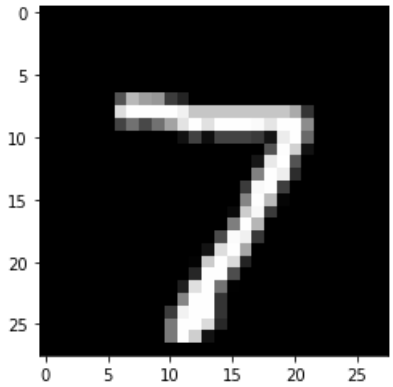
TESTING THE MODEL W/ NEW TEST SAMPLES

PART 1. QUICK DEBBUGING

--------- Prediction ---------



```
Prediction (Softmax) from the neural network:  [[0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]]

Final Output: 7
```

NOTE:

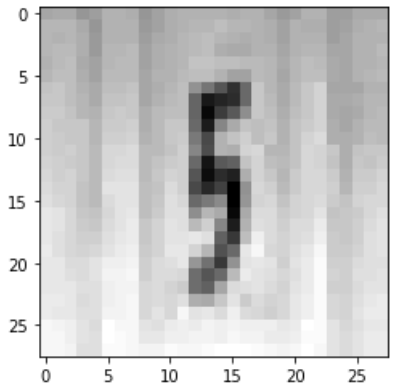
Here the model is doing great, it's able to predict what number is drawn on the input image.

QUICK DEBUGGING PART 2.

Comparing model predictions with MNIST and Street view samples

--------- Prediction ---------

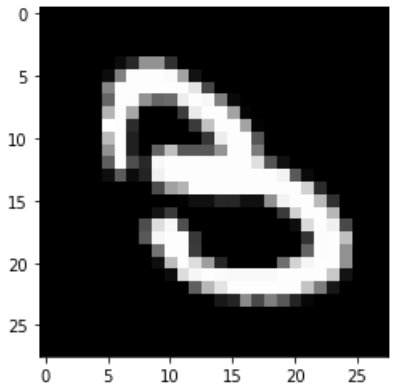
```
--------- preprocessed image ---------
```



```
type:  <class 'numpy.ndarray'>
shape:  (1, 28, 28, 1)
```

```
Prediction (Softmax) from the neural network:  [[1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
Final Output: 0


--------- preprocessed image ---------
```



```
type:  <class 'numpy.ndarray'>
shape:  (1, 28, 28, 1)


Prediction (Softmax) from the neural network:  [[0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]]
Final Output: 3
```

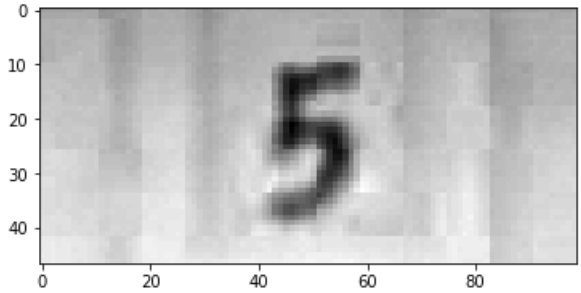
NOTE:

It seems that the model is not able to predict the digit from *The Street View House Numbers (SVHN)* Dataset.

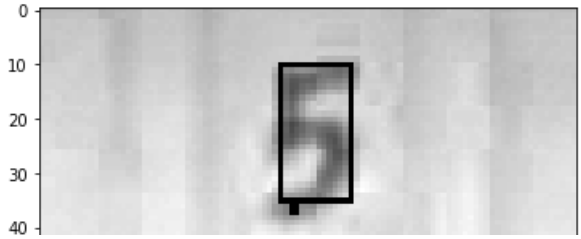
PART. 3 Testing Yashvardhan Kukreja method from Medium

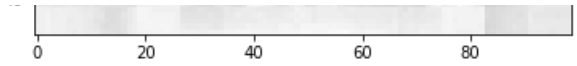
Found this article after some research, wich helped me to handle this problem: https://medium.com/@yash.kukreja.98/recognizing-handwritten-digits-in-real-life-images-using-cnn-3b48a9ae5e3

```
----------------Original Image (Grayscaled)--------------------
```

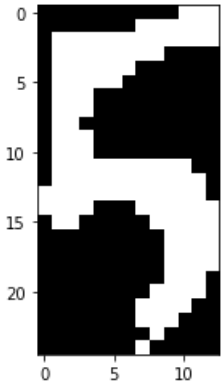


```
----------------Contoured Image--------------------
```

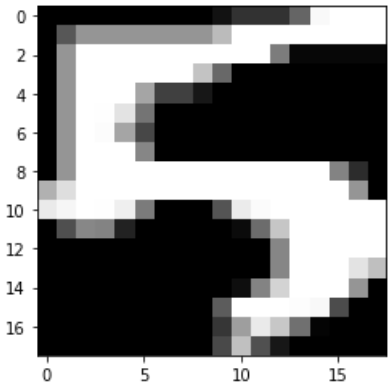




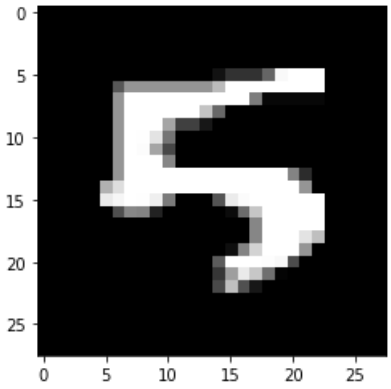

```
----------------Tresholding applied--------------------
```




```
----------------Resized Image--------------------
```




```
----------------Padded Image--------------------
```




```
Prediction (Softmax) from the neural network:  [[0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]]
Final Output: 5
```

DEBUG NOTE:

without adding coutours, tresholding and padding to the sample image, the model is not able to makes predictions.
The Model was mainly trained on images of a specific shape, size, and color (MNIST like images). ***It can ONLY predict those specific images.*** *We could make the model able to detect numbers in a wider range of situations, or with a more flexible manner, by adding more diverse images into the training set. We have to gathering those data by ourselves though.*

PART. 4 Testing with a VideoCapture

RESULTS

- 1ST STEP - IMAGE PROCESSING:

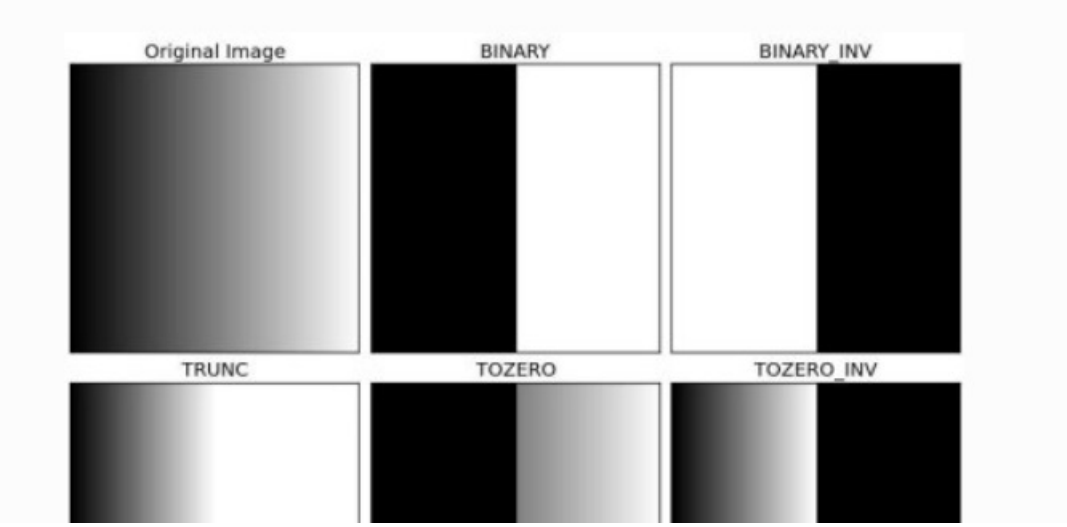
An image with 3 color channel (RGB) is 3 times heavier than a grayscale image, convert it to grayscale allow us to process it faster.

The original frame is converted into grayscale, then blurred for removing noise for the future thresholding.

- 2ND STEP - DRAWING CONTOURS:

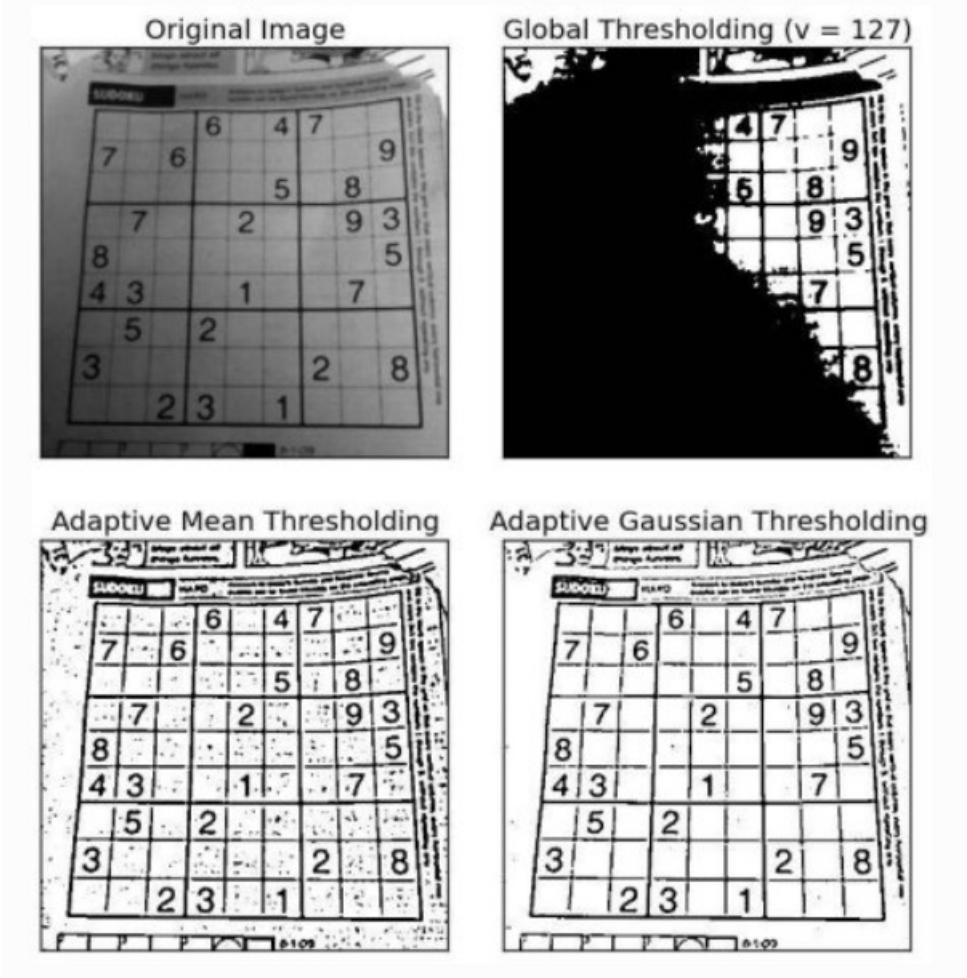
Binary Thresholding

Binary Thresholding works by defining a pixel intensity as a threshold, all pixels above that value is white, and all the pixels above is black. Binary thresholding doesn't seems ideal for video processing, because light tend to change a lot over frames.



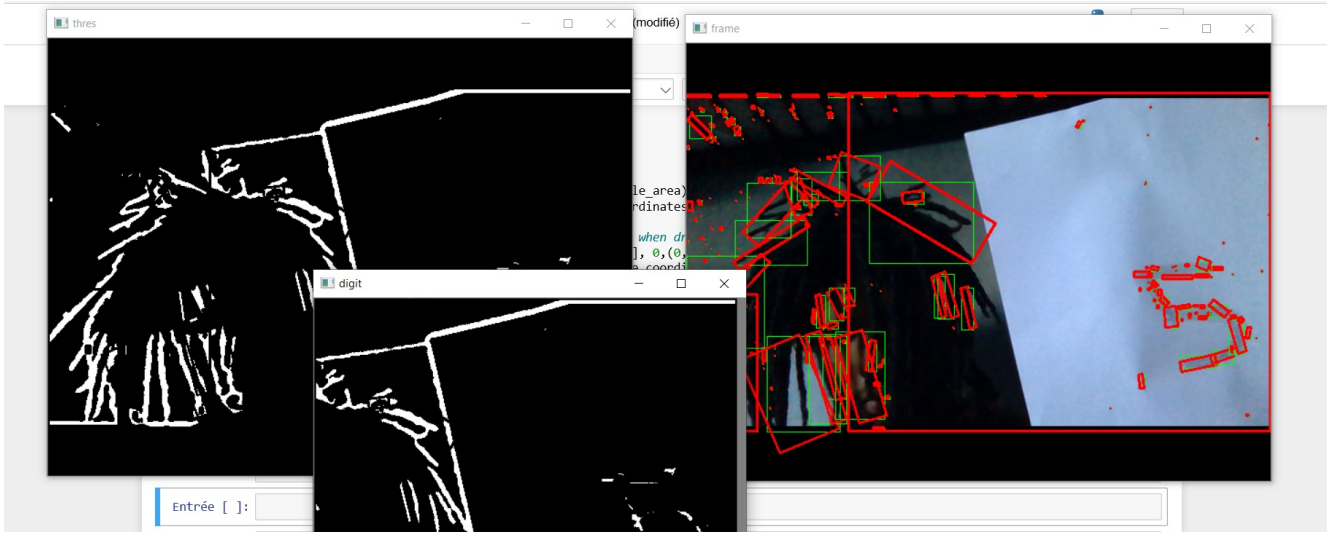
Adaptive Thresholding

The Adaptive threshold works better with variable light source. It takes into account the adjacent pixels during the process.



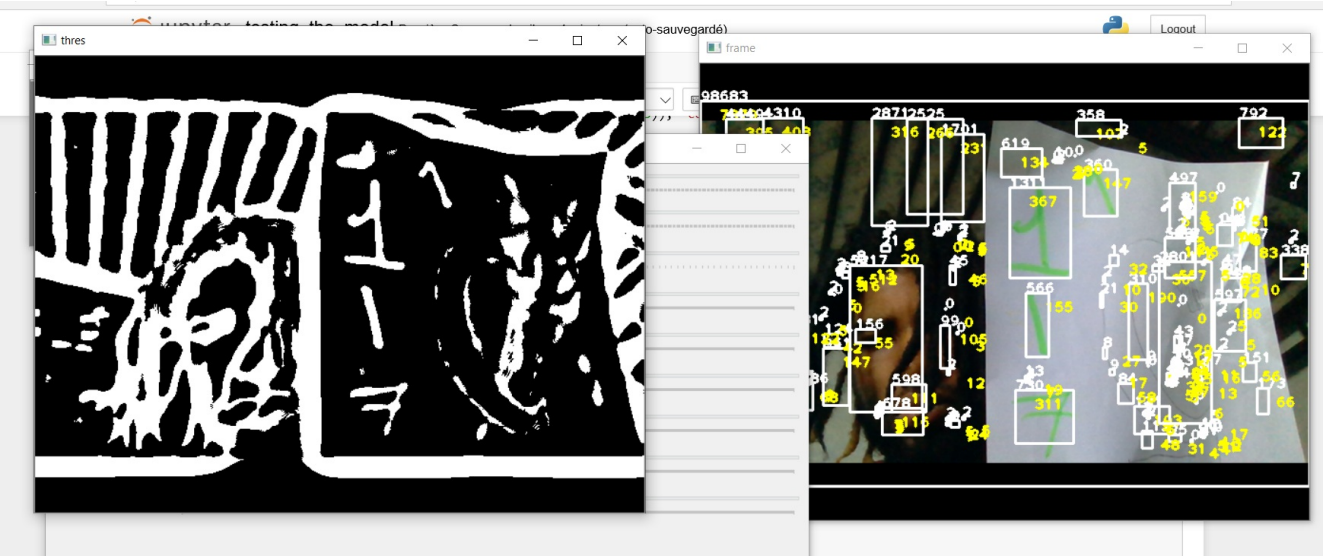
Contours

Contours is used on binary image to define the edge of the Region Of Interest (ROI), it allow us to detect points coordinates around the binary object. From those coordinates we can draw the edges by linking those points.



- 3ND STEP - REDUCING NOISE:

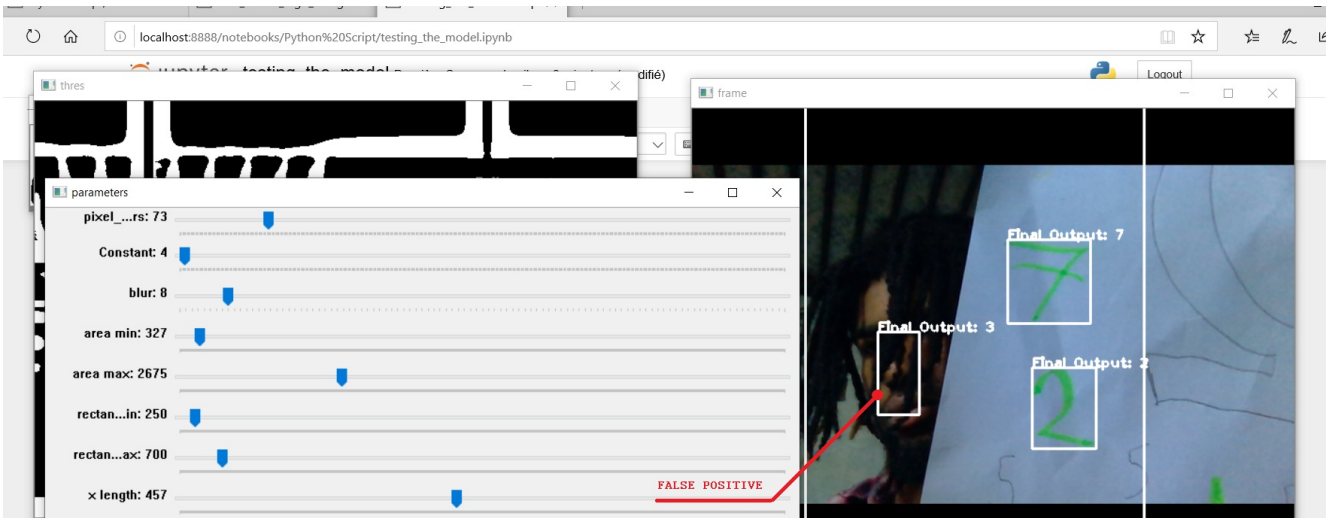
As we can see, there is too much noise to detect only the digit drawn on the sheet. By using **if conditions**, the **length** of the bounding box rectangle *(represented by the yellow numbers on the frame)*, and the **area** cover by the bounding box *(represented by the white numbers)*, we are able to get rid of the majority of the noise.

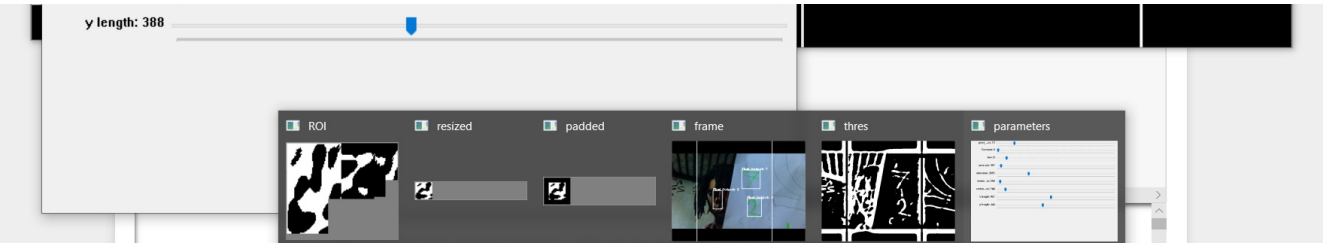


FINAL STEP - MAKE PREDICTION:

Note: not mentioned before, but i added limits (vertical white lines) based on the width of the screen to reduce the countours detection area. without them, big object in the background, or great pixels contrast, tend to make noise.

Here is an example of false positive:

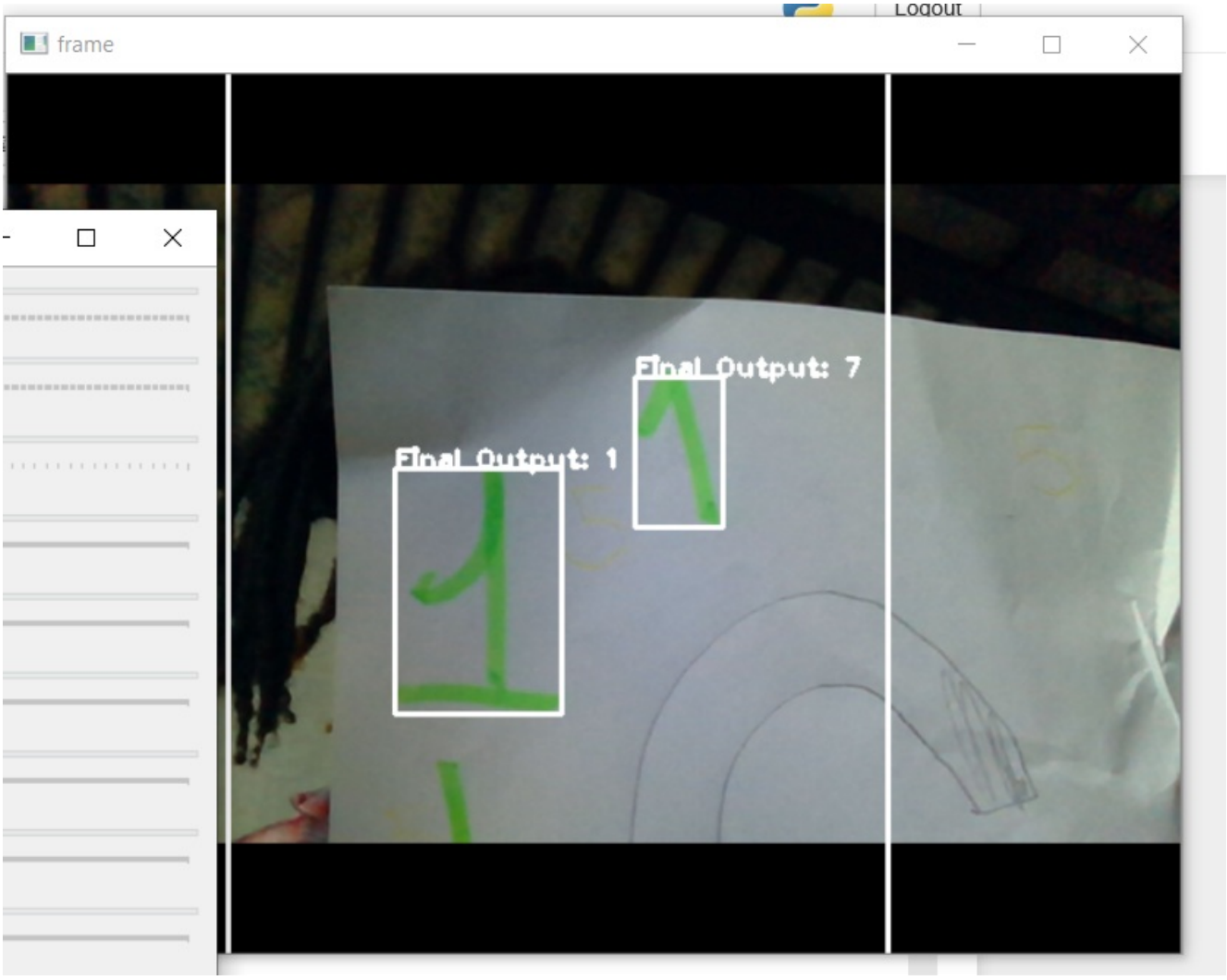




My face is detected as a "3" by the convolutional neural network, this is due to some noise not filtered out. Nonetheless, the number 2 and 7 has been correctly indentified.

Another example:

A **"1" is drawn** on the sheet, but the algorithm predict a **final output of "7"**.
Maybe it's due to the fact that the shape of ones and sevens is kind of similar, we could maybe train the model with more data from the MNIST dataset *(a sub-sample of 10 000 images was used to train this model)*, but simply add an horizontal line to the bottom of the "1" seems sufficient enough, to make the difference between those to two digits.



In [ ]: