Object Recognition and Classification Task of the Military Aircraft Recognition dataset

Introduction

This is a document that describes a solution of Object Recognition and Classification Task for Trainee ML vacancy. Original pdf file proposes 7 tasks. I, personally, disagree with the order of tasks proposed in the pdf document. In this description I will order tasks as I believe would be the correct order.

Task 3: What is my task?

The described task in this document is object detection of planes on the satellites images. Given the dataset X: Military Aircraft Recognition dataset, I need to train a machine learning model to detect bounding boxes of planes on a given satellite image and recognize plane types presented in image.

Dataset url:
https://www.kaggle.com/datasets/khlaifiabilel/military-aircraft-recognition-dataset/data

Task 2: Exploratory Data Analysis

Dataset structure

Analyzed dataset is a remote sensing image Military Aircraft Recognition dataset that includes 3842 images, 20 types, and 22341 instances annotated with horizontal bounding boxes and oriented bounding boxes.
Original dataset has 3 folders:

- Images - folder of images in jpg format. Every image has 3-channels (RGB format).
- Imagesets - consists of two text files. Train.txt has ids of train set and test.txt has ids of testset. Train set consists of 1331 image ids. Test set consists of 2511 image ids.
- Annotations - consists of two folders, one for Horizontal bounding boxes and one for Oriented bounding boxes. Bounding boxes are presented in xml files.

XML file structure:

- Filename
- Image dims (h, w, depths)
- Database (probably the resource of the image)
- Segmented

- Objects:
  Each object has:
  1) xmin, xmax, ymin, ymax - parameters of bounding box
  2) Name - plane type

Xml files have some problems, they are not a valid source of information regarding size parameters of an image. After analyzing xml files I've found several images with 0 width/height, and only 1 channel (when in reality the image had 3 channels). Thus said, xml files could be used only for retrieving coordinates of bounding boxes.

Images

Image examples:

From those different images we can see that the size and color of planes are very strong features and will be utilized by our model. This means that we need to keep

everything in RGB format and think of what size we should choose for resizing images in pre-processing state.

Also, we need to note that some images can have clouds or fogs in them. It could possibly lead to bad results in our model

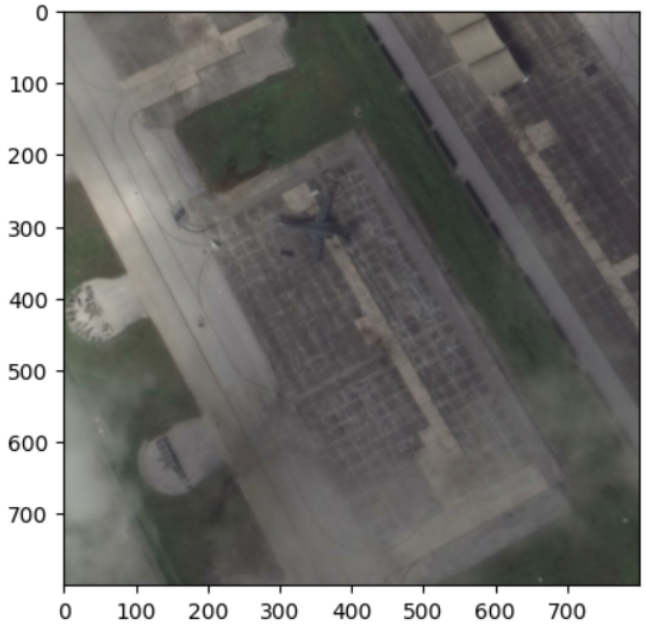
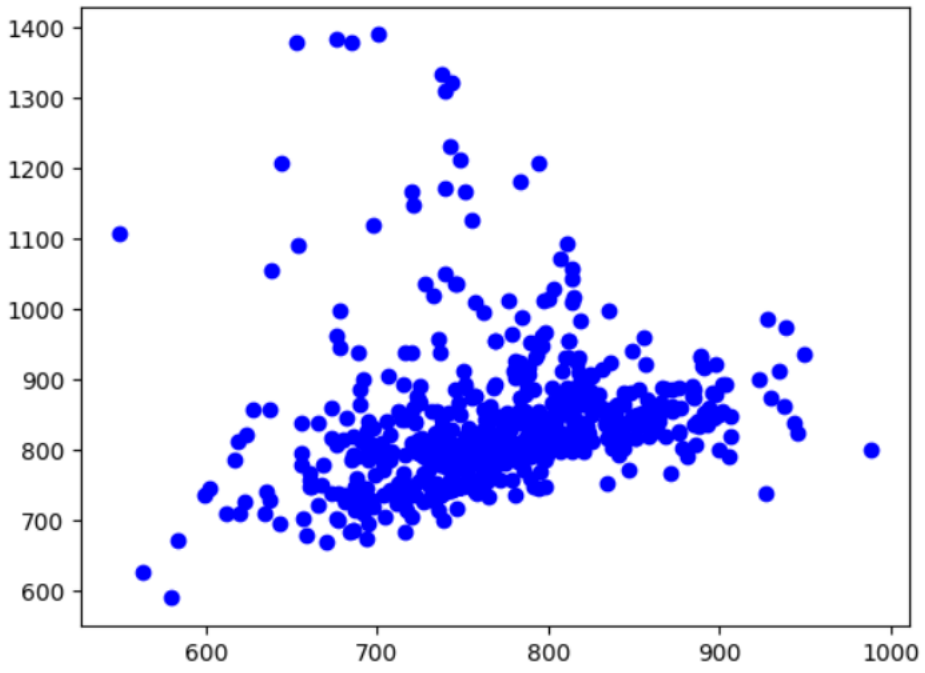


Image with clouds

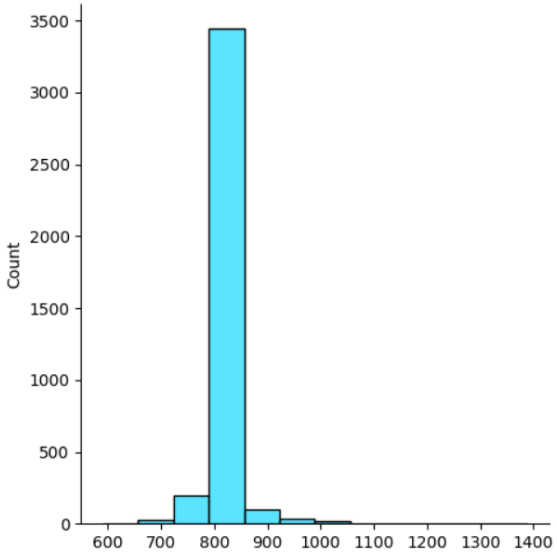
Size analysis

Width to height plot:

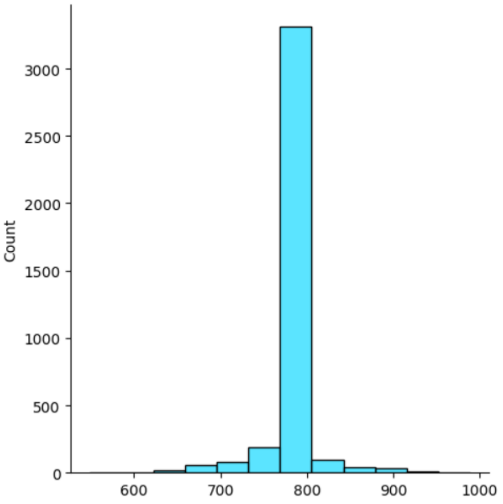


We can see that most of the data lies in 600-900 range. There are also some outliers regarding height and width, and even their combination (we have an image with height > 1100 and width lower than 600).

Height distribution

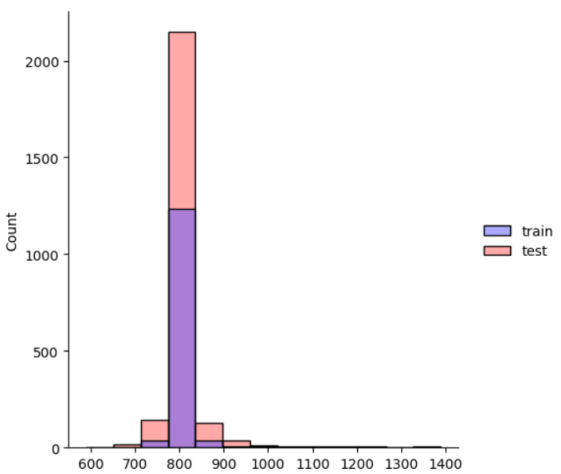


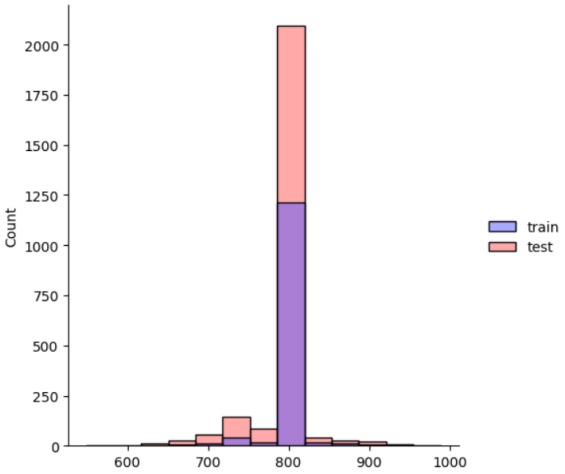
Width distribution



From the height and width distribution we can see that on average image width is lower than height. That will not be a problem for our model. It is also calculated that there are only ~90 images where height is more than 900 and only 17 where width is more than 900. So we can make a hypothesis that this might be a good value for resizing purpose.

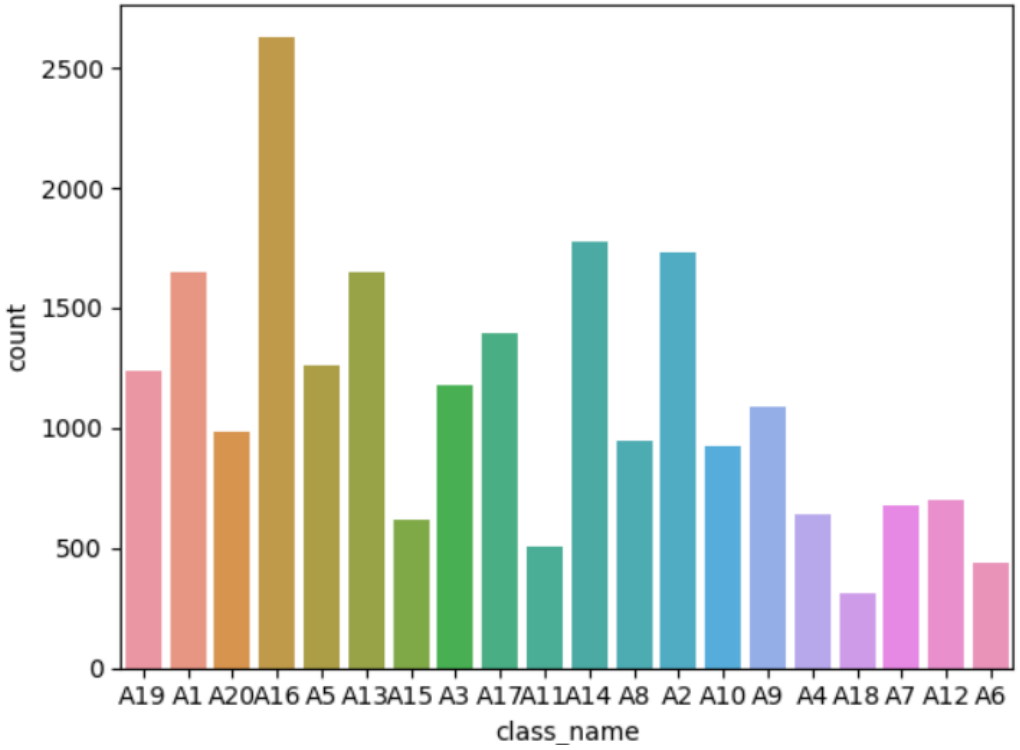
Height/Width train/test distribution (Pretty much same distributions across train/test set)



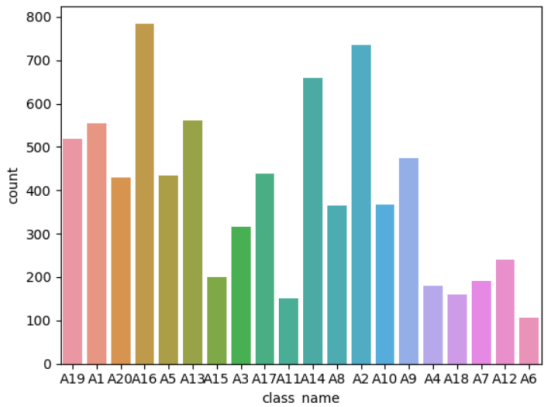


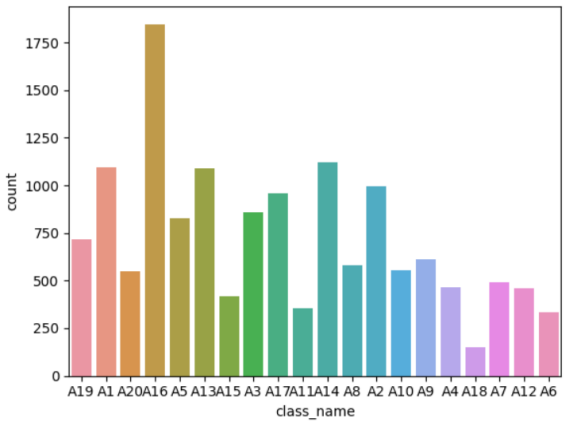
Class distribution

Some data imbalance is presented, varying from 308 to 2632 values per class. The lowest value is for A11 plane and the highest is for A16 plane. We should keep this in mind since it can affect model performance.



Train/Test sets count distributions have almost same distributions. Concentration of A16 plane in test set is larger than in train set, that is why distribution plots looks slightly different. I do not think that this difference will affect our model, but it may introduce some bias in evaluation of the model.

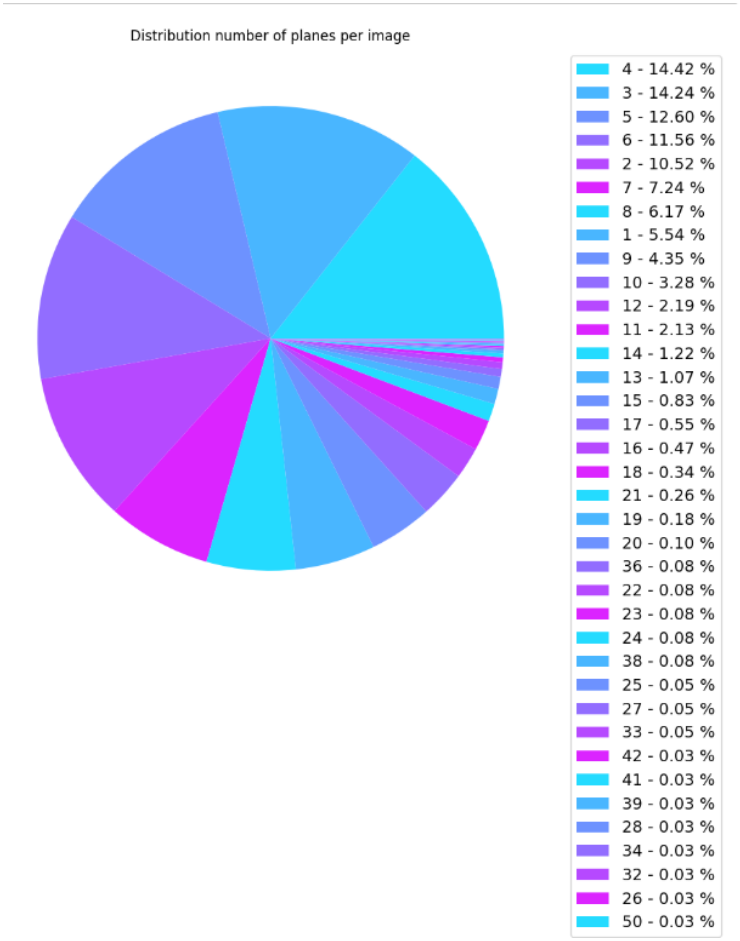




Bounding boxes

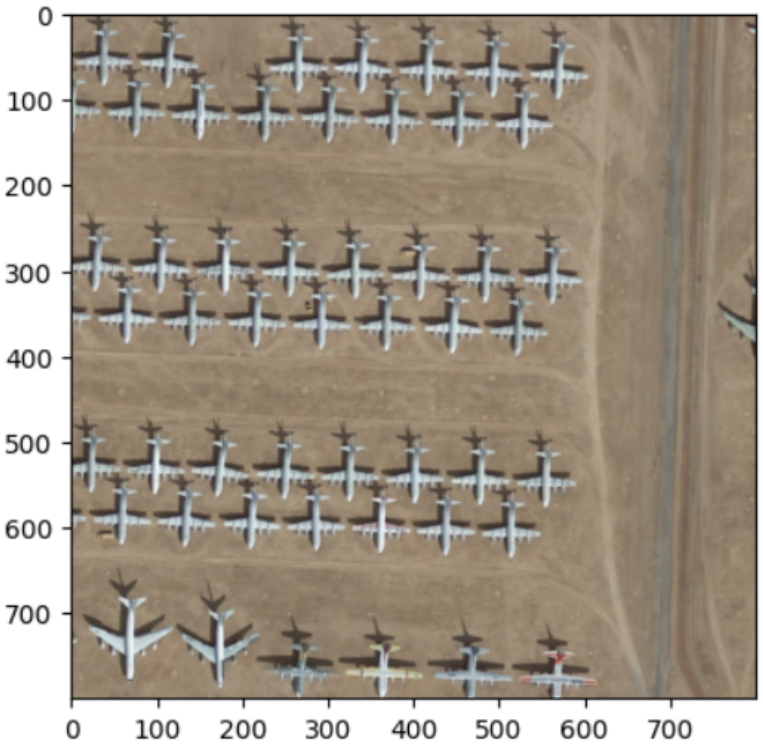
Visualization of bounding boxes:

Distribution of object count per image:



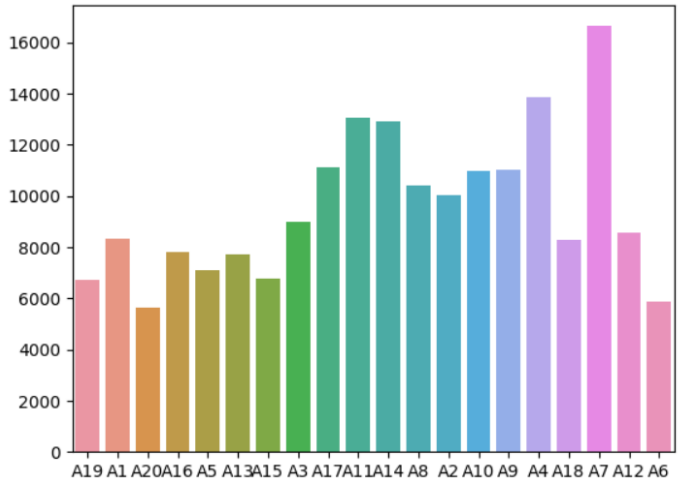
We can see a huge imbalance for images containing large amounts of planes. Only 1 image for 50/41/42 object counts.

Image with 50 planes:
This is not the hardest case for object detection. We just need to find the model robust enough to such images.



Bounding boxes areas analysis

Most of the bounding boxes aspect ratios lies between 0 and 0.25, so we have a huge amount of small objects. It is also interesting that some of the planes occupies more area than others.
Median area (w * h) of plane per category:



Task 4/1: Preparing dataset for training

As all trained models are YOLO models, the same procedure for dataset downloading and preparation was used:

1) Downloading dataset with kaggle

```
# Download dataset
!kaggle datasets download -d khlaifiabilel/military-aircraft-recognition-dataset
```

2) Unzipping dataset

```
# Unzip dataset to data folder
!unzip military-aircraft-recognition-dataset.zip -d "data"
```

3) Create folders structure for data as YOLO requires

```
# Create appropriate folders
!mkdir "data for yolo"
!mkdir "data for yolo/labels"
!mkdir "data for yolo/images"
!mkdir "data for yolo/labels/train"
!mkdir "data for yolo/labels/test"
!mkdir "data for yolo/images/train"
!mkdir "data for yolo/images/test"
```

4) Convert dataset to yolo format. For it:
   a) Parse xml horizontal bounding boxes files
   b) For each image create .txt file where one line will describe one bounding box in format:
   class_id x_center y_center width height
5) Copy image files to created directory:

```
# Copy images to YOLO
def move_files_to_folder(list_of_files, destination_folder):
    for f in list_of_files:
        try:
            shutil.move(f, destination_folder)
        except:
            print(f)
            assert False

train_img_files = [IMAGE_DIR + str(obj_id) + ".jpg" for obj_id in train_set_ids]
test_img_files = [IMAGE_DIR + str(obj_id) + ".jpg" for obj_id in test_set_ids]

# Move the splits into their folders
move_files_to_folder(train_img_files, 'data for yolo/images/train')
move_files_to_folder(test_img_files, 'data for yolo/images/test')
```

6) Create YOLO dataset yaml file

```
# Make .yaml file for yolo
d = {
    'path':'../data for yolo/',
    'train': 'images/train',
    'val': 'images/test',
    'nc': 20,
    'names': class_id_to_name_mapping
    }

with open('dataset.yml', 'w') as yaml_file:
    yaml.dump(d, yaml_file, default_flow_style=False)
```

Task 5/6: Models training; Incremental Approach

For resolving this task YOLOv5 model was used. In my opinion, this model is an optimal solution for this task, due to its relatively fast training and high results.

Choosing model input dimensions

Yolo model works with square images. It can take an image of any size, but in pre-processing step it compiles square image with size n x n using padding technique. Model saves the aspect ratio of original image. Since our dataset consists of mostly small objects, the size of resized image is a crucial parameter. So what I did before main training was to train two YOLOv5s models with different image resize parameters, 640 px and 800 px. I've run two models for 50 epochs with default hyperparameters, and the results I got were that overall precision and recall were better for model with 800px image resize size.

Training baseline YOLO

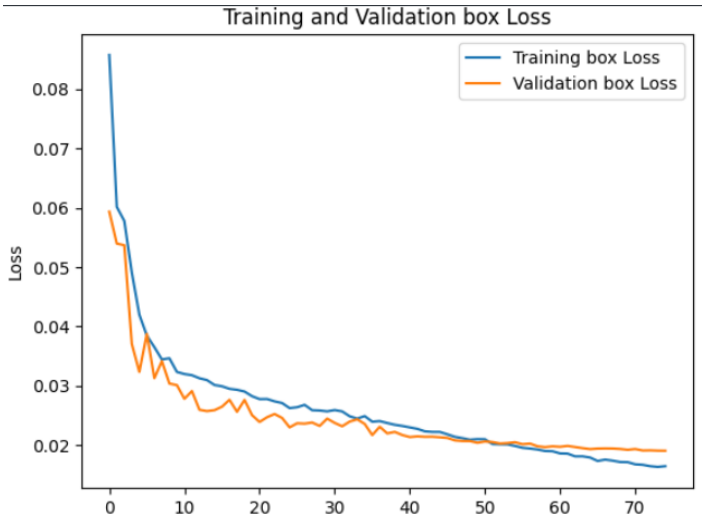
Baseline YOLOv5s was trained with default parameters, image size 800px and batch size 32. It was trained for 75 epochs. Weights were pre trained on COCO dataset. Obtained results are displayed in the next image:

Class	Images	Instances	P	R	mAP50	mAP50-95:
all	2511	14471	0.842	0.826	0.875	0.666
A19	2511	717	0.754	0.833	0.853	0.633
A1	2511	1092	0.85	0.798	0.871	0.687
A20	2511	551	0.825	0.824	0.849	0.63
A16	2511	1847	0.988	0.63	0.901	0.693
A5	2511	827	0.667	0.773	0.745	0.518
A13	2511	1091	0.712	0.715	0.738	0.525
A15	2511	418	0.408	0.553	0.439	0.284
A3	2511	860	0.882	0.961	0.964	0.686
A17	2511	958	0.969	0.92	0.978	0.783
A11	2511	356	0.748	0.876	0.861	0.688
A14	2511	1119	0.938	0.824	0.946	0.724
A8	2511	579	0.918	0.898	0.942	0.748
A2	2511	993	0.992	0.867	0.965	0.704
A10	2511	556	0.952	0.924	0.984	0.719
A9	2511	611	0.92	0.881	0.942	0.711
A4	2511	463	0.938	0.844	0.948	0.736
A18	2511	148	0.678	0.709	0.74	0.594
A7	2511	489	0.93	0.982	0.987	0.812
A12	2511	462	0.805	0.773	0.866	0.678
A6	2511	334	0.969	0.928	0.974	0.758

Baseline YOLO results evaluation

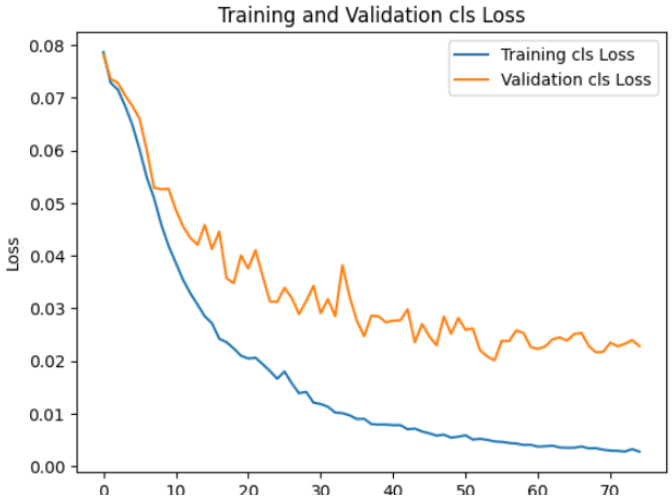
After training baseline YOLO I've made an evaluation of it on the whole validation dataset.

Box loss



Box loss looks well. It generally gives us an understanding that the dataset is well annotated and that train dataset is good representation of test set in terms of bounding boxes.

Class loss



Unfortunately, graph of class loss looks nothing like box loss graph. Two main reasons for this is model simplicity and class imbalance.

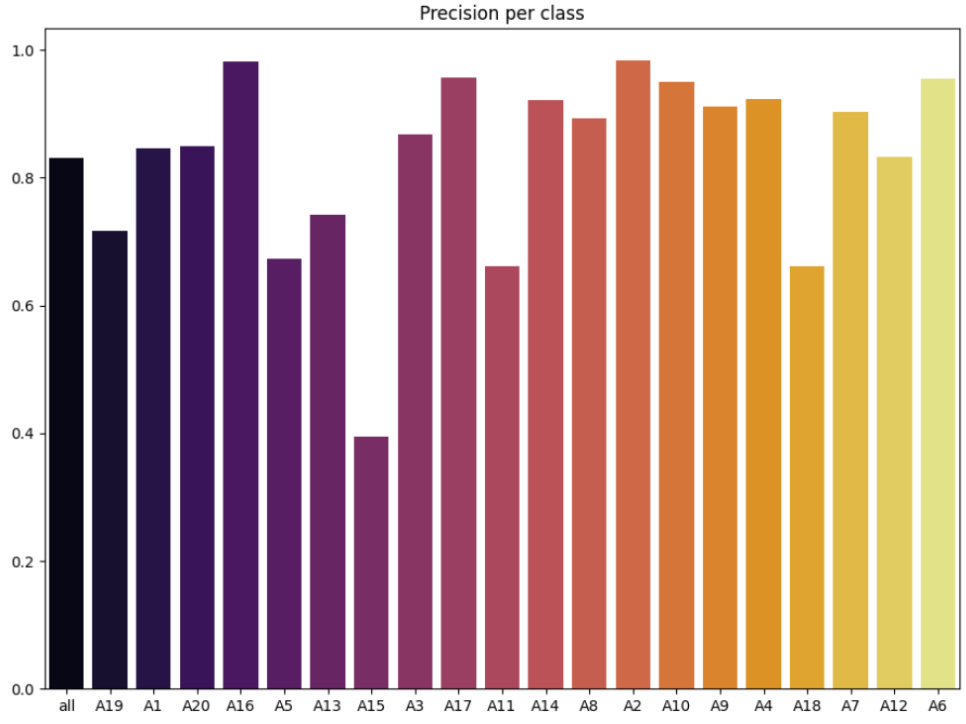
Validation run results

After validating the model on validation set, next results were obtained:

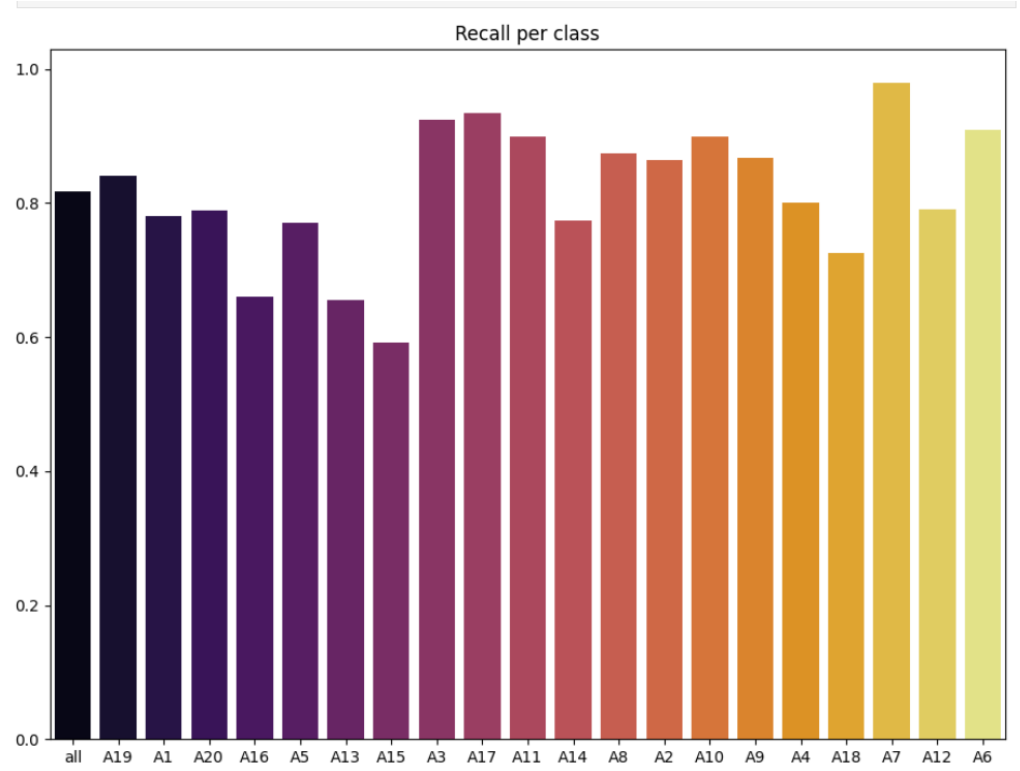
Precision | Recall

all	2511	14471	0.831	0.817	0.864	0.65
A19	2511	717	0.717	0.84	0.83	0.606
A1	2511	1092	0.846	0.78	0.857	0.665
A20	2511	551	0.849	0.789	0.822	0.6
A16	2511	1847	0.982	0.661	0.907	0.686
A5	2511	827	0.673	0.771	0.731	0.5
A13	2511	1091	0.741	0.655	0.736	0.512
A15	2511	418	0.394	0.591	0.435	0.281
A3	2511	860	0.868	0.924	0.944	0.657
A17	2511	958	0.956	0.935	0.974	0.774
A11	2511	356	0.661	0.899	0.857	0.675
A14	2511	1119	0.922	0.774	0.922	0.705
A8	2511	579	0.892	0.874	0.919	0.718
A2	2511	993	0.984	0.864	0.951	0.692
A10	2511	556	0.949	0.899	0.978	0.723
A9	2511	611	0.911	0.867	0.928	0.695
A4	2511	463	0.923	0.801	0.922	0.712
A18	2511	148	0.661	0.726	0.752	0.605
A7	2511	489	0.903	0.98	0.984	0.796
A12	2511	462	0.832	0.79	0.873	0.672
A6	2511	334	0.954	0.91	0.962	0.735

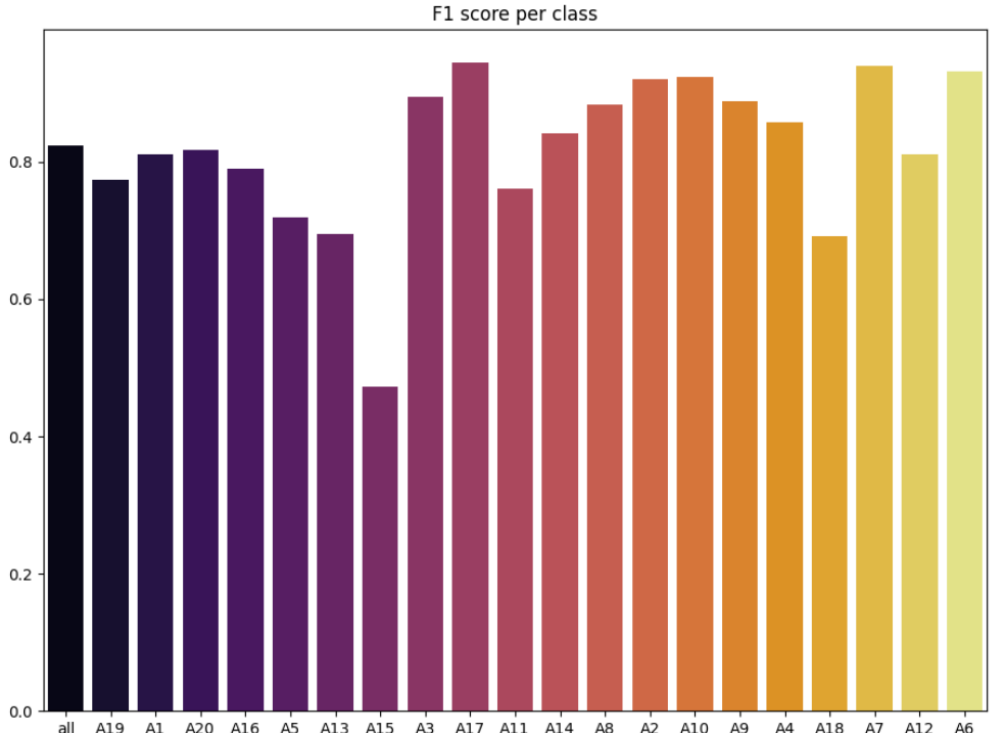
Precision per class:



Recall per class:

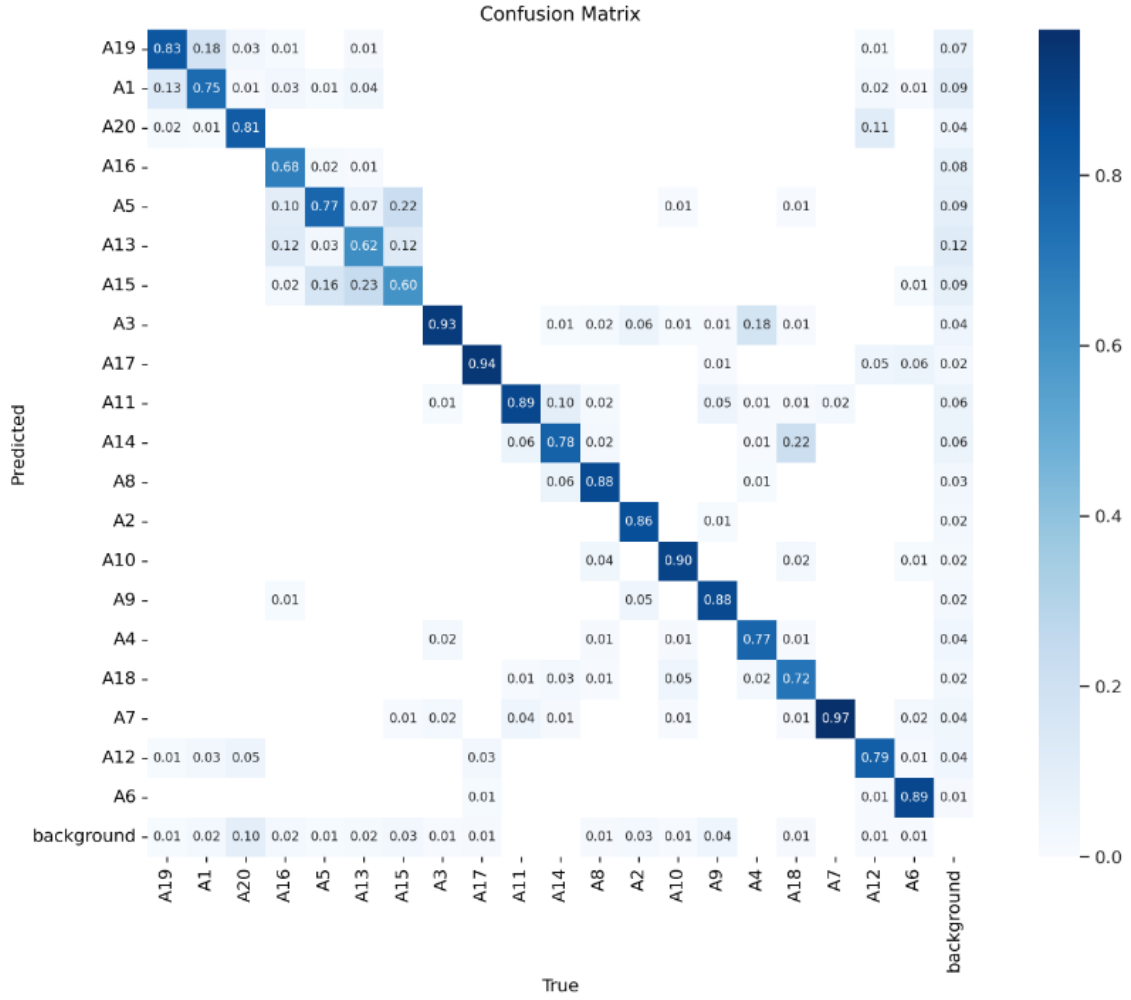


F1 score per class



Overall results are pretty good. We have an overall Precision score of 0.831 and overall Recall score of 0.817, which gives us an F1 score of 0.823. We can also see that the worse Precision and Recall both are for A15 plane. F1-score on A15 is 0.48. It basically means, that model often confuses A15 with other classes

We can approve this by looking at a confusion matrix:



We can see that A15 plane is often misclassified as A13 or A5 planes and vice versa. Here, again, class imbalance is a problem, but it also can be a general similarity between those planes.
We can approve this by looking at those planes in our dataset

A15:

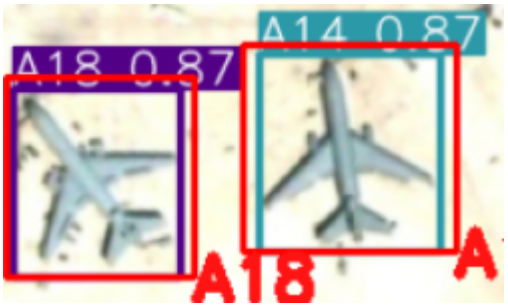
A5:

A13:

We can see that they are pretty similar. Moreover, from images we can understand that A13 is more like a bigger version of A5. And since we have images of different size even for one type of plain, it might be pretty hard for our model to predict such planes.

Orientation problem

While looking at a misclassified pictures, I've also noted that some of the planes recognized really good until they change their orientation.
E.g.



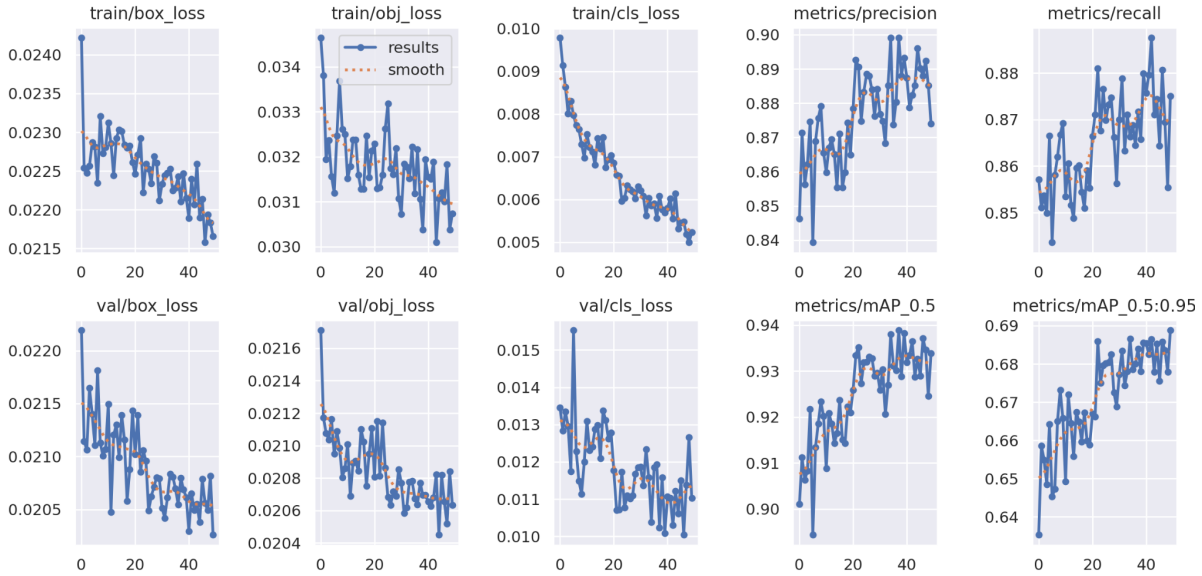
Those are both A18 planes, but since A14 looks straightly up, I believe that the model might miss classify it with A18. Thus said, it would be better to perform data analysis of oriented bounding boxes, but I might don't have enough time for this.

Training YOLO 2

I've decided to go with incremental approach. Thus, I've chosen the same model with some tweaks. Firstly, the class weights were included in the training process. Secondly, additional augmentation was added to influence training on different rotation images. Also, Vertical flip with probability of 0.5 was added. The model was trained on image size 800, batch 48 and 115 epochs. Weights were pre trained on COCO dataset.

Losses

Note, that due to google colab restrictions, model was trained in two sessions. Unfortunately, google colab destroyed results of the first training, so this graphs does not really represent anything.



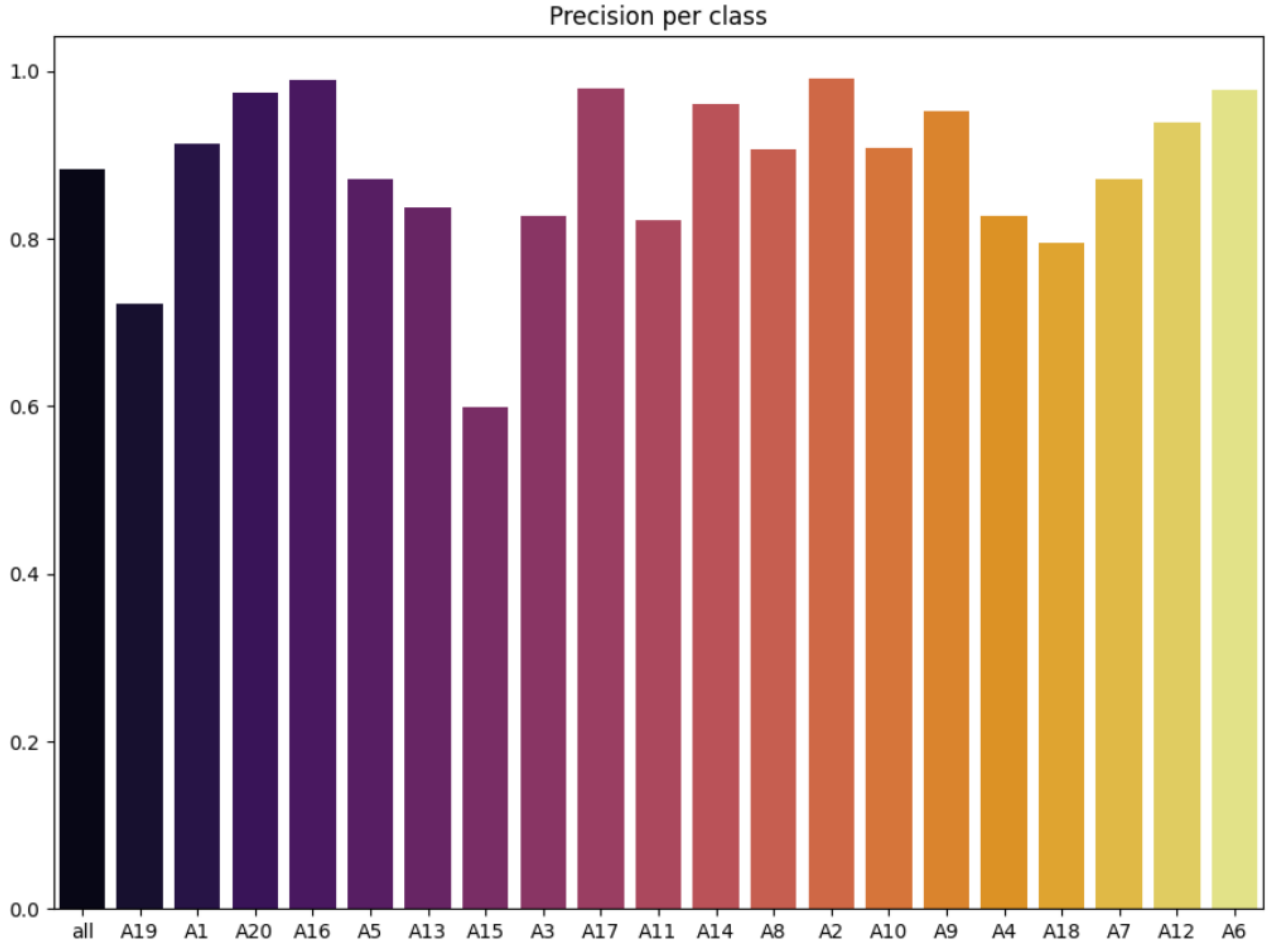
Validation run results

Overall results looks like this:

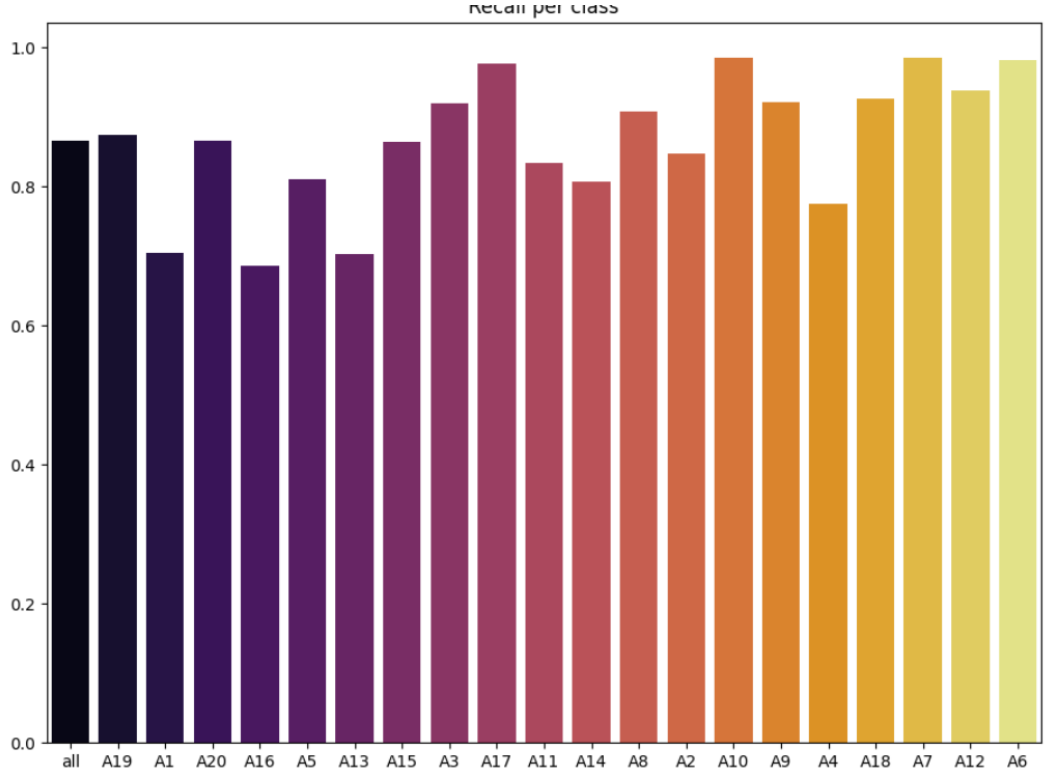
Class	Images	Instances	P	R	mAP50	mAP50-95: 100% 79/79 [01:20<00:00, 1.02s/it]
all	2511	14471	0.883	0.866	0.933	0.694
A19	2511	717	0.722	0.874	0.864	0.616
A1	2511	1092	0.913	0.705	0.893	0.709
A20	2511	551	0.975	0.866	0.931	0.641
A16	2511	1847	0.989	0.686	0.949	0.717
A5	2511	827	0.871	0.81	0.901	0.608
A13	2511	1091	0.838	0.703	0.856	0.581
A15	2511	418	0.599	0.864	0.812	0.517
A3	2511	860	0.827	0.919	0.943	0.658
A17	2511	958	0.98	0.977	0.99	0.793
A11	2511	356	0.823	0.834	0.888	0.695
A14	2511	1119	0.961	0.808	0.964	0.743
A8	2511	579	0.907	0.908	0.949	0.737
A2	2511	993	0.992	0.848	0.972	0.704
A10	2511	556	0.909	0.986	0.993	0.712
A9	2511	611	0.953	0.922	0.969	0.713
A4	2511	463	0.828	0.775	0.892	0.684
A18	2511	148	0.795	0.926	0.936	0.737
A7	2511	489	0.871	0.986	0.979	0.769
A12	2511	462	0.939	0.939	0.979	0.769
A6	2511	334	0.977	0.982	0.993	0.769

We can see an increase in Precision Recall and map50 for every class and for all validation set. Precision increased from 0.831 to 0.883, the recall increased from 0.817 ti 0.866. The recall of the new model is 0.874. For A15 class, Precision increased from 0.399 to 0.599 and Recall increased from 0.591 to 0.864.

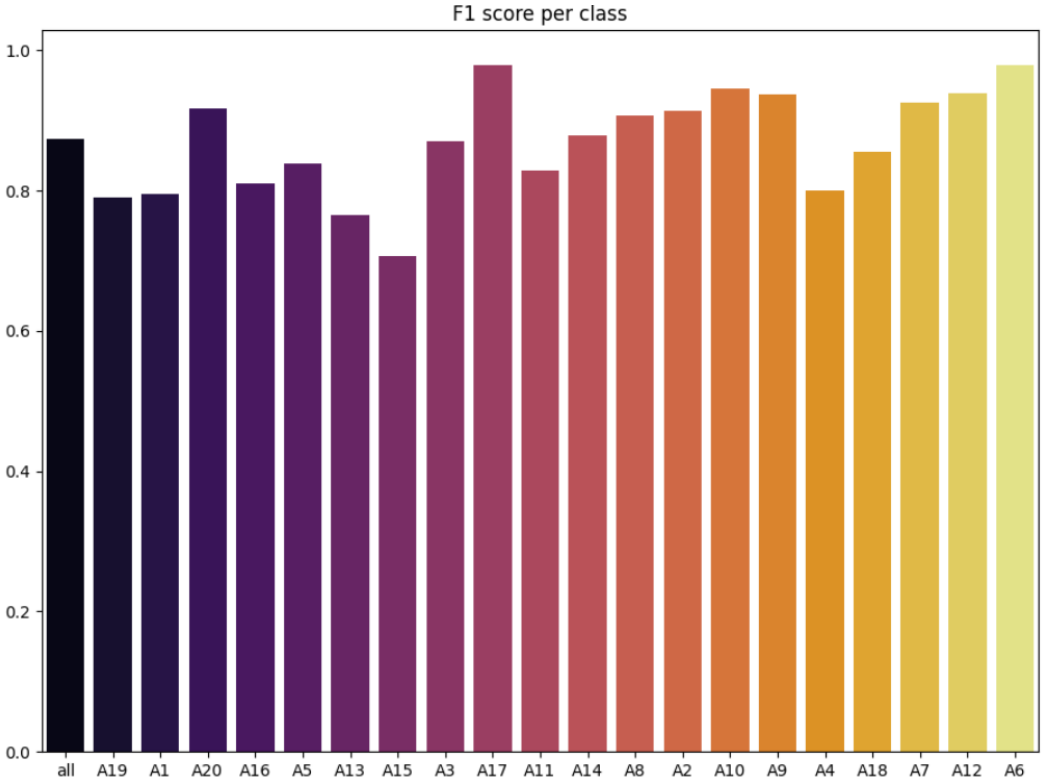
Precision per class



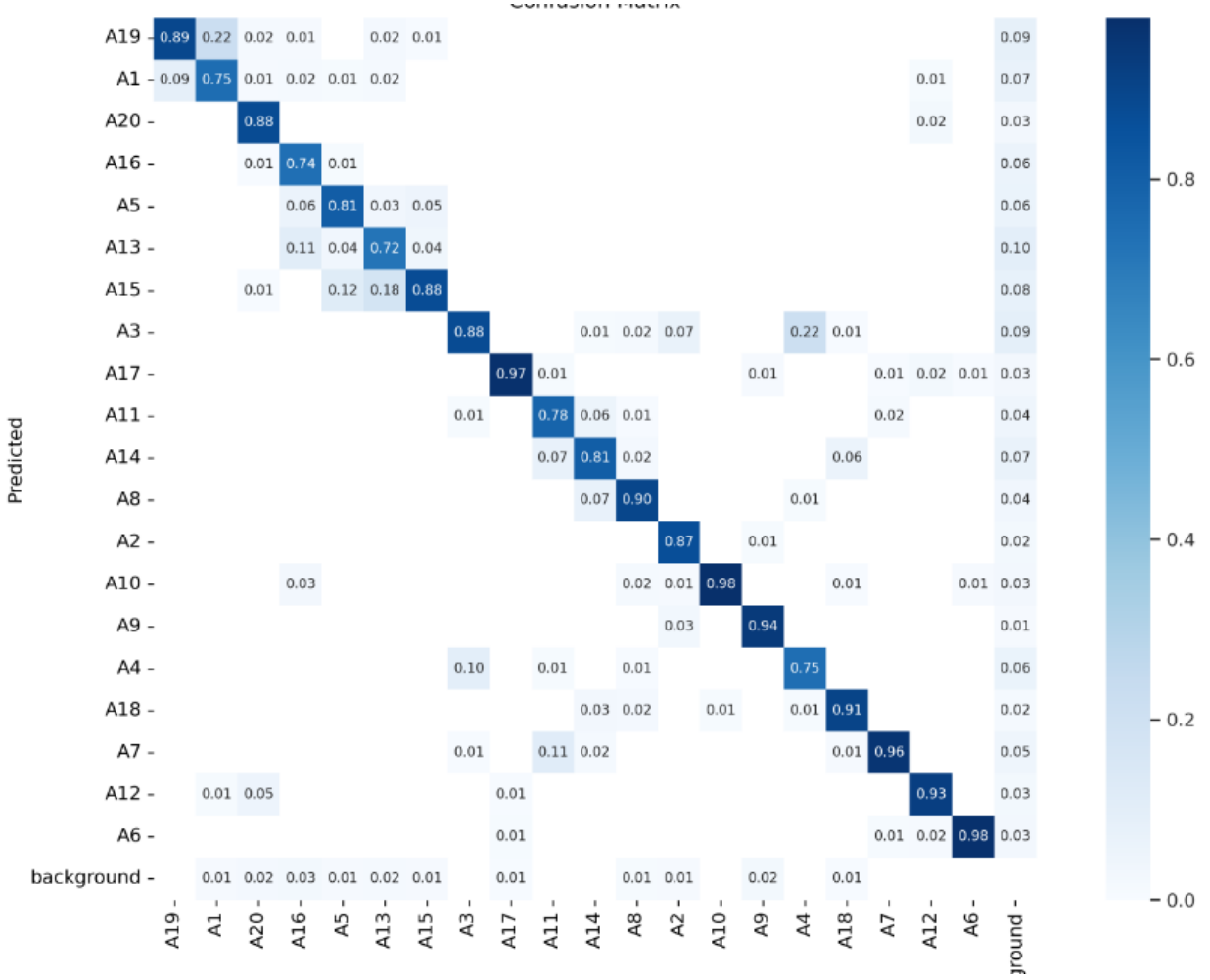
Recall per class



F1-score per class



A15 class still has a problem of being misclassified by other planes.



After general evaluation of the model, I've counted the number of pictures where the model prediction of classes was not identical to the ground truth (suppose we have 2 A20 planes on the image, and our model predicts that there are one A20 and one A3 type). Of course, this statistic does not mean anything, since model could easily have same list of classes in the image but the classes assignment to objects can be wrong. But it is a fast way to obtain images where model completely missed during predictions. From analyzing those images, we can see two types of problems in our model.

On this image, model has not detected any objects at all. This is a hard case for it since a) the image is blurred/low resolution and c) the color of planes is similar to the ground (asphalt).

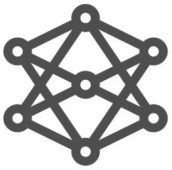
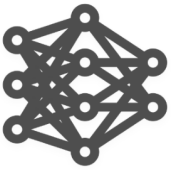
On this image we can see that the model proposes two bounding boxes for the same object. This could be solved by using non-max suppression without splitting it per class.

But in my believe, both of those problems were fixable by presenting a slightly better model.

Training YOLO 3, but another YOLO

Yolo has several structures depending on the complexity you need.

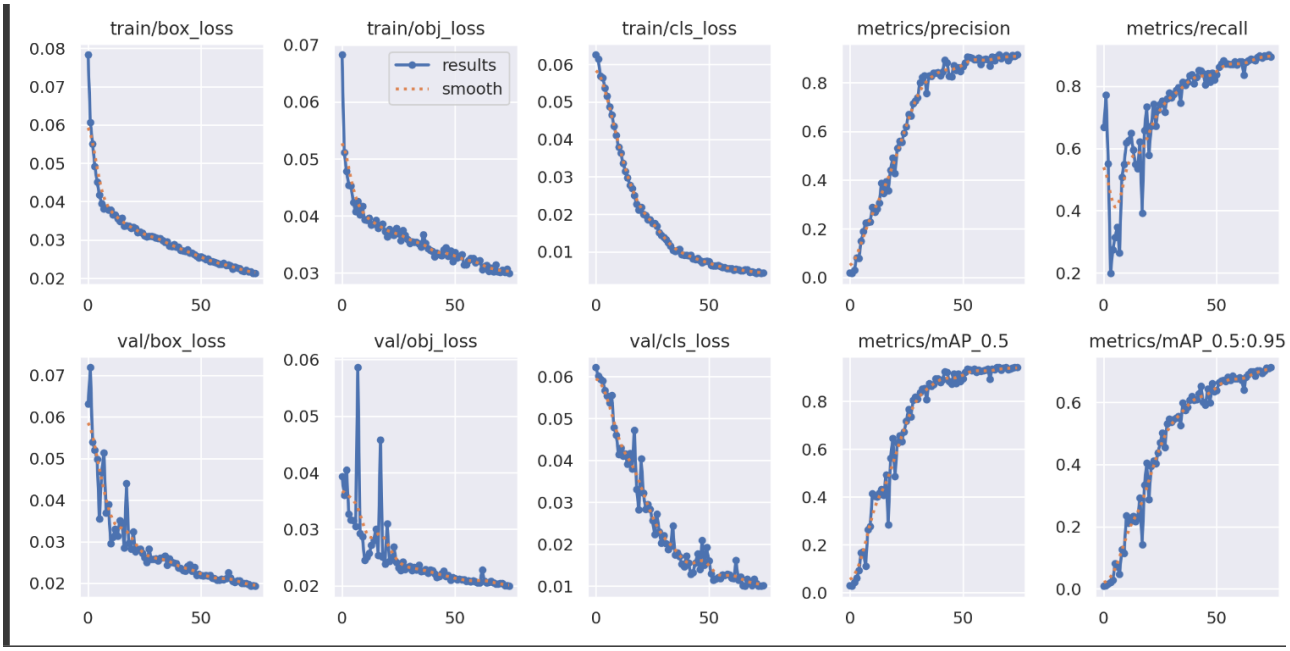


Small	Medium	Large	XLarge
YOLOv5s	YOLOv5m	YOLOv5l	YOLOv5x
15 MB_{FP16} 2.4 ms_{V100} 37.0 mAP_{COCO}	42 MB_{FP16} 3.4 ms_{V100} 44.3 mAP_{COCO}	92 MB_{FP16} 4.4 ms_{V100} 47.7 mAP_{COCO}	170 MB_{FP16} 6.9 ms_{V100} 50.8 mAP_{COCO}

Previously, I've trained YOLOv5s model. But, since it could not fully converge to minimal loss (have not played with hyperparameters, so that might be truly a reason) and due to missed bounding boxes on the image, I've decided to try YOLOv5m model. Weights were pre trained on COCO dataset. I've also increased a rotate angle augmentation (from [-90 90] to [-115 115])
I've trained YOLOv5m for 75 epochs, on 800px image size. Unfortunately, I had to decrease a batch size, since Google Colab had it limitation on the GPU memory usage. Class weights were also used during training.

Training path



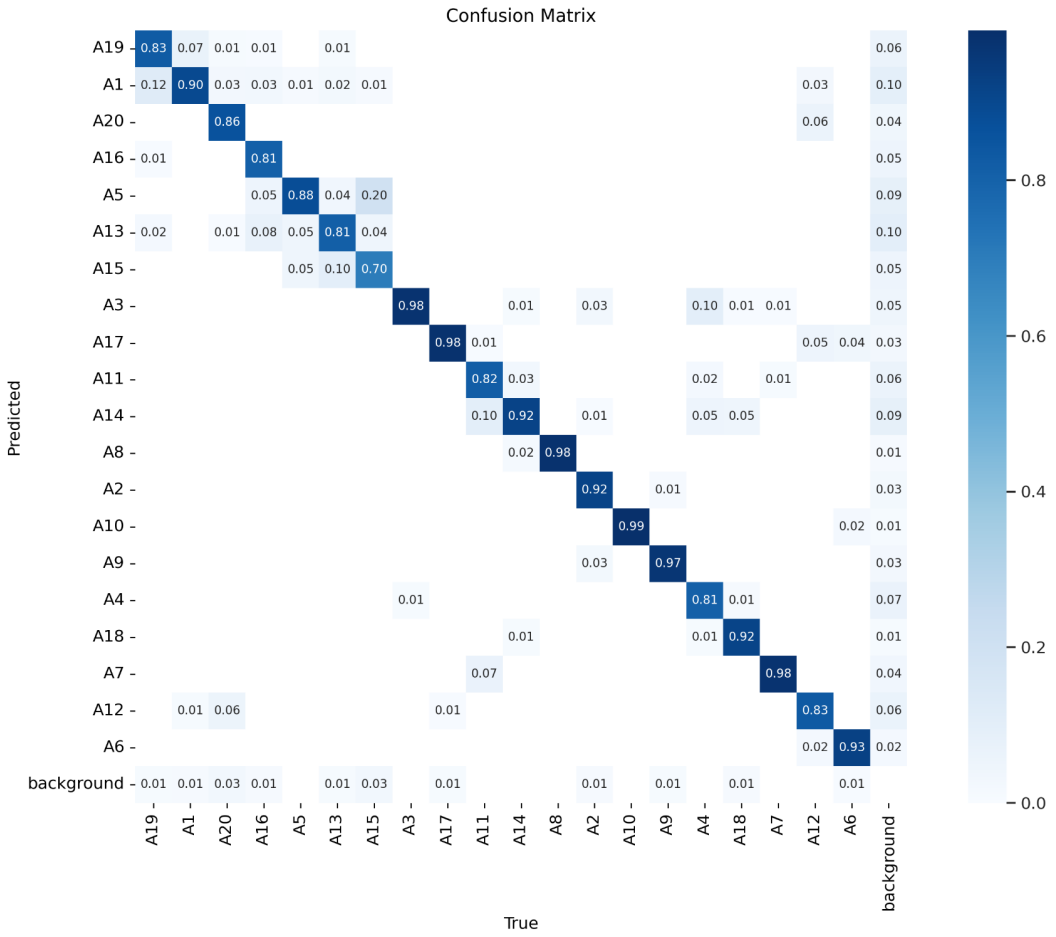
We can see from loss graphs, that while training the model, it had some tweaks in it regarding recall and validation obj_loss on start. It might be due to pre trained weights or due to optimization method. But, at the end validation loss and recall has stabilized to norm.

Model evaluation

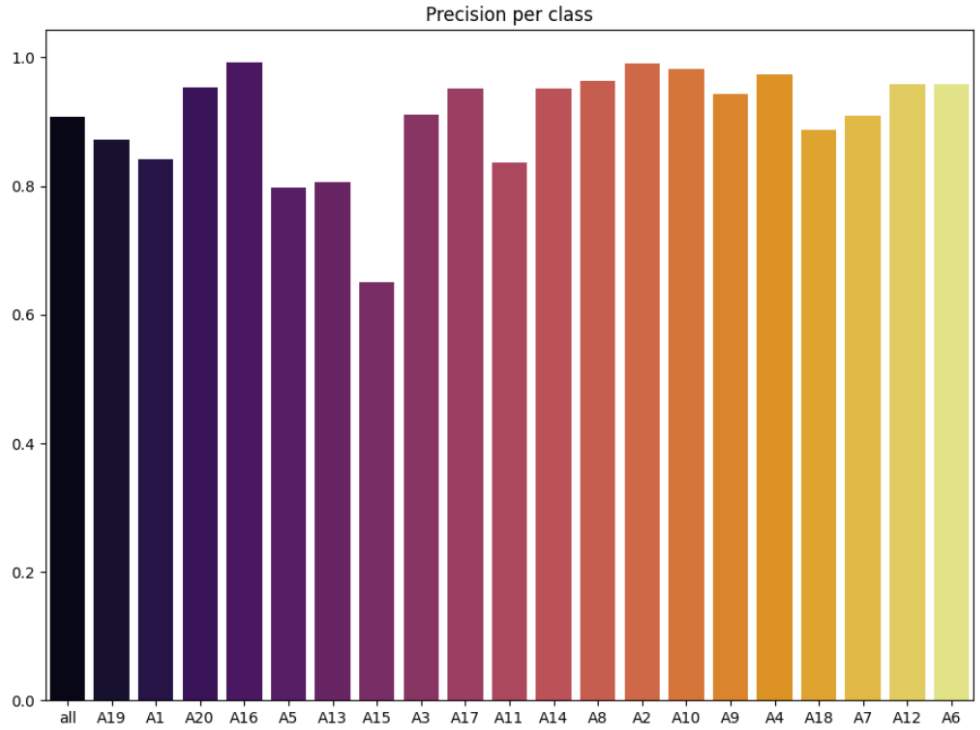
I've evaluated the model on the same validation and here are the results I've obtained:

Class	Images	Instances	P	R	mAP50	mAP50-95:
all	2511	14471	0.907	0.871	0.929	0.7
A19	2511	717	0.873	0.819	0.906	0.677
A1	2511	1092	0.842	0.886	0.918	0.74
A20	2511	551	0.953	0.843	0.929	0.679
A16	2511	1847	0.993	0.733	0.939	0.735
A5	2511	827	0.797	0.842	0.869	0.585
A13	2511	1091	0.806	0.662	0.802	0.564
A15	2511	418	0.651	0.711	0.655	0.445
A3	2511	860	0.911	0.967	0.981	0.659
A17	2511	958	0.952	0.976	0.99	0.772
A11	2511	356	0.837	0.846	0.901	0.705
A14	2511	1119	0.951	0.886	0.965	0.737
A8	2511	579	0.964	0.936	0.972	0.749
A2	2511	993	0.991	0.86	0.958	0.713
A10	2511	556	0.983	0.987	0.995	0.747
A9	2511	611	0.944	0.936	0.979	0.726
A4	2511	463	0.974	0.807	0.961	0.723
A18	2511	148	0.888	0.939	0.944	0.743
A7	2511	489	0.91	0.978	0.977	0.779
A12	2511	462	0.958	0.831	0.959	0.755
A6	2511	334	0.958	0.967	0.99	0.777

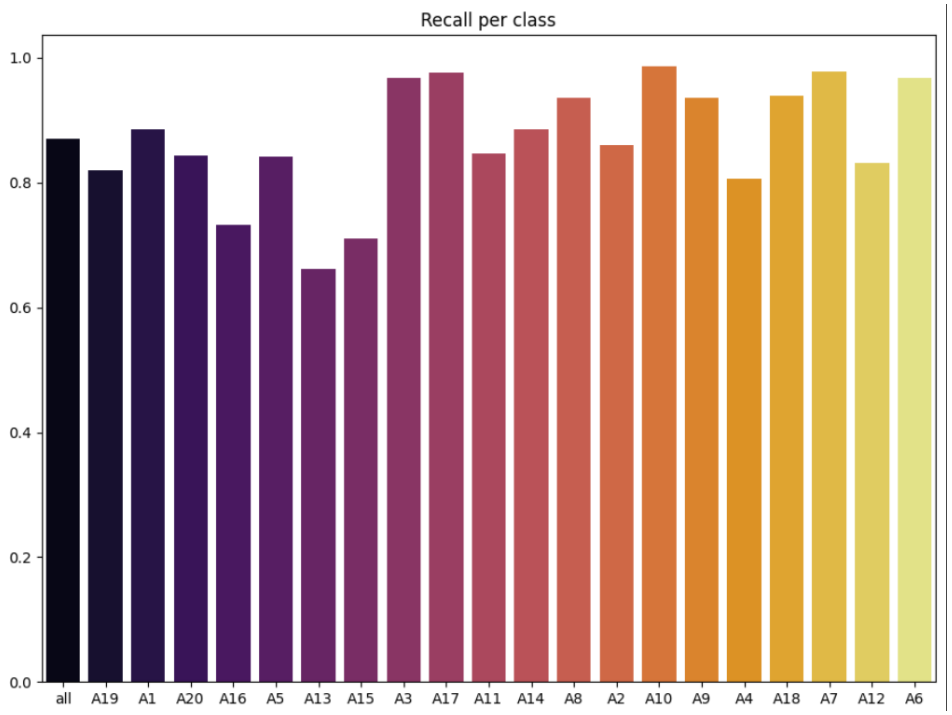
Comparing this model to the previous one, we can see an overall increase in Precision and Recall. From 0.883 to 0.907 Precision and from 0.866 to 0.871 for Recall. For A15 class the precision went up from 0.599 to 0.651 but the recall went down from 0.864 to 0.711. From confusion matrix it is clear that now the A15 plane is not recognized as A13 as often as it was, but the new model now confuses A5 with A15 plane more often.
Overall F1-Score is 0.874.



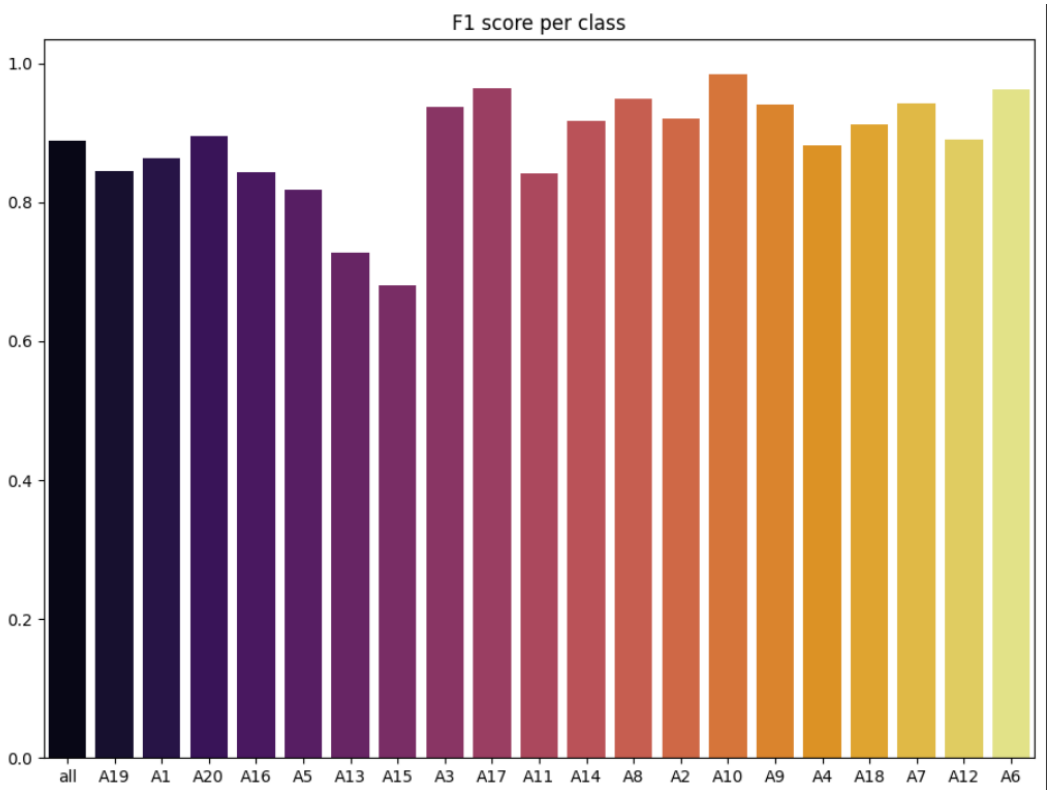
Precision per class



Recall per class

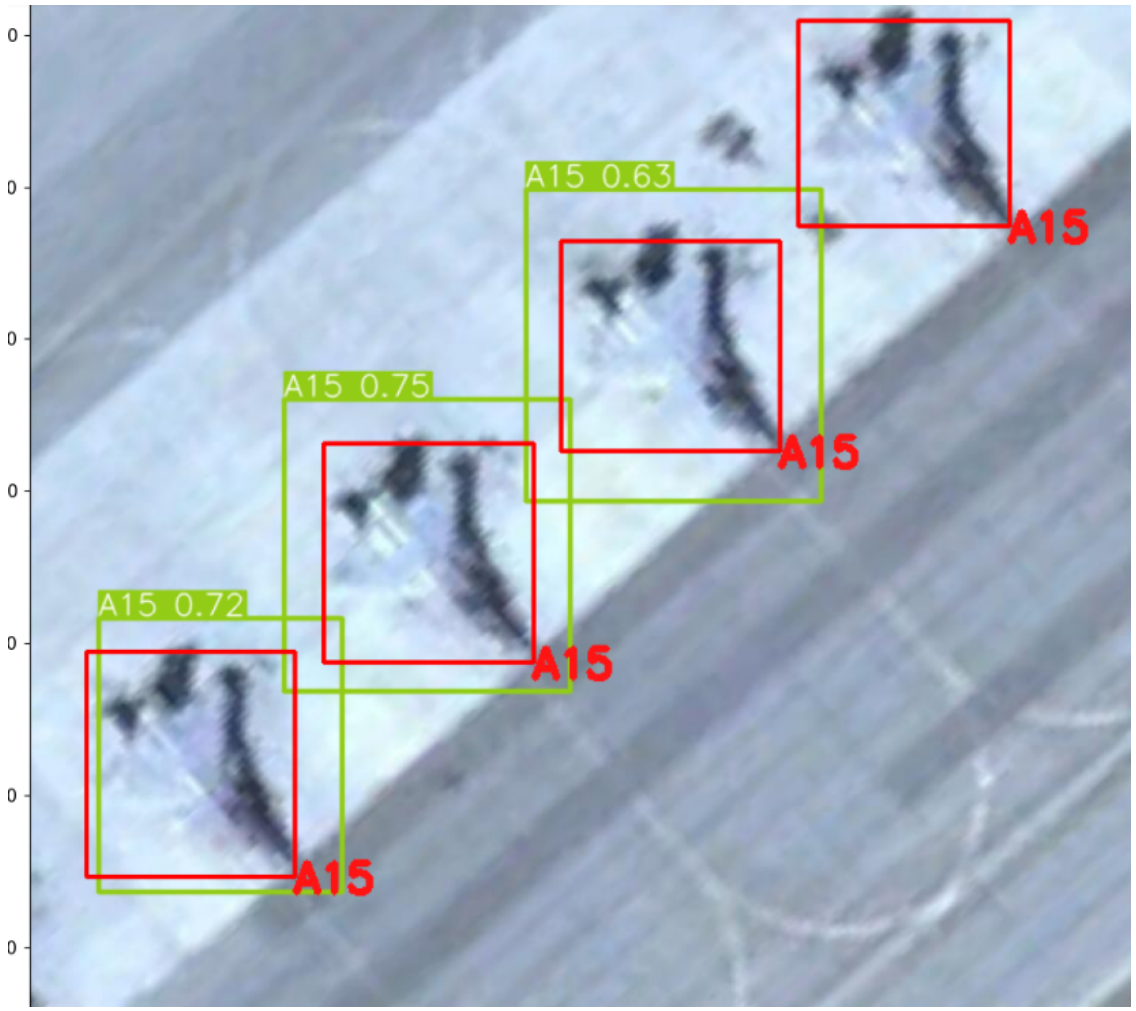


F1-score per class

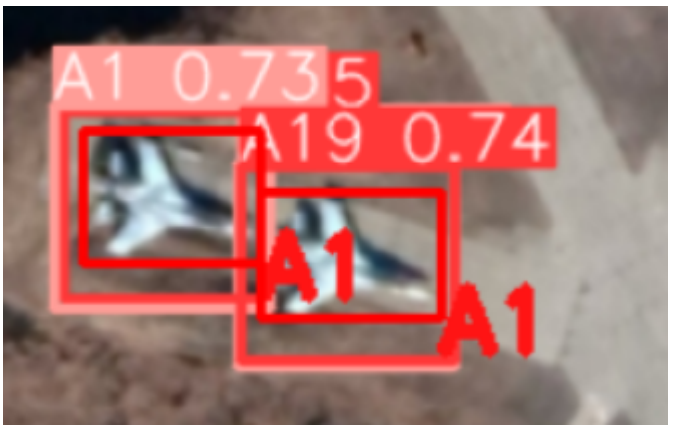


Speaking of the algorithm I've used in previous model, for fast retrieval of misclassified images, I've counted only 115 of them using this model compared to 140 of them using previous model.

Current model also recognized 3 out of 4 planes on the hard case image.



It means, that after hyperparameters tuning the model could enhance its performance even more.
Unfortunately, it has not stopped doing two predictions for one objects on image (though, using YOLOv5m it happens rarely).
E.g.



This problem could be solved by further training of the model I believe. But we can also use agnostic non max suppression as described earlier. It will generally increase a precision of our model (since we will have less False Positives) but it will also increase a recall. I've validated the model with such parameters and the results I've obtained were 0.91 in Precision and 0.868 in Recall.

It is worth mentioning that there might be slight problem with the dataset.

Here, for example, our model has recognized an end of the plane at the left side and the front of the plane on the right side. But, there is no bounding boxes for those parts in xml files. This means that dataset might require some processing and deeper analysis.

Task 7: Integrating model into automatic object recognition system

As we have satellites images, I assume that we will not use object recognition on some flying device (that would mean that we might need a smaller version). Personally, I would host this model on some devops tool like aws or

kubernetes and I would do a simple API call with an image(s) in it and some parameters for the model (e.g. classification or recognition task, agnostic nms etc…)

Conclusion

During this test task I've trained several YOLO models to solve the Object detection and Recognition problem on Military Aircraft Recognition dataset. The results I've got were Precision: 0.907, Recall: 0.871, F1-Score: 0.874. The main problem of the model was proposal of several bounding boxes for one object, that shows that model cannot distinguish some of the classes. Proposal for fixing this would be:

1) Hyperparameters tuning (This was not done here)
2) Larger model
3) Ensemble with smaller model. The algorithm is next: We retrieve results from Yolo, if there are two almost similar bounding boxes between hardly distinguishable classes we'll give crop of that plane to a pre trained on those classes simple CNN model, the purpose of which is to distinguish between those two classes

Author: kriuchkvskyi.vlad@gmail.com