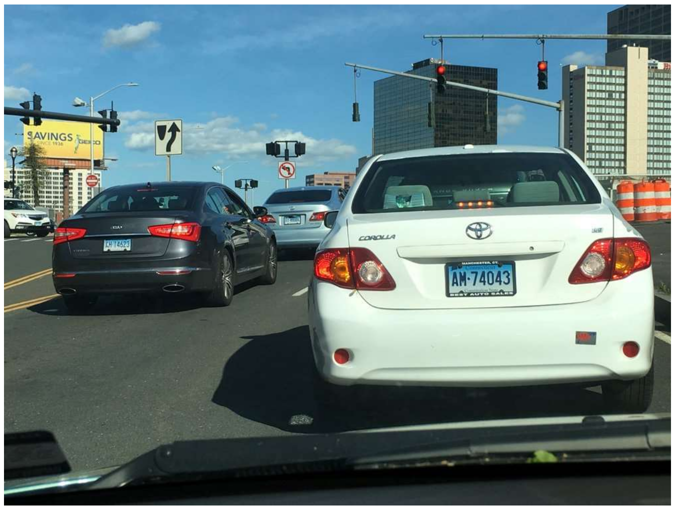
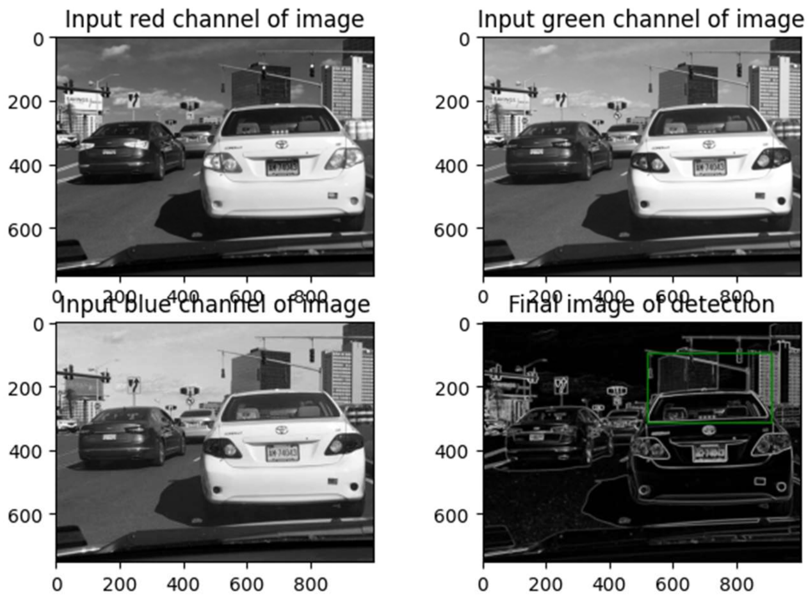
Things that need to be installed for the OCR for the extension. PIL, Pytesseract, Numpy

Difficult Images

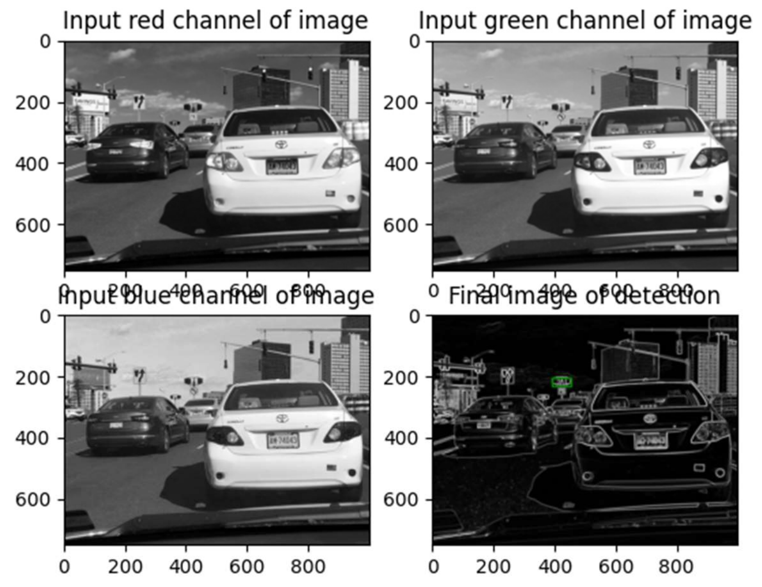


I picked the following image to challenge the algorithm I used.

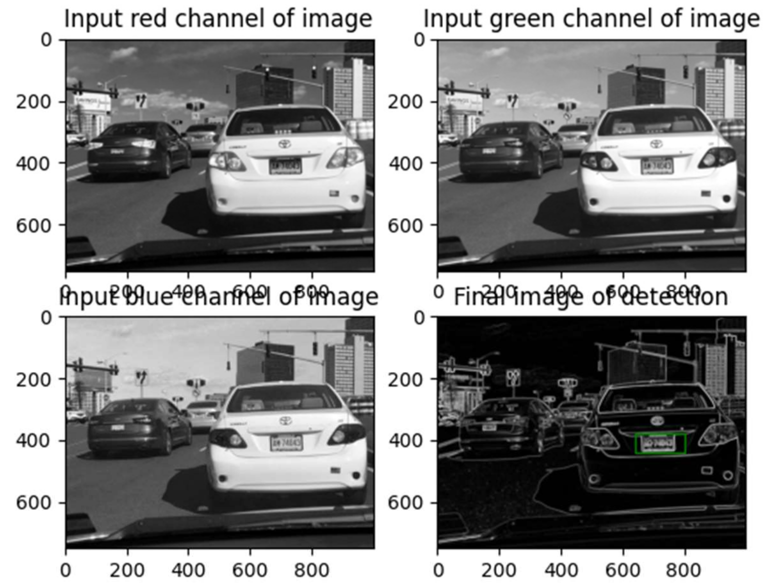
The algorithm I used to detect images is based on finding areas of high contrast. Each pixel's colour is based on the differences of the pixel's surrounding target pixel. Higher standard deviation leads to the "whiter" value. From here we dilate and erode to generate components. Knowing this, large areas that contrast with surroundings would create a false true. An example of this would be a white car with tinted windows. The white body and the black window would create high contrast and would cause the algorithm to detect it as a number plate. It also has similar aspect ratio to a number plate as well, which causes further problems. And is also a larger component than the number plate.



As we can see in the test run, the suspicions were correct and that the window has been detected as a larger component. To try fix this I tried to use a smaller range of aspect ratio such that it would pick up more narrow and wide aspect ratio. Instead of 1.5 < x < 5. I used 2 < x <5.

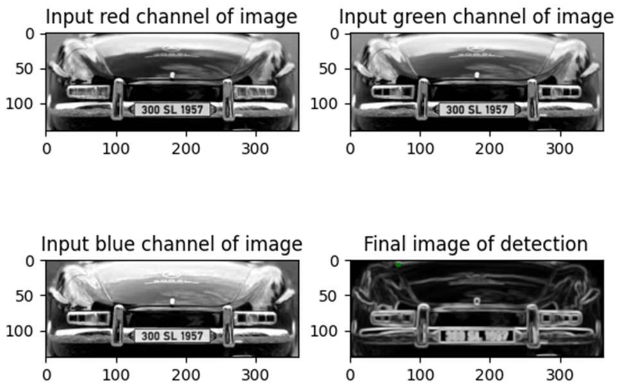

Input red channel of image
Input green channel of image
Input blue channel of image
Final image of detection

After changing the aspect ratio, the results are as seen above. The number plate though larger than the traffic light, has not been detected. On revision, I saw that the number plate has a blue background on it and therefore the contrast is not as large as we would like it to be. So I changed the threshold to a lower value (from 160 to 140). This produces the following image:

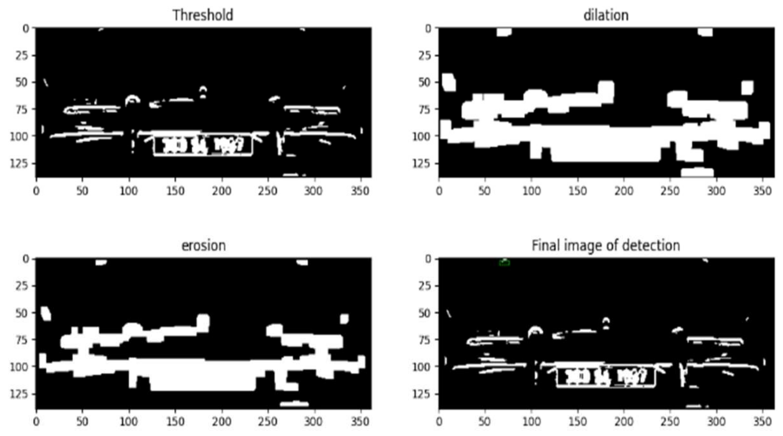

Input red channel of image
Input green channel of image
Input blue channel of image
Final image of detection

success!

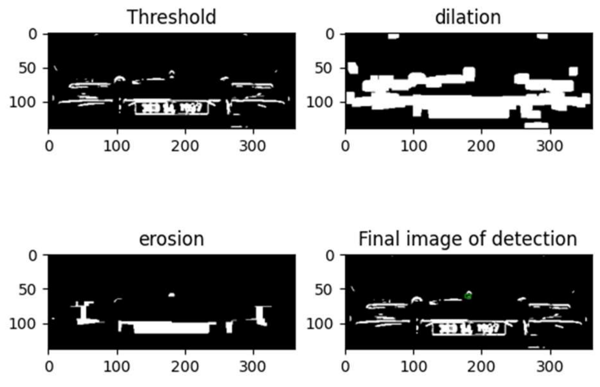
However, changing the threshold value may affect other images. A possible solution to this may be to use adaptive thresholding. However, this may also not be effective as adaptive thresholding may not be effective here as there are multiple objects and multiple different colours of background. IE the tarmac, the sky, the buildings and cars.


Input red channel of image
Input green channel of image
Input blue channel of image
Final image of detection

Again knowing that the basis of this algorithm is to detect number plates because of the contrast difference between the text of the number plate and the background of the plate. So a bunch of black text with white background, will cause a large component. If an image has lots of contrast around the number plate then when we dilate it will clump into one component. That's why I chose this image to challenge the algorithm. This gives us the following:



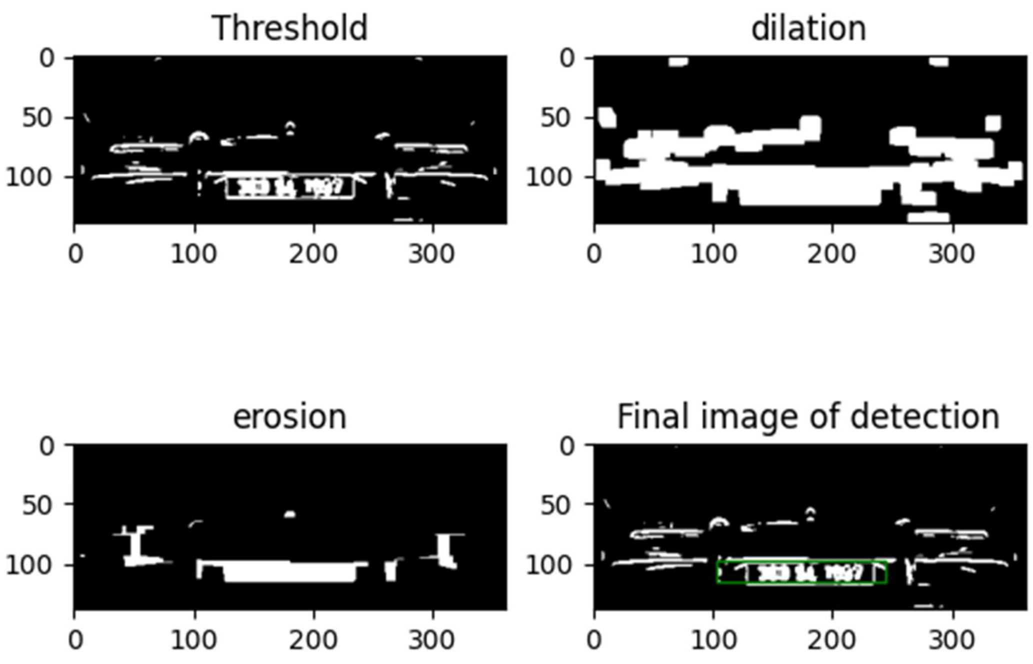
As can be seen all of the reflection from the car has clumped together with the number plate through dilation. Then because it clumps and the aspect ratio is too low, it doesn't detect the large clump whatsoever. Thus, to tackle this we want to get rid of all the thin horizontal lines on erosion. So if we use a 5x3 mask on erosion then the thin horizontal lines will disappear.



As can be seen on the left our approach was correct and the reflections on the car have been removed leaving only the number plate after erosion. However the number plate is still not detected.

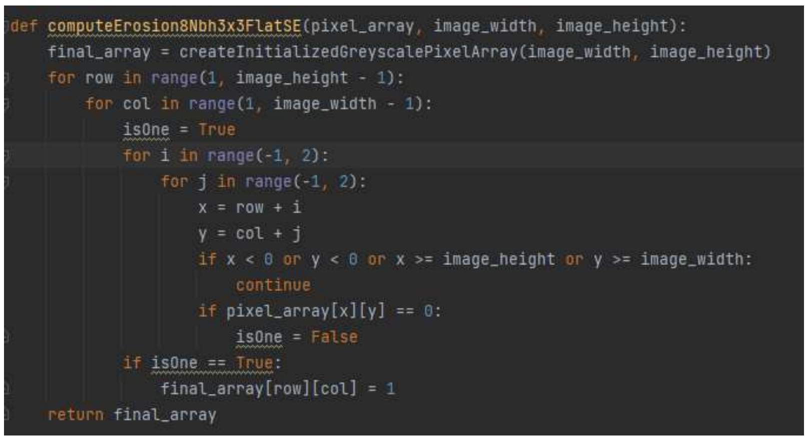
So we measure the number plate and see that the width is approx. 130 pixels and height of 20 pixels. Giving an aspect ratio of

6.5 which is out of our range and the component has been ignored due to our limit of an aspect ratio of 5. So, after changing our aspect ratio upper limit to 8 we get the following: (This will need to be changed in the code. It is currently set at 5.)



Success! We have overcome our high contrast image obstacle.

Note to achieve this will need to do the following:



```
def computeErosion8Nbh3x3FlatSE(pixel_array, image_width, image_height):
    final_array = createInitializedGreyscalePixelArray(image_width, image_height)
    for row in range(1, image_height - 1):
        for col in range(1, image_width - 1):
            isOne = True
            for i in range(-1, 2):
                for j in range(-1, 2):
                    x = row + i
                    y = col + j
                    if x < 0 or y < 0 or x >= image_height or y >= image_width:
                        continue
                    if pixel_array[x][y] == 0:
                        isOne = False
            if isOne == True:
                final_array[row][col] = 1
    return final_array
```

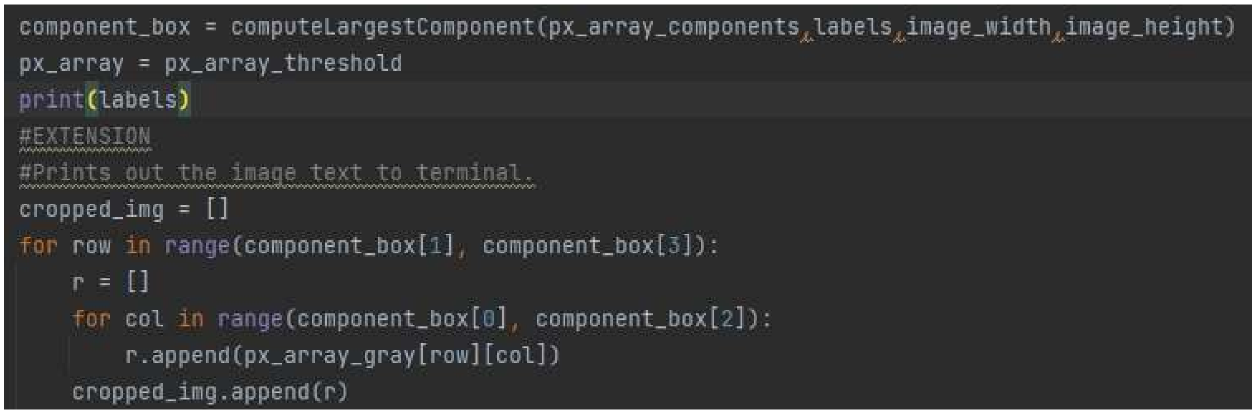
change: for i in range(-1,2):
to:for i in range(-2,3):

Reading the license plate

To read the license plate from the image I will use an OCR which is a can transform images that are hand written, screen shots, or print outs to turn data through a mixture of techniques of computer vision, AI, and pattern recognition.

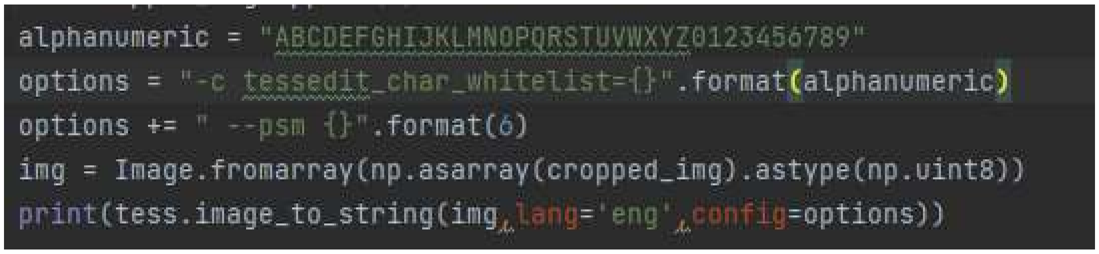
The one I used is pytesseract which is a wrapper for the tesseract OCR.

First we go through the normal algorithm and determine the boundary box of the license plate as usual. Then using the box boundary as the pixel coordinates for the original image we create a new numpy array of the image in grayscale by using loops.



```
component_box = computeLargestComponent(px_array_components,labels,image_width,image_height)
px_array = px_array_threshold
print(labels)
#EXTENSION
#Prints out the image text to terminal.
cropped_img = []
for row in range(component_box[1], component_box[3]):
    r = []
    for col in range(component_box[0], component_box[2]):
        r.append(px_array_gray[row][col])
    cropped_img.append(r)
```

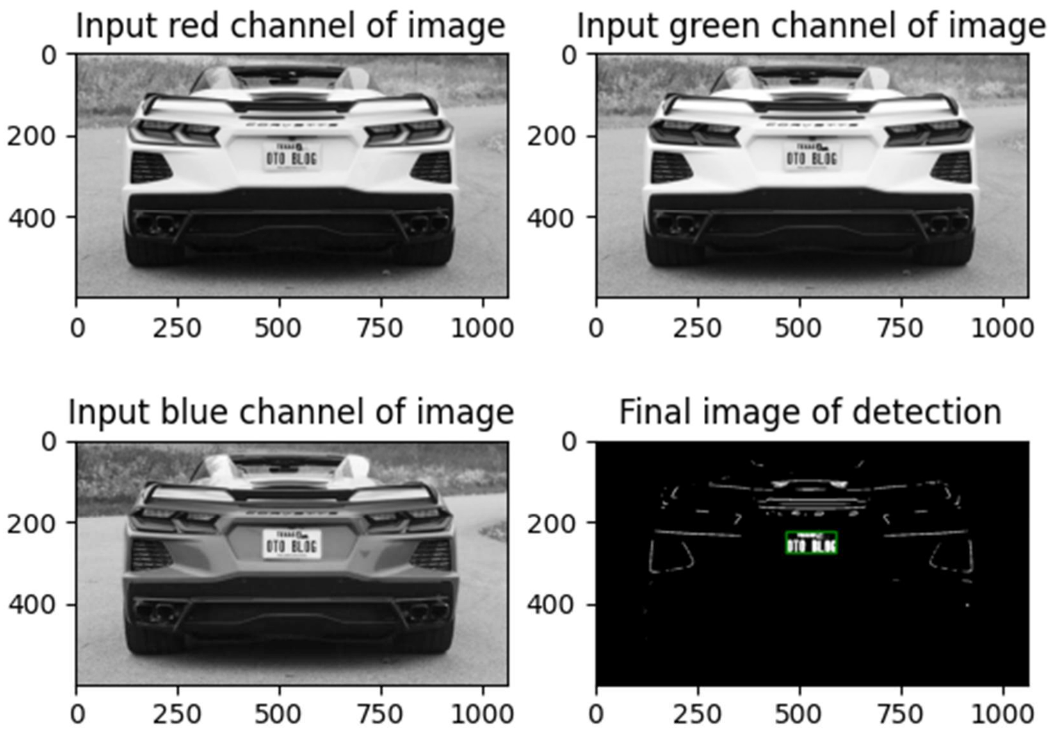
Since this image is cropped and only focused on the number plate, all the noise from the image is removed and simplifies the task for the OCR.

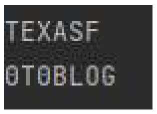


```
alphanumeric = "ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789"
options = "-c tessedit_char_whitelist={}".format(alphanumeric)
options += " --psm {}".format(6)
img = Image.fromarray(np.asarray(cropped_img).astype(np.uint8))
print(tess.image_to_string(img,lang='eng',config=options))
```

I then saved the numpy array as unsigned 8 bit integer as images are usually 32 bit signed integers. If this was not applied the image would appear as a pure black image.

Set the values that the OCR should recognise and then set the page segmentation method to 6 which is the method which assumes there is a single uniform block of text as is a license plate.


Input red channel of image
Input green channel of image
Input blue channel of image
Final image of detection
0
200
400
0
250
500
750
1000
TEXAS
OTO BLOG



```
TEXASF
0T0BLOG
```

Success! The license plate has been read successfully.