Detailed Design Specifications

Devin M. Alexander

John Mitas

Reid Barden

We experienced a few slight hurdles during the beginning phase of our project due to problems locating and securing a proper lab in which to set up the WBT, which dampened our ability to begin learning its interface. This gave us a fair amount of time to speculate on potential design avenues, as well as time to research different classifiers. It was during this time that we weighed the pros and cons of multiple classifier attributes, such as size, running time, accuracy, and precision. Ultimately we decided to use TensorFlow’s image classifier, which allows us to retrain the final layer of a number of convolutional neural networks (CNN). The CNNs that we decided to retrain were MobileNet 0.5, MobileNet 0.75, and Inception v3. At this point we have run a number of classifications on successively larger data sets, and continue to achieve Promising results.

*Methodology*:

We begin by connecting the WBT to a machine running Linux, via an Ethernet cable, and making an X11 connection via ssh. We use VNCViewer to view the window. We open the Spectrum Analyzer application, and confirm that the settings for each image retrieval session are set in a consistent manner. For our data sets, we have kept the attenuation set to 20dB, the sample rate at 42, and the bandwidth at 40. The X11 window displaying the application is then moved to the upper-left hand corner (0,0 coordinate) of the screen, which is the correct position for the snapshotting script to begin autonomously grabbing a predefined number of images of the waterfall display. It is these images that will later be used to retrain the CNN. The bash script has been included in the file set contained in this submission. The script takes the following parameters, (# of images) (desired width) (desired height) (x-coordinate) (y-coordinate) (destination directory) (beginning frequency) (end frequency). Upon execution, the script will input the proper number to the X11 window to center on the beginning frequency, wait for the waterfall image to ‘fill out’, and will take the specified number of images. The script properly formats the images, and saves them to the destination directory. It will then shift the center frequency to the left, and the process is continued until the ending frequency is reached. This allows for a range of images of any specific frequency to be grabbed without manual intervention.

Each time the script is run, it will save the images to a folder that we have named to correspond with the frequency range that the images will contain (i.e. FM, WIFI, No\_signal). These folders, once adequately populated, are passed via Python script to TensorFlow, along with a CNN and a predefined number of retraining steps. The more steps taken, the higher the degree of accuracy we should expect in our final classifier, to some limit. After proper execution of a retraining, any new image can be passed via Python script to be compared against the retrained graph, and the classifier will out put the name of the frequency (or absence thereof) that it believes it recognizes, along with a percent of confidence.

Our first retraining consisted of 750 images of various signals, and 750 images of images containing no discernible signals. We started with this binary approach to ensure that the classifier at very minimum could make a distinction between a signal or not. For this retraining, we used the MobileNet .75 CNN, and upon execution were able to achieve correct classification results at a confidence level of 99.9999%.

Our second retraining focused on distinguishing between different distinct signals and the absence of a signal. We used the script to gather 1100 images with a bandwidth of 40 and a center frequency in the range of 88-108 (for image variance), and saved the images to a folder named FM. We recorded 1100 images with a bandwidth of 40 and a center frequency in the range of 2400-2420 (for image variance), and saved the images to a folder named WIFI. Furthermore, we included a file called NoSignal to the same file directory, containing 750 waterfall images containing no signal. Upon execution, we saw the same degree of accuracy that we encountered on our first iteration. We did notice, however, that if an image containing a signal that is of a frequency which we did not include in our classification set is passed to the classifier, the classifier is forced to choose a set. The image is therefore classified incorrectly, but the confidence level drops to between 70%– 90%.

We have since done a much larger classification on 6945 images, encompassing seven different classes, and taking 12,000 retraining steps. Our confidence level has dropped to ~97-99%, however we still see a very promising degree of accuracy. We believe that by scaling the project up we can eliminate false positives, and may keep a high level of confidence.

Iteration Specifications:

Iteration 1:

During this iteration, we had our initial meeting with Mr. Cushing and discussed the project specifications. Mr. Cushing left us with a WBT to begin learning on. It was imperative at the time that we locate a lab that could receive proper signals. In the meantime, the group set out to explore our options, do some research, and discuss potential project routes. We had initially thought we would be using a software called YOLO for image classifications, and therefore each group member installed CUDA, OpenCV, and YOLO, and were able to do object-detection in real time via web cam. It was also during this time that we began to explore Tensorflow, the Deep Machine Learning tool we would ultimately end up using.

Iteration 2:

We continued research with YOLO, its VOC classifier, and TensorFlow. We learned that TensorFlow can pair nicely with the Inception v3 CNN, which provides a much higher degree of accuracy and precision than the VOC classifier. Later in the iteration, Dr. Dahlberg secured a corner lab in the East Engineering Hall in which we could set up the WBT and begin to become familiarized with it.

Iteration 3:

At this point we had had brief meetings with Mr. Cushing via Skype to discuss our progress, as well as our weekly meetings with Professor Dahlberg. The group was still discussing potential methods for image classifications, and specifics such as whether we should focus more heavily on the waveform or the waterfall image sets. It was ultimately decided that we start with the waterfall images because of the assumption that a classifier would have more to ‘work with’. We acquired the documentation for the WBT, and made connection to it via ssh. We became comfortable with all of the software on the WBT.

Iteration 4:

It was during this phase that we received the hard drive containing signal recordings from Mr. Cushing. The first shell script was written to aid us in image retrieval from the device. The team also made the first image classification, which is detailed in the methodology section above. We further refined the bash script, and added functionality to further automate our needs. The team reached out to the Virginia Tech team in hopes of collaboration, but to little success.

Iteration 5:

During this phase we continued to make progress on the project as defined by the previous iteration. We performed two more image classifications, which are outlined in the methodology section above. We emailed Mr. Cushing with a detailed outline of our achievements as well as our goals moving forward. We altered the shell script to that an image classification could be used on a system running Mac OSX, and we created a small script to simplify the process of passing an image to the classifier. We also put together a short presentation/demonstration to show the class. Unfortunately at this stage, we had yet to hear back from our Virginia Tech counterparts.

Iteration 6:

Until this iteration, all of our classifications had been executed on the MobileNet CNN, but we did our first retraining using the much larger and more powerful Inception v3 CNN. Even with 12,000 steps, this process terminated successfully in less than five minutes on the machine outline below. This was promising for a number of reasons: we should expect to run increasingly larger data sets in a reasonable amount of time, and we could hope to ensure a high degree of accuracy. We gave our presentation to the class, which included a demonstration of the classifier, discussion of our achievements, successes, implementations, goals, and fielding questions. It was also during this phase that we began to gather or code for submission and developing a project plan for the winter break.

Iteration 7 (ongoing):

We plan to continue through our with our ongoing classification growth. We have also discussed the development of software that can reside on the WBT itself to do classifications, but we must first examine the exact limitations in terms of available memory and space, as well as any hardware restrictions that may exist. If the conditions are right, porting our current implementations tp the WBT should be fairly painless as the WBT uses the same Operating System that we have been using thus far for all of our implementations.

Current environment specifications:

OS: Ubuntu 16.04

GPU: NVIDIA GeForce GTX1060 6GB

CUDA Version: 8.0 V8.0.61

Python Version: 2.7

cuDNN Version: 6

OpenCV Version: 3.3.0

TightVNC Viewer Version 1.3.10