Project 3
Object Detection, Semantic Segmentation, and Instance Segmentation

Due date: 23:59 11/4th (2021)

0. Acknowledgements

The goal of this assignment is to get hands-on experience designing and training deep convolutional neural networks using PyTorch and Detectron2. Starting from a baseline config file and model, you will improve an object detection framework to detect planes in aerial images. You will evaluate the performance of your code by uploading your predictions to this Kaggle competition (Will launch later). Note that this assignment is SIGNIFICANTLY more challenging than the previous assignments. Please start early.

1. Instructions

Most instructions are the same as before. Here we only describe different points.

1. Please upload a pdf ({Your-SFUID}.pdf) and a zip package to Canvas as before. The zip package must contain the following in the following layout. **data** folder is large for this project. Please do not include the data folder.
    - {SFUID}/
        - lab3.ipynb (remove code output from the notebook, otherwise the file will be too large to be uploaded)
    - In addition, the CSV file of your predicted test labels needs to be uploaded to Kaggle.
2. Project 3 has 33 points.

2. Overview

The goal of this assignment is to get hands-on experience in training deep convolutional neural networks using PyTorch targeting three fundamental tasks of computer vision, object detection, semantic segmentation, and instance segmentation. Starting from a baseline configs, you will design an improved deep net framework to detect planes in aerial images and obtain the segmentation mask of each plane. You will evaluate the performance of your architecture by uploading your predictions to this Kaggle competition. One is expected to

search online, read additional documents referred to in this hand-out, and/or reverse-engineer template code.

Deep Learning Framework: PyTorch

In this assignment, you will use PyTorch, which is currently one of the most popular deep learning programming frameworks and is very easy to pick up. It has a lot of tutorials and an active community answering questions on its discussion forums.

Detection Framework: Detectrone2

Implementing a powerful object detection from scratch is hard and time consuming work. However, there are several open source frameworks in this regard which made it much easier to train and test the current famous detectors. In this assignment, you will use Detectrone2 which is powered by PyTorch. You can find the full documentation in this link. Part 1 has been adapted from a Detectron2 Beginner's Tutorial.

Google Colab Setup

You will be using Google Colab as before, a free environment to run your experiments. If you have your own GPU and deep learning setup, you can also use your computers. If you choose Google Colab, here are instructions on how to get started:

1. Open Colab, click on 'File' in the top left corner and select upload 'New Notebook'. Upload the notebook (lab3.ipynb) file.
2. In your Google Drive, create a new folder, for example, "CMPT_CV_lab3". This is the folder that will be mounted to Colab. All outputs generated by Colab Notebook will be saved here.
3. Within the folder, create a subfolder called 'data' and put the corresponding data files. First, copy train.json into the 'data' folder. Next, you need to copy many files under 2 directories named "train" and "test". Since there are too many files, instead of downloading files and uploading, it is highly recommended to use (shortcuts) instead. Open the following 2 links (train/ and test/), click their names, choose "Add shortcut to Drive", then specify your "CMPT_CV_lab3/data".
4. If you have a fast Internet connection, you can download and upload data files here.
5. It seems that reading the files directly from the google drive will highly affect your training procedure time, please read link1 and link2 in this regard to speed up your training.
6. Follow the instructions in the notebook to finish the setup.

This lab does not have a zip project_package file. All the links are in this hand-out. If one wants to run things locally, this link is the package file with the data.

Keep in mind that you need to keep your browser window open while running Colab. Colab does not allow long-running jobs but it should be sufficient for the requirements of this assignment (expected training time is about 20 minutes for the baseline).

Time management

The training of deep learning frameworks can take several hours to several months in different settings. Therefore, writing an efficient code is very important. In addition, you need to be careful when you are running your code in Colab as any disconnection will close your session. In this case, you need to run the initializations and import the packages again before going to other parts.

Make sure that you get benefits from all the options, for example, you can run all cells together by selecting "run all" option from the menu, or you can select multiple cells and just run them which is very helpful in case that you already trained your model and you want to skip the training parts and use checkpoints instead.

The given notebook is a template for helping you, but it is not necessarily the most efficient implementation. Feel free to change the code in case of improving the efficiency, but remember to mention the idea behind the modifications to the given parts in your report.

Part 1: Object Detection [6pts]

Dataset

For this part of the assignment, you will be working with the iSAID dataset (the above train/test folders contain data and you need not download data from iSAID website). This dataset consists of 655,451 object instances for 15 categories across 2,806 high-resolution images.

We have modified the standard dataset to create our own, which consists of 198 training images and 72 test images with just 1 category (Plain). The training dataset has labels for your training, and your trained model will predict answers for the test set for evaluation. For your better final performance, you should split your training data into the training set and the validation set, then tune your hyper-parameters as in the lecture. The results on the test set (in CSV format) need to be uploaded to the Kaggle competition to know your final performance. Note that the number of submissions to Kaggle is limited to a few times per day, so try to tune your model before uploading CSV files. Also, you must make at least one submission for your final system. The best performance will be considered.

You need first to write a data loader `"get_detection_data"` similar to `"get_balloon_dicts"` in the mentioned tutorial. This function processes the given dataset and returns a python list. The difference of the function inputs is that instead of the '`img_dir`' your function should get '`set_name`' ("train", "test", and optionally "val") as the input and process the corresponding data set. For the test set, you can put a condition that ignores the json file as 'test.json' does not exist, in this case, the annotations will be an empty list for the test images and the other values like height and width could be obtained

from the image file . You can also divide your training data to training and validation for your experiments.

For more details, please read this document regarding each keyword (like BoxMode).

After getting the dictionary from the function, remember to register your data and metadata in the `DatasetCatalog`. Finally, visualize 3 random samples of the training data using "`detectron2.utils.visualizer`" to make sure that the data is correct. This visualization is not required in your submission.

Note that some planes are missing in the annotations. In order to check if this is because of the dataset or your code, you can manually open the JSON file and check the number of bounding boxes for that image. Of course, the missing annotations in your visualization is ok and we do not consider it as a bug of your code.

Regarding the missing planes in the dataset, your network still should be able to overcome these noises and get the baseline result. However, the missing annotations could affect the final result. If you want to improve it, you can manually add the planes to the dataset or use other learning-based methods to solve the noisiness of the data.

Configs

There is a large collection of baselines trained with detectron2 which you can find in Detectron2 Model Zoo in addition to their config files and the pretrained models.

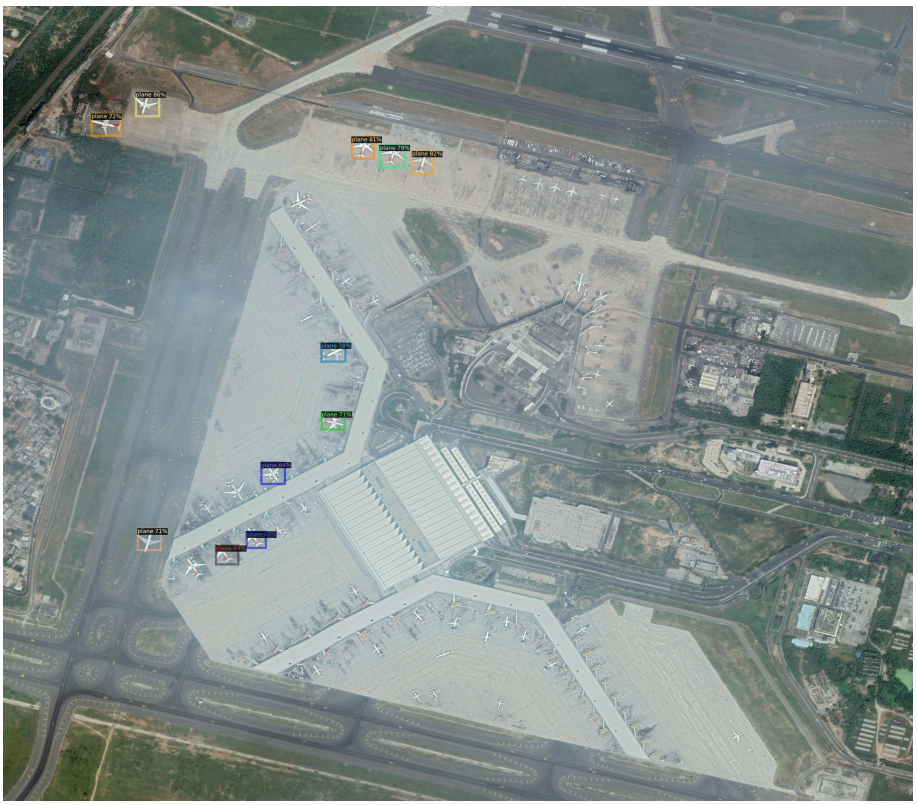
For this part of the assignment, we expect you to use "`faster_rcnn_R_101_FPN_3x.yml`" as the baseline config that you can use to run and get a baseline result with the following changes on the configs: `(MAX_ITER = 500, BATCH_SIZE_PER_IMAGE = 512, IMS_PER_BATCH = 2, BASE_LR = 0.00025)`. Modify these values to improve your results.

Training and Evaluation

You need to create an output directory and train the detector using a "`DefaultTrainer`" and the new configs. The training of the baseline should take about 20 minutes. Before that, create an output folder in the same directory to save the trained model and corresponding files.

After training, use "`COCOEvaluator`", "`build_detection_test_loader`", and "`SCORE_THRESH_TEST = 0.6`" to evaluate your model on the training/validation data. Consider the Average Precision (AP) at IoU=.50 (similar to PASCAL VOC) as the target metric. This value probably will be around 0.250 for the baseline without any improvement.

Finally, visualize 3 random samples of the test data as well as saving the output file of "coco_instances_results.json".



Improve your model

As stated above, your goal is to create an improved object detector by making framework and config choices. A good combination of frameworks and configs can highly improve your accuracy. For improving the detector, you should consider all of the following.

1. Data Processing. The given images are in very high resolutions. It is not a good idea to directly use this image, because planes would appear very small in the input that is passed to your ConvNet. So one way is to divide an image into smaller blocks, then pass each image block to ConvNet for training. Given a block, you need to look at the ground-truth bounding box information and keep only those that are inside the block. This means that at test time, given a high resolution image, you need to divide into blocks, and pass each block to ConvNet then merge resultant bounding boxes into the same file by some coordinate transformation. There are a lot of degrees of freedom here. You may wonder what is a good block size as planes may appear at different sizes depending on the images. Then, a natural idea is to try different block sizes. You may also wonder what to do if ground-truth bounding boxes are partially inside the block. What you should do is determined by the performance of the system. There is no correct answer here. You should explore any ideas you might have.

2. Data augmentation. Try using different transforms by writing custom Data Loaders.

3. Object Detection Method. Following the models in the MODEL_ZOO page. There are several options for choosing the method (Faster R-CNN, RetinaNet, RPN) in addition to

several options of architecture for each method. Considering the cons and pros of each method in addition to the training times. You need to pick the best one suitable for the task.

You can find more info on different object detection methods in the following links: [Link1. Link2]

4. Pretrained Models. You can also try to use the pretrained models which are provided in MODEL_ZOO and try to freeze different layers.

Finally, there are a lot of approaches to improve a model beyond what we listed above. Feel free to try out your own ideas, or interesting ML/CV approaches you read about. Since Colab makes only limited computational resources available, we encourage you to rationally limit training time and model size.

5. Hyperparameters tuning. You can use any values for your parameters like learning-rate, number of epochs, etc. Just remember to mention the changes in your report.

Part 2: Semantic Segmentation [10 pts]

The goal of this part is to implement and train a deep neural network for image segmentation task.

Dataset

You will work with the same dataset for this part by also considering the “segmentation” key regarding each plane in train.json. Each segmentation is provided with a list of pixel coordinates. You need to convert them in order to obtain the corresponding input image and ground-truth mask of each plane.
Convert all of the masks and cropped images to a fixed size (e.g. 128*128) to pass them to the network.

Network

We have provided an implementation of a sample network for the training. You need an Encoder to encode the features of the image and a Decoder to generate the new image (segmentation mask in here) from the encoded features. In this regard, the provided code consists of 3 different modules: `conv`, `down`, and `up` respectively as the conv layer, conv layer with max-pooling and conv layer with upsampling. `MyModel` class is predefined to use these layers.
You need to modify the network to improve the performance. The current network consists of only 4 layers, therefore, one way is to increase the number of layers of both encoder and decoder. One other possible way is to use skip connections between the layers. You can add the connections by modifying the above modules.

Loss Function

There are several loss functions for image segmentation (you can see a sample list in here). As the baseline, the Binary Cross Entropy Loss with logits should be working as we only have one class of objects in the dataset. You can also add other loss functions to improve your results.

Training

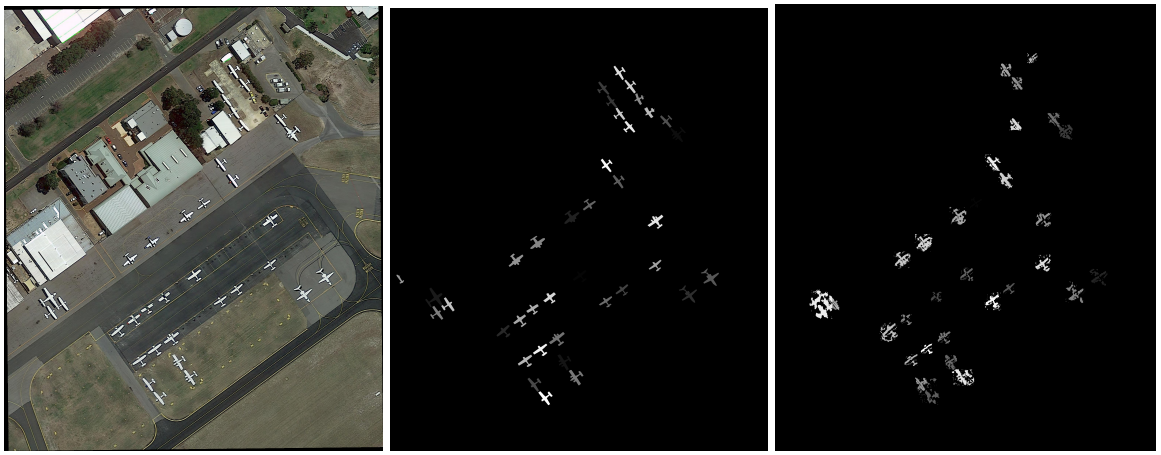
The baseline optimizer for the notebook is SGD with the learning rate of 1e-3. However, based on your network, you may need to change the optimizer and hyperparameters such as learning rate to find the best configuration for your model. Note that picking a good optimizer and learning rate can highly affect the total training time of your network.

Evaluation

For this part, you need to obtain the Intersection-over-Union (IoU) of the ground truth mask and the predicted output as the metric. Evaluate the IoU of all the instances and report the average of them as the final score of your model. Visualize 3 samples from the test images by manually cropping the planes or using the results of the detector of part 1. You do not have the ground-truth segmentation mask, so the mask is not required.

Part 3: Instance Segmentation [13pts]

Having both the detection and the segmentation modules give us the opportunity to also have the instance segmentation results for the dataset. In this regard, you need only to replace the output of your trained object detector instead of the given ground-truth bounding boxes in Part2 and combine the instances of each image to visualize the predicted instance segmentation. We have provided the required functions to convert predicted instance segmentation masks to a CSV file. Visualize 3 samples from the test set of your results (the ground truth mask is not required) and submit the CSV file to Kaggle. Instances could have different intensity as it is shown in the figures, however, using different colors for visualization is recommended.



A valid submission to Kaggle with higher accuracy than our baseline submission in the private leaderboard (with dice coefficient as the metric) and the visualization of the samples in the report will get 6 marks. The public and private scores of the baseline are very close. The rest of the mark is based on the private leaderboard, which will be published after the deadline. The following table shows the markings based on the relative rankings on the private leaderboard.

- Top 10% : 7 pts
- Top 30% : 6 pts
- Top 50% : 5 pts
- Top 70% : 4 pts
- Top 90% : 2 pts
- Rest : 0 pts

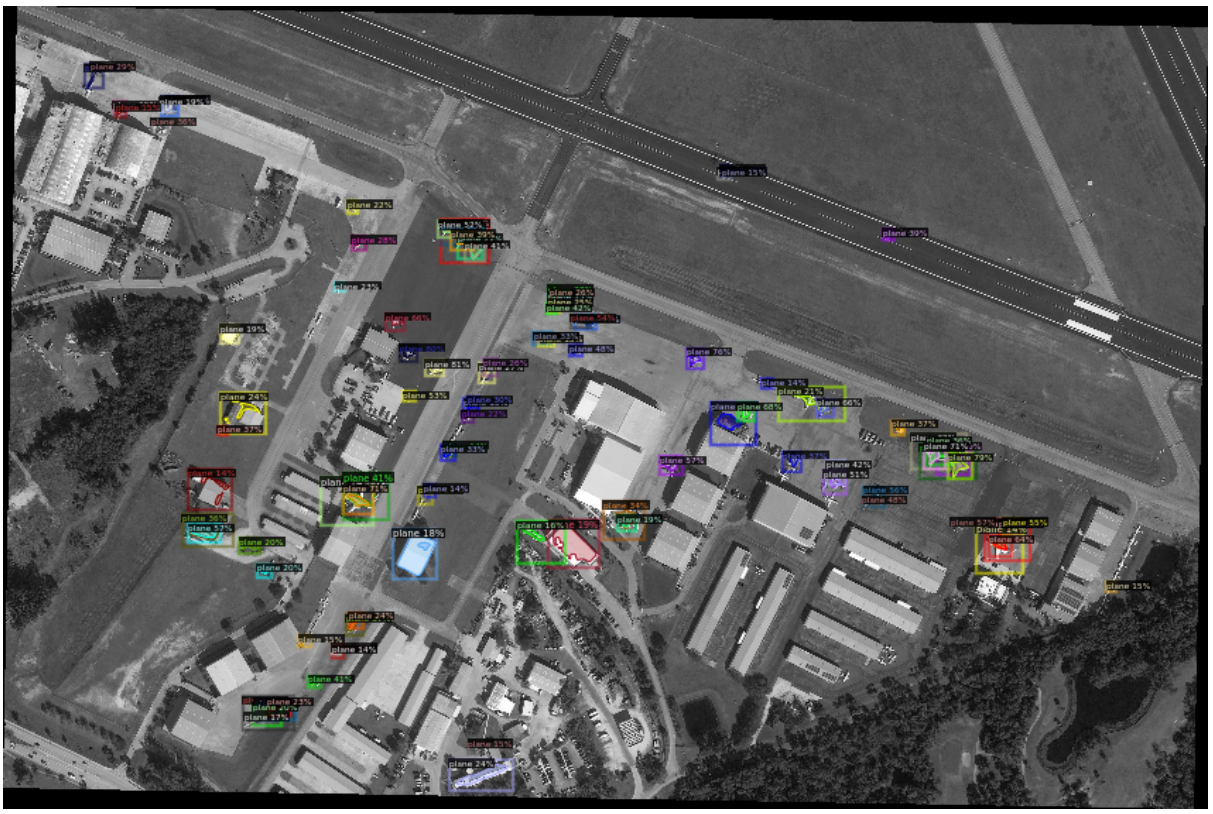
Suppose 73 students submitted, then top 10% mean that the ranking must be better than or equal to the 7th to be given 7pts. Note that the marking is based on the private leaderboard which will be evaluated by the test results and will be published after the deadline, before that, you can see the public leaderboard based on the training data. We will take the snapshot of the ranking 3 days after the due-date timing and calculate the points.

Please report your number in the public leaderboard at the time of submission and also the ID which you use in Kaggle.

Part 4: Mask R-CNN [4pts]

In this part, you are supposed to use the implemented version of Mask R-CNN similar to the detection part from the Detectrone2 and compare with your results of Part 3. Use "mask_rcnn_R_50_FPN_3x.yaml" for your configs as well as the same tricks that you did to improve the results of your detector.

- Provide the same visualization and evaluation for this part and compare the results.
- Explain the cons and pros of each method.
- Check the detection results with the results of Part 1. How much are the results different? Explain which one you think is better and why?



Kaggle Submission

Running Part 3 in the Colab notebook creates a pred.csv file in your Google Drive. The csv file needs to be uploaded to Kaggle.

A few useful resources

- Detectron2 Beginner's Tutorial
- Creating shortcuts to your google drive
- Full documentation of Detectrone2 in this link.
- This post is a helpful resource on understanding semantic segmentation with U-Net.
- Colab implementation of U-Net using pytorch.
- This post explained the difference between segmentation-mask representations.
- An explanation of Run Length Encoding (RLE) scheme for COCO annotations.
- https://arxiv.org/abs/1505.04597 - U-Net: Convolutional networks for biomedical image segmentation.
- Metrics for semantic segmentation

Tips

- Above images are just examples to represent the target task, try your best to visualize the results.
- All edits and configs which lead to a significant accuracy improvement must be listed in the report. You must include at least one ablation study for such an edit (or a

combination of edits), that is, submitting results with and without the edit to Kaggle, and reporting the performance improvement.

Submission Checklist

- Part 1:
  - List of the configs and modifications that you used.
  - Factors which helped improve the performance. Explain each factor in 2-3 lines.
  - Final plot for total training loss and accuracy. This would have been auto-generated by the notebook.
  - The visualization of 3 samples from the test set and the predicted results.
  - At least one ablation study to validate the above choices, i.e., a comparison of performance for two variants of a model, one with and one without a certain feature or implementation choice. In addition, provide visualisation of a sample from the test set for qualitative comparison.
- Part 2:
  - Report any hyperparameter settings you used (batch_size, learning_rate, num_epochs, optimizer).
  - Report the final architecture of your network including any modification that you have for the layers. Briefly explain the reason for each modification.
  - Report the loss functions that you used and the plot the total training loss of the training procedure
  - Report the final mean IoU of your model.
  - Visualize 3 images from the test set and the corresponding predicted masks.
- Part 3:
  - The name under which you submitted on Kaggle.
  - Report the best score (should match your score on Kaggle).
  - The visualisation of results for 3 random samples from the test set.
  - CSV file of your predicted test labels needs to be uploaded to Kaggle.
- Part 4:
  - The visualisation and the evaluation results similar to Part 1.
  - Explain the differences between the results of Part 3 and Part 4 in a few lines.