

Optical Mark Recognition

May 25, 2021

*Ahmed Nabil Lotfy
Ahmed Ahmed Elrewaidy
Eslam Khalid Elfauomy
Akram Hesham Ragab
Basma Mohamed Ahmed*

TABLE OF CONTENT:

- *Image Preprocessing*
 - *Original size to resized image*
 - *RGB to Grayscale*
 - *Image Normalization*
 - *Image Blur*

Image Preprocessing:

In this step, the pipeline that the image take to be ready for the next step which is edge detection will be expressed. But before discussing this step we need to show the very first step which is reading the images from the file.

All our images exist in a file that is called samples in the same directory of the project itself (that can be optimized by implementing a GUI for more abstraction, and we intend to do this in a next version) and it's the job of the algorithm to read all the image in this file and run the main function (which is essentially the whole system) over all and every image that needs to be corrected. Then it will compare the answers. But this is still a very far away detail to be discussed now.

Now that we have access to the sheets we need to resize it without any distortions (maintaining the aspect ratio of the original image)

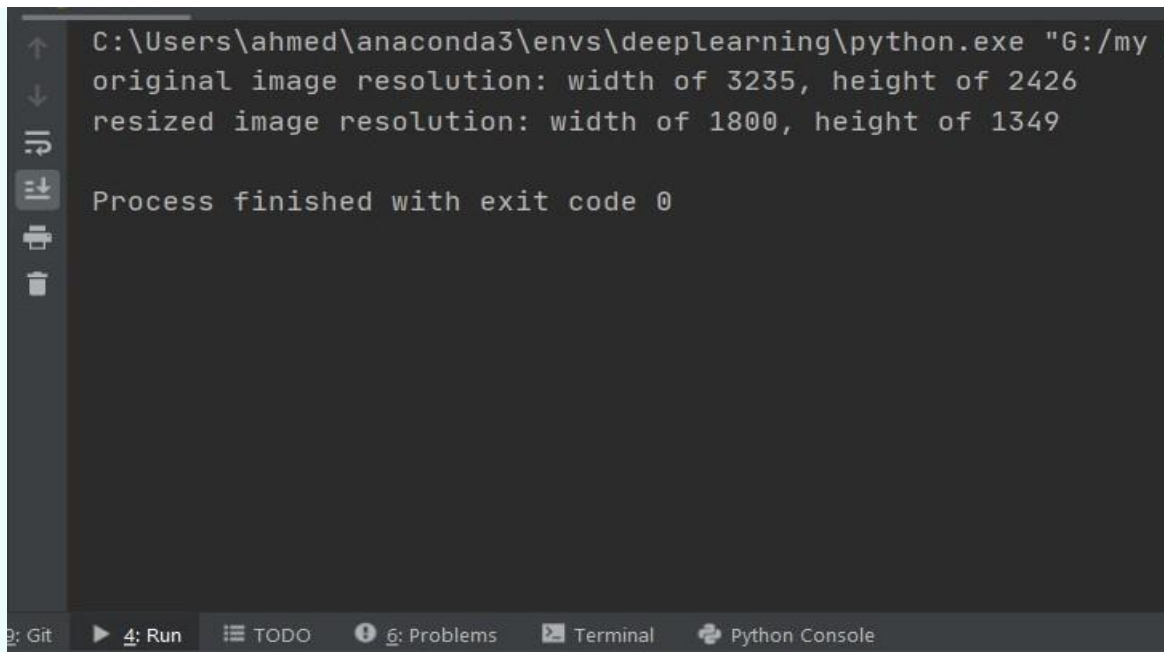
A screenshot of a terminal window with a dark background. The terminal shows the command prompt path C:\Users\ahmed\anaconda3\envs\deeplearning\python.exe. The output text reads: 'original image resolution: width of 3235, height of 2426' followed by 'resized image resolution: width of 1800, height of 1349'. Below this, it says 'Process finished with exit code 0'. The terminal interface includes a sidebar on the left with icons for file operations and a bottom status bar with tabs for 'Git', 'Run', 'TODO', 'Problems', 'Terminal', and 'Python Console'.

Fig1 the original vs resized image resolutions

As seen in the figure above, we set either the width or the height with an arbitrary number, in this example width of 1800 pixels, and the change in the other dimension (height) will change proportionally to the change of width maintaining the same aspect ratio. Now let's start image conversion preprocessing.

In OpenCV which is the image preprocessing main library used to build those systems, when image is read, it's read as BGR not RGB. We have to convert it because we will use the matplotlib library just right now which in its turn only accepts images that are RGB.

The next step will be converting the RGP image to grayscale images. Why? That's better for edge detection, we can consider it as an important step before edge detection.

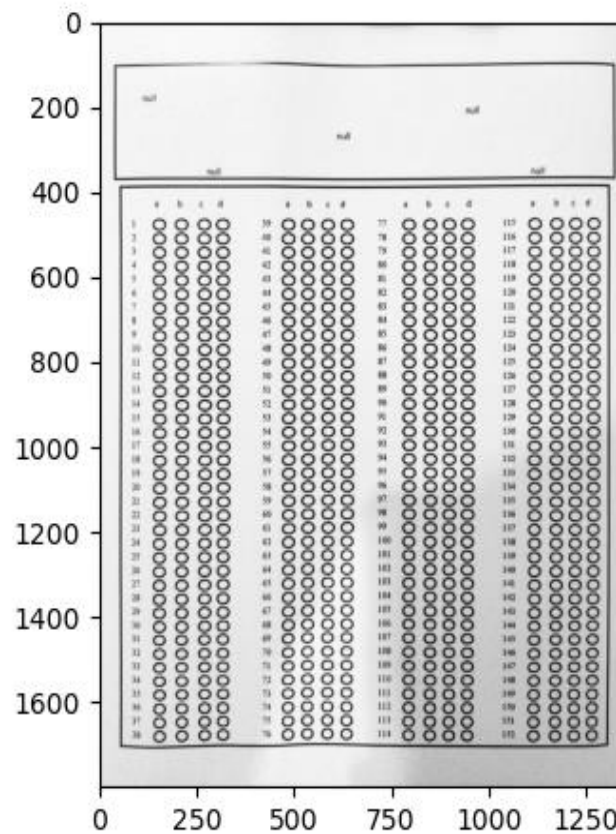


Fig 2 grayscale image

Now it's time for normalizing the image.

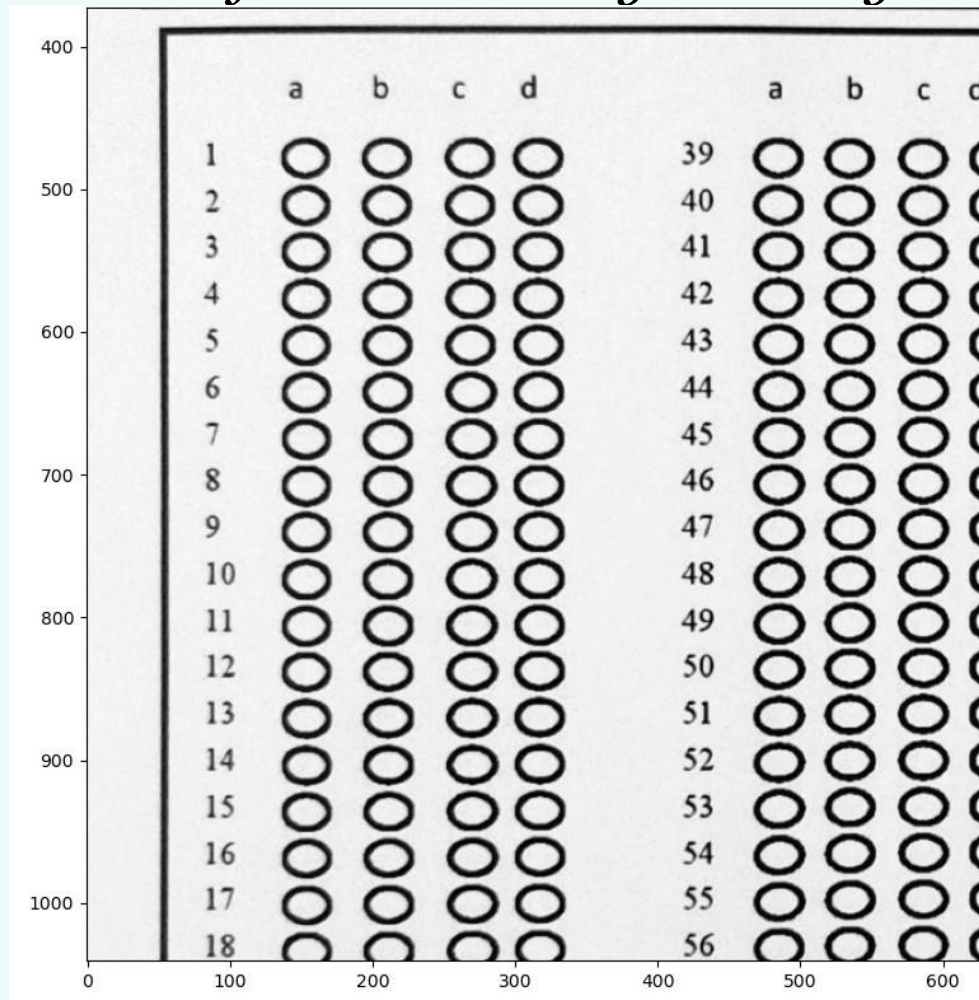


Fig 3 A part of the image after normalization

Normalization puts all pixels values in the image to a specific scale, which helps eliminating the domination of values over each other. However, the algorithm is working alright with or without normalization. It's usually used in machine and deep learning a little more often.

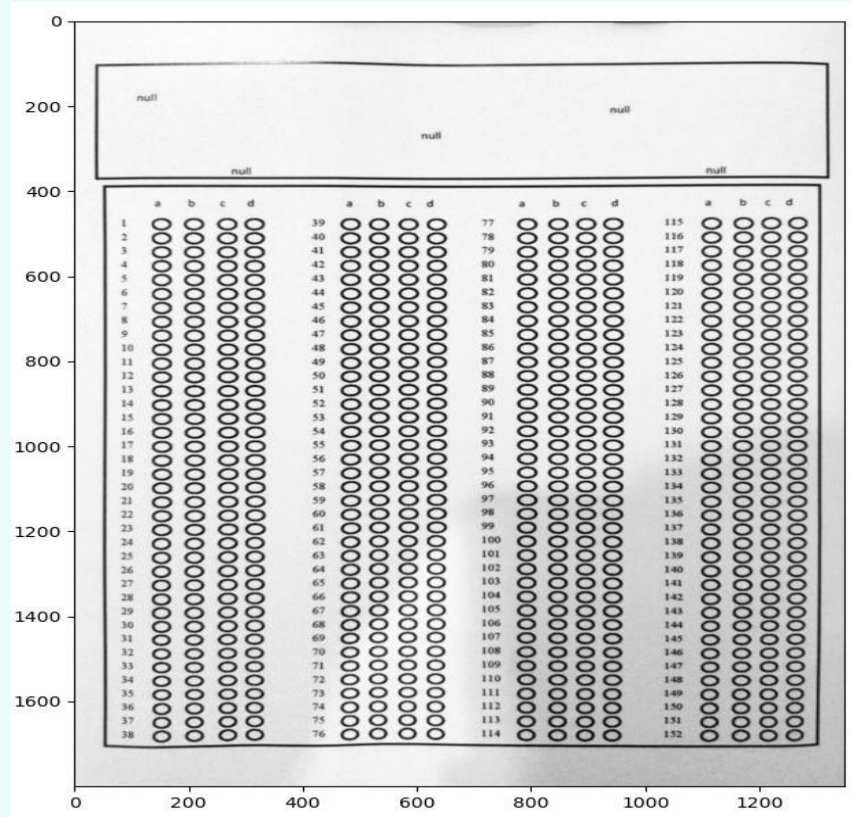


Fig 4 image blur

*Here comes a necessary step for edge detection in figure 2.6. If converting to grayscale is important to detect edges, applying a blurring algorithm to the image is essential as it really helps the edge detection algorithm to detect edges correctly. In our case we will use the Gaussian blur with a $5 * 5$ kernel (filter) size. However, we are not going to discuss further implementation details in this section.*

Now that we have image with a blur filter, we are ready to apply an edge detection. We have chosen canny edge detector for this system and this detector will be discussed in detail in the implementation chapter.