Research Engineers Task 2

# How many faces?

You are provided with images of people at meetings, gatherings, group photos, etc. Count the number of faces you can spot in each image. There will be no more than 15 faces in each of the images. Assume that half or more of each face will be visible.

Here are the first three images corresponding to the 3 sample test cases which will be provided







## Input Format

The first line of input will contain two integers, R and C, representing the number of rows and columns of image pixels, respectively.

A 2D grid of pixel values will be provided (in regular text format through STDIN) that represents the pixel-wise values from the images. The images were originally in JPG or PNG formats.

Each pixel will be represented by three comma-separated values denoting the Blue, Green, and Red components, respectively. The pixel values will be in the range 0 to 255. There will be a space between consecutive pixels in the same row.

No input test case exceeds 15MB in size, most are within 5MB and many are less than 1MB. So you may gradually iterate on your solution to handle larger and more complex cases. Take care to account for very different kinds of faces!

**Output Format**

Output a single integer, the number of faces spotted in the provided image.

**Sample Input**

This example is for explanation only. The real inputs will be larger than this and will contain 30 rows and 60 columns.

3 3

0,0,200 0,0,10 10,0,0

90,90,50 90,90,10 255,255,255

100,100,88 80,80,80 15,75,255

The first line indicates the number of rows and columns 3x3. The above is an image represented by 3x3 pixels. For each pixel, the Blue, Green, and Red values are provided, separated by commas.

The top left pixel has Blue=0 , Green=0 , Red=200 .

The top right pixel has Blue=10 , Green=0 , Red=0 .

The bottom right pixel has Blue=15 , Green=75 , Red=255 .

The bottom left pixel has Blue=100 , Green=100 , Red=88 .

**Sample Output**

1

**SOLUTIONS**

1. Create a GitHub repository for your solution to the above problem and share it with us.

The link to the GitHub repository containing my solution: [Click Here](https://github.com/Daniel-DS-dev/Popoola-Daniel---IQube-Lab-Task-2-)

1. Discuss in detail how you solved the problem:

**NOTE**: I also discussed in details the solution to the task inside the notebook containing the code, so you can follow the explanation better there since you can see the programs/code and the explanation at the same time.

\* The task is to count the number of faces in particular images, I was provided with 3 images that have been converted to 2D grids of pixel values. The 2D grid of pixel value of each image was provided in separate text files.

To count the number of faces in each image, I had to first convert each image back to its image form or array form before it can be usable on any deep learning or machine learning model.

The first thing I did was to read in the image data of each image into my notebook, using Python's function, 'open()'. Using this function, I read in image data from each text file line by line. I created 3 different lists for each image data named: 'img1', 'img2' and 'img3'.

\* The image data I read in was read in regular text format, so I had to convert it back into its original image format (The images were originally in JPG or PNG formats). Then I will build a model to detect the number of faces in each image.

\* I will read the text document in line by line, and store them in separate lists, then I will write a function to clean each image data.

The image data of each image are stored in three different text files named: 'input00.txt', input01.txt' and 'input02.txt'

I will read-in each text document line by line, and store them in different lists

I will use Python's 'open()' function to open each file, and access it's content

**NOTE**: The text files are in a folder named 'input', so I must include the full file path for each text document to be able to read in the text document into my notebook.

After reading in the image data using the open() function, the data type of all the values in each image data was of type string, and my lists contained unwanted spaces that were read in with the data. Also, the first line in each text file had the dimension of the image, so I had to remove that from the list also. So, I had to do some cleaning.

For example in the first image data read in from the file, 'input00', the first line contains the dimension of the first image, followed by space indicated by '\n' then the second line contains pixel values of the first row in the image data.

Each pixel will be represented by three comma-separated values denoting the Blue, Green, and Red components, respectively

So, we can see the first pixel is '2, 6, 184', and the last pixel is '1, 10, 151', and then another space ('\n')

The first file has 418 rows and has 870 pixels per row, the second image has 173 rows and 291 pixels per row, and the third image contains 288 rows and 460 pixels per row

I then removed the first line of each list, and stored them in separate lists, so my image data lists now had only the pixel values of their image.

Then I cleaned up each image data and stored each cleaned data in a numpy array in a format that can be viewed as an image.

The final image array of each image will be a rank3 array, which will have a shape in the form of (x1, x2, 3)

Where:

1. x1 = number of rows
2. x2 = number of columns, or number of pixels per row in each image and
3. The number, 3 is the number of color channels in our array.

E.g. the shape of the first image after cleaning should be (418, 870, 3)

To get a rank3 array from a list, the list must have 2 nested lists within it. (a list of lists of lists)

Since our data was read-in in string format, All the pixels in each row of our image data is stored as a string value which contains pixels separated by a space as a delimiter. i.e. for row 1 in image 1 we have '2, 6, 184 .... 1, 10, 151'

Remember, each pixel is represented by three comma-separated values denoting the Blue, Green, and Red components (3 colour channels) respectively. The first pixel in the first row of the first image is '2,6,184'

I needed to store each pixel in its own list, and then each row will contain all the pixels in that row in a single list, and then all the rows in that image will be stored in another single list.

E.g. The first image has 418 rows, and 870 pixels per row. Each 870 pixels per row will stored in separate lists i.e. we will have 870 lists in each row, and then each 418 rows containing 870 lists each which will be stored in a single list. And that way we have a list of lists (418 lists) of lists (870 lists). i.e. the first row of the first image should look like: [[2, 6, 184] ... [1, 10, 151]]

Then I wrote a function, **'clean\_image\_data\_list'** to clean all the image data, i.e. remove unwanted space, put them in the wanted format, etc.

After which I checked to make sure I still had the same number of rows for each image data that I should have.

Then I converted each image data list into numpy arrays, and changed their data type to 'uint8'

The data type 'uint8' can be described as: Unsigned integer (0 to 255)

Our pixel values will be in the range 0 to 255, so the data type 'uint8' is perfect for us.

Now I will convert it to a numpy array with datatype of 'uint8' which is the datatype for image data.

After converting into numpy arrays, my image data is now in the right format, and I visualized each image using the 'matplotlib' and cv2 'module'

The input format of the images's color channels were in the BGR (Blue Green Red) format instead of RGB (Red Green Blue), so I changed the format to RGB and my images were now in the right formats and ready to be applied to a face detection model.

**Face Detection Model.**

The main task here is to count the number of faces in each picture.

As a human, I can see that we have 2 faces in the first picture, 3 images in the second picture, and 5 pictures in the 3rd picture, but a computer doesn't know this. Computer Vision is the field of data science where we teach computers to see images, and the world at large like a human does.

For a computer to see the world like humans, we feed it lots of image data, depending on the object we want it to see.

Here we want the computer to count the number of faces, ideally we will get lots of human faces and feed them into a machine learning or deep learning model preferably deep learning because we get to use things like Neural Networks which can learn from hundreds of thousands of data.

I don't have any training data for the task at hand, and getting them would take a while, so I am going to use an a pretrained model to detect the faces in each of our pictures.

The pretrained model I am going to use is the Multitask Cascaded Convolutional Neural Network (mtcnn)

**Face Detection using the Multitask Cascaded Convolutional Neural Network (mtcnn)**

The MTCNN is popular because it achieved then state-of-the-art results on a range of benchmark datasets, and because it is capable of also recognizing other facial features such as eyes and mouth, called landmark detection.

The network uses a cascade structure with three networks; first the image is rescaled to a range of different sizes (called an image pyramid), then the first model (Proposal Network or P-Net) proposes candidate facial regions, the second model (Refine Network or R-Net) filters the bounding boxes, and the third model (Output Network or O-Net) proposes facial landmarks.

The model is called a multi-task network because each of the three models in the cascade (P-Net, R-Net and O-Net) are trained on three tasks, e.g. make three types of predictions; they are: face classification, bounding box regression, and facial landmark localization.

The MTCNN module is very easy to use I created an instance of the model/network using the default weights. The default weights are the weighs gotten from the training data used to train this model, I can add the weights of my own training data if I have any which I don't, so I used the default weights.

Then applied the images to the model. I created a function **'detect\_faces'** for this.

Then I created another function **'draw\_image\_with\_boxes'** to see the part of the image the model detected as faces, and to make sure it detected the right thing.

Then I applied both functions to the images and printed out the number of faces in each image. all of which the model got right.

1. **Are there other ways the problem could be solved? Discuss briefly**

Yes, there are other ways the problem could be solved.

In terms of the model used to count the number of faces:

* 1. Instead of using a pretrained model, we could build a model from scratch, we will gather tens to hundreds of thousands of images of different faces at different angles under different conditions, and train a neural network using this data, calculate the loss and optimize the model until it is good enough to be deployed.
  2. Another way I could have solved it is to use other pretrained models to detect the number of faces. Examples of other pretrained models are:The pretrained model in OpenCV’s library that makes use of Haarcascades, or the HOG (Histogram of Oriented Gradients) model from the ‘dlib’ module, or the CNN face detection model also from the ‘dlib’ module.

I could have also used cv2’s DNN model.