

# Face Recognition Project Report

Amir Arsalan Sanati

## • Introduction

This report presents a face recognition project implemented in Python. The project utilizes preprocessing and data augmentation techniques, as well as neural networks, to construct a 16-class face recognition model with high performance.

The model distinguishes faces of 16 different known Persian celebrities. The celebrities are as follows:

```
class_names = {  
    0: 'Ali Davi',  
    1: 'Mohsen Chavoshi',  
    2: 'Mohamad Esfahani',  
    3: 'Taraneh Alidostnia',  
    4: 'Bahram Radan',  
    5: 'Sogol Khaligh',  
    6: 'Homayoon Shajarian',  
    7: 'Sahar Dolatshahi',  
    8: 'Mehran Ghafourian',  
    9: 'Mehran Modiri',  
    10: 'Reza Attaran',  
    11: 'Javad Razavian',  
    12: 'Seyed Jalal Hoseini',  
    13: 'Alireza Beyranvand',  
    14: 'Nazanin Bayati',  
    15: 'Bahareh Kianafshar',  
}
```

Figure 1 Classes

In this project, three files perform the main functions, which I will discuss briefly and then examine each in detail.

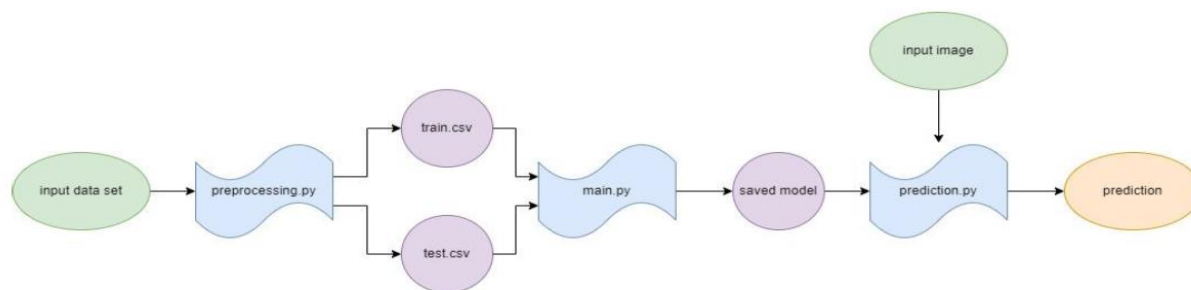


Figure 2 The architecture and workflow of the project.

**Preprocessing.py**, This file contains data augmentation functions and algorithms that augment the dataset photos and generate train.csv and test.csv files from raw input photos.

**Main.py**, In this file, the codes related to the architecture of the neural network, its fitting, and the evaluation and storage of the model have been implemented. This file uses the contents of train.csv and test.csv.

**Prediction.py**, In this file, the frontend of the project, which is a Windows desktop application, is implemented. It takes a photo from the user, loads the models saved by main.py, predicts the corresponding photo and finally shows the prediction, which includes the probability of the photo belonging to each of the 16 classes in the dataset.

- **Preprocessing by hand**

The first step of preprocessing which I had to go through myself, was to crop each image of the raw input data set, so that each image would contain only and only the face of the target person, not any other object.

- **Preprocessing.py**

The first function that I will deal with is the **read\_images** function, this function counts the number of photos belonging to each class and gives us a general statistic about the photos, like the size of the photos, their number, etc.

This function is used after augmenting the raw input photos to check if the photos of each class are according to the desired format (jpg)? How many photos does each class have? Generally, how many photos does the train data contain?

```

def read_images(search_directory):
    if search_directory.startswith('\\'):
        search_directory = search_directory[0:-1]
    count = 0
    min_size = (1e10, 1e10)
    min_size_addr = None
    # List of common image file extensions
    valid_image_extensions = ['.jpg']
    invalid_image_extensions = ['.jpeg', '.png', '.gif', '.bmp']

    # Iterate over all files and directories in the given directory
    for root, dirs, files in os.walk(search_directory):
        try:
            file_dir = int(root.replace(f'{search_directory}\\', ''))
            image_per_class[file_dir] = 0
            # Check if the file has an image extension
        except:
            continue

        for file in files:
            if any(file.lower().endswith(ext) for ext in invalid_image_extensions):
                raise Exception(f'image {root}\\{file} has an invalid extension.')
            if any(file.lower().endswith(ext) for ext in valid_image_extensions):
                im = Image.open(f'{root}\\{file}')
                if min(im.size) < min(min_size):
                    min_size = im.size
                    min_size_addr = f'{root}\\{file}'
                del im
                count += 1
                image_per_class[file_dir] += 1

    print('total count: ', count)
    print('image per class count: ')
    for cls in image_per_class.keys():
        print(f'{class_names[cls]} : {image_per_class[cls]}')

    print('min size: ', min_size)
    print('min size addr', min_size_addr)

```

Figure 3 read\_images

The next function is **resize\_and\_save**, which resizes photos located in a specific location to a desired size (here 200 x 200) and saves them in a desired location.

The purpose of implementing this function is to add the raw input images (raw images which are not augmented) to the training data. That is, in addition to seeing the augmented photos, the model also needs to see the raw input photos before augmentation. In this function, it is checked if a photo's mode is not RGB, it is converted into RGB format because our neural network architecture only accepts RGB photos.

```

2 usages Amir Arsalan Sanati
def resize_and_save(input_directory, output_directory: str):
    if output_directory.endswith('\\'):
        output_directory = output_directory[0:-1]
    image_extensions = ['.jpg', '.jpeg', '.png', '.gif', '.bmp']

    # Iterate over all files and directories in the given directory
    for root, dirs, files in os.walk(input_directory):

        for file in files:
            if any(file.lower().endswith(ext) for ext in image_extensions):
                im = Image.open(f'{root}\\{file}')
                if im.mode != 'RGB':
                    im = im.convert("RGB")
                im = im.resize((200, 200))
                file_name = f'{output_directory}\\_{file}_resized.jpg'
                im.save(file_name)
                print(f"Saved {file_name}")

    print("Image resizing and saving complete.")

```

*Figure 4 resize\_and\_save*

The next function to discuss would be **create\_csv**. This function creates a csv file from all train photos (augmented and resized), each row of this file shows one of the train photos that has been flattened, along with a 16 valued vector (we have 16 classes). For every photo belonging to a specific class, only the value corresponding to the index of that class in the vector is one, the other values would be zeros.

Because the number of photos of each class may be large (for example, 2 thousand) and all photos of all classes may not fit together in the RAM space, the data for each class is brought in to the RAM, added to the corresponding file, and then will be deleted from the RAM (the previous implementation was such that, the whole data was brought to RAM and then this data was saved in the csv file).

```

Amir Arsalan Sanati
def create_csv(search_directory, output_csv):
    image_extensions = ['.jpg']
    rows = 0
    columns = 0
    for root, dirs, files in os.walk(search_directory):
        classes = np.zeros(16)
        try:
            file_dir = int(root.replace(search_directory, ''))
            classes[file_dir] = 1
        except:
            continue
        if file_dir == 9:
            continue
        data_csv = []
        print(f'class {file_dir}')
        for file in files:
            if any(file.lower().endswith(ext) for ext in image_extensions):
                im = np.array(Image.open(F'{root}\\{file}')).flatten()
                if im.size != 120_000:
                    raise Exception(f'image {root}\\{file} doesn't have the right shape')
                # Convert the NumPy array back to a Pillow image
                data_csv.append([*im, *classes])
                del im
        with open(output_csv, 'a', newline='') as csvfile:
            # Create a CSV writer object
            csv_writer = csv.writer(csvfile)

            # Write the data to the CSV file
            csv_writer.writerows(data_csv)

        rows += len(data_csv)
        columns = len(data_csv[0])
        del data_csv

    print(f"CSV file '{output_csv}' created successfully. it has {rows} rows and {columns} columns.")

```

Figure 5 create\_csv

The next function is the geometry\_augment function, this function gets the path to the input images (raw photos + photos that are color augmented) and a path to a desired location in order to save the augmented images. It uses the ImageDataGenerator of the keras library, which is a pipeline mechanism for augmenting photos, and the flow\_from\_directory method of this object to generate 300 new photos from each photo (in this case).

As you can see, this method, augments the data geometrically. I have also implemented a method for color augmentation of data which you will see in short.

Also note that at the end of this function, resize\_and\_save is called.

```

Amir Arsalan Sanati
def geometry_augment():
    datagen = ImageDataGenerator(
        rotation_range=30,
        width_shift_range=0.2,
        height_shift_range=0.2,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True,
        fill_mode='constant',
        cval=0
    )

    for j in range(16):
        if j == 9:
            continue
        directory = f'color_augmented_images\\{j}'
        save_to_dir = f'train_images\\{j}'
        i = 0
        for batch in datagen.flow_from_directory(directory=directory, target_size=(200, 200), batch_size=1,
                                                save_to_dir=save_to_dir, save_format='jpg', color_mode='rgb'):
            i += 1
            if i == 300:
                break
        resize_and_save(directory + f'\\{j}', save_to_dir)

```

Figure 6 geometry\_augment

The next function is `augment_each_class`. This function is the same as the function above, only with the difference that each class can be augmented by any number (not necessarily 300).

```

Amir Arsalan Sanati
def augment_each_class(num, directory, save_to_dir):
    datagen = ImageDataGenerator(
        rotation_range=30,
        width_shift_range=0.2,
        height_shift_range=0.2,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True,
        fill_mode='constant',
        cval=0
    )

    i = 0
    for batch in datagen.flow_from_directory(directory=directory, target_size=(200, 200), batch_size=1,
                                            save_to_dir=save_to_dir, save_format='jpg', color_mode='rgb'):
        i += 1
        if i == num:
            break
    resize_and_save(directory, save_to_dir)

```

Figure 7 augment\_each\_class

The next implemented function is **color\_augment**. As its name suggests, this function generates new photos by changing the color characteristics of the image (the color of each pixel, creating noise in the image, exchanging the color channels of the image and...). The imgaug library is used for this method.

First, the photos of each class are read, given to imgaug, and then the augmented photos are stored in the same directory.

```
Amir Arsalan Sanati
def color_augment(source_path: str):
    if source_path.endswith('\\'):
        source_path = source_path[:-1]
    image_extensions = ['.jpg', '.jpeg', '.png', '.gif', '.bmp']
    ia.seed(random.randint(1, 100))
    seq = iaa.Sequential([
        # Small gaussian blur with random sigma between 0 and 0.5.
        # But we only blur about 50% of all images.
        iaa.Sometimes(
            0.7,
            iaa.GaussianBlur(sigma=(0, 0.5))
        ),
        # Strengthen or weaken the contrast in each image.
        iaa.LinearContrast((0.75, 1.5)),
        # Add gaussian noise.
        # For 50% of all images, we sample the noise once per pixel.
        # For the other 50% of all images, we sample the noise per pixel AND
        # channel. This can change the color (not only brightness) of the
        # pixels.
        iaa.AdditiveGaussianNoise(loc=0, scale=(0.0, 0.05 * 255), per_channel=0.5),
        # Make some images brighter and some darker.
        # In 20% of all cases, we sample the multiplier once per channel,
        # which can end up changing the color of the images.
        iaa.Multiply((0.6, 1.4), per_channel=1),
    ], random_order=True)

    for root, dirs, files in os.walk(source_path):
        images = []
        image_dir = False
        for file in files:
            if file.lower().endswith(ext) for ext in image_extensions):
                image_dir = True
                im_np = Image.open(f'{root}\\{file}').resize((200, 200))
                if im_np.mode != "RGB":
                    im_np = im_np.convert("RGB")
                images.append(im_np)

        if not image_dir:
            continue
        images = np.array(images)
        images_aug = seq(images=images)
        i = 0
        for image in images_aug:
            im = Image.fromarray(image)
            im.save(f'{root}\\{i}_augmented.jpg')
            i += 1

        print(f'saved augmented images for {root}')
```

Figure 8 color\_augment

The next function is **create\_test\_csv**. This function has the same functionality as the `create_csv` function, but for the data in the test folder.

```
Amir Arsalan Sanati
def create_test_csv(output_csv):
    image_extensions = ['.jpg', '.jpeg', '.png', '.gif', '.bmp']
    source_path = f'test_images\\'
    csv_data = []

    for root, dirs, files in os.walk(source_path):
        classes = np.zeros(16)
        try:
            file_dir = int(root.replace(source_path, ''))
            classes[file_dir] = 1
        except:
            continue

        for file in files:
            if any(file.lower().endswith(ext) for ext in image_extensions):
                im = Image.open(f'{root}\\{file}')
                if im.mode != 'RGB':
                    im = im.convert('RGB')
                im = im.resize((200, 200))
                im_np = np.array(im).flatten()
                csv_data.append([*im_np, *classes])

    with open(output_csv, 'w', newline='') as csv_file:
        # Create a CSV writer object
        csv_writer = csv.writer(csv_file)

        # Write the data to the CSV file
        csv_writer.writerows(csv_data)

    print(f'test file created with {len(csv_data)} rows and {len(csv_data[0])} columns.')
```

Figure 9 create\_test\_csv

The next functions that are addressed are **create\_csv\_for\_class** and **create\_test\_csv\_for\_class**. These functions do the same work as the functions described above, with the difference that the vector that was added to the end of each photo and determined the photo's belonging to a class (included 16 zeros and ones), here is just a number which will say whether a specific photo belongs to the intended class or not. In fact, these functions generate training and test data for each class separately. In the main.py section, we will discuss why to implement such a function.

In `create_csv_for_class` function, first, all the photos belonging to a certain class (specified in the function parameter) are added to `data_csv` with the label 1. After that an equal number of images from the rest of the classes will be randomly added with the label 0.

The `create_test_csv_for_class` function also collects and stores all the test photos of the specified class with the label 1, and the same number of test photos from the rest of the classes with the label 0.



Amir Arsalan Sanati

```
def create_csv_for_class(dest_path: str, root_path: str, class_num, class_count):
    if root_path.endswith('\\'):
        root_path = root_path[0:-1]

    if dest_path.endswith('\\'):
        dest_path = dest_path[0:-1]
    image_extensions = ['.jpg']
    source_path = f'{root_path}\\{class_num}'
    data_csv = []
    for root, dirs, files in os.walk(source_path):
        for file in files:
            if any(file.lower().endswith(ext) for ext in image_extensions):
                im = Image.open(f'{root}\\{file}')
                im = np.array(im).flatten()
                if im.size != 120_000:
                    raise Exception(f'image {root}\\{file} doesn't have the right shape')
                data_csv.append([*im, 1])

    each_class = class_count // 15
    for i in range(16):
        if i == class_num:
            continue
        source_path = f'{root_path}\\{i}'
        for root, dirs, files in os.walk(source_path):
            filtered_files = list(filter(lambda x: (x.lower().endswith(ext) for ext in image_extensions), files))
            sampled_files = random.sample(filtered_files, each_class)
            for sample in sampled_files:
                im = Image.open(f'{root}\\{sample}')
                im = np.array(im).flatten()
                if im.size != 120_000:
                    raise Exception(f'image {root}\\{sample} doesn't have the right shape')
                data_csv.append([*im, 0])

    random.shuffle(data_csv)
    with open(f'{dest_path}\\{class_num}_train.csv', 'w', newline='') as csvfile:
        csv_writer = csv.writer(csvfile)

        # Write the data to the CSV file
        csv_writer.writerows(data_csv)

    print(f'{dest_path}\\{class_num}_train.csv file created with {len(data_csv)} rows and {len(data_csv[0])} columns.')
```

Figure 10 create\_csv\_for\_class

```

1 usage Amir Arsalan Sanati
def create_test_csv_for_class(dest_path: str, class_num, test_path: str):
    if test_path.endswith('\\'):
        test_path = test_path[0:-1]

    if dest_path.endswith('\\'):
        dest_path = dest_path[0:-1]
    image_extensions = ['.jpg', '.jpeg', '.png', '.gif', '.bmp']
    source_path = f'{test_path}\\{class_num}'
    data_csv = []
    for root, dirs, files in os.walk(source_path):
        for file in files:
            if any(file.lower().endswith(ext) for ext in image_extensions):
                im = Image.open(f'{root}\\{file}')
                if im.mode != 'RGB':
                    im = im.convert('RGB')
                im = im.resize((200, 200))
                im_np = np.array(im).flatten()
                data_csv.append([*im_np, 1])

    remaining_test = len(data_csv)
    test_images_list = []

    for root, dirs, files in os.walk(test_path):
        try:
            file_dir = int(root.replace(f'{test_path}\\', ''))
        except:
            continue
        if file_dir == class_num:
            continue

        for file in files:
            if any(file.lower().endswith(ext) for ext in image_extensions):
                test_images_list.append(f'{root}\\{file}')

    samples = random.sample(test_images_list, remaining_test)
    for sample in samples:
        im = Image.open(sample)
        if im.mode != "RGB":
            im = im.convert("RGB")
        im = im.resize((200, 200))
        im = np.array(im).flatten()
        data_csv.append([*im, 0])

    random.shuffle(data_csv)
    with open(f'{dest_path}\\{class_num}_test.csv', 'w', newline='') as csvfile:
        csv_writer = csv.writer(csvfile)

        # Write the data to the CSV file
        csv_writer.writerows(data_csv)

    print(f'{dest_path}\\{class_num}_test.csv file created with {len(data_csv)} rows and {len(data_csv[0])} columns.')

```

Figure 11 create\_test\_csv\_for\_class

The last method which will be discussed is **train\_test\_split**. As its name suggests, this function will get a path to raw input data (not augmented) and splits them to train (raw train, still not augmented) and test sections (with respect to the test\_size parameter) and then saves the selected test images in a desired location.

```

1 usage  Amir Arsalan Sanati
def train_test_split(input_data_path, output_path: str, train_size):
    if output_path.endswith('\\'):
        output_path = output_path[0:-1]

    if input_data_path.endswith('\\'):
        input_data_path = input_data_path[0:-1]
    class_number = 0
    if not (0 <= train_size <= 1):
        raise Exception(f'train_size should be a float number, the given number was({train_size})')
    image_extensions = ['.jpg', '.jpeg', '.png', '.gif', '.bmp']
    for root, dirs, files in os.walk(input_data_path):
        try:
            if dirs:
                class_number = root.replace(f'{input_data_path}\\', '')
            except:
                continue
            if not files:
                continue
            filtered_list = list(filter(lambda x: (x.lower().endswith(ext) for ext in image_extensions), files))
            print(f'splitting class {class_number} with size {len(filtered_list)}')
            sample_size = int(len(filtered_list) * (1 - train_size))
            samples = random.sample(filtered_list, sample_size)
            for sample in samples:
                source_path = f'{root}\\{sample}'
                dest_path = f'{output_path}\\{class_number}\\{sample}'
                shutil.move(source_path, dest_path)

```

Figure 12 train\_test\_split

At the end, you can augment the raw input images in two ways. One way is to augment each class and generate a single train.csv and a single test.csv (with a 16 valued label for each entry of the files). The other way is to augment each class and generate train and test files separately (with a single 0 or 1 label). The second way is chosen to be used in this project and as I said, the reason will be explained in the main.py section.

```

train_test_split('D:\\new data set\\raw input', 'D:\\new data set\\test data', 0.8)
color_augment('D:\\new data set\\raw input')
input('proceed ? ')
for i in range(16):
    augment_each_class(2000, f'D:\\new data set\\raw input\\{i}', f'D:\\new data set\\train data\\{i}')
input('proceed ? ')
read_images('D:\\new data set\\train data')
for i in range(16):
    create_csv_for_class('D:\\new data set', 'D:\\new data set\\train data', i, 2000)
    create_test_csv_for_class('D:\\new data set', i, 'D:\\new data set\\test data')

```

Figure 13 usage

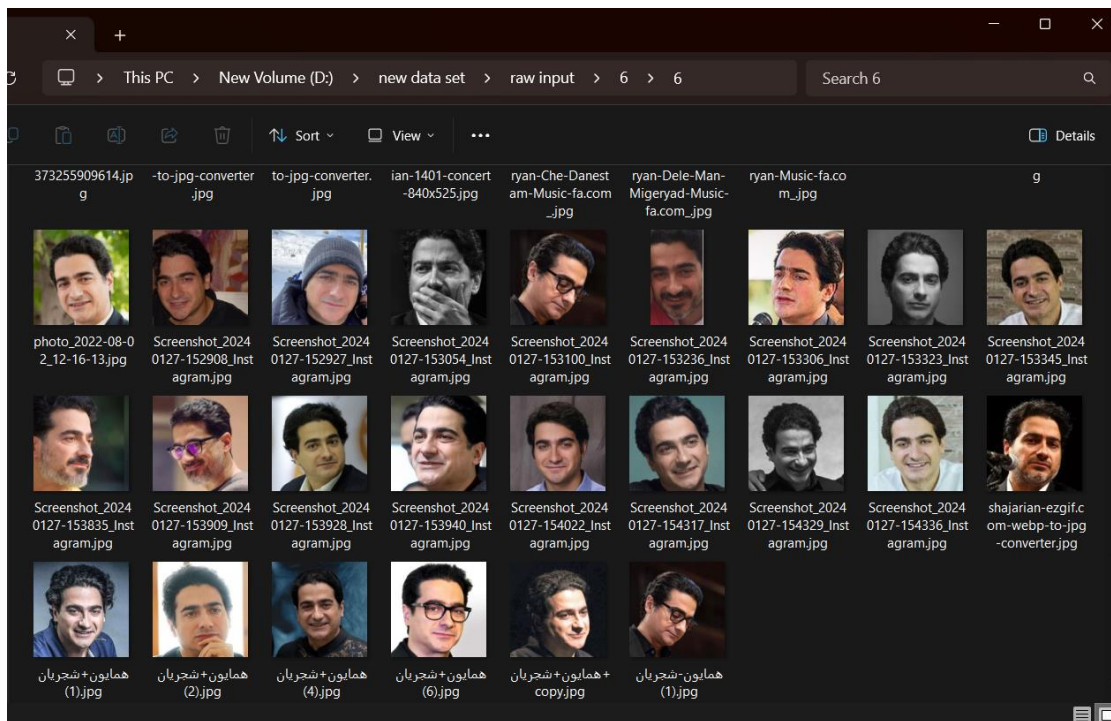


Figure 14 raw input images

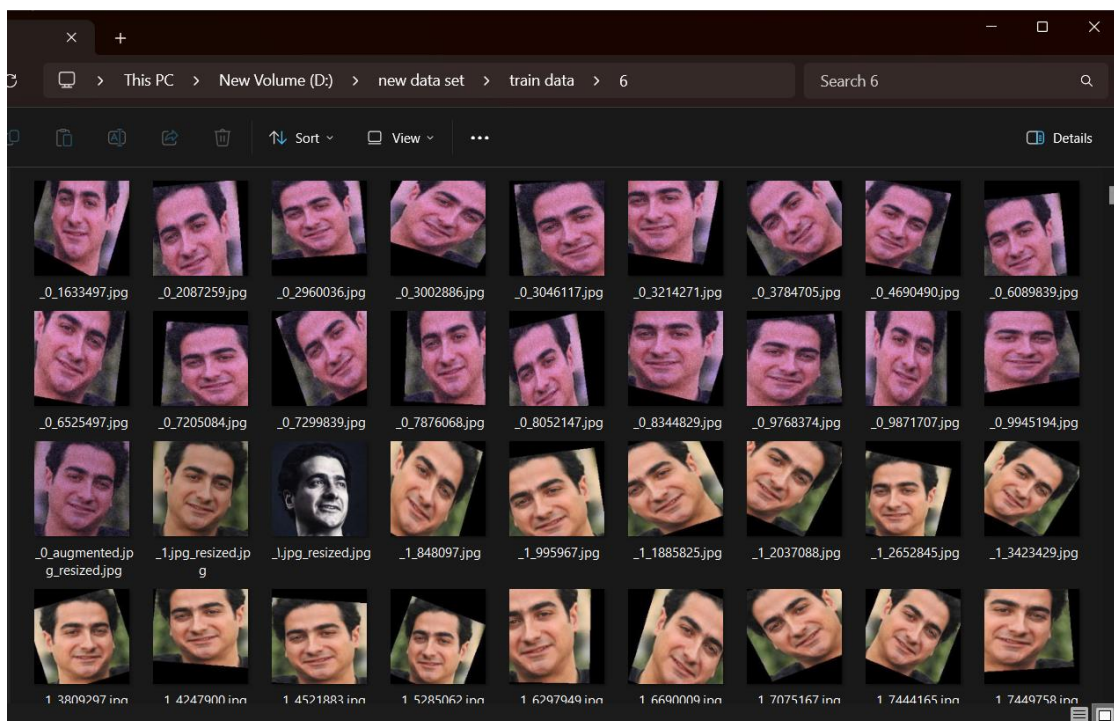


Figure 15 color augmented and geometry augmented photos

- **Main.py**

In this section, project's neural network is implemented, trained, and evaluated with the data prepared by the previous step.

- Architecture of the neural network

First, I will show you the structure of the network architecture, what it looks like and how it works. Then the code will be shown.

At first, the data set was very small. Such that, even with applying lots of data augmentation, the model still couldn't predict properly (of course a bigger data set was collected after that). The architecture of the CNN has changed so many times and I can't cover all the architectures I've tried, so I will only explain the first and the last architecture I came up with.

- **THE FIRST ONE:**

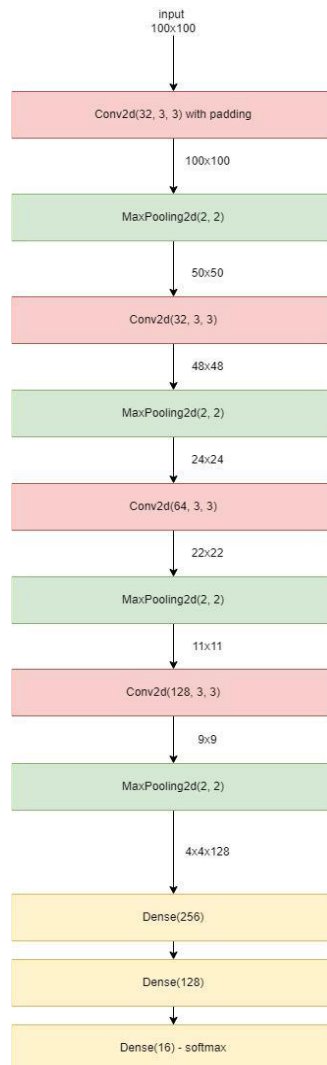


Figure 16 The first architecture

In this architecture, photos were stored with a size of 100 x 100, therefore it was possible to store a large number of photos (10 thousand photos) on RAM. But one of the problems with this model is its size. The size of 100 x 100 seems to be small because face recognition images have so many details and by reducing the size of the photos to 100 x 100, a significant part of these details may be lost, and the model will not be able to recognize correctly. One solution was to increase the size of the photos (for example, 200 by 200). However, the large number of photos could not be stored in the RAM anymore. Also, the graphic card exceeded its memory limit in some cases which lead to failure in training of model. Therefore, if we increase the size of the training photos, memory problems will arise, and if we reduce the number of the training photos, the training procedure will not result in a good model. Furthermore, facial recognition networks should have deeper structures so they can recognize more complex patterns from the given image, but again, due to the memory limit, it was not possible to increase the number of layers and have a relatively deep model. Finally with all these interpretations and by making all the possible changes that seemed helpful, the accuracy of the model on the test data was not suitable at all (20%) and I constantly encountered the problem of overfit and the situation did not get any better. Therefore, it was necessary to increase the size of the photos (for example, 200 x 200), the number of photos (for example, 2000 photos or even more for each class), and also to have a deeper model. But the memory limitation was very problematic. So, instead of training one model for 16 classes and trying to fit all the required data into the computer memory resources, why not having 16 separate models and train them separately on their own large data sets? This way it is possible to (first) increase the layers and have a relatively deep model for each class, (second) increase the training data for each class, (third) not facing memory problems. Because each class is trained separately in different times and no two classes are trained at the same time, so each class (when training) have access to all the memory it wants. That's why functions such as `create_csv_for_class`, `create_test_csv_for_class` and `augment_each_class` are implemented.

In the end the network architecture is as follows:

You may ask, why this structure?

Well, we need more layers, more layers bring more trainable parameters, more parameters require more memory from the graphic card, and my resources do not meet these requirements. Performing multiple convolution layers repeatedly allows me to apply a convolution layer with bigger kernel size, but, with less trainable parameters. For example, instead of applying one convolutional layer with kernel 7x7 (49 parameters), you can apply two consecutive conv layers with kernel size 3x3 (18 parameters).

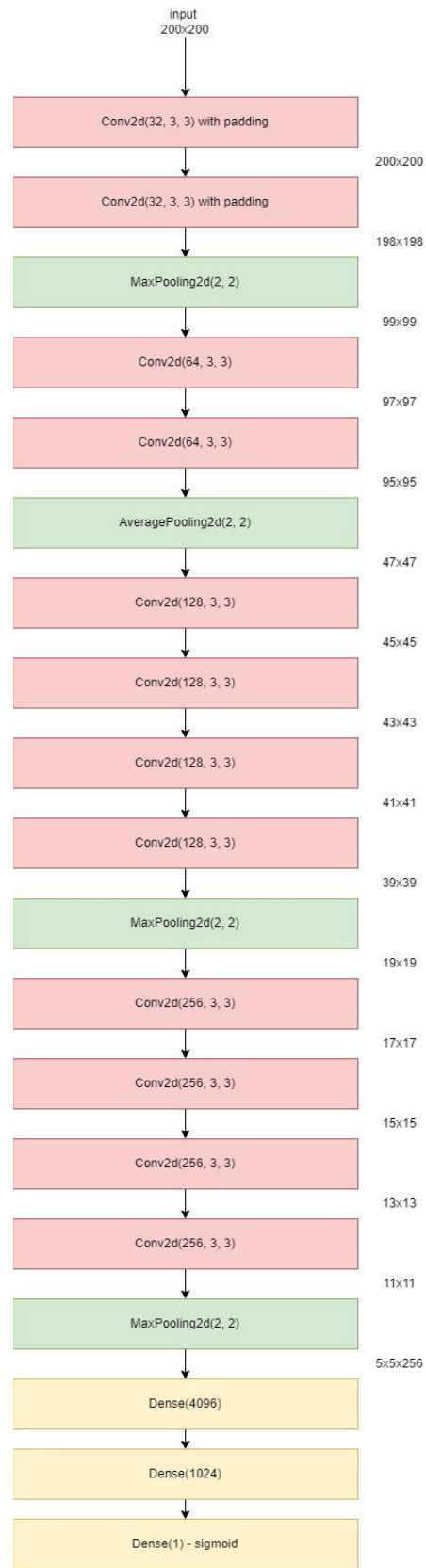


Figure 17 The last architecture



This architecture is somewhat similar to vgg-net architecture. It has deeper layers and much more trainable parameters than the previous model (600 thousand parameters against 33 million), which shows us that the power of this model is more than the previous models.

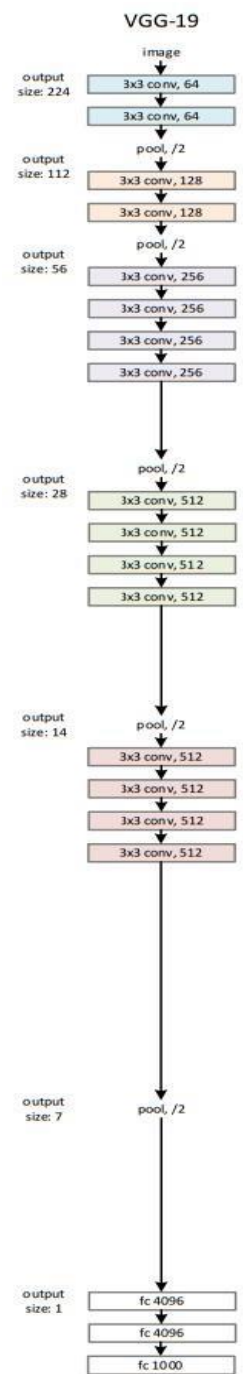


Figure 19 VGG-NET

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 200, 200, 32)	896
conv2d_1 (Conv2D)	(None, 198, 198, 32)	9248
max_pooling2d (MaxPooling2D)	(None, 99, 99, 32)	0
conv2d_2 (Conv2D)	(None, 97, 97, 64)	18496
conv2d_3 (Conv2D)	(None, 95, 95, 64)	36928
average_pooling2d (AveragePooling2D)	(None, 47, 47, 64)	0
conv2d_4 (Conv2D)	(None, 45, 45, 128)	73856
conv2d_5 (Conv2D)	(None, 43, 43, 128)	147584
conv2d_6 (Conv2D)	(None, 41, 41, 128)	147584
conv2d_7 (Conv2D)	(None, 39, 39, 128)	147584
max_pooling2d_1 (MaxPooling2D)	(None, 19, 19, 128)	0
conv2d_8 (Conv2D)	(None, 17, 17, 256)	295168
conv2d_9 (Conv2D)	(None, 15, 15, 256)	590880
conv2d_10 (Conv2D)	(None, 13, 13, 256)	590880
conv2d_11 (Conv2D)	(None, 11, 11, 256)	590880
max_pooling2d_2 (MaxPooling2D)	(None, 5, 5, 256)	0
flatten (Flatten)	(None, 6400)	0
dense (Dense)	(None, 4096)	26218496
dense_1 (Dense)	(None, 1024)	4195328
dense_2 (Dense)	(None, 1)	1025
Total params: 33,062,433		
Trainable params: 33,062,433		
Non-trainable params: 0		

Figure 18 Model summary



Finally, by training the models on their own big data and testing them, the accuracy reached to an average of 85% from 20%.

```
1/1 [=====] - 1s 1s/step - loss: 0.0488 - accuracy: 0.9643 - precision: 1.0000 - recall: 0.9286  
that was class 0, saving the model in D:\images for the project\saved_models\saved_model_0.
```

```
1/1 [=====] - 1s 921ms/step - loss: 0.6422 - accuracy: 0.8500 - precision: 1.0000 - recall: 0.7000  
that was class 1, saving the model in D:\images for the project\saved_models\saved_model_1.
```

```
1/1 [=====] - 1s 1s/step - loss: 0.3625 - accuracy: 0.8462 - precision: 0.8000 - recall: 0.9231  
that was class 2, saving the model in D:\images for the project\saved_models\saved_model_2.
```

```
2/2 [=====] - 2s 567ms/step - loss: 1.1226 - accuracy: 0.6667 - precision: 0.6400 - recall: 0.7619  
that was class 3, saving the model in D:\images for the project\saved_models\saved_model_3.
```

```
2/2 [=====] - 0s 15ms/step - loss: 1.0241 - accuracy: 0.8333 - precision: 0.9375 - recall: 0.7143  
that was class 4, saving the model in D:\images for the project\saved_models\saved_model_4.
```

```
1/1 [=====] - 1s 1s/step - loss: 0.5193 - accuracy: 0.8333 - precision: 0.9000 - recall: 0.7500  
that was class 5, saving the model in D:\images for the project\saved_models\saved_model_5.
```

```
1/1 [=====] - 1s 1s/step - loss: 0.2825 - accuracy: 0.8929 - precision: 0.8667 - recall: 0.9286  
that was class 6, saving the model in D:\images for the project\saved_models\saved_model_6.
```

```
1/1 [=====] - 1s 1s/step - loss: 0.2525 - accuracy: 0.9333 - precision: 1.0000 - recall: 0.8667  
that was class 7, saving the model in D:\images for the project\saved_models\saved_model_7.
```

```
1/1 [=====] - 1s 1s/step - loss: 0.7486 - accuracy: 0.7308 - precision: 0.8000 - recall: 0.6154  
that was class 8, saving the model in D:\images for the project\saved_models\saved_model_8.
```

```
1/1 [=====] - 1s 1s/step - loss: 0.3096 - accuracy: 0.8929 - precision: 0.8667 - recall: 0.9286  
that was class 9, saving the model in D:\images for the project\saved_models\saved_model_9.
```

```
1/1 [=====] - 1s 1s/step - loss: 0.3570 - accuracy: 0.8846 - precision: 0.8571 - recall: 0.9231  
that was class 10, saving the model in D:\images for the project\saved_models\saved_model_10.
```

```
1/1 [=====] - 0s 52ms/step - loss: 0.7997 - accuracy: 0.8077 - precision: 0.9000 - recall: 0.6923  
that was class 11, saving the model in D:\images for the project\saved_models\saved_model_11.
```

```
1/1 [=====] - 1s 1s/step - loss: 0.6072 - accuracy: 0.8214 - precision: 0.8000 - recall: 0.8571  
that was class 12, saving the model in D:\images for the project\saved_models\saved_model_12.
```

```
1/1 [=====] - 0s 62ms/step - loss: 1.1533 - accuracy: 0.8571 - precision: 1.0000 - recall: 0.7143  
that was class 13, saving the model in D:\images for the project\saved_models\saved_model_13.
```

```
1/1 [=====] - 0s 64ms/step - loss: 1.0821 - accuracy: 0.6786 - precision: 0.6471 - recall: 0.7857  
that was class 14, saving the model in D:\images for the project\saved_models\saved_model_14.
```

```
1/1 [=====] - 1s 1s/step - loss: 0.7140 - accuracy: 0.7857 - precision: 0.7500 - recall: 0.8571  
that was class 15, saving the model in D:\images for the project\saved_models\saved_model_15.
```

Figure 20 Test accuracy of models

```

model = Sequential()

model.add(Conv2D(32, (3, 3), padding='same', input_shape=(200, 200, 3), activation='relu'))
model.add(Conv2D(32, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(AveragePooling2D(2, 2))
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(256, (3, 3), activation='relu'))
model.add(Conv2D(256, (3, 3), activation='relu'))
model.add(Conv2D(256, (3, 3), activation='relu'))
model.add(Conv2D(256, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Flatten())
model.add(Dense(4096, activation='relu'))
model.add(Dense(1024, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

loss = BinaryCrossentropy(from_logits=False)
optimizer = Adam(learning_rate=0.0001)
metrics = ['accuracy', Precision(), Recall()]
model.compile(optimizer=optimizer, loss=loss, metrics=metrics)

print('model created and complied')
print(f'model summary:\n{model.summary()}')
for i in range(15, 16):

    train_data = np.loadtxt(f'D:\\new data set\\{i}_train.csv', delimiter=',')
    print(f'class {i} train data loaded -> train: {train_data.shape}')
    x_train = train_data[:, :120_000] / 255.0
    y_train = train_data[:, 120_000:]
    del train_data
    x_train = x_train.reshape(len(x_train), 200, 200, 3)
    print(f'train data : {x_train.shape}\ny train : {y_train.shape}\n')
    batch_size = 100
    epochs = 20
    model.fit(x_train, y_train, epochs=epochs, batch_size=batch_size, verbose=2)
    print('training completed, loading the test data:')
    test_data = np.loadtxt(f'D:\\new data set\\{i}_test.csv', delimiter=',')
    print(f'test data loaded -> test: {test_data.shape}')
    x_test = test_data[:, :120_000] / 255.0
    y_test = test_data[:, 120_000:]
    del test_data
    x_test = x_test.reshape(len(x_test), 200, 200, 3)
    print(f'test data : {x_test.shape}\ny test: {y_test.shape}\n')
    model.evaluate(x_test, y_test)
    print(f'that was class {i}, saving the model in D:\\new data set\\saved_models\\saved_model_{i}.')
    model.save(f'D:\\new data set\\saved_models\\saved_model_{i}.')
    del x_train, x_test, y_train, y_test
    model.reset_states()

```

Figure 21 main.py

## • Prediction.py

This section contains the frontend of the project. As mentioned earlier, it is in the form of a desktop application, and the way it works, is that it first loads the models saved in the previous step, and then a page appears where you can drag and drop your desired photo on this page. Then the application will show you the predictions of the models.

```
if __name__ == "__main__":
    models = [None for i in range(16)]
    for i in range(16):
        print('loading model: ', i)
        models[i] = load_model(f'D:\\new data set\\saved_models\\saved_model_{i}.h5')

    app = QApplication(sys.argv)
    mainWindow = MainWindow(models)
    for i in range(16):
        item = QTableWidgetItem(class_names[i])
        mainWindow.table.setItem(i, 0, item)
    mainWindow.show()

    sys.exit(app.exec())
```

```
1 usage  Amir Arsalan Sanati *
class MainWindow(QMainWindow):
    Amir Arsalan Sanati *
    def __init__(self, face_models):
        super().__init__()
        self.setWindowTitle("Image face recognizer")
        self.setGeometry(100, 100, 800, 600)
        self.face_models = face_models
        # Create a central widget
        central_widget = QWidget()
        self.setCentralWidget(central_widget)

        # Create a vertical layout for the central widget
        layout = QVBoxLayout(central_widget)

        # Create a label for drag and drop section
        self.label = QLabel("Drag and drop an image here", self)
        self.label.setAlignment(Qt.AlignCenter)

        # Create a table for the table section
        self.table = QTableWidgetItem(16, 2, self)
        self.table.setHorizontalHeaderLabels(["Person's name", "Probability"])

        # Add the label and table to the layout
        layout.addWidget(self.label)
        layout.addWidget(self.table)

        # Set the layout for the central widget
        central_widget.setLayout(layout)
```

The image prediction section is implemented in the **process\_image** function. This function is called inside the **dropEvent** function, which detects the photo drop events and extracts the dropped photo address.

```
Amir Arsalan Sanati
def dropEvent(self, event):
    if event.mimeData().hasUrls():
        file_path = event.mimeData().urls()[0].toLocalFile()
        y_pred_new = self.process_image(file_path)
        print('received')
        for index in range(len(y_pred_new)):
            item = QTableWidgetItem(f'{y_pred_new[index]}')
            mainWindow.table.setItem(index, 1, item)
        pixmap = QPixmap(file_path)
        self.label.setPixmap(pixmap.scaled(400, 300, Qt.AspectRatioMode.KeepAspectRatio))
        event.accept()

1 usage Amir Arsalan Sanati *
def process_image(self, image_path):
    # Perform further processing with the image path
    im = Image.open(image_path)
    if im.mode != 'RGB':
        im = im.convert('RGB')

    im = im.resize((200, 200))
    im_copy = np.array(im)
    im_copy = im_copy / 255.0
    im_copy = im_copy.reshape((1, 200, 200, 3))
    y_pred = [0 for i in range(16)]
    for i in range(16):
        y_pred[i] = self.face_models[i].predict(im_copy)[0][0]
        y_pred[i] *= probability_reduction[i]
        if y_pred[i] < 0.01:
            y_pred[i] = 0

    print(y_pred)
    return y_pred
```

Figure 22 Some parts of prediction.py

- **Probability reduction:**

After training and saving the models, I tried the model in different ways to see its performance. I realized that some of the models didn't train well, and these models sometimes show very bad predictions. To make things a little better, a trust coefficient is defined in prediction.py which basically is a number between 0 and 1 and is multiplied by the prediction of each model. The values of probability\_reduction are obtained by experience and as you can see, it's basically the value of the test accuracy of the model I wanted to reduce the value of its predictions.

```

probability_reduction = {
  0: 1,
  1: 1,
  2: 1,
  3: 1,
  4: 1,
  5: 1,
  6: 1,
  7: 1,
  8: 1,
  9: 1,
  10: 1,
  11: 1,
  12: 0.82,
  13: 1,
  14: 0.6786,
  15: 1,
}

```

Figure 23 probability\_reduction

- SOME EXAPMLES:

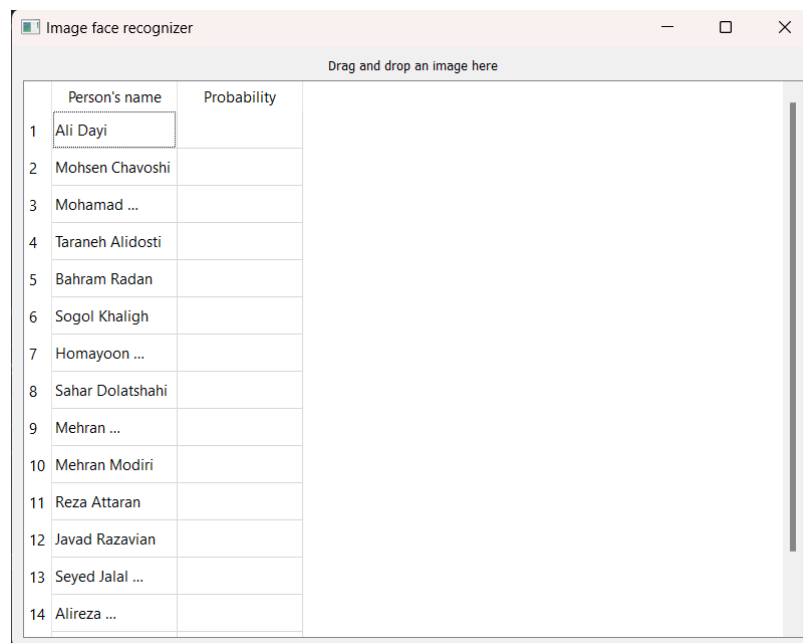




Image face recognizer



	Person's name	Probability
10	Mehran Modiri	0.9922210574150085
11	Reza Attaran	0.3487700819969177
12	Javad Razavian	0.055583320558071136
13	Seyed Jalal Hoseini	0.036166479438543314
14	Alireza Beyranvand	0.8427643179893494
15	Nazanin Bayati	0
16	Bahareh Kianafshar	0

Image face recognizer



	Person's name	Probability
7	Homayoon Shajarian	0
8	Sahar Dolatshahi	0
9	Mehran Ghafourian	0.028932999819517136
10	Mehran Modiri	0.9922210574150085
11	Reza Attaran	0.2005304992198944
12	Javad Razavian	0.42990028858184814
13	Seyed Jalal Hoseini	0.18530666388414063

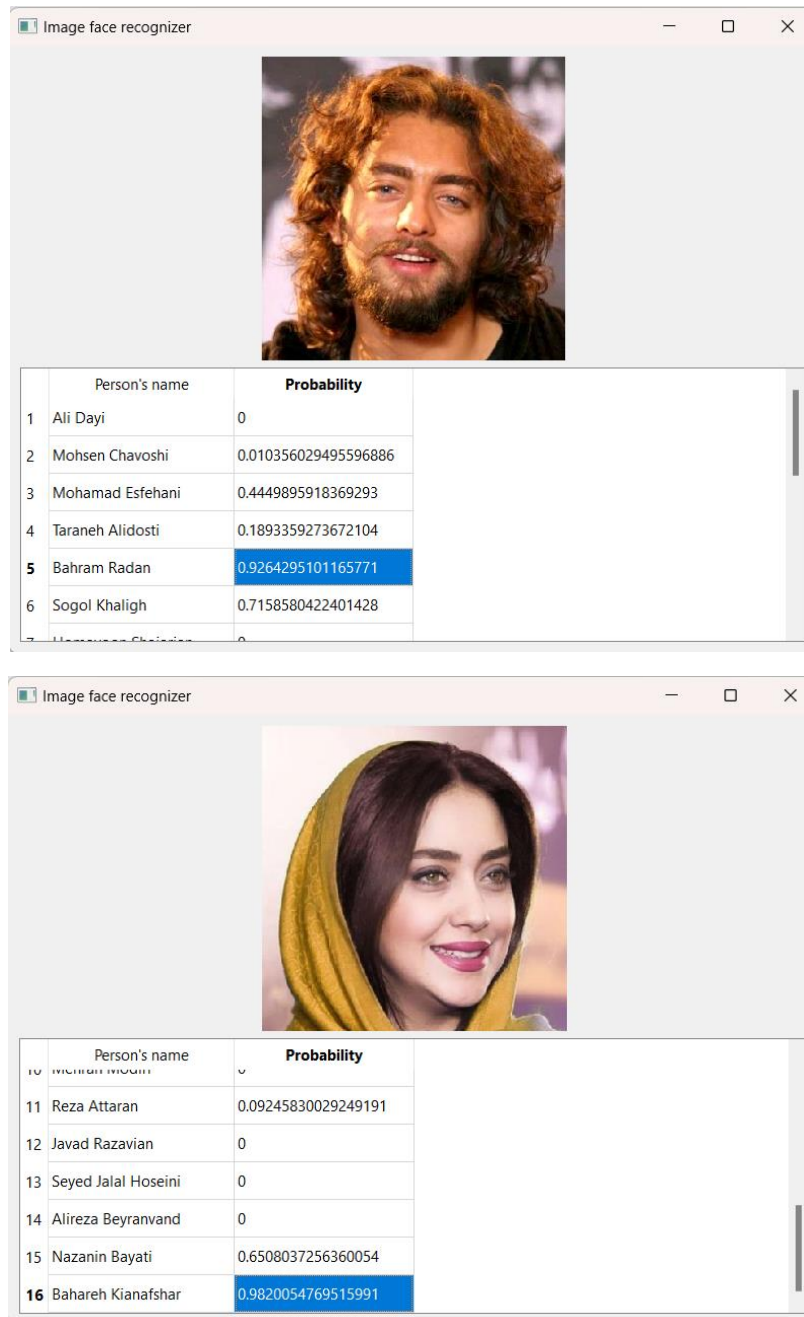


Figure 24 Testing some images

- **More evaluation**

After all training and evaluating and playing around stuff, I gathered a completely new data set of 170 images (approximately 10 images each class) and tested these images on my application. Here are the results:

TRUE:	1	2	3	None	sum
Ali Dayi	10	1			11
Mohsen Chavoshi	2	1			3
Mohamad Esfehni	3	4	1		8
Taraneh Alidosti	6	4	1		11
Bahram Radan	11	1		2	14
Sogol Khaligh	8	1	1	2	12
Homayoon Shajarian	9	2			11
Sahar Dolatshahi	8		1	1	10
Mehran Ghafourian	4	1	1	7	13
Mehran Modiri	8	1	1		10
Reza Attaran	7		3		10
Javad Razavian	8	2		1	11
Seyed Jalal Hoseini	9	3			12
Alireza Beyranvand	6	1		7	14
Nazanin Bayati		5	4	1	10
Bahareh Kianafshar	5	3	1		9
	<b>104</b>	<b>30</b>	<b>14</b>	<b>21</b>	<b>169</b>

Figure 25 170 test images results

The '1' column is the number of images which the model guessed completely correctly (by completely correct I mean, this class which predicted the highest probability, was actually the class of the test image)

The '2' column is the number of images which the model also guessed correctly, but the prediction value of intended class was at rank 2 (another class predicted the image with higher probability and the image didn't belong to that class).

The '3' column is the number of images which the models guessed correctly but at rank 3.

The 'none' column, is the number of images guessed completely wrong (the rank of the image's class was greater than 3).

If you see the numbers, you would get that, approximately 90% (87%) of the test images were guessed correctly (the images class were among the top three choices of the model). Among the correct guesses, 70% of them were rank 1, 20% of them were at rank 2, and the rest were at rank 3.

Also if we remove the two worst individual models (Mehran Ghafourian and Alireza Beyranvand) the percentage of correct guesses will reach to 95% of the hole test images (that means, we don't want to predict their images, and they will be removed from the list of celebrities which we intend to predict their faces).



TRUE:	1	2	3	None	sum
Ali Dayi	10	1			11
Mohsen Chavoshi	2	1			3
Mohamad Esfehane	3	4	1		8
Taraneh Alidosti	6	4	1		11
Bahram Radan	11	1		2	14
Sogol Khaligh	8	1	1	2	12
Homayoon Shajarian	9	2			11
Sahar Dolatshahi	8		1	1	10
Mehran Modiri	8	1	1		10
Reza Attaran	7		3		10
Javad Razavian	8	2		1	11
Seyed Jalal Hoseini	9	3			12
Nazanin Bayati		5	4	1	10
Bahareh Kianafshar	5	3	1		9
	<b>94</b>	<b>28</b>	<b>13</b>	<b>7</b>	<b>142</b>

Figure 26 Bad models removed

More improvements can be applied to the model by gathering more data for train images and designing separate architectures for each class. But a much easier solution to use would be transferred learning or using strong pre-trained models like ResNet to predict the faces. I didn't want to use pre-trained models or even transferred learning, I wanted to tune the model myself and see how it goes. But you definitely can do whatever you want to do and it's up to your curiosity.

**THE END.**