xView

Jakub Smatana, Ondrej Kužlík

June 18th 2023

1 Description of the algorithm and theory, similar projects

1.1 xView

The xView dataset (`http://xviewdataset.org/`) is a large-scale satellite imagery dataset that was developed for object detection and localization tasks. It was created to advance the development of computer vision algorithms and systems for humanitarian and disaster response efforts.

The primary focus of the xView dataset is on detecting and localizing objects of interest, such as buildings, vehicles, and other infrastructure elements, within the satellite images. It includes annotations for over 60 object classes, including aircraft, helipads, bridges, and more. The annotations are provided as bounding boxes, indicating the location and extent of each object in the images.

1.2 Faster R-CNN

Faster R-CNN (Region-based Convolutional Neural Network) is an object detection algorithm that builds upon the original R-CNN and its variants.

The key idea behind Faster R-CNN is to integrate the region proposal generation and object detection stages into a single end-to-end network. It achieves this by introducing two main components: a Region Proposal Network (RPN) and a Fast R-CNN network.

- Region Proposal Network (RPN): The Region Proposal Network (RPN) in Faster R-CNN generates candidate object regions by sliding a window over the image and predicting if there is an object present. These proposals are then used for further processing in object detection.
- Region of Interest (RoI) Pooling: The proposed regions from the RPN are used to extract fixed-sized feature maps from the convolutional feature map using a technique called RoI pooling. These feature maps are then fed into the next stage for classification and bounding box regression.

- Fast R-CNN: The Fast R-CNN network takes the RoI feature maps as input and performs object classification and bounding box regression. The network is trained end-to-end using multi-task loss, which combines the classification and regression losses.

1.3 Vision Transformer (ViT)

VisionTransformer is a neural architecture that utilizes attention layers to learn relationships between parts of input images. A version of this architecture, modified for object detection task, is more generally known as DETR (shorthand for Detection with Transformers)[1].

The DETR model utilizes a convolutional backbone (most commonly ResNet) with given pre-trained weights that is used to get feature maps from a given input image. The feature maps are subsequently flattened to create a sequence of them, which is then fed into the encoder. The encoder embeds these into so-called object queries, which serve as positional encodings. Each encoding is then added to the input of attention layers. The purpose of each object query is to look for a particular object in the input. The output of the decoder is fed into two detection heads, first is to classify each of the object queries into one of the classes (or into a special no-class value, if the object does not fit any of the known classes), the second is to perform bounding box regression.

The DETR uses bipartite matching loss, where each of the predicted queries is matched with the best fitting ground truth class and bounding box. Subsequently, L1 and IoU loss are used to optimize the model.

1.4 Similar projects

Several projects attempting to use deep learning methods to detect objects in satellite imagery already exist. Amongst the foremost and most cited one is YOLT: You Only Look Twice, which uses an adapted version of the YOLO algorithm [5]. Other approaches include various modifications to CNN based approaches, most commonly using some version of R-CNN [3]. The goal of the project was to attempt to recreate these attempts, as well as attempt to include another, less used approach in Vision Transformers.

2 Description of the implementation

2.1 Faster R-CNN

The dataset consist of roughly 600 000 objects in 800 images of different sizes. First training was done on the original pictures, but resized to 2400x2400. The network as it was implemented had hard time to distinct small object so the train images were then split to size 580x580 with some filtration of very small objects.

The implementation of the Faster R-CNN is done in PyTorch library using the tutorial on fine tuning the object detector[4]. MobileNetV2 features are used

as a backbone and ROI Align as pooler. Furthermore some preprocessing was done to images such as normalization, augmentations - collor jitter, grayscale, gaussian blur, auto contrast. SGD was used as an optimizer with learning rate scheduler to adjust learning rate during training.

Data were split into training, validation and testing dataset with 80:10:10 ratio. Training was performed on GPU for 20 epochs.

2.2 Vision Transformer

The same dataset was used to train the DETR. The implementation borrowed from a Huggingface implementation of the DETR, utilizing the included processor class to dynamically resize the images. During the project, it was discovered that the architecture does not reliably work with a high amount of object queries utilized, which led to the choice of a similar approach as in the R-CNN implementation, with input images being split into smaller parts before being processed. This was done in order to reduce the amount of objects per input image to the architecture.
The used amount of object queries was 250, however, this number might also have been too high and it could have been more optimal to stay at 100. However, this decision was taken in order to not having to split the image into way too small parts, which would in turn lead into unachievable learning times considering the long time periods the architecture needs.

The data were then split into a training and testing set, with the ratio used being 80:20. As mentioned previously, time required to train the architecture is long, especially considering the larger nature of inputs. Therefore, a default period of 5 epochs was chosen for the training, with the potential of training for more if the model showed potential.

3 Installation and startup instructions

3.1 Faster R-CNN

The *train_pytorch.py* file is the python script that was used for the training. However, without the very large dataset that is not included in the files(only small sample for testing) the training will not work. The model is included in the *models* folder.

The evaluation can be done by running the *faster-rcnn-results* notebook with all libraries installed. The *requirements.txt* file can be used to install libraries.

3.2 Vision Transformer

No special installation necessary, aside from including the data in `.tif` format inside a `train_images` directory, as well as the presence of the `samples.csv` file in the work directory. All of these should be either included or available from the xView dataset homepage at request. The model is then ready to be run by

simply running the cells in the notebook, however, be advised as it takes a long time to run.

4 Description and results of the evaluation

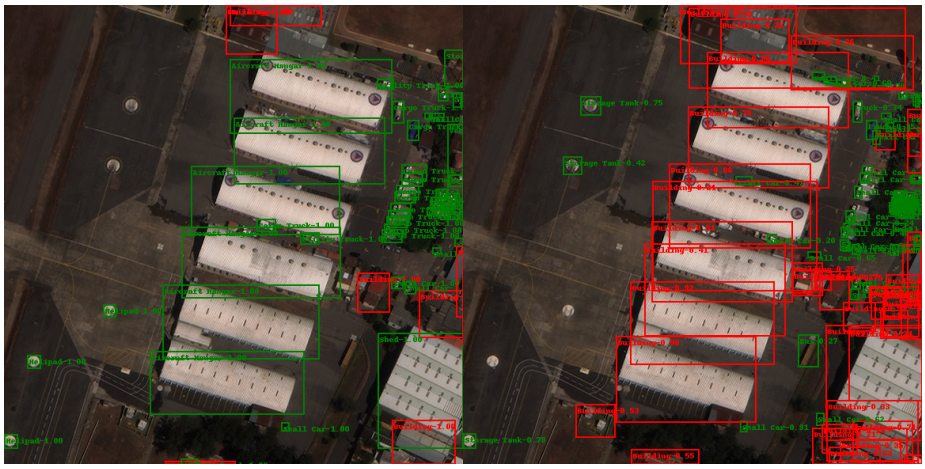
4.1 Faster R-CNN

4.1.1 Image results

The results of the Faster R-CNN were really mixed. On the one hand, the visual evaluation of the results were pretty decent. The model predicted objects in places, where they were and in images with no objects returned also empty prediction. However on the other hand, mAP score was terrible because there were so many objects on each picture and the model was not guessing the classes correct in some instances. Other reason for poor mAP score was a fact, that the FPN network was not fine tuned as I expected it to be and some extra steps could have been done to achieve better results.

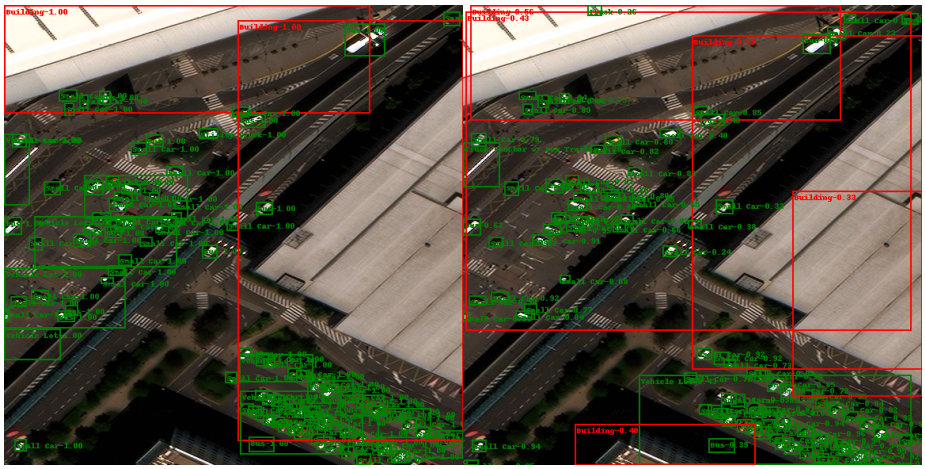
In the result folder, I included side to side images of the prediction and ground truth for some typical pictures. In some of the cases, the model predicted classes like *Bus* instead of *Truck* or *Building* instead of *Shed*. These classes are very similar and the evaluation is then completely ruined. However, if the models is used to predict buildings, it works in decent fashion.

As I saved the model that was trained on objects that were larger then 8x8 pixels, it does not predict vast amount of small objects.



(a) Results of Faster R-CNN, ground truth on the left, predictions on the right

One of the biggest issues in the model was the class imbalance. In figure (a), the *Aircraft Hangar* was mistaken for *Building* and *Helipad* for *Storage Tank*.



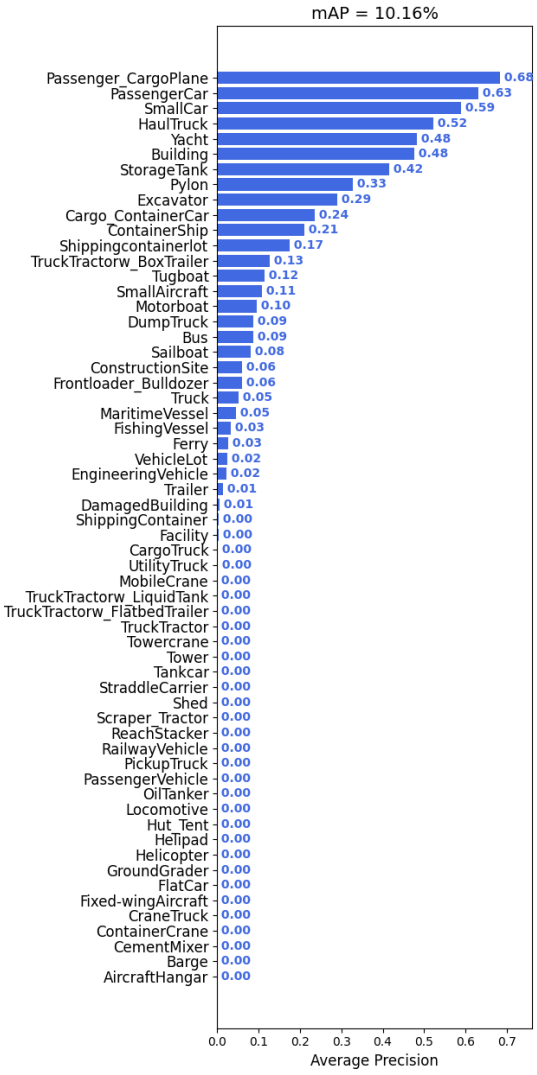
(b) Results of Faster R-CNN, ground truth on the left, predictions on the right

As you can see in the figure (b), the model predicted some *Buldings* that were not in ground truth, but had decent success with predicting the *Small Cars*.

(c) Results of Faster R-CNN, ground truth on the left, predictions on the right

The figure (c) shows that the *Buldings* can be predicted with high accuracy, the problem is with the classes that have small sample size.

4.1.2 Metric results



(d) mAP score over all classes

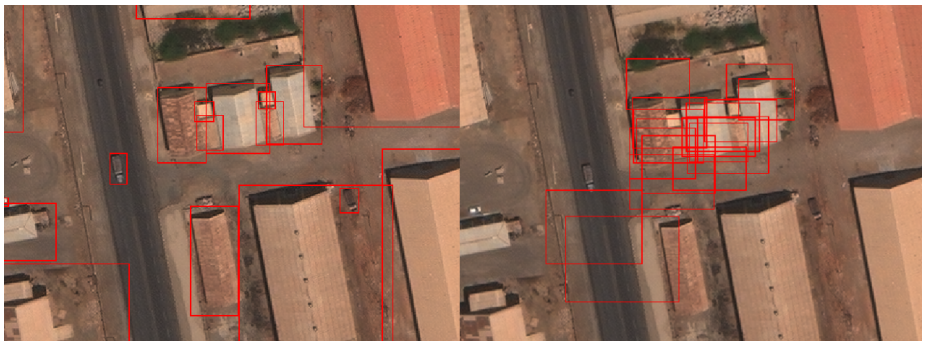
The overall mAP score over all classes with IoU threshold of 0.5 was 10.16%. The biggest reason of such a low mAP score was the class imbalance, which I was not able to overcome. But to sum it up, this very simple model works quite well, even though it has some simple parts as the image cropping or the class imbalance issues. The mAP score was calculated with external github code[2].

4.2 Vision Transformer

The implementation of the Vision Transformer was unfortunately more underwhelming than initially hoped. At first, I wanted to evaluate the basic transformer architecture, however, I switched to the DETR one since it was more attuned to the task at hand and there was not any work done using this architecture that I could readily find at the time.

Due to the unavailability of decent models of this kind in this area, the choice of the predetermined weights for the convolutional backbone of DETR was decided using a 'what works best' approach, ending up being the weights for the ResNet implemented in the original DETR paper. This also meant borrowing the same weights for the learnable parameters within transformers, determining that while they were not perfect for the task, they served as a better starting point than random weight initialization.

However, the results of this model, after learning for several epochs, have been quite bad. Utilizing the aforementioned processor classes methods for processing the output of the model, one can obtain information about the predicted bounding boxes, as well as the confidence scores for them. However, despite the training loss decreasing steadily throughout training, the resulting boxes all suffered from having low confidence (which even slowly decreased as the model learned for longer). Overall, the confidence scores of the ViT model, calculated by $p_{predicted\_class} * p_{predicted\_box\_has\_object}$ was only **0.26**.
In my approach, I have also tried utilizing the aforementioned external tool[2] for evaluation of mAP score. As may have been indicated by the confidence scores, the results of this tool have also been very subpar, with it reaching a 2.1% score. The reason for this is the general low performance of the model.



(d) Unsatisfying results of DETR model, ground truth on the left, predictions on the right

The results lead me to unfortunately have to consider this attempt at the implementation a failure. There are few key points to consider when trying to implement this successfully: getting sufficient computational power and resources to run this for many more epochs, as well as considering adding more augmentations to the data. Another approach would be to attempt to either

find better fitting weights for the convolutional backbone (or even attempt to use the convolutional layers from the other part of the project) and for the VisionTransformer itself, which would prove to be a better starting point, thus potentially reducing the required amount of time to learn better.

Consideration might also be given to another choice of learning function. While the bipartite loss used by the DETR model should theoretically work well, the model quite struggled with lots of small objects overlapping each other, ending up not regressing to any general kind of box.

In the end, I also think I went in over my head with this attempt. No experience with this kind of model has shown it to be rather difficult to debug, leading to lots of mistakes hindering the learning and progress. Having better insight into the model might also be useful in finetuning the project, though bettering the insight can also be considered one of the more successful outcomes of the project.

References

[1] Nicolas Carion et al. "End-to-end object detection with transformers". In: *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part I 16*. Springer. 2020, pp. 213–229.

[2] Cartucho. *mAP*. `https://github.com/Cartucho/mAP`. 2018.

[3] Austen Groener, Gary Chern, and Mark Pritt. "A comparison of deep learning object detection models for satellite imagery". In: *2019 IEEE Applied Imagery Pattern Recognition Workshop (AIPR)*. IEEE. 2019, pp. 1–10.

[4] PyTorch. *TORCHVISION OBJECT DETECTION FINETUNING TUTORIAL*. `https://pytorch.org/tutorials/intermediate/torchvision_tutorial.html`. 2023.

[5] Adam Van Etten. "You only look twice: Rapid multi-scale object detection in satellite imagery". In: *arXiv preprint arXiv:1805.09512* (2018).