Age Detection System – Project Report

Ammar Ahmed (023-19-0107, Sec A)

Submitted To: Dr. Asif Ali Rajput

Introduction:

Age Detection System is an application that can estimate the person's age from an image that is given as an input. In recent years, many areas such as attendance tracking, law enforcement, public security, banking, airports, and so on have been using facial recognition systems that contain information about individuals such as ethnic background, gender, and age. Thus, application systems such as this one are becoming more prevalent by the day in many areas. The application uses a Convolutional Neural Network model that is trained from the scratch using the publicly available dataset called UTKFace Dataset.

Methodology:

The CNN model was trained from scratch using the UTKFace dataset, which consists of around 23,000 images of people with ages that range from 1-100 years. The model will classify images into one of 100 classes. The UTKFace dataset is not designed just for facial age estimation but also includes many other parameters such as gender and ethnicity, which means it can be used for various other tasks. However, we will be only interested in the age part of the dataset. Note that the dataset consists of only images that are named in the format (age_gender_ethnicity). Therefore, we will have to manually construct a dataset such that we have only images that have the corresponding age as a label.

Following is the format of the images:

Step 1:

Since the dataset only comes with images, the first task is to split the dataset into training and testing. And keep the training and testing data in separate folders so that we can read the data later. This is done using the 'OS module' in python. The algorithm that I

used to achieve this was publicly available. The algorithm iteratively looped through the folder of images, taking each image and randomly putting it in either the testing folder or the training folder based on the condition that if the current index is divisible by 10 then put it in the testing folder else, put it in the training folder. Note that by applying this condition we will end up with fewer images in the testing folder compared to that of the training, which is exactly what we want.

Step 2:

Once we have the training and testing images in their respective folders, we can move on to discussing how to load this data into the code. For this, I designed a method called load_and_return that takes two arguments; the source path of the folder, and image size. It declares two lists called loaded_imgs and loaded_lbls. It then iterates through each image inside the source path and resizes it by specified image size and converts it into a grayscale image (as part of normalization) before finally appending it to the image list. As for the label, we split the name of each image and take only the age part out of the name format (age_gender_ethnicity). These labels are then appended to the labels list before finally being transformed into a binary array using Label Binarizer. At this point, we should have the required images and their corresponding labels. The method simply returns these two lists.

Step 3:

The next step is to create a method that returns our Convolutional Neural Network model. The method is called getModel which takes 5 arguments. The image size, number of kernels that we will use at each convolution layer, total number of classes, number of hidden layer neurons, and the number of convolution layers. And then train that model. However, before training this model, I had to make some adjustments. First, it would not be efficient to have 100 classes, since it would be very hard to get good accuracy in a reasonable amount of time given the huge amount of data (I learned this the hard way, while training the model). So, I had to define an age range. For this experiment, I used following age ranges: [1, 11, 21, 32, 43, 54, 65, 71, 82, 90, 102]. Any person having an age that lies between any two adjacent ages; say 21 and 32, will be classified as 32 years old. This reduces the classes down to 10. I achieved this by including the third parameter called age_range in the load_and_return method discussed in step 2.

Step 4:

To construct the model, I used 5 convolutional layers, 5 max-pooling layers, and 3 dense layers. The first three convolutional layers had 32 kernels with (5x5) as the size of each

kernel and the other 2 convolutional layers had 64 kernels, each of size (3x3). All 5 max-pooling layers had the same pool and stride size of (2x2). Finally, for the three dense layers, the first two had 256 neurons, and the last dense layer had 10 neurons with activation function as SoftMax since we will be interested in the probabilities for each class label. Finally, the model was compiled using optimizer SGD with a learning rate of 0.01.

Step 5:

For training our model, I gave the training data and training labels loaded in step 2 to our model to train on. The epoch size was set at 20 and the batch size was set at 128. Even with an epoch size of 20, it took about 420mins ~ 7 hours to train the model on approximately 21,000 training images and labels. The final achieved accuracy on training was around 63%. This was enough for me to test the model on testing data. The accuracy of testing data was around 50% which was satisfactory. Although, I could have increased the model's efficiency by increasing the number of epochs but it took way too long to train as mentioned earlier.

Following are the results of the training:

```
Epoch 1/20
167/167 [==============================] - 891s 5s/step - loss: 1.8039 - accuracy: 0.3815 - val_loss: 2.2653 -
val_accuracy: 0.1423
Epoch 2/20
167/167 [==============================] - 948s 6s/step - loss: 1.5102 - accuracy: 0.4583 - val_loss: 1.8363 -
val_accuracy: 0.3174
Epoch 3/20
167/167 [==============================] - 1133s 7s/step - loss: 1.4061 - accuracy: 0.4821 - val_loss: 1.5746 -
val_accuracy: 0.4111
Epoch 4/20
167/167 [==============================] - 1012s 6s/step - loss: 1.3303 - accuracy: 0.4989 - val_loss: 1.5202 -
val_accuracy: 0.4766
Epoch 5/20
167/167 [==============================] - 1086s 6s/step - loss: 1.2780 - accuracy: 0.5144 - val_loss: 1.4410 -
val_accuracy: 0.4989
Epoch 6/20
167/167 [==============================] - 1128s 7s/step - loss: 1.2358 - accuracy: 0.5227 - val_loss: 2.0743 -
val_accuracy: 0.4588
```

```
Epoch 10/20
167/167 [==============================] - 1597s 9s/step - loss: 1.1160 - accuracy: 0.5567 - val_loss: 2.4253 -
val_accuracy: 0.4614
Epoch 11/20
167/167 [==============================] - 2192s 13s/step - loss: 1.1008 - accuracy: 0.5628 - val_loss: 1.2321 -
val_accuracy: 0.5454
Epoch 12/20
167/167 [==============================] - 2095s 13s/step - loss: 1.0727 - accuracy: 0.5710 - val_loss: 1.3114 -
val_accuracy: 0.5378
Epoch 13/20
...
Epoch 19/20
167/167 [==============================] - 1180s 7s/step - loss: 0.9448 - accuracy: 0.6141 - val_loss: 1.3117 -
val_accuracy: 0.5365
Epoch 20/20
167/167 [==============================] - 1179s 7s/step - loss: 0.9252 - accuracy: 0.6235 - val_loss: 1.5728 -
val_accuracy: 0.4141
```

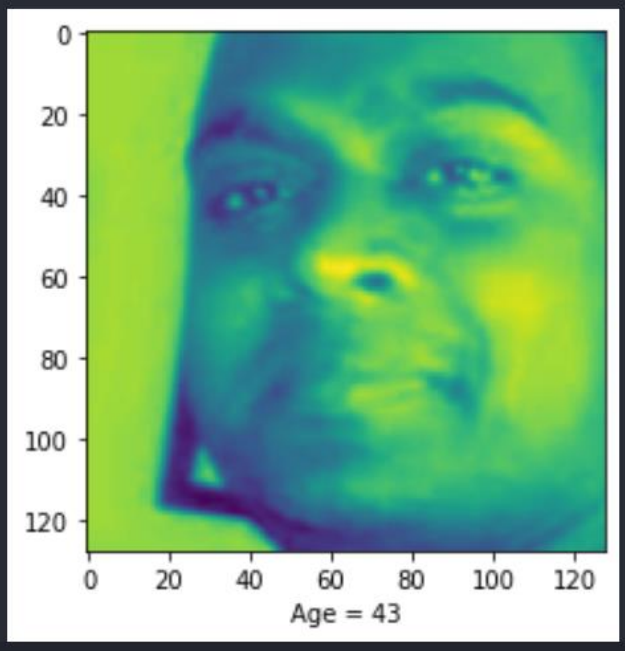
Evaluation:

Results of the model on testing data:

```
1  i = randint(0, 900)
2  plt.imshow(test_imgs[i])
3  plt.xlabel(f'Age = {age_range[np.argmax(y_pred[i])]}')
4
```

✓ 1.3s

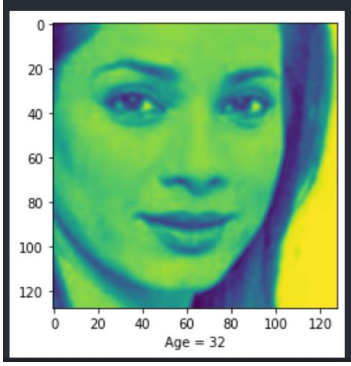
```
Text(0.5, 0, 'Age = 43')
```



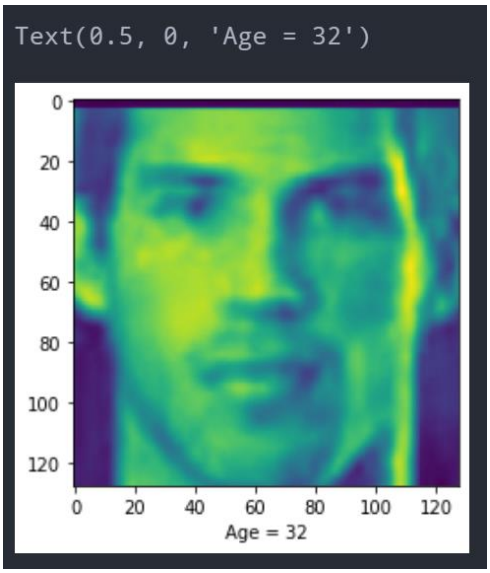

```
1  i = randint(0, 900)
2  plt.imshow(test_imgs[i])
3  plt.xlabel(f'Age = {age_range[np.argmax(y_pred[i])]}')
4
```

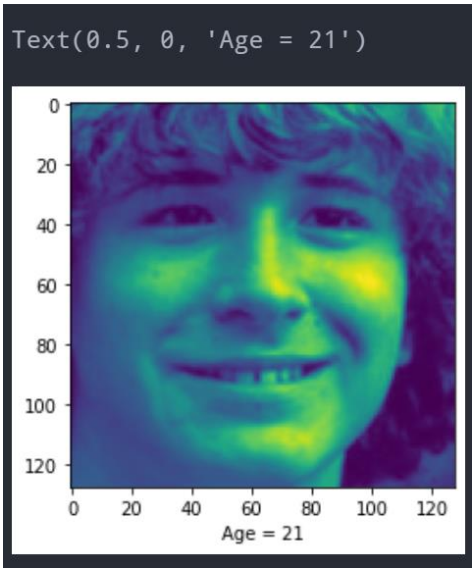
✓ 2.8s Python

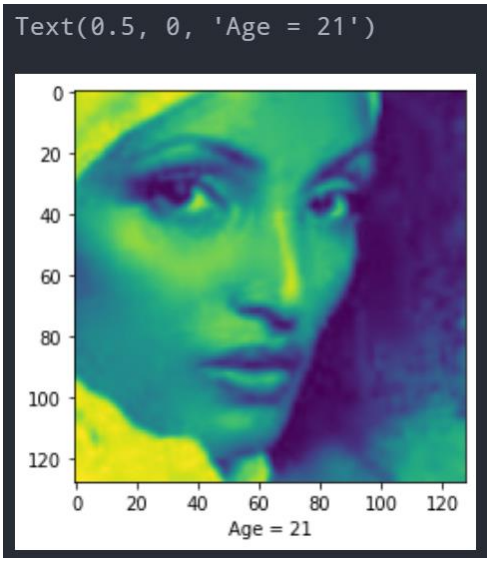
```
Text(0.5, 0, 'Age = 32')
```

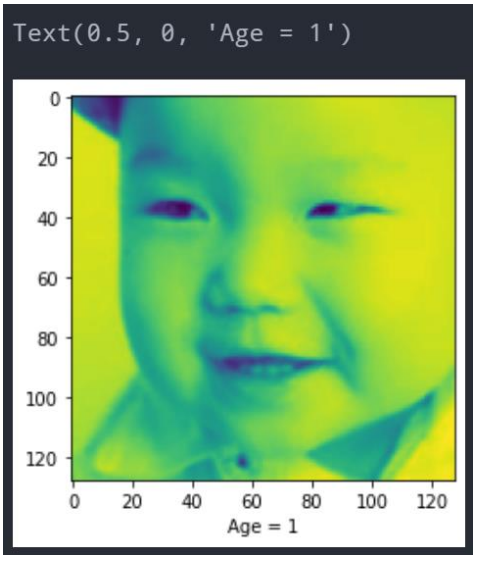


Some more results:

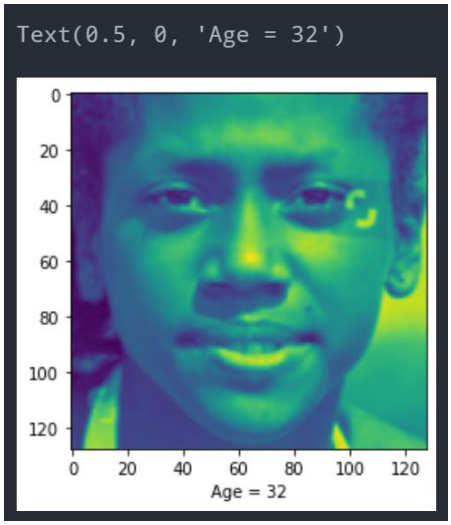




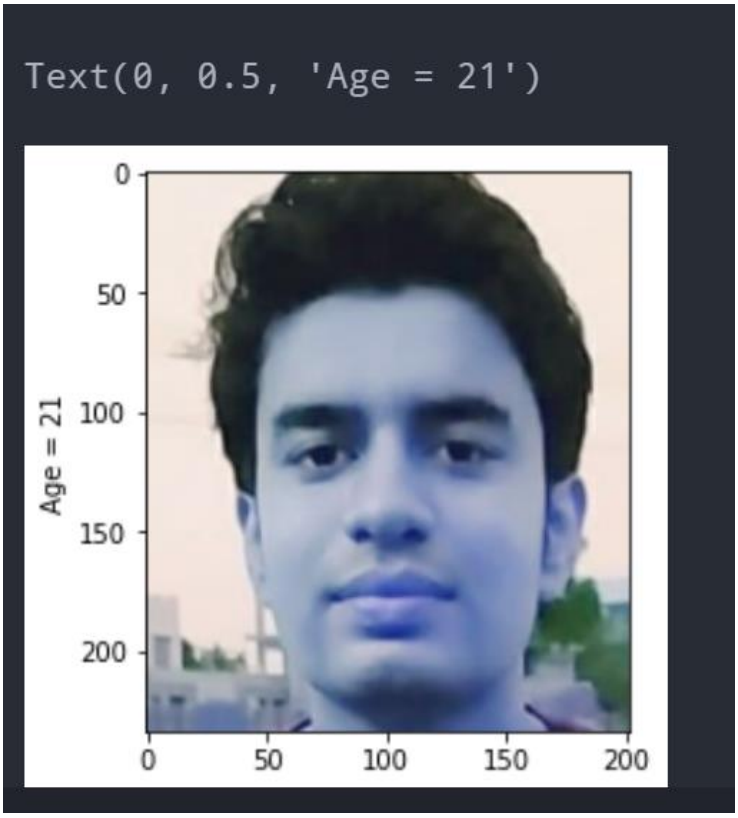




It gets it wrong sometimes as well:

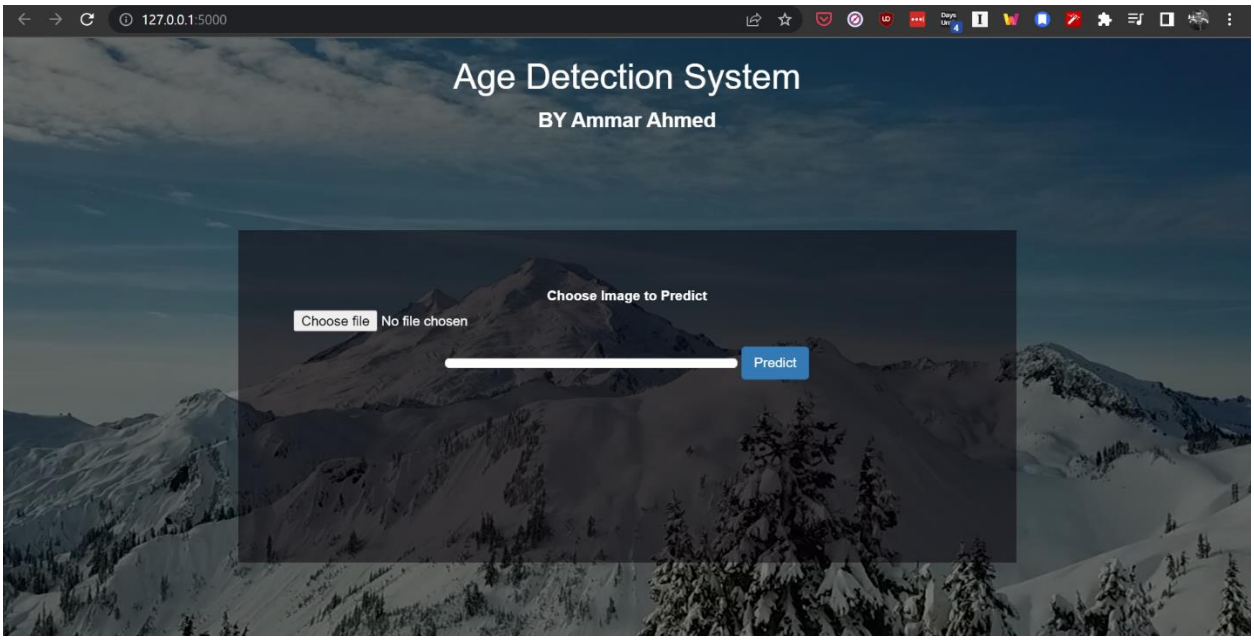


And finally, (because I had to) see what age the model predicts of my picture. The image was given after cropping:

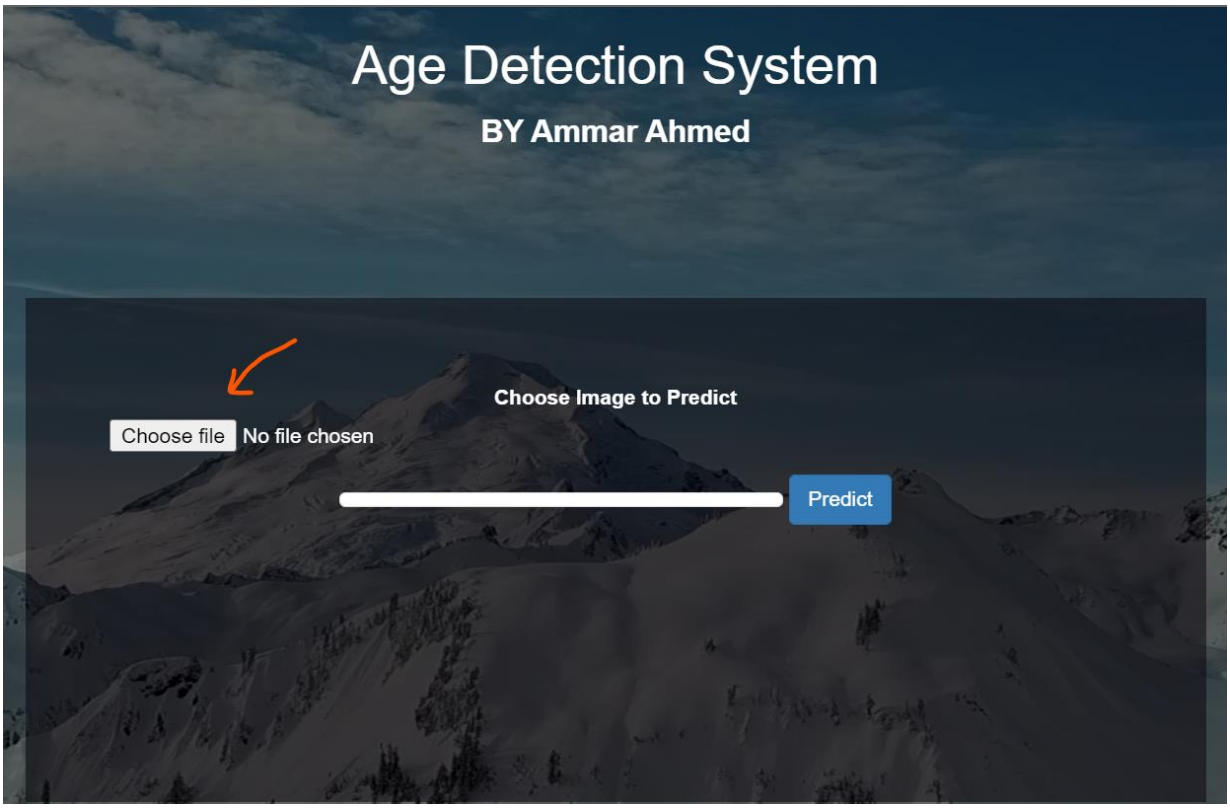


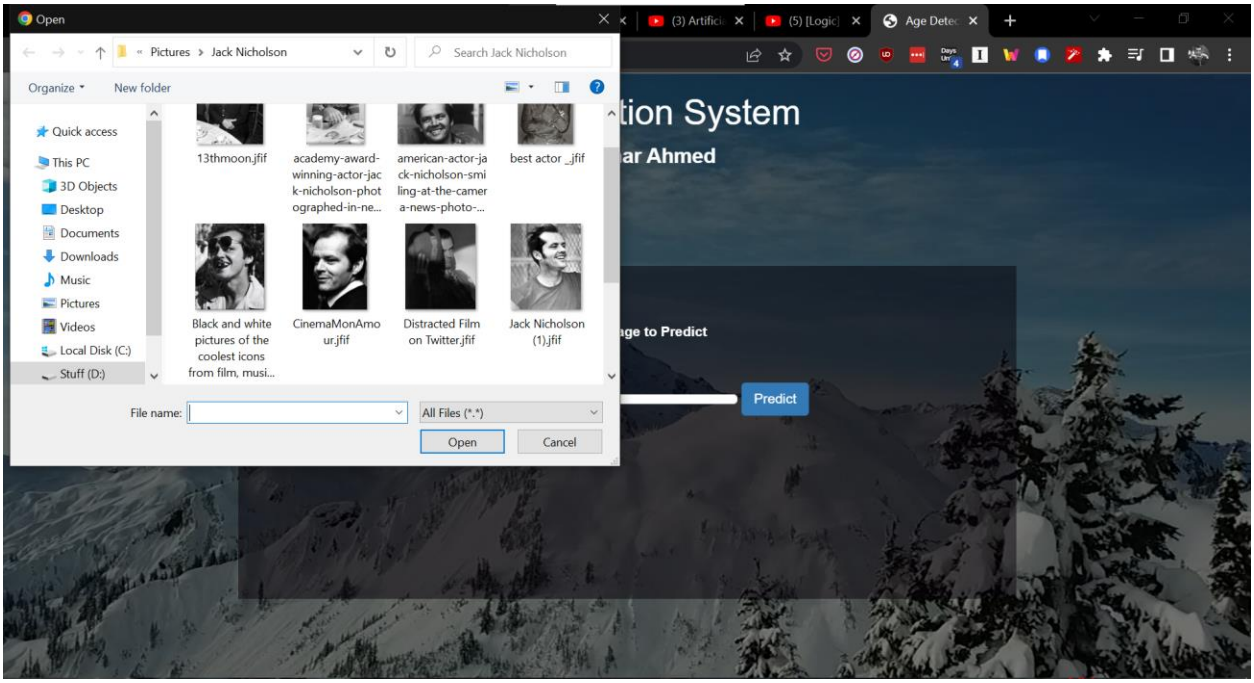
Deployment:

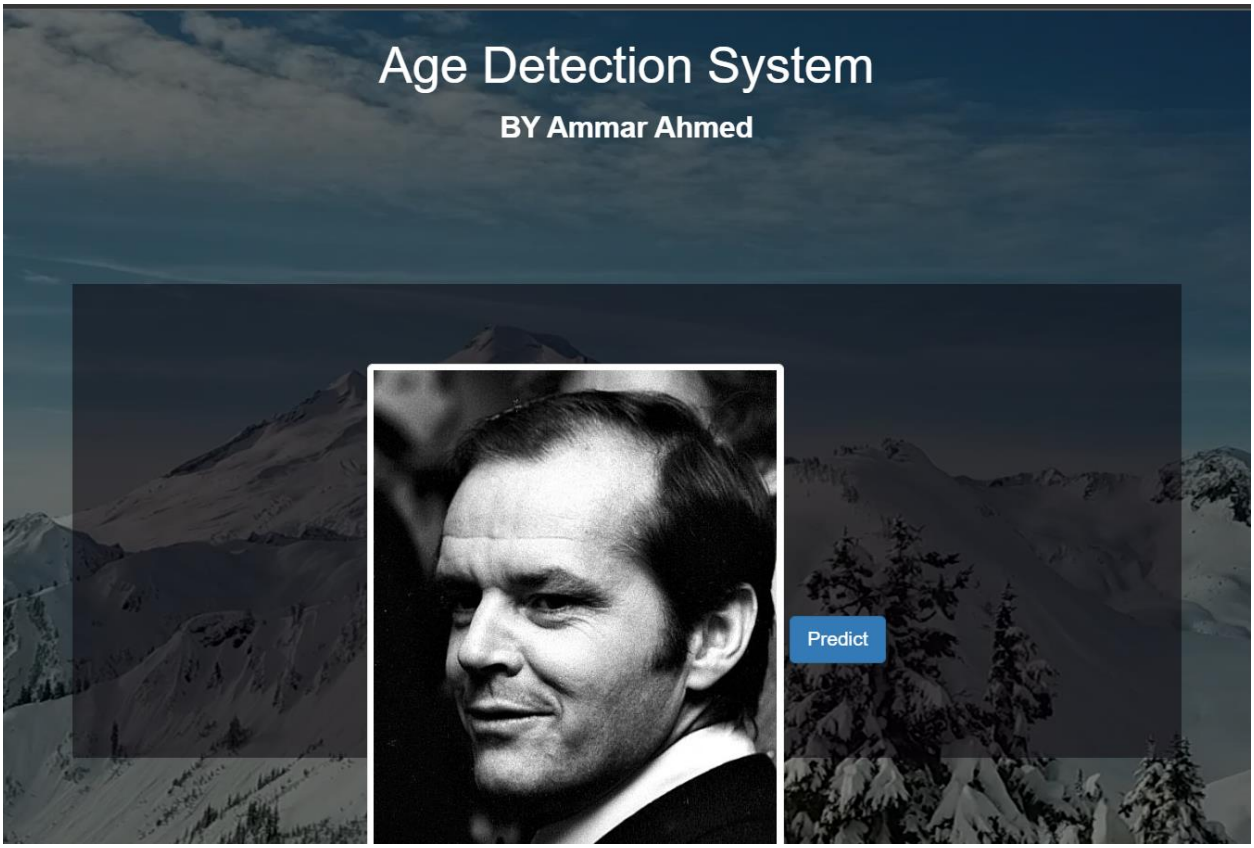
After the model has been trained, we need to save the model weights in order to deploy the model. For the deployment of the model, I used Flask. First, I created an interface using HTML, CSS, and Bootstrap. Second, I attached my html file to the flask framework. I loaded my model (that I saved along with its weights) in the flask file. Following is the initial interface along with the working of the deployed model.



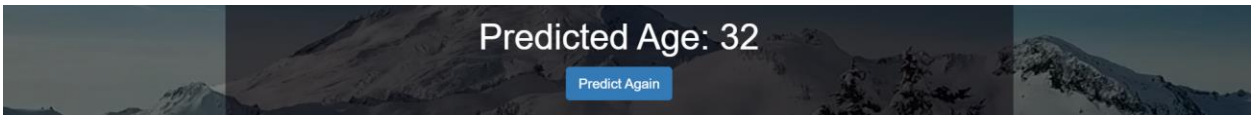
We can select a picture by clicking on the 'choose file' button.




Open
Pictures › Jack Nicholson
Search Jack Nicholson
Organize
New folder
Quick access
This PC
3D Objects
Desktop
Documents
Downloads
Music
Pictures
Videos
Local Disk (C:)
Stuff (D:)
13thmoon.jfif
academy-award-winning-actor-jack-nicholson-photographed-in-ne...
american-actor-jack-nicholson-smiling-at-the-camera-news-photo-...
best actor _jfif
Black and white pictures of the coolest icons from film, musi...
CinemaMonAmour.jfif
Distracted Film on Twitter.jfif
Jack Nicholson (1).jfif
File name:
All Files (*.*)
Open
Cancel
(3) Artifici
(5) [Logic]
Age Detec
tion System
ar Ahmed
ge to Predict
Predict


Age Detection System
BY Ammar Ahmed
Predict

After prediction:



You can try out another image, by clicking on the predict again, which will take you back to the homepage.

END