Task 2:

Given a database of arrays that represent images, how do we know if a new image added is within the database or not.

Starting hypothesis:

We can start analysing 3 areas. Checksums, hash functions and other data representation methods. Since we know all we have to do is check if an array of numbers is within the library, I decided to use a MD5 hash to hash the whole file that is produced by the program. It will save this MD5 hash to a list. We can then reference this list as a pseudo database. We can now take the new image, transform it into a matrix and then apply the MD5 hash to it. We can then check if the outputted hash of the new image is inside the database.

At this point we will be just representing the database as a simple list object in python. This can then be easily indexed and accessed by built in python methods.

Implementation:

The front portion of the code is simple. We reuse the code from task 1 and convert the image file the user has inputted into a matrix and save it as a .txt file.

Here is the code for that function. I have modified it to be a function for code reusability.

```python
import hashlib
from PIL import Image

file_name = input("please input the file name of this image\n")
def picture_to_matrix(file_name):
    try:
        img = Image.open(file_name)
        #we check if the image file is existent
    except:
        print("Error file specified does not exist")
        #error if the file is not found :(
    else:
        img = Image.open(file_name)
        grey = img.convert("LA")
        with Image.open(file_name) as image:
            width, height = image.size
        print(width)
        print(height)
        data = grey.load()

        holding_img_data = []
        for x in range(width):
            for y in range(height):
                holding_img_data.append(data[x,y])
        img_data = []
        for cord in holding_img_data:
            img_data.append(cord[0])

        # now we write this to a file
        file_handle = open(str(file_name).strip(".jpg")+".txt","w")
        for i in img_data:
            file_handle.write(str(i)+",")
        file_handle.write(str(width)+",")
        file_handle.write(str(height)+",")
        file_handle.close()

picture_to_matrix(file_name)
```

The second part of the code it the fun part. Since the MD5 function from imported hashlib has only a set number we can input into the hashfucntion, we have to break it up and this what the function md5 is doing.

We now can parse in the file name of the matrix into the function md5, returning a string of MD5 hash. We save this hash to an array. To check if the new image file the user provides is in the "database", we simply hash the new image, and we check if that hash matches any hash in the array. If it does, it is a match, otherwise it is not in the array.

Here is the code:

```
def md5(file_name):
    hash_md5 = hashlib.md5()
    with open(file_name, "rb") as file_handle:
        for chunk in iter(lambda: file_handle.read(4096), b""):
            hash_md5.update(chunk)
    file_handle.close()
    return hash_md5.hexdigest()

#these are the "Database" that we will be testing

database = []
#we mannually input the hashes into the "database"
database.append(md5("face1.txt"))
database.append(md5("face2.txt"))
database.append(md5("face3.txt"))
database.append(md5("face4.txt"))
database.append(md5("face5.txt"))
#checking if the MD5 of an image is withn the databse:
if md5(str(file_name).strip(".jpg")+".txt") in database:
    print("it is in the database")
else:
    print("it is not in the database")
```

Further prospects and limitations of the system:

One of the main drawbacks of this is the rigidness of the detection system. MD5 is very sensitive to the data parsed into it to hash. As a result, any slight deviation from the original, something as simple as an increase in brightness into the matrix or the corruption of one single value in the matrix parsed into MD5 will result in an error. However to the human eye, they can still be considered the "same" image.

One other alternative is to compare the image matrix to matrix, one upside is that we can see how the 2 are different and detect the percentage similarity of the 2 matrixes. We can also set a threshold value for how many percent close is the ideal. However this is quite time and space complex and require additional computational power. This is also flawed as it can quite easily tricked. Since all we require is some portion of the image to line up, this can raise many more false positives than the hashing checksum algorithm.