

# Configuration Manual

MSc Research Project  
Data Analytics

Rohit Jadhav  
Student ID: 20205350

School of Computing  
National College of Ireland

Supervisor: Mr. Vladimir Milosavljevic

National College of Ireland  
Project Submission Sheet  
School of Computing



<b>Student Name:</b>	Rohit Jadhav
<b>Student ID:</b>	20205350
<b>Programme:</b>	Data Analytics
<b>Year:</b>	2021-2022
<b>Module:</b>	MSc Research Project
<b>Supervisor:</b>	Mr. Vladimir Milosavljevic
<b>Submission Due Date:</b>	15/08/2022
<b>Project Title:</b>	Automatic Weapon Detection in CCTV systems Using Deep Learning
<b>Word Count:</b>	1921
<b>Page Count:</b>	17

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.

<b>Signature:</b>	
<b>Date:</b>	15th August 2022

**PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:**

Attach a completed copy of this sheet to each project (including multiple copies).	<input type="checkbox"/>
<b>Attach a Moodle submission receipt of the online project submission</b> , to each project (including multiple copies).	<input type="checkbox"/>
<b>You must ensure that you retain a HARD COPY of the project</b> , 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.

<b>Office Use Only</b>	
Signature:	
Date:	
Penalty Applied (if applicable):	

# Configuration Manual

Rohit Jadhav  
20205350

## 1 Introduction

This configuration manual includes detailed instructions on how to meet the thesis's storage, setup, software, and hardware needs titled "Automatic Weapon Detection in CCTV systems Using Deep Learning". It also includes all the important snapshots which can be used to give insights about thorough process of the research that is followed.

## 2 Hardware & Software Requirements

### 2.1 Hardware Setup

Table 1 shows local hardware configurations while implementing this research.

Component	Specification
Make	Apple Macbook
Model	MacBook Air (M1, 2020)
Memomry (RAM)	8 GB
GPU	Apple M1 (7 cores)
Disk	251 GB SSD

Table 1: Local system configurations

For advanced object detection algorithm training, paid version of Google Colab Cloud service is used. Google Colab Cloud also has free tier service for application having lesser complexity. As we are dealing multi-class object detection high-end system configuration is required for minimizing training time hence Pro version of Google Colab was used. Table 2 depicts detailed specification of Google Colab Pro.

Component	Specification
GPU Make	Tesla
GPU Model	Tesla P100-PCIE
GPU Name	Python 3 Google Compute Engine Backend (GPU)
Memomry (RAM)	16 GB
Disk	167 GB

Table 2: Google Cloud system configurations

## 2.2 Software Setup

Table 3 shows, list of all the softwares used during the implementation of the project. This table also includes Operating System and Programming Language details. It also contains all the web applications and packages employed during the course of this study.

Component	Specification
Operating System	macOS Monterey
Programming Language	Python
Cloud IDE	Google Colab Pro
Model supplementary	Darknet, OpenCV, CUDA (used by Darknet)
Model Visualizations, Implementation & Evaluation	darknet, openCV, matplotlib
Python Packages	drive, os, shutil, numpy, cv2, matplotlib, google.colab
Object Annotation	YOLO-Label
Achritecture Designs	draw.io

Table 3: Google Cloud system configurations

## 3 Data Preprocessing

### 3.1 Data Downloading and Sorting

First of all, google drive is mounted to Google Colab Pro. This requires special permission from the user. Once user grants the permission, gdrive is mounted with Google Colab. Figure 1 shows process of mounting grive to Colab Pro along with necessary imports for the project.

```
# mounting the gdrive
import os;
import shutil;
import numpy as np;
from google.colab import drive
drive.mount('/content/gdrive')

Mounted at /content/gdrive

# creating general link to avoid spaces as model doesnt work with folders with spaces
ln -s /content/gdrive/My\ Drive/ /mydrive
```

Figure 1: Google Drive Mount & Important Packages

Also the process doesn't work well with the file names having spaces in them. As 'My drive' has space in it and it cannot be avoided, simple path link is created.

### 3.2 Data gathering

Open Images v6 has Over 9.6 million photos with annotations for segmentation, object detection, and classification specifically for the use of object detection applications Kuznetsova, Rom, Alldrin, Uijlings, Krasin, Pont-Tuset, Kamali, Popov, Mallocci, Kolesnikov, Duerig and Ferrari (2020). From this dataset, four classes were downloaded namely

Handgun, Torch, Stapler and Remote control. Data from the Open Images was already annotated but it had different format of annotation. To convert that format into the format accepted by YOLO models 'convert\_annotations.py' is used from the Open Images repository.

Over 5500 images were present on the Kaggle dataset for the weapon detection jubaerad (2020). This dataset was divided into two sections. One of the section contained different angled images of Rifles, Bombs, Landmines, Handguns, etc. and the other section had images taken from CCTV footage available on YouTube. Only first section of the images were already annotated amongst them. Second section also had images having multiple objects like Rifles, SMG, Pistol, Rocket Launcher in form of CCTV footage. As study focuses only on Handguns (Revolver/Pistols), only images with revolvers and pistols were taken into consideration. On top of it some images were downloaded from google as confusion objects were missing in this dataset. After addition of confusion objects images the whole dataset was ready for the process of object annotation.

### 3.3 Data Annotation

For Data annotation, YOLO-Label open source application is used <sup>1</sup>. As study focuses on different versions of YOLO models, this works best in this usecase.

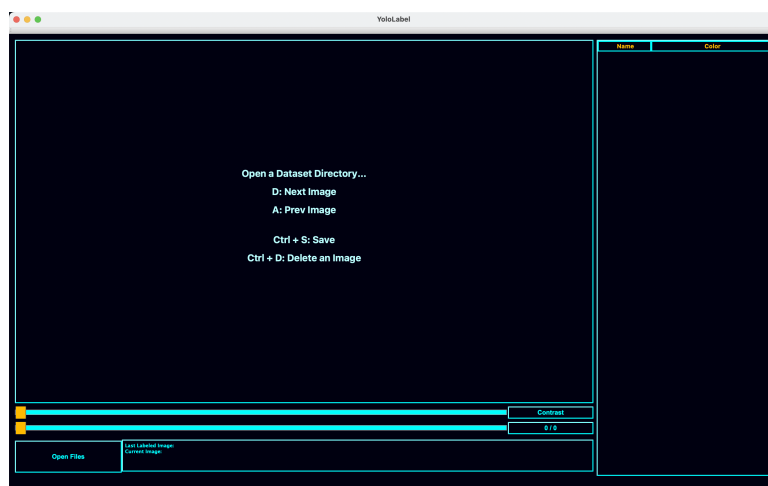


Figure 2: YOLO-Label user interface

Once YOLO-Label is launched, it asks for the location of the images. After selection of location of the images, it asks for the .txt or .names file which contains all the classes required for the object detection. Figure 3 depicts the process of object annotation in YOLO-Label. On the right hand side corner all the object detection classes are present.

<sup>1</sup>[https://github.com/developer0hye/Yolo\\_Label](https://github.com/developer0hye/Yolo_Label)

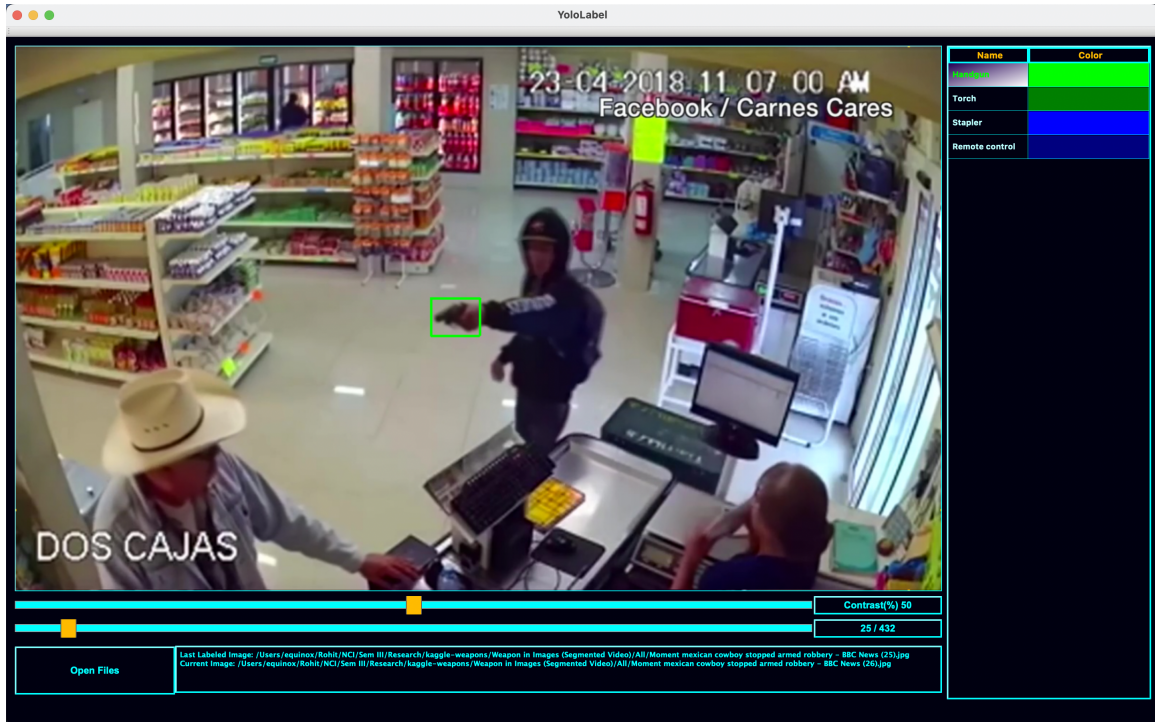


Figure 3: YOLO-Label process

User can mark any number of objects in the images. For every object inside the images associated row is added in .txt file with same name as image. Figure 4 depicts a case of an image having three objects producing three different rows in .txt file.

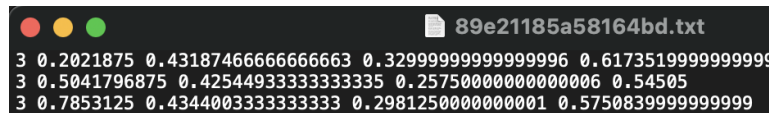


Figure 4: .txt file produces with YOLO-Label having multiple classes in one file

As there can be vast range of images, this process can be time consuming at times.


### 3.4 Data Uploading

Open Images dataset repository is cloned from github directly. Once repository cloning is completed, main.py with downloader command can be used to download all the classes required for the training. multiclass flag is set to download all the images in one single folder. Even if limit 100 is set it is not guaranteed that the each class will have 100 number of images as some classes may have lesser number of total images for that particular class. Figure 5 shows github url and the process for downloading multiple classes at once.

```
# open images v6 images
# !git clone https://github.com/theAIGuysCode/OIDv4_ToolKit.git
!cd OIDv4_ToolKit
# !pip3 install -r requirements.txt

/content/gdrive/MyDrive/WeaponDetection/yolo4/darknet/OIDv4_ToolKit

[ ] # handgun along with confusion objects
!python main.py downloader --classes Handgun Torch Stapler Remote_control --type_csv train --limit 100 --multiclass 1



[INFO] | Downloading ['Handgun', 'Torch', 'Stapler', 'Remote control'] together.
```

Figure 5: Open Images OID process

Kaggle dataset is uploaded with the help of google.colab library. figure 6 shows the process of uploading the dataset into google drive. Images can be directly uploaded in particular folder using drive.google.com. In both the cases, system works exactly the same.

```
#upload local kaggle annotated files to data/obj
from google.colab import files

uploaded = files.upload()
```

Figure 6: Manual Upload process

## 4 Data Modelling

### 4.1 General Setup

Darknet is the key element of the process as all the models employed in this study use darknet as their backbone. Therefore, first darknet is cloned from its github repository. Modified Makefile with necessary changes to enable OpenCV and GPU to utilize full potential of Google Colab Pro platform. After changing Makefile, it is built with 'make' command.

Before Proceeding to further stage, CUDA version is checked in order to perform seamless process ahead. **Note:** CUDA comes by default with Google Colab.

```

# cloning the darknet repo
#!git clone https://github.com/AlexeyAB/darknet

# granting the permission to the darknet folder
%cd darknet/
!chmod a+x ./darknet

/content/gdrive/My Drive/WeaponDetection/yolo4/darknet

# making obj folder inside data folder in darknet
# !mkdir /content/gdrive/MyDrive/WeaponDetection/yolo4/darknet/data/obj/

# make file changes as we are using gpu from google colab
# !sed -i 's/OPENCV=0/OPENCV=1/' Makefile
# !sed -i 's/GPU=0/GPU=1/' Makefile
# !sed -i 's/CUDNN=0/CUDNN=1/' Makefile
# !sed -i 's/CUDNN_HALF=0/CUDNN_HALF=1/' Makefile
# !sed -i 's/LIBSO=0/LIBSO=1/' Makefile

# verify CUDA
!/usr/local/cuda/bin/nvcc --version

nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2020 NVIDIA Corporation
Built on Mon Oct 12 20:09:46 PDT 2020
Cuda compilation tools, release 11.1, V11.1.105
Build cuda_11.1.TC455_06.29190527_0

```

Figure 7: Darknet Setup

Darknet works in specific ways and all the configurations need to be exact in order for it to work. Images are only uploaded in data/obj folder. If the images are kept in different folder and multi-class object detection algorithm is running, sometimes 0.00% accuracy is received for some of the classes. Frequently used paths are saved as constants (see 8)

```

#Set all the Directory paths
yolo_path='/content/gdrive/MyDrive/WeaponDetection/yolo4'
darknet_path= yolo_path + '/darknet'
image_path= darknet_path + '/data/obj/'
os.chdir(darknet_path)
!pwd

```

Figure 8: Commonly used paths

Some helper function are written for visualization and create randomness in testing phase. (see 9)



```

# define helper function to show the chart or the prediction image
def imShow(path):
    import cv2
    import matplotlib.pyplot as plt
    %matplotlib inline

    image = cv2.imread(path)
    height, width = image.shape[:2]
    resized_image = cv2.resize(image, (3*width, 3*height), interpolation = cv2.INTER_CUBIC)

    fig = plt.gcf()
    fig.set_size_inches(18, 10)
    plt.axis("off")
    plt.imshow(cv2.cvtColor(resized_image, cv2.COLOR_BGR2RGB))
    plt.show()

import random

def choose_random_test_file(fname):
    return random.choice(open(fname).readlines())

```

Figure 9: Helper functions

## 4.2 Data split for Train & Test

Uploaded data is splitted into training and testing set with 90:10 ratio.

```

# splitting the files for training and testing
path_list_test=path_list[:int(len(path_list)*0.10)]

path_list=path_list[int(len(path_list)*0.10):]

```

Figure 10: Train Test Split

train.txt and test.txt files are created with the help of splitted data. train and test files only contain the names of the file with either relative or absolute path of the image.

```

# creating test.txt and train.txt files in data folder
%cd ..

#Create train.txt file with path_list
with open('train.txt', 'w') as train:
    #Iterate through all the elements in the list
    for i in path_list:
        #Write the current path at the end of the file
        train.write(i)
#Create test.txt file with path_list_test
with open('test.txt', 'w') as test:
    #Iterate through all the elements in the list
    for i in path_list_test:
        #Write the current path at the end of the file
        test.write(i)

```

Figure 11: Creating Train Test Files

## 4.3 obj and config files

obj.names file contains all the classes required in the study. It is just simple file with each class having separate row. coco.names file is file which contains all the 80 classes employed during the training of Microsoft's COCO dataset. coco.names is replaced with obj.names to make sure there are no clashes between the class names in testing phase. It was observed that if coco.names file is not replaced, wrong labels come on the predictions.

```
# move the classes.txt file as obj.names as it contains all the classes
!cp {darknet_path}/OIDv4_ToolKit/classes.txt {darknet_path}/data/obj.names
# changing the coco.names file as well as it affects the bounding boxes after training
!cp {darknet_path}/OIDv4_ToolKit/classes.txt {darknet_path}/data/coco.names
```

Figure 12: obj.names file

obj.data file contains number of classes required for the model. It also contains locations of train and test files along with location of obj.names. It also contains vital information of the location of the weight files. This location can also be used to start the training process from the last checkpoint if it gets interrupted in between.

As obj.data file contains the backup location of weights trained, it is recommended to give different backup location path to different versions of YOLO so that all the weights are persisted properly without any conflict.

```
#Create obj.data
with open('obj.data','w') as data:
    #Write number of classes
    data.write('classes = 4\n') # changing this everytime according to number of classes
    #Write fully qualified path of the train.txt file
    data.write('train = data/train.txt+\n')
    #Write fully qualified path of the train.txt file
    data.write('valid = data/test.txt+\n')
    #Write fully qualified path of the obj.names file
    data.write('names = data/obj.names+\n')
    #Specify folder path to save trained model weights
    data.write('backup = /mydrive/WeaponDetection/yolo4/training_with_4_objects')
```

Figure 13: obj.data file

Though almost everything is similar, config files are different for each of the YOLO models.

#### 4.3.1 Config for YOLOv4

Height and width for YOLOv4 implementation are both set to 416. The number of processed samples/iteration is indicated by the batch variable. The batch size for this implementation is 64. The subdivisions variable indicates how many small batches are handled simultaneously, it is set to 16. classes denotes number of classes present in the object detection in this case it is set to 4. max\_batches, and steps can vary depending upon number of classes present. filters is set to 27 with  $filters = (classes + 5) * 3$  formula. max\_batches is set to 8000 as there are 4 unique classes present. steps are set as 6400,7200 that is 80% of 8000 and 90% of 8000 respectively. Config file was made using yolo4-custom.cfg. modifications with respect to usecase of project were performed on the same file and yolo4-weapon-4.cfg was created.

```

[net]
# Testing
#batch=1
#subdivisions=1
# Training
batch=64
subdivisions=16
width=416
height=416
channels=3
momentum=0.949
decay=0.0005
angle=0
saturation = 1.5
exposure = 1.5
hue=.1

learning_rate=0.001
burn_in=1000
max_batches=8000
policy=steps
steps=6400,7200
scales=.1,.1

#cutmix=1
mosaic=1

#:104×104 54:52×52 85:26×26 104:13×13 for 416

[convolutional]
batch_normalize=1
filters=32
size=3
stride=1
pad=1
activation=mish

```

Figure 14: YOLOv4 config file snippet 1

```

[convolutional]
batch_normalize=1
size=3
stride=1
pad=1
filters=256
activation=leaky

[convolutional]
size=1
stride=1
pad=1
filters=27
activation=linear

[yolo]
mask = 0,1,2
anchors = 12, 16, 19, 36, 40, 28, 36, 75, 76, 95, 72, 146, 142, 110, 192, 243, 459, 401
classes=4
num=9
jitter=.3
ignore_thresh = .7
truth_thresh = 1
scale_x_y = 1.2
iou_thresh=0.213
cls_normalizer=1.0
iou_normalizer=0.07
iou_loss=ciou
nms_kind=greedynms
beta_nms=0.6
max_delta=5

```

Figure 15: YOLOv4 config file snippet 2

### 4.3.2 Config for Scaled-YOLOv4

config file for Scaled-YOLOv4 is almost same as YOLOv4. Only few minor changes are there with respect to configurations like letter\_box variable is set active in Scaled-YOLOv4. config file was made using yolo4-csp.cfg. Modification with respect to usecase

of project were performed on the same file and yolo4-csp-weapon.cfg was created.

```
[net]
# Testing
#batch=1
#subdivisions=1
# Training
batch=64
subdivisions=16
width=416
height=416
channels=3
momentum=0.949
decay=0.0005
angle=0
saturation = 1.5
exposure = 1.5
hue=.1

learning_rate=0.001
burn_in=1000
max_batches=8000
policy=steps
steps=6400,7200
scales=.1,.1

mosaic=1

letter_box=1

ema_alpha=0.9998

#optimized_memory=1

#23:104×104 54:52×52 85:26×26 104:13×13 for 416
```

Figure 16: Scaled-YOLOv4 config file snippet 1

```
[convolutional]
batch_normalize=1
size=3
stride=1
pad=1
filters=256
activation=mish

[convolutional]
size=1
stride=1
pad=1
filters=27
activation=logistic

[yolo]
mask = 0,1,2
anchors = 12, 16, 19, 36, 40, 28, 36, 75, 76, 55, 72, 146, 142, 110, 192, 243, 459, 401
classes=4
num=9
jitter=.1
scale_x_y = 2.0
objectness_smooth=0
ignore_thresh = .7
truth_thresh = 1
#random=1
resize=1.5
iou_thresh=0.2
iou_normalizer=0.05
cls_normalizer=0.5
obj_normalizer=4.0
iou_loss=ciou
nms_kind=diounms
beta_nms=0.6
new_coords=1
```

Figure 17: Scaled-YOLOv4 config file snippet 2

## 5 Implementation

train.txt, test.txt, obj.names, obj.data and config files are essential in order to work with YOLO models. Once this files are created, training can be started. 'darknet detector train' along with location of the config is used to train the YOLO models if training from scratch. As for this study, pre-trained weights for both YOLOv4 and Scaled-YOLOv4 were used to get better and faster results location of respective pre-trained weights are also passed with same command.

### 5.1 Training YOLOv4 model

For YOLOv4 implementation, yolo4-weapon-4.cfg config file was used. Also pre-trained weight named yolov4.conv.137 was trained on Microsoft's COCO dataset was used for better performance.

```
# training from COCO set
!./darknet detector train data/obj.data cfg/yolov4-weapon-4.cfg yolov4.conv.137 -dont_show -map
```

Figure 18: training YOLOv4

Sometimes, Even with paid subscription of Google Colab Pro it was observed that after few hours of training if the system is idle or if there is no movement from user, GPU used to get deallocated. Also quite often, it can be a scenario where due to some other issue GPU is disconnected. For instances like this, to start retraining from last saved checkpoint the same 'darknet detector train' command can be run with location last trained weights.

```
# training from last checkpoint
!./darknet detector train data/obj.data cfg/yolov4-weapon-4.cfg /mydrive/WeaponDetection/yolo4/training_with_4_objects_merged/yolov4-weapon-4_last.weights -dont_show -map
```

Figure 19: restarting training YOLOv4

### 5.2 Training Scaled-YOLOv4 model

For Scaled-YOLOv4 implementation, yolo4-csp.cfg config file was used. Also pre-trained weight named yolov4-csp.conv.142 was trained on Microsoft's COCO dataset was used for better performance.

```
# training scaled yolo with COCO pre-trained model
%cd [darknet_path]
!./darknet detector train data/obj.data cfg/yolov4-csp-weapon.cfg yolov4-csp.conv.142 -dont_show -map
```

Figure 20: training Scaled-YOLOv4

In case of interruptions, same process like YOLOv4 can be followed.

```
# training scaled-yolo from last checkpoint
!./darknet detector train data/obj.data cfg/yolov4-csp-weapon.cfg {yolo_path}/training_with_4_objects_scaled_yolo/yolov4-csp-weapon_last.weights -dont_show -map
```

Figure 21: restarting training Scaled-YOLOv4

**Note :** Every time model is trained charts are drawn while training is in progress. Naming convention followed by this chart file is according to config file used for the training. General chart.png gives insights about latest model being trained at the moment. In case of any interruptions and restarting the training from last checkpoint, both of these chart files get overwritten losing the progress up until that point. Therefore, it is advised to download chart files before restarting the training process.

## 6 Evaluation

For every 1000 iteration new file is stored in backup folder. Every weight file has different mAP@0.50. There are two ways to observe mAP of the model. First one is to check the mAP with command 'darknet detector map' with individual weight files. Second option is to get insights from chart produces by model during the training.

### 6.1 Evaluation of YOLOv4

```
# checking map of individual weights of multi weight
weight_path = '/mydrive/WeaponDetection/yolo4/training_with_4_objects_merged/'
for f in os.listdir(weight_path):
    weight_file = weight_path + f;
    print(f'Running file {weight_file}')
    !./darknet detector map data/obj.data cfg/yolov4-weapon-4.cfg {weight_file} -points 0

159 conv 1024 3 x 3/ 1 13 x 13 x 512 -> 13 x 13 x1024 1.595 BF
160 conv 27 1 x 1/ 1 13 x 13 x1024 -> 13 x 13 x 27 0.009 BF
161 yolo
[yolo] params: iou_loss: ciou (4), iou_norm: 0.07, obj_norm: 1.00, cls_norm: 1.00, delta_norm: 1.00, scale_x_y: 1.05
nms_kind: greedy (1), beta = 0.600000
Total BFLOPS 59.585
avg_outputs = 490173
Allocate additional workspace_size = 52.43 MB
Loading weights from /mydrive/WeaponDetection/yolo4/training_with_4_objects_merged/yolov4-weapon-4_last.weights...
seen 64, trained: 320 K-images (5 Kilo-batches_64)
Done! Loaded 162 layers from weights-file

calculation mAP (mean average precision)...
Detection layer: 139 - type = 28
Detection layer: 150 - type = 28
Detection layer: 161 - type = 28
60
detections_count = 110, unique_truth_count = 42
class_id = 0, name = Handgun, ap = 68.90% (TP = 20, FP = 7)
class_id = 1, name = Torch, ap = 65.00% (TP = 2, FP = 0)
class_id = 2, name = Stapler, ap = 66.67% (TP = 2, FP = 1)
class_id = 3, name = Remote control, ap = 90.00% (TP = 5, FP = 1)

for conf_thresh = 0.25, precision = 0.76, recall = 0.69, F1-score = 0.73
for conf_thresh = 0.25, TP = 29, FP = 9, FN = 13, average IoU = 61.94 %

IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
mean average precision (mAP@0.50) = 0.726410, or 72.64 %
Total Detection Time: 2 Seconds
```

Figure 22: mAP YOLOv4

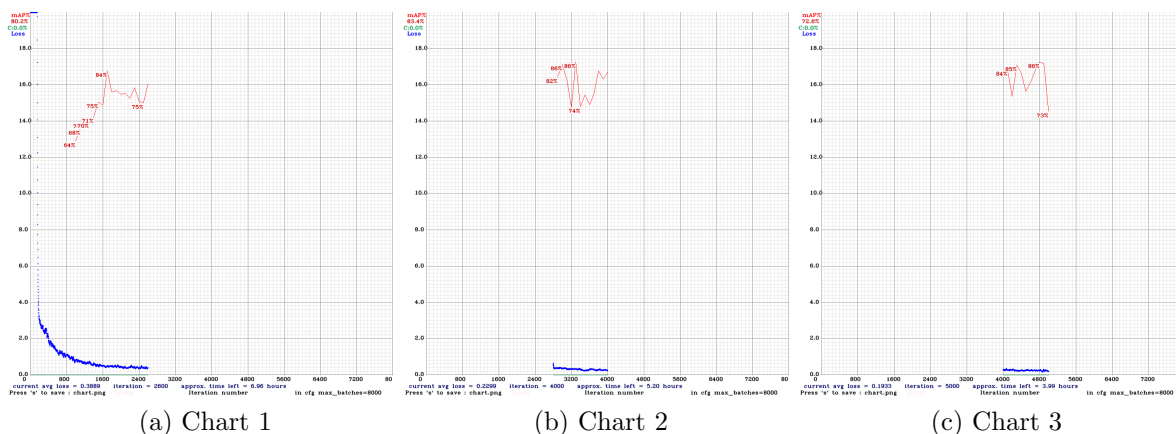


Figure 23: YOLOv4 charts

For implementation of YOLOv4 model GPU was disconnected twice. Therefore, there were three different charts. (see 23).

Model training for YOLOv4 was stopped once there was no significance change in mAP and average loss was less than 0.2.

## 6.2 Evaluation of Scaled-YOLOv4

```
# checking map of individual weights of multi weight
weight_path = '/mydrive/WeaponDetection/yolo4/training with 4 objects scaled yolo/'
for f in os.listdir(weight_path):
    weight_file = weight_path + f;
    print(f'Running file {weight_file}')
    !./darknet detector map data/obj.data cfg/yolov4-csp-weapon.cfg {weight_file} -points 0
1/3 conv 2/ 1 x 1/ 1 13 x 13 x1024 -> 13 x 13 x 2/ 0.009 BF
174 yolo
[yolo] params: iou loss: ciou (4), iou_norm: 0.05, obj_norm: 0.40, cls_norm: 0.50, delta_norm: 1.00, scale_x_y: 2.00
nms kind: dioums (2), beta = 0.600000
Total BFLOPS 50.281
avg_outputs = 357837
Allocate additional workspace_size = 52.43 MB
Loading weights from /mydrive/WeaponDetection/yolo4/training with 4 objects scaled yolo/yolov4-csp-weapon_last.weights...
seen 64, trained: 422 K-images (6 Kilo-batches_64)
Done! Loaded 175 layers from weights-file

calculation mAP (mean average precision)...
Detection layer: 144 - type = 28
Detection layer: 159 - type = 28
Detection layer: 174 - type = 28
60
detections count = 604, unique truth count = 42
class_id = 0, name = Handgun, ap = 57.07% (TP = 16, FP = 6)
class_id = 1, name = Torch, ap = 68.75% (TP = 2, FP = 1)
class_id = 2, name = Stapler, ap = 83.33% (TP = 2, FP = 2)
class_id = 3, name = Remote control, ap = 100.00% (TP = 5, FP = 2)

for conf_thresh = 0.25, precision = 0.69, recall = 0.60, F1-score = 0.64
for conf_thresh = 0.25, TP = 25, FP = 11, FN = 17, average IoU = 52.88 %

IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
mean average precision (mAP@0.50) = 0.772888, or 77.29 %
```

Figure 24: mAP Scaled-YOLOv4

While training for Scaled-YOLOv4 there were no interruptions. Hence we have single chart for the whole process (see 25).

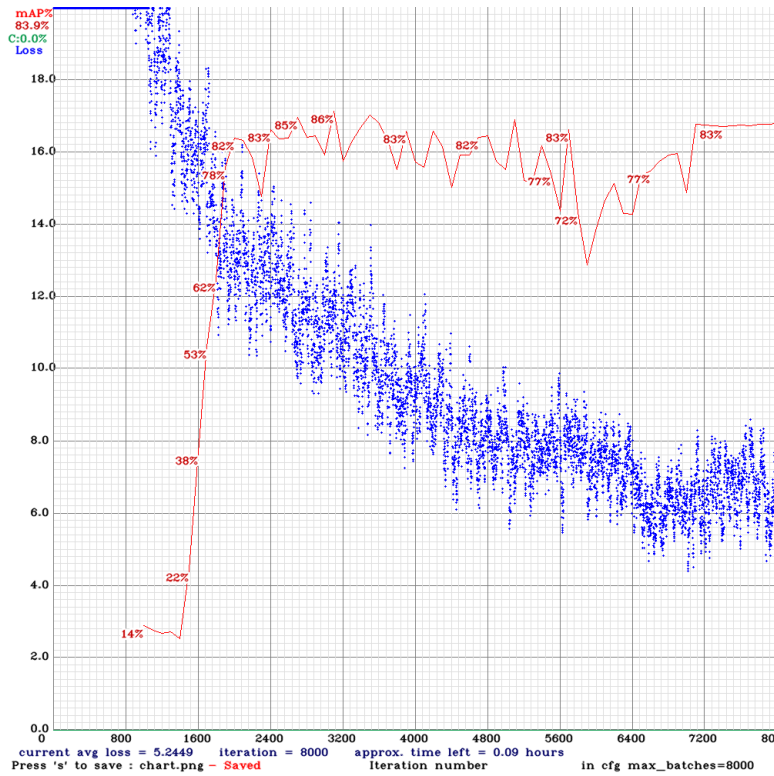


Figure 25: Scaled-YOLOv4 chart

mAP is computed after every 4 Epochs with valid dataset described in obj.data ( $1\text{Epoch} = \text{pictures} - \text{in} - \text{training}/\text{batch}$ ). Charts gives understandings about the loss function and average loss along with transition of mAP over the course of training.

## 7 Testing

Models with highest mAP are used for testing. After performing tests for images results are stored in 'predictions.jpg' file. helper function 'imshow' is used to show the prediction file.

When models are used for testing video files, it only shows the calculations performed during the testing and results are stored in new file with added bounding boxes as predictions.



## 7.1 Testing YOLOv4

```
# Testing the random image
fname = choose_random_test_file( /content/gdrive/MyDrive/WeaponDetection/yolo4/darknet/data/ + 'test.txt')
fname = fname.replace("\n", "")
#test out our detector!
print("Running test for file: {fname}")
!./darknet detect cfg/yolov4-weapon-4.cfg /content/gdrive/MyDrive/WeaponDetection/yolo4/training_with_4_objects_merged/yolov4-weapon-4_best.weights {fname} -dont_show_inshow( 'predictions.jpg' )

154 conv 512 1 x 1/ 1 13 x 13 x1024 -> 13 x 13 x 512 0.177 BF
155 conv 1024 3 x 3/ 1 13 x 13 x 512 -> 13 x 13 x1024 1.595 BF
156 conv 512 1 x 1/ 1 13 x 13 x1024 -> 13 x 13 x 512 0.177 BF
157 conv 1024 3 x 3/ 1 13 x 13 x 512 -> 13 x 13 x1024 1.595 BF
158 conv 512 1 x 1/ 1 13 x 13 x1024 -> 13 x 13 x 512 0.177 BF
159 conv 1024 3 x 3/ 1 13 x 13 x 512 -> 13 x 13 x1024 1.595 BF
160 conv 27 1 x 1/ 1 13 x 13 x1024 -> 13 x 13 x 27 0.009 BF
161 yolo

[yolo] params: ion_loss: c1ou (4), ion_norms: 0.07, obj_norms: 1.00, cls_norms: 1.00, delta_norm: 1.00, scale_x_y: 1.05
mme_kind: greedyms (1), beta = 0.600000
Total BFLOPS 59.585
avg_outputs = 480133
Allocate additional workspace size = 52.43 MB
Loading weights from /content/gdrive/MyDrive/WeaponDetection/yolo4/training_with_4_objects_merged/yolov4-weapon-4_best.weights...
mem 64, trained: 327, Kimages (4 K110-batches_64)
Done! Loaded 162 layers from weights-file
Detection layer: 139 - type = 28
Detection layer: 150 - type = 28
Detection layer: 161 - type = 28
/content/gdrive/MyDrive/WeaponDetection/yolo4/darknet/data/obj/MomentmexicancowboytoppedarmedrobberyM8CNews49.jpg: Predicted in 20.268000 milli-seconds.
Handgun: 100%
Unable to init server: Could not connect: Connection refused.
(predictions:4409): Gtk-WARNING **: 12:48:26.639: cannot open display:



```

Figure 26: Testing YOLOv4 model on Images

```
# object detection on video
!./darknet detector demo data/obj.data cfg/yolov4-weapon-4.cfg (yolo_path)/training_with_4_objects_merged/yolov4-weapon-4_best.weights -dont_show (yolo_path)/test_videos/gun-video.mp4 -i 0 -out_filename (yolo_path)/result_videos/gun-video-result.mp4

Handgun: 61%
FPS:40.6 AVG_FPS:43.2
cvWriteFrame
Objects:
Handgun: 60%
FPS:40.7 AVG_FPS:43.2
cvWriteFrame
Objects:
Handgun: 60%
FPS:40.6 AVG_FPS:43.2
cvWriteFrame
Objects:
Handgun: 55%
FPS:40.7 AVG_FPS:43.2
cvWriteFrame
Objects:
Handgun: 57%
FPS:40.5 AVG_FPS:43.2
```

Figure 27: Testing YOLOv4 model on Videos

## 7.2 Testing Scaled-YOLOv4

```
# Testing the random image
fname = choose_random_test_file(['/content/gdrive/MyDrive/WeaponDetection/yolo4/darknet/data/' + 'test.txt'])
Click to show 36 definitions.
str: fname
      fname')
      sapon.cfg {yolo_path}/training_with_4_objects_scaled_yolo/yolov4-csp-weapon_best.weights {fname} -dont_show
View
'data/obj/e6508678cae0be52.jpg'
187 conv 512 3 x 3/ 1 13 x 13 x 512 -> 13 x 13 x 512 0.797 BF
168 conv 512 1 x 1/ 1 13 x 13 x 512 -> 13 x 13 x 512 0.089 BF
169 conv 512 3 x 3/ 1 13 x 13 x 512 -> 13 x 13 x 512 0.797 BF
170 route 169 164 -> 13 x 13 x1024
171 conv 512 1 x 1/ 1 13 x 13 x1024 -> 13 x 13 x 512 0.177 BF
172 conv 1024 3 x 3/ 1 13 x 13 x 512 -> 13 x 13 x1024 1.595 BF
173 conv 27 1 x 1/ 1 13 x 13 x1024 -> 13 x 13 x 27 0.009 BF
174 yolo
[yolo] params: iou_loss: ciou (4), iou_norm: 0.05, obj_norm: 0.40, cls_norm: 0.50, delta_norm: 1.00, scale_x_y: 2.00
nms_kind: diou_nms (2), beta = 0.600000
total BFLOPS 50.281
avg_outputs = 357837
Allocate additional workspace_size = 52.43 MB
Loading weights from /content/gdrive/MyDrive/WeaponDetection/yolo4/training_with_4_objects_scaled_yolo/yolov4-csp-weapon_best.weights...
seen 64, trained: 198 K-images (3 Kilo-batches 64)
Done! Loaded 175 layers from weights-file
Detection layer: 144 - type = 28
Detection layer: 159 - type = 28
Detection layer: 174 - type = 28
data/obj/e6508678cae0be52.jpg: Predicted in 18.387000 milli-seconds.
torch: 79%
Unable to init server: Could not connect: Connection refused
(predictions:4461); Gtk-WARNING **: 12:51:43.551: cannot open display:
```




Figure 28: Testing Scaled-YOLOv4 model on Images

```
# object detection on video
1./darknet detector demo data/obj.data cfg/yolov4-csp-weapon.cfg {yolo_path}/training_with_4_objects_scaled_yolo/yolov4-csp-weapon_best.weights
-dont_show {yolo_path}/test_videos/gun-video-1.mp4 -i 0 -out_filename {yolo_path}/result_videos/gun-video-1-scaled-result.mp4

cvWriteFrame
Objects:
Handgun: 67%
Handgun: 30%
FPS:45.8      AVG_FPS:40.8

cvWriteFrame
Objects:
Handgun: 68%
Handgun: 30%
FPS:46.0      AVG_FPS:40.8

cvWriteFrame
Objects:
Handgun: 70%
Handgun: 32%
FPS:46.1      AVG_FPS:40.8

cvWriteFrame
Objects:
Handgun: 69%
Handgun: 29%
```

Figure 29: Testing Scaled-YOLOv4 model on Videos

Figure 30 shows snapshot taken from the video file on which YOLO models were performed. Snapshot was taken when video was paused in fullscreen.



Figure 30: Snapshot of video result file

## References

jubaerad (2020). Weapons in Images.

**URL:** <https://www.kaggle.com/datasets/jubaerad/weapons-in-images-segmented-videos>

Kuznetsova, A., Rom, H., Alldrin, N., Uijlings, J., Krasin, I., Pont-Tuset, J., Kamali, S., Popov, S., Mallocci, M., Kolesnikov, A., Duerig, T. and Ferrari, V. (2020). The open images dataset v4: Unified image classification, object detection, and visual relationship detection at scale, *IJCV* .