```
#importing libraries
from keras.models import Model
from keras.models import Sequential
from keras.layers import Dense,Flatten , Lambda , Input



#import Vgg16 model from keras
from keras.applications.vgg16 import VGG16
from keras.applications.vgg16 import preprocess_input

#for preprocessing
from keras.preprocessing import image
from keras.preprocessing.image import ImageDataGenerator



#load and preprocess image
from tensorflow.keras.utils import load_img
from tensorflow.keras.utils import img_to_array



#import other libraries
import numpy as np
import pandas as pd
from glob import glob
import matplotlib.pyplot as plt
```

```
import os
#all files and directories in the currentdirectory
files = os.listdir()

# Print the list
for file in files:
```

Automatic saving failed. This file was updated remotely or in another tab. Show diff

```
    .config
    drive
    sample_data
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
    Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
train_path =("/content/drive/MyDrive/2. Datasets/2. BasicFace Images/Final Training Images")
test_path=("/content/drive/MyDrive/2. Datasets/2. BasicFace Images/Final Testing Images")
```

```
model = VGG16(input_shape=(224,224, 3) ,weights = 'imagenet', include_top = False)
```

```
    Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop
    58889256/58889256 [==============================] - 0s 0us/step
```

```
for layer in model.layers:
  layer.trainable=False
```

```
Folder = glob("/content/drive/MyDrive/2. Datasets/2. BasicFace Images/Final Testing Images/*")
```

flatten the output of model

```
x = Flatten()(model.output)

y=len (Folder)


print(y)
```

```
16
```

```
predictions = Dense(y,activation= "softmax")(x)
```

Final vgg model using transfer learning

```
model = Model(inputs = model.input,outputs= predictions)

model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0

 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792

 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928

 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0

 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856

 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584

 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0

 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168

 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080
```



```
 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0

 block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160

 block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0

 block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0

 flatten (Flatten)           (None, 25088)             0

 dense (Dense)               (None, 16)                401424

=================================================================
Total params: 15,116,112
Trainable params: 401,424
Non-trainable params: 14,714,688
_________________________________________________________________
```

⏰ # **Important NOTES**

- image size decrease ,channel increase

  padding - TO attain the size of image

  pooling - to reduce the size of image ,no parrameter to learn in pooling

```
# Print the trainable property of each layer
for layer in model.layers:
    print(layer.trainable)
```

```
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
True
True
```

```
model.compile(
        loss = 'categorical_crossentropy',
        optimizer = 'adam',
        metrics = ['accuracy']
        )
```

```
train_datagen = ImageDataGenerator (
    rescale = 1/255,
    horizontal_flip=True,
    vertical_flip=True,
)

test_datagen= ImageDataGenerator(rescale=1/255)
```



```
training_set= train_datagen.flow_from_directory("/content/drive/MyDrive/2. Datasets/2. BasicFace Images/Final
                                                class_mode="categorical")
```

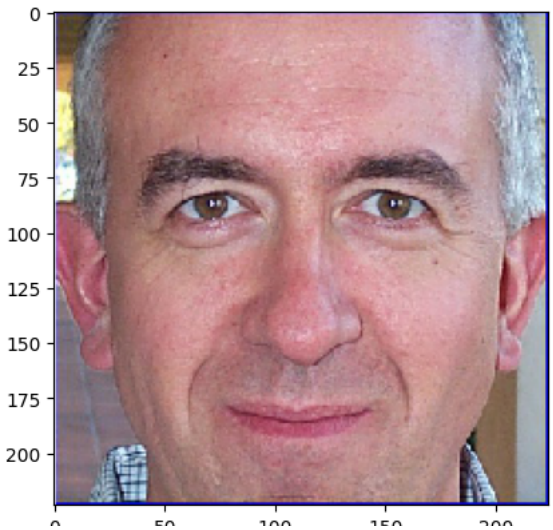
```
Found 244 images belonging to 16 classes.
```

```
testing_set= test_datagen.flow_from_directory("/content/drive/MyDrive/2. Datasets/2. BasicFace Images/Final Te
                                              class_mode="categorical")
```

```
Found 64 images belonging to 16 classes.
```

```
#load inage from a directory
image = load_img("/content/drive/MyDrive/2. Datasets/2. BasicFace Images/Final Testing Images/face14/1face14.j

plt.imshow(image)
```

```
<matplotlib.image.AxesImage at 0x7cee34da3640>
```



```
image2= load_img("/content/drive/MyDrive/2. Datasets/2. BasicFace Images/Final Testing Images/face12/2face12.j

plt.imshow(image2)
```

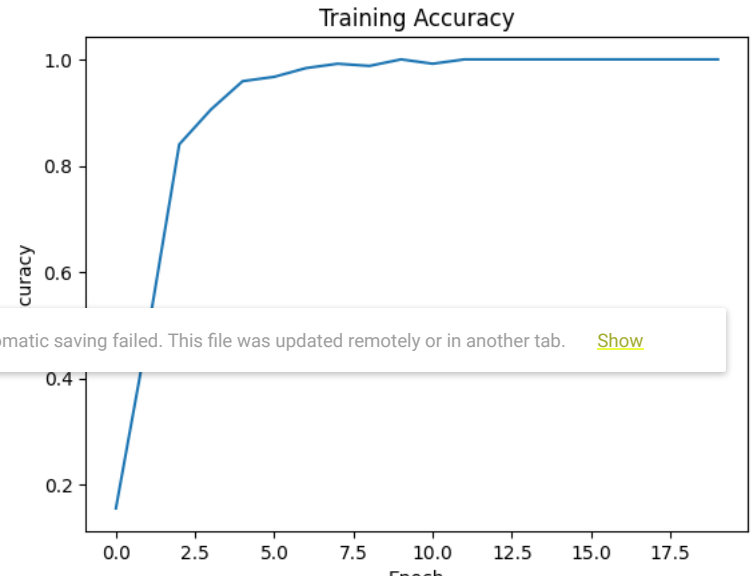
```
<matplotlib.image.AxesImage at 0x7cee35df6ce0>
```


```
final_model=model.fit(
    training_set,
    validation_data=testing_set,
    epochs=20,
    steps_per_epoch=len(training_set),
    validation_steps=len(testing_set),
)
```

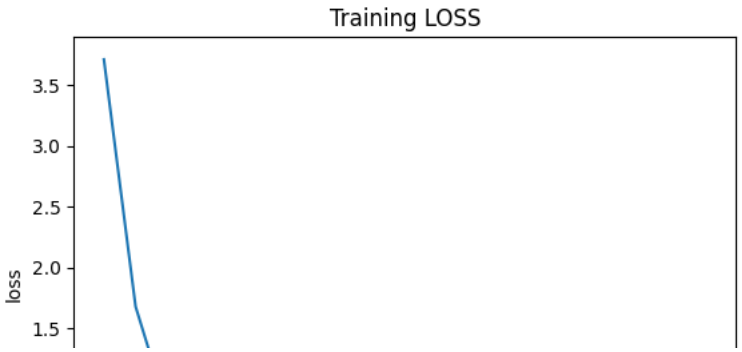
```
Epoch 1/20
8/8 [==============================] - 6s 563ms/step - loss: 3.7115 - accuracy: 0.1557 - val_loss: 1.9125 - val_accuracy: 0.3594
Epoch 2/20
8/8 [==============================] - 4s 546ms/step - loss: 1.6788 - accuracy: 0.4836 - val_loss: 1.0531 - val_accuracy: 0.7969
Epoch 3/20
8/8 [==============================] - 6s 665ms/step - loss: 0.8308 - accuracy: 0.8402 - val_loss: 0.5199 - val_accuracy: 0.8750
Epoch 4/20
8/8 [==============================] - 5s 573ms/step - loss: 0.4439 - accuracy: 0.9057 - val_loss: 0.3260 - val_accuracy: 0.9219
Epoch 5/20
8/8 [==============================] - 5s 627ms/step - loss: 0.2700 - accuracy: 0.9590 - val_loss: 0.1294 - val_accuracy: 0.9688
Epoch 6/20
8/8 [==============================] - 4s 570ms/step - loss: 0.1690 - accuracy: 0.9672 - val_loss: 0.1365 - val_accuracy: 0.9688
Epoch 7/20
8/8 [==============================] - 5s 590ms/step - loss: 0.1140 - accuracy: 0.9836 - val_loss: 0.1235 - val_accuracy: 0.9844
Epoch 8/20
```

```
8/8 [==============================] - 5s 573ms/step - loss: 0.0713 - accuracy: 0.9918 - val_loss: 0.0702 - val_accuracy: 0.9844
Epoch 9/20
8/8 [==============================] - 4s 534ms/step - loss: 0.0574 - accuracy: 0.9877 - val_loss: 0.0919 - val_accuracy: 0.9844
Epoch 10/20
8/8 [==============================] - 5s 679ms/step - loss: 0.0517 - accuracy: 1.0000 - val_loss: 0.0758 - val_accuracy: 0.9844
Epoch 11/20
8/8 [==============================] - 4s 562ms/step - loss: 0.0446 - accuracy: 0.9918 - val_loss: 0.0492 - val_accuracy: 0.9844
Epoch 12/20
8/8 [==============================] - 6s 723ms/step - loss: 0.0389 - accuracy: 1.0000 - val_loss: 0.0667 - val_accuracy: 1.0000
Epoch 13/20
8/8 [==============================] - 4s 530ms/step - loss: 0.0347 - accuracy: 1.0000 - val_loss: 0.0756 - val_accuracy: 0.9688
Epoch 14/20
8/8 [==============================] - 6s 765ms/step - loss: 0.0326 - accuracy: 1.0000 - val_loss: 0.0515 - val_accuracy: 0.9688
Epoch 15/20
8/8 [==============================] - 4s 533ms/step - loss: 0.0236 - accuracy: 1.0000 - val_loss: 0.0400 - val_accuracy: 1.0000
Epoch 16/20
8/8 [==============================] - 5s 568ms/step - loss: 0.0257 - accuracy: 1.0000 - val_loss: 0.0440 - val_accuracy: 0.9688
Epoch 17/20
8/8 [==============================] - 6s 702ms/step - loss: 0.0251 - accuracy: 1.0000 - val_loss: 0.0444 - val_accuracy: 0.9688
Epoch 18/20
8/8 [==============================] - 5s 559ms/step - loss: 0.0251 - accuracy: 1.0000 - val_loss: 0.0371 - val_accuracy: 1.0000
Epoch 19/20
8/8 [==============================] - 5s 591ms/step - loss: 0.0191 - accuracy: 1.0000 - val_loss: 0.0355 - val_accuracy: 1.0000
Epoch 20/20
8/8 [==============================] - 5s 644ms/step - loss: 0.0210 - accuracy: 1.0000 - val_loss: 0.0399 - val_accuracy: 0.9844
```

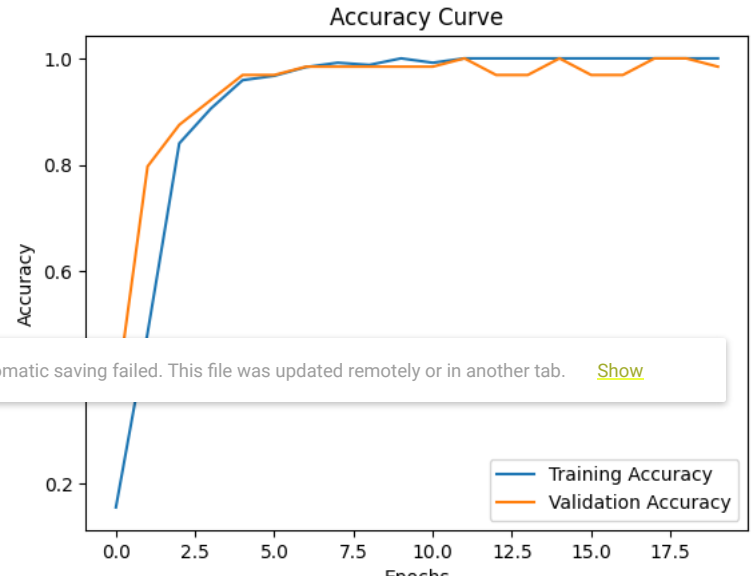
```
# Plotting the accuracy curve
plt.plot(final_model.history['accuracy'])
plt.title('Training Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.show()
```



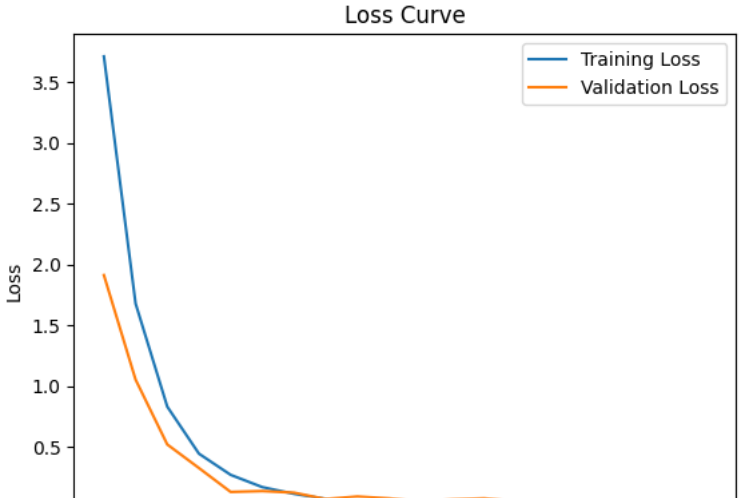

```
# Plotting the losss curve
plt.plot(final_model.history['loss'])
plt.title('Training LOSS')
plt.xlabel('Epoch')
plt.ylabel('loss')
plt.show()
```




```
# Get accuracy values
accuracy = final_model.history['accuracy']
val_accuracy = final_model.history['val_accuracy']

# Plot accuracy curve
plt.plot(accuracy, label='Training Accuracy')
plt.plot(val_accuracy, label='Validation Accuracy')
plt.title('Accuracy Curve')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```




```
# Get loss values
loss = final_model.history['loss']
val_loss = final_model.history['val_loss']
# Plot loss curve
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.title('Loss Curve')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```




```
# Save the model
model.save("/content/drive/MyDrive/1. Colab Notebooks/5. June July  2023/2. Internship/MODEL 1 - Basic face da
```

```
from tensorflow.keras.models import load_model
model1=load_model("/content/drive/MyDrive/1. Colab Notebooks/5. June July  2023/2. Internship/MODEL 1 - Basic
```

```
model1.summary()
```

```
Model: "model_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_4 (InputLayer)        [(None, 224, 224, 3)]     0

 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792

 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928

 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0

 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856
```

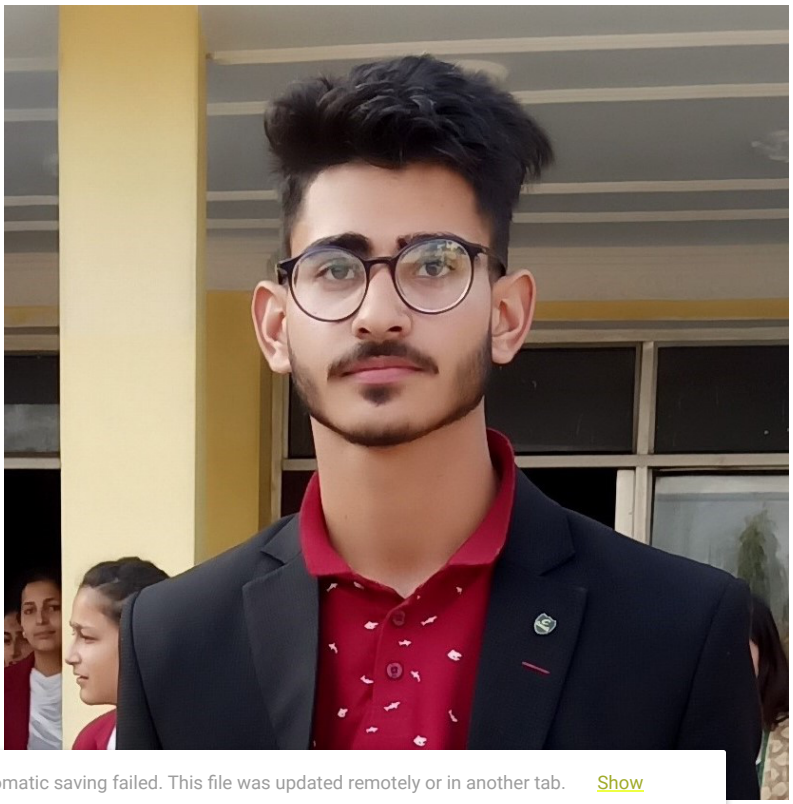


```
 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168

 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080

 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080

 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0

 block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160

 block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0

 block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0

 flatten_3 (Flatten)         (None, 25088)             0

 dense_3 (Dense)             (None, 16)                401424

=================================================================
Total params: 15,116,112
Trainable params: 401,424
```

```
Non-trainable params: 14,714,688
_________________________________________________________________
```

Test1 - Unknown image

```
from PIL import Image

# Load the image
image = Image.open("/content/drive/MyDrive/1. Colab Notebooks/5. June July  2023/2. Internship/MODEL 1 - Basic
image.show()
```




```
import numpy as np


import tensorflow as tf

#Resize the image
target_size = (224, 224)
image = image.resize(target_size)


# Convert the image to a NumPy array
image_array = tf.keras.preprocessing.image.img_to_array(image)
```

```
# Normalize
normalized_image_array = image_array / 255.0

# Expand the dimensions to create a batch of size 1
#This is commonly done when you want to pass a single image as input to a model that expects a batch of images
input_image = tf.expand_dims(normalized_image_array, axis=0)


print(np.shape(input_image))
print(np.shape(image))
```

```
(1, 224, 224, 3)
(224, 224, 3)
```

```
print( input_image )
```

```
tf.Tensor(
[[[[0.6117647  0.59607846 0.56078434]
   [0.60784316 0.5921569  0.5568628 ]
   [0.60784316 0.5921569  0.5568628 ]
   ...
   [0.5411765  0.5294118  0.49803922]
   [0.5411765  0.5294118  0.5019608 ]
   [0.54509807 0.53333336 0.5058824 ]]

  [[0.60784316 0.5921569  0.5568628 ]
   [0.6039216  0.5882353  0.5529412 ]
   [0.6039216  0.5882353  0.5529412 ]
   ...
   [0.5372549  0.5254902  0.49411765]
   [0.5372549  0.5254902  0.49803922]
   [0.5372549  0.5254902  0.49803922]]

  [[0.6        0.58431375 0.54901963]
   [0.59607846 0.5803922  0.54509807]
   [0.6        0.58431375 0.54901963]
   ...
   [0.53333336 0.52156866 0.49019608]
   [0.53333336 0.52156866 0.49411765]
   [0.53333336 0.52156866 0.49411765]]

  ...

  [[0.35686275 0.02745098 0.10196079]
   [0.36078432 0.03137255 0.10588235]
   [0.36862746 0.03137255 0.10196079]
   ...
   [0.74509805 0.67058825 0.58431375]
   [0.75686276 0.68235296 0.59607846]
   [0.67058825 0.6        0.5058824 ]]

  [[0.3647059  0.03137255 0.10588235]
   [0.37254903 0.03137255 0.10588235]
   [0.36862746 0.03137255 0.10196079]
   ...
   [0.7137255  0.627451   0.5411765 ]
   [0.62352943 0.53333336 0.4509804 ]
   [0.6039216  0.5137255  0.42745098]]

  [[0.36862746 0.03921569 0.10196079]
   [0.36862746 0.03921569 0.10196079]
   [0.3647059  0.03529412 0.10196079]
   ...
   [0.5254902  0.41568628 0.33333334]
   [0.54509807 0.4392157  0.35686275]
   [0.6745098  0.5686275  0.4862745 ]]]], shape=(1, 224, 224, 3), dtype=float32)
```

```
# Assuming you have loaded and preprocessed the image as 'input_image' and have the loaded model as 'model'

# Perform prediction
predictions = model1.predict(input_image)
```

```
1/1 [==============================] - 7s 7s/step
```

```
class_labels = ['face1', 'face10', 'face11', 'face12', 'face13', 'face14', 'face15', 'face16', 'face2', 'face3


# Get the predicted class index with the highest probability
predicted_class_index = np.argmax(predictions[0])

# Get the corresponding class label
class_labels = ['face1', 'face10', 'face11', 'face12', 'face13', 'face14', 'face15', 'face16', 'face2', 'face3
predicted_class_label = class_labels[predicted_class_index]

# Get the predicted probability for the predicted class
predicted_probability = predictions[0][predicted_class_index]


print("Predicted class: ", predicted_class_label)
print("Probability of detection: ", predicted_probability)
```

```
Predicted class:  face9
Probability of detection:  5.7554257e-06
```

Predicted class: face9

Probability of detection: 5.7554257e-06

Test 3 - Known image belong to face 10 class

```
image2 = Image.open("/content/drive/MyDrive/2. Datasets/2. BasicFace Images/Final Testing Images/face10/1face1

image2.show()
```


```
print(np.shape(image2))
```

```
(350, 350, 3)
```

```
image2=image2.resize(target_size)

image_array2 = tf.keras.preprocessing.image.img_to_array(image2)

normalized_image_array2 = image_array2 / 255.0


print(np.shape(normalized_image_array2))
```

```
(224, 224, 3)
```

```
input_image2 = tf.expand_dims(normalized_image_array2, axis=0)
```

```
print(np.shape(input_image2))
```

```
(1, 224, 224, 3)
```

```
predictions2 = model1.predict(input_image2)
```

```
1/1 [==============================] - 0s 32ms/step
```

```
print(predictions2)
```

```
[[1.2877179e-04 9.9778157e-01 1.6351299e-04 3.0071496e-05 8.1629853e-04
  2.3588078e-04 1.4722506e-06 2.5138859e-05 5.7554257e-06 2.3165862e-04
  7.5533804e-05 9.4630568e-06 1.0192848e-04 3.6026361e-06 2.1205402e-04
  1.7711153e-04]]
```

```
# Get the predicted class index with the highest probability
predicted_class_index2 = np.argmax(predictions2[0])

predicted_class_label2 = class_labels1[predicted_class_index2]

# Get the predicted probability for the predicted class
predicted_probability2 = predictions1[0][predicted_class_index2]

print("Predicted class: ", predicted_class_label2)
print("Probability of detection: ", predicted_probability2)
```

```
Predicted class:  face10
Probability of detection:  0.003421458
```

Compare matching probability of this ime with all other classes


```
# Iterate over each class label in the training set
for i, class_label in enumerate(class_labels1):
    # Get the predicted probability for the current class
    predicted_probability = predictions2[0][i]

    # Check if the predicted class matches the true class label
    is_matching = (predicted_class_index2 == i)

    # Append the class label and matching probability to the list
    matching_probabilities.append((class_label, predicted_probability))

# Create a list to store the matching probabilities and faces for each class
matching_results = []

# Print the class labels and their corresponding matching probabilities
for class_label, probability in matching_probabilities:
    print(f"Class: {class_label}, Matching Probability: {probability}")
```

```
Class: face1, Matching Probability: 0.00012877178960479796
Class: face10, Matching Probability: 0.9977815747261047
Class: face11, Matching Probability: 0.0001635129883652553
Class: face12, Matching Probability: 3.007149643963203e-05
Class: face13, Matching Probability: 0.0008162985322996974
Class: face14, Matching Probability: 0.00023588078329339623
Class: face15, Matching Probability: 1.4722505738973268e-06
```

```
Class: face16, Matching Probability: 2.5138859200524166e-05
Class: face2, Matching Probability: 5.755425718234619e-06
Class: face3, Matching Probability: 0.00023165861784946173
Class: face4, Matching Probability: 7.553380419267341e-05
Class: face5, Matching Probability: 9.463056812819559e-06
Class: face6, Matching Probability: 0.00010192848276346922
Class: face7, Matching Probability: 3.6026360703544924e-06
Class: face8, Matching Probability: 0.0002120540157193318
Class: face9, Matching Probability: 0.0001771115348674357
```

```
# Find the class label with the maximum matching probability
max_matching_probability = max(matching_probabilities, key=lambda x: x[1])

# Print the class label and its maximum matching probability
print(f"Class: {max_matching_probability[0]}, Maximum Matching Probability: {max_matching_probability[1]}")
```

```
Class: face10, Maximum Matching Probability: 0.9977815747261047
```

CODE ENDS HERE.

Automatic saving failed. This file was updated remotely or in another tab. Show diff