Phase 2

Authors: MyongJoon Chang, Mark McAvoy, Eurey Noguchi

Our detection program uses Google TensorFlow Object Detection API, thus a properly installed TensorFlow Object Detection API is required for the program to run

The instructions for properly installing can be found here:

<https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/installation.md>

Copying the contents of our zip file into the ../models/research/objectdetection and run the code through

python ourtry.python --input --output

During Phase2 our main-focus had been correctly training our model to detect door signs by retraining TensorFlow’s already existing model. This required us to first prepare the dataset that the model could be retrained on. The format the Object Detection used was a TFRecord file type, which was created after creating a bounding box on an IMG file. For the training data set this required us to first convert the training videos into image files and then annotate the images to create bounding boxes which could be converted into the TFRecord file format.

Using OpenCV we converted the videos into image files getting 2 image frames per second. Annotating and conversion into the TFRecord file format was not a challenge, due to the huge amounts of information readily available. However, the main problem was getting the training to work on Google Clouds ML-Engine.

The main problem that our team faced was starting the ML-Engine. This was mainly due instructions found on github, Medium and other sources being outdated. In such a fast pace atmosphere where updates were happening with updates happening for different dependencies a no instruction or guide could be taken for granted. Even something as simple as packing packages to upload on ML-Engine required a lot of digging around and trial and error, eventually requiring us to use a different method to many of the instructions found on the internet (even the TensorFlow’s main instruction was outdated).

After getting the training to start on the ML-Engine another problem that occurred was after training for 2~3 hours an error would occur, turning the last couple hours into a waste. This greatly limited the speed of our advancement since it limited how many trial and errors that we could try day.

After figuring out all the errors and managing to train a single test model for a minimum of 2000 steps, we realized that some models were bad at detecting door signs even after training. Thus we started to try training on numerous different models which could all be found on the TensorFlow detection model zoo. Some models were so inadequate that even after 5000 steps it continue to have losses in the several tens of thousands with an accuracy barely passing 10%. After successfully re-training 10 models we found that the rfcn\_resnet gave the best results with the loss at a minuscule low level and accuracy significantly high.

We altered our program to then use the frozen model and look for door signs. Testing our model on our training data showed that it was extremely accurate at recognizing door signs with only a couple false positives. False positives were easily ignored by requiring our model to only output bounding boxes with a objects that it had confidence of at least 80%.

However, our focus on the Object Detection API led to a huge mistake and ultimately our downfall. Originally our plan had assumed that the Tesseract OCR, which could be easily and readily implemented, would be sufficient at reading the numbers within the bounded box that our Object Detection API could isolate. We could not have been more wrong. This false hope had started during Phase1 when the Tesseract OCR had correctly read numerous door sign numbers, however this had been pure coincidence.

Our original plan for this project had been to advance in this order, the input file would be read using OpenCV and broken up frame by frame. Our object detection model would then take the input images, check if there is a door sign and if the confidence was greater than 80% create a bounding box around it. Using these coordinates, we would use OpenCV to create a copy and crop the image at the bounding box and feed this isolated image into Tesseract. Lastly the Tesseract OCR would recognize the digits in the cropped image and give up the output that we were looking for.

However, the Tesseract OCR was horrible at recognizing numbers, but was great at reading words (it read Professor names or door labels with surprising accuracy). To try and help the Tesseract OCR to recognize numbers we tried numerous methods. First, we converted the images to a gray scale, hoping that it would make the digits clearer. Next, we use OpenCV’s edge detection to “sharpen” the image. Next, we tried to use OpenCV’s Adaptative Gaussian Thresholding to try and isolate all irrelevant noise, and last, we also tried to change the brightness and contrast. However, all of these steps failed to help the Tesseract recognize digits.

Therefore what we currently have is a program which is great at recognizing the door sign, however horrible at recognizing the numbers of the door sign.