AGE DETECTION MODEL

TEAM STRANGE

JAYANT KR PANDEY (LEADER)

PRIYANSHU CHOWDHURY

YASH ARYAN SINGH

VISHAL KR MANDAL

RITIK KUMAR

PURNENDU PRITAM DAS

INTRODUCTION:

Aging, an inevitable and intricate biological process, has always intrigued humanity. Understanding age not only helps us gain insight into the human condition but also has profound implications across various fields, such as healthcare, entertainment, marketing, and security. Deep learning, a subset of machine learning, has emerged as a potent tool to unravel age-related mysteries by analyzing patterns, textures, and features in images.

Our project embarks on a journey to explore the capabilities of deep learning in age estimation, seeking to answer fundamental questions such as:

1. How accurately can deep learning algorithms predict a person's age based on facial features and images?

2. What role do convolutional neural networks (CNNs), recurrent neural networks (RNNs), and other deep learning architectures play in age detection?

3. What datasets are available for training and testing age detection models, and how do they contribute to the accuracy of predictions?

DATA SOURCE:

https://drive.google.com/drive/folders/1E9m9dZYLga9kc9NGPHfZa75JNJgpgv3M

The folders in the dataset are named according to the age of people in it.

IMPORTING LIBRARIES:

```
Importing Libraries
```

```python
import pandas as pd
import tensorflow as tf
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
import cv2
import os
```

[2] ✓ 0.0s Python

DATA PRE-PROCESSING:

```python
parent_folder = 'face_age'
image_arrays = []
for age_folder in os.listdir(parent_folder):
    age_folder_path = os.path.join(parent_folder, age_folder)
    if os.path.isdir(age_folder_path):
        for filename in os.listdir(age_folder_path):
            if filename.endswith(('.jpg', '.jpeg', '.png', '.bmp')):
                image_path = os.path.join(age_folder_path, filename)
                img = cv2.imread(image_path)
                if img is not None:
                    img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

                    if int(age_folder)<=5:
                        image_arrays.append((img_gray,1))

                    elif int(age_folder)<=12:
                        image_arrays.append((img_gray,2))

                    elif int(age_folder)<=18:
                        image_arrays.append((img_gray,3))

                    elif int(age_folder)<=30:
                        image_arrays.append((img_gray,4))

                    elif int(age_folder)<=45:
                        image_arrays.append((img_gray,5))

                    elif int(age_folder)<=65:
                        image_arrays.append((img_gray,6))

                    elif int(age_folder)<=80:
                        image_arrays.append((img_gray,7))

                    else:
                        image_arrays.append((img_gray,8))
```

```python
image_arrays[1]
```

```
(array([[239, 237, 235, ..., 178, 177, 177],
        [240, 237, 235, ..., 178, 177, 177],
        [240, 238, 235, ..., 177, 176, 176],
        ...,
        [237, 236, 236, ..., 155, 158, 160],
        [237, 236, 236, ..., 158, 159, 162],
        [237, 236, 236, ..., 160, 161, 163]], dtype=uint8),
 1)
```

Age Grouping:

We created age groups, such as 0-5, 6-12, 13-18, 19-30, 31-45, 46-60, 61-80, and 81-95, in our machine learning model for simplifying the modeling process, enhancing interpretability, and aligning with the practical significance of these groups in our application. These age groups were chosen after thorough experimentation and analysis to maximize model accuracy and performance while ensuring compliance with ethical considerations. They facilitate decision-making, scalability, and maintenance, making them a suitable approach for addressing our project's specific needs and objectives.

DATA AUGMENTATION:

- Splitting Data:

```python
from sklearn.model_selection import train_test_split

train_data, temp_data = train_test_split(image_arrays, test_size=0.3, random_state=43)

val_data, test_data = train_test_split(temp_data, test_size=0.5, random_state=42)
```

- Separating Dependent and independent variable:

Separating Dependent and Independent variable

```python
x_train = [item[0] for item in train_data]
y_train = [item[1] for item in train_data]

x_test = [item[0] for item in test_data]
y_test = [item[1] for item in test_data]

x_val = [item[0] for item in val_data]
y_val = [item[1] for item in val_data]
```

Looking structure of data

```python
x_train[0]
```

```
array([[ 76,  74,  68, ...,  77,  82,  85],
       [ 77,  73,  68, ...,  76,  80,  83],
       [ 77,  73,  68, ...,  75,  77,  79],
       ...,
       [ 55,  53,  50, ...,  48,  72, 100],
       [ 54,  56,  55, ...,  46,  68,  95],
       [ 55,  58,  60, ...,  41,  62,  89]], dtype=uint8)
```

```python
y_train[0]
```

```
5
```

```python
x_test[0]
```

```
array([[ 2,  1,  1, ..., 30, 32, 33],
       [ 2,  1,  1, ..., 29, 31, 32],
       [ 1,  1,  1, ..., 28, 29, 30],
       ...,
       [23, 21, 19, ...,  8, 13, 16],
       [23, 21, 19, ...,  8, 13, 16],
       [22, 21, 19, ...,  8, 13, 15]], dtype=uint8)
```

```python
len(x_train)
```

```
6828
```

```python
len(x_test)
```

```
1464
```

```python
len(x_val)
```

```
1463
```

- Resizing image:

```python
x_train[0].shape
```

```
(200, 200)
```

```
Resizing images pixel
```

```python
new_size = (100, 100)
resized_x_train = []

for image in x_train:
    resized_image = cv2.resize(image, new_size)
    resized_x_train.append(resized_image)


resized_x_train = np.array(resized_x_train)

print(resized_x_train.shape)

resized_x_test = []


for image in x_test:
    resized_image = cv2.resize(image, new_size)
    resized_x_test.append(resized_image)


resized_x_test = np.array(resized_x_test)

resized_x_val = []

for image in x_val:
    resized_image = cv2.resize(image, new_size)
    resized_x_val.append(resized_image)

resized_x_val = np.array(resized_x_val)
```

```
(6828, 100, 100)
```

- Normalizing Data Set

```
Converting to numpy arrey
```

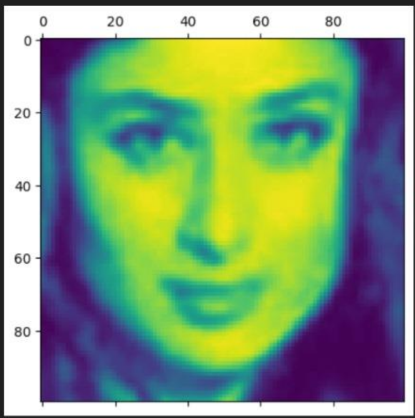
```python
x_train_np = np.array(x_train)
x_test_np = np.array(x_test)
x_val_np = np.array(x_val)
y_train_np = np.array(y_train)
y_test_np = np.array(y_test)
y_val_np = np.array(y_val)

x_train_normalized = resized_x_train / 255.0
x_test_normalized = resized_x_test / 255.0
x_val_normalized = resized_x_val / 255.0
```

- Visualizing Dataset:

```
plt.matshow(resized_x_test[0])
```

[20] ✓ 0.1s Python

```
<matplotlib.image.AxesImage at 0x1e0b50037f0>
```




```
x_test_normalized[0]
```

[23] ✓ 0.0s Python

```
array([[0.00784314, 0.00392157, 0.00784314, ..., 0.08627451, 0.11372549,
        0.1254902 ],
       [0.00392157, 0.00392157, 0.00392157, ..., 0.0745098 , 0.10196078,
        0.11372549],
       [0.        , 0.        , 0.00392157, ..., 0.07058824, 0.09411765,
        0.10980392],
       ...,
       [0.09019608, 0.07843137, 0.07843137, ..., 0.02745098, 0.03529412,
        0.06666667],
       [0.08627451, 0.0745098 , 0.0745098 , ..., 0.02352941, 0.03137255,
        0.05882353],
       [0.08627451, 0.0745098 , 0.0745098 , ..., 0.02352941, 0.03137255,
        0.05490196]])
```

```
x_test_normalized.shape
```

[24] ✓ 0.0s Python

```
(1464, 100, 100)
```

```
x_train_normalized.shape
```

[25] ✓ 0.0s Python

```
(6828, 100, 100)
```

- Flattering Dataset:

Flattering X_dataset

markdown

```
X_train_flattened = x_train_normalized.reshape(len(x_train_normalized), 100*100)
X_test_flattened = x_test_normalized.reshape(len(x_test_normalized), 100*100)
X_val_flattened = x_val_normalized.reshape(len(x_val_normalized), 100*100)
```

[26] ✓ 0.0s Python

```
X_train_flattened.shape
```

[27] ✓ 0.0s Python

```
(6828, 10000)
```

- Convert labels to one-hot encoded format:

Convert labels to one-hot encoded format

markdown

```
num_classes = 96

from keras.utils import to_categorical
y_train_one_hot = to_categorical(y_train_np, num_classes=num_classes)
y_test_one_hot = to_categorical(y_test_np, num_classes=num_classes)
y_val_one_hot = to_categorical(y_val_np, num_classes=num_classes)
```

[29] ✓ 0.0s Python

- Reshaping Dataset:

Reshaping x_dataset

markdown

```
x_train_reshaped = X_train_flattened.reshape(-1, 100, 100, 1)
x_val_reshaped = X_val_flattened.reshape(-1, 100, 100, 1)
x_test_reshaped = X_test_flattened.reshape(-1, 100, 100, 1)
```

[30] ✓ 0.0s Python

MODEL ARCHITECTURE:

CNN Architecture

```python
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.regularizers import l2

model = Sequential([
    Conv2D(32, kernel_size=(3, 3), input_shape=(100, 100, 1), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Conv2D(256, kernel_size=(3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Conv2D(512, kernel_size=(3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Conv2D(256, kernel_size=(3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(9, activation='softmax')
])
optimizer = Adam(learning_rate=0.001)
model.compile(optimizer=optimizer,
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

[35] ✓ 0.9s Python

MODEL TRAINING:

Model training

+ Code + Markdown

```python
from tensorflow.keras.callbacks import EarlyStopping

early_stopping = EarlyStopping(
    monitor='val_accuracy',
    patience=10,
    restore_best_weights=True
)

history = model.fit(
    x_train_reshaped,
    y_train_one_hot,
    epochs=100,batch_size=16,
    validation_data=(x_val_reshaped, y_val_one_hot),
    callbacks=[early_stopping]
)
```

[35] ✓ 3m 5.0s Python

```
Epoch 1/100
427/427 [==============================] - 15s 26ms/step - loss: 1.6532 - accuracy: 0.3650 - val_loss: 1.3653 - val_accuracy: 0.4675
Epoch 2/100
427/427 [==============================] - 10s 23ms/step - loss: 1.2206 - accuracy: 0.5267 - val_loss: 1.1111 - val_accuracy: 0.5379
Epoch 3/100
427/427 [==============================] - 10s 23ms/step - loss: 1.0810 - accuracy: 0.5709 - val_loss: 1.0469 - val_accuracy: 0.5830
Epoch 4/100
427/427 [==============================] - 10s 23ms/step - loss: 0.9742 - accuracy: 0.6112 - val_loss: 1.0894 - val_accuracy: 0.5783
Epoch 5/100
427/427 [==============================] - 10s 23ms/step - loss: 0.9039 - accuracy: 0.6353 - val_loss: 1.0164 - val_accuracy: 0.5762
Epoch 6/100
427/427 [==============================] - 10s 23ms/step - loss: 0.8271 - accuracy: 0.6658 - val_loss: 1.0028 - val_accuracy: 0.6049
Epoch 7/100
427/427 [==============================] - 10s 23ms/step - loss: 0.7633 - accuracy: 0.6861 - val_loss: 1.0330 - val_accuracy: 0.5803
Epoch 8/100
427/427 [==============================] - 10s 23ms/step - loss: 0.7036 - accuracy: 0.7154 - val_loss: 1.0089 - val_accuracy: 0.6186
Epoch 9/100
427/427 [==============================] - 10s 23ms/step - loss: 0.6208 - accuracy: 0.7525 - val_loss: 1.0100 - val_accuracy: 0.6124
Epoch 10/100
427/427 [==============================] - 10s 23ms/step - loss: 0.5513 - accuracy: 0.7743 - val_loss: 1.0989 - val_accuracy: 0.6104
Epoch 11/100
427/427 [==============================] - 10s 23ms/step - loss: 0.4839 - accuracy: 0.8084 - val_loss: 1.1833 - val_accuracy: 0.6042
Epoch 12/100
427/427 [==============================] - 10s 23ms/step - loss: 0.4131 - accuracy: 0.8363 - val_loss: 1.3691 - val_accuracy: 0.5878
Epoch 13/100
427/427 [==============================] - 10s 23ms/step - loss: 0.3429 - accuracy: 0.8626 - val_loss: 1.3977 - val_accuracy: 0.5933
Epoch 14/100
427/427 [==============================] - 10s 23ms/step - loss: 0.2900 - accuracy: 0.8903 - val_loss: 1.6242 - val_accuracy: 0.6124
Epoch 15/100
427/427 [==============================] - 10s 23ms/step - loss: 0.2614 - accuracy: 0.9001 - val_loss: 1.5280 - val_accuracy: 0.5871
Epoch 16/100
427/427 [==============================] - 10s 23ms/step - loss: 0.2249 - accuracy: 0.9145 - val_loss: 1.8085 - val_accuracy: 0.5892
Epoch 17/100
427/427 [==============================] - 10s 23ms/step - loss: 0.2104 - accuracy: 0.9209 - val_loss: 2.1422 - val_accuracy: 0.5954
Epoch 18/100
427/427 [==============================] - 10s 23ms/step - loss: 0.1764 - accuracy: 0.9348 - val_loss: 2.0044 - val_accuracy: 0.5960
```

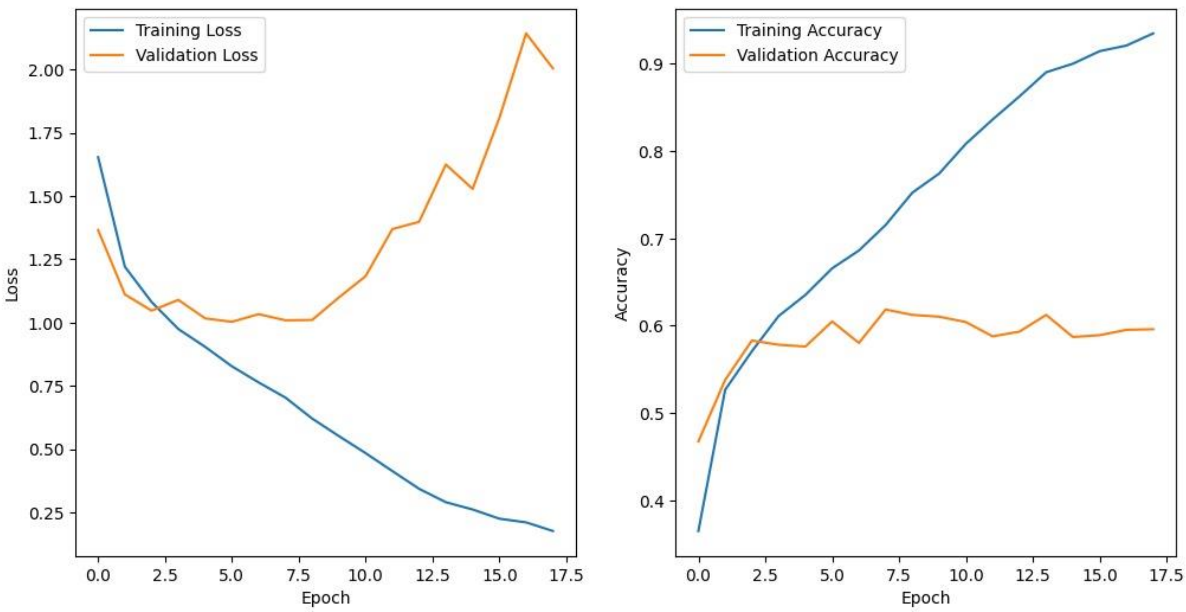
LOSS AND ACCURACY CURVE:

```
Plotting Loss and Accuracy curves
```

markdown

```python
train_loss = history.history['loss']
val_loss = history.history['val_loss']
train_acc = history.history['accuracy']
val_acc = history.history['val_accuracy']


plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.plot(train_loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()


plt.subplot(1, 2, 2)
plt.plot(train_acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```

[41] ✓ 0.2s Python



EVALUATION:

Evaluation on test data

```python
x_test_reshaped = x_test_reshaped.reshape(-1, 100, 100, 1)
```

[42] ✓ 0.0s

```python
y_predict = model.predict(x_test_reshaped,batch_size=16)
```

[43] ✓ 1.1s

```
92/92 [==============================] - 1s 9ms/step
```

```python
model.evaluate(x_test_reshaped, y_test_one_hot, batch_size=16)
```

[44] ✓ 1.1s

```
92/92 [==============================] - 1s 10ms/step - loss: 1.0774 - accuracy: 0.6093
```

```
[1.077383279800415, 0.6092896461486816]
```

CONFUSION MATRIX:

Plotting confusion Matrix

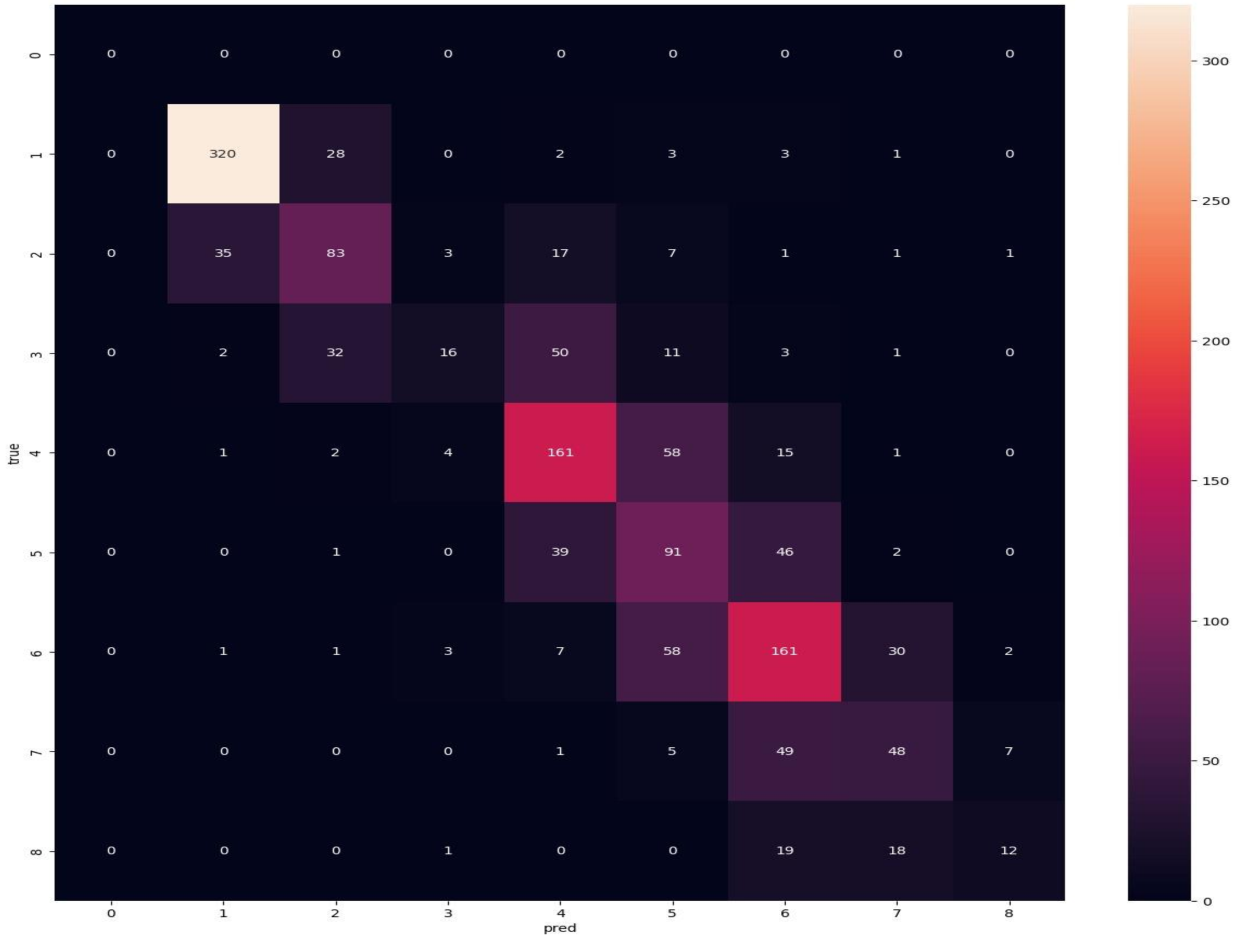
```python
y_predict_labels=[np.argmax(i) for i in y_predict]
cn=tf.math.confusion_matrix(labels=y_test_np,predictions=y_predict_labels)
```

[45] ✓ 0.0s Python

```python
import seaborn as sn
plt.figure(figsize=(15,15))
sn.heatmap(cn, annot=True, fmt='d')
plt.xlabel('pred')
plt.ylabel('true')
```

[46] ✓ 0.4s Python

```
Text(158.22222222222223, 0.5, 'true')
```



SAVING THE MODEL:

Saving the model

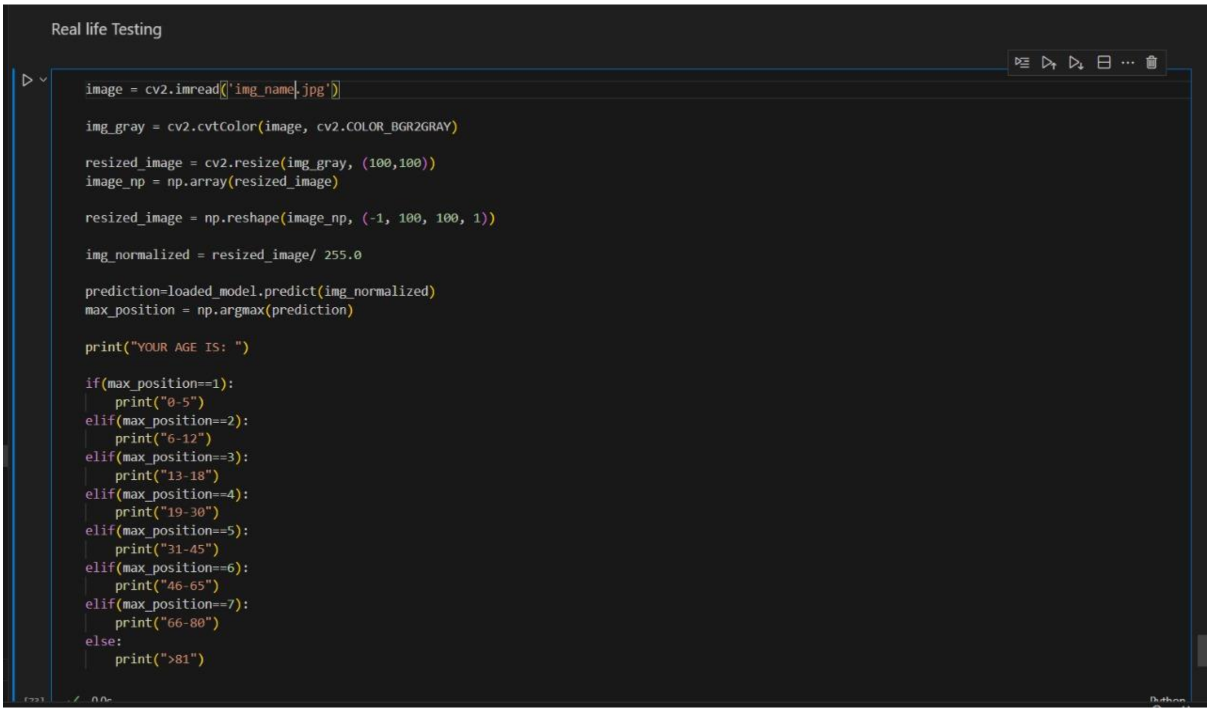
```python
def predict_with_model(model, input_data):
    return model(input_data)

# Save the model
model.save("saved_model")
```

Python

```python
import tensorflow as tf

loaded_model = tf.keras.models.load_model("saved_model")
```

Python

Real Time Testing



Real life Testing

```python
image = cv2.imread('img_name.jpg')

img_gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

resized_image = cv2.resize(img_gray, (100,100))
image_np = np.array(resized_image)

resized_image = np.reshape(image_np, (-1, 100, 100, 1))

img_normalized = resized_image/ 255.0

prediction=loaded_model.predict(img_normalized)
max_position = np.argmax(prediction)

print("YOUR AGE IS: ")

if(max_position==1):
    print("0-5")
elif(max_position==2):
    print("6-12")
elif(max_position==3):
    print("13-18")
elif(max_position==4):
    print("19-30")
elif(max_position==5):
    print("31-45")
elif(max_position==6):
    print("46-65")
elif(max_position==7):
    print("66-80")
else:
    print(">81")
```