Building Footprint Detection and Damage Assessment from Satellite Images

Max Alexandrov

December 2023

1 Detection of Buildings, Rubble, Planes and Cars

NOTE: all the functions described in this section can be found in the data proccesing file in the GitHub.

For detecting buildings, rubble, cars and planes, I chose to utilise a pre-trained YOLOv8x object detection model and fine-tune it on my data. Since no training data was provided in the GitHub repository, only the 6 images which I will refer to as the "test set", I created one from the datasets publicly available online on the Roboflow website. The data collection and processing stages are described in the section below.

1.1 Data Collection and Processing

For my final dataset, I combined existing labelled data for building, rubble, plane and car detection, with the txt files necessary for the YOLO model already pre-made. Although most scientific papers that study object detection usually acquire such data by themselves, collecting and generating it is time-consuming, which was impossible under my time constraints. Another advantage of the datasets I found is that they already come with augmented images, such as the addition of Gaussian and Median blurs, rotation and noise addition. Such augmentations help the model generalise better. My resulting dataset is available in the GitHub repository, and contains images and their respective labels, as well as the .yaml file necessary for running the model. The 4 classes in the dataset are not evenly split, with buildings and rubble comprising most of the dataset due to the nature of the images the model is to be used on (buildings and rubble taking up the majority of the image). Choosing the right data for this task was crucial, as the resulting model is heavily influenced by that factor more than anything else. Because all the datasets individually only had one class, I wrote a function that changed the class in “cars” to 1 and in “planes” to 2, "rubble" became class 3, while "building" remained classed as 0.

It is normally beneficial to use the YOLO model on images of the same size as the the ones it was trained on. This is not the case here, as most of the images in the training dataset are roughly of size 640*640, while the satellite images are more than double the size. To account for that, the test images can be split into two parts, and those images can be passed into the model individually. This approach was tried, but showed no improvements compared to the original images.

I also tried to convert my training and test images to grey scale, add more blur to the training set, sharpen the edges and change the image brightness. Neither of these augmentations helped improve the quality of the results. The functions to perform these operations can be found in the GitHub repository.

I did not have the time to create a proper validation set, but that would be part of the future work.

1.2 Model Training

I imported the pre-trained YOLO model from ultralytics and fine-tuned the model on my combined dataset. I trained it for 200 epochs, and readjusted the following hyperparameters from their default values:

1. Intersection over Union (IoU), changed from default 0.5 to 0.8;
2. Confidence, changed from default 0.5 to 0.7;
3. Cosine learning rate set to True.
4. Learning rate increased to 0.0005 from default 0.0001.

The model shows good results on images 2, 3, 5 and mediocre results on 6. Most buildings are detected, and large areas of rubble are picked up. All 3 planes are recognized in the last image, most of the buildings, however, are missed out. The cars are not picked up on any images due to their size on the test set photos. Perhaps, adding more cars to the training data could improve the performance of the model in car detection. The detection on images 1 and 6 (which appear to originate from Eastern Ukraine), however, is poor, with mostly rubble getting detected. That is due to the vast difference between these images and the training data. The buildings on images 1 and 4 are much smaller than the ones seen in the other images and the training data. Due to that fact, the model struggles to spot them. One possible solution to tackle this is discussed in the next section.

The next step is to generate the buildings' footprints. For the buildings in this test set, the task is trivial. Since buildings in this set are of rectangular shapes, and we already have the rectangular boxes extracted by the YOLO model for the detected buildings, we can simply plot them. Overlaying them with the original images is then trivial. However, this approach has a flaw. Although it works for our case, where the buildings are rectangular, it would fail if the shape of the building was, for instance, hexagonal. The reason for that being the training labels I used, which are always rectangular. Thus, the model detects the building itself, but not strictly its edges. A solution is mentioned in the section below.

2 Possible Problem Solutions

2.1 Improving detection on Images 1 and 4

Poor detection on images 1 and 4 is due to the vast difference between these images and the training data. The buildings on images 1 and 4 are much smaller than the ones seen in the other images and the training data. Due to that fact, the model struggles to spot them. One possible solution to tackle this issue is by augmenting the training data in a certain way. Since the model fails to spot smaller objects, in addition to the training data available now, we want to add an extra augmentation to each image at hand. We want to shrink the relative size of objects on every training image, but keep the image size the same as the original. It is important to note that we do not replace the current data, but are adding more to it. The process is described in more detail below.

For clarity, let us call the original set of training images we are given set A, and the augmented versions will be called set B. In addition to the already existing images A, we want to add a 'shrunk' version of each image in set B. Each image in A (which we can say, for simplicity purposes, has dimensions of 100*200) will get shrunk to size 50*100, and the resulting difference between dimensions 100*200 and 50*100 will get padded. Such images will form set B. Of course, the text labels for each image will get readjusted appropriately, as the proportion and location of the boxes relative to the image will change. Such technique will make the objects look naturally smaller in the images from set B, with the model still exposed to the original, 'normal' images from A during training.

However, this direct approach poses a problem. If we shrink the original images from set A, such operation will result in the loss of quality of the resulting images. Therefore, rather than keeping the originals and creating shrunk copies of them, we can upscale set A, hence creating set B, and then pad the original image in A. For example, an image from set A of dimensions 100*200 will first be copied and upscaled to 200*400 using a pre-trained upscaling neural network from the Open-CV library, such as LAPSRNN. Such neural networks upscale images without the loss of quality. The original image will then get padded from shape 100*200 to 200*400. This trick will ensure that we still introduce the model to relatively smaller objects, without losing image detail. However, I was not able to implement this approach due to the fact that upscaled images are too big to fit into memory with the GPU I was using to train the model. However, with a bigger GPU, such approach could be employed to solve the problem.

2.2 More Accurate Footprint Detection

One possible solution to detect the edges of the building rather than the building itself is to perform a two-stage analysis process. Firstly, detect the building itself using a YOLO model and get the coordinates of where it is on the image. Then, once the location is known, one can perform an edge detection algorithm on that location (for example, the Sobel algorithm). Using the algorithm on a specific area, rather than the whole image, should make edge detection on satellite images doable. Unfortunately, I did not have the time to implement this method.

3 Assessing the Damage to the Buildings

While the YOLO model more or less successfully detects whole or destroyed buildings (rubble), it does not consider damaged buildings. The only way to teach the model how to do it is to introduce it to labelled damaged buildings during training and fine-tune it on such data. One such dataset that I found is the xView dataset from ultralytics [2], which has a 'damaged building' class in it. I tried this approach, but could not effectively train the model, as the training images are too big. This meant that I could only fit two images per batch, which wasn't enough to train the model effectively due to the gradient being unstable. On top of that, the xView dataset does not provide the information about the level of damage a building has sustained. A better approach is to acquire smaller images with a damaged building class rated from low to high. Such images can be found in [1] upon request. I have written the main author and requested it, but have not got a response yet. However, with such data acquired, once the model has been exposed to damaged buildings during training, it would be able to generate tagged images of damaged buildings, and could categorise the damage from 'low' to 'high'.

References

[1] KIOS. Rescuenet-yolo8 dataset. https://universe.roboflow.com/kios-alpsc/rescuenet-yolo8 , apr 2023. visited on 2023-04-07.

[2] Darius Lam, Richard Kuzma, Kevin McGee, Samuel Dooley, Michael Laielli, Matthew Klaric, Yaroslav Bulatov, and Brendan McCord. xview: Objects in context in overhead imagery, 2018.