Table of Contents

Introduction

The goal of this assignment is to apply the practical knowledge learnt in this unit to create a machine perception application. This assignment involves multiple tasks which include image processing, feature detection and extraction, and machine/deep learning, with the ultimate goal of extracting the digits of an image in various lighting and spatial conditions.

Due to the complexity of this problem, there is no simple pipeline to represent it as a single task can demand a whole pipeline by itself, however as a starting point and a guide we can use the following:

Image processing → feature detection → feature extraction → image classification / detection

In this assignment you will find tasks that needed all of the steps and some that only needed a few.

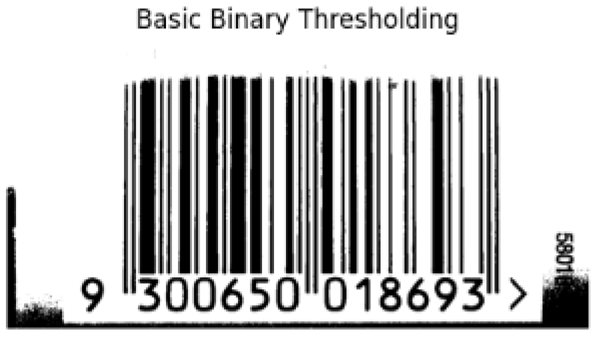
Task 1

Prompt: *Develop a Python program that reads colour images from a designated directory img1.jpg, img2.jpg, etc. For each image, the program should determine if a barcode number is present and, if so, extract the region containing the barcode number. Be aware that some test images may be negative, meaning they do not contain a valid EAN-13 barcode number.*

Initial Ideas

My initial approach/idea was to train an object detection model to recognize the barcode in the images and return a bounding box. This would involve finding / creating a dataset and then creating or customizing an existing object detection model such as yolo. This approach was fine however it showed low success rate when customizing a pre trained model, and while there was plenty of barcode datasets, there weren't any that focused on identifying the digits below the barcode. This approach would also struggle against rotated / transformed images.

The second resort was to use a small pipeline involving image processing such as greyscale / thresholding, as well as feature detection techniques finding edges and lines. This approach showed lots of promise; however, I was unable to make it work consistently on 3d angled images.



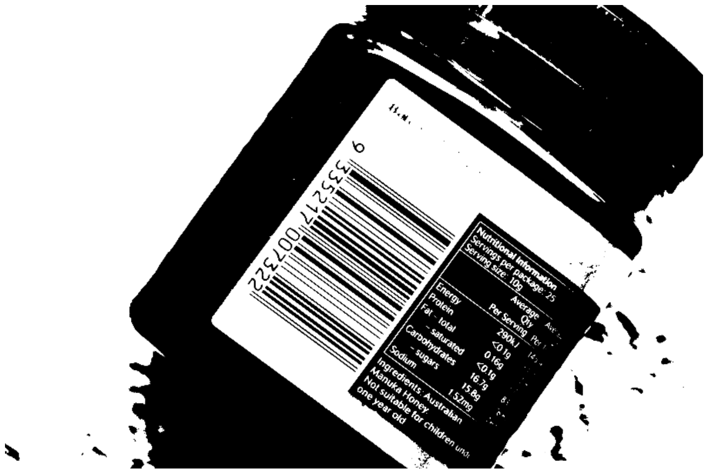
Final Implementation

The final approach was to use another pipeline utilizing OpenCV's barcode detection functionality, which would return a border box/coordinate of the detected barcode. This uses parallel lines with no machine learning similar to our initial approach, however it is much more refined.

This still had many issues that had to be dealt with.

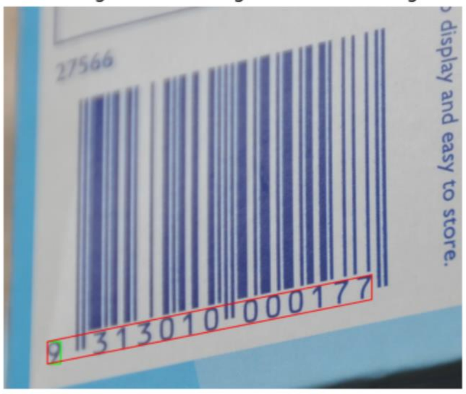
Consistency:

It would often detect false barcodes, and struggle to identify rotated barcodes, as well as sheared angles. This was solved by image processing techniques such as greyscale, gaussian blur, thresholding and adjusting angles for more accurate detection.

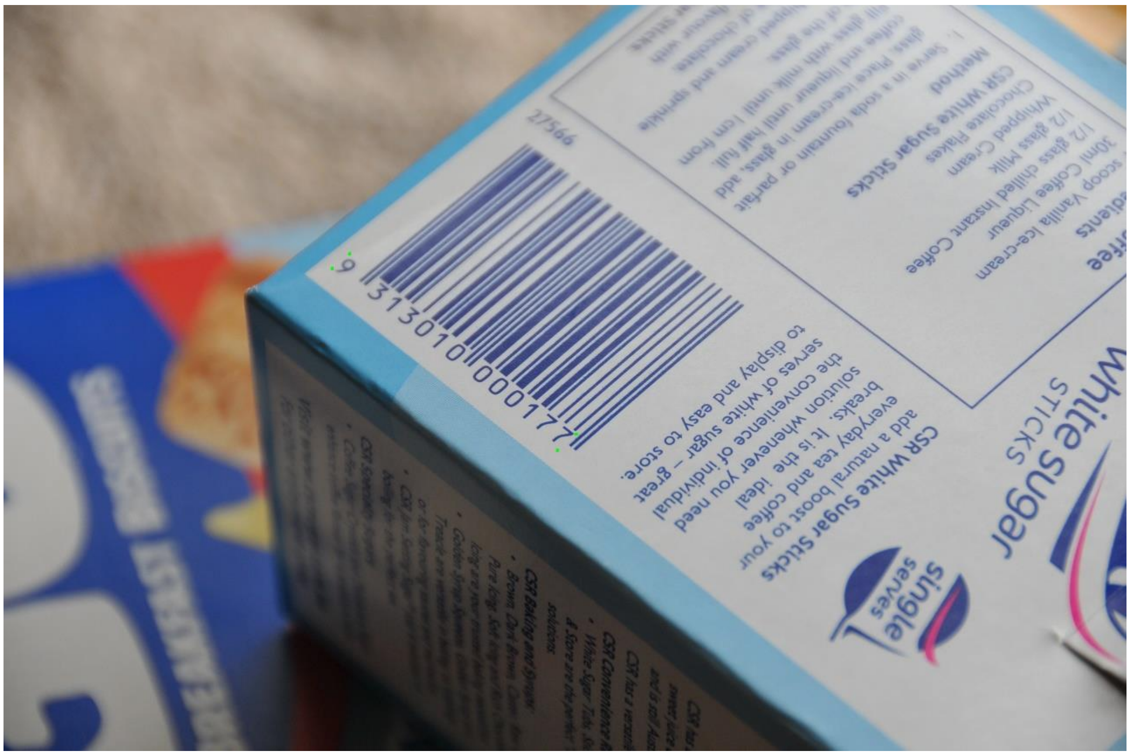


Transformations:

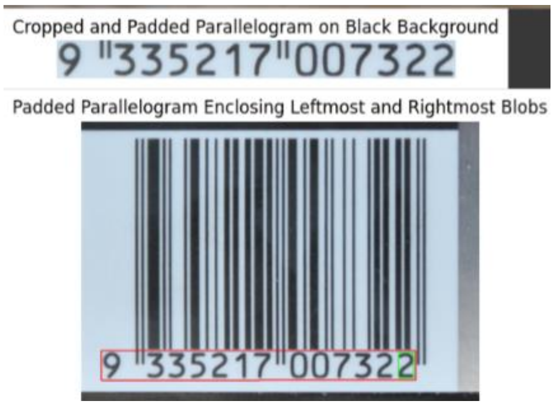
Another issue was finding the correct orientation of the barcode, especially for the slanted case, this was by far the most challenging part of the assignment. I used the first part to get a cropped image of the barcode, which in some cases was still slanted (3d angle). I then used blob detection combined with relative are and aspect ratio filters to identify the possible digits, if a group of digits was found that would be used to find the correct orientation as well as get a cropped digit image.

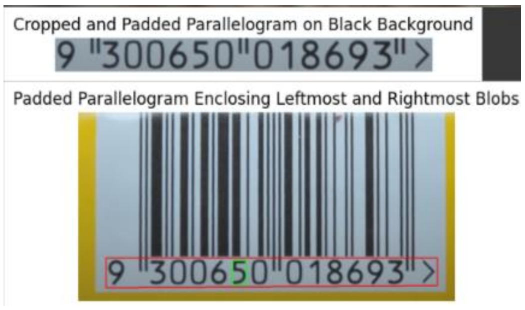


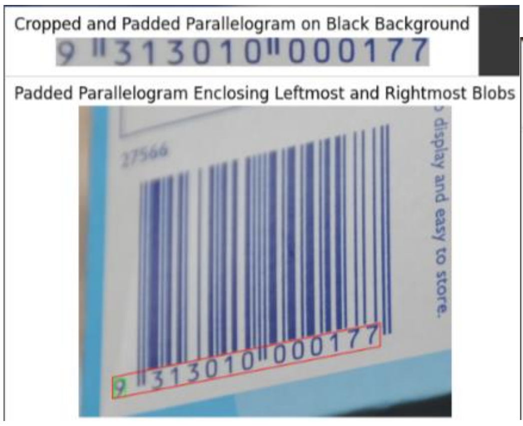
We then added some padding, and had to reverse our cropping and transformations to find the same points on the original image. This was a challenge as the size of the image would alter when it was rotated, which meant we had to account for scaling/offset which was very tricky. (see the green corners below around the digits, hard to spot)

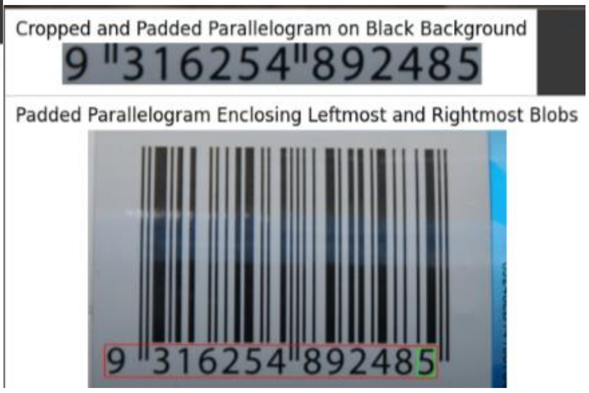


Validation Results

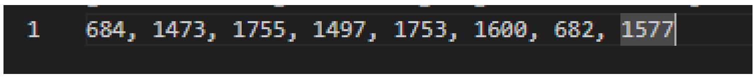

Cropped and Padded Parallelogram on Black Background
9 "335217"007322
Padded Parallelogram Enclosing Leftmost and Rightmost Blobs
9 "335217"007322


Cropped and Padded Parallelogram on Black Background
9 "300650"018693">
Padded Parallelogram Enclosing Leftmost and Rightmost Blobs
9 "300650"018693">


Cropped and Padded Parallelogram on Black Background
9 "313010"000177
Padded Parallelogram Enclosing Leftmost and Rightmost Blobs


Cropped and Padded Parallelogram on Black Background
9 "316254"892485
Padded Parallelogram Enclosing Leftmost and Rightmost Blobs

Here is an example of the coordinates,



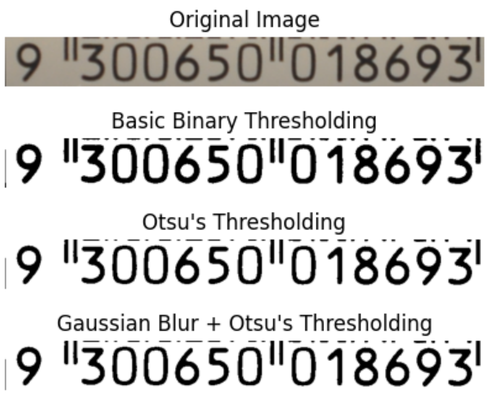
```
684, 1473, 1755, 1497, 1753, 1600, 682, 1577
```

Task 2

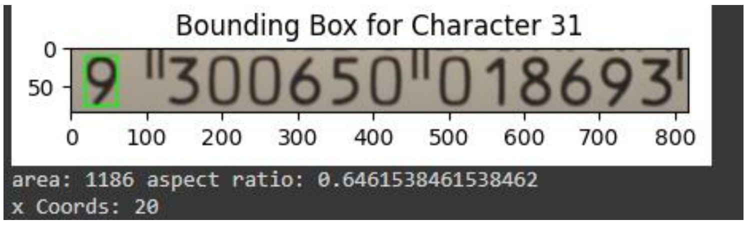
Prompt: *For this task, you are provided with images of the detected regions containing barcode numbers, as identified in Task 1. You may assume that the detected region has been geometrically transformed if required. Write a Python program to extract individual digits as text files d01.txt, d02.txt, etc. that contains their top left and bottom right coordinates x1, y1, x2, y2. (see Fig. 2). Your program must also output the segmented digits as image files d01.png, d*

Implementation

Task 2 was a very simple blob detection case, the only issue was the detection of unwanted blobs, I solved this using an area filter relative the area of the image so resolution wouldn't affect it, as well as an aspect ration filter to fit the digits more closely. Preprocessing was also implemented to make the digits easier to detect, techniques such as grey scale and thresholding were used.


Original Image
Basic Binary Thresholding
Otsu's Thresholding
Gaussian Blur + Otsu's Thresholding

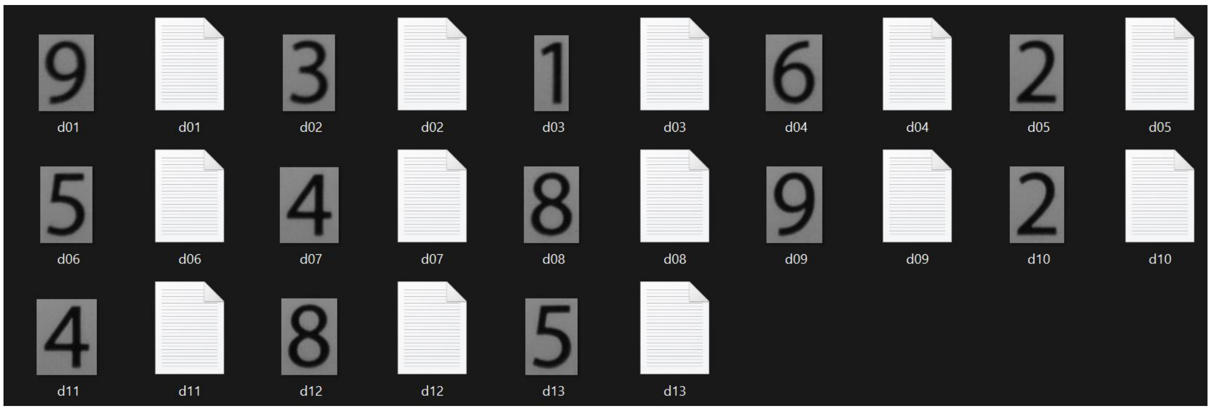
This is an example of how our blob detection worked,

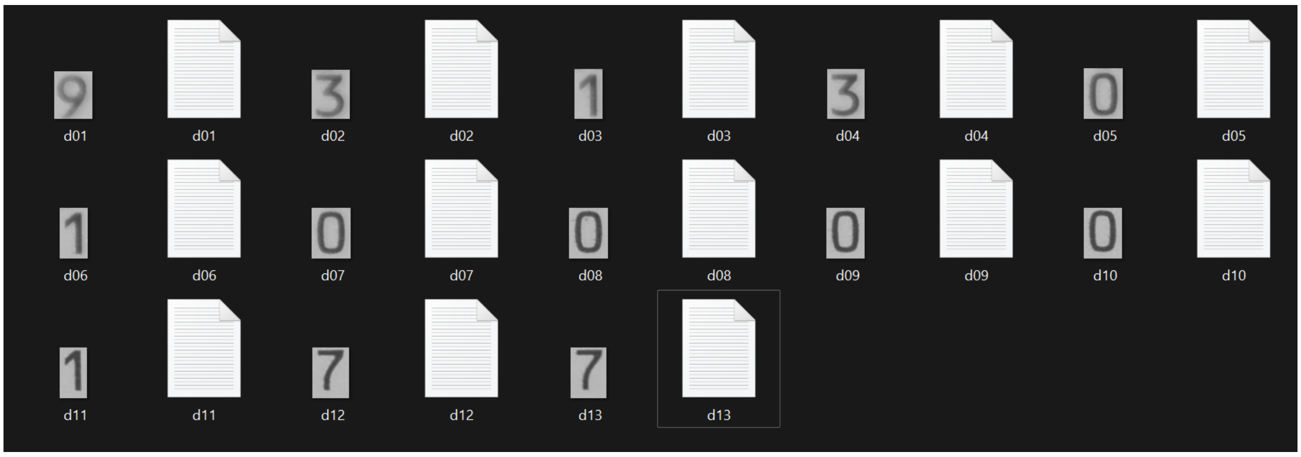


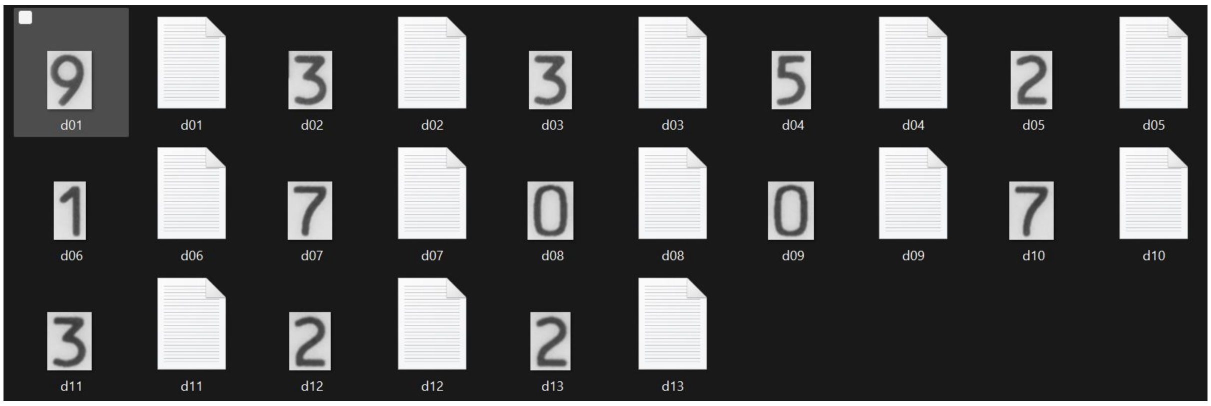
```
area: 1186 aspect ratio: 0.6461538461538462
x Coords: 20
```

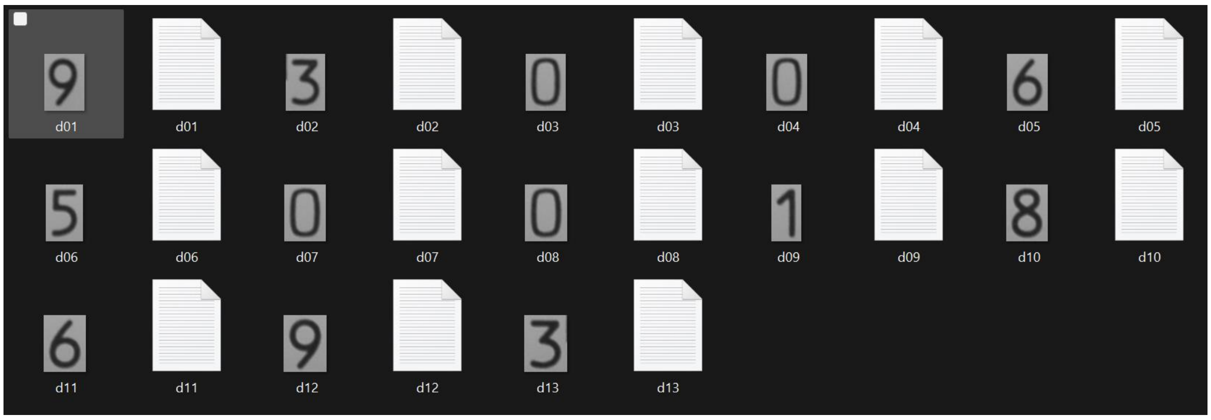
Validation Results

Here is an example of the results for barcode1,3,4,5,







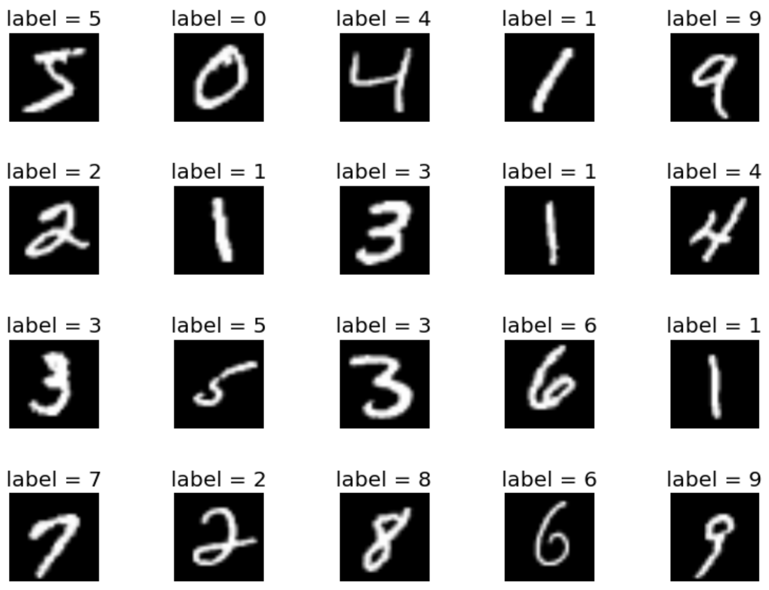


Task 3

Prompt: *In this task, you are given sets of images of individual digits that are found from Task 2. Each set of images corresponds to a barcode number. Write a program that recognises the digits from each set and, for each input digit image, outputs the recognized digit as a text file (see the following figure).*

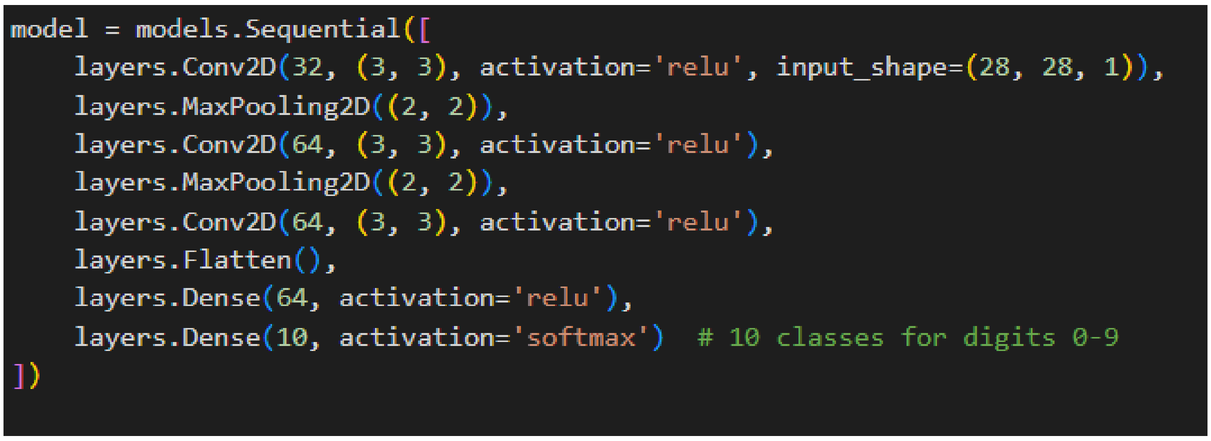
Implementation

My first and final approach for task 3 was using deep learning using a CNN network trained on the MNIST dataset. This involved importing the dataset, splitting the testing and training data and designing a CNN.

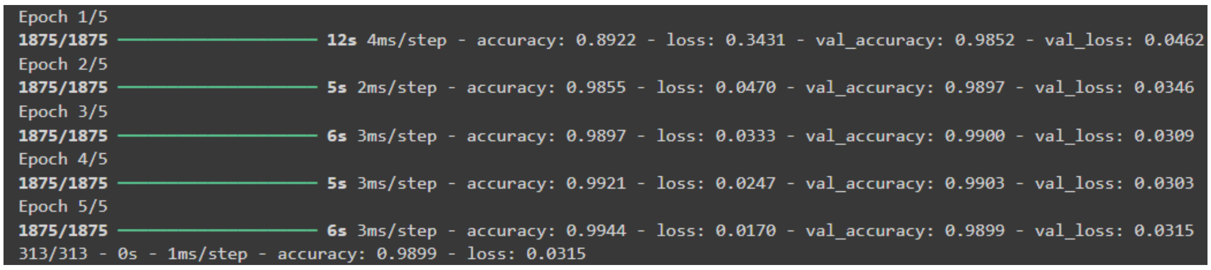


(Google image of MNIST dataset)

Since this problem is trivial and is an elementary step in learning CNN, I found that 3 convolution layers were more than sufficient achieving ~ 99% testing data accuracy.



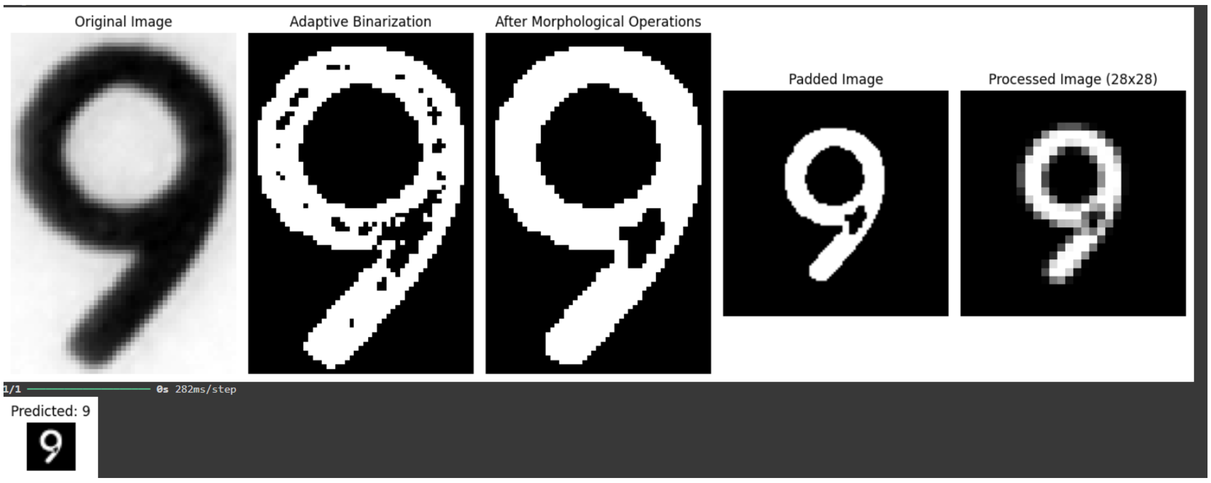
```
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')  # 10 classes for digits 0-9
])
```



```
Epoch 1/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 12s 4ms/step - accuracy: 0.8922 - loss: 0.3431 - val_accuracy: 0.9852 - val_loss: 0.0462
Epoch 2/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - accuracy: 0.9855 - loss: 0.0470 - val_accuracy: 0.9897 - val_loss: 0.0346
Epoch 3/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.9897 - loss: 0.0333 - val_accuracy: 0.9900 - val_loss: 0.0309
Epoch 4/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9921 - loss: 0.0247 - val_accuracy: 0.9903 - val_loss: 0.0303
Epoch 5/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.9944 - loss: 0.0170 - val_accuracy: 0.9899 - val_loss: 0.0315
313/313 - 0s - 1ms/step - accuracy: 0.9899 - loss: 0.0315
```

This unfortunately didn't mean that it worked seamlessly on our cropped digits. This is due to multiple factors such as: the size of the MNIST images, MNIST digits are white with a black background, and they also contain a lot of padding.

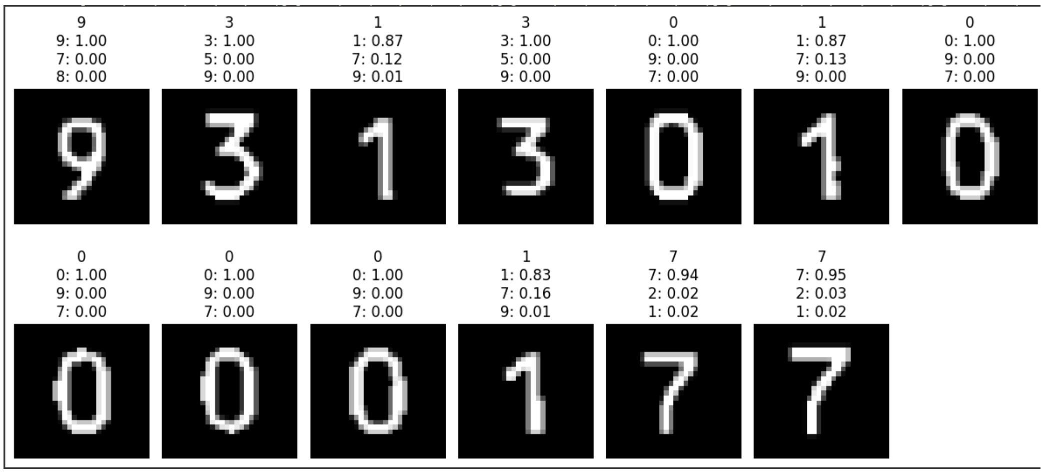
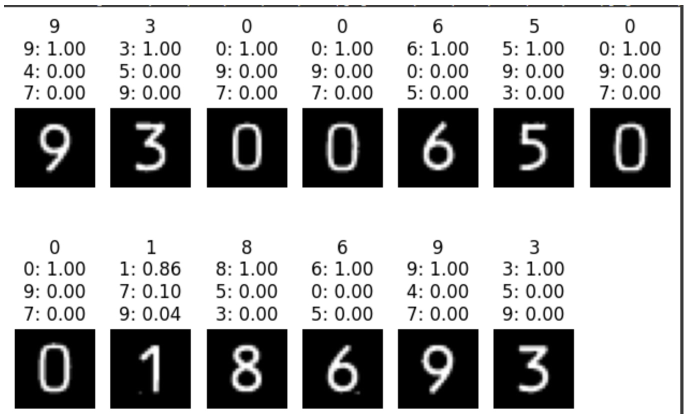
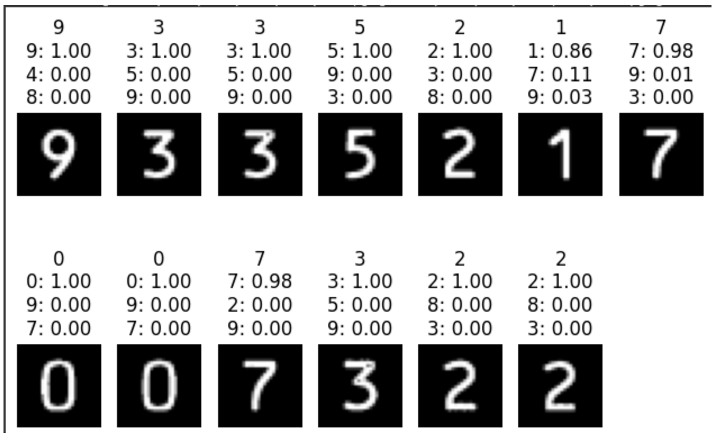
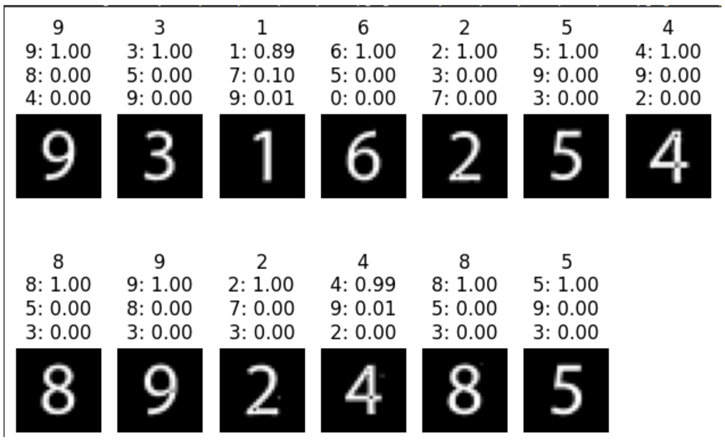
Hence, we had to do some heavy image processing such as inverting the image, thresholding, morphological operations, adding square padding and resizing. This was all done to make our images as similar as possible to the MNIST dataset



```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 282ms/step
Predicted: 9
```

As you can see in the results section, this eliminated most of the errors, however the model struggles the most between similar digits such as 1 and 7, however it is very infrequent that its prediction is wrong, its more likely that the top prediction is correct however not 100% certainty.

Validation Results


9 9: 1.00 7: 0.00 8: 0.00
3 3: 1.00 5: 0.00 9: 0.00
1 1: 0.87 7: 0.12 9: 0.01
3 3: 1.00 5: 0.00 9: 0.00
0 0: 1.00 9: 0.00 7: 0.00
1 1: 0.87 7: 0.13 9: 0.00
0 0: 1.00 9: 0.00 7: 0.00
0 0: 1.00 9: 0.00 7: 0.00
0 0: 1.00 9: 0.00 7: 0.00
0 0: 1.00 9: 0.00 7: 0.00
1 1: 0.83 7: 0.16 9: 0.01
7 7: 0.94 2: 0.02 1: 0.02
7 7: 0.95 2: 0.03 1: 0.02

9 9: 1.00 4: 0.00 7: 0.00
3 3: 1.00 5: 0.00 9: 0.00
0 0: 1.00 9: 0.00 7: 0.00
0 0: 1.00 9: 0.00 7: 0.00
6 6: 1.00 0: 0.00 5: 0.00
5 5: 1.00 9: 0.00 3: 0.00
0 0: 1.00 9: 0.00 7: 0.00
0 0: 1.00 9: 0.00 7: 0.00
1 1: 0.86 7: 0.10 9: 0.04
8 8: 1.00 5: 0.00 3: 0.00
6 6: 1.00 0: 0.00 5: 0.00
9 9: 1.00 4: 0.00 7: 0.00
3 3: 1.00 5: 0.00 9: 0.00

9 9: 1.00 4: 0.00 8: 0.00
3 3: 1.00 5: 0.00 9: 0.00
3 3: 1.00 5: 0.00 9: 0.00
5 5: 1.00 9: 0.00 3: 0.00
2 2: 1.00 3: 0.00 8: 0.00
1 1: 0.86 7: 0.11 9: 0.03
7 7: 0.98 9: 0.01 3: 0.00
0 0: 1.00 9: 0.00 7: 0.00
0 0: 1.00 9: 0.00 7: 0.00
7 7: 0.98 2: 0.00 9: 0.00
3 3: 1.00 5: 0.00 9: 0.00
2 2: 1.00 8: 0.00 3: 0.00
2 2: 1.00 8: 0.00 3: 0.00

9 9: 1.00 8: 0.00 4: 0.00
3 3: 1.00 5: 0.00 9: 0.00
1 1: 0.89 7: 0.10 9: 0.01
6 6: 1.00 5: 0.00 0: 0.00
2 2: 1.00 3: 0.00 7: 0.00
5 5: 1.00 9: 0.00 3: 0.00
4 4: 1.00 9: 0.00 2: 0.00
8 8: 1.00 5: 0.00 3: 0.00
9 9: 1.00 8: 0.00 3: 0.00
2 2: 1.00 7: 0.00 3: 0.00
4 4: 0.99 9: 0.01 2: 0.00
8 8: 1.00 5: 0.00 3: 0.00
5 5: 1.00 9: 0.00 3: 0.00

Task 4

Prompt: *In this task, you complete the perception pipeline by combining the individual programs that you have implemented for the previous tasks. For a given input image • If it is a negative image, your program must produce nothing; • If it is a positive image, your program must produce a single text file containing the barcode number found.*

Task 4 was perhaps the simplest, It was a matter of taking each individual task and connecting their inputs and outputs to complete the pipeline,

Task 1, focused on image processing, rotations / transformations and cropping the area of interest.

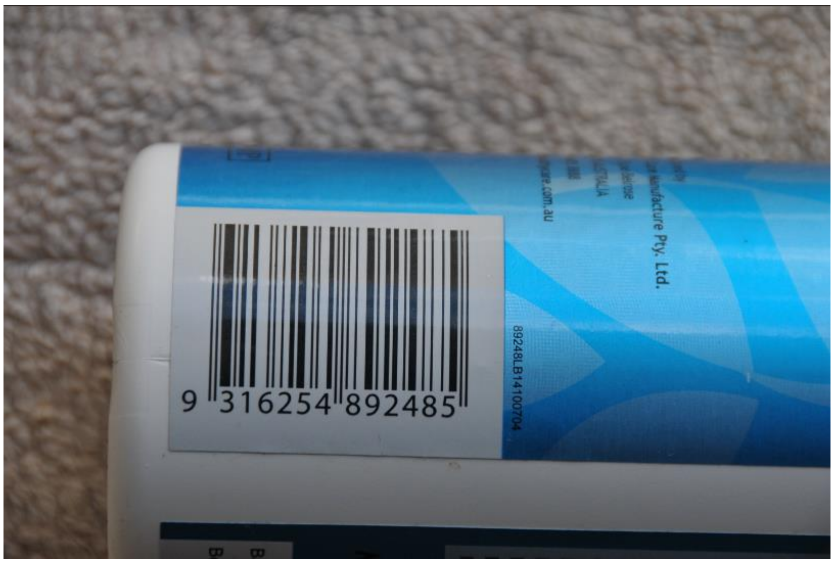
Task 2, focused on feature extraction, extracting the digit blobs.

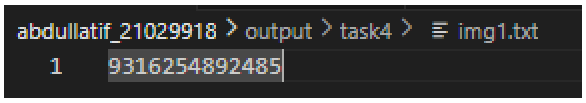
Task 3, focused on object detection / classification, ie. Classifying the individual digits.

Task 4 involved connecting these tasks to form a pipeline.

Validation Results

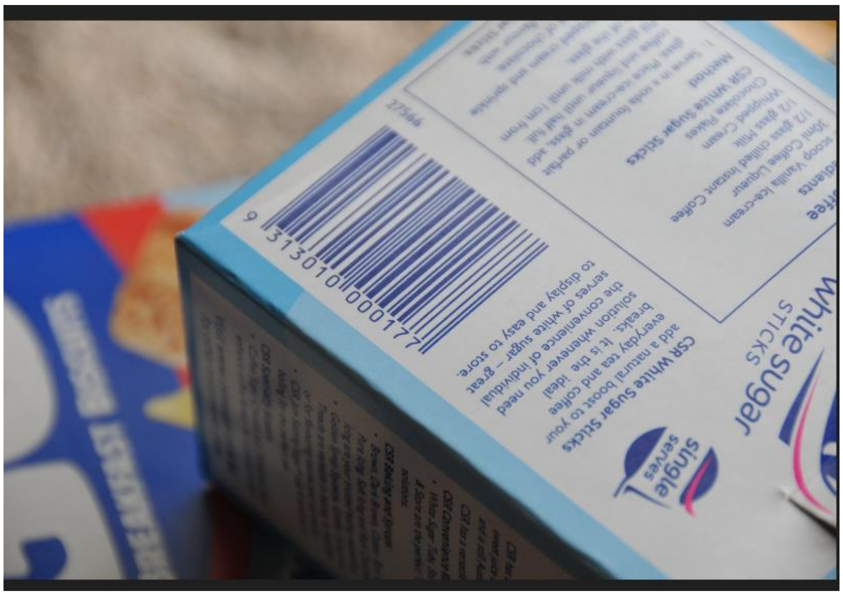
Img1.

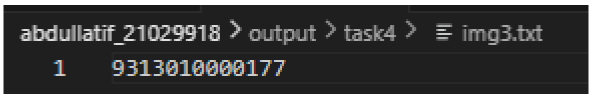




```
abdullatif_21029918 > output > task4 > img1.txt
9316254892485
```

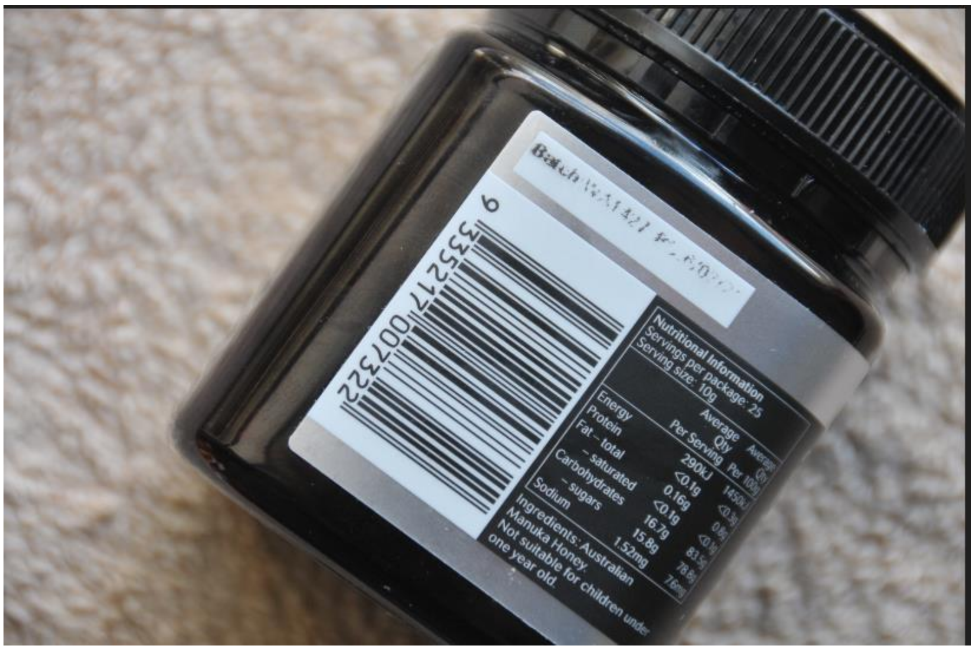
Img3.

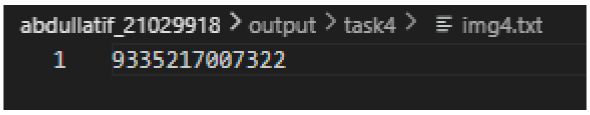




```
abdullatif_21029918 > output > task4 > img3.txt
  1    9313010000177
```

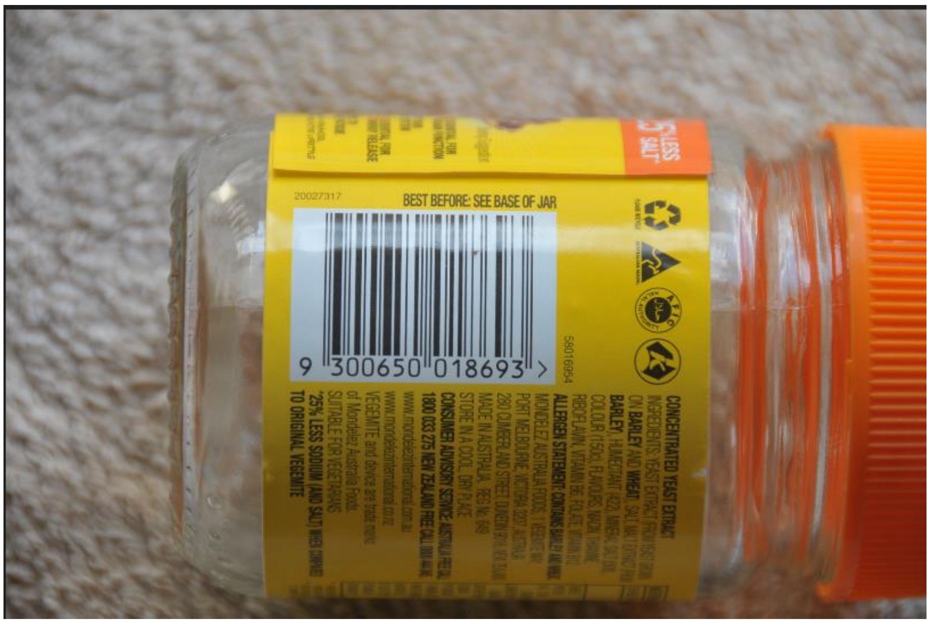
Img4.

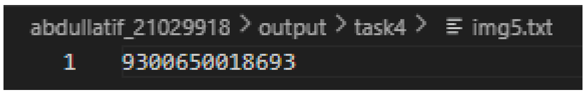




```
abdullatif_21029918 > output > task4 > img4.txt
  1    9335217007322
```

Img5.





```
abdullatif_21029918 > output > task4 > img5.txt
1    9300650018693
```

Conclusion

Overall, this assignment was a useful and good opportunity to implement and practice what we have learnt in this unit in a semi-large project. I was also able to learn and expand on my knowledge a lot more through the practice and research required for this assignment. Our application meets all criteria for the validation images as well as extra images, for all tasks 1 to 4.

References

- Lecture and practical materials – comp3007
- https://pyimagesearch.com/2021/01/20/opencv-getting-and-setting-pixels/
- https://www.geeksforgeeks.org/detect-and-read-barcodes-with-opencv-in-python/
- https://www.geeksforgeeks.org/python-os-path-join-method/
- https://www.tensorflow.org/datasets/catalog/mnist