Group 8 Final Report
Thomas Keeley
David English

Introduction

The exploration of Deep Learning concepts and application of neural networks persistently pushes the boundary throughout several technical domains. Deep Learning as a science and network design as an art successfully challenge the traditional machine learning that have been exercised for decades. The dimensionality offered in these layered networks along with enhanced computing resources has expanded the capabilities of machine learning in modeling data sources with much success. One particular type of data being targeted in Deep Learning is images. Several capabilities have been explored in applying neural networks to images such as image segmentation, image classification, object recognition, and object detection. This project[1] focuses on the application of object detection, specifically, the ability to detect small vehicles in satellite/overhead imagery.

There have been many benchmark models developed in the domain of object detection with several applications in analyzing scenes that humans see on a daily basis. This project will instead focus on object detection in satellite imagery by comparing two widely adapted network architectures, You Only Look Once (YOLO) and Retinanet[2]. A description of the available datasets in applying object detection techniques to satellite images will be presented with an emphasis on the data used in this project. Next, a walkthrough of the design, advantages and potential disadvantages of YOLO and Retinanet then the implementation of these networks in the context of small vehicles in satellite imagery will be discussed. Finally, the results and comparison will be presented with proposed next steps for further exploration.

Data

A collection of satellite imagery datasets have been made publicly available to help enable researchers in crafting innovative solutions to analyzing large volumes of overhead imagery in various contexts. Many of which are made available through competitions such as SpaceNet[3] which has traditionally targeted building footprints. Our initial intent was to use the Cars Overhead With Context (COWC)[4] dataset, but the 7 images in that set were far too large to be preprocessed, much less to train a model on. After trying several different methods of slicing or compressing the images we realized that we needed to move to a different dataset. At this point we realized we would need to use a different data set. Another widely used dataset is xView[5] which contains over 1 million images at a very-high resolution of 0.3 meters with 60 classes of

[1] https://github.com/datsgwu/Final-Project-Group8
[2] Lin et al., 2018
[3] https://spacenet.ai/
[4] https://gdo152.llnl.gov/cowc/
[5] http://xviewdataset.org/

objects. The dataset leveraged by this team however, is DOTA: A Large-scale Dataset for Object Detection in Aerial Images[6]. DOTA is similar to xView in that it captures multiple classes of objects. However, rather than providing annotations of horizontal bounding boxes for the objects, the annotations reflect rotated bounding boxes that capture the shape and orientation of the object.

The initial intent of this project was to leverage the rotated bounding boxes in conducting object detection with greater contextual accuracy. Unfortunately, time constraints and limited computing resources hindered the ability of successfully modeling this advantage that DOTA provides. Instead, the DOTA annotations were manually converted to reflected fitted horizontal bounding boxes. Further, the annotations were ingested and filtered to capture only those that describe the bounding areas of small vehicles. The images come in a wide range of physical coverage areas and aspects. Additional filtering was conducted to extract images with a smaller field of view. The images were then sliced to produce smaller, more focused images. The associated annotations were then refactored to map to the newly created images. This will allow the model to recognize and classify the small vehicles with greater accuracy.

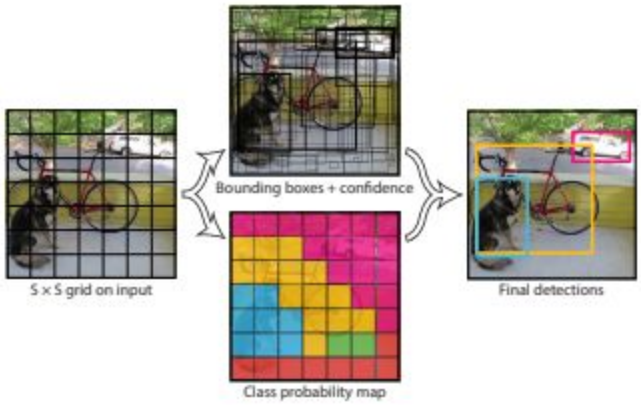
Deep Learning Network

Object detection algorithms can usually be grouped into two main categories: single stage detectors and two stage detectors. Single stage architectures make up the earlier applications of object detection and have been widely used. These types of models are named accordingly as they take a single pass at detecting objects whereas two stage detectors are conducted by first producing regional proposal networks in the first stage and then detecting objects in each proposed region. There is typically a tradeoff between single stage architectures and two stage detectors where single stage is a simpler and faster approach. However, greater performance and accuracy may be offered through a two stage detector.

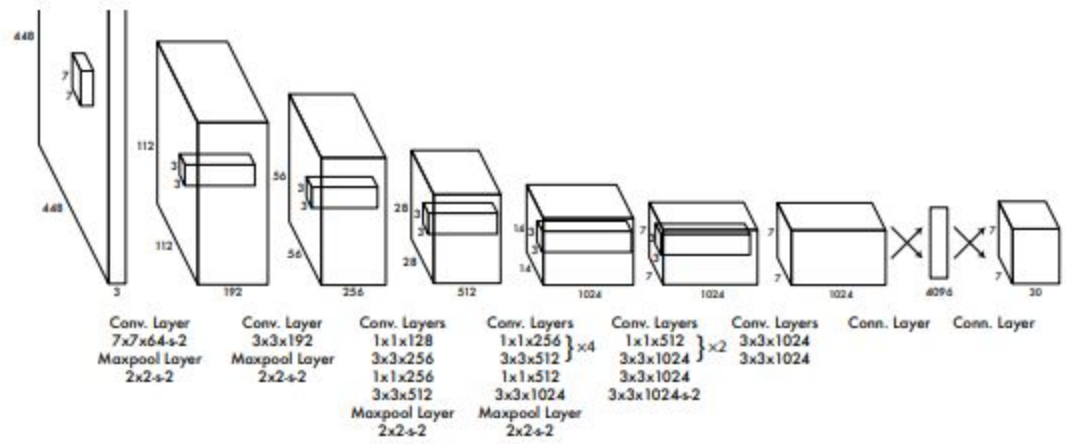
You Only Look Once, or YOLO, is an object detection network designed to address the speed limits and complexity of first drawing boundary boxes on an image and then passing back through with a classifier to classify what was in each box[7]. As the name implies, YOLO is designed to pass once through an image predicting what objects are present and where simultaneously. This means that YOLO is substantially faster than older object detection networks. It is also more generalizable as it looks at the whole image as opposed to simply what was in the bounded box when classifying. For the same reason YOLO also makes fewer background errors, where a piece of background is misidentified as an object, than other detection networks (more than 50% less than R-CNN for example).

[6] https://captain-whu.github.io/DOTA/dataset.html
[7] Redmon et al., 2016 (https://arxiv.org/pdf/1506.02640.pdf)

YOLO starts by dividing each image into an SxS grid of cells. Each cell then predicts B bounding boxes and confidences associated with those boxes. These confidences are simultaneously the model's confidence that there is an object in the box and how confident it is in the box's boundaries. Each grid cell also produces a separate class probability for each class of object that is being modeled. These probabilities assume that there is an object in the box.

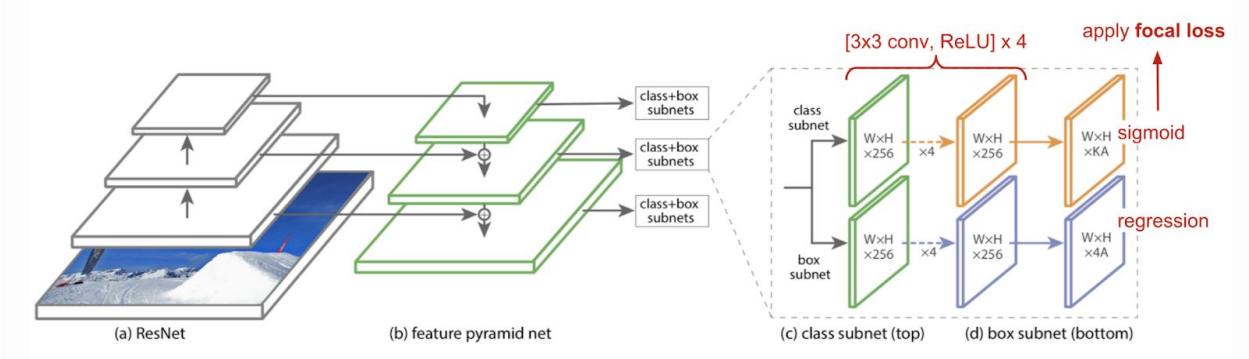


The standard YOLO model follows the architecture of the earlier GoogLeNet, using 24 convolutional layers followed by 2 fully connected layers. Due to training time concerns and model size concerns we elected to instead use Fast Yolo, which has only 9 convolutional layers. The remainder of Fast Yolo is identical to the standard model.



Retinanet is a single-stage detector that aims to recognize and process dense objects. Retinanet implements a pyramid structure in extracting features from the image. This feature pyramid[8] is a collection of several convolutional layers that promote the models ability to consider features at multiple scales. This is crucial in detecting multiple classes of objects that appear in various sizes as well as considering images of different aspect ratios. What sets Retinanet apart from other methods is the implementation of focal loss. This loss function addresses the class imbalance between the objects of interest and the background pixels. Essentially, a greater focus is placed on the regions and objects that are more difficult to classify by promoting a higher weight whereas background weights are discounted.

[8] Lin et al., 2017



Experimental Setup

A significant initial effort was made to access, ingest and process the training materials made available in the DOTA dataset. The annotations describing the image position of the various object classes comes in a structured text format. The several classes of objects offered in DOTA were filtered to capture only the small vehicles as that is the target of this analysis. All small vehicle annotations were extracted from text files and reformatted to reflect a horizontal bounding box by factoring out the horizontal and vertical extent values for each object. As stated, DOTA provides oriented bounding box annotations. The use of these annotations will be discussed in the conclusion. The annotations and image path variables are then compiled in a structured data frame and saved to be ingested for modelling.

A pytorch implementation of Retinanet by GitHub user yhenon[9] is utilized in modeling this object detection use case. This particular implementation is tested and formatted to be conducted using the COCO[10] dataset. COCO is a large scale object detection dataset with over 80 object classes that focus more on everyday objects. This type of imagery differs from the type of analysis involved in overhead imagery in the aspect and size of objects. The modules and methods in this repository are slightly modified and compiled to develop a Retinanet implementation to the small vehicle annotations in DOTA.

Retinanet uses a ResNet[11] backbone in building the model architecture. Choosing which version of Resnet to use will determine the depth and complexity of the model. A comparative analysis will be conducted by training models of different layer depths using ResNet18, ResNet50, and ResNet101. The implementation of Retinanet will process a full batch of training images and continually calculate the classification, regression and overall loss throughout training. A validation set of images and annotations is held out to evaluate the performance of the model at each epoch by calculating the mean average precision (mAP). In analyzing the accuracy of an object detection algorithm given the bounding box annotations and class of object, the metric of

[9] https://github.com/yhenon/pytorch-retinanet
[10] https://cocodataset.org/
[11] K. He, X. Zhang, S. Ren, and J. Sun, 2015

intersection over union (IoU) is first calculated. IoU examines the amount of overlapping area between a target bounding box and the predicted bounding box. Given a threshold hold value between 0-1, the predicted bounding box will be determined to be a valid prediction for an object area. Further, the classification for object type will be considered in evaluating the accuracy of the prediction. To avoid overfitting in the Retinanet model we implemented early stopping with a patience of 5. This helped to avoid overfitting and extrapolation by stopping the updating of the coefficients when the validation set accuracy began to decline.

We decided to use YOLO as a baseline model for Retinanet to benchmark against. YOLO seems to be a fairly standard object detection/classification model and is often used as a benchmark. It is also the smallest of the standard benchmarks and we had concerns about model complexity and training time. As discussed in section 3, due to training speed and computation time concerns we chose to use the Fast-YOLO as opposed to the full YOLO model. Fast-YOLO uses only 9 convolutional layers instead of the 24 used by standard YOLO. This is a clear trade-off from accuracy to speed. We used the implementation of YOLO (both standard and fast) put together by GitHub user Postor. This implementation had the advantage of being specifically developed to be used on the DOTA dataset, and so using it in our project was fairly straight forward. Even so, the training of Fast-YOLO had to be manually halted after 24 hours and the accuracy was extremely low.

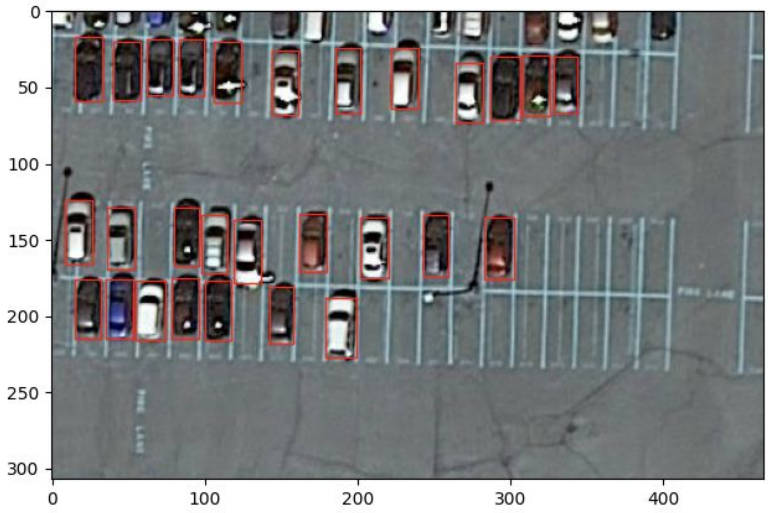
Results

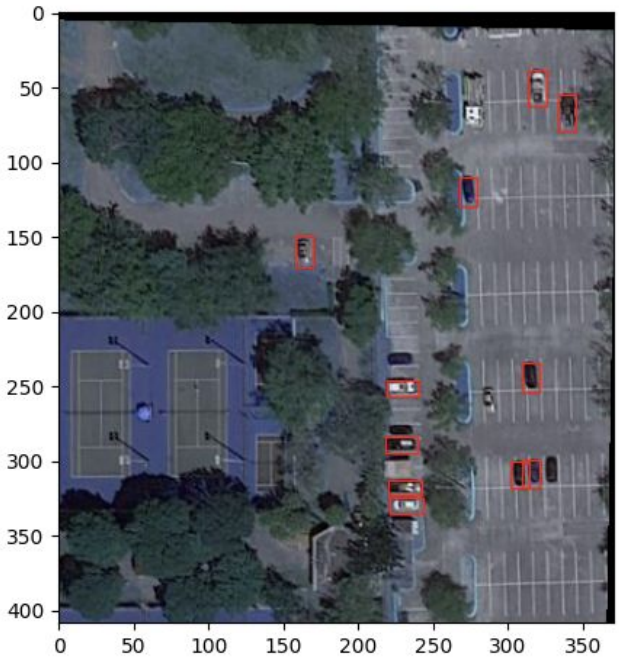
Initial modelling efforts using Retinanet produced far less than acceptable results. It was later understood that the size of the images being used in this implementation were too large for the convolutional layers to process the micro-scale of the small vehicle objects in the images. The dimension of the images from DOTA ranged from vertical and horizontal dimensions of approximately 200 pixels to approximately 20,000 pixels. Retinanet is optimized to process images in smaller dimensions so, in response, an additional preprocessing method was developed to slice the images into smaller dimensions and appropriately refactor the annotations to reflect the positioning within the smaller image slices.

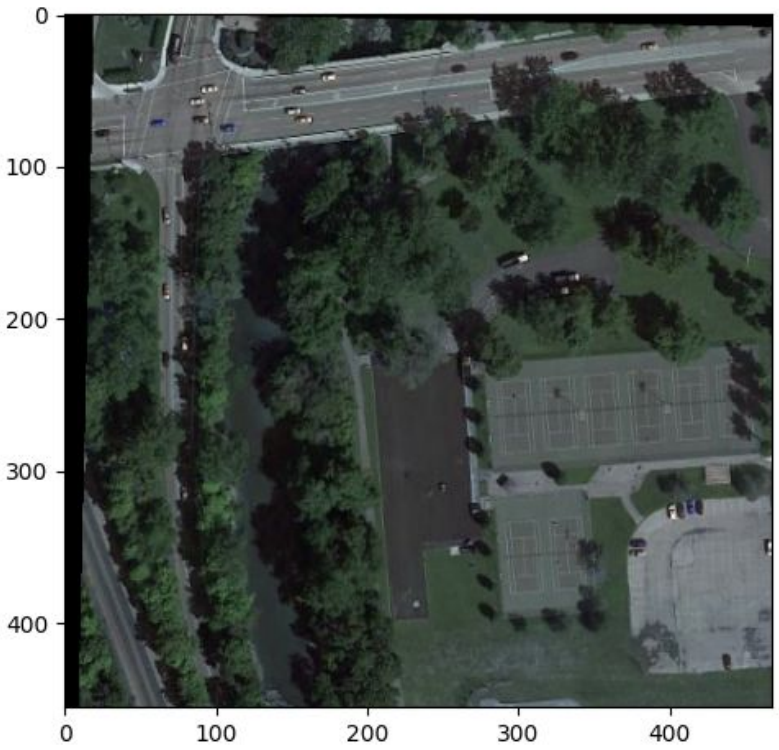
Model	Training Time	Inference Time	mAP
Retinanet - Resnet18	00:03:15:00	127.2	0.44
Retinanet - Resnet50	00:05:30:00	178.9	0.47
Retinanet - Resnet101	00:06:45:00	200.9	0.55
Fast YOLO	01:00:00:00[12]	26.3	.0046

[12] The model fit was halted after 24 hours because of concerns that the google GPU would be turned off.

In reviewing the performance of the Retinanet implementations, there is a clear tradeoff when it comes to speed and accuracy. The decreased performance is implied in understanding the added complexity and processing requirements of the additional layers in the Resnet architecture. The accuracy as measured by mAP is encouragingly increasing with the added depth. However, these measures of accuracy are slightly less than acceptable when compared to the DOTA leaderboard in classifying small vehicles which is topped at 0.81. Though these results leave much room for improvement, the abbreviated effort in compiling this modelling effort has prompted a collection of future improvement efforts. In examining the following visual representations of predictions on the test set of images, there is clear loss of ability when it comes to object size.







Visual inspection of these three images with displayed predictions shows the falloff as aspect ratio decreases. Late efforts in cropping images to smaller dimensions did not preserve the assumption that vehicles and objects would be seen in similar aspects when cropped to the same dimension.

The accuracy of the YOLO model is substantially lower than the various iterations of Retinanet. YOLO is designed with the idea that there is a single object, or a piece of a single object, in each of the boxes it divides the image into. The DOTA image set is so object dense that there can be many objects in each box which makes it very difficult for YOLO to accurately draw boundaries and classify objects. The normal solution to this would be to increase the number of slices (decreasing the size of each box and increasing the likelihood that there is only one object in the box). The problem with this approach is that the training time (and inference time) of the model varies with the square of the number of slices and so even small increases would greatly increase the already long training time.

Summary and Conclusions

From our analysis it seems reasonably clear that RetinaNet vastly outperforms Fast-YOLO, with the Mean Average Prediction being several orders of magnitude higher for all 3 different versions of RetinaNet. Fast-YOLO has a known accuracy tradeoff relative to standard YOLO, so a possible extension to this analysis would be to compare YOLO to RetinaNet. We could not do this comparison because of hardware concerns. It is worth noting that Fast-YOLO is much

faster to analyze new images than RetinaNet, so it may be preferable in certain cases, such as when a much larger cluster can be used for training and there is a strict SLA on model predictions. Finally, it is worth considering that YOLO was designed to simultaneously identify objects and classify them by breaking the image into boxes and evaluating the likelihood that a single object is in a given box. The collection of images is so object dense that this assumption does not hold. Between YOLO and Retinanet, the latter is clearly the choice approach in modelling this particular use case.