

Configuration Manual

MSc Research Project
MSc Data Analytics

Venkata Devaraju Nandimandalam
Student ID: X18181422

School of Computing
National College of Ireland

Supervisor: Hicham Rifai

National College of Ireland
MSc Project Submission Sheet
School of Computing



Student Name: Venkata Devaraju Nandimandalam
Student ID: X18181422
Programme: MSc Data Analytics **Year:** 2019-2020
Module: Research Project
Lecturer: Hicham Rifai
Submission Due Date: 28th September 2020
Project Title: Military and Non-Military Vehicle Detection by Faster R-CNN and SSD300 Using Transfer Learning
Word Count: **1149** **Page Count:** **13**

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: N.V. Devaraju
Date: 24th September 2020

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	<input type="checkbox"/>
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	<input type="checkbox"/>
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	<input type="checkbox"/>

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Venkata Devaraju Nandimandalam
Student ID: X18181422

1 Hardware and Software Requirements

Google Collab is a cloud-based environment that is an open platform used to develop deep learning algorithms. Gmail account is required to sign into Google Collab. In the beginning each user is allocated with a default 12.73 GB of RAM which can be upgraded up to 25GB and also 64 GB of Hard Disk Space. The research work is performed using the below-mentioned specifications.

RAM	12.73GB RAM
Hard Disk Space	64 GB
OS	Windows 10
Runtime	GPU

The deep learning models are developed using Detectron2 and MM Detection Frameworks. To install these, the below requirements are to be met (*Detectron2: A PyTorch-based modular object detection library*, 2019)

Detectron2:

Python ≥ 3.6

PyTorch ≥ 1.4

Open CV optional (*Installation — detectron2 0.2.1 documentation*, 2019)

MM Detection:

Python 3.6+

PyTorch 1.3+

CUDA 9.2+

GCC 5+

m2c1-full (*Installation — MMDetection 1.0.0 documentation*, 2018).

Tensor Board is used to Visualize the created model

Labelling Tool is used to Label the images.

2 Data Collection and Preparation

Data Collected from ImageNet and VIVID Vehicle tracking website as shown in figure 1 and figure 2. Data is publicly available for research purposes.

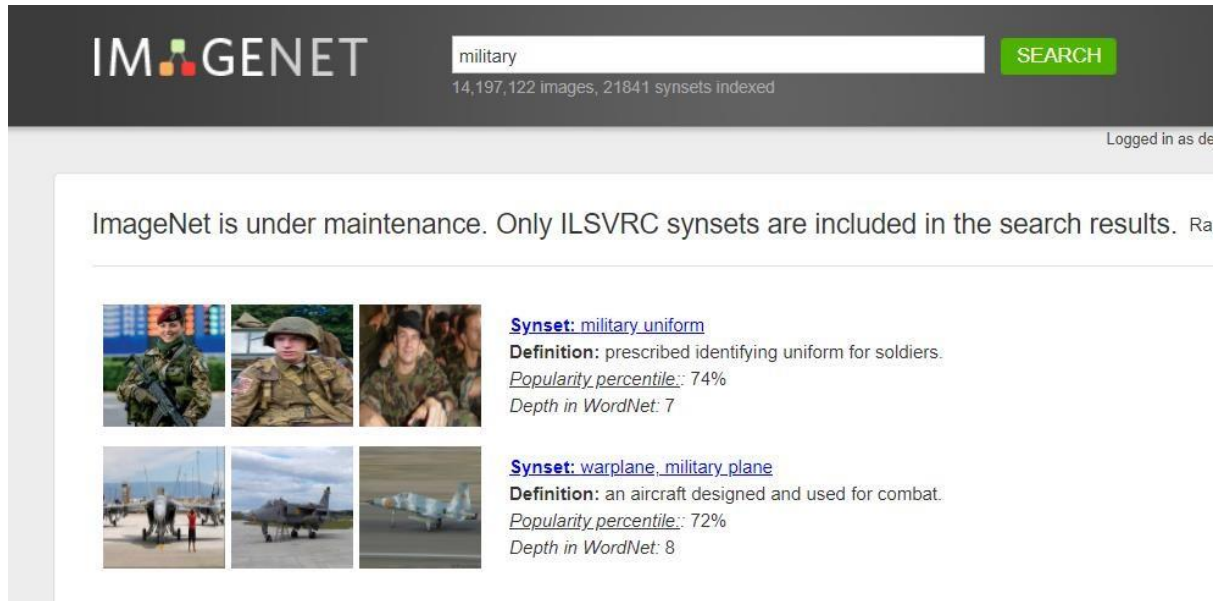


Figure 1: Data Collected From ImageNet

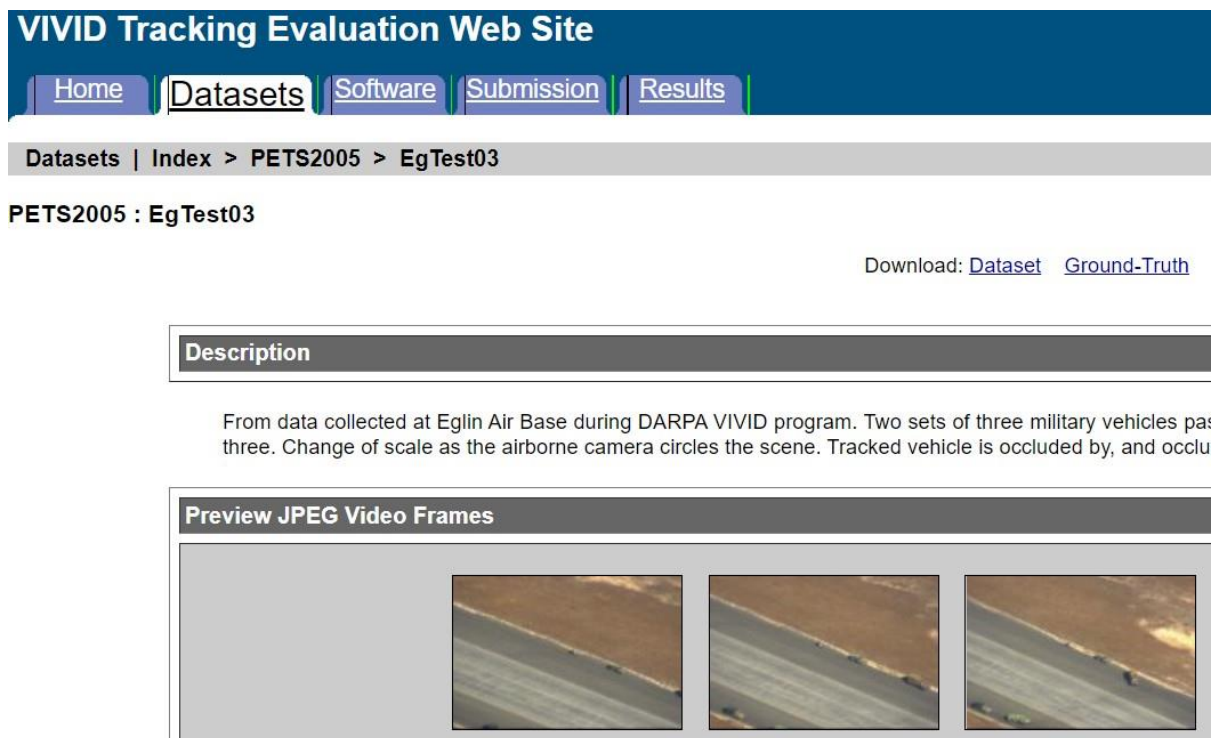


Figure 2: Data Collected From VIVID Website

The gathered images are labelled according to each class using the Labelling tool and saved in pascalvoc format as shown in figure 3 (Pokhrel, 2020).

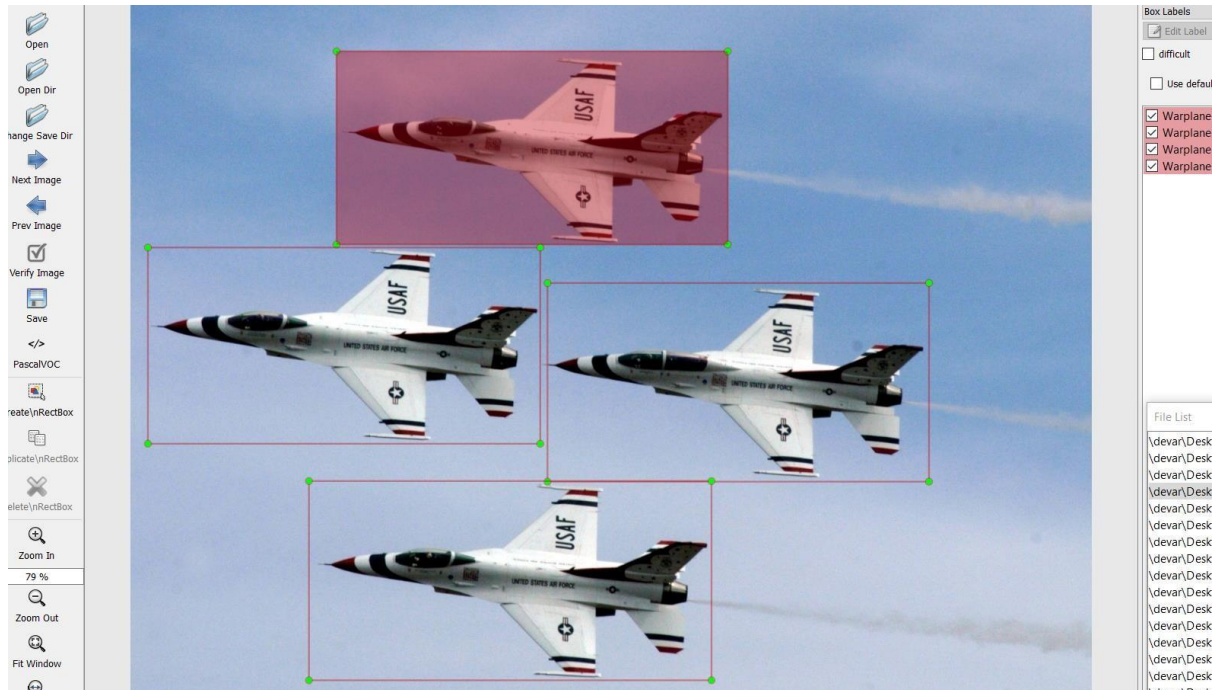


Figure 3: Labeling images

For creating COCO JSON file python code is written in google collab the figure 4 shows mounting google drive to collab to load data.

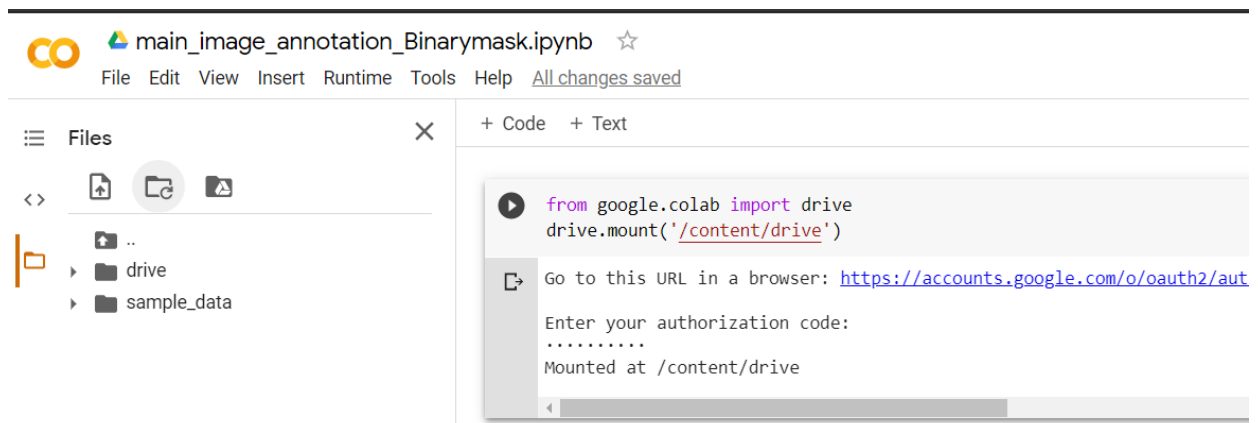


Figure 4: Mounting Google Drive

After loading data, binary mask images are created. These are helpful to generate JSON file and it contains ground truths and boundary box values, etc. In mask images, the white colour box indicates the object and the black colour indicates the background.

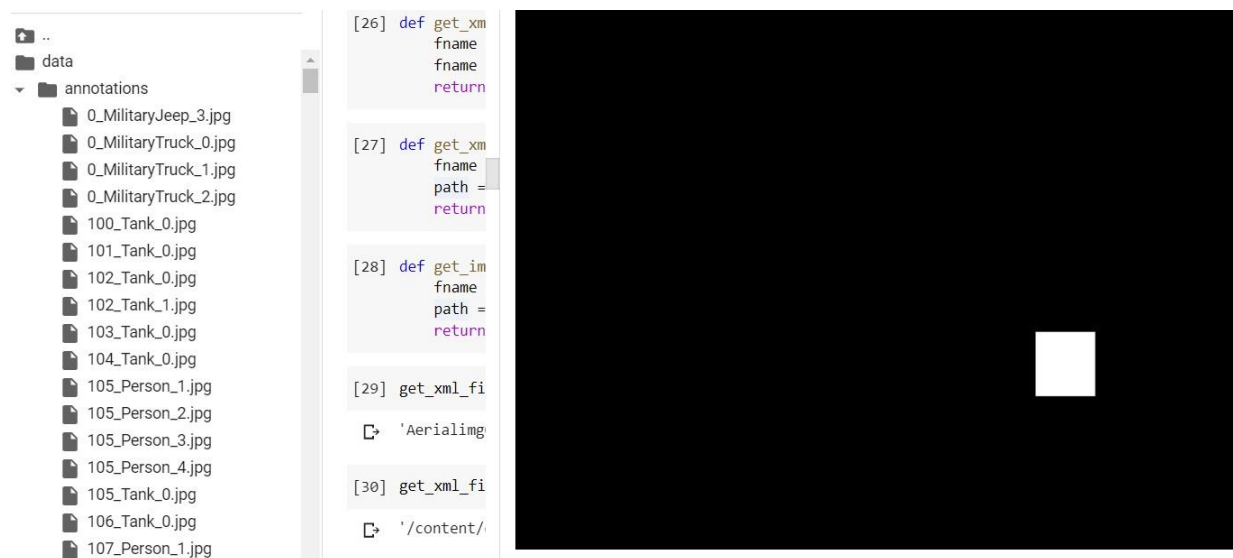


Figure 5: Binary Mask Images

Below figure 6 represents the creation of the coco Json file. Three files are generated separately to train, validate, and test.

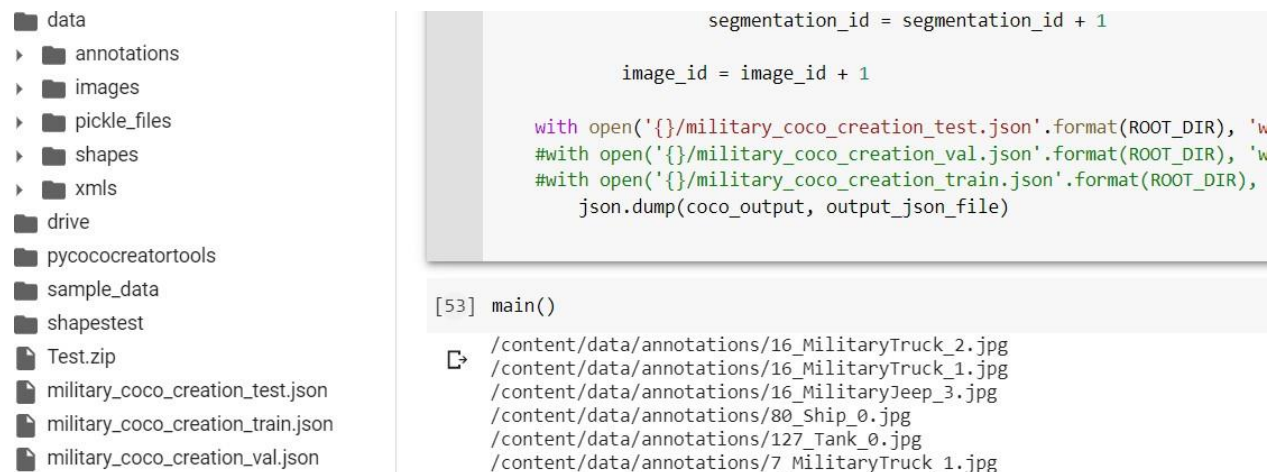


Figure 6: coco json file

3 Implementation and Evaluation of Models

Faster R-CNN model is implemented using Detectron2 library. The following figure 7 displays the Environment setup of detectron2.

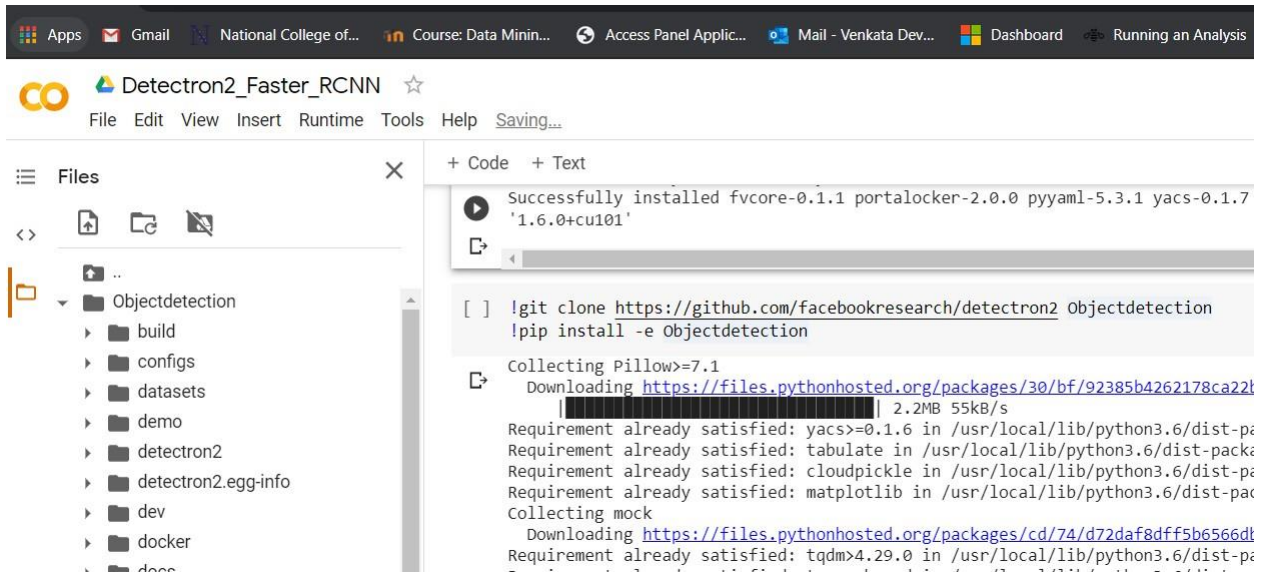


Figure 7: Detectron2 Setup

Figure 8 shows the required libraries and loading the data from drive to object detection directory.

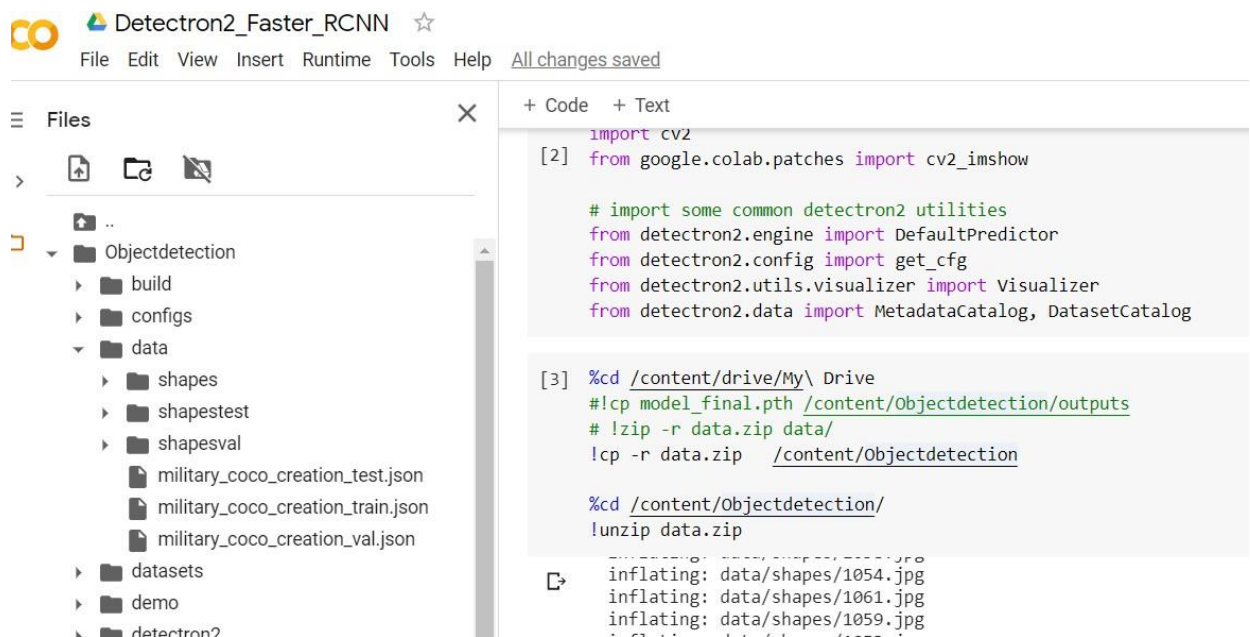


Figure 8: Importing Libraries

Figure 9 shows registering the JSON files and image shapes under the coco instances and the metadata catalog contains dataset information like no of classes etc which can be useful to visualize and evaluate.

```
[8] %cd /content/Objectdetection
    /content/Objectdetection

[9] from detectron2.data.datasets import register_coco_instances
    register_coco_instances("military_image_train", {}, "data/military_coco_creation_train.json", "da
    register_coco_instances("military_image_val", {}, "data/military_coco_creation_val.json", "data/s
    register_coco_instances("military_image_test", {}, "data/military_coco_creation_test.json", "data

[10] from detectron2.data import DatasetCatalog, MetadataCatalog
    military_image_metadata = MetadataCatalog.get("military_image_train")
    dataset_dicts = DatasetCatalog.get("military_image_train")

    [08/09 10:12:10 d2.data.datasets.coco]: Loaded 1100 images in COCO format from data/military_coco_

[11] from detectron2.data import DatasetCatalog, MetadataCatalog
    military_image_metadata_test = MetadataCatalog.get("military_image_test")
    dataset_dicts_test = DatasetCatalog.get("military_image_test")

    [08/09 10:12:17 d2.data.datasets.coco]: Loaded 200 images in COCO format from data/military_coco_
```

Figure 9: Data Registering

Randomly visualizing the ground truth annotations with class names in figure 10.

```
import random

for d in random.sample(dataset_dicts, 15):
    img = cv2.imread(d["file_name"])
    visualizer = Visualizer(img[:, :, :-1], metadata=military_image_metadata, scale=0.8)
    vis = visualizer.draw_dataset_dict(d)
    cv2.imshow(vis.get_image()[:, :, :-1])
```




Figure 10: Visualizing Annotations

Preparing the Faster R-CNN model for training as shown in figure 11 using config file and training data using pre-trained weights from the model zoo with learning rate, batch size, and with 20k iterations.

```

from detectron2.engine import DefaultTrainer
#from detectron2.evaluation.coco_evaluation import COCOEvaluator
import os

cfg = get_cfg()
cfg.merge_from_file("configs/COCO-Detection/faster_rcnn_R_101_FPN_3x.yaml")
cfg.DATASETS.TRAIN = ("military_image_train",)
cfg.DATASETS.TEST = ("military_image_train",)

cfg.DATALOADER.NUM_WORKERS = 2
cfg.MODEL.WEIGHTS = "detectron2://COCO-Detection/faster_rcnn_R_101_FPN_3x/137851257/model_final_f
cfg.SOLVER.IMS_PER_BATCH = 2
cfg.SOLVER.BASE_LR = 0.01

#cfg.SOLVER.WARMUP_ITERS = 1000
cfg.SOLVER.MAX_ITER = 20000 #adjust up if val mAP is still rising, adjust down if overfit
cfg.SOLVER.GAMMA = 0.05
cfg.MODEL.ROI_HEADS.BATCH_SIZE_PER_IMAGE = 128
cfg.MODEL.ROI_HEADS.NUM_CLASSES = 8
cfg.TEST.EVAL_PERIOD = 10000

```

Figure 11: Training the Faster R-CNN model

In figure 12 we can see the Faster R-CNN model with FPN using ResNet backbone.

```

os.makedirs(cfg.OUTPUT_DIR, exist_ok=True)
trainer = CocoTrainer(cfg)
trainer.resume_or_load(resume=False)

trainer.train()

```

```

... [08/09 10:22:30 d2.engine.defaults]: Model:
GeneralizedRCNN(
  (backbone): FPN(
    (fpn_lateral2): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1))
    (fpn_output2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (fpn_lateral3): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1))
    (fpn_output3): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (fpn_lateral4): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1))
    (fpn_output4): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (fpn_lateral5): Conv2d(2048, 256, kernel_size=(1, 1), stride=(1, 1))
    (fpn_output5): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (top_block): LastLevelMaxPool()
  (bottom_up): ResNet(
    (stem): BasicStem(
      (conv1): Conv2d(
        3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False
        (norm): FrozenBatchNorm2d(num_features=64, eps=1e-05)
      )
    )
    (res2): Sequential(
      (0): BottleneckBlock(

```

Figure 12: FPN using ResNet Backbone

Training data Average Precision and Average Recall values are calculated as shown in figure 13 at different IOU thresholds and at different object sizes. For each class, average precision value is calculated (COCO Consortium, 2016).

```

p All changes saved
+ Code + Text
index created!
Running per image evaluation...
Evaluate annotation type *bbox*
COCOeval_opt.evaluate() finished in 0.16 seconds.
Accumulating evaluation results...
COCOeval_opt.accumulate() finished in 0.03 seconds.
Average Precision (AP) @[ IoU=0.50:0.95 | area= all | maxDets=100 ] = 0.820
Average Precision (AP) @[ IoU=0.50 | area= all | maxDets=100 ] = 0.997
Average Precision (AP) @[ IoU=0.75 | area= all | maxDets=100 ] = 0.984
Average Precision (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.768
Average Precision (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.754
Average Precision (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.847
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets= 1 ] = 0.671
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets= 10 ] = 0.859
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets=100 ] = 0.859
Average Recall (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.788
Average Recall (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.798
Average Recall (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.883
[08/09 12:36:29 d2.evaluation.coco_evaluation]: Evaluation results for bbox:
| AP | AP50 | AP75 | APs | APm | APl |
|-----|-----|-----|-----|-----|-----|
| 81.979 | 99.732 | 98.353 | 76.764 | 75.411 | 84.729 |
[08/09 12:36:29 d2.evaluation.coco_evaluation]: Per-category bbox AP:
| category | AP | category | AP | category | AP |
|-----|-----|-----|-----|-----|-----|
| Warplane | 75.061 | Plane | 87.552 | Tank | 86.589 |
| Warship | 86.239 | Ship | 88.886 | Person | 72.403 |
| MilitaryTruck | 81.955 | MilitaryJeep | 77.149 |
[08/09 12:36:29 d2.engine.defaults]: Evaluation results for military_image_train in csv format:
[08/09 12:36:29 d2.evaluation.testing]: cypypaste: Task: bbox
[08/09 12:36:29 d2.evaluation.testing]: cypypaste: AP,AP50,AP75,APs,APm,APl
[08/09 12:36:29 d2.evaluation.testing]: cypypaste: 81.9793,99.7321,98.3534,76.7642,75.4107,84.7289
[08/09 12:36:29 d2.utils.events]: eta: 0:00:00 iter: 19999 total_loss: 0.116 loss_cls: 0.025 loss_
[08/09 12:36:29 d2.engine.hooks]: Overall training speed: 19997 iterations in 2:09:50 (0.3896 s / it)
[08/09 12:36:29 d2.engine.hooks]: Total training time: 2:13:40 (0:03:50 on hooks)

```

Figure 13: Precision and Recall values on Train Data

Visualizing the precision and recall values of trained Faster R-CNN model using the Tensor Board as shown in figure 14.

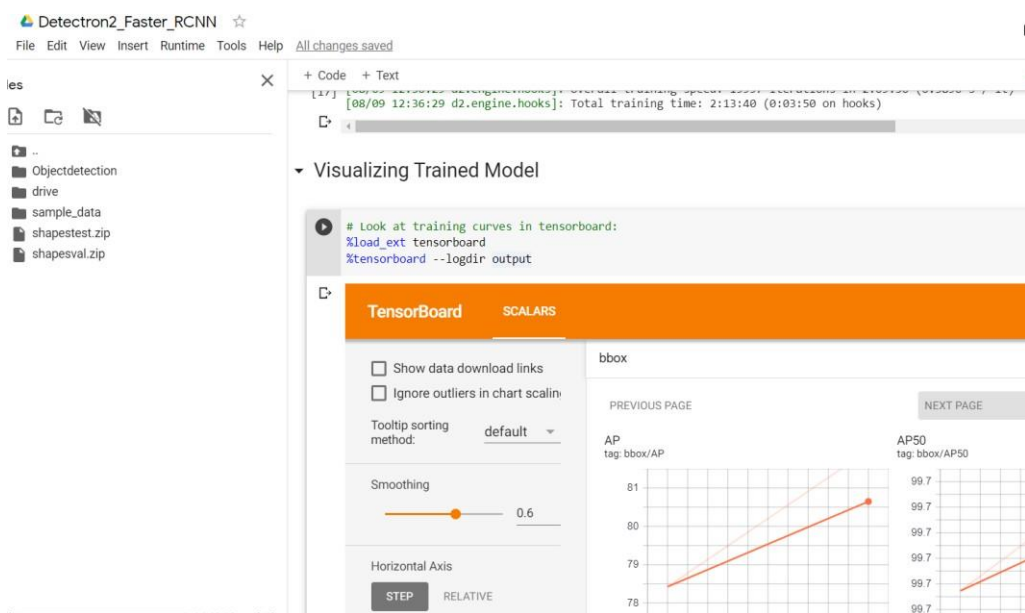


Figure 14: Visualizing Results

Output Predictions of the created Faster R-CNN model on test data is shown in figure 15.

Visualizing the Test Output Predictions

```

from detectron2.utils.visualizer import ColorMode
#dataset_dicts = get_military_dicts("military_image_train")
for d in random.sample(dataset_dicts_test, 15):
    im = cv2.imread(d["file_name"])
    outputs = predictor(im)
    v = Visualizer(im[:, :, ::-1],
                  metadata=military_image_metadata_test,
                  scale=0.8,
                  # instance_mode=ColorMode.IMAGE_BW # remove the colors of unsegmented pixels
    )
    v = v.draw_instance_predictions(outputs["instances"].to("cpu"))
    cv2.imshow(v.get_image()[:, :, ::-1])

```

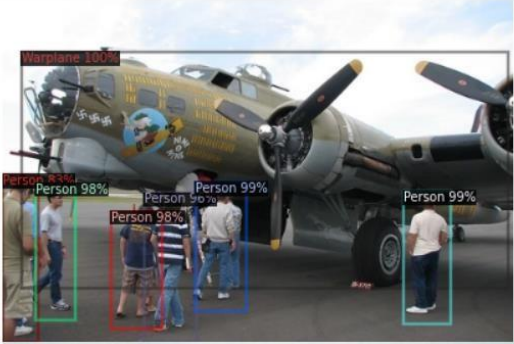


Figure 15: Faster R-CNN Predictions

As shown in figure 16, predicting the objects from an image taken randomly from google to test the Faster R-CNN model performance.

Files

- Objectdetection
- drive
- sample_data
 - images.jfif
 - images1.jfif
 - shapestest.zip
 - shapesval.zip

+ Code + Text

Random Image Taken From Google

```

from detectron2.utils.visualizer import ColorMode

im = cv2.imread('/content/images1.jfif')
outputs = predictor(im)
v = Visualizer(im[:, :, ::-1],
              metadata=military_image_metadata,
              scale=0.90,
              # instance_mode=ColorMode.IMAGE_BW # remove t
)
v = v.draw_instance_predictions(outputs["instances"].to("cpu"))
cv2.imshow(v.get_image()[:, :, ::-1])

```

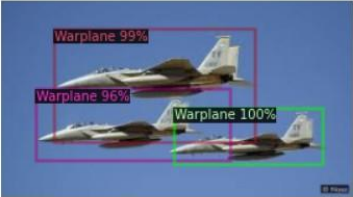


Figure 16: Tesing on Google Images

Creating the Environment to train SSD300 model using MM Detection as shown in figure 17 and downloading the required libraries.

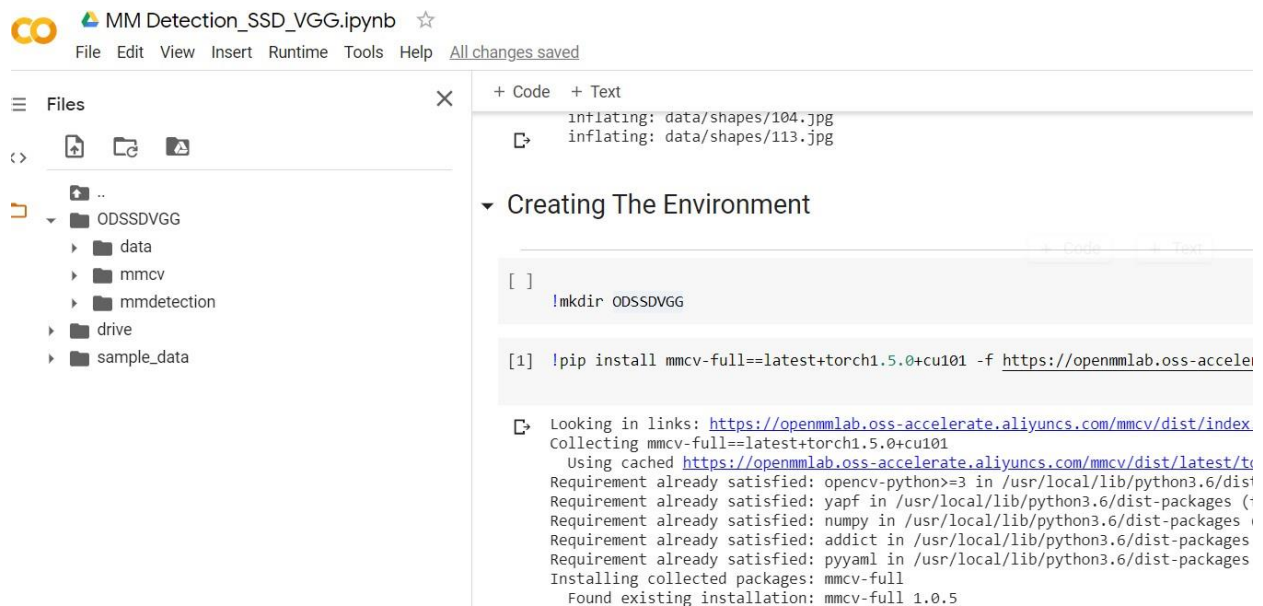


Figure 17 : Creating the Environment

Training the SSD300 model using train.py and passing the config file SSD300_VGG_16.py which contains model parameters and in figure 18 it shows Cuda, PyTorch, python, and GCC versions.

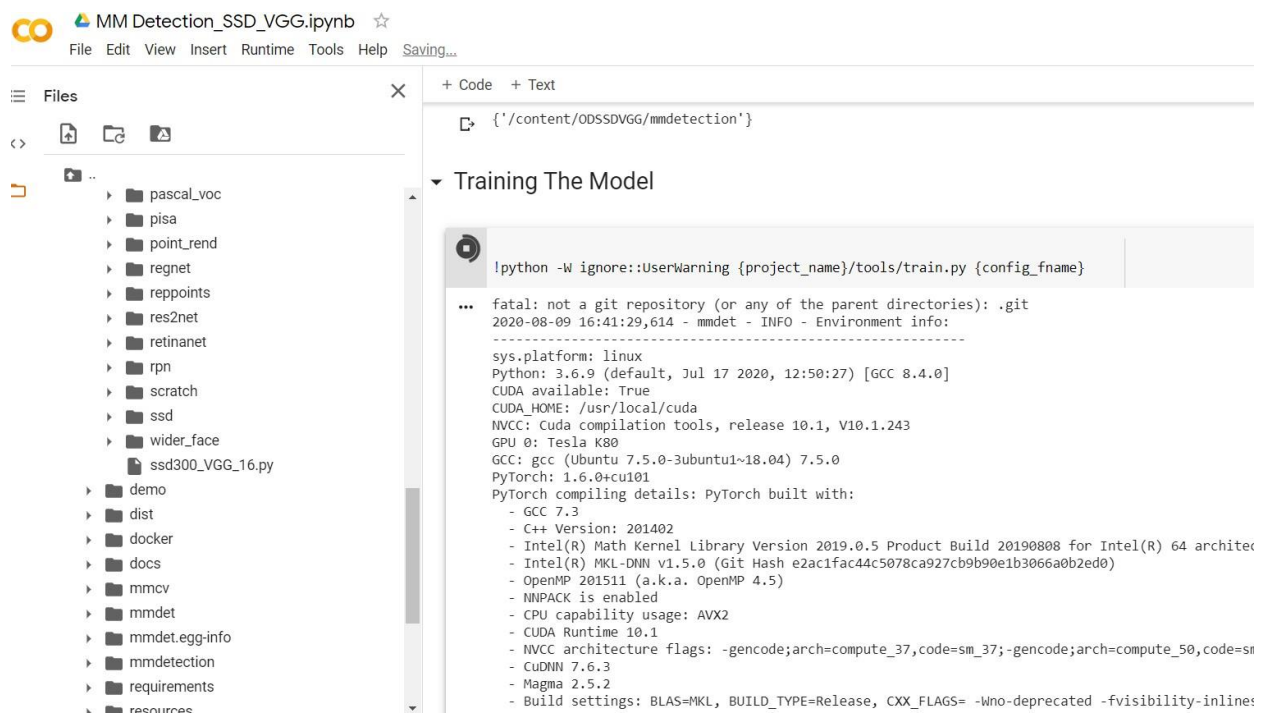


Figure 18: Training SSD300

SSD300 model Average Precision and Average Recall values are displayed as shown in figure 19.

```

X T CODE T TEXT
[17] Average Recall (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=1000 ] = 0.684
Average Recall (AR) @[ IoU=0.50:0.95 | area= large | maxDets=1000 ] = 0.684
2020-08-09 18:24:40,833 - mmdet - INFO - Saving checkpoint at 30 epochs
[>>] 1100/1100, 17.2 task/s, elapsed: 64s, ETA: 0s2020-08-09 18:25:45,594 - mmdet
Loading and preparing results...
DONE (t=0.05s)
creating index...
index created!
Running per image evaluation...
Evaluate annotation type *bbox*
DONE (t=1.90s).
Accumulating evaluation results...
DONE (t=0.45s).
Average Precision (AP) @[ IoU=0.50:0.95 | area= all | maxDets=100 ] = 0.680
Average Precision (AP) @[ IoU=0.50 | area= all | maxDets=1000 ] = 0.924
Average Precision (AP) @[ IoU=0.75 | area= all | maxDets=1000 ] = 0.798
Average Precision (AP) @[ IoU=0.50:0.95 | area= small | maxDets=1000 ] = 0.701
Average Precision (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=1000 ] = 0.517
Average Precision (AP) @[ IoU=0.50:0.95 | area= large | maxDets=1000 ] = 0.617
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets=100 ] = 0.736
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets=300 ] = 0.736
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets=1000 ] = 0.736
Average Recall (AR) @[ IoU=0.50:0.95 | area= small | maxDets=1000 ] = 0.735
Average Recall (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=1000 ] = 0.584
Average Recall (AR) @[ IoU=0.50:0.95 | area= large | maxDets=1000 ] = 0.687

```

Figure 19: Precision and Recall Values for SSD300

Visualizing the model mean average precision and recall values using the tensor board as shown in figure 20.

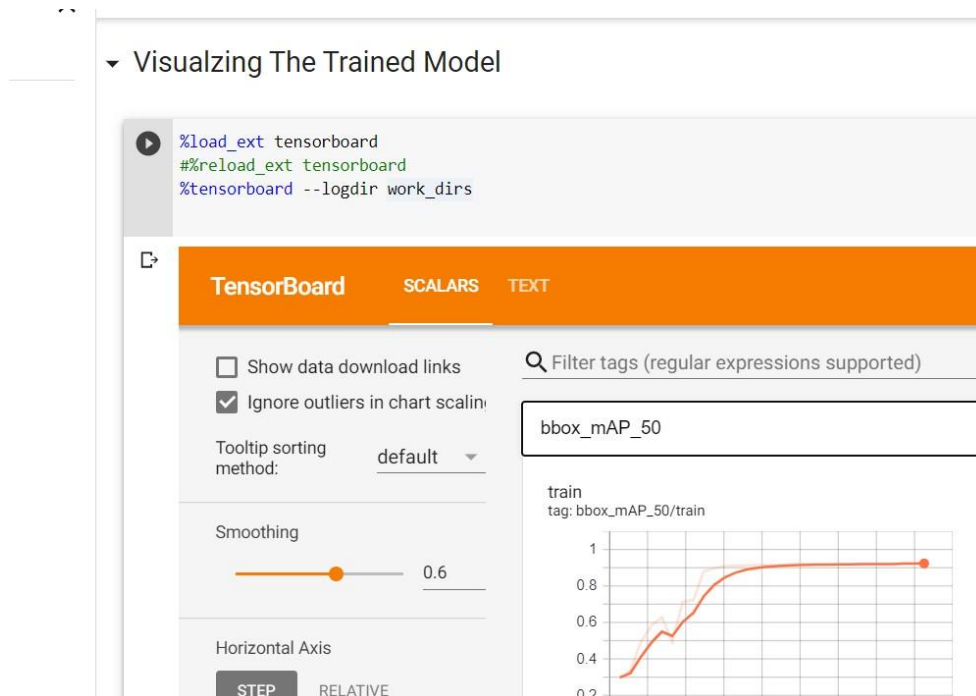


Figure 20: Visualizing precision and recall values

SSD300 model output predictions are shown in figure 21. Threshold score was set to 0.6. The model show boundary boxes with a score greater than or equal to 0.6.

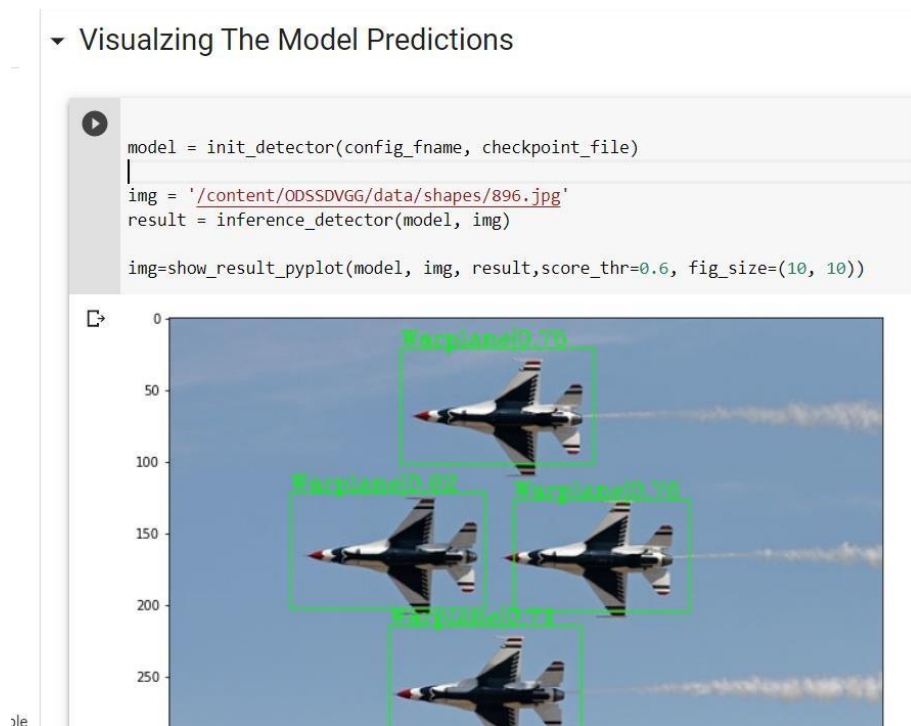


Figure 21: SSD300 Model Predictions

Testing SSD300 model performance on the image taken from google as shown in figure 22.

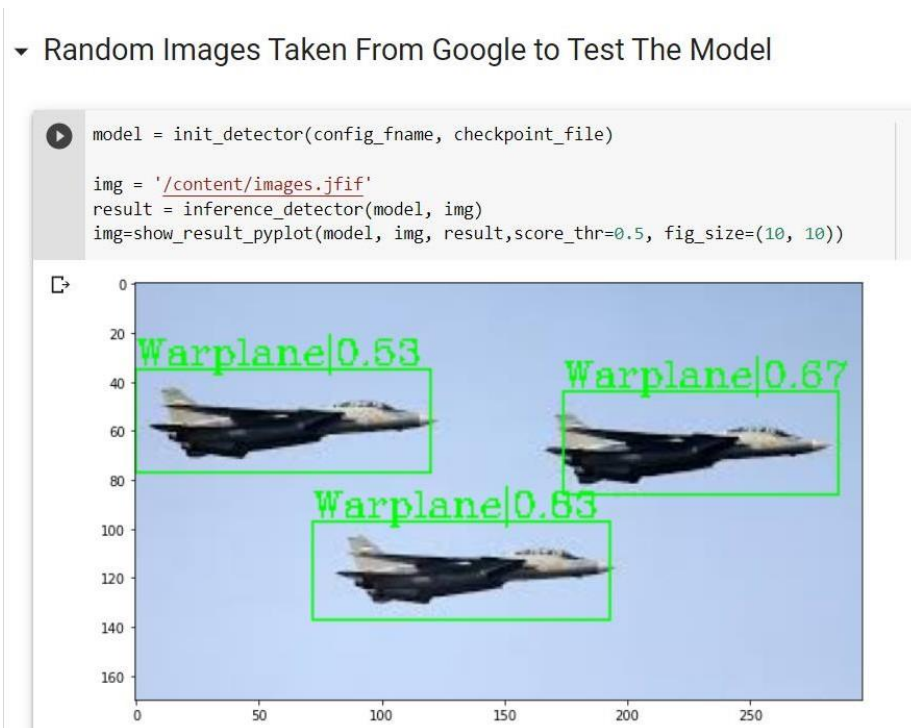


Figure 22: Testing on Google Images

References

COCO Consortium (2016) *COCO - Common Objects in Context*. Available at: <https://cocodataset.org/#detection-eval> (Accessed: 13 August 2020).

Detectron2: A PyTorch-based modular object detection library (2019). Available at: <https://ai.facebook.com/blog/-detectron2-a-pytorch-based-modular-object-detection-library/> (Accessed: 13 August 2020).

Installation — detectron2 0.2.1 documentation (2019). Available at: <https://detectron2.readthedocs.io/tutorials/install.html> (Accessed: 13 August 2020).

Installation — MMDetection 1.0.0 documentation (2018). Available at: <https://mmdetection.readthedocs.io/en/v1.2.0/INSTALL.html> (Accessed: 13 August 2020).

Pokhrel, S. (2020) *Image Data Labelling and Annotation — Everything you need to know* | by Sabina Pokhrel | *Towards Data Science, towards data science*. Available at: <https://towardsdatascience.com/image-data-labelling-and-annotation-everything-you-need-to-know-86ede6c684b1> (Accessed: 13 August 2020).