - val_age_out_loss: 6.6756 -
val_gender_out_accuracy: 0.8896 - val_gender_out_mae: 0.1258 -
val_age_out_accuracy: 0.0456 - val_age_out_mae: 6.6756
Epoch 30/30
475/475 [==============================] - 11s 23ms/step - loss: 3.9082 -
gender_out_loss: 0.0863 - age_out_loss: 3.8219 - gender_out_accuracy: 0.9634 -
gender_out_mae: 0.0554 - age_out_accuracy: 0.0462 - age_out_mae: 3.8219 -
val_loss: 7.0379 - val_gender_out_loss: 0.3844 - val_age_out_loss: 6.6535 -
val_gender_out_accuracy: 0.8940 - val_gender_out_mae: 0.1241 -
val_age_out_accuracy: 0.0467 - val_age_out_mae: 6.6535
```

Plot Results

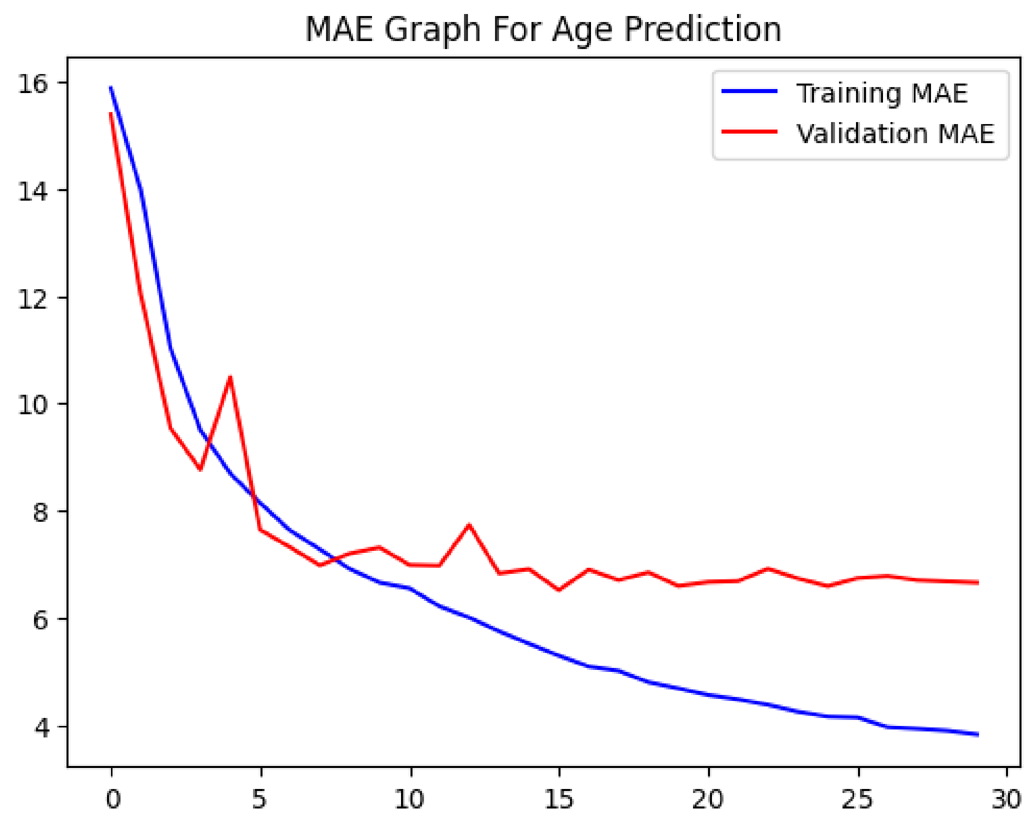
[26]:
```python
# Get training and validation accuracy for gender prediction
acc = history.history['gender_out_accuracy']
val_acc = history.history['val_gender_out_accuracy']
epochs = range(len(acc))

# Plot training and validation accuracy
plt.plot(epochs, acc, 'b', label='Training Accuracy')
plt.plot(epochs, val_acc, 'r', label='Validation Accuracy')
plt.title('Accuracy Graph For Gender Prediction')
plt.legend()
plt.show()

# Get training and validation loss for gender prediction
loss = history.history['loss']
val_loss = history.history['val_loss']

# Plot training and validation loss
plt.plot(epochs, loss, 'b', label='Training Loss')
plt.plot(epochs, val_loss, 'r', label='Validation Loss')
```

```
plt.title('Loss Graph For Gender Prediction')
plt.legend()
plt.show()
```






```
[27]: # Get training and validation MAE for age prediction
loss = history.history['age_out_mae']
val_loss = history.history['val_age_out_mae']
epochs = range(len(loss))

# Plot training and validation MAE for age prediction
plt.plot(epochs, loss, 'b', label='Training MAE')
plt.plot(epochs, val_loss, 'r', label='Validation MAE')
plt.title('MAE Graph For Age Prediction')
plt.legend()
plt.show()
```



[28]:

```
# Predictions on the test set
test_predictions = model.predict(X_test)

# Extracting gender predictions
gender_predictions = (test_predictions[0] > 0.5).astype(int).flatten()

# Calculating accuracy for gender prediction
from sklearn.metrics import accuracy_score
gender_accuracy = accuracy_score(y_gender_test, gender_predictions)

print("Gender Prediction Accuracy on Testing Set:", gender_accuracy*100,"%")

# Extracting age predictions
age_predictions = test_predictions[1].flatten()

# Calculating mean absolute error (MAE) for age prediction
from sklearn.metrics import mean_absolute_error
age_mae = mean_absolute_error(y_age_test, age_predictions)

print("Age Prediction MAE on Testing Set:", age_mae)
```

```
149/149 [==============================] - 1s 7ms/step
Gender Prediction Accuracy on Testing Set: 89.56136651202024 %
Age Prediction MAE on Testing Set: 6.748970196750887
```

[29]:
```python
# Index of the image to be evaluated
image_index = 100

# Print original gender and age labels
print("Original Gender:", gender_dict[y_gender[image_index]], "Original Age:",␣
 ↪y_age[image_index])

# Predict gender and age labels using the 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 and age labels
print("Predicted Gender:", pred_gender, "Predicted Age:", pred_age)

# Display the image
plt.axis('off')
plt.imshow(X[image_index].reshape(128, 128), cmap='gray');
```

```
Original Gender: Male Original Age: 40
1/1 [==============================] - 0s 188ms/step
Predicted Gender: Male Predicted Age: 37
```

[30]:
```python
# Index of the image to be evaluated
image_index = 3000

# Print original gender and age labels
print("Original Gender:", gender_dict[y_gender[image_index]], "Original Age:",
 ↪y_age[image_index])

# Predict gender and age labels using the 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 and age labels
print("Predicted Gender:", pred_gender, "Predicted Age:", pred_age)

# Display the image
plt.axis('off')
plt.imshow(X[image_index].reshape(128, 128), cmap='gray');
```

```
Original Gender: Female Original Age: 21
1/1 [==============================] - 0s 21ms/step
Predicted Gender: Female Predicted Age: 28
```

[31]:
```python
# Index of the image to be evaluated
image_index = 10000

# Print original gender and age labels
print("Original Gender:", gender_dict[y_gender[image_index]], "Original Age:",
 ↪y_age[image_index])

# Predict gender and age labels using the 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 and age labels
print("Predicted Gender:", pred_gender, "Predicted Age:", pred_age)

# Display the image
plt.axis('off')
plt.imshow(X[image_index].reshape(128, 128), cmap='gray');
```

```
Original Gender: Female Original Age: 12
1/1 [==============================] - 0s 18ms/step
Predicted Gender: Female Predicted Age: 10
```

[32]:
```python
# Index of the image to be evaluated
image_index = 5000

# Print original gender and age labels
print("Original Gender:", gender_dict[y_gender[image_index]], "Original Age:",␣
 ↪y_age[image_index])

# Predict gender and age labels using the 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 and age labels
print("Predicted Gender:", pred_gender, "Predicted Age:", pred_age)

# Display the image
plt.axis('off')
plt.imshow(X[image_index].reshape(128, 128), cmap='gray');
```

```
Original Gender: Male Original Age: 32
1/1 [==============================] - 0s 24ms/step
Predicted Gender: Male Predicted Age: 31
```

[33]:
```python
# Index of the image to be evaluated
image_index = 2000

# Print original gender and age labels
print("Original Gender:", gender_dict[y_gender[image_index]], "Original Age:",␣
 ↪y_age[image_index])

# Predict gender and age labels using the 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 and age labels
print("Predicted Gender:", pred_gender, "Predicted Age:", pred_age)

# Display the image
plt.axis('off')
plt.imshow(X[image_index].reshape(128, 128), cmap='gray');
```

```
Original Gender: Female Original Age: 36
1/1 [==============================] - 0s 26ms/step
Predicted Gender: Female Predicted Age: 38
```

[37]:

```python
# Index of the image to be evaluated
image_index = 2300

# Print original gender and age labels
print("Original Gender:", gender_dict[y_gender[image_index]], "Original Age:",␣
 ↪y_age[image_index])

# Predict gender and age labels using the 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 and age labels
print("Predicted Gender:", pred_gender, "Predicted Age:", pred_age)

# Display the image
plt.axis('off')
plt.imshow(X[image_index].reshape(128, 128), cmap='gray');
```

```
Original Gender: Male Original Age: 2
1/1 [==============================] - 0s 42ms/step
Predicted Gender: Male Predicted Age: 3
```

[35]:
```python
# Index of the image to be evaluated
image_index = 2900

# Print original gender and age labels
print("Original Gender:", gender_dict[y_gender[image_index]], "Original Age:",␣
    ↪y_age[image_index])

# Predict gender and age labels using the 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 and age labels
print("Predicted Gender:", pred_gender, "Predicted Age:", pred_age)

# Display the image
plt.axis('off')
plt.imshow(X[image_index].reshape(128, 128), cmap='gray');
```

```
Original Gender: Female Original Age: 8
1/1 [==============================] - 0s 20ms/step
Predicted Gender: Female Predicted Age: 9
```

[36]:
```python
# Index of the image to be evaluated
image_index = 2709

# Print original gender and age labels
print("Original Gender:", gender_dict[y_gender[image_index]], "Original Age:",
 ↪y_age[image_index])

# Predict gender and age labels using the 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 and age labels
print("Predicted Gender:", pred_gender, "Predicted Age:", pred_age)

# Display the image
plt.axis('off')
plt.imshow(X[image_index].reshape(128, 128), cmap='gray');
```

```
Original Gender: Male Original Age: 21
1/1 [==============================] - 0s 19ms/step
Predicted Gender: Male Predicted Age: 24
```