Weapon detection

Nizar Assad

BSc Computer Science

Millennium Point

School of Computing and Digital Technology

Faculty of Computing, Engineering and the Built Environment

Birmingham City University

Submitted April 2022

Abstract

Object detection plays a major role in helping out police investigations by making the evidence analysis time shorter. The meeting point between object detection and digital forensic science is the ability to detect objects of interest (evidences) in videos or in a large number of images with the main advantage of saving time. We trained our model using our own custom dataset by gathering and manually annotating pictures of handguns and knives and merging them using the most recent technologies, to detect the two objects that are most of the time responsible for a crime and that needs to be outlined and detected. For this purpose, we propose an object detection model with the highest possible accuracy and the lowest possible inference time. Two approaches have been compared taking a Faster RCNN model by TensorFlow (tf2) and a YOLO model by Pytorch (yolov5) for inference on image and video footage. The best result was achieved with the yolov5 model with a mean average precision of 92% and an accuracy of 84%.

List of keywords: Object detection, surveillance camera, CCTV, guns, knives.

Acknowledgements

First and foremost, I would like to thank my advisor, Prof. Abdel-Rahman Tawil, for his supervision throughout the course of my studies at Birmingham City University. Prof. Edlira Vakkaj has tirelessly provided his encouragement and guidance, which has helped me to define my research goals and to shape the scope and focus of this dissertation. His suggestions and careful critique during all stages of this dissertation were indispensable to the creation of this document. In this regard, I would also like to thank Dr. Jean Grey for her detailed reading and guidance towards a more cohesive, structurally-sound work. I very much appreciate the thoughtful reading and suggestions for improvements and future work provided by Dr. Khalid Ismail.

Table of Contents

Glossary

“mAP”	Mean average precision
PANet	incorporated in the model to enhance the process of instance segmentation
ONNX	Open neural network exchange
TF	Tenserflow object detection algorithm
Yolo	You only look once (object detection algorithm)

List of Figures

List of tables

1 Introduction

Undoubtedly, the issue of increasing security in public places has always been the focus of researchers and relevant officials. Significant measures have been taken to increase security in places such as the metro, the airport and places of pilgrimage, including inspection systems, the installation of surveillance cameras and the deployment of an officer. But all of the above is done by human resources, which has many downsides such as fatigue, distraction and many other factors affecting the quality of monitoring. One of the security measures is carrying a weapon and pulling a gun. Our main goal is to make a system operating at a fast pace to report the object that is detected as an alarm notification along with the output which is the image containing a bounding box on the detected weapon.

The idea experimented in this study is to employ deep learning algorithms in security cameras and to identify firearms and knives. The suggested system can provide as much protection as feasible in addition to monitoring public spaces. As armed robbers prohibit workers from contacting the authorities, banks, goldsmiths, chain shops, and other businesses aid, if an automatic weapon detection system detects a weapon, it immediately alerts security agents, in addition with providing the needed data as an evidence.

In this study, a version of the Yolov5 deep neural network was used to detect weapons. And the performance of the network was compared with two other newly created versions like TF2 by TensorFlow (TensorFlow Core, 2022) and VGG 16, these two models use a different technology which is the region based convolutional neural network RCNN. Finally, the Yolov5 outperformed the competition and is the proposed system of automatic detection of pistols and knives we decide to use after training with a custom dataset, our model performed a detection of weapons with 92% precision.

1.1 Problem Definition

Several research have presented and studied the notion of automatic CCTV picture analysis and detection of risky situations. Based on the temporal change of fire intensity, Marbach et al. suggested an automatic fire detection system (Marbach, 2022). Darker et al. introduced the original notion of automated detection of gun crime as part of the MEDUSA project located in the United Kingdom. This group also worked on recognizing signs that might suggest that someone is concealing a handgun (Darker, 2021). One of the most challenging aspects of object recognition is that an item might appear radically different depending on the angle from which it is observed. The photos of a gun object, for example, differ from one another because they depict the pistol from various perspectives. It can look like a rectangular object if captured from above or below, causing a completely different detection result also if filmed from the front where the gun points. Detectors are designed to distinguish things from a variety of angles. The second main challenge for this project is the occlusion, and illumination conditions. Even the angle at which the CCTV is positioned, can sometimes obscure or brighten the object we wish to capture, making it difficult to detect them. These situations are even more problematic in our case by the fact that most CCTV cameras capture in low resolution or employ infrared technology (IR) at night (CCTV Work - CCTV Aware, 2022). Lastly, the speed of inference, detectors for video must be trained to do analysis in a constantly changing environment. It means that, in order to identify moving objects, object detection algorithms must not only accurately label significant items but also be extremely quick during prediction.

1.2 Scope

In our project, we will try to develop object detection system for surveillance camera, they are many types of CCTV, the main two types are the indoor and the outdoor. In this project we will be focusing on the indoor evironments as we can't really deploy the same model for all type of surveillance cameras. The outdoor cameras need a higher a very high accuracy as objects might need to be captured from far distance. The lighting plays a major role of difference, in an outdoor environment, lighting varies every second due to the high activity (cars, walking civilians, phones, day lighting, street light etc…)

1.3 Rationale

Because of the vast quantity of data, we can investigate, use, and manipulate, this project concept was picked. Although numerous ideas have been proposed to benefit from this data, many parts of the process may still be improved to further leverage this information and make the field of CCTV detection more profitable for police investigations. In this project, we'll focus on all the weak points mentioned in the Problem Definition and try to improve them using various techniques from increasing the model speed and accuracy to finding new ways of deploying the model.

1.4 Project Aim and Objectives

The aim of this report is to avoid losing time during the crime investigation, this development intends to analyse the evidences present in a crime scene through the use of object detection.

- **Investigate** the current programs used for object detection and see what can be improved.
- **Evaluate** the existing object detection models that are using conventional neural network and object detection and that are already functionable.
- **Identify** the most effective and adapted tools for creating the model and have a deep look at the Region-based Convolutional Neural Network.
- **Develop and train** an object detection model using the right development environment.
- **Test** the pre-trained method with a video dataset to evaluate the object recognition in different environments.
- **Analyse** the average accuracy of the model.
- **Tune** the architecture for highest average accuracy.
- **Deploy** the finalized model to work.

1.5 Background Information

Most of the experts from national Institutes related to the area of forensics have been interested in this topic in the last decade, which gives us a strong point to make our choice, also chosen for the sake of the citizen security and protection; it is well known that criminal activities are committed to a higher extent in the business sector. Wholesale and retail trade (48.8%), manufacturing industries (11.5%), and transportation and storage (10.6%) (Proyecto, 2022) were the industries with the highest percentages of victims, the most common weapons used in crimes were firearms or explosives (16.2%), blunt objects (6.4%), and knife or sharp blade (4.0%) (Proyecto, 2022).

2 Literature Review

2.1 Themes

A thematic approach has been undertaken to identify the areas that need to be understood to develop the object detection model. From this a number of keywords for each have been used to obtain information from the literature.

The following themes are to be investigated using the related keywords.

- Deep learning:
    - Convolutional neural network
    - “YOLO”
    - Artificial intelligence
    - Object detection
    - Machine learning
- Digital forensics:
    - Crime investigation
    - Evidence analysis
    - Digital forensics
- Video surveillance:
    - CCTV
    - Video surveillance

2.2 Review of Literature

In forensic science, a police officer analysing digital evidence must filter through a large amount of visual information. This raises the question of how we might shorten the time required to complete this task. This report will review current research for the object detection model we’re wishing to design with the use of deep learning, in order to help out police investigations and speed up the process.

Since the 1990s, the globe has been through a digital revolution that has influenced our way of life. CCTV's play a major part in this revolution. Surveillance cameras are almost everywhere, but the exploitation of this data is still not as efficient as it should be (Taylor & Gill, 2014). Using this type of data for a crime investigation can give a big amount of useful information that can help solving cases. To deal with this, we're going to use some deep learning techniques. More precisely, a neural network architecture.

Over the last decade, Artificial intelligence have achieved breakthroughs in a variety of pattern recognition domains, going from image processing to voice recognition (Albawi, Mohammed and Al-Zawi, 2022). This technology is being used and constantly developed by businesses and cities all around the world to decrease and prevent crimes, as well as to solve ongoing investigations more rapidly (Faggella, 2019). Machine learning, which is an AI subset, has impacted the Forensics science in a colossal way, Innovative approaches for digital investigations are being developed at a rapid pace in the domains of computer science and information technology (Qadir & Varol , 2020).

Since the 1970s, the field of digital forensics has evolved to keep up with the wide spread option of technology, and the mean in which these technologies are used for criminal activity and can also be used to prevent crimes. The progression of digital forensics started with skin detection, driven by colour instead of ML, then the document search automation with the use of machine learning which was a text-based search and has marked the beginning of a new era. And finally, this technology has evolved to be in the vision field and detect pictures. This area is still going to be developed much more through time and will have a serious impact in crime prevention and much more (Casey, 2022).

Digital forensic analysis process is composed of three main phases. Data collection, consisting on gathering the information we would like to work on. Examination and analysis, which consist on using the collected data to find useful data for our specific case. Lastly, the reporting phase, focuses on making conclusions and presenting expert testimony.

In terms of applicability to digital forensic, Machine learnings techniques can be interpreted into two different sections. The refiners who keep the knowledge source Current, and the learners which we are interested about, that gathers initial knowledge. For our model, we'll need to use a sub symbolic system as it is one of the best learning forms while dealing with previously unseen data (Mitchell, 2010). Convolutional neural networks are considered the most popular neural network architectures in the deep learning branch (Parthasarathy, 2017).

Faster RCNN was developed by Ross Girshick in (Allegion, 2019), was first introduced in 2015. Since then, many versions came up for a higher computational efficiency, less time to

process tasks and improvement in performance (Ananth, 2022). To locate the object within the picture, all of the RCNN object detection algorithms used regions. The YOLO network does not consider the entire picture. Rather, portions of the image with a high likelihood of containing the item by splitting the input image into SxS grid, outputs a class probability and offset values for the bounding box. The diagram below shows the process of detection:

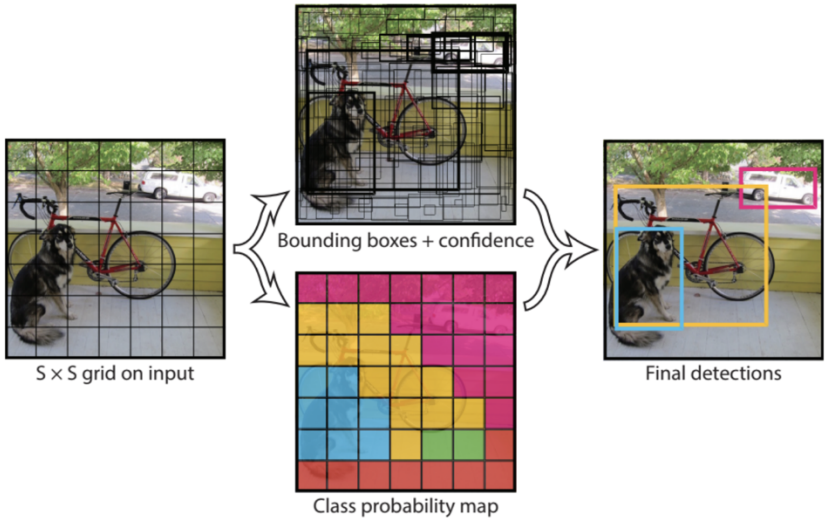


Figure 1: Yolov5 detection process

You Only Look Once, or YOLO, is an object identification that has very distinct technique from region-based algorithms. The bounding boxes and class probabilities for these boxes are predicted by a single convolutional network in YOLO (Gandhi, 2022). In 2019, Arif Warsi has posted a research where he used different videos to validate the results of Yolov3 compared to FASTER RCNN, the detector worked effectively in detecting handguns in a variety of scenarios with varying rotations, sizes, and forms. The findings revealed that Yolov3 was better as a substitute of Faster RCNN. It has a substantially quicker processing time, approximately comparable accuracy, and can be utilized in real time (Warsi, 2022). Another study proved that the fifth version of the yolo model, Yolov5, which is an implementation done in Pytorch by Ultralytics, is until today, the best real time object detection model, due to its inference speed and the accuracy it offers, which is exactly what's needed for a CCTV object detection to work efficiently (Dwivedi, 2022).

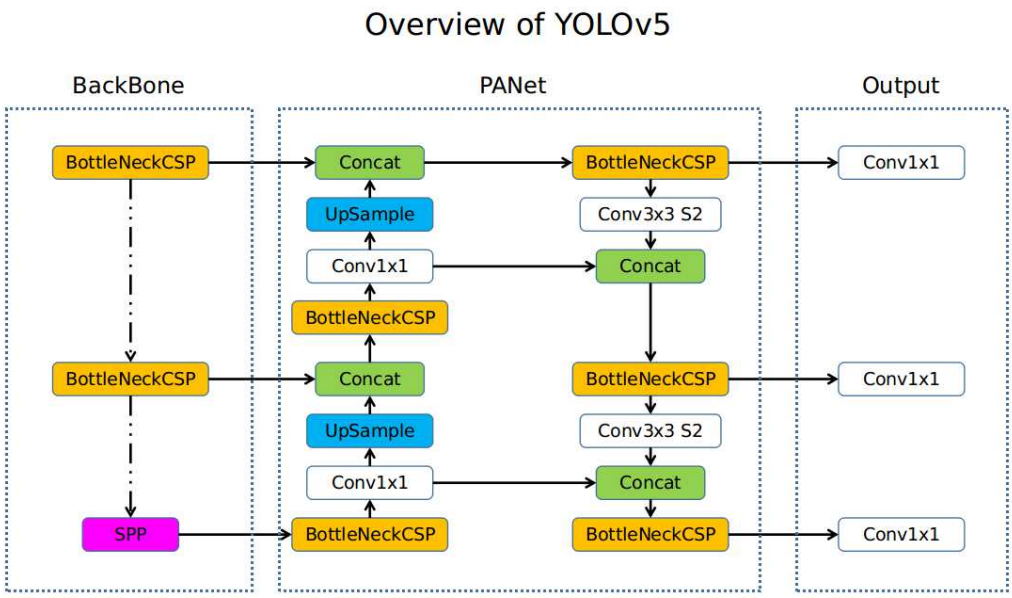


Figure 2: Yolov5 architecture

Yolov5 has four primary sections in its architecture: input, backbone, PANet, and output. The input terminal is mostly used for data pre-processing, such as mosaic data augmentation and adaptive image filling. Yolov5 incorporates adaptive anchor frame computation on the input to adapt to diverse datasets, allowing it to automatically determine the starting anchor frame size as the dataset changes. Multiple convolution and pooling are used by the backbone network to extract feature maps of various sizes from the input image using a cross-stage partial network (CSP) and spatial pyramid pooling. BottleneckCSP is used to minimize the amount of calculation and increase the inference speed, whereas the SPP structure realizes feature extraction from various scales for the same feature map, generating three-scale feature maps, and optimizing detection accuracy. Strong localization features are conveyed from lower feature maps to higher feature maps using the PAN structure. This structure increases the detection capabilities by combining the features gathered from multiple network layers in Backbone fusion. The head output is mostly used to anticipate targets of various sizes on feature maps as a final detection step (Fang, 2021).

Once the Yolov5 has been released, there has been five different model sizes, the best fitted one is the Yolov5s which the smallest and has proven to be the fastest algorithm in terms of processing images. As we can deduct from the diagram below:

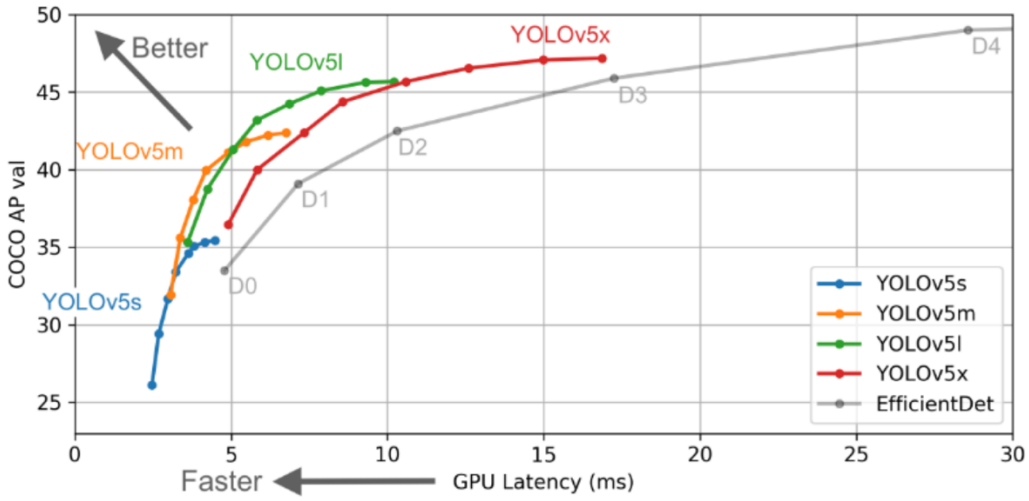


Figure 3:yolov5 stats
(UltraLytics repo)

Without a well labelled and efficient training data, the model wouldn't be useful at all as it is mandatory for any algorithm to work and to be deployed. For this we used Roboflow, a developer framework for improved data collecting, pre-processing, and model training procedures in computer vision, it provides public datasets as well as the ability to submit our own custom data. Roboflow supports a number of different annotation formats. Image orientations, resizing, contrasting, and data augmentations are all processes in the data pre-processing process (Dwyer, 2021). In June 2020, Roboflow has made a partnership with Yolov5 which offers deeper integration between the Roboflow python package and the model training and inference scripts. Having a custom dataset is far more important than any benchmark we can get with the coco dataset, because the obtained result shows how the model will react in a real-life dataset. It's this gap that we aim to bridge within this project (Solawetz, 2020).

A solution to our problem of overloading the human operator is to apply automated image-understanding algorithms, which, rather than replacing the human operator, alert them if a potentially dangerous situation is at hand.

Our system aims on detecting specific objects from a surveillance camera and then sending an alarm to an agent that will take care of reviewing the output data and operate in a regular way to deal with the issue.

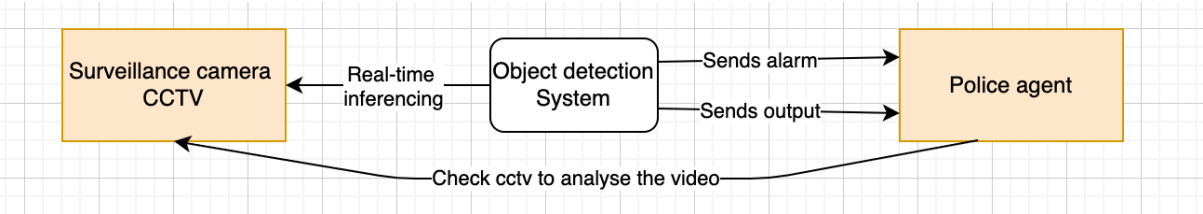


Figure 4: Concept solution

The main part of the project is to work on the object detection model to be fast and efficient to process in case of a situation where human lives are in danger.

3 Method and Implementation

3.1 Introduction

In this section, we'll propose a design process method which consist of developing a trained yolov5 model that will be able to do real-time object detection for guns and knives, we've gathered all the required information, now it's time to use the acquired knowledge in order to have a more concrete view of the overall process of actually building the project.

3.2 Methodology

In this project, we propose the YOLO algorithm, by presenting the architecture in detail through subsections and emerge deeper into the design of train, validate and test activities.

For the sake of this project, we're going to use a deep learning algorithm to extract features in an automatous way and also to self-train in a hierarchical fashion. The best fitted algorithm to use is the YOLOv5, that's because it processes data in a very fast way, which is very crucial and required for our program to run like expected. The starting point for designing the model is to do a requirements analysis. For this, we went through publicly

available CCTV recordings to analyse the data and see what aspects we should work on to improve the detection process. The main points we want to focus on:

- Poor quality of resolution.
- Low resolution.
- Object is visible only for a limited period of time in a scene.

After analysing the data, we have listed a set of requirements that the model should respect.

- The yolov5 model should be of a high accuracy due to the low resolution.
- It should detect objects in real time.
- The model should keep the number of false alarms as low as possible

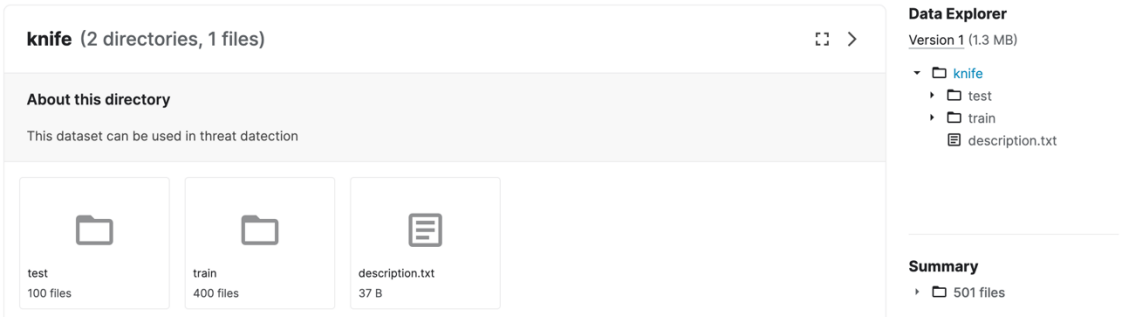
Now that we have a clear idea of the requirements, we can proceed with developing the model. Which starts by creating the dataset.

3.3 Implementation

3.3.1 Datasets

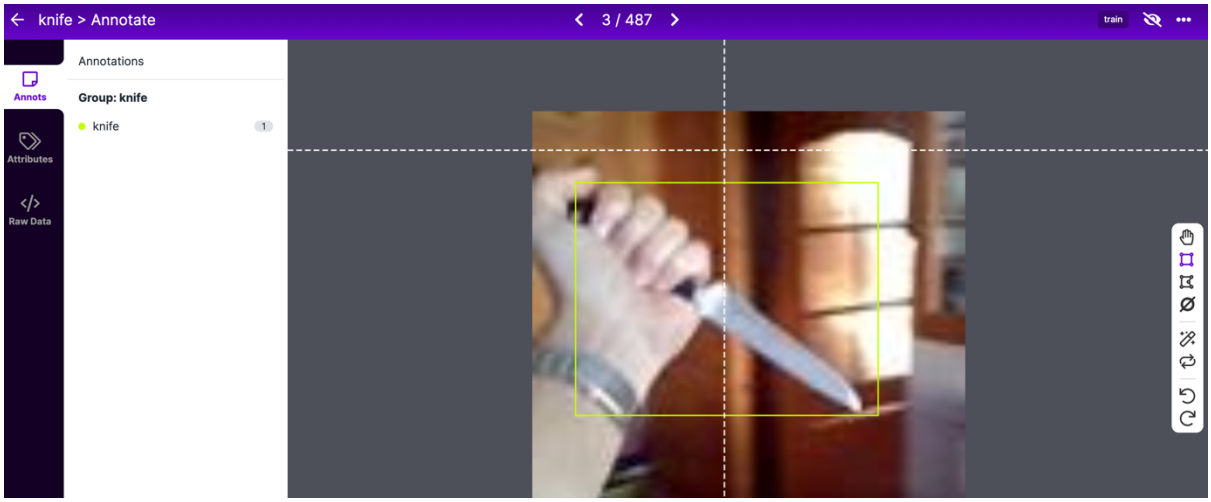
3.3.1.1 Knife dataset:

We have created our own image dataset for knives using Roboflow, we've manage to find a good set of images on Kaggle that we can use. After analysing the activity statistics and comments, we noticed that this dataset was optimized for CCTV knife detection. It contains 500 images which is fairly enough for our model to train well.



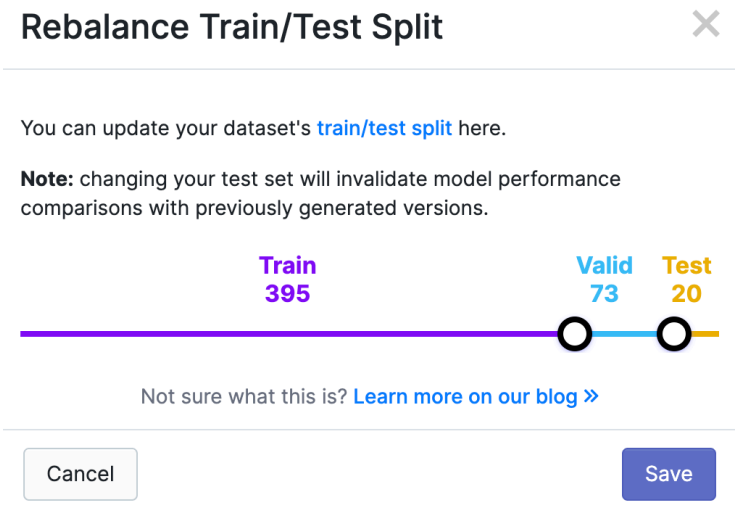
After this we downloaded the dataset to our local machine and started the uploading process to Roboflow. The first step was to manually annotate every single image with a label and

specify the class (Knife). This process takes a while depending on the number of images your dataset contains.



Now that we've labelled all our images and specified the class object for each one, we can move on to the next step of the generate process which is splitting the dataset between test, train and validation sets.

- Training set: Images that trains the model to recognize a specific object.
- Testing set: tests the model on detecting an object after the training is done.
- Validation set: give an estimate of model skill while tuning model's hyperparameters and calculates the precision, recall and accuracy of a model.

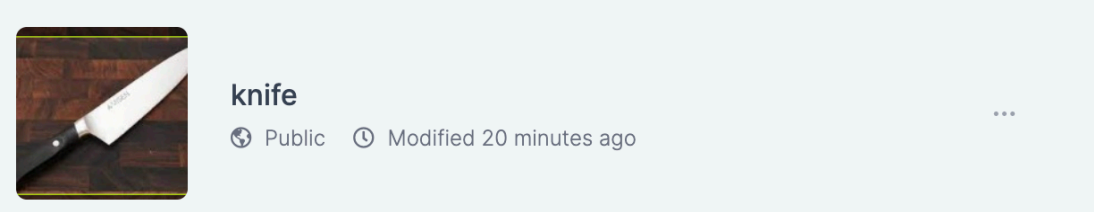


Roboflow offers the rebalance feature, which permits the user to manually select a portion for each set. We've assigned most images for train and valid. The testing part will be done with actual CCTV footage.

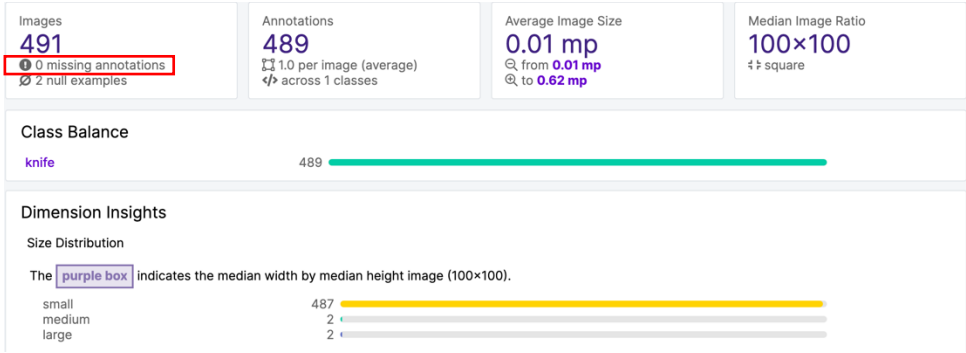
The next step is the Pre-processing which helps decreasing training time and increasing performance by applying image transformations to all images of the dataset. It provides two important features. The first one is the auto-Orient which is activated in our case. This directive speeds up encoding the image to sample data without unwanted artefacts. Roboflow takes care of everything, all we need to do is select “auto-orient”.

The second feature is resizing, it downsizes images for smaller file sizes and faster training. It is by default set to “416 x 416”. We apply it to our dataset as well.

Now that we’ve been through all the steps. We can generate a version of the dataset, accessible from the “Projects” page.



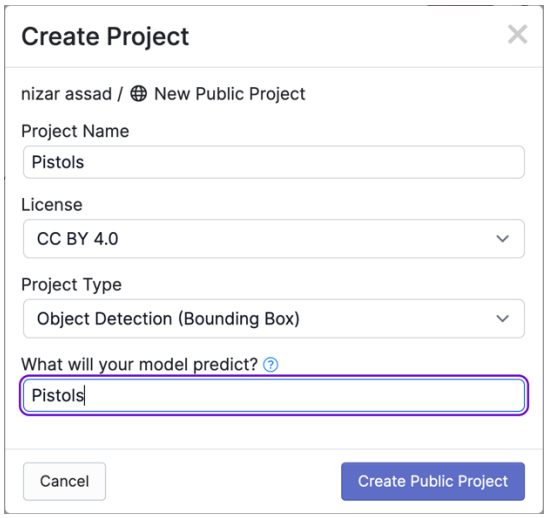
Roboflow enable the user to run a health check on the dataset and check the quality of it.



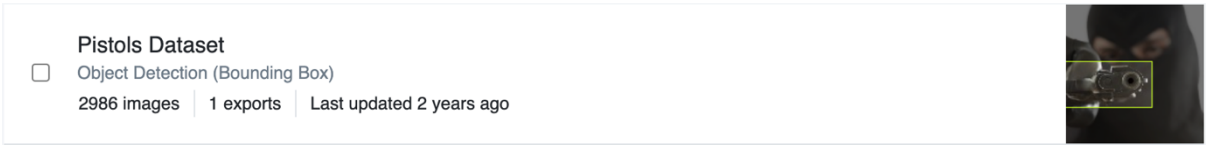
In our case, we can see that the data is clean of unannotated images and well balanced in terms of class and dimensions.

3.3.1.2 Pistol dataset:

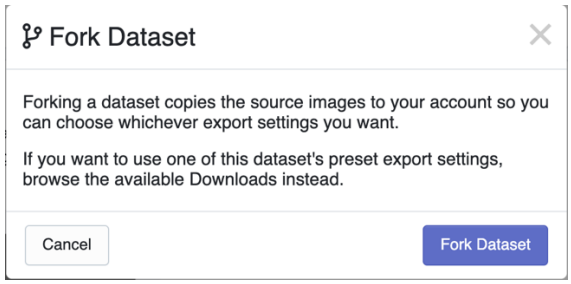
The next thing to do is to get a dataset for guns as we will be working on the detection for two different objects. For this, I found an available annotated public dataset offered in Roboflow, it has a very good feedback and it’s a very famous platform, so we decided to use it. We navigated to the projects page, clicked on “create a project”. Assigned it with the right project type and model predicted and clicked on create public project.



Now that this new Pistol project is created we can choose to use a public dataset instead of importing our own images.



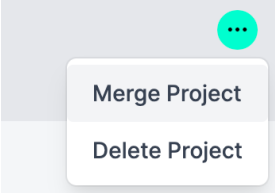
We can fork this dataset to our repository and proceed with this dataset.



We will go through the same steps as for the knife dataset, which includes splitting between train, valid, and test datasets, we used the default splitting to keep the same percentage of each set, same as the knife dataset. Then the pre-processing phase where we applied the exact same effects of resizing and auto-orient, finally we generated the dataset.

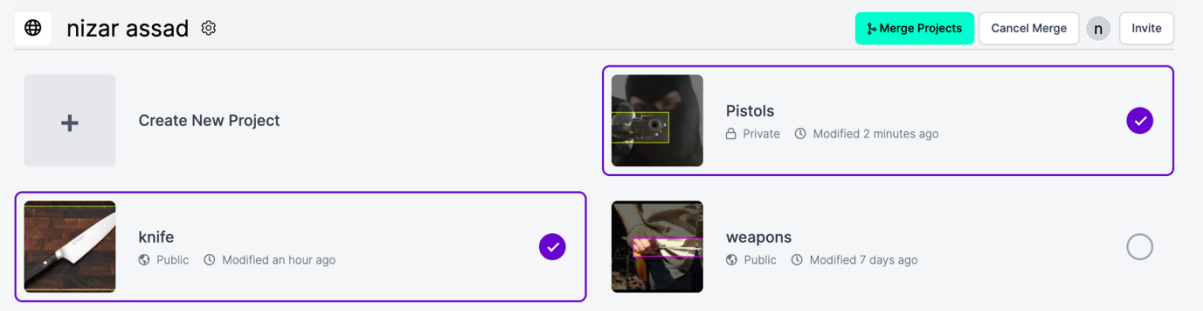
3.3.1.3 Merging the datasets:

Now that we have the two datasets ready to use, Roboflow provides a merging feature which enables to merge different datasets with different object classes to train in one model.

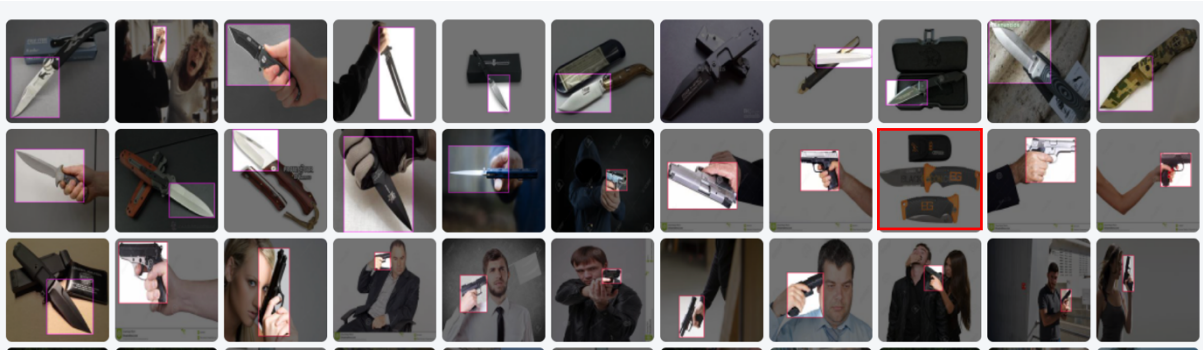


We can merge by selecting the two datasets and click on the green button “Merge Projects”.

All we should do is name the project and we’re good to go with our newly created dataset that we named “weapons”. This dataset contains two object classes that we wish our model to detect.



Here’s a quick look at how the weapon dataset looks like:



Notice the presence of the two objects in the dataset images.

Before using the finalized merged data, we had to go through all the images to see if they were all correctly annotated to avoid any type of negative images as we can notice in the red box above. Making the dataset as perfect as possible is crucial and will help making the model more accurate.

3.3.2 Yolov5 algorithm

In this work, we have used transfer learning for training Yolov5 model for gun/knife detection.

We will do the coding part in Google Colab which is a Jupyter notebook-based runtime environment allowing code to run entirely on the cloud. This helps us train our model efficiently even without a powerful machine or a high-speed internet access.

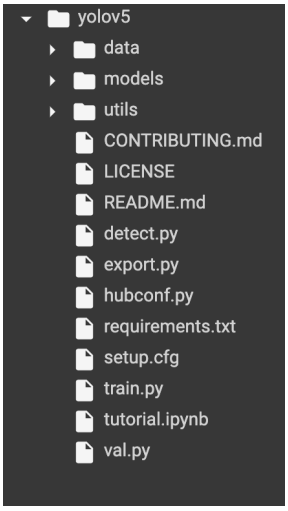
We'll start by installing requirements in the first cell.

- Yolov5 repository: we will use the official yolov5 repository provided on Github which can be cloned to our Notebook content using "git clone" followed by the repository link. (The line must start with "!" to execute command from underlying operating system)
- After cloning the git repository, we'll need to change to the yolov5 directory in order to get access to all files inside it, starting with the "requirements.txt" which contains a bunch of libraries we need for our code to run (numpy, OpenCv, torch, matplotlib...).
    - NumPy: used to perform mathematical operations on arrays.
      Version used: 1.18.5
    - OpenCV: Library of python bindings, used for image processing and computer vision tasks.
      Version used: 4.1.2
    - Torch: Open source machine learning algorithm, a scientific computing framework.
      Version used: 1.7.0
    - Cuda: it provides a driver and runtime API.
      Version used: cu111

  We'll use the "pip" which is a command that looks for the package, resolves the dependencies, and installs everything in the current environment.
- We installed Roboflow which is going to take care of cloning the dataset from the website to our environment.

```
!git clone https://github.com/ultralytics/yolov5
%cd yolov5
%pip install -qr requirements.txt
%pip install -q roboflow
```

Here's a quick look at our project folder structure:



Now that we have installed all needed packages, we're going to start by importing torch to enable GPU usage, then "os" to set up the environment and locate the dataset for training.

```
import torch
import os
```

Next, we want to check what version of torch and cuda is installed in our environment using the "__version__" attribute and then run the "get_device_properties" command to get the running device information as we want to use it later for tuning the model.

```
print(f"torch {torch.__version__} ({torch.cuda.get_device_properties(0).name if torch.cuda.is_available() else 'CPU'})")
```

Now let's run the cell which contain all the above codes we discussed:

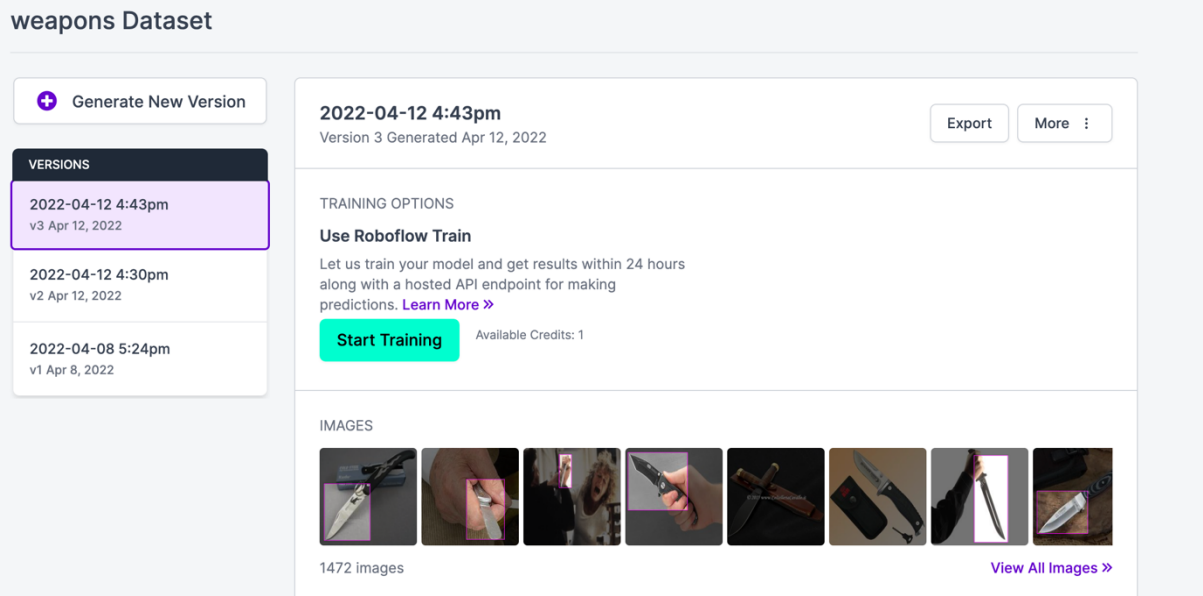
```
Cloning into 'yolov5'...
remote: Enumerating objects: 12816, done.
remote: Counting objects: 100% (17/17), done.
remote: Compressing objects: 100% (14/14), done.
remote: Total 12816 (delta 7), reused 11 (delta 3), pack-reused 12799
Receiving objects: 100% (12816/12816), 11.81 MiB | 24.29 MiB/s, done.
Resolving deltas: 100% (8899/8899), done.
/content/yolov5
     |████████████████████████████████| 596 kB 5.0 MB/s
     |████████████████████████████████| 145 kB 5.1 MB/s
     |████████████████████████████████| 178 kB 44.3 MB/s
     |████████████████████████████████| 1.1 MB 58.7 MB/s
     |████████████████████████████████| 67 kB 6.0 MB/s
     |████████████████████████████████| 54 kB 2.9 MB/s
     |████████████████████████████████| 138 kB 66.4 MB/s
     |████████████████████████████████| 63 kB 1.8 MB/s
  Building wheel for roboflow (setup.py) ... done
  Building wheel for wget (setup.py) ... done
ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This behaviour is the sourc
google-colab 1.0.0 requires requests~=2.23.0, but you have requests 2.27.1 which is incompatible.
datascience 0.10.6 requires folium==0.2.1, but you have folium 0.8.3 which is incompatible.
albumentations 0.1.12 requires imgaug<0.2.7,>=0.2.5, but you have imgaug 0.2.9 which is incompatible.
torch 1.10.0+cu111 (Tesla T4)
```

We'll ignore the error message as it is not affecting our process. Notice the last sentence which shows the torch and cuda versions (1.10.0 + cu111) and also the device property which is T4 in our case.

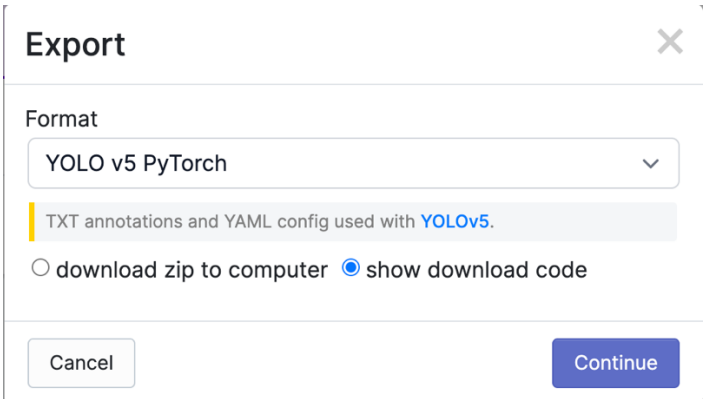
Next step is to import Roboflow and instantiate a Roboflow object with the model format and notebook name in order to make us of our API key.

```
from roboflow import Roboflow
rf = Roboflow(model_format="yolov5", notebook="ultralytics")

upload and label your dataset, and get an API KEY here: https://app.roboflow.com/?model=yolov5&ref=ultralytics
```

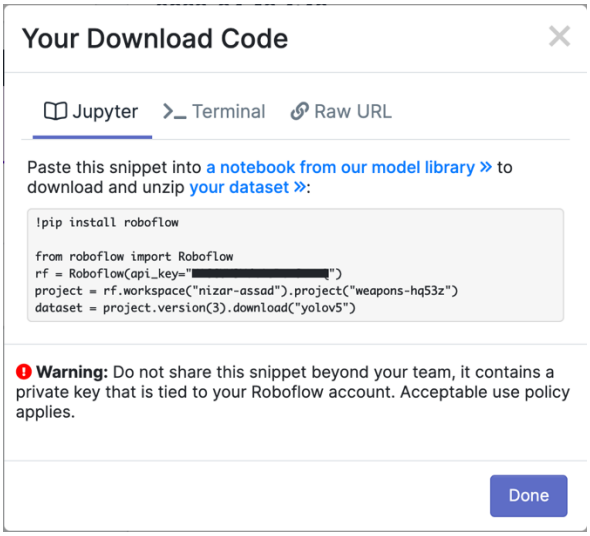
The output gives as a link that redirect us to Roboflow, we'll open the link sign in with our previously created account (where we created the weapon dataset).



Once there, we click on the "export" button which will pop up a small page where we select the export format, which is YOLO v5 PyTorch, and we'll select download code instead of the zip file like shown below.



A download code will be generated in Jupyter format to enable us to use it directly in Colab.



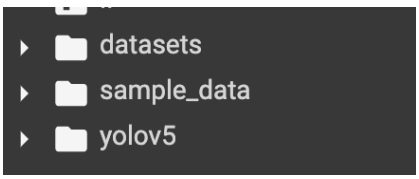
Before using this code, we'll need to go back to Colab and set up the environment for the dataset directory for the notebook to specify where the weapon dataset will download. We'll use "os" for this.

```
os.environ["DATASET_DIRECTORY"] = "/content/datasets"
```

Once this cell is executed, we can paste the download code in a new cell and run it.

```
!pip install roboflow

from roboflow import Roboflow
rf = Roboflow(api_key="HA6CUY3Vdt1sRtv6vmzQ")
project = rf.workspace("nizar-assad").project("weapons-hq53z")
dataset = project.version(3).download("yolov5")
```

Once we have our datasets and yolov5 directories ready to use, we can start the training process.



We'll use "train.py" which is already available within the yolov5 directory, this "py" script contains the train function which will take care of all the training process. The picture below shows an overview of the train.py main function.

```
def train(hyp, opt, device, callbacks):
    save_dir, epochs, batch_size, weights, single_cls, evolve, data, cfg, resume, noval, nosave, workers, freeze = \
        Path(opt.save_dir), opt.epochs, opt.batch_size, opt.weights, opt.single_cls, opt.evolve, opt.data, opt.cfg, \
        opt.resume, opt.noval, opt.nosave, opt.workers, opt.freeze
    callbacks.run('on_pretrain_routine_start')


    w = save_dir / 'weights'
    (w.parent if evolve else w).mkdir(parents=True, exist_ok=True)
    last, best = w / 'last.pt', w / 'best.pt'


    if isinstance(hyp, str):
        with open(hyp, errors='ignore') as f:
            hyp = yaml.safe_load(f)
    LOGGER.info(colorstr('hyperparameters: ') + ', '.join(f'{k}={v}' for k, v in hyp.items()))


    if not evolve:
        with open(save_dir / 'hyp.yaml', 'w') as f:
            yaml.safe_dump(hyp, f, sort_keys=False)
        with open(save_dir / 'opt.yaml', 'w') as f:
            yaml.safe_dump(vars(opt), f, sort_keys=False)


    data_dict = None
    if RANK in [-1, 0]:
        loggers = Loggers(save_dir, weights, opt, hyp, LOGGER)
        if loggers.wandb:
            data_dict = loggers.wandb.data_dict
            if resume:
                weights, epochs, hyp, batch_size = opt.weights, opt.epochs, opt.hyp, opt.batch_size
```

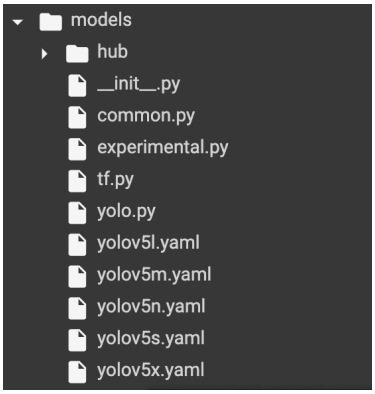
We'll also use the "data.yaml" file which is available in the dataset download file ("datasets/weapons/") which specifies the location of the yolov5 images folder, the yolov5 labels folder, and information on our custom classes.

We'll use the yolov5s model, which is a pretrained yolov5 architecture. It's the smallest and the fastest one in the yolov5 family, as we can see from the table below.

Model	size (pixels)	mAP^{val} 0.5:0.95	mAP^{val} 0.5	Speed CPU b1 (ms)	Speed V100 b1 (ms)	Speed V100 b32 (ms)	params (M)	FLOPs @640 (B)
YOLOv5n	640	28.0	45.7	**45**	**6.3**	**0.6**	**1.9**	**4.5**
YOLOv5s	640	37.4	56.8	98	6.4	0.9	7.2	16.5
YOLOv5m	640	45.4	64.1	224	8.2	1.7	21.2	49.0
YOLOv5l	640	49.0	67.3	430	10.1	2.7	46.5	109.1
YOLOv5x	640	50.7	68.9	766	12.1	4.8	86.7	205.7

***Table 1**: Yolov5 model versions details*

We can check the yolov5s structure in the "models" directory:



Here's what the yolov5s architecture looks like:

yolov5s.yml

```yaml
# YOLOv5 🚀 by Ultralytics, GPL-3.0 license

# Parameters
nc: 80  # number of classes
depth_multiple: 0.33  # model depth multiple
width_multiple: 0.50  # layer channel multiple
anchors:
  - [10,13, 16,30, 33,23]  # P3/8
  - [30,61, 62,45, 59,119]  # P4/16
  - [116,90, 156,198, 373,326]  # P5/32

# YOLOv5 v6.0 backbone
backbone:
  # [from, number, module, args]
  [[-1, 1, Conv, [64, 6, 2, 2]],  # 0-P1/2
   [-1, 1, Conv, [128, 3, 2]],  # 1-P2/4
   [-1, 3, C3, [128]],
   [-1, 1, Conv, [256, 3, 2]],  # 3-P3/8
   [-1, 6, C3, [256]],
   [-1, 1, Conv, [512, 3, 2]],  # 5-P4/16
   [-1, 9, C3, [512]],
   [-1, 1, Conv, [1024, 3, 2]],  # 7-P5/32
   [-1, 3, C3, [1024]],
   [-1, 1, SPPF, [1024, 5]],  # 9
  ]

# YOLOv5 v6.0 head
head:
  [[-1, 1, Conv, [512, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 6], 1, Concat, [1]],  # cat backbone P4
   [-1, 3, C3, [512, False]],  # 13

   [-1, 1, Conv, [256, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 4], 1, Concat, [1]],  # cat backbone P3
   [-1, 3, C3, [256, False]],  # 17 (P3/8-small)

   [-1, 1, Conv, [256, 3, 2]],
   [[-1, 14], 1, Concat, [1]],  # cat head P4
   [-1, 3, C3, [512, False]],  # 20 (P4/16-medium)

   [-1, 1, Conv, [512, 3, 2]],
   [[-1, 10], 1, Concat, [1]],  # cat head P5
   [-1, 3, C3, [1024, False]],  # 23 (P5/32-large)

   [[17, 20, 23], 1, Detect, [nc, anchors]],  # Detect(P3, P4, P5)
  ]
```

To kick off the training we running the training command with multiple options:

- **Img:** represents image size (416x416)
- **Batch:** batch size

Here's an example of one of our training batches:

Here we can see red bounding boxes which represents knives and pink boxes which represents pistols.

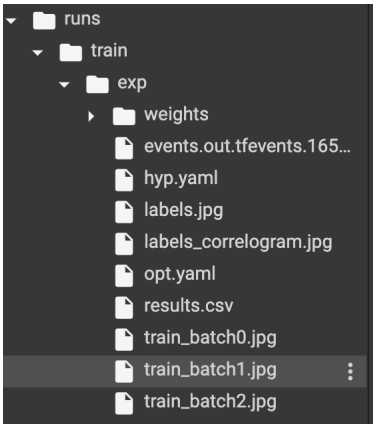
- **Epochs:** number of training epochs (150 is the default number)
- **Data:** path to the “data.yaml” file which specifies the location for our training and validation datasets, the number of classes we want to detect, and the names corresponding to the classes.
- **Weights:** path to pre-trained weight (yolov5s)
- **Cache:** cache images to speed the training

Before running the cell, we want to make sure that in the Notebook settings, our hardware accelerator has GPU selected otherwise the training time would take a much longer time to process.

```
!python train.py --img 416 --batch 16 --epochs 150 --data {dataset.location}/data.yaml --weights yolov5s.pt     --cache

Downloading https://ultralytics.com/assets/Arial.ttf to /root/.config/Ultralytics/Arial.ttf...
train: weights=yolov5s.pt, cfg=, data=/content/datasets/weapons-3/data.yaml, hyp=data/hyps/hyp.scratch-low.yaml, epochs=150, batch_size=16, imgsz=416,
github: up to date with https://github.com/ultralytics/yolov5 ✅
YOLOv5 🚀 v6.1-132-g014acde torch 1.10.0+cu111 CUDA:0 (Tesla T4, 15110MiB)

hyperparameters: lr0=0.01, lrf=0.01, momentum=0.937, weight_decay=0.0005, warmup_epochs=3.0, warmup_momentum=0.8, warmup_bias_lr=0.1, box=0.05, cls=0.
Weights & Biases: run 'pip install wandb' to automatically track and visualize YOLOv5 🚀 runs (RECOMMENDED)
TensorBoard: Start with 'tensorboard --logdir runs/train', view at http://localhost:6006/
Downloading https://github.com/ultralytics/yolov5/releases/download/v6.1/yolov5s.pt to yolov5s.pt...
100% 14.1M/14.1M [00:00<00:00, 130MB/s]
```

After the training has been launched we can notice the presence of a new file within the yolov5 directory called runs which contains every detail of the training process, like results recorded from each epoch and also saves the weights checkpoints and the best weight recorded by the system.

Here's a look at the "runs" file structure:



After several attempts to train our model, we've faced many issues as the runtime disconnects and the training stops, we then use the last checkpoint "last.pt" which represents the last saved weight by the system and we resume training process, we'll notice that the "train" file will contain a brand new "exp" which start the training from the last checkpoints and this file will be our new file of interest.

Tuning the model: The best way we could tune our model is by retraining the previous weight "best.pt" which helped with the precision and decreased the false positive predictions.

To keep track of how the model is reacting with the dataset, we use the "result.csv" file which is a table that shows all the information in real time of how the model is doing with each epoch during the training process. If the mean average precision is increasing it means our model is getting more familiar with the objects we wish to detect and that is a good sign.

results.csv ×

1 to 47 of 47 entries Filter

epoch	train/box_loss	train/obj_loss	train/cls_loss	metrics/precision	metrics/recall	metrics/mAP_0.5	metrics/mAP_0.5:0.95	val/box_loss	val/obj_loss	val/cls_loss	x/lr0	x/lr1
0	0.097048	0.020992	0.024701	0.67222	0.17521	0.13288	0.043852	0.070111	0.012694	0.0081364	0.0032813	0.0032813
1	0.069595	0.022464	0.0095044	0.28474	0.32465	0.21823	0.069647	0.067195	0.0090393	0.007402	0.0065709	0.0065709
2	0.065667	0.020789	0.0052398	0.41551	0.3369	0.29572	0.087574	0.066412	0.0081714	0.0050868	0.0098166	0.0098166
3	0.060039	0.019818	0.0040755	0.37147	0.36634	0.25141	0.085008	0.061352	0.0086437	0.019344	0.009802	0.009802
4	0.055037	0.019429	0.0049285	0.46243	0.39674	0.29747	0.11683	0.062232	0.0085128	0.0033361	0.009802	0.009802
5	0.052355	0.020469	0.0043318	0.37543	0.28363	0.22428	0.097961	0.063019	0.0096272	0.011135	0.009736	0.009736
6	0.050041	0.020049	0.0036342	0.64531	0.53674	0.53317	0.19063	0.053015	0.0080513	0.0025023	0.00967	0.00967
7	0.048189	0.019643	0.0035433	0.43176	0.57657	0.47021	0.21558	0.049406	0.0085559	0.0070296	0.009604	0.009604
8	0.047432	0.019105	0.0028245	0.65305	0.46836	0.50936	0.2239	0.055042	0.0081545	0.0041895	0.009538	0.009538
9	0.046829	0.020224	0.0028621	0.59992	0.61029	0.58965	0.25823	0.050668	0.0080487	0.0026035	0.009472	0.009472
10	0.044544	0.019065	0.0035807	0.61515	0.58447	0.61536	0.27552	0.049251	0.0077291	0.0022599	0.009406	0.009406
11	0.04435	0.018766	0.0026728	0.65957	0.61159	0.64959	0.28627	0.050957	0.0074868	0.0027866	0.00934	0.00934
12	0.044184	0.01892	0.002517	0.70836	0.67276	0.70297	0.32517	0.045437	0.0075582	0.00093632	0.009274	0.009274
13	0.042236	0.018372	0.0020089	0.7422	0.62859	0.67432	0.30758	0.048071	0.0075251	0.0015809	0.009208	0.009208
14	0.042159	0.018282	0.0019481	0.69524	0.65374	0.69169	0.33051	0.046355	0.0075344	0.0031717	0.009142	0.009142
15	0.043205	0.017426	0.0019953	0.71472	0.54877	0.62305	0.29658	0.043614	0.0074672	0.0046648	0.009076	0.009076
16	0.041815	0.017558	0.0019677	0.619	0.47957	0.48816	0.23687	0.046563	0.0078111	0.006683	0.00901	0.00901
17	0.040442	0.018198	0.0018586	0.69204	0.6523	0.68221	0.32546	0.043813	0.0074041	0.0036269	0.008944	0.008944
18	0.041006	0.01727	0.0026574	0.7503	0.56836	0.62795	0.31127	0.046857	0.0073429	0.0015159	0.008878	0.008878
19	0.04016	0.017819	0.0024011	0.66897	0.57617	0.60271	0.28914	0.047455	0.0077607	0.010355	0.008812	0.008812
20	0.040848	0.017419	0.0023268	0.70834	0.67039	0.703	0.35434	0.045796	0.0073403	0.0017988	0.008746	0.008746
21	0.039397	0.017062	0.0015166	0.76816	0.63621	0.70243	0.35772	0.042457	0.0072292	0.0037579	0.00868	0.00868
22	0.039247	0.016181	0.0017277	0.7051	0.69062	0.70878	0.35937	0.043627	0.0070711	0.0018294	0.008614	0.008614
23	0.037887	0.017154	0.0013385	0.77919	0.63333	0.69865	0.37897	0.041346	0.0071048	0.0019023	0.008548	0.008548
24	0.037399	0.016593	0.0017053	0.75864	0.71653	0.75834	0.41306	0.040384	0.0070017	0.0011346	0.008482	0.008482
25	0.037248	0.016058	0.001601	0.77173	0.69883	0.73978	0.36292	0.042058	0.0070463	0.00087137	0.008416	0.008416
26	0.037129	0.015839	0.0014209	0.66277	0.60804	0.64899	0.3437	0.042073	0.007211	0.007028	0.00835	0.00835

Table 2: Table of real time training results metrics per epoch during training

After the training is done, we need to download the “best.pt” weight which represents the best weight recorded during training and it is more interesting than “last.pt” which only shows the weight from last epoch of training.

To avoid losing all data in case runtime is disconnected after training is done, we decided to automatically export the weight using the cell shown below:

```
from google.colab import files
files.download('./runs/train/exp/weights/best.pt')
```

This cell will take care of saving the weight in our local machine for future usage.

3.3.3 Improving Yolov5 model performance

After the model is downloaded to our local machine, we tried optimizing it using onnx which is significantly faster than Pytorch. For this we converted the model weight “best.pt” generated from our training to the onnx format as the Deci website does not support Pytorch models yet.

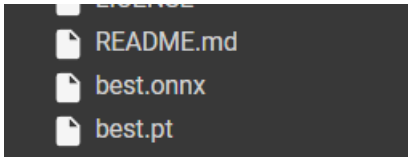
We created a new notebook, clones the repository and install the required packages just like while training the model.

```
!git clone https://github.com/ultralytics/yolov5
%cd yolov5
!pip install -r requirements.txt
!pip install -U nvidia-tensorrt --index-url https://pypi.ngc.nvidia.com
```

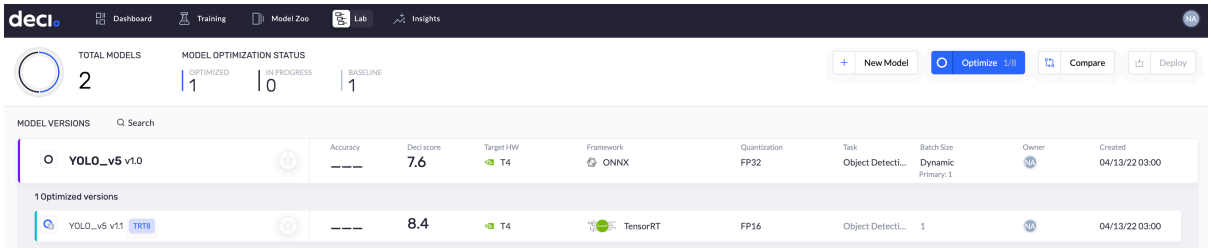
Then we used the “export.py” which is available within the yolov5 directory and used the “--weight” option to get the weight and “—include torchscript onnx” which permits us to do the conversion

```
!python /content/yolov5/export.py --weights best.pt --include torchscript onnx

Downloading https://ultralytics.com/assets/Arial.ttf to /root/.config/Ultralytics/Arial.ttf...
export: data=data/coco128.yaml, weights=['best.pt'], imgsz=[640, 640], batch_size=1, device=cpu, half=False, inp
YOLOv5 🚀 v6.1-132-g014acde torch 1.10.0+cu111 CPU

Fusing layers...
Model summary: 213 layers, 7026307 parameters, 0 gradients, 15.8 GFLOPs

PyTorch: starting from best.pt with output shape (1, 25200, 11) (13.7 MB)

TorchScript: starting export with torch 1.10.0+cu111...
```

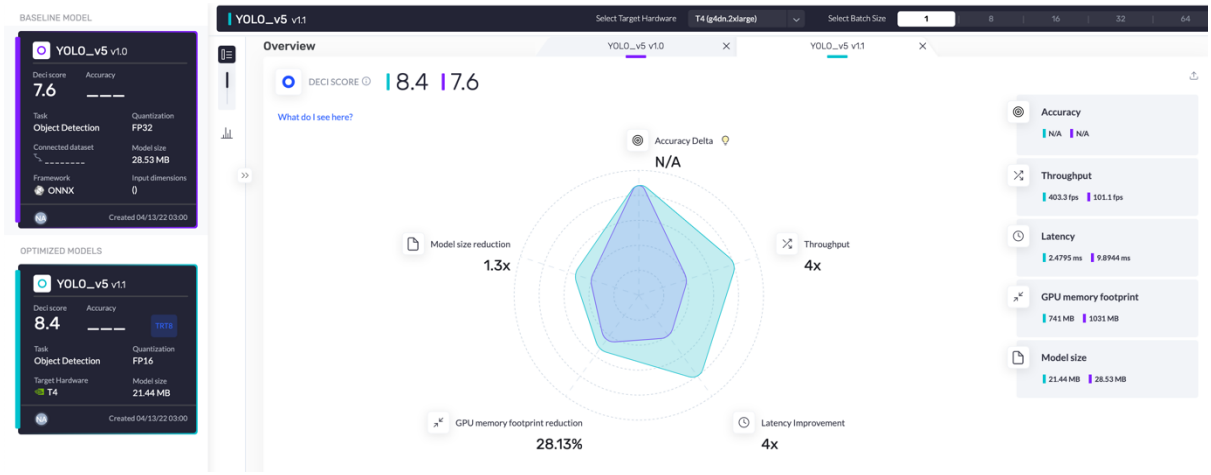
In the notebook content we can notice that the “best.onnx” was successfully generated.



Then we exported the “best.onnx” model in our local machine and imported it in a website called Deci which optimizes automatically an ONNX model and runs benchmark on it.



All we had to do is to import the model and click on the optimize button on the top right. Select the device property (We got it from the “get_device_preperties” command we’ve previously ran) and then follow up with the process of optimization, it takes few minutes to run the benchmark and process the model update. Giving us the v1.1 which is incredibly faster and more accurate as we can see in the comparison below:



The model we trained has scored 7.6 and the optimized one scored 8.4

Unfortunately, we’ve faced many issues while trying to run inference and deploy the resulted “YOLOv5.pkl” model, the issue comes from the company as the technology is new, still not very popular and full of bugs. We kept the model in the side hoping we can make use of it’s incredible performances someday in the future.

4 Evaluation

4.1 Evaluation Methodology

4.1.1 Evaluation Metrics

Once the training process is done, the system automatically runs the validation and outputs the results for the “best.pt” weight.

```
150 epochs completed in 0.993 hours.
Optimizer stripped from runs/train/exp2/weights/last.pt, 14.3MB
Optimizer stripped from runs/train/exp2/weights/best.pt, 14.3MB

Validating runs/train/exp2/weights/best.pt...
```

We then can see the two output measurements that are generated from validation, they are essential to build a perfect object detection model which gives more precise and accurate results. We used precision, recall, “mAP”. These metrics are mathematically defined as follows:

$$\text{Precision} = \text{TP} /(\text{TP} + \text{FP})$$
$$\text{Recall} = \text{TP}/(\text{TP} + \text{FN})$$

- TP: True positive predictions
- TN: True negative predictions
- FP: False positive predictions
- FN: False negative predictions

The mean average precision’s mathematical function combines the precision and the recall.

$$AP = \frac{1}{11} \sum_{Recall_i} Precision(Recall_i) = 1$$

The “P” stands for precision, which measures how accurate is our predictions. “R” stands for recall, which measures how good we model finds all the positive predictions and “mAP” for mean average precision. After more than 10 attempts to train the yolov5 model on the weapon dataset, the best results achieved are as follow:

```
     P         R      mAP@.5
0.922     0.891       0.929
```

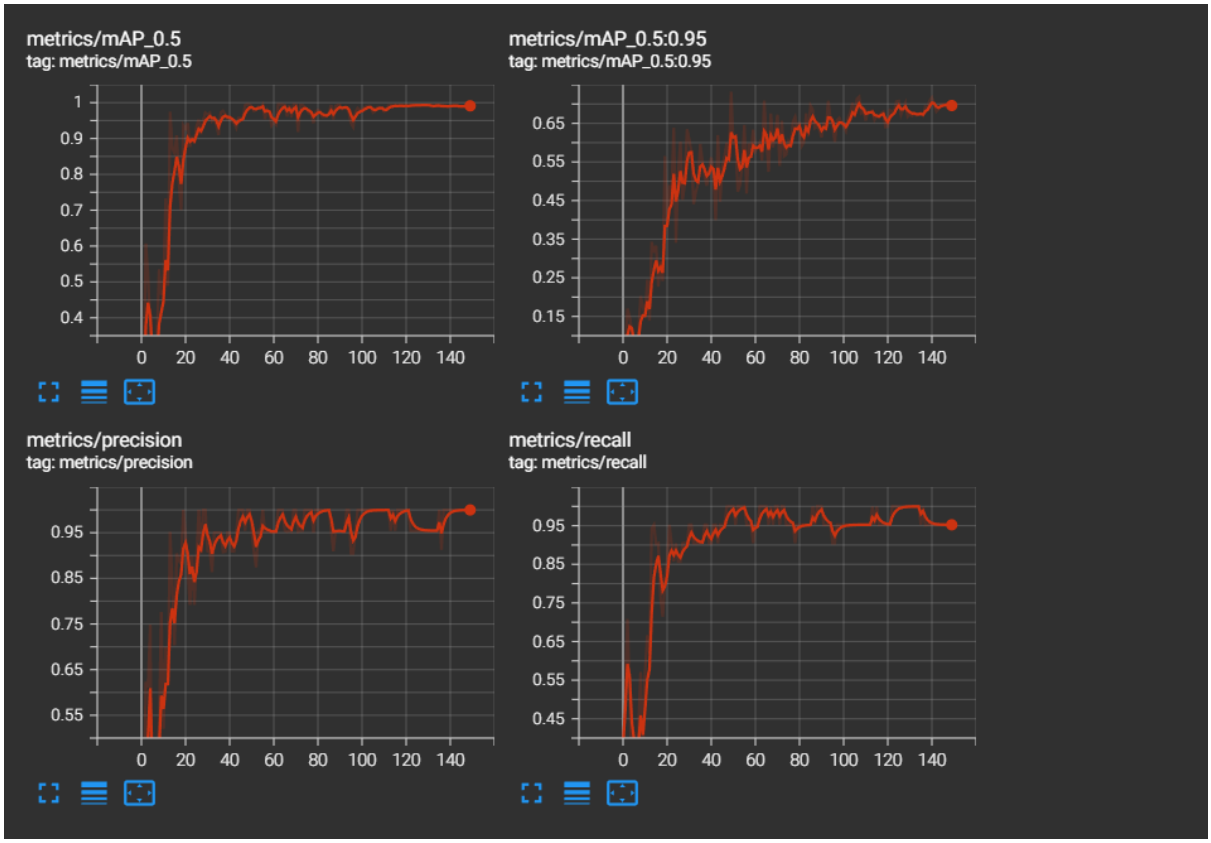
From this result, we can deduce that our model is doing a very good job on the custom dataset we've provided.

We then used Tenserboard by Tenserflow which is a measurements platform that outputs all the needed information about our trained model, it enables tracking experiment matrices like precision, recall, mAP and loss and much more functionalities that we can analyze.

This cell runs the evaluation and outputs the framework:

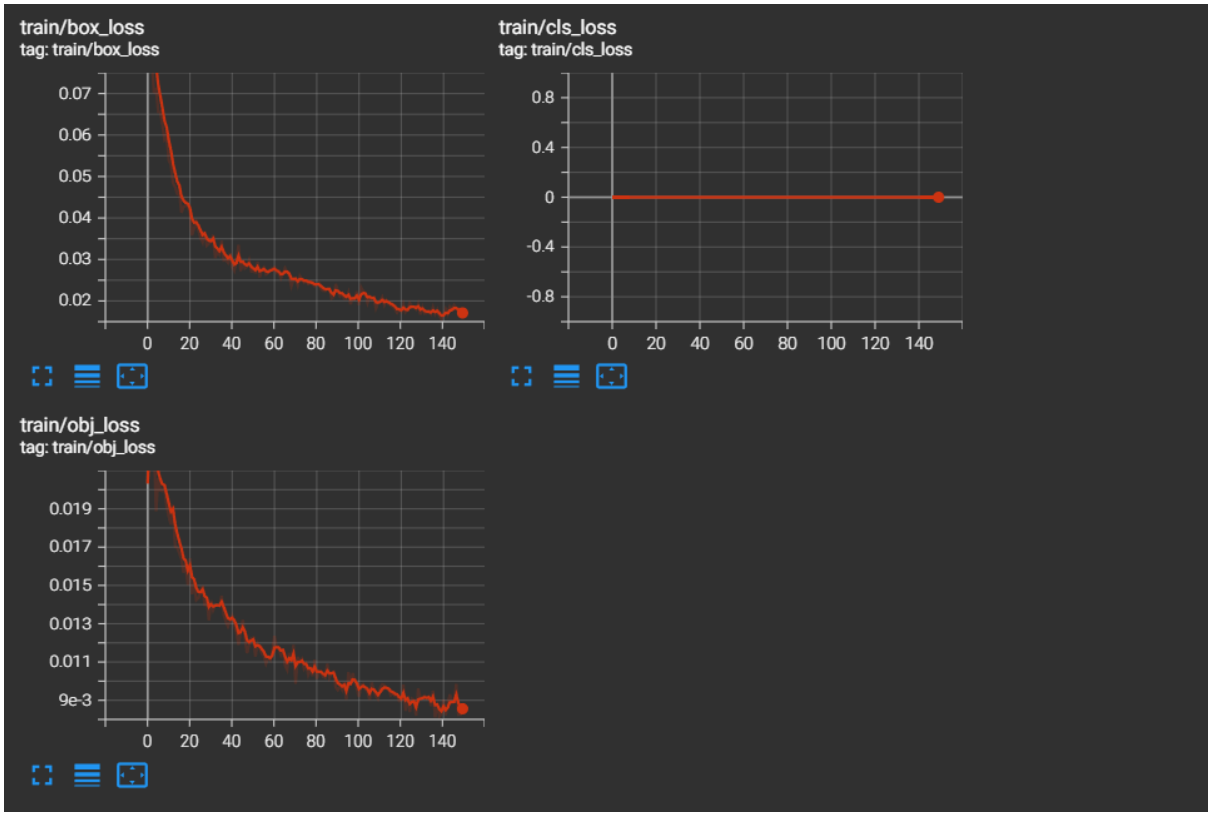
```
%load_ext tensorboard
%tensorboard --logdir runs
```

After few seconds of execution, here's the output:



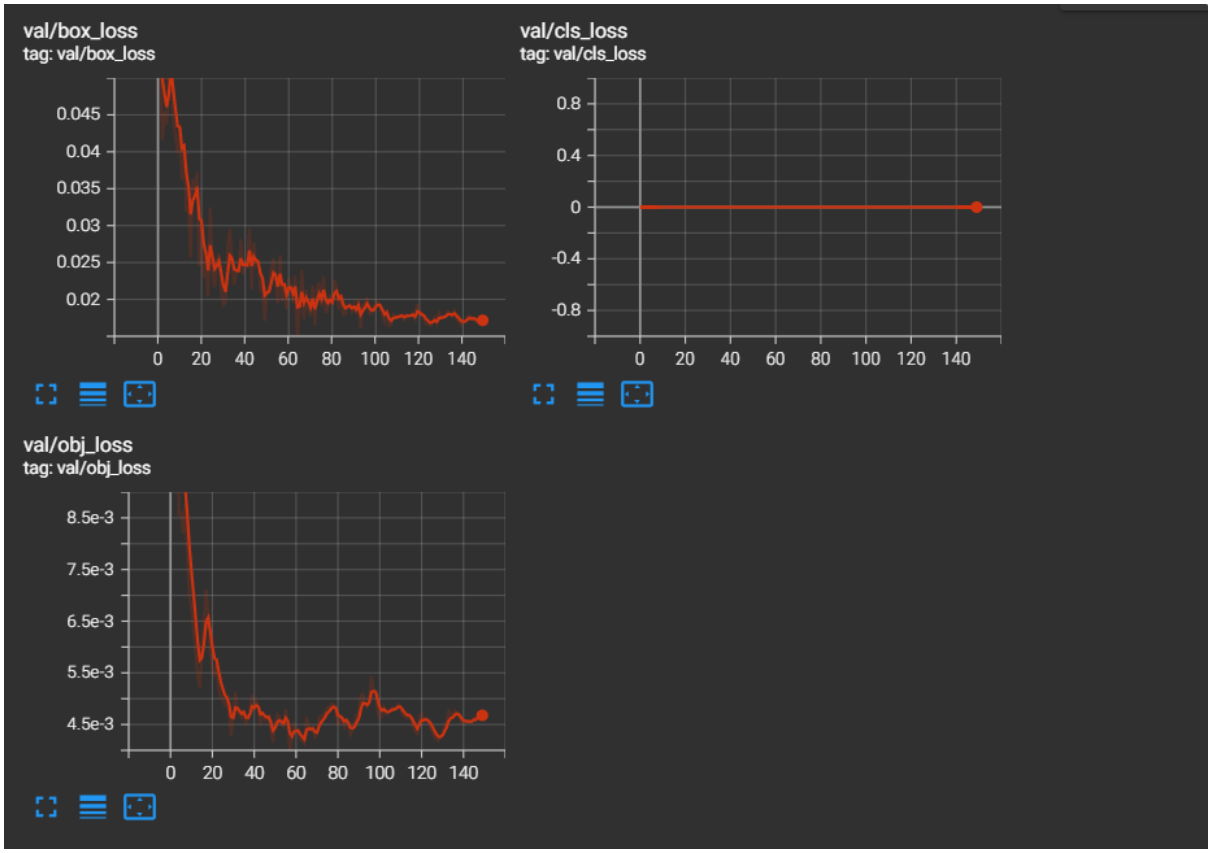
The scalar we want to focus on the most is the top left one which shows the train model's mean average precision on a scale of 1 for all the 140 epochs of training. As we can notice, it recorded a "mAP" of 0.92 which means that our model is accurate. Also, for the recall and precision graphs that shows very good results.

The second set of scalars shows the loss for training set:



- The **"box_loss"** diagram measures how precise the predicted boxes are to the ground truth object, the regression through epochs is a good sign which means our boxes were more accurate over time.
- The **"cls_loss"** measures the correctness of the classification of each predicted bounding box.
- The **"obj_loss"** measures the classification loss and the localization loss.

The third set of scalars is very important as it shows the loss in the validation set which gives a more realistic look at our model skills:



The **“box_loss”** and **“obj_loss”** both show a regression which is a very good sign of our model performance.

Tenseboard also gives us a large view at the model architecture which is very helpful as we want to understand how the model operates:

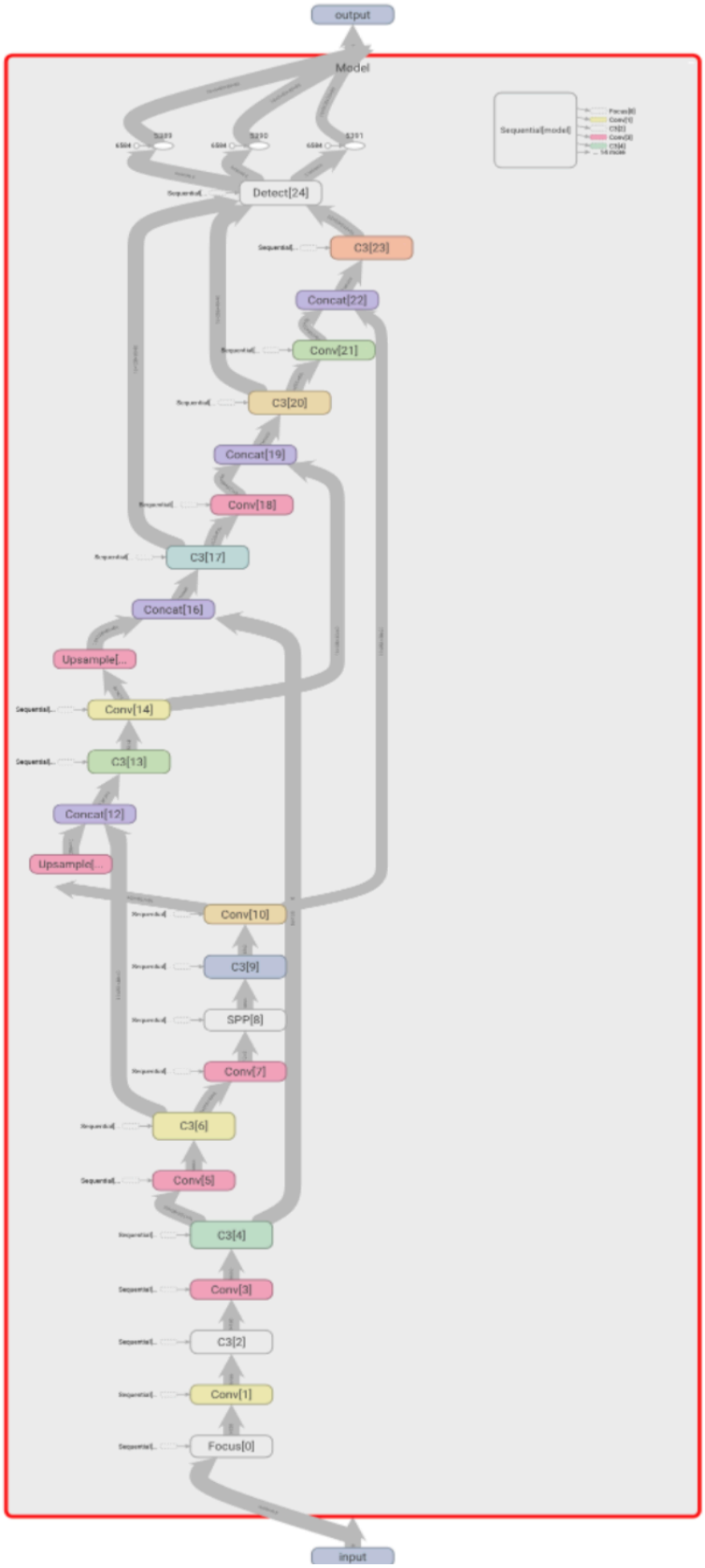


Figure 5: yolov5 architecture (Tenserboard)

4.1.2 Testing datasets:

We believe that evaluating a model using graphs and mathematical functions is very important. But we should not ignore what's more important, testing the model with real data would certainly give us a better hint on the model's performances, which is crucial in our project, human lives might depend on it so we can't afford missing any part of the testing and evaluation process and even the smaller details of detection must count.

To test our model, we used various techniques and multiple data types to get the best view of how the model will react in every single situation.

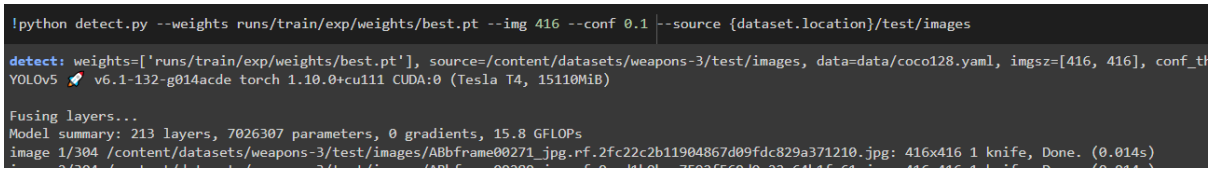
The first dataset we used was to evaluate the model was the testing dataset which was included within our "datasets" file from Roboflow, we've assigned 10% of our dataset to test our model.

We used "detect.py" which is a python script provided by the yolov5 directory that enable us to run inference using our trained weight "best.pt"

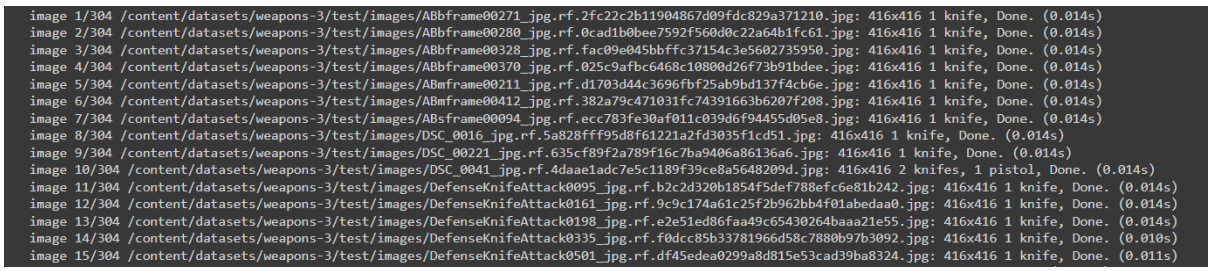
This script contains the run function that takes care of running the inference for us:

```
@torch.no_grad()
def run(
        weights=ROOT / 'yolov5s.pt',  # model.pt path(s)
        source=ROOT / 'data/images',  # file/dir/URL/glob, 0 for webcam
        data=ROOT / 'data/coco128.yaml',  # dataset.yaml path
        imgsz=(640, 640),  # inference size (height, width)
        conf_thres=0.25,  # confidence threshold
        iou_thres=0.45,  # NMS IOU threshold
        max_det=1000,  # maximum detections per image
        device='',  # cuda device, i.e. 0 or 0,1,2,3 or cpu
        view_img=False,  # show results
        save_txt=False,  # save results to *.txt
        save_conf=False,  # save confidences in --save-txt labels
        save_crop=False,  # save cropped prediction boxes
        nosave=False,  # do not save images/videos
        classes=None,  # filter by class: --class 0, or --class 0 2 3
        agnostic_nms=False,  # class-agnostic NMS
        augment=False,  # augmented inference
        visualize=False,  # visualize features
        update=False,  # update all models
        project=ROOT / 'runs/detect',  # save results to project/name
        name='exp',  # save results to project/name
        exist_ok=False,  # existing project/name ok, do not increment
        line_thickness=3,  # bounding box thickness (pixels)
        hide_labels=False,  # hide labels
        hide_conf=False,  # hide confidences
        half=False,  # use FP16 half-precision inference
        dnn=False,  # use OpenCV DNN for ONNX inference
):
```

We used several options to run inference, starting with the path to our best.pt weight which is under weights file, "--img" option for the size,"--conf" for a faster inference time, and finally the "--source" which is the path to our testing images.



After running this cell, we can have a brief idea of how the detection time and the detected objects per image.

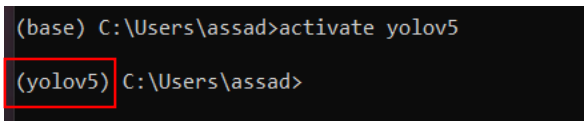


The inference speed is around (0.014s) which is a very good score compared to other models we've used before like TF2 for example that took (0.12s). Then we can go and check any of the testing images to see how the detection went.

Now that we've tested the model inside the notebook, we can move on and try inference on our local machine, we've faced some problems running the model within our macos device, so we decided it was time to switch to a windows device and work on it there as it is more optimized for this kind of tasks.

We first need to open the anaconda prompt where python is installed, get the yolov5 repository by Ultralytics from Github and clone it in our local machine using "git clone" command. We decided to put the yolov5 folder within our user directory.

Next, we need to activate the yolo environment using the activate command.



After the environment is activated we're going to use cd in order to change to the yolov5 directory. The first thing we should do is install all the required packages for our model to run correctly, which are present in the requirements.txt file provided by the yolov5 repository.

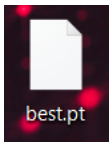
Let's first check if the txt file is present for this we use the "dir" command which lists all the directory.

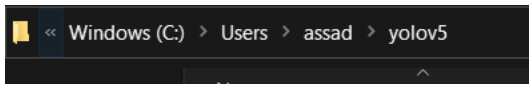
```
05/04/2022  19:34            16,170 README.md
05/04/2022  19:34               963 requirements.txt
```

Now let's install all the packages, for this we use the "pip install" command followed by the "-r" variable. We can also use "-u" which is variable that will update all the packages to avoid any type of dependency error.

```
(yolov5) C:\Users\assad\yolov5>pip install -r requirements.txt
WARNING: pip is configured with locations that require TLS/SSL, however th
Requirement already satisfied: matplotlib>=3.2.2 in c:\users\assad\anacond
```

All the packages are installed, now we should grab our best.pt weight previously downloaded to our local machine and add it to our yolov5 directory.





Now we're good to go, let's start running inference locally using our trained model. We're using an NVIDIA Geforce RTX 2070 GPU which helps with the speed.To run inference, we use the "detect.py" provided by Ultralytics. We then specify the weight and source. After the fusing layers process, the output will be present in the path specified below.

```
(base) C:\Users\assad\yolov5>python detect.py --weights best.pt --source gun.jpg
detect: weights=['best.pt'], source=gun.jpg, data=data\coco128.yaml, imgsz=[640, 640], conf_thres=0.25, iou
max_det=1000, device=, view_img=False, save_txt=False, save_conf=False, save_crop=False, nosave=False, clas
ostic_nms=False, augment=False, visualize=False, update=False, project=runs\detect, name=exp, exist_ok=Fals
ness=3, hide_labels=False, hide_conf=False, half=False, dnn=False
fatal: not a git repository (or any of the parent directories): .git
YOLOv5  2022-4-5 torch 1.9.0+cu111 CUDA:0 (NVIDIA GeForce RTX 2070 with Max-Q Design, 8192MiB)

Fusing layers...
Model summary: 213 layers, 7026307 parameters, 0 gradients, 15.8 GFLOPs
image 1/1 C:\Users\assad\yolov5\gun.jpg: 448x640 1 knife, Done. (0.017s)
Speed: 1.0ms pre-process, 16.9ms inference, 3.0ms NMS per image at shape (1, 3, 640, 640)
Results saved to runs\detect\exp21
```

We first started with a picture of three guns to see if the model is accurate enough to detect the three objects at once. Even though some of the guns were not completely included in the image, the model has still succeeded to detect the 3 pistols.

Input **Output**

Next, we tried to input a knife image to make sure both classes can be detected. This experiment was also a success.

Input **Output**

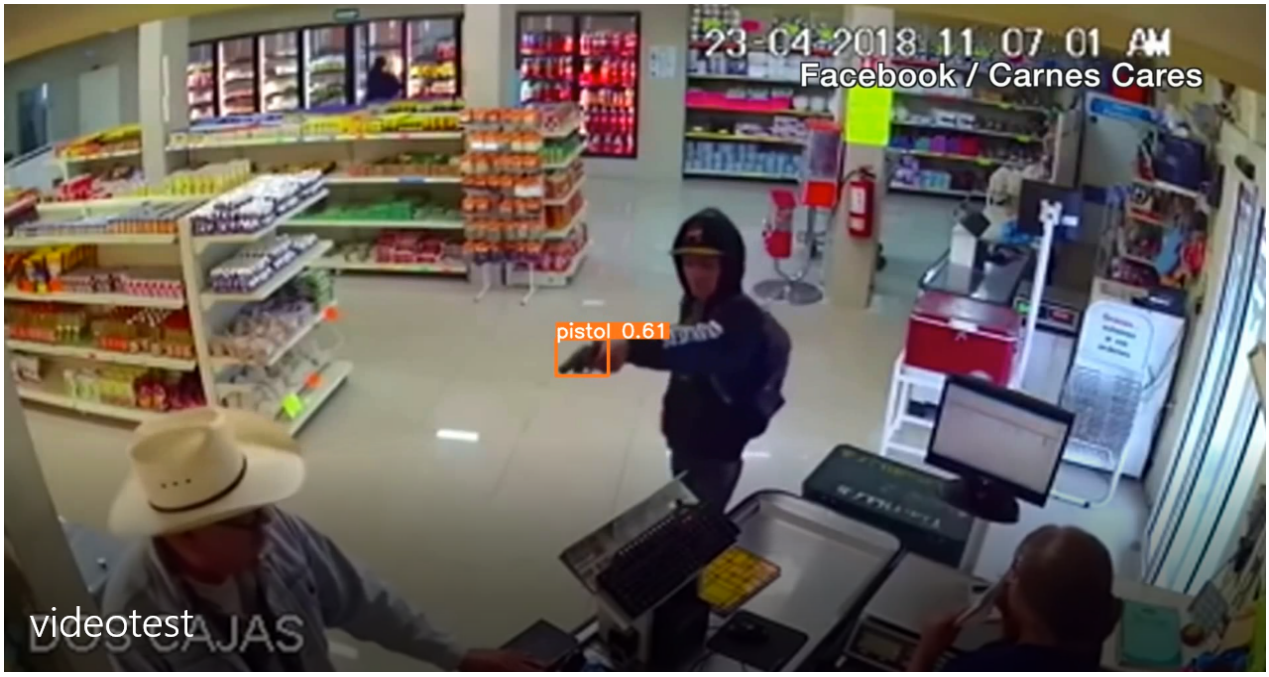
Our model support mp4 as well, so we just need to insert a video in the yolov5 directory and then run it using “detect.py”.We tried running inference on a simple mp4 video, the result were positive. Even if the gun was hidden by the actors hands, the model still managed to detect the pistol efficiently.

Even though our model has proven to be very efficient with images, our main goal is to detect on a CCTV camera, so we need to try it on a video format, we found a video of a random person trying to steal using a gun, we downloaded it and tried it using the exact same process as with the images.

```
(base) C:\Users\assad\yolov5>python detect.py --weights best.pt --source videotest.mp4
detect: weights=['best.pt'], source=videotest.mp4, data=data\coco128.yaml, imgsz=[640, 640], conf_thres=0.25, iou_thres=
0.45, max_det=1000, device=, view_img=False, save_txt=False, save_conf=False, save_crop=False, nosave=False, classes=Non
e, agnostic_nms=False, augment=False, visualize=False, update=False, project=runs\detect, name=exp, exist_ok=False, line
_thickness=3, hide_labels=False, hide_conf=False, half=False, dnn=False
fatal: not a git repository (or any of the parent directories): .git
YOLOv5  2022-4-5 torch 1.9.0+cu111 CUDA:0 (NVIDIA GeForce RTX 2070 with Max-Q Design, 8192MiB)

Fusing layers...
Model summary: 213 layers, 7026307 parameters, 0 gradients, 15.8 GFLOPs
video 1/1 (1/1234) C:\Users\assad\yolov5\videotest.mp4: 384x640 Done. (0.021s)
video 1/1 (2/1234) C:\Users\assad\yolov5\videotest.mp4: 384x640 Done. (0.013s)
video 1/1 (3/1234) C:\Users\assad\yolov5\videotest.mp4: 384x640 Done. (0.011s)
video 1/1 (4/1234) C:\Users\assad\yolov5\videotest.mp4: 384x640 Done. (0.011s)
video 1/1 (5/1234) C:\Users\assad\yolov5\videotest.mp4: 384x640 Done. (0.011s)
video 1/1 (6/1234) C:\Users\assad\yolov5\videotest.mp4: 384x640 Done. (0.013s)
```

Notice the inference speed on every frame of the video, it goes around (0.01s per frame), very satisfied result.

The handgun was successfully detected by the model even though the angle was not in our favour and the quality was very bad (144p).

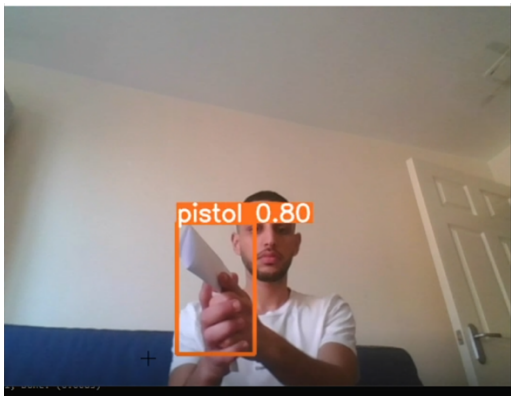


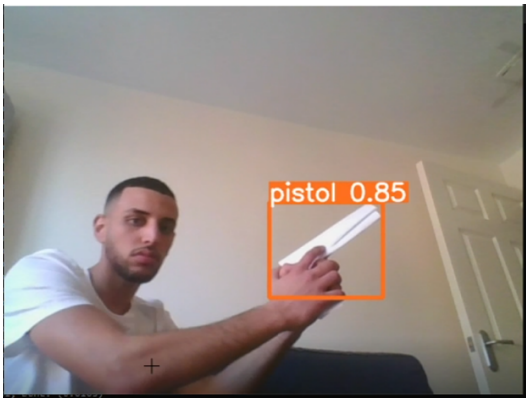
We've tested our model using various methods, now we need to put it in a more realistic situation. Let's try running our model on our device ad serve the input directly from the webcam as it is closer to a CCTV camera detection. I tried to play the actor by holding objects from different angle and see how the model would react. Then have a more realistic example of what we're trying to achieve in our project.

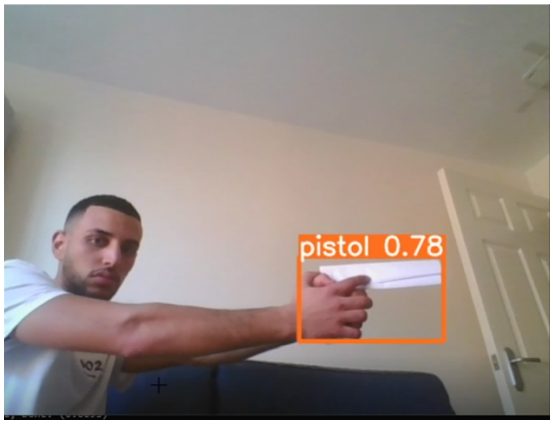
To connect the webcam to our model we use "0" as an input to the source option, the system automatically launches the webcam and start running inference.

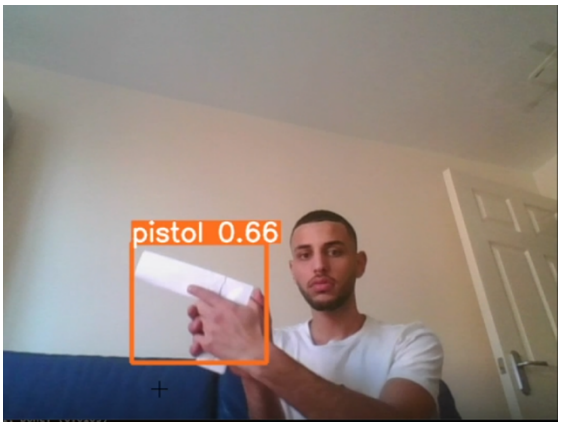
```
(base) C:\Users\assad\yolov5>python detect.py --weights best.pt --source 0
```

I couldn't find a gun so I tried creating a fake one and tried to hold it from different angles to see the results. The results were positive from all different angles.

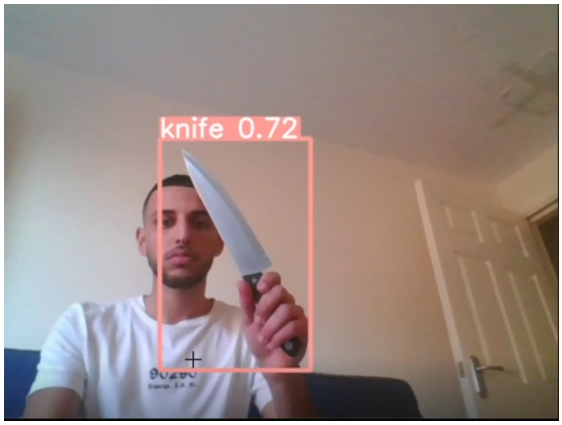


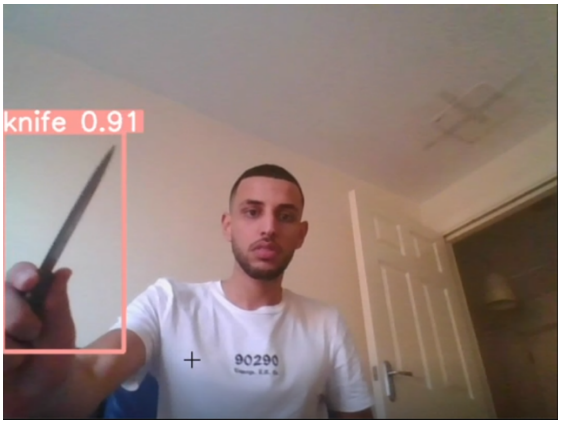






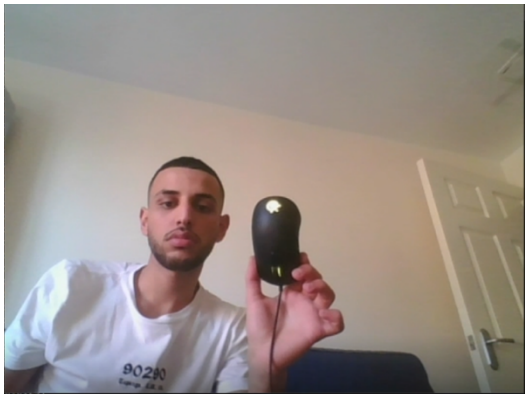
The pistol was detected with no issue, we tried using knives to experiment both objects we wish to detect:

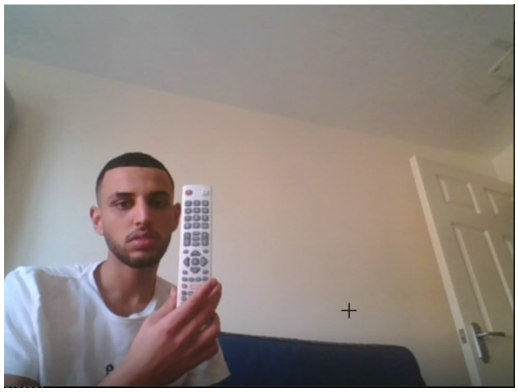




We can see that the knife detection was also a success, even if we tried different angles, knowing that a knife is very hard to detect from the front due to its sharp form, the model still managed to get it right, and different sizes were also detected correctly.

After evaluating the model using different type of input, I tried to use different object to see if there's false positive predictions or any kind of inconsistency.





The objects used that are far from being weapons were not detected by the model, not even once. Which means the inference processed on every frame had been a success, and the false positive detections does not frequently occur.

4.2 Results

Several metrics were computed to evaluate each model after training with the same testing set provided from the merged dataset we have created in the beginning of our project research, we have used the Pretrained Tf2 model by Tenserflow, and the results were decent, having achieved a precision of (0.85) which was doing a very good job in detecting objects from video test or movie clips. But the inference speed was not very satisfying as it

varied around (0.17 s per frame), once we started testing the model on cctv footage, we noticed the false detections has occurred more frequently and due to our project goal, we can't afford having false alarms every time the model makes a mistake in detecting the wrong object. So, we've decided it was time to upgrade and move on to a better technology which going to be more fitted to our situation. We tried using Yolov5 object detection model which we trained using a custom dataset, it was created using Roboflow and checked using their latest technology "Dataset Health check", the results captured during the evaluation shows that we made a good choice choosing this model. The precision was very high (0.92) and the amount of false positive detections was very small to neglectable, the inference speed varied around (0.9 s per frame).

We have managed to find other researches on the same field using different architectures, like the yolov3 which was trained and test by Salido .J, he used the exact method of gathering images and creating his own dataset, he then trained the model on a COCO dataset to compare the two versions of the yolo model. The model we're using for the comparison is the one that has finally decided to be used by the researcher himself. (Salido, Lomas, Ruiz-Santaquiteria and Deniz, 2021). Then we used the VGG 16 architecture which was processed for detection by a Software Engineer located in North Carolina, He trained the model using ImageNet (Qi, Guangdi, Yusheng and Xin, 2018). These two model examples are going to help us comparing our models that have been tested on a dataset of 400 images.

Models	#TP	#FP	#FN	Precision (%)	Recall (%)	mAP (%)
VGG16	453	40	10	90	97	94
Yolov3	200	11	25	94	88	90
TF2	354	32	14	91	96	95
Yolov5	360	28	22	92	89	91

Table 3. Assessment metrics obtained from different models

4.3 Discussion

To define the presence of handguns and knives, we estimated a YOLO-V5 based classification model using two object classes. Researchers have employed YOLO to detect many things related to their research. However, the purpose of this article is to see how the YOLOv5 neural network performs in recognising the weapon when there is an obstructed backdrop and hard detecting angles. From the research, we chose several videos to serve

as a baseline. For recognising firearms in various films, they employed Faster RCNN with VGG16 architecture. We combined several datasets with the same data type as they used, like the knife dataset which was taken from other researches that have been working on the same purpose. YOLOv5 was used to train the dataset. Our main goal was to achieve more than they did in terms of accuracy and precision.

During the evaluation, the detection results are checked frame by frame in the photos and videos, and a detection is considered true positive if the gun and bounding box overlap by more than 50%. Ground truth is established when the human sight identifies the gun. We utilised our own eyes to check if the object detection model would notice the weapons first.

We can see from the table of model comparison that the best algorithm in terms of performance is the TF2 model, but in the real world, some factors might interrupt how a certain model acts, inference speed and footage resolution were the key factors that influenced our choice. Yolov5 have proven to be very fast and the best fitted for our situation even though It still needs to be trained in order to reduce the false positive detections.

We found that the Yolov5 algorithm performs well on a variety of datasets and can be utilised in a variety of real-time scenarios. We believe that modifying the real information data assortment and further enhancing the information explanation have the greatest potential for improving execution.

Now that we have created the model, we have tried to respect all the requirements, even though we didn’t succeed deploying the model in a real CCTV operator, we’ve managed to reach all the goals that we have set during the literature review and other work analysis. We managed to train a model with a very efficient data which helped with the main issue of low resolution and bad quality videos, and we also managed to make the inference speed as high as possible with the newest technologies that arrived in the last two years.

We plan to continue our work on the algorithm in order to provide a complete ready-to-market solution for CCTV-operators.

5 Conclusions

The efficiency of the algorithm has been tested in various conditions such as a variety of guns and knives, different positions of the armed person, low image quality, etc. We've managed to make our model experience many types of situations that it would face in a real scenario. The performance of the model was evaluated using Tensorboard wish provided us with scalars showing the precision, the recall, the loss on the training and validation sets, and also the mean average precision.

This work showed that it is feasible to use convolutional neural network to create a weapon detector for surveillance cameras. The mused method proved that the yolov5 was the most qualified architecture for this type of projects rather than a faster RCNN model which was proved to be less efficient and slower by the number of objects detected in the same amount of time. The Yolov5 model proved to be more fitted for low quality videos and objects captured from different angles in a frame. Faster RCNN is more optimized to detect objects in images and high-quality content.

6 Recommendations for future work

We want to continue working on the algorithm in order to give CTTV operators with a full and ready-to-market solution. We plan to do further testing in three different scenarios. The "bank" scenario is a well-lit interior location in which the camera is near to the criminal. On the other hand, the "street" scenario must contend with bad lighting and a large distance between the person and the camera, resulting in low resolution of the things to be recognized.

We're also planning on creating a program that would enhance the colors of the cctv by default before feeding it to the system, ad also try to add some effect that will increase the quality so that the model will perform in better conditions.

The next thing we should work on is a real life cctv deployment, the financial situation didn't permit for us to get the required devices in order to make the implementation realistic, so we tried our model using a webcam. But if we get to do the real life deployment, we would get a device that would be linked to a camera and process the detection. The best device for this task is the NVIDIA Jetson Xavier which is a developer kit intended to be designed into any autonomous machines that requires modern AI like our project example. This device has proven to be very efficient from various different researches. It can handle alarm systems as well (Deploy autonomous machines, 2021).

Now that we have generated an efficient model detection, that can detect guns and knives from a CCTV surveillance camera. The next thing that is crucial for our system to efficiently work is an alarm system. Otherwise our model wouldn't be very interesting and It will still require human contribution to watch the bounding boxes when they appear, an alarm system should be linked to the model and should turn on whenever the model detects the object of interest for more than 3 seconds, just to make sure the object detected is a true positive, because we don't want false alarm to occur. This system can be built using React JS. which will send receive a message of a detecting object and will send a notification to the end user.

Finally, we would like to develop a platform that will. Help the end user to set the object detection system to a surveillance camera without having to go through any coding process, not all people know how to use a command line in order to deploy a model and set the right parameters.

7 References

- *Marbach, G., 2022. An image processing technique for fire detection in video images. [online] researchgate. Available at: <https://www.researchgate.net/publication/223683356_An_image_processing_technique_for_fire_detection_in_video_images> [Accessed 14 April 2022].*

- *Darker, I.; Gale, A.; Ward, L.; Blechko, A. Can CCTV Reliably Detect Gun Crime? In Proceedings of the 41stAnnual IEEE International Carnahan Conference on Security Technology, Ottawa, ON, USA, 8–11 October2007; pp. 264–271.*

- *Proyecto, J., 2022. Encuesta Nacional de Victimización a Empresas 2018. [online] Inei.gob.pe. Available at: <https://www.inei.gob.pe/media/MenuRecursivo/publicaciones_digitales/Est/Lib1592/libro.pdf> [Accessed 15 April 2022].*

- Taylor, E. & Gill, M., 2014. CCTV: Reflections on its use, abuse and ... - springer. Available at: https://link.springer.com/chapter/10.1007%2F978-1-349-67284-4_31 [Accessed January 17, 2022].

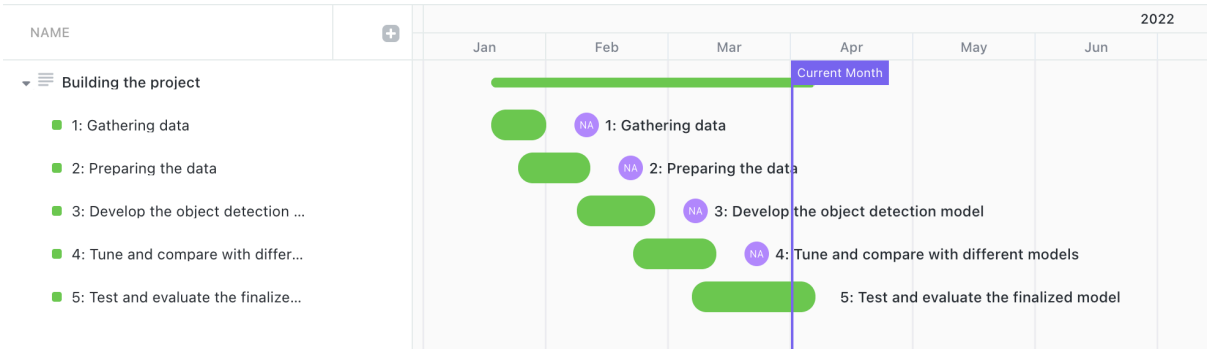
- Albawi, S., Mohammed, T. and Al-Zawi, S., 2022. Understanding of a convolutional neural network. [online] Ieeexplore.ieee.org. Available at: <https://ieeexplore.ieee.org/abstract/document/8308186> [Accessed 16 January 2022].

- Faggella, D., 2019. AI for Crime Prevention and detection - 5 current applications. Emerj. Available at: https://emerj.com/ai-sector-overviews/ai-crime-prevention-5-current-applications/ [Accessed January 18, 2022].

- Qadir , A.M. & Varol , A., 2020. (PDF) the role of machine learning in Digital Forensics. Researchgate. Available at: https://www.researchgate.net/publication/342193343_The_Role_of_Machine_Learning_in_Digital_Forensics [Accessed January 18, 2022].

- Casey, E., 2022. Handbook of Digital Forensics and Investigation. [online] Google Books. Available at: <https://books.google.co.uk/books?hl=fr&lr=&id=xNjsDprqtUYC&oi=fnd&pg=PP1&dq=digital+forensics+investigation&ots=X4vIH19IxO&sig=njJCQa8sU-Jd1trZudMZgxOLox0#v=onepage&q=digital%20forensics%20investigation&f=false> [Accessed 30 January 2022].
- Mitchell, F., 2010. Article: The use of Artificial Intelligence in ... - sas-space. SAS-space. Available at: https://sas-space.sas.ac.uk/5533/1/1922-2707-1-SM.pdf [Accessed January 18, 2022].
- Parthasarathy, D., 2017. A brief history of cnns in image segmentation: From R-CNN to mask R-CNN. Medium. Available at: https://blog.athelas.com/a-brief-history-of-cnns-in-image-segmentation-from-r-cnn-to-mask-r-cnn-34ea83205de4 [Accessed January 19, 2022].
- Ren, S., 2015. Faster R-CNN: Towards real-time object detection ... - neurips. Available at: https://proceedings.neurips.cc/paper/2015/file/14bfa6bb14875e45bba028a21ed38046-Paper.pdf [Accessed January 19, 2022].
- Warsi, A., 2022. *Gun Detection System Using Yolov3*. [online] Ieeexplore.ieee.org. Available at: <https://ieeexplore.ieee.org/document/9057329> [Accessed 16 April 2022].
- Dwyer, B., 2021. *Roboflow Blog*. [online] Roboflow Blog. Available at: <https://blog.roboflow.com/> [Accessed 17 April 2022].
- Solawetz, J., 2020. *How to Train YOLOv5 On a Custom Dataset*. [online] Roboflow Blog. Available at: <https://blog.roboflow.com/how-to-train-yolov5-on-a-custom-dataset/> [Accessed 17 April 2022].
- Fang, J., 2021. *A Deployment Scheme of YOLOv5 with Inference Optimizations Based on the Triton Inference Server*. [online] Ieeexplore.ieee.org. Available at: <https://ieeexplore.ieee.org/abstract/document/9442557> [Accessed 23 April 2022].
- Salido, J., Lomas, V., Ruiz-Santaquiteria, J. and Deniz, O., 2021. Automatic Handgun Detection with Deep Learning in Video Surveillance Images. *Applied Sciences*, 11(13), p.6085.
- Qi, Z., Guangdi, H., Yusheng, L. and Xin, Z., 2018. Binocular vision vehicle detection method based on improved Fast-RCNN. *Journal of Applied Optics*, 39(6), pp.35-40.
- NVIDIA. 2021. *Déployez des machines autonomes évolutives optimisées par l'IA*. [online] Available at: <https://www.nvidia.com/fr-fr/autonomous-machines/embedded-systems/jetson-agx-xavier/> [Accessed 24 April 2022].

8 Appendices



1: Gathering data

For the first part of the process, we have been testing different datasets. We found available public datasets that we could use but for the sake of our project, we decided to move on with our own custom dataset. We decided to use Roboflow for all the data processing.

2: Preparing the data

The main challenge while preparing the data, was to have an efficient set of images for training and validation, that will combine both guns and knives. We decided to move on with a public dataset that was published by a trustworthy user on Roboflow, we then used the merging technology which helped us combining both the images and the labels for training on both objects. Once we had a clean ready to use dataset we decided it was time to move on to the next phase where the models get involved in the process.

3: Develop the object detection model

This was the most crucial phase of our project, which included finding the best fitted model architecture that will respect all the criteria we required. We started using the VGG 16 architecture but quickly found out it wasn't very adapted for low resolution content which was a clear failure as we want to run our model on surveillance cameras. We then tried using Tenserflow models, the results where decent, but after a long research, we discovered yolo models which were the best in terms of real time object detection. We started using the yolov3 model, then discovered the most recent version which was the yolov5 and did a research on it. We discovered that this version of yolo outperformed all the concurrence and that this was definitely the model that we needed for our project. We also discovered that yolov5 creators, made a partnership with roboflow to optimize the

So, we started from developing the model. The high amount of people interested in this field helped us getting tips and ways to train our model in the most optimized way. So we proceeded with training our model. This phase took more than expected as the desired outcome was not reached until a while later, which had put us in a very convenient situation for what we had to do next.

4: Tune and compare with different models

We tried to tune our model using different techniques, the most efficient way was to train our model several times until it reaches the desired results. We also compared our yolov5 model with the previous models we tried to use like VGG16 and Tf2 and with some of the available models form other researches by comparing their precision, recall and loss.

5: Test and evaluate the model

Once we finished training and tuning the model, we ran several evaluation metrices to analyse the performance of our model, we also had to learn what every section of the analysis meant in order to read the evaluation data. It was time to test the finalized model on the adequate dataset, which required testing on videos with low resolutions and putting our model in every type of situation to see how the algorithm would react and how the output would look like. The expected output was reached a bit later than expected but we were happy with our model.