Extracting Information from Commercial Aircrafts

Ege Cakmak
cakmake@my.yorku.ca
York University - Lassonde School of Engineering (EECS)
Toronto, Ontario, Canada

Figure 1: Turkish Airlines Boeing 777-300ER. Image from https://www.jetphotos.com/photo/9408240

ABSTRACT

With the enhancement of technology, airline flights have been becoming more common every day, increasing the number of airplanes being used by the mankind. I am proposing a way to identify airplanes using Computer Vision techniques in this paper.

KEYWORDS

computer vision, airplane identification, text detection, text recognition, deep learning, ocr

1 INTRODUCTION

As mentioned in the abstract section, airplanes have been becoming very common over the past years. Therefore, identifying airplanes have become a challenge that could be beneficial to the society if solved.

In this paper, my aim is to extract as much as information as possible from large body commercial and cargo aircrafts such as Boeing 787-9 Dreamliner or Boeing 747-400F using the information existing on the body of the aircrafts. The two main sources of information that I will be looking for in images in this paper are as follows.

- `Livery`: Text that gives information about the operator of an aircraft.
- `Registration Code`: A combination of letters and digits to identify an aircraft uniquely. Different countries can have different registration code formats.

In addition, there are also two main variable conditions in these images that we need to account for. These are as follows.

- `Viewing Angle`: The viewing angle the images are taken will cause the registration code to look warped which will decrease the OCR engine's accuracy severely.

- `Angle of the aircraft to the ground`: The angle of the aircraft relative to the ground will also affect OCR engine's performance.

Using the aircraft images available online, I aim to attempt implementing a computer vision pipeline to extract information from them using as little deep learning as possible to increase the learning outcomes. My method is written in Python and utilizes OpenCV and many other software that will be mentioned throughout this paper.

2 PRIOR WORK

There is one prior work that is highly related to my project written by Mekonnen [2] in 2017. In the paper, author attempts to identify substantially smaller aircrafts based solely on their registration codes using OCR techniques and promises 85% accuracy. Unfortunately, author did not publish any source code, but the paper is available online.

3 MOTIVATION

Although I have no specific motivation for choosing this project, the reason why I decided to work on this project is mostly because of my interest in aviation. And also, I thought that there would be many interesting Computer Vision challenges that I would need to solve while working on this project, which would increase my learning outcomes.

Further, although not very reliable, this method I am proposing can be used to create datasets for registration codes of airplanes from real life images for deep learning. Moreover, this method might become beneficial in cases where other ways of identifying airplanes are absent and the only way to gain information about the airplane is visually. For instance, if we have an image of a commercial airplane with no other information supplied then the only way we can obtain information about the aircraft is simply by looking at it. My aim is to automate this process.

4 METHODOLOGY

My methodology involves several stages. In summary, I acquired images of large body commercial aircrafts where their livery and registration code is visible, from web sites like JetPhotos. After that, I ran Mask R-CNN on these images to filter objects other than the airplane in context. Having the filtered image, I used a non deep learning method to detect the location of the registration code of the aircraft in the image. Finally, I ran an OCR engine on the detected location and the whole body of the aircraft to recognize the registration code and the airline livery respectively. There are also some other steps that were taken to improve the accuracy of the results such as preprocessing the image for better OCR performance and filtering out the results from the OCR engine. These will also be explained in detail in the following sections.

In this paper, I aimed for maximizing the accuracy of my method at the cost of extending the time required for my application to run and also tried to use the least amount deep learning possible to increase my learning outcomes.

4.1 Gathering Images

For this part, I simply went to JetPhotos.com and looked for images of commercial aircrafts taken from the side. The kind of images I was looking for was similar to the one in Figure 1.

4.2 Segmenting Images

As mentioned before, I segmented the images I gathered to filter out unwanted details or objects. To do so, I used a Python software that implements Mask R-CNN on Python using TensorFlow and Keras. This software [**?**] also includes the Mask R-CNN model that is trained on the MS-COCO dataset which includes a variety of objects including airplanes. Therefore I did not have to train/re-train a Mask R-CNN model.

By making a couple of simple calls and little tweaks to the author's code, I was able to segment out the airplanes from the images using this software. Additionally, Mask R-CNN also provided the bounding box and mask of the airplane as a separate list of lists, which greatly helped me in the next sections. Figure 2 is an example output from this software after running it on the image in Figure 1.

Figure 2: Turkish Airlines Boeing 777-300ER Segmented

4.3 Detecting the Registration Code

Once we have the segmented image, we are ready to detect the possible bounding boxes for the registration code of the airplane.

To locate the registration code, I detected all text-like objects on the airplane without using any deep learning models (i.e. text detectors). To do so, I used OpenCV's threshold function to threshold the image and dilated it several times. At each dilated instance, I found all the contours and the smallest bounding boxes around them. More detail on this part will be given in the following subsections.

4.3.1 Separating Channels of the Image. Before proceeding with thresholding the image, I had to separate the channels of the RGB image since the color of the text or its background's color can be different, and thresholding only the RGB image itself might miss some of the bounding boxes in some cases.

4.3.2 Thresholding Each Channel of The Image. Once I had the channels of the image, I thresholded each channel using OpenCV's threshold function. When doing thresholding we also need to decide

whether we are doing a binary thresholding or a binary inverted thresholding. However, since thresholding is a computationally cheap operation, I simply thresholded for both cases.

The reason why we need to watch for both cases is follows. Once we split the RGB image into its channels, we get a grayscale image for each channel. For example, if we look at Figure 2, we can see that the airplane we have is generally white and the registration code is darker than the color of the airplane. But in the thresholded image, the text will appear black. However, in the coming steps we will be leveraging morphology, therefore we require the text to be white since it is the part of the image we are interested in. Due to this, we simply tell the thresholding function to apply an inverted binary thresholding in these cases. For the other cases, where the background of the airplane is darker and registration code's color is lighter, then we can tell the thresholding function to apply a regular binary thresholding. However, keep in mind that we can not analyze the color of the airplane since the body of it can include a variety of colors and also, we do not know the background color of the registration code. For instance, the body of the airplane might be white in general but the registration code might have a blue background which would ultimately cause our code to fail.

4.3.3 Applying Morphology on the Thresholded Images. Once we have the thresholded images, we would like to get all the white spots in these thresholded images, get the bounding boxes around the white spots and filter for the bounding boxes that potentially have the registration code. But if we try to find the bounding boxes now it will not work as each white spot likely references to a character of the registration code. Thus, we dilate all of the white spots several times to make the white spots referencing to the characters of the registration code to become a one whole chunk. But we do not know when this one whole chunk will appear. As a result of that we get all the candidate bounding boxes every time we dilate the thresholded image.

4.3.4 Finding the Contours around the White Spots. Every time we would like to get all the possible bounding boxes, we draw the contours around the white spots and get the smallest rectangles around these spots. However if we follow this path, it is likely we will get a lot of candidates. That is why we adopt a heuristic here. Since we know the size of the airplane, we can estimate the aspect ratio of the bounding box that has the registration code. Thus, we can simply get all the candidate bounding boxes that have a similar aspect ratio to the relative aspect ratio we can calculate from the size of the mask. Once we have all the candidate bounding boxes we can simply record the coordinates for these bounding boxes to an array and return this array to use these coordinates later on.

4.4 Running OCR on the Candidate Bounding Boxes

In this section, we simply run OCR on the bounding boxes we find in the previous section. But one might ask that how we know which bounding box has the registration code. We actually DO NOT know which bounding box has the registration code. That is why we run OCR on all of the bounding boxes and filter out the results.

For the OCR we are using Tesseract to recognize any text in these bounding boxes. Unfortunately, Tesseract performs very variably

Figure 3: Segmented Image with 1 Iteration of Dilation

Figure 4: Segmented Image with 2 Iterations of Dilation

under different conditions. For example, when the airplane has some angle relative to the ground it might fail. In Figure 2 the plane has some angle relative to the floor and ultimately the registration code has an angle as well. Further, if the viewing angle the image is taken from is not facing the airplane straight from one of its sides then the texts on the airplane might seem warped and ultimately cause Tesseract to fail. If you look carefully in Figure 5 you will see that the registration code on the airplane looks warped which is an issue we need to overcome.

For the first issue above, we can easily find the skew of the text in each bounding box and rotate the bounding box accordingly. And, for the second one, we need to make a Four Point Transformation (i.e. Warping) in the area that covers the bounding box to get a bird's eye view of the bounding box. Both of these operations will be explained in detail in the following subsections.

Also, it is worth note it here that in addition to two of the above operations we will also be normalizing the contrast of the area in the bounding box and enlarging it to improve performance of Tesseract.

4.4.1 Normalizing Contrast and Enlarging the Image of The Bounding Box. Before proceeding with the more complicated preprocessing operations we apply some trivial augmentations on the image.

We first crop the original image for the bounding box we have and enlarge it using interpolation. By using interpolation we minimize the loss of details. After resizing the image, since we know both the global and local maximas and minimas, we normalize the contrast of each pixel in the cropped image. The difference before and after these augmentations can be clearly seen in Figures 5 and 6.

Figure 5: An Example Registration Code Bounding Box After Enlarging and Normalizing Contrast

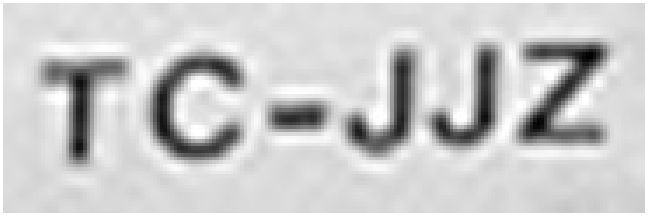

Figure 6: An Example Registration Code Bounding Box After Enlarging and Normalizing Contrast

4.4.2 Aligning the Image Horizontally. As we mentioned, our text might have some angle relative to the ground. To overcome this issue, we can rotate the cropped image to align horizontally. To do so, we can once again threshold the cropped image, find the bounding box around the text, find the angle of the bounding box and rotate the cropped image accordingly. But, one might ask how do I know the thresholding mode I need to choose. In this case, since we cropped the original image, we can assume the background is uniform. Hence, we can do a very simple color analysis by adding all pixels' intensity values' that range from 0 to 255 and find the average intensity per pixel for the cropped image by dividing the sum with the number of pixels in the cropped image. If this average is closer to 255 then we can assume that the background is white and apply an inverted thresholding and vice versa. By the way we are also doing Otsu's thresholding along with whatever thresholding mode we choose for improved performance.

Once we threshold the image, we can find the smallest bounding box of the text, find this bounding box's angle and rotate the image accordingly. In Figures 7 and 8, the difference between before and after the operation can be observed.

4.4.3 Warping the Image. The example images we used previously do not suffer from the viewing angle issue, therefore for demonstration purposes we will be using a different image.

After applying all of the preprocessing steps in the previous steps, we can see in Figure 9 that the registration code looks warped due to the viewing angle, which causes the performance of Tesseract to drop greatly.

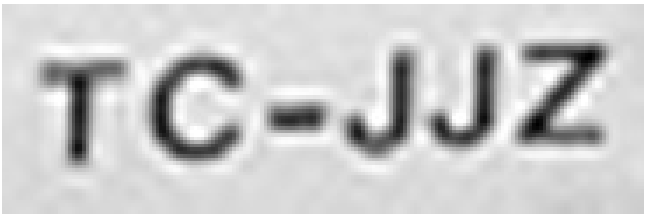

Figure 7: An Example Registration Code Bounding Box Before Rotation

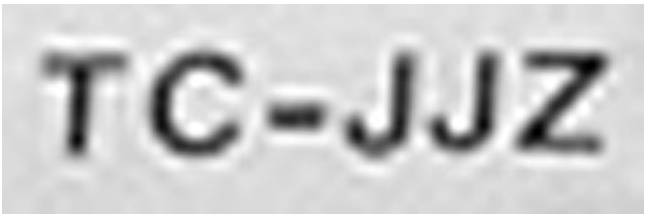

Figure 8: An Example Registration Code Bounding Box After Rotation

To mitigate this issue, we need to do a Four Point Transformation operation to get a bird's eye view of the image.However for that we need to manually find a rhomboidal bounding box of the text so that the Four Point Transformation operation gives satisfactory results. But, this is against my promise since I promised an automatic method. Therefore, I adopted a strategy to find those points we need automatically.

To find the new rhomboidal bounding box's corners' coordinates, we need to know about the viewing angle which we do not. But, the level of italic-ness of the text can say something about it. To measure the italic-ness of the text we threshold it just like in the previous subsection and find the edges in the image using Canny Edge Detector. Once we have the edges, we can use Hough Transformation to find the lines in the image. The most steep ones among all the Hough Lines are going to be going over the edges of the characters of the registration code and will have the same angle as them. Figure 11 demonstrates these lines. One might notice that not all the lines have the same angle. We can mitigate this issue by finding the average angle.

Once we find the angle, finding the distance we need to shift the bottom points to get the rhomboidal reduces into a trigonometry problem. What we know so far is the following.

- `Opposite Line's Length`: This is equal to the bounding box's height.
- `Angle of the Edge to the Horizontal` : We just found this.

Using these and looking at Figure 12, we can clearly see that we would like to solve for the adjacent line's length which is equal to the shift distance. This can be done easily as follows.

$$\frac{opposite}{adjacent} = tan\alpha = \frac{bbheight}{sd} \text{ Solve for sd.} \quad (1)$$

$$\frac{bbheight}{tan\alpha} = sd \tag{2}$$

Once we find the shift distance, we can now calculate the new rhomboidal bounding box's bottom points and apply the Four Point Transformation on Figure 9 which will yield Figure 10.

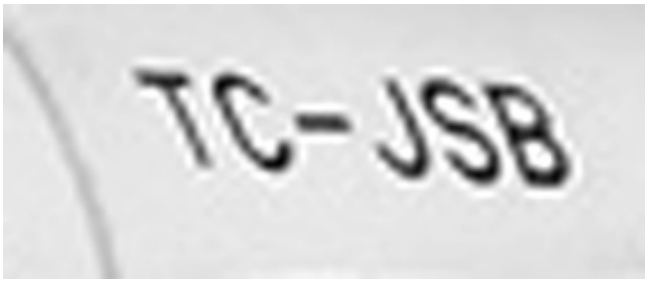

Figure 9: An Example Registration Code Bounding Box Before Warping

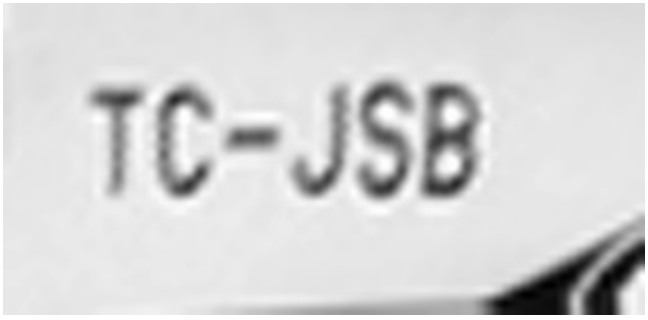

Figure 10: An Example Registration Code Bounding Box After Warping

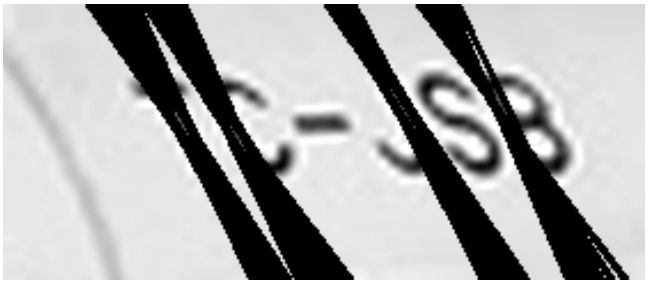

Figure 11: An Example Registration Code Bounding Box During Warping with The Hough Lines

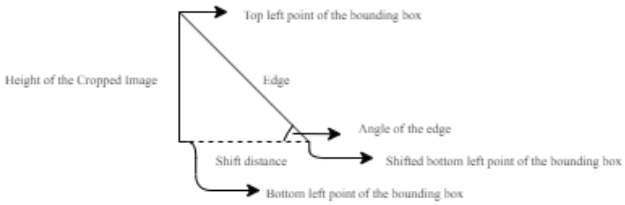

Figure 12: Visual Explanation for Finding The Shift Distance

4.4.4 Filtering OCR Results. Once we are done with all the preprocessing steps, we can finally run our OCR engine on the bounding box. After running the OCR engine on all candidate bounding boxes, we end up with chunks of strings from different parts of the airplane. We need to estimate the result that is likely the registration code. In this part of the paper, I will be explaining my method for filtering the results.

In the first step of filtering, we run a regular expression (regex) on each possible result. Our regular expression looks for two cases. This step is followed by a simple filter that filters out possible (no longer) results with a length less than 3.

- `Case 1`: Text consist of only alphanumeric characters. Example : N1881B
- `Case 2`: Text consist of only alphanumeric characters with a dash in between any of its letters. Example : TC-INF

After the above filtering operation, we go over the results once again, create a bucket for each word, and count the number of times each word is detected. But Tesseract unfortunately might read dashes multiple times. Therefore, before putting each word into its bucket we check whether it consists of any dashes, and if that is the case we clean the text by grouping by characters except for the dashes using regex and rebuild the registration code.

Finally, we sort these buckets decreasingly by the number of times each word occurs. However, we give priority to possible results including dashes as most airplane registration codes include dashes.

4.5 Detecting the Airline

To detect the airline, we can run Tesseract on the whole body of the aircraft in a mode that allows it to collect as much as text possible. These modes can be set using a config variable. Starting from an image like in Figure 2, we first need to crop our picture to only include the aircraft with least amount of background as possible to improve Tesseract's performance. Since we already have the mask for this image, we can push the top and bottom boundaries closer to the airplane iteratively and at each iteration we can check how many pixels each boundary intersects with. Since we do not need the tail, we can even cut the tail of the airplane as well to improve our chances. After squeezing our image like a sandwich we get an image like in Figure 13.

We are now ready to run Tesseract on this image. But we are not done, as the results we get are likely to have mistakes. To solve this problem, we go through each word detected by the OCR,

Figure 13: Figure 2 Cropped for OCR

append each to a string and do a comparison between this string and every word of a list of airline names scraped from the internet by calculating Levenshtein distance using fuzzywuzzy.

I was able to find all the existing airline names in the World on Wikipedia and scrape them into a text file using a Python script I wrote. This script can be accessed publicly at the following link.

https://github.com/egecakmak/Wikipedia-Airline-Names-Scraper

5 RESULTS

To evaluate the performance of my model, I gathered 71 random images from JetPhotos. These images were fed to my Python script and the results can be seen in Table 1. Below are the statistics of the results.

- `Total Images`: 71
- `Registration Code Bounding Boxes Detected Correctly`: 45 (%63.4)
- `Registration Codes Detected Correctly`: 19 (%26.8)
- `Registration Codes Detected Partially`: 8 (%11.3)
- `Airlines Detected Correctly`: 12 (%17.0)
- `Airlines Detected Partially`: 2 (%0.03)

Since this set of images were gathered randomly, conditions such as the viewing angle, livery colors, angle of the airplane, position of the airplane and quality of the images vary greatly. As a result, the results on these set of images unfortunately are not close to what I was aiming for. But as I stated in my proposal, my model is targeted for cases where the viewer is viewing the airplane from exactly the sides of it and also there are images in this set where my model would fail even if it worked flawlessly due to reasons like airline name being in Cyrillic alphabet or obstructed by the wings of the airplane. I believe that this set of images are too difficult for my model. That's why I decided to pick an easier and smaller set of images which I intended to work on while writing the proposal for this project. This set of images consists of 22 images and the results can be observed in Table 2. Below are the statistics of the results.

- `Total Images`: 22
- `Registration Code Bounding Boxes Detected Correctly`: 19 (%86.4)
- `Registration Codes Detected Correctly`: 12 (%55.5)
- `Registration Codes Detected Partially`: 4 (%33.3)
- `Airlines Detected Correctly`: 13 (%59.1)
- `Airlines Detected Partially`: 2 (%9.1)

Comparing the results from both sets of images, we can see that all of our statistics have substantially increased. This is because the images I picked for the second dataset is relatively easier as I mentioned earlier. However, I believe that even these results are not good enough and I will be discussing regarding this issue in the Conclusion section. But before we proceed, I would like to share some of my findings while examining both results and the images.

- Warping (Four Point Transformation) while preprocessing the image actually works. For example I noticed that Tesseract does not recognize the registration code in some images where there is a viewing angle and no warping applied. But after applying the warping, Tesseract can accurately recognize the registration code. The same applies for rotating as well.
- I noticed that in some cases deep learning based text recognizers do not detect the bounding box for the registration code whereas my model detects it.
- It appears that the performance of my model can be improved by tweaking my model to work better with different cases/airlines.

6 LIMITATIONS, NOTES AND CONCLUSION

My model unfortunately suffers from many limitations at the moment. For instance, when the registration code has a similar intensity with its background it might not be detected since we are using thresholding to detect the texts on the airplane. Though this can probably be fixed by tweaking our thresholds, it is likely there will be other cases my model still fails for the same reason. It is simply unfeasible to tweak my model to handle all cases and doing so would take a lot of time which I am lacking at the moment. Figure 14 is an example where my model fails to detect the registration code.

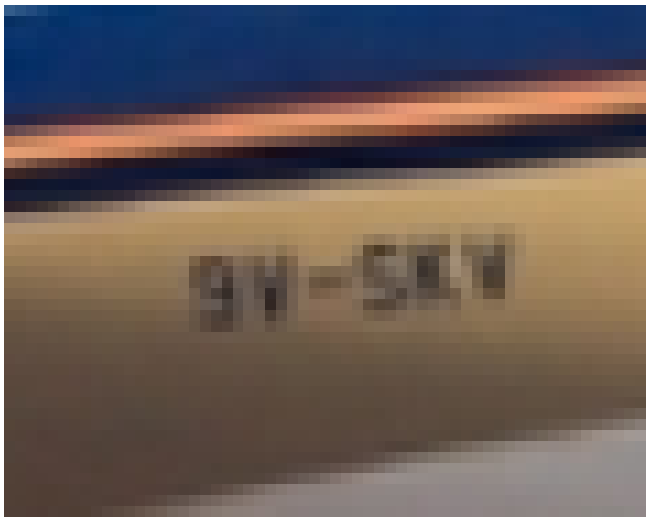

Figure 14: A Bad Example Where Model Fails to Detect the Registration Code

Although, I was able to find multiple cases where my model fails, it is worth to note that it is possible that my model will have issues if the registration code is too close to the other parts of the airplane/livery as it might get mixed with them during dilation.

In conclusion, I have proposed a model for extracting the airline name and the registration code from images of large body commercial airplanes. For this project, although I was able to develop a model that works to some extent, I still believe that my model is not a complete solution. However if we check the results carefully,

Image	RCBBDC	RCDC	ADC	Notes
11332_1574980271	Y	Y	N	-
14761_1574975177	N	N	N	-
17012_1574951483	N	N	N	-
18739_1574963803	Y	N	N	-
22160_1574955851	N	N	N	-
22998_1574995037	Y	Y	N	-
23228_1574983078	Y	N	N	-
23617_1574954063	Y	Y	Y	-
24698_1574980176	N	N	N	-
25271_1574992887	Y	P	Y	1*
30066_1574969277	Y	N	N	-
30112_1575003884	Y	N	N	-
30624_1575026851	Y	P	P	1*2*
31028_1574971669	Y	Y	N	3*
32084_1575883349	N	N	N	-
33561_1574986372	N	N	N	-
34728_1574974306	N	N	N	-
34829_1575884113	Y	P	Y	4*
36132_1574981468	N	N	N	5*
36353_1574970634	Y	Y	N	-
37058_1574974173	N	N	Y	-
40959_1575015324	Y	N	Y	-
42396_1574989620	Y	Y	N	-
44406_1575015266	Y	Y	Y	-
44982_1574994392	N	N	N	-
45084_1574972924	N	N	N	-
46816_1574983007	N	N	N	-
48847_1574963205	Y	P	N	3*
49273_1575890667	Y	P	N	6*
49695_1574977889	Y	N	N	-
50951_1574981490	Y	P	N	7*
50957_1574958989	Y	N	N	-
51383_1574969703	Y	Y	Y	-
53160_1574961350	Y	N	N	-
55868_1575003510	N	N	N	-
56589_1574986039	N	N	N	-
58155_1575861997	Y	Y	N	-
58884_1575886750	N	N	N	-
59226_1574969480	Y	Y	N	9*
60177_1574980916	Y	Y	Y	-
60249_1575004425	N	N	N	-
60469_1574984712	Y	P	N	9* 10*
61555_1574951802	N	N	N	-
64759_1575002226	Y	Y	Y	-
66165_1575001836	N	N	N	-
67882_1574957560	N	N	N	-
68152_1574976430	N	N	N	-
68213_1574969850	Y	Y	N	-
68650_1574981315	N	N	N	-
70337_1575920385	Y	Y	N	-
70443_1574950353	N	N	N	-
73286_1574976599	Y	Y	N	-
73416_1575029568	Y	Y	N	-
73703_1574974331	Y	P	N	8*
73715_1574955122	Y	Y	N	-
75090_1574959399	Y	N	N	-
75943_1574978226	Y	N	N	-
77751_1574974727	Y	N	N	-
85390_1574970338	Y	Y	Y	-
86813_1574953927	N	N	N	-
87100_1574959319	Y	P	P	10* 11*
90608_1574963790	Y	Y	Y	-
91604_1574985871	N	N	N	-
92127_1574955955	Y	N	N	-
92134_1574975478	N	N	N	-
92615_1574989442	Y	N	N	-
93833_1574950344	Y	N	N	-
96405_1574974896	Y	P	Y	12*
97289_1574999392	Y	P	N	13*
97474_1574959358	Y	N	N	-

Table 1: Results on Imageset 1

ABBREVIATIONS

- RCBBDC: Registration Code Bounding Box Detected Correctly
- RCDC: Registration Code Detected Correctly
- ADC: Airline Detected Correctly

TABLE 1 NOTES

- 1*: Registration code missing only 1 character.
- 2*: The livery in this image causes the result to be incorrect. Checking the image, it can be seen that the result is biased because of the livery.
- 3*: Only one character is incorrect in the registration code.
- 4*: Registration code missing 2 characters.
- 5*: Image is from 1988 and livery is not in latin characters.
- 6*: The correct result exists in the result list but partitioned into two pieces.
- 7*: Registration code has 3 incorrect characters.
- 8*: Registration code has 1 incorrect character.
- 9*: Airline livery partially visible.
- 10*: Registration code has 2 incorrect characters.
- 11*: Airline name is included in the candidate registration codes list.
- 12*: The correct result exists in the result list but partitioned into multiple pieces.
- 13*: Registration code has 1 extra character.

we can see that in most cases my model was able to locate the registration code but was unable to recognize the text within the bounding box and also the same applies for extracting the airline name. This is because of unreliability of Tesseract under different conditions in the wild. Therefore, I believe that by retraining an existing neural network specifically for reading registration codes and airline names, and using that instead of Tesseract in this model will substantially improve the performance of my model. But I opted not to do that to increase my learning outcomes of using OpenCV and also because of lack of time to collect a dataset and train a neural network. But again, I believe that this model performs even better than I thought it would considering this is for a class project,

Image	RCBBDC	RCDC	ADC	Notes
13196_1553794497	Y	P	Y	1*
14461_1575055895	Y	Y	Y	-
26045_1575042399	Y	N	Y	-
28637_1574944051	Y	N	N	-
30111_1574913578	Y	Y	Y	-
30237_1575057328	P	N	P	2*
30276_1574959145	Y	Y	N	-
31888_1574896207	Y	P	N	3*
33917_1575057088	Y	Y	N	-
39978_1575050003	N	N	N	-
43895_1574946300	Y	P	Y	3*
45051_1574954216	Y	Y	P	4*
48446_1575010345	Y	N	N	-
56441_1553801892	Y	Y	N	-
70734_1575048120	N	N	Y	-
73967_1575008747	Y	Y	Y	-
95350_1574895634	Y	P	Y	5*
aircanada	Y	Y	Y	-
lufthansa	Y	Y	Y	-
tc-jjz	Y	Y	Y	-
tc-jsb	Y	Y	Y	-
ukraine	Y	Y	Y	-

Table 2: Results on Imageset 2

ABBREVIATIONS

- RCBBDC: Registration Code Bounding Box Detected Correctly
- RCDC: Registration Code Detected Correctly
- ADC: Airline Detected Correctly

TABLE 2 NOTES

- 1*: Only the first 3 characters of the registration code are correct.
- 2*: Airline name was detected as the registration code.
- 3*: Only 1 character of the registration code is incorrect.
- 4*: Airline name is included in the candidate registration codes list.

and could have been improved even further if there were no time restrictions.

7 ACKNOWLEDGEMENTS

Many thanks to my professor Calden Wloka for his guidance and help on tackling the issues I have encountered throughout this project.

REFERENCES

[1] Automated Aircraft Identification by Machine Vision (2017) Helsinki Metropolia University of Applied Sciences, Theseus.fi, https://www.theseus.fi/bitstream/handle/10024/124569/Mekonnen_Israel.pdf.

[2] Mask R-CNN for object detection and instance segmentation on Keras and TensorFlow (2017), Waleed Abdulla, https://github.com/matterport/Mask_RCNN.