Helping the Visually Impaired Navigate at Bus Stops

Tang Yu Han Brandon

1Temasek Junior College, 22 Bedok South Rd, Singapore 469278

Abstract

The Land Transport Authority’s (LTA) “Land Transport Master Plan 2040”, outlines the aim of “Transport for All” as LTA plans to reduce the barriers of transport that different groups of commuters face when taking public transport. A group that faces large amounts of difficulty taking public transport is the Visually Impaired (VI). We aim to reduce the barriers to public transport that the VI face by developing an all-in-one system that the VI can use to know what buses are coming when at bus stops. This system uses object detection together with optical character recognition (OCR) machine learning techniques from the cloud platform Azure to identify if there is a bus coming towards the bus stop and what the bus number of the bus is. This information is then relayed via text-to-speech for the elderly to digest. We manage to successfully create the platform and test it on several videos.

**1 Introduction**

Singapore, being a small and dense city state, strongly relies on the use of public transport to reduce the impact of congestion and pollution that comes from rampant use of private transport. Thus, Singapore utilises many strategies to curb private vehicle usage, such as the Certificate of Entitlement and the Electronic Road Pricing system.

However, the other part to reduce private vehicle usage is to increase the ease of the alternative, public transport. While LTA has indeed increased the standards of public transport in Singapore over the years, groups of people such as those facing mobility issues or sensory issues are often neglected.

Enabling these neglected groups not only increases the appeal of Singapore’s public transport system, but also serves to increase the cohesiveness and unity of Singapore as it sends a strong message that no one is left behind as Singapore advances into the future.

Of these neglected groups, a large portion is made up by the VI. A study by the Singapore Eye Research Institute in 2015 found that diabetic retinopathy has to date resulted in visual impairment of more than 26100 Singaporeans[1]. Apart from diabetic retinopathy, there are many other causes of visual impairment, such as glaucoma and cataract. A risk factor of some of these ailments is age[2], which makes transport for the VI even more relevant to Singapore given that Singapore is suffering from an ageing population.

The VI face difficulties in transportation due to the lack of information available to them. In the case of bus transportation, this is with regards to where to stand (unable to find the first boullard), where the bus is, or what bus is currently at the bus stop. Furthermore, through interviews with several VI, it is found that many visually-sighted people often do know or care about the difficulties of the VI when asked for help, thus making seeking help rather difficult for the VI. Additionally, seeking help frequently leads the VI to feel less independent and erodes their self confidence. Thus, there is a need to leverage on new technologies to create an innovative solution that enables the VI to take in more information about their surroundings and take public transport more independently.

In this paper, we attempt to build an all-in-one system that leverages on the cloud platforms to analyse a feed of images taken from a camera (which will be worn by the VI), identify buses and bus numbers on the buses and relay this information via text-to-speech back to the VI.

We chose to rely on cloud computing rather than doing the machine learning locally for 2 main reasons. Firstly, doing the machine learning on the cloud enables the predictions to be done on the cloud, reducing the need to heavy amounts of resources while still being able to make fast predictions. Secondly, relying on machine learning algorithms on the cloud platforms mean that we are always using the most up-to-date and well-trained neural networks out there. The cloud platform we have chosen to work with is Microsoft Azure[3] due to its seemingly simpler workflow and student sponsorship program[4].

**1. 1 Related Work**

The concept of using artificial intelligence to help the VI with “seeing” is not something new, a very impressive product current in market is Microsoft’s “Seeing AI[5]” that uses machine learning to describe what a VI person is seeing and relaying that information to the VI. Google also has a similar product: “Lookout[6]”.

However, both these options are currently very generic and are not tailored to use in identification of bus numbers, especially at a large distance, as they are optimised for closer ranged objects such as people or supermarket products. Furthermore, Lookout is only available on Google’s Pixel smartphones, thus making it rather inaccessible for those without such devices.

As such, there is a need to work on applying such machine learning technologies on bus transportation. Perhaps, with validation that this technique works, big name companies such as Microsoft and Google will attempt to fit this technology into their existing products so that the VI have a single combined platform where they are able to use AI to help them perceive the world.

**2 Materials and Methods**

Equipment needed:

1. A 64-bit computer
2. A camera, the higher the solution, the better the results
3. Speakers or earpieces to hear the output sound

Prerequisite Software:

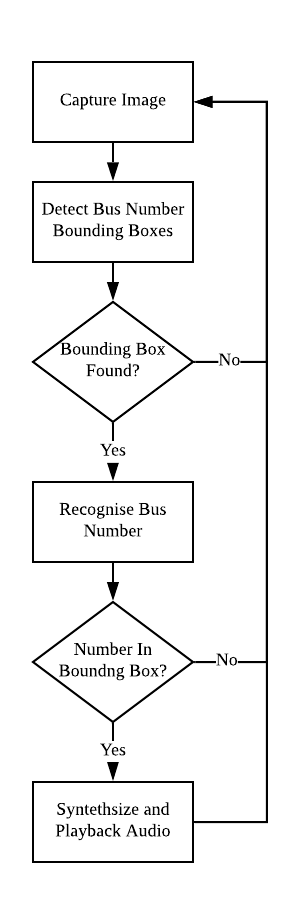
1. Python 3.7+[7]
2. Numpy (Python Module)[8]
3. Matplotlib (Python Module)[9]
4. Azure Cognitive Services Speech SDK (Python Module)
5. Jupyter Notebook[10]
6. FFmpeg (Linux Package)[11]

Other Prerequisites

1. Microsoft Azure subscription

**2.1 Overview of System Logic**

Our system performs 4 main processes. First it activates the camera to take a picture and store it as an image. After which, this image is imported into python and sent to Azure servers through a POST request to an Azure Custom Vision[12] prediction application program interface (API). The API returns the coordinates of its predicted bounding boxes for the bus numbers. We then check if there exists a bounding box which Custom Vision has a high confidence of. If so, we proceed to crop the image to the coordinates of the bounding box and use the Azure Recognize Text API[13] to extract the bus number from the image. If the API is able to find a valid number within the image, this is sent the Azure “Text-to-Speech” API[14] which will write to an audio file to be played.

****

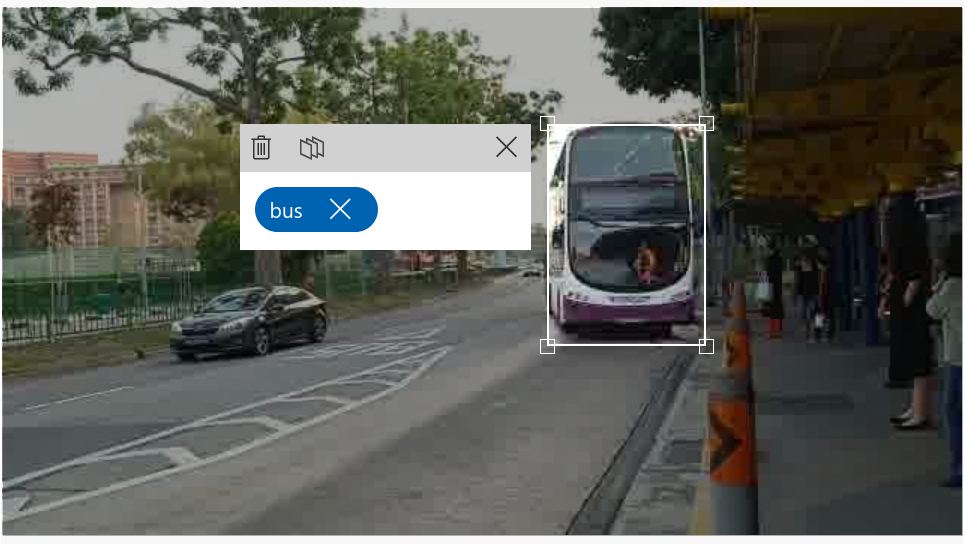
e used to improve Domato.

*Figure 2.1.0: Logical Flow Diagram of System*

**2.2 Azure Custom Vision Set-Up**

Although Azure has a pre-trained computer vision neural network[15] that detects common objects in images (including buses), this network is only able to identify the entire bus but not the actual location of the bus number on the bus. Thus, using it would result in less accurate responses from the OCR section later on.

As such, there is a need to train a custom computer vision neural network to fit our task. Azure provides a convenient platform for this in the form of Azure Custom Vision.



*Figure 2.2.0: Bus Detection with Pre-Trained Computer Vision Network on Azure*

**2.2.1 Data Gathering**

As with any machine learning project, a large amount of quality data is important to generate an accurate neural network. Although there are many datasets for vehicles available, most of them are either not based in Singapore or of the wrong angle. For the network to learn properly, the buses need to be the same as those it will be tested on (i.e. Singapore buses), furthermore, it must be from the perspective of a human standing at a bus stop and watching the on-coming vehicles.

With the above constraints, the only reasonable way to attain data would be to manually film videos of buses coming from the perspective of someone standing at a bus stop. Additionally, to reflect real life scenarios more closely, a tripod was not used as the neural network would have to take into account variations in the orientation, position and stability of the camera in relation to the buses that were coming. Hence, the preferred method of data gathering was to stand with a phone and film the buses.

After the videos were filmed, a bash script that used FFmpeg was written to extract the frames from each of the video. It then deleted all except every 10th frame to ensure reduce overfitting which could result from excessive repeated training images.

**2.2.2. Data Augmentation**

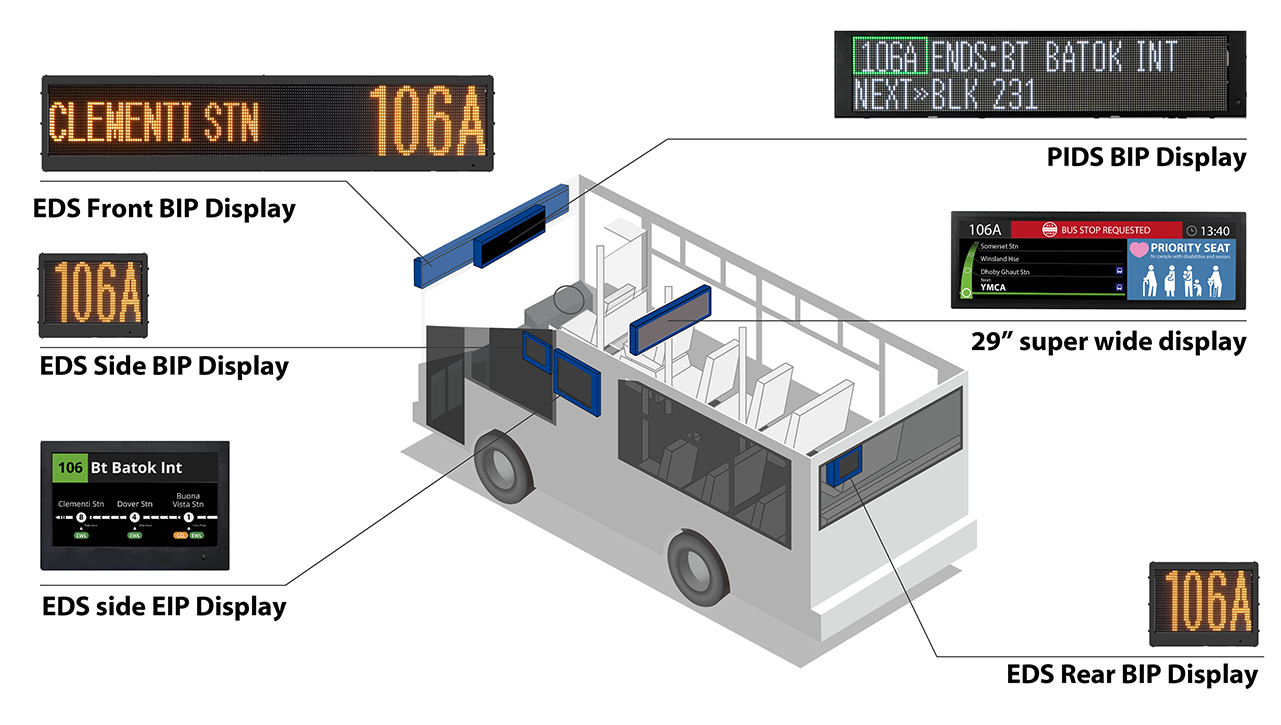
While in many cases, data augmentation such as image inversion or rotations can be used to help generate more data, for this project, the test images are guaranteed to be upright and the bus is always going to be on the left of the bus stop as the VI will be facing the oncoming traffic and Singapore is a left driving country. As such, there was no benefits and potentially worsened results if we performed such data augmentation.

**2.2.3 Data Labelling**

To reduce the difficulty of the object detection problem, we opted to only use one class “bus\_number” in our object detection model. The bounding box for the images were drawn with the built-in labelling tool from Custom Vision’s web interface. They were drawn across the top of the front of each bus as shown in the image below.



*Figure 2.2.1: Labelling Bus Numbers on Custom Vision’s Web Interface*



[Destination] [Bus Number]

*Figure 2.2.2: Destination and Bus Number within the labelled bounding boxes*

Choosing the label the bounding boxes around only the bus numbers and destinations as opposed to labelling the entire bus helped to ensure that only these sections of the bus were later cropped and sent to the OCR network. Sending the entire bus would reduce the network’s focus on the actual bus numbers and false bus numbers could be generated as a result as the OCR network finds other numbers in the image (such as car license plate numbers).

Additionally, we chose to include the destination in the bounding boxes for the images as well to give the algorithm an easier time to find the bus number. This is because the destination is always found adjacent to the bus number. Thus we theorised that the neural network could learn to search for 2 orange sections adjacent to each other. This would be easier than finding the bus number on its own, which would also have the complication of the neural network choosing to put a bounding box around the destination as opposed to the bus number as they look visually similar and in a similar location on training images.

**2.2.3 Training the Neural Network**

While this is often the most tedious part of machine learning projects, the use of Azure Custom Vision greatly simplified the process it automated the splitting of the labelled images and the training loop for their neural network. Furthermore, the training was done on Azure servers and thus could utilise their more powerful graphics processing units (GPUs). Training thus only took about 10 compute minutes to complete. Performance of the neural network can be seen under section 3: Results.

**2.2.4: Incorporating Custom Vision Network into System**

After the training is done, the neural network can be accessed via a POST request to the API end-point for Custom Vision Prediction. The image is sent in binary as the request body. The response will contain JavaScript Object Notation (JSON) data on the position of the different bounding boxes and the confidence score.

For each image, we assume there is only at most 1 bus within the image, thus we identify the bounding box within the response with the highest confidence score. If this confidence score is higher than a certain threshold (we set ours to be 20%), then there is considered to be a bus in the image and the bounding box coordinates are subsequently used to crop the image, else the image is discarded and the system sleeps for a while before taking another image to process.

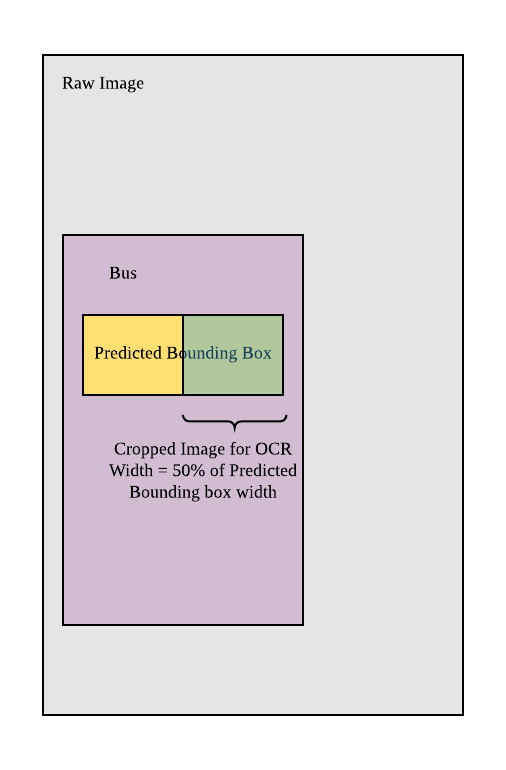
|  |
| --- |
| {"probability":0.0173978712,"tagId":"d136a86f-b60f-47f7-91d9-24bbf479d74e","tagName":"bus\_number","boundingBox":{"left":0.008975584,"top":0.006480351,"width":0.0128555633,"height":0.050089933}} |

*Figure 2.2.3: JSON Data of a Bounding Box in the Response. Probability is the confidence score.*

**2.3.0 Data Preparation for OCR**

After the location of the bus number on the bus is found out by Custom Vision, we use the Python Image Library to crop the image to be ready to send to the OCR algorithm.

We cropped the image to the right 50% of the predicted bounding box. This was to crop away as much of the destination in the bounding box as there was in order to ensure that the bus number fills up most of the space of the image.



*Figure 2.3.0: Illustration of Cropping*

Furthermore, we performed 2 types of image resizing on the cropped image.

Firstly, if the number of pixels horizontally or vertically was less than 50, the OCR network on Azure would not accept the image, thus we padded the image with white space to fill up this requirement.

Secondly, if the image was more than 50 pixels tall, we would resize the image to be just 50 pixels tall and shrink the width appropriately to preserve the aspect ratio. This sounds counterintuitive as the decrease in resolution is seen to reduce image quality which would typically lead to worse OCR performance. However, as the bus number is formed from individual lights on the front panel of the bus, in close up shots that lead to high resolution, the lights could appear separated, leading to the OCR being unable to detect the number.



*Figure 2.3.1: Bus number Before Lowering Resolution (note the distinctly visible grey strips)*



*Fig 2.3.2: Bus Number After Resolution Reduction (less visible grey strips)*

**2.3.1 Performing the OCR**

Performing the actual OCR was relatively easy as we used used the pre-trained Azure Cognitive Services “Recognise Text API”. This was chosen instead of the other 2 available OCR APIs on Azure (“OCR API” and “Read API”) as OCR API was based on an older recognition model and Read API is optimised for text heavy images such as documents which does not fit our OCR needs.

Similar to the Custom Vision API, a POST request is sent and the response contains the different text values. To ensure that we get only the bus number and not any remains of the destination, we search the response by words and pick the first word that begins with a number.

**2.4.0 Text-to-Speech Synthesis**

To inform the VI about the upcoming bus, we need to convey the information through audio. Again, we rely on Azure’s Cognitive Services which has a text-to-speech synthesising API. We synthesize the sentence “Bus {bus\_number} is coming now!” and write it to an audio file which is then played.

**3 Results**

*Table 3.0: Configuration and Performance of the Custom Vision Model*

|  |  |
| --- | --- |
| No. of Labelled Images | 386 |
| No. of Negative Images | 572 |
| Training Time | ~10 Minutes |
| Precision | 100.0% |
| Recall | 93.5% |
| mAP | 96.1% |

*Table 3.1: Performance of System as a Whole*

|  |  |
| --- | --- |
| Videos Tested | 6 |
| Videos Successful | 5 |

*Table 3.2: Example of Successful Attempt*

|  |  |
| --- | --- |
| Raw Image |  |
| Bounding Box |  |
| Cropped Image |  |
| Bus Number | 298 |

*Table 3.3: Example of Unsuccessful Attempt*

|  |  |
| --- | --- |
| Raw Image |  |
| Bounding Box |  |

**4. Discussion**

Overall, the network works relatively well for images taken at the correct angle and from the right distance.

The overall good performance of the system is likely due to the lack of variety amongst Singapore public buses. There are only a handful of bus designs, thus with enough data, our Custom Vision Neural Network is able to quickly learn how to identify which parts of the bus to look for to attain the bus number.

However, once the angle for the images changes (for example, refer to table 3.3), the network has a harder time finding the correct bus number. That being said, this is likely not a big issue as the system is intended to be used from only 1 general angle (with the user at the bus stop and facing the oncoming buses).

**5. Limitations and Future Work**

While doing this project, we were faced with 2 major constraints, time and computational resources.

Machine learning in its current state is very data-driven, as such, a large amount of data is needed for good results, however, the process of collecting data is tiring and tedious, thus were only able to collect about 1000 frames of images for this project. By collecting more images of different buses and bus models, we may be able to improve the network’s performance.

Furthermore, we lacked access to dedicated hardware to train models locally. Our only desktop did not have a dedicated GPU to accelerate the training. Thus it was not feasible to train large networks such as YOLOv3[16] or SSDMobileNet[17]. As such, we decided that it would be most ideal to leverage Azure’s Custom Vision for our project.

Additionally, our current system is still only able to work from a computer with a webcam and we have yet to make it available as a mobile application which will be the main way the VI would have access to the system.

Although we have attained a successfully working system to detect bus numbers for the VI at bus stops, there is room to improve. With better selected and customised neural network architecture and data, we may be able to improve its accuracy. Apart from just detecting bus numbers, this could be improved to tell the user where to stand to wait for the bus or where to face to allow the VI a more comprehensive solution to reduce the barrier of bus transportation. Furthermore, it could be improved by incorporating it into a bluetooth wearable such as a pair of spectacles that contain a camera as this would be more convenient for the VI to use while on the go.

**6. Conclusion**

We create a system to detect the bus number of oncoming buses and relay that information to the VI through audio means.

**Acknowledgements**

I would like to thank my school Temasek Junior College for giving me the opportunity to do this project. Furthermore, I would like to thank Microsoft for offering a Student Sponsorship on Azure that enabled me to use that platform for this project.

**References**

[1] "Singapore's Eye Health." Snec, https://www.snec.com.sg/giving/singapores-eye-health. Accessed January 06, 2020.

[2] "Risk Factors for Visual Impairment." PORTAL MyHEALTH, http://www.myhealth.gov.my/en/risk-factors-for-visual-impairment/. Accessed January 06, 2020.

[3] "Microsoft Azure Cloud Products, Services, Solutions | Microsoft Azure." Microsoft, https://azure.microsoft.com/en-us/. Accessed January 06, 2020.

[4] "Azure for Students – Free Account Credit | Microsoft Azure." Microsoft, https://azure.microsoft.com/en-us/free/students/. Accessed January 06, 2020.

[5] "Seeing AI | Talking camera app for those with a visual impairment." Microsoft, https://www.microsoft.com/en-us/ai/seeing-ai. Accessed January 06, 2020.

[6] Vincent, James. "Google releases Lookout app that identifies objects for the visually impaired." The Verge, March 13, 2019, https://www.theverge.com/2019/3/13/18263426/google-lookout-ai-visually-impaired-blind-app-assistance.

[7] "Welcome to Python.org." Python, December 20, 2019, https://www.python.org/. Accessed January 06, 2020.

[8] "NumPy — NumPy." Numpy, https://numpy.org/. Accessed January 06, 2020.

[9] "Matplotlib: Python plotting — Matplotlib 3.1.2 documentation." Matplotlib, https://matplotlib.org/. Accessed January 06, 2020.

[10] "Project Jupyter | Home." Jupyter, https://jupyter.org/. Accessed January 06, 2020.

[11] "FFmpeg." Ffmpeg, https://www.ffmpeg.org/. Accessed January 06, 2020.

[12] "Custom Vision." Home, https://www.customvision.ai/. Accessed January 06, 2020.

[13] "Printed, handwritten text recognition - Computer Vision." Azure Cognitive Services | Microsoft Docs, April 17, 2019, https://docs.microsoft.com/en-us/azure/cognitive-services/computer-vision/concept-recognizing-text. Accessed January 06, 2020.

[14] "Text to Speech | Microsoft Azure." Microsoft, https://azure.microsoft.com/en-us/services/cognitive-services/text-to-speech/. Accessed January 06, 2020.

[15] "Computer Vision | Microsoft Azure." Microsoft, https://azure.microsoft.com/en-us/services/cognitive-services/computer-vision/. Accessed January 06, 2020.

[16] "YOLO: Real-Time Object Detection." Pjreddie, https://pjreddie.com/darknet/yolo/. Accessed January 06, 2020.

[17] "Review: SSD — Single Shot Detector (Object Detection)." Towardsdatascience, https://towardsdatascience.com/review-ssd-single-shot-detector-object-detection-851a94607d11. Accessed January 06, 2020.

**Appendix**

System Main Script

|  |
| --- |
| #!/usr/bin/env python  # coding: utf-8  # # Identification of Bus Numbers from Images  #  # ### Steps  # 1. Find bounding box of bus numbers  # 2. Crop image to bounding box location  # 3. OCR on cropped image  # 4. Text to Speech to tell the visually impaired about the bus  # In[12]:  # Libaries  import requests, json, numpy as np, time  import azure.cognitiveservices.speech as speechsdk  from PIL import Image  from matplotlib.pyplot import imshow  get\_ipython().run\_line\_magic('matplotlib', 'inline')  get\_ipython().system('ls')  # ## Use Azure Custom Vision to Find Bounding Box of Image  # In[47]:  threshold = 0.2 #Threshold on what probability corresponds to a valid bounding box  test\_image = "bus\_ext\_6.jpg"  custom\_vision\_api = "https://southcentralus.api.cognitive.microsoft.com/customvision/v3.0/Prediction/b35dc00f-1a23-4f90-a2f1-c406952ff467/detect/iterations/Bus\_Numbers\_1/image"  prediction\_key = "secret"  with open(test\_image, 'rb') as image\_file:  custom\_vision\_response = requests.post(custom\_vision\_api, data=image\_file, headers={"Prediction-Key": prediction\_key, "Content-Type": "application/octet-stream"} )  print("Custom Vision Response:", custom\_vision\_response.text)  json\_response= json.loads(custom\_vision\_response.text)  bounding\_boxes = json\_response['predictions']  # In[48]:  # Extract Best Bounding Box with Probability > 0.5  max\_probability = -1  for bounding\_box in bounding\_boxes:  #print("Box:", bounding\_box['probability'])  max\_probability = max(max\_probability, bounding\_box['probability'])  if max\_probability < threshold:  print("No Valid Bounding Boxes Found")  for i in bounding\_boxes:  if i['probability'] == max\_probability:  bounding\_box = i  print(bounding\_box)  # In[49]:  bounding\_box = bounding\_box['boundingBox']  print("Bounding Box:", bounding\_box)  # ## Use Python Image Libary to Crop Image at Bounding Box  # In[50]:  # Import Test Image into Python  raw\_image = Image.open(test\_image)  width, height = raw\_image.size  # Set Points for Cropped Image to Bounding Box  left = width\*bounding\_box['left']  right = left + width\*bounding\_box['width']  top = height\*bounding\_box['top']  bottom = top + height\*bounding\_box['height']  # Crop Image  bus\_num\_image = raw\_image.crop((left, top, right, bottom))  print("Image of Bus Number")  imshow(np.asarray(bus\_num\_image)) #Display the Image  # In[51]:  ocr\_crop\_percentage = 0.5  # Crop Away ocr\_crop\_percentage of Left Side for OCR Reasons  width, height = bus\_num\_image.size  left = width \* ocr\_crop\_percentage  right = width  top = 0  bottom = height  ocr\_image = bus\_num\_image.crop((left, top, right, bottom))  # Resize Image with Interpolation if height too big (somehow OCR on Azure doesn't work too well with too sharp of bus numbers)  width, height = ocr\_image.size  if height > 50:  ocr\_image = ocr\_image.resize((int(width\*50/height),50),Image.ANTIALIAS) # Ensure the aspect ratio doesn't change  print("OCR Ready Image")  imshow(np.asarray(ocr\_image)) #Display the Image  # Save OCR Ready Image  ocr\_image\_file = "ocr.png"  ocr\_image.save(ocr\_image\_file)  # In[52]:  # Fits image into a square of at least 50x50 pixels by padding white space  def Reformat\_Image(ImageFilePath):  from PIL import Image  image = Image.open(ImageFilePath, 'r')  image\_size = image.size  width = image\_size[0]  height = image\_size[1]  if(width != height or (width < 50 and height < 50)):  bigside = width if width > height else height  if bigside < 50:  bigside = 50  background = Image.new('RGBA', (bigside, bigside), (255, 255, 255, 255))  offset = (int(round(((bigside - width) / 2), 0)), int(round(((bigside - height) / 2),0)))  background.paste(image, offset)  background.save(ocr\_image\_file)  print("Image has been resized !")  print(background.size)  else:  print("Image is already a square, it has not been resized !")    Reformat\_Image(ocr\_image\_file)  # ## OCR with Azure Cognitive Services  #  # Uses the Recognise Text API which operates asyncronously  # In[53]:  # Sending Image file to Recognise Text API  ocr\_key = "secret"  ocr\_api = "https://southcentralus.api.cognitive.microsoft.com/vision/v2.0/recognizeText"  #ocr\_image\_file= "helloworld.png"  params ={"mode": "Printed"}  with open(ocr\_image\_file, 'rb') as image\_file:  print("Sending", ocr\_image\_file)  ocr\_response = requests.post(ocr\_api, data=image\_file, headers={"Ocp-Apim-Subscription-Key": ocr\_key, "Content-Type": "application/octet-stream"}, params=params )  print("Resource location", ocr\_response.headers['Operation-Location'])  ocr\_response  # In[54]:  # Request for the Result  request\_result\_api = ocr\_response.headers['Operation-Location']  while True:  ocr\_status = requests.get(request\_result\_api, headers={"Ocp-Apim-Subscription-Key": ocr\_key})  print("Response Text:", ocr\_status.text)  json\_response= json.loads(ocr\_status.text)  if json\_response['status'] == "Succeeded":  print("OCR Finished")  break  else:  time.sleep(0.5)  # In[55]:  # Extract Lines from Response  lines = json\_response["recognitionResult"]["lines"]  if len(lines) == 0:  print("No Text Identified...")  # Finding first word that begins with a number  predicted\_number = ""  for line in lines:  for word in line["words"]:  #print(word["text"][0])  if word["text"][0].isdigit():  predicted\_number = word["text"]  break  if predicted\_number == "":  print("Failed to Get a Number...")    else:  print("Predicted Bus Number:", predicted\_number)  # ## Synthesise Speech to Output File  # In[56]:  speech\_key, service\_region = ocr\_key, "southcentralus"  speech\_config = speechsdk.SpeechConfig(subscription=speech\_key, region=service\_region)  audio\_filename = "bus\_number.wav"  audio\_output = speechsdk.AudioOutputConfig(filename=audio\_filename)  speech\_synthesizer = speechsdk.SpeechSynthesizer(speech\_config=speech\_config, audio\_config=audio\_output)  text = "Bus "+ predicted\_number + " is comming now!"  result = speech\_synthesizer.speak\_text\_async(text).get()  if result.reason == speechsdk.ResultReason.SynthesizingAudioCompleted:  print("Speech synthesized to [{}] for text [{}]".format(audio\_filename, text))  elif result.reason == speechsdk.ResultReason.Canceled:  cancellation\_details = result.cancellation\_details  print("Speech synthesis canceled: {}".format(cancellation\_details.reason))  if cancellation\_details.reason == speechsdk.CancellationReason.Error:  if cancellation\_details.error\_details:  print("Error details: {}".format(cancellation\_details.error\_details)) |