In [3]:

```
import os
import cv2
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import LabelEncoder
import time
import tensorflow as tf
from tensorflow import keras
import numpy as np
from sklearn.metrics import accuracy_score
```

In [4]:

```
folder_paths = [
    "C:/Users/Administrator/Desktop/Humans/female",
    "C:/Users/Administrator/Desktop/Humans/male"
]
```

In [5]:

```
dataset = []
```

In [6]:

```
for i in folder_paths:
    folder_name = os.path.basename(i)

    # Iterate over the images in the subdirectory
    for file_name in os.listdir(i):
        image_path = os.path.join(i, file_name)

        if os.path.isfile(image_path):  # Only consider files
            # Load the image using OpenCV
            image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

            # If the image was successfully loaded
            if image is not None:
                # Resize the grayscale image to 250x250 pixels
                resized_image = cv2.resize(image, (200, 200))

                # Flatten the image and append each pixel as a separate feature
                flattened_image = resized_image.flatten().tolist()

                # Append the flattened image along with the folder name to the dataset
                dataset.append(flattened_image + [folder_name])
```

In [7]:
```
columns = [f"pixel_{i}" for i in range(200 * 200)] + ["label"]
df = pd.DataFrame(dataset, columns=columns)
df = df.sample(frac=1, random_state=42)
df
```

Out[7]:

| | pixel_0 | pixel_1 | pixel_2 | pixel_3 | pixel_4 | pixel_5 | pixel_6 | pixel_7 | pixel_8 | pixel_9 | ... | pixel_39991 | pixel_39992 | pixel_39 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **132** | 3 | 3 | 4 | 6 | 8 | 10 | 11 | 14 | 12 | 8 | ... | 20 | 20 | |
| **1662** | 184 | 183 | 182 | 183 | 185 | 185 | 183 | 182 | 184 | 185 | ... | 190 | 192 | |
| **5241** | 98 | 101 | 103 | 123 | 142 | 120 | 149 | 172 | 127 | 106 | ... | 244 | 245 | |
| **6807** | 194 | 186 | 184 | 186 | 183 | 181 | 181 | 176 | 174 | 178 | ... | 23 | 28 | |
| **3460** | 124 | 125 | 126 | 125 | 127 | 126 | 125 | 125 | 125 | 124 | ... | 140 | 136 | |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| **3772** | 10 | 10 | 10 | 10 | 10 | 10 | 11 | 12 | 12 | 12 | ... | 78 | 89 | |
| **5191** | 234 | 234 | 234 | 233 | 232 | 231 | 229 | 229 | 230 | 230 | ... | 229 | 230 | |
| **5226** | 30 | 70 | 62 | 28 | 50 | 53 | 42 | 13 | 36 | 5 | ... | 30 | 32 | |
| **5390** | 252 | 252 | 252 | 252 | 252 | 252 | 252 | 252 | 252 | 252 | ... | 252 | 252 | |
| **860** | 38 | 38 | 38 | 38 | 39 | 39 | 39 | 39 | 39 | 39 | ... | 45 | 46 | |

6844 rows × 40001 columns

In [8]:
```
X = df.drop(columns=['label'])  # Features
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)
X_scaled
```

Out[8]:
```
array([[0.01176471, 0.01176471, 0.01568627, ..., 0.05098039, 0.05098039,
        0.05098039],
       [0.72156863, 0.71764706, 0.71372549, ..., 0.75294118, 0.74901961,
        0.74509804],
       [0.38431373, 0.39607843, 0.40392157, ..., 0.94117647, 0.95294118,
        0.94509804],
       ...,
       [0.11764706, 0.2745098 , 0.24313725, ..., 0.10588235, 0.09019608,
        0.04705882],
       [0.98823529, 0.98823529, 0.98823529, ..., 0.98823529, 0.98823529,
        0.98823529],
       [0.14901961, 0.14901961, 0.14901961, ..., 0.18039216, 0.18039216,
        0.17647059]])
```

In [9]:
```
y = df['label']
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)
y_encoded
```

Out[9]:
```
array([0, 0, 1, ..., 1, 1, 0])
```

In [10]:
```
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_encoded, test_size=0.2, random_state=42
```

In [11]:

```
# Create a Sequential model
model = keras.Sequential([
    keras.layers.Dense(128, activation='relu', input_shape=(X_train.shape[1],)),
    keras.layers.Dense(64, activation='relu'),
    keras.layers.Dense(1, activation='sigmoid')
])

# Compile the model
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])

start_time = time.time()

# Train the model
model.fit(X_train, y_train, epochs=10, batch_size=32, validation_split=0.2)

training_time = time.time() - start_time
print("Training Time:", training_time, "seconds")

# Make predictions on the testing data
y_pred = model.predict(X_test)
y_pred_classes = np.round(y_pred)  # Round the probabilities to get binary predictions

# Evaluate the model's accuracy
accuracy = accuracy_score(y_test, y_pred_classes)
print("Accuracy:", accuracy)
```

```
Epoch 1/10
137/137 [==============================] - 83s 175ms/step - loss: 1.0990 - accuracy: 0.6212 - val_los
s: 0.5811 - val_accuracy: 0.6959
Epoch 2/10
137/137 [==============================] - 20s 149ms/step - loss: 0.7944 - accuracy: 0.6541 - val_los
s: 0.5607 - val_accuracy: 0.7005
Epoch 3/10
137/137 [==============================] - 21s 150ms/step - loss: 0.5777 - accuracy: 0.7128 - val_los
s: 0.5751 - val_accuracy: 0.6950
Epoch 4/10
137/137 [==============================] - 21s 151ms/step - loss: 0.5500 - accuracy: 0.7311 - val_los
s: 0.5234 - val_accuracy: 0.7425
Epoch 5/10
137/137 [==============================] - 21s 154ms/step - loss: 0.5166 - accuracy: 0.7447 - val_los
s: 0.5746 - val_accuracy: 0.7096
Epoch 6/10
137/137 [==============================] - 20s 148ms/step - loss: 0.4971 - accuracy: 0.7541 - val_los
s: 0.5858 - val_accuracy: 0.7187
Epoch 7/10
137/137 [==============================] - 20s 148ms/step - loss: 0.4671 - accuracy: 0.7756 - val_los
s: 0.5183 - val_accuracy: 0.7461
Epoch 8/10
137/137 [==============================] - 20s 148ms/step - loss: 0.4335 - accuracy: 0.8016 - val_los
s: 0.4833 - val_accuracy: 0.7708
Epoch 9/10
137/137 [==============================] - 20s 146ms/step - loss: 0.4970 - accuracy: 0.7676 - val_los
s: 0.4654 - val_accuracy: 0.7909
Epoch 10/10
137/137 [==============================] - 20s 146ms/step - loss: 0.4124 - accuracy: 0.8078 - val_los
s: 0.4505 - val_accuracy: 0.7909
Training Time: 439.6637508869171 seconds
43/43 [==============================] - 3s 22ms/step
Accuracy: 0.7918188458729
```

In [ ]:

In [ ]: