CS230

Object Detection with Satellite Imagery: Using YOLO to Detect and Localize Objects in Aerial Satellite Imagery

Canyon Robins
Computer Science
Stanford University
crobins@stanford.edu

Jaydon Krok
Computer Science
Stanford University
jkrok@stanford.edu

Abstract

At this very moment, there are just under 5000 satellites orbiting the planet. In 2018 we saw an increase in the number of satellites compared to 2017 of 4.79%, and this number is expected to continue to increase in coming years[1]. This increase in satellites will inevitably bring with it an increased proliferation of satellite imagery, which can be used to solve global-scale problems from tracking the effects of climate change and allocating resources during natural disasters, to predicting traffic flows in real time and tracking shipping resources around the world. Data at this scale can only be efficiently analyzed automatically, but until recent advances in deep learning this was not possible. In this paper, we set out to detect and localize 15 distinct classes of objects using a dataset containing aerial images obtained from satellites. Using transfer learning with two YOLO Models, we output the predicted classes and corresponding bounding boxes for each object contained in the given satellite image. We were able to achieve mAP for our best class of 0.87, but our mAP over all classes was only 0.1336, which is less than desired performance. We were able to identify clear next steps, and our model architecture shows promise for stronger performance with future work.

1 Introduction

Already today, far more satellite imagery is being collected than can be usefully analyzed with current technologies. Private companies such as SpaceX and Blue Origin are increasingly making it cheaper and more accessible to launch satellites, and governments like India and China are working to build their own space programs. This means we can expect the number of satellites in orbit to steadily increase in the coming years. The increase in satellites will almost certainly correspond with an increase in satellite imagery. These images, analyzed correctly, can then help solve global problems from resource tracking and allocation to scientific research. Many applications require the ability to correctly label significant features such as building footprints and roadways. As we transition into having bigger and more complex datasets, it is becoming increasingly important for us to have an automated system to detect objects from the troves of satellite images at our disposal. This is because it is no longer possible to manually comb through the millions of satellite images taken each day. In this project we attempt to label 15 distinct classes of objects using a dataset containing aerial images obtained from satellites. Our dataset consists of 1869 satellite images containing 15 classes. Using transfer learning with two YOLO Models, we output the predicted classes and corresponding bounding boxes contained in the given satellite image.

CS230: Deep Learning, Winter 2018, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

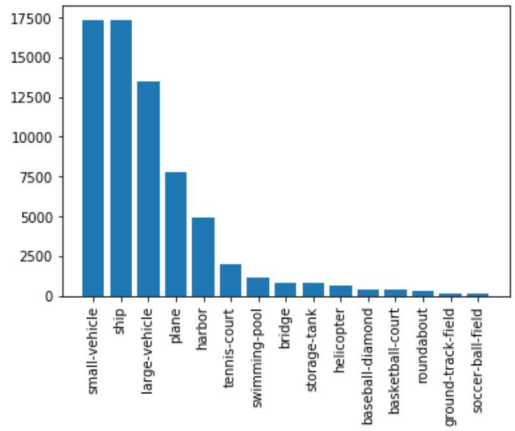
2 Related work

Before Redmon et al. published their seminal YOLO paper in 2015[2], Object detection was mainly accomplished through models such as DPM and R-CNN. YOLO was shown to be much faster than these other models because of it's ability to process a full image in a single neural network. Since then YOLO has been improved to it's current version of YOLOv3[3] which increases accuracy by using a bigger network while still not sacrificing too much computational speed. YOLO has been used in various object detection tasks such as pedestrian detection in video surveillance.[4] The authors of that paper found that YOLO struggled to distinguish small close human figures in a group. They found that gradually changing the network architecture somewhat improved YOLO's performance on these clustered small groups of people. YOLOv3 can also struggle with high resolution input images because it down samples the images to a lower resolution which in turn makes it difficult to detect small or distant objects[5], thus it can sometimes be better to not down sample the high resolution image if you are trying to detect small or distant objects. Finally Radovic et al. showed that after training for 45 000 iterations and tuning their YOLO model's hyperparameters it is possible to achieve a high detection and classification accuracy using YOLO on aerial imagery (84%)

3 Dataset and Features

For our project we will be using the DOTA satellite aerial images dataset for object detection[7]. This dataset contains 1869 aerial images from different sensors and platforms. Each image is of the size in the range of 800 x 800 to 4000 x 4000 pixels and contains objects with a variety of scales, orientations, and shapes. We have 15 associated classes representing different objects to be detected such as a baseball diamond, small vehicle or a large vehicle. The split for how many times each class appears in our dataset can be seen below:

Figure 1: *number of occurrences of each class in dataset*

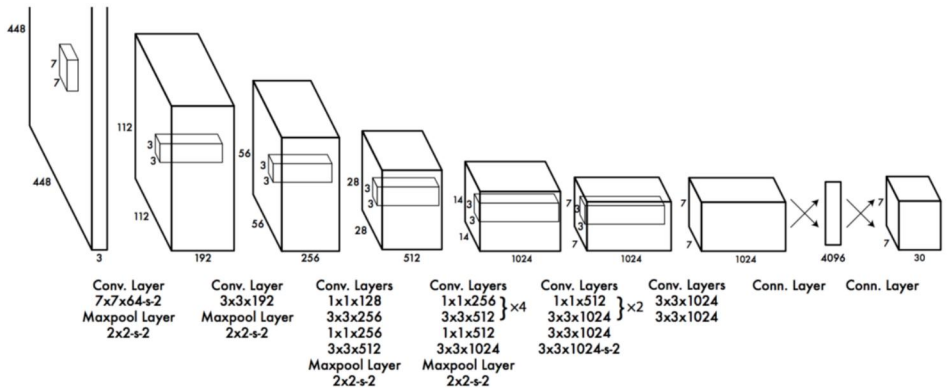


The bounding boxes for these objects are skewed quadrilaterals encoded as x, y pairs (which will mean lower IOU for the YOLO model in many cases). We have pre-processed these images to alter these skewed labels to the standard YOLO label format. Our original data also came with labels which described whether the objects to be detected in the image are difficult or not to detect. To improve our performance given our time constraints we removed all the images containing difficult examples from our dataset (about one-third). We also wrote code that crops and flips certain images, improving our detection of certain classes. We then took these altered images and added them to our training data. We shuffled the resulting images and used a 60/20/20 train/val/test split.

4 Methods

We used transfer learning on two darknet models[8]: As a baseline training model we used a 15 layer "tiny" YOLOv2 and for our main model we used the "regular" 30 layer YOLOv2 model.

Figure 2: *standard YOLO model*



We used transfer leaning on the darknet implementation of YOLOv2 model. The darknet model was pre-trained on ImageNet. We modified the codebase and architecture of the darknet YOLO model to account for our 15 classes rather than the original 80 classes. We re-configured our image resolution to 718x718 (1024 x 1024 for our tiny YOLOv2 architecture) and trained with batch size of 64 and subdivision size of 8 on a 30 layer network (15 layers for tiny YOLOv2). We used the loss function given in the original YOLO paper[2]:

Figure 3: *YOLO loss function*

$$
\begin{aligned}
&\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\
&+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] \\
&+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2 \\
&+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2 \\
&+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2
\end{aligned}
$$

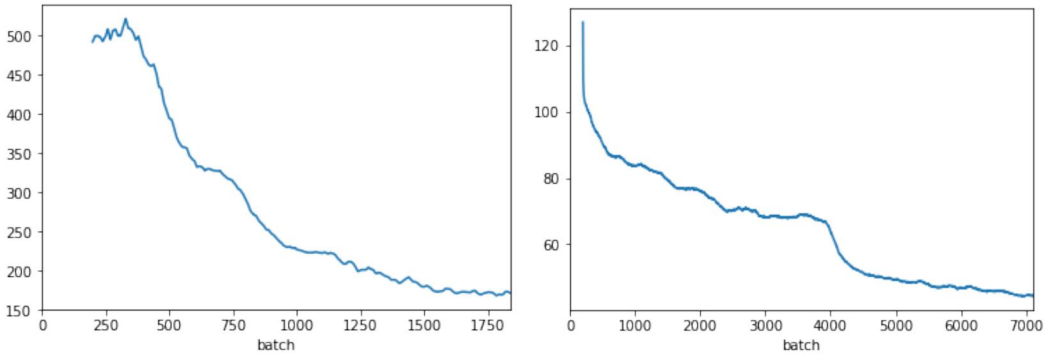
This is the sum-squared error between the predictions and the ground-truth. This loss comprises the classification loss, the localization loss (errors between the ground truth and predicted boundary box) and the confidence loss (how confident we are that there is or is not an object in our boundary box.) [9]

5 Experiments/Results/Discussion

After the aforementioned data cleaning, we decided to implement a tiny YOLOv2 model with only 15 layers and see what results we could achieve. We stopped training our model after about 2000 iterations. After inspecting our loss of our model, we noticed our model was under-fitting our data. We therefore decided to proceed with a larger network of 30 layers. To accommodate the larger number of parameters, we downsized the input image resolution from 1024 x 1024 (for the tiny model) to 768 x 768. We trained this model for 7000 iterations. As expected the larger model

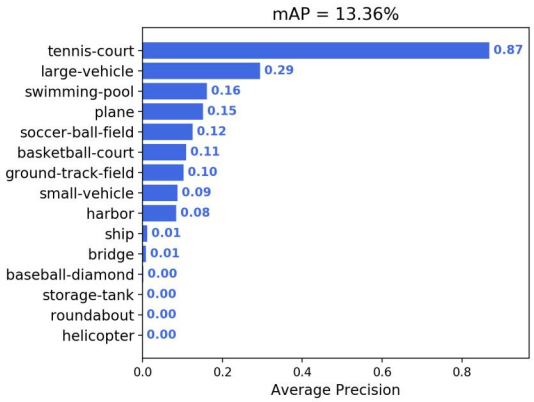
had a lower loss of 46 compared to the tiny model loss of over 182. The standard model was also much better at class recognition specifically than the tiny model. When training the tiny model we found that mini batch sizes of 64 with 2 subdivisions works well. This is probably because the tiny architecture is much smaller than the regular YOLOv2 architecture and thus we can process more images at a time with our GPU. When training the regular YOLOv2 model, we found mini batch sizes of 64 with 8 subdivisions works well with the regular YOLOv2 model. We furthermore found that using a learning rate of 0.05 with momentum of 0.9 worked well for our model.

Figure 4: *Tiny YOLOv2 and regular YOLOv2 training loss, respectively*



A common evaluation metric for object detection is mean Average Precision (mAP), which is the average precision of the maximum precisions at different recall values. The graph below shows our mAP for our test set

Figure 5: *YOLO mAP by class*



Since there is an online leaderboard of other models' performances on our dataset[7] We can compare our results to see how well our model did. Our model's overall mAP of 13.36% puts us under 0.04 percentage points away from making the 77 person leaderboard. Most models on the leaderboard used RCNN's or other models besides YOLO. However our average precision for the tennis-court; large vehicle; soccer-ball-field; basketball court; swimming pool were in the range of results on the leaderboard. Our precision values are at the bottom of the leaderboard. However we only trained our model for 7 000 iterations. It is recommended to train YOLO models for 2000 iterations for each class you have[10] which in our case of 15 classes, would mean an ideal number of iterations to train would be 30 000 iterations. This means we only trained just under 25% of the ideal training time. We are confident with more training we would improve our scores and make it higher on the leaderboard.

Figure 6: *High mAP predicting tennis courts*

Our model does surprisingly well on the Tennis-Court class, this is most likely the case because there is a very distinct color difference between the tennis-courts and their surrounding borders. They are also very uniform in shape, are often coupled side to side in our dataset and are relatively large objects compared to some of the other class objects such as boats (in which our model performs poorly.) We also perform poorly on small and clustered objects. This phenomenon was mentioned in the original YOLO model to occur because YOLO imposes strong spatial constraints on bounding box predictions and thus can only predict a limited number of small objects if they appear close together.

Figure 7: *Low mAP predicting vehicles*

6 Conclusion/Future Work

We found that though YOLO is a fast and versatile algorithm, it is not aptly suited in its current form for efficient object detection of aerial satellite images. This is mainly because of the nature of satellite images. They have high resolution and can have small objects clustered close together and in various orientations. We did manage to achieve a mAP of 13.36%, after training our model for only 7000 iterations, which put us close to some otherYOLO models that have trained on the same dataset.[7] Since we performed poorly on most of the classes that are rarely represented in our dataset. Next, we would try using cropping, flipping, and other augmentation tricks to balance the relative distribution of classes in our dataset. Furthermore if we had more time and computational resources to do tackle the problem, we would also train our model for 30 000 iterations and possibly explore tackling this problem with another model such as faster R-CNN.

7 Acknowledgements

This project was a great learning experience for both of us and we thoroughly enjoyed the challenge of implementing out first deep learning project. We would like to thank our project mentor Weini Yu for all the help, advice and guidance she gave us over the duration of the course. We would also like to extend this gratitude to Andrew Ng, Kian Katanforoosh and the rest of the CS230 course staff.

8 Contributions

Both group members contributed equally to the project. Canyon was in charge of data handling, augmentation and visualization and Jaydon was in charge of training the YOLO models.

9 GitHub Repo and Drive Link for Project

https://github.com/canrobins13/230_project_final_code
https://drive.google.com/drive/folders/1Rh4lhxjAltU50YuRwC5hQCIvW2KznL1G?usp=sharing

References

[1] “How Many Satellites Are Orbiting the Earth in 2018?” Pixalytics Ltd, 22 Aug. 2018, www.pixalytics.com/sats-orbiting-the-earth-2018/. Accessed 19 Mar. 2019.

[2] Redmon, Joseph, et al. "You only look once: Unified, real-time object detection." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

[3] Redmon, Joseph, and Ali Farhadi. "Yolov3: An incremental improvement." arXiv preprint arXiv:1804.02767 (2018).

[4] Molchanov, V. V., et al. "Pedestrian detection in video surveillance using fully convolutional YOLO neural network." Automated Visual Inspection and Machine Vision II. Vol. 10334. International Society for Optics and Photonics, 2017.

[5] Růžička, Vít, and Franz Franchetti. "Fast and accurate object detection in high resolution 4K and 8K video using GPUs." 2018 IEEE High Performance extreme Computing Conference (HPEC). IEEE, 2018.

[6] Radovic, Matija, Offei Adarkwa, and Qiaosong Wang. "Object recognition in aerial images using convolutional neural networks." Journal of Imaging 3.2 (2017): 21.

[7] “A Large-Scale Dataset for Object Detection in Aerial Images.” DOTA, captain-whu.github.io/DOTA/dataset.html.

[8] Redmon, Joseph. Darknet: Open Source Neural Networks in C, pjreddie.com/darknet/.

[9] Hui, Jonathan, and Jonathan Hui. “Real-Time Object Detection with YOLO, YOLOv2 and Now YOLOv3.” Medium, Medium, 18 Mar. 2018, medium.com/@jonathan_hui/real-time-object-detection-with-yolo-yolov2-28b1b93e2088.

Github Repositories used:

[10] Ringringyi. “Ringringyi/DOTA_YOLOv2.” GitHub, 1 Sept. 2018, github.com/ringringyi/DOTA_YOLOv2.

https://github.com/ringringyi/DOTA_YOLOv2

[11] Pjreddie. “Pjreddie/Darknet.” GitHub, 14 Sept. 2018, github.com/pjreddie/darknet.

https://github.com/pjreddie/darknet

[12] AlexeyAB. “AlexeyAB/Darknet.” GitHub, 19 Mar. 2019, github.com/AlexeyAB/darknet#how-to-train-to-detect-your-custom-objects.

https://github.com/AlexeyAB/darknet#how-to-train-to-detect-your-custom-objects

[13] Cartucho. “Cartucho/MAP.” GitHub, 16 Mar. 2019, github.com/Cartucho/mAP.

https://github.com/Cartucho/mAP

[14] Jessemelpolio. “Jessemelpolio/Faster_RCNN_for_DOTA.” GitHub, 7 July 2018, github.com/jessemelpolio/Faster_RCNN_for_DOTA.

https://github.com/jessemelpolio/Faster_RCNN_for_DOTA

[15] Captain-Whu. “CAPTAIN-WHU/DOTA_devkit.” GitHub, 19 Mar. 2019, github.com/CAPTAIN-WHU/DOTA_devkit.

https://github.com/CAPTAIN-WHU/DOTA_devkit

[16] Python Libraries: TensorFlow, Keras, PyTorch, NumPy, Pandas, MatPlotLib, Seaborn, OpenCV