1) Introduction. An overview of the project and an outline of the shared work.

The goal of our project was to train a machine learning model to recognize small vehicles (cars, pickup trucks, minivans ect…) in satellite images. We hoped to be able to figure out the directions the vehicles were facing as well, but ultimately, we had insufficient time to implement that. To do this we had to train a evaluate a model against a baseline model. Our research suggested that You Only Look Once (YOLO) was the most common baselining model and we decided to use several different versions of Retinanet with different numbers of convolutional layers as the model we were testing.

2. Description of your individual work. Provide some background information on the development of the algorithm and include necessary equations and figures.

My portion of this project was twofold. First, I did all of the baselining work implementing Yolo, Fast-YOLO and the mAP performance analysis. Second, I also implemented early stopping as a way to prevent overfitting in our Retinanet model.

I adapted a Github repository (https://github.com/postor/DOTA-yolov3.git) is an implementation of YOLO version 3 designed to work with the DOTA dataset, which made implementation easier. There was no evaluation code in the repository, so I wrote a mAP calculator for it to allow us to compare model performances. The training time of the model was extremely long, so we decided to use Fast-YOLO instead of the standard architecture to try to train faster. This version of the model has fewer convolutional layers and so fewer layers to propagate through and train. The tradeoff is that the model also has lower accuracy. At the time we did not appreciate how low YOLO's accuracy was going to be, but Fast-YOLO still took more than 24 hours to train (we manually killed it after 24), so I do not believe we could have gotten meaningful results from the standard YOLO model either.

We knew the metric we were going to use to evaluate the models was mAP, so I implemented early stopping in the training code for Retinanet based on the mAP of the validation set. If the mAP didn't increase (i.e the average agreement between our model boxes and the annotated model boxes) for 5 or more iterations, then the training stopped. This was to prevent overfitting to our test data or catastrophic loss of the pretrained model weights. We settled on 5 based on training time and model accuracy.

3. Describe the portion of the work that you did on the project in detail. It can be figures, codes, explanation, pre-processing, training, etc.

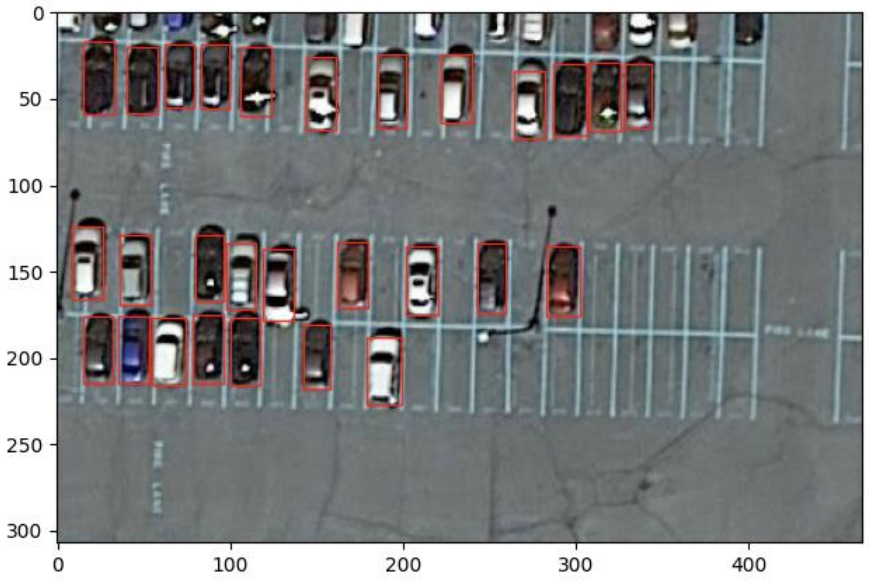
See above.

4. Results. Describe the results of your experiments, using figures and tables wherever possible. Include all results (including all figures and tables) in the main body of the report, not in appendices. Provide an explanation of each figure and table that you include. Your discussions in this section will be the most important part of the report.

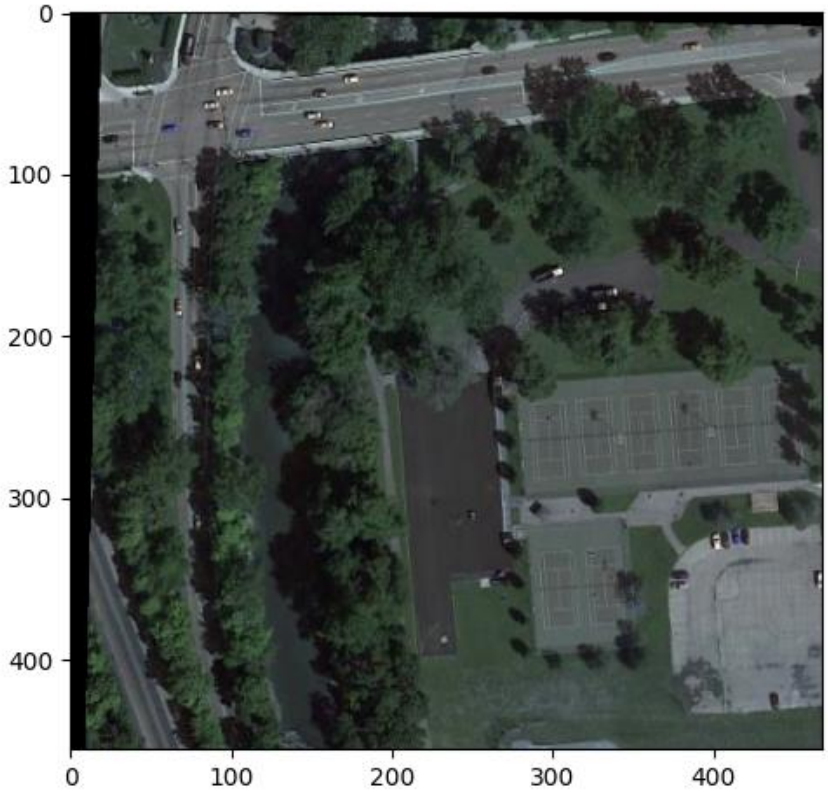
Model	Training Time	Inference Time	mAP
Retinanet - Resnet18	00:03:15:00	127.2	0.44

Retinanet - Resnet50	00:05:30:00	178.9	0.47
Retinanet - Resnet101	00:06:45:00	200.9	0.55
Fast YOLO	01:00:00:00	26.3	.0046

The above table summarizes the results of the four models. The performance of the Retinanet models was what we expected. As we increased the number of convolutional layers the accuracy increased as did the training and inference times. One of the major factors contributing to the accuracy of the Retinanet models not being higher was the highly variable image size. In some images, cars were more than 50 or 75 pixels a side. In others, cars were only a few pixels on a side. The smaller the objects got the harder it was for the model to detect they were objects at all, much less categorize them accurately. For example, in the first image almost all cars (except the ones cut off at the top) are identified correctly.



In this second image almost none of the cars are correctly identified. Note that the two images are roughly the same size in pixels, but the second image is much more zoomed out. This means the objects are correspondingly smaller and therefore harder for the model to identify.



The performance of the YOLO model was much more surprising. YOLO is not designed to work on images that have as many objects as densely packed together as our images do. It is designed with the assumption that there is a single object whose center is at the center of one of the slices. Each box estimates the probability that there is an object whose center is in it and then attempts to classify that object. If there are multiple object inside the box the model will not be able to identify them as the probability of there being an object is bounded by 100%. If the objects are not of the same class the model gets even more confused because it identifies features of multiple classes of objects and will not be confident in any class. We could reduce the size of the boxes to reduce the number of times multiple objects wind up in the same box, but training time varies with the square of the number of slices (or linearly with the number of boxes). Given that the training time for YOLO was already a limiting factor we cannot increase it any more.

5. Summary and conclusions. Summarize the results you obtained, explain what you have learned, and suggest improvements that could be made in the future.

Retinanet performs substantially better than YOLO. This is likely because our training set does not meet the assumptions of the YOLO model, but does meet the assumptions of Retinanet. In the future I would try to baseline against several other standard image recognition models as well as attempt to run the YOLO model with smaller boxes to see if the increased accuracy might speed up the training process.

6. Calculate the percentage of the code that you found or copied from the internet. For example, if you used 50 lines of code from the internet and then you modified 10 of lines and added another 15 lines of your own code, the percentage will be 50−10 50+15 ×100.

Train.py: (198)/(198+20)X100 = 95.19%

Yolo2.py: (10 – 10)/(10 + 39) X100 = 0%

Note that this is a shall scrip that calls several hundred lines of unedited python code, so may also be ~100%

Yolo_evaluate.py: (202)/(202+134) X100=60.12%

7. References.

https://github.com/postor/DOTA-yolov3.git

https://github.com/yhenon/pytorch-retinanet.git

Redmon et al., 2016 (https://arxiv.org/pdf/1506.02640.pdf)