Electrical and Computer Engineering

EECE499: Capstone Project Report

Weapon Detection System

Abdullah Hassan

Aida AlAwadhi

Noora AlHamidh

Yazeed AlShehri

Supervisor: **Dr. Vinod Pangracious**

April 20, 2023

Contents

List of Figures

Chapter 1

Introduction

Safety is a vital need for all people. Yet there are unfortunate incidents where a weapon threatens lives. Our project is a weapon detection system to be implemented with drones and CCTV cameras. With it security personnel and users can get critical warning of threats before its too late. A major element of our project is machine learning. With it we can create a model that automatically detect weapons and weapon type in practically real time. Once a weapon is detected users will also receive an message with details of the detection. Footage can also be viewed in order to double check for false positives. There are enough people to be carefully, continuously and consistently monitoring our safety. But with our system we can have the that continuous focus on searching for threats. Thus our user will be quicker to action and save lives.

Chapter 2

The Needs and Values of The Weapon Detection System

The primary need for creating the machine learning driven WDS(Weapon Detection System) is to enhance public safety by detecting the presence of weapons in real-time and alerting relevant authorities. By quickly forwarding the threat detected to the relevant authority or security personnel, the system helps to reduce response time to the treat, increasing the chances of neutralizing the threat before any harm is caused.

The WDS provides an extra layer of safety and security to the public and enables law enforcement to respond proactively to potential threats before they can cause harm. WDS uses a machine learning model trained to accurately identify the presence of weapons and the type of weapon, both of which are accompanied with a confidence level in the threat classification reported. The threat confidence percentage is important as it helps the relevant authorities or security personnel make informed decisions based on the severity of threat. The system accurately provides the confidence percentage in there being a threat, which helps in determining the appropriate response and level of alertness. By detecting firearms and weapons in real-time, the authorities can intervene before any harm is done, leading to a reduction in crime. WDS aims to reduce casualties shootings, terrorism incidents and other crimes involving firearms, enhancing public safety, and reducing incidents of armed violence. Another reason for developing the WDS is to reduce crime rates. Additionally, the public knowledge of the existence of such advanced proactive security measures in place deters evildoers from committing crimes.

Additionally, WDS is capable of weapon classification. The system not only detects the presence of weapons but also provides the type of weapon detected, allowing for a more informed response from security personnel. Last but not least, we increase public confidence. Utilizing WDS to detect firearms and weapons in real-time will increase the public's confidence in the ability of authorities to keep the community safe. This can lead to increased trust in the authorities and a safer society.

2.1 State of the art

Systems on the current market are just focused on guns. In nations with successful gun control such systems aren't useful but the public is still in danger. In such nations criminals are more likely to use weapons such as knives. In such scenarios an early warning systems such as our would have in more successful outcomes.Further more many of these companies are limited to north America and are not allowed to operate else where.

The most common and historically used type of weapon detection systems are acoustic based. They respond to the sound of bullets firing. This means first responds/guards are only alerted once the crime happens and are delayed from entering the scene.

Other systems like metal screenings are also used. Recently some advanced AI metal detectors can distinguish between guns and other objects like phones and keys [8]. However, criminals have been able to sneak guns and other weapons past metal detectors and similar system through security flaws of the preemies.

More recently has there been AI based systems combine with CCTV. Again these systems are solely focused on gun detection and do not include other weapons [9].

Chapter 3

Weapon Detection Methodology

3.1 Convolutional neural network deep learning technique

A major component of our project is machine learning. Machine learning is a sub-field of artificial intelligence, where algorithms learn from experience to perform some task. We are using machine learning since tasks such as object detection are too complex for engineers to program traditionally.

More specifically we will be using neural networks(NN), which is a sub-field of machine learning. Neural networks are inspired by the brain where many simple neurons with many connections enable us to perform highly complex tasks. NN algorithms are structured into several node layers: the input layer, the hidden layer(s), and an output layer. The information is entered through the network through the input layer. The hidden layers take their inputs from the outputs of the previous layer (the input or some other hidden layer), analyze, process, and send it to the next. [1]

Each layer has a set of nodes, and each node has a particular weight and threshold. The weight determines the impact of the connection between nodes. A positive weight indicates that a node excites the other node, a negative value suppresses the other node and a larger magnitude has a higher impact. The threshold of the a node determines when it will activate sending the output to the next layer. [3] Neural networks use some feedback process to adjust the values of the threshold and weight of nodes. [5]

Convolution neural networks (CNN) are particularly powerful for image classification. Convolutions are hidden layers that do mathematical functions such as summarizing and filtering. This allows the network to extract valuable features for processing. Different hidden layers may focus on analyzing specific image features. [1] CNNs also include the pooling layer which performs dimensionality reduction to limit reducing complexity. Furthermore, in the fully connected layer, the nodes of the output layer are directly connected to the nodes from the previous and it classifies based on the extracted features. [2] Reference the figure below 3.1 for an illustration of a typical CNN .

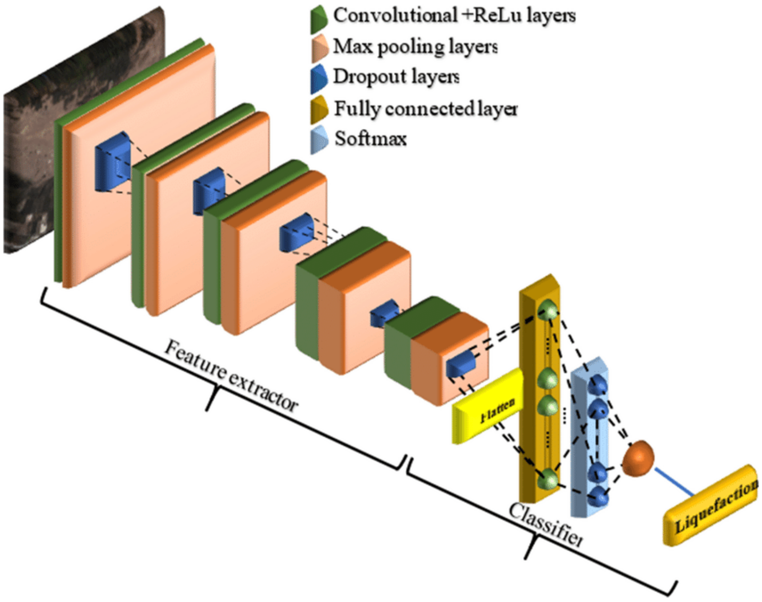


Figure 3.1: A diagram of a typical convolutional neural network

[4]

3.2 Weapon Detection System using CNN technique

Our Weapon Detection system will utilize the convolutional neural network deep learning technique to detect, identify and classify weapons. As shown in figure 3.2 below, an input data of images or a video stream is fed to the machine learning algorithm as the first step. A prediction is made showing whether is detected or not, the classification of the weapon if it is detected and the confidence level of how sure the algorithm is of it's detection. An evaluation is then made to check the accuracy and precision of the algorithm and this evaluation is fed back into the training algorithm for tuning and adjustments. Now that the prediction side of the algorithm is discussed, we will now discuss the training part of the algorithm. Firstly, a set of data called the training data is passed through to be trained in the algorithm. It consists of pictures and video frames of different people holding different types of weapons that are labeled in order for the algorithm to know which weapon is which. After training the training algorithm is added to the model as a whole which in turn is added to the machine learning algorithm to aid in making a prediction.

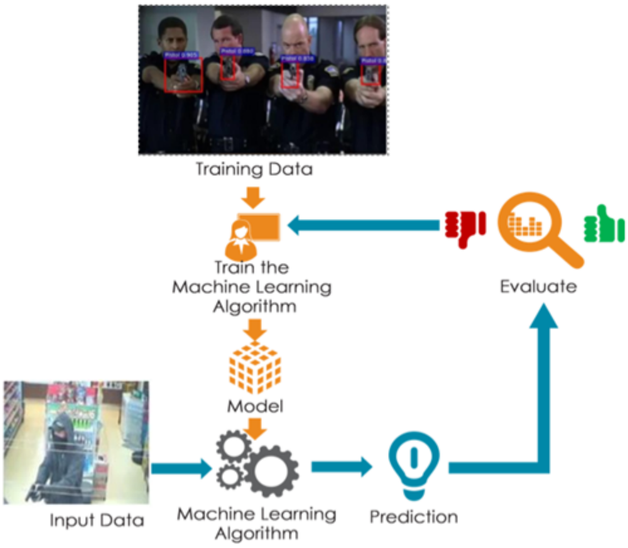


Figure 3.2: A diagram of how WDS will utilize CNN deep learning technique

Chapter 4

Project Description

Our system will be an application that automatically detect different weapons on surveillance drones/CCTV and alerts security personnel and/or other users. The application will be able to read and analyze live camera footage close to real-time. The alters will include information on the detection. Within the application, security personnel can also look at the video footage manually.

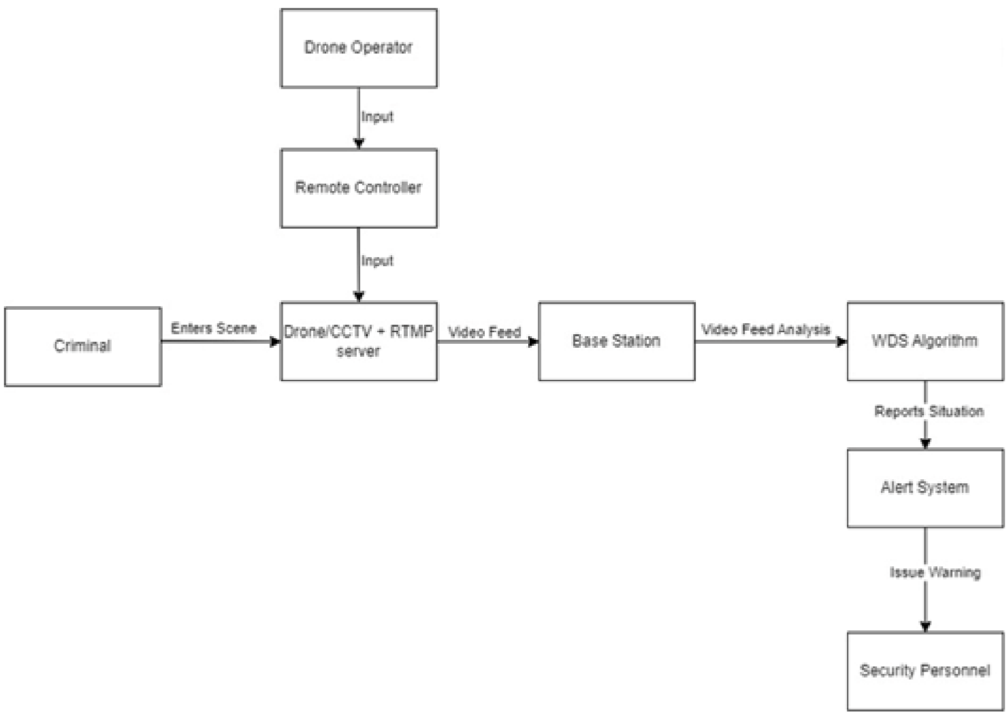


Figure 4.1: Block Diagram of our system

Such as our block diagram 4.1 shows, the drone will be operated by the drone operator via a remote control. The message diagram shows 4.2 how the remote control while passing the operators input to drone will be displaying feedback and the drones video stream to the drone operator.

As an be seen in 4.1 once our criminal enters the view monitored area they are captured by the drone. In our message sequence diagram 4.2 this is occurs as the message being sent is the criminal entering the surroundings.

Both our block diagram 4.1 and message sequence char 4.2t present the interaction of the drone, the base station: a computer, and the "WDS Algorithm": our trained machine-learning model. The stream of frames from the drone will be sent live to the base station. Then base station will be input the frames into our WDS Algorithm.

In out message sequence hart wee show to alternative cases 4.2. Nothing will happen if no weapons are detected. In the case a weapon is in a frame of footage, the "WDS Algorithm" will be able to detect it. It will also classify the weapon type such as a gun or knife. The classification of different weapons is especially vital for security personnel to know in order to adapt strategies to the situation. Additionally, the model will highlight the section of the frame(s)/video that it has detected the weapon. Highlighting the video where the weapon is detected is incredibly useful for personnel to double-check for false positives and prevent panic.

Once the model has detected the weapon, an alert will be created and handled by our warning system .Theses alerts will bee quickly sent by the warning system to security personal as shown in both diagrams 4.1,4.2.The alert will include information such as the weapon detected, the time, and the camera that detected it. Our alert sub-system enables our project to be more useful as security personnel can react more quickly to security threats.

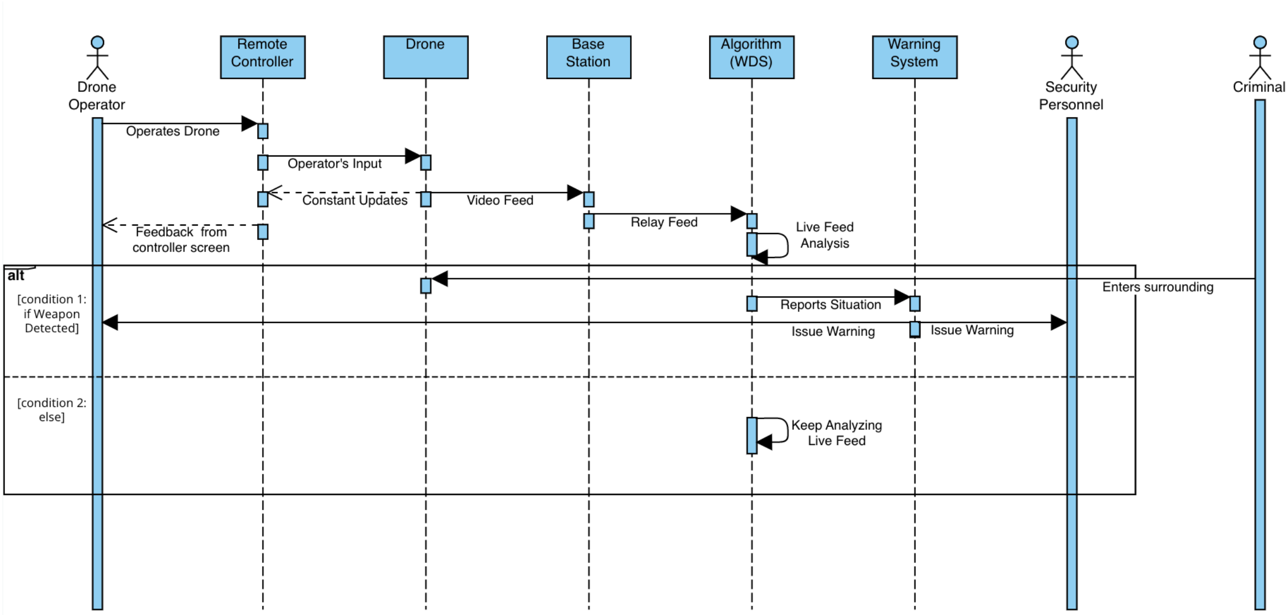


Figure 4.2: Message sequence chart of our system

4.1 Project requirements

- Large and diverse dataset of images with armed and non-armed people to use to train the model as well as a large and diverse dataset of images containing different weapons to train the model to accurately recognize various types of weapons; including firearms, blades, and other dangerous objects. This dataset should include images from different angles, lighting conditions, and resolutions to ensure the algorithm can generalize well to new situations. Alternatively, WDS could have a minimum CCTV system requirement on which it may operate to avoid running the WDS on none-compatible systems.
- High-performance computing power to process the data in real-time and ensure the algorithm can classify images quickly and efficiently. This is critical for applications where the algorithm needs to be integrated into real-time video streams.
- An optimized algorithm that can quickly process and classify images in real-time.
- A compatible CCTV system to host and utilize WDS.
- A method to handle results such as false positives or false negatives; incorporating human review, namely security personal is the primary candidate.

4.2 Project constraints

- Careful planning and oversight are needed to address privacy and ethical concerns, such as avoiding false alarms and preventing the technology from being misused. The technology must be created and used in a way that respects people's privacy and rights, and it must minimize the possibility of abuse or false escalation.
- The accuracy of the algorithm can be impacted by variations in camera angles, lighting, and resolution; therefore, a robust algorithm that can adapt to these changes is needed. To make sure the algorithm can operate dependably under various circumstances, it should be tested and validated in a variety of real-world scenarios.
- It is challenging to find weapons in partially obscured or occluded images, which calls for strong object detection and recognition abilities. Due to the variety of ways that weapons can be concealed or obscured, this can be difficult, and it necessitates that the algorithm be extremely adaptable and generalizable.
- It can be difficult to strike a balance between accurately detecting weapons and processing images in real-time, and this calls for careful algorithmic tuning and optimization. To make sure the algorithm can meet the requirements of the application, this may entail making trade-offs between accuracy, processing speed, and resource utilization.
- Cost-effective implementation. Implementation can be significantly hampered by the expense and complexity of acquiring, storing, and processing vast amounts of data and computing resources. To ensure that the technology can be deployed and maintained in an economical manner, very careful planning and budgeting are required.

Chapter 5

Product Standardization and Manual

5.1 Our specifications

5.1.1 Drone

- Quiet
- Good camera quality
- High Field of view
- Live video transmission
- Fulfills legal requirements
- Stability and long flight duration
- Minimum 1 km coverage

5.1.2 WDS algorithm / Model

- High precision, accuracy and particularly recall
- Low latency
- Highly compatible

5.1.3 Base station

- SMS/Email compatible
- High end hardware:Good CPU and GPU, Can process high quality video streams

5.2 Drone specifications

- Price:2,500AED
- 10km distance with video transmission with 30fps
- Level 5 wind resistance –> 29-38kph
- 4k video –> clearly see small objects
- 12MP photo
- 180 angle available –> wide view
- Less than 249g –> UAE legal requirements
- 31 mins flight time (higher with additional battery)
- Latency 200ms
- Video format MP4
- Max speed 16 m/s

5.3 How to Operate Drone

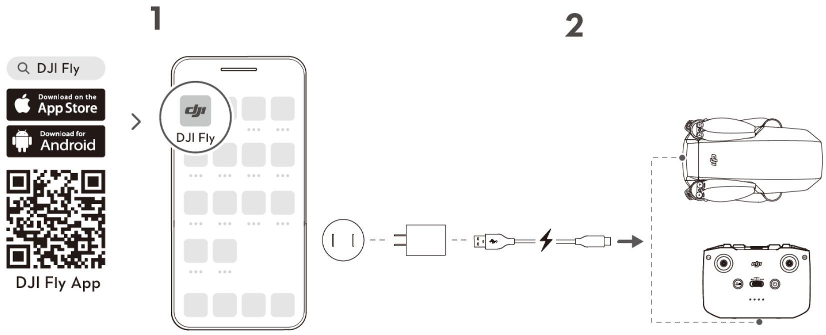


Figure 5.1: 1: Download the app. 2: Charge the remote and drone

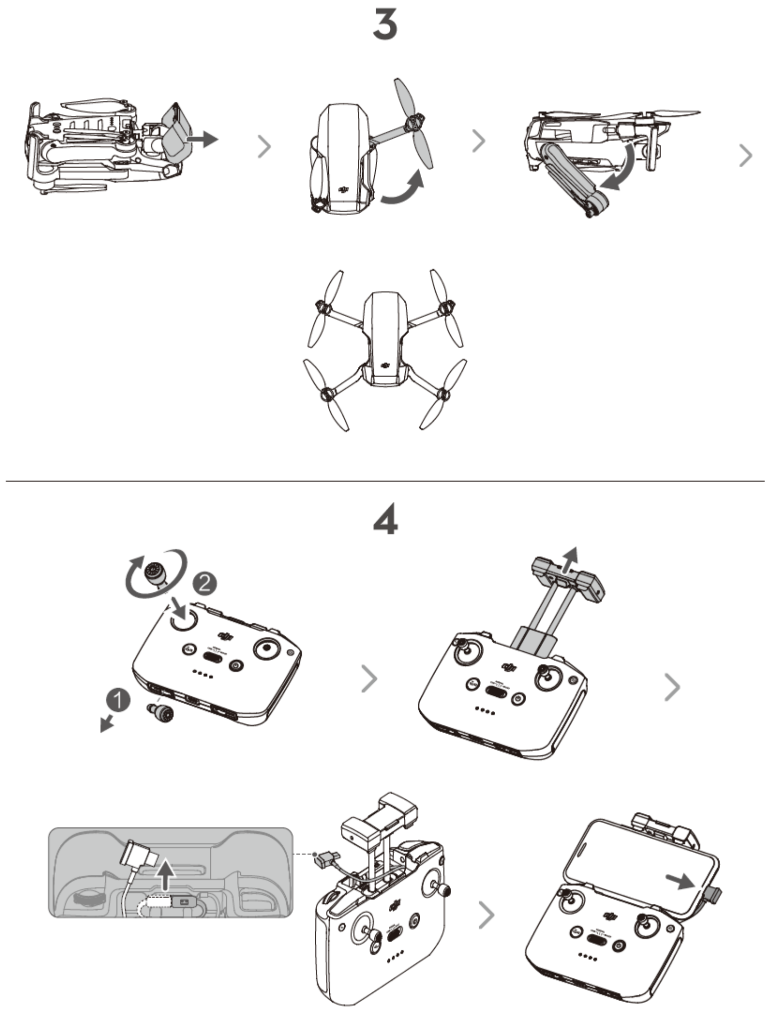


Figure 5.2: 3: Open up the drone. 4: Prepare the controller and connect the phone to it

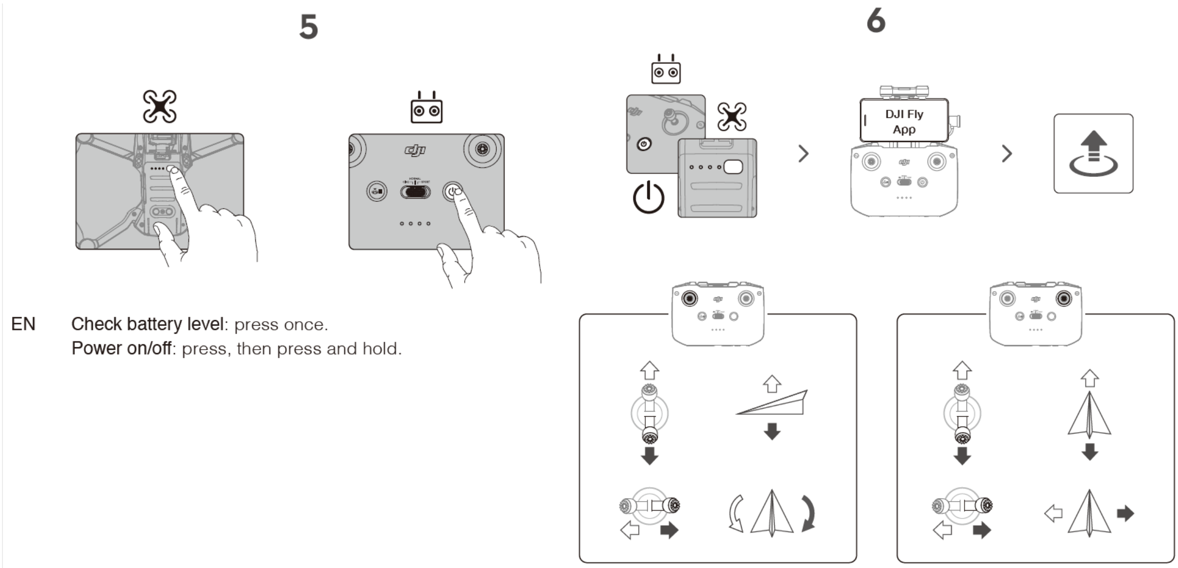


Figure 5.3: 5: Controller power and battery features. 6: Controller operation

Figure 5.4: Further details and complete instructions can be found in the app's drone manual

5.4 How to setup and operate our Software

- System requirements for running the algorithm: latest and most modern OS, CPU and GPU specifications.
- Installation guide: install the model algorithm and live demo file on any updated modern IDE and the application to operate the drone.

- Operating the algorithm: use the model algorithm on any updated IDE and install necessary libraries, packages etc.
- Troubleshooting methods
- Maintenance procedures: update the algorithm to patch any bugs or adapt to updated data-sets.

5.4.1 Inputs and setup

- Add guard faces and user email address
- Input configurations for live stream and video footage
- Hardware requirements for the system
- Optimum features of the project

5.4.2 Expected Outputs

- Output format of analyzed video footage
- Output interpretation
- Exporting video footage results
- Alert message (email/SMS etc.)

Chapter 6

Alternative Solutions

6.1 Weapon Detection using SVM and SIFT

As a feature extractor, the histogram of oriented gradients (HOG) and sped-up robust features (SURF) are used. As well as the machine learning algorithms, SIFT (scale-invariant feature transform) and SVM (support vector machine) are employed to recognize weapons in images. However, these algorithms are not very accurate and have a slow performance for real-time applications, taking more than 14 s per picture to detect a weapon [7]. Additionally, these algorithms employ manually created feature extraction criteria rather than learning features on their own.

6.2 Damage Inspector Drone

Potholes and building cracks can be detected using a flying drone's camera. The drone will gather images of detected potholes and cracks on buildings and streets. To help operators reach the spot quickly, the drone will also save the location of the potholes and cracks found.

6.3 Preferred Solution

The weapon detection project will focus on CCTV and drone cameras using Convolutional Neural Network (CNN) machine learning algorithm. CNN enhances performance and reduces a significant amount of human effort to save time. The convolutional neural network (CNN) has produced ground-breaking results in terms of object categorization and detection.

In traditional image processing issues like grouping, detection, and localization, it has so far produced the best results compared with SIFT- and SVM-based algorithms [6]. Figure 6.1 depicts the accuracy chart of CNN vs other ML.

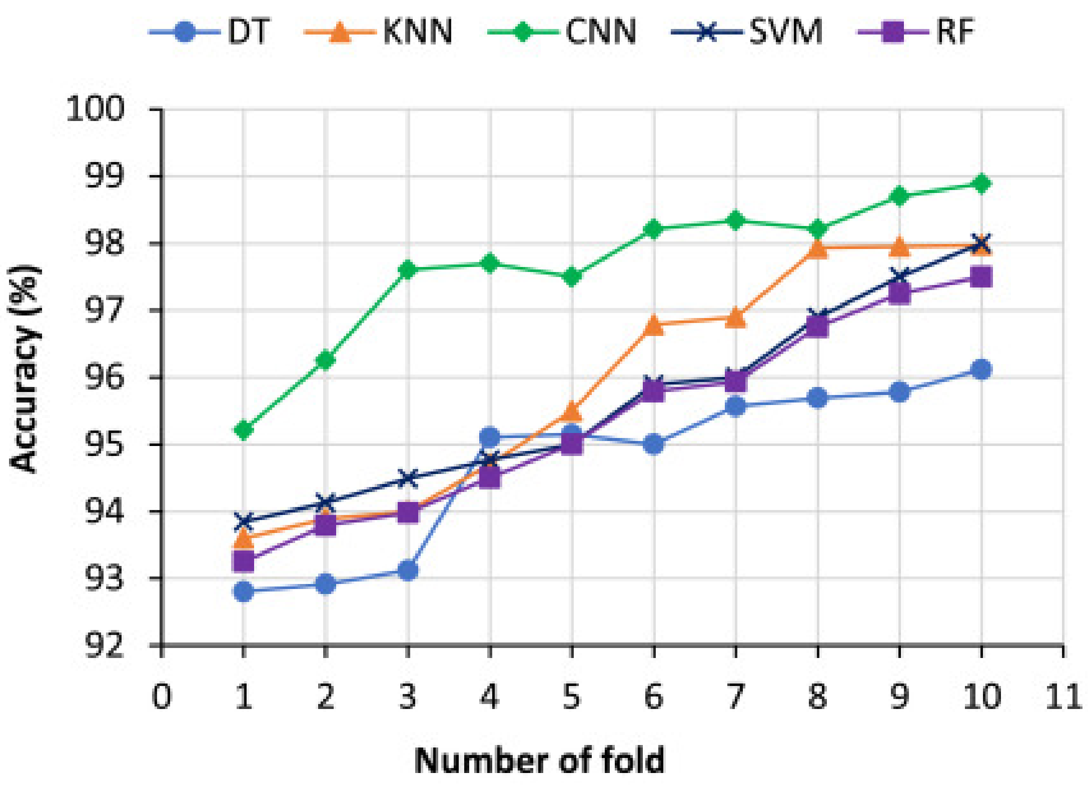


Figure 6.1: Accuracy chart comparing CNN vs other ML algorithms

A crucial application of deep learning in the area of computer vision is the real-time detection of weapons utilizing Convolutional Neural Networks (CNNs). A specific type of neural network called CNN is made to analyze visual data, such as pictures and videos.

Collecting a large dataset of photos that include both weapons is the first step in implementing real-time weapon detection using CNNs. The size of weapons, as well as other characteristics, may be recognized by the Network using this dataset. After gathering the data, we utilized it to train a CNN. The CNN gains knowledge of the characteristics that are crucial for telling apart weapons from non-weapons during the training procedure. New photos can then be categorized as either including a weapon or not using the trained CNN. Usually, the CNN is integrated with a video feed to implement real-time detection. The CNN scans each frame of the video feed as it is being processed to check for the presence of weapons. An alarm that notifies security staff of the discovery of a weapon can be set off. It's critical to remember that employing CNNs to detect weapons in real-time is still a developing technology and that it has some limitations in terms of accuracy and efficacy. The system's performance may be impacted by elements including illumination, camera angles, and occlusion. As a result, it's crucial to thoroughly assess the system's performance in various settings to make sure it is reliable and efficient.

Real-time weapon identification using CNNs can be implemented utilizing the powerful TensorFlow Object Detection API. This API offers pre-trained CNN-based object identification models, like Faster R-CNN and SSD, that can be adjusted for the task of detecting weapons. The first step in using the TensorFlow Object Detection API for weapon detection is to build the dataset of images with weapons and labeled bounding boxes. The pre-trained object detection model for the goal of detecting weapons can then be adjusted using this dataset. The model gains the ability to identify the distinctive characteristics of weapons, such as their shape, during the fine-tuning phase. The model can be used to spot weapons in real-time video streams once it has been adjusted. The CNN analyzes the video frames and creates bounding boxes around any recognized weapons. These bounding boxes can be used to send out an alarm or monitor how the weapon is moving.

Overall, CNN-based real-time weapon identification may be implemented using the TensorFlow Object Detection API, which is a strong tool. To make sure the model is accurate and useful in real-world circumstances, it is essential to carefully assess how well it performs.

6.3.1 Criteria fulfilled

- Security and safety.
- No direct physical contact.
- Fast response.
- Precision and recall..
- Wide range area detection.

Chapter 7

Description of the work

The work for a machine learning project that uses cameras to identify persons carrying firearms and weapons would involve several steps:

7.1 Completed work

- **Data collection:** compiling a sizable dataset of photographs and videos depicting individuals using firearms and other weapons, as well as a dataset of individuals not wielding such items. This phase will include proposing our project to the board for our AUD Capstone project.
- **Data Preprocessing:** Preparing the data for the machine learning model by cleaning and structuring it. [6]
- **Feature Extraction:** Identifying and removing crucial details from photos and videos that can be used to tell who is carrying a gun or other weapon from who is not is a process known as feature extraction. [6]
- **Model training:** Using the preprocessed data and extracted characteristics to train a machine learning model. [7]

7.2 What's next

- **Model Evaluation:** Testing the trained model on a different dataset in order to assess its performance. This phase will include reconfiguring the project according to the design evaluation and presenting the project to the board as part of the midterm of the AUD Capstone project. [6]
- **Model deployment:** Including the tested model in a system that uses camera input to instantly identify people carrying firearms and other weapons. This phase will include presenting our final project to the board for our AUD Capstone project.
- **Project maintenance:** Continuously monitoring and enhancing the model to make sure that its performance and accuracy remain constant throughout time.

Chapter 8

Project deliverables

The project deliverables for a machine learning project that uses cameras to identify persons carrying weapons would typically include the following:

- A thorough project plan defining the project's goals, timeframes, objectives, and deliverables.
- A group of pictures and videos showing people carrying weapons and firearms as well as a set showing people without weapons or firearms.
- A computer learning model that has been taught, examined, and improved to recognize people who are carrying guns and firearms.
- A report outlining the model's performance, including recall, accuracy, and other pertinent metrics.
- A system for real-time model deployment that may involve hardware, software, and system integration and method of incorporating a trained model into a camera-based system for instant identification.
- A guide or paperwork that explains how to operate the system and evaluate the outcomes.
- A strategy for the system's continued improvement and upkeep, including a timetable for updating the model when new data become available along with a report outlining the procedures followed, the outcomes discovered, and the model's performance.

- A presentation that highlights the project's findings.
- The system's code, as well as any required libraries, programs, or models.
- A strategy for upkeep and upgrades, including how to keep enhancing the model's performance over time.
- Depending on the particulars of the project and the client's requirements, the deliverables may change.

Chapter 9

ChatGPT comparison

Here we are asking ChatGPT to analyze a live stream RTMP link video feed for facial recognition using TensorFlow 2

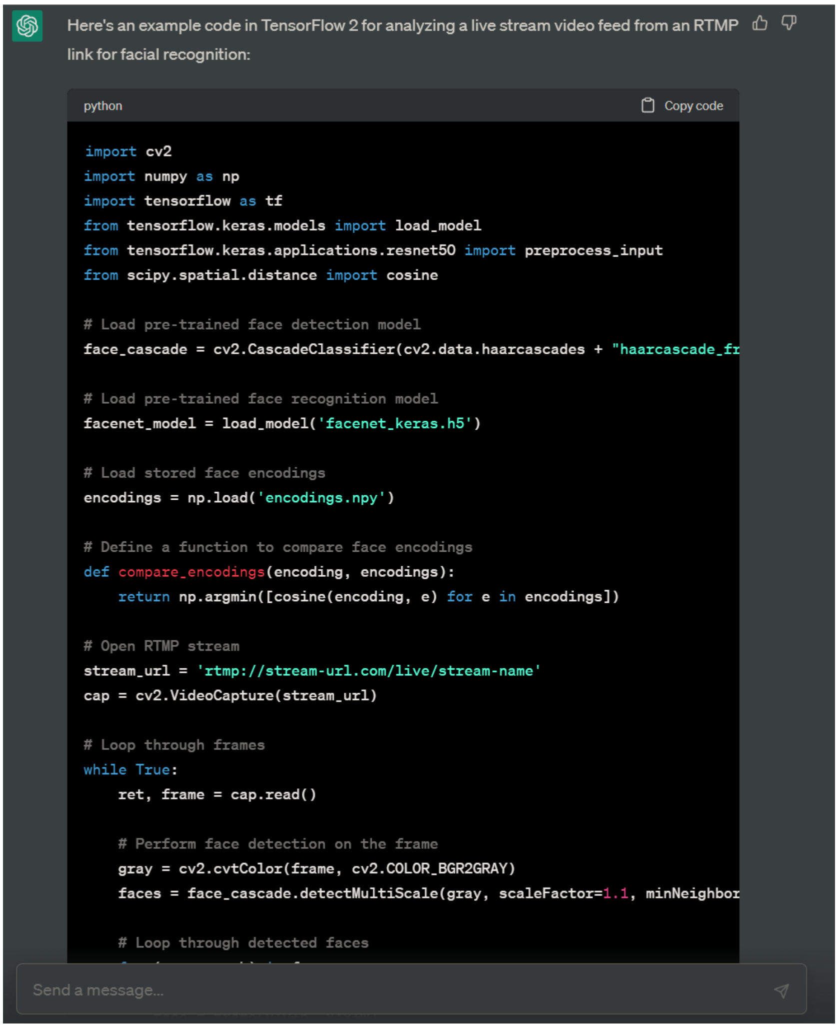
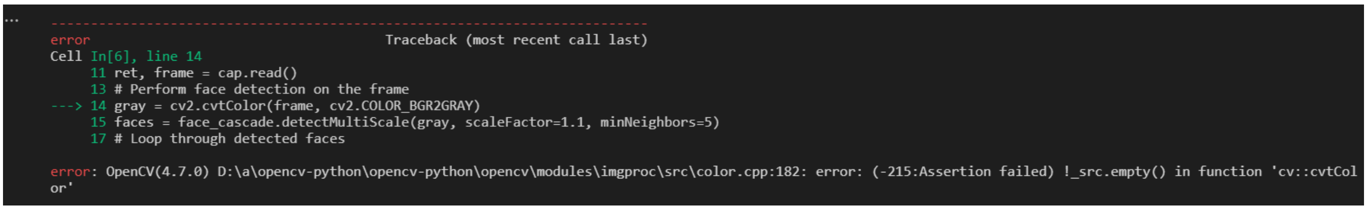


Figure 9.1: Prompting ChatGPT

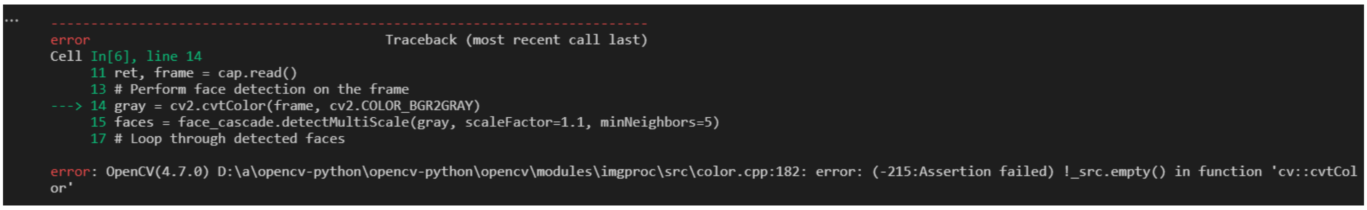
ChatGPT was able to create code that would give the programmer a head start, but not good enough to entirely replace the programmer as the code had errors. For example, it often had made-up libraries or wrote code with missing dependencies, faulty function arguments, considerable plagiarism, etc. Below are a couple of the errors faced in compiling to demonstrate.



```
...
    ---------------------------------------------------------------------------
    error                                     Traceback (most recent call last)
    Cell In[6], line 14
         11 ret, frame = cap.read()
         13 # Perform face detection on the frame
    ---> 14 gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
         15 faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5)
         17 # Loop through detected faces

    error: OpenCV(4.7.0) D:\a\opencv-python\opencv-python\opencv\modules\imgproc\src\color.cpp:182: error: (-215:Assertion failed) !_src.empty() in function 'cv::cvtCol
    or'
```

Figure 9.2: Error: faulty function arguments



```
...
    ---------------------------------------------------------------------------
    error                                     Traceback (most recent call last)
    Cell In[6], line 14
         11 ret, frame = cap.read()
         13 # Perform face detection on the frame
    ---> 14 gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
         15 faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5)
         17 # Loop through detected faces

    error: OpenCV(4.7.0) D:\a\opencv-python\opencv-python\opencv\modules\imgproc\src\color.cpp:182: error: (-215:Assertion failed) !_src.empty() in function 'cv::cvtCol
    or'
```

Figure 9.3: Error: library error

ChatGPT chat bot is a great tool to assist an expert programmer on their project. With the right back and forth prompts utilizing the programmers knowledge, ChatGPT can help in the production of the project considerably. It is important to note that ChatGPT-3.5 most recent training data is from Sept. 2021, in other words it is outdated. Additionally, we are limited in the prompt input length and output length. Therefore, for long code, we can only input code in segments, increasing the chances for error, especially considering the faulty memory and recall abilities of this version of ChatGPT model.

Chapter 10

Future Expansion

We have major future expansions for our weapon detection project, mainly within two goals, AUD Capstone and WDS project as a whole.

10.1 AUD Capstone future expansion

For our AUD senior design capstone, our main goals are to implement our weapon detection system in a drone and present our final presentation of the project to the jury.

10.2 WDS project future expansion

For the weapon detection system as a whole, we would like to tune and optimize our algorithm further, network numerous drones to work together at the scene, path plan the drones to automate their flying paths, identify security personnel and differentiate them from criminals and finally detect concealed weapons.

We also plan on proposing our idea to investors or a Dubai government entity as a business startup idea.

Chapter 11

Project management issues

Project management for a machine learning project that uses cameras to identify persons carrying firearms and weapons would involve addressing several key issues related to personnel, resources, budget, tasks, and schedule.

11.1 Personnel

- Locating and enlisting the required team members with the required proficiency in project management, computer vision, and machine learning.
- Finding and putting together a team of people with the knowledge and experience required to finish the project, such as software developers, computer vision experts, and machine learning experts.
- Giving team members jobs and responsibilities will help to ensure that all tasks are finished quickly and effectively.

11.2 Resources

- Finding and procuring the resources, such as hardware and software, required to finish the project.
- Finding and putting together a team of people with the knowledge and experience required to finish the project, such as software developers, computer vision experts,

and machine learning experts.

- Giving team members jobs and responsibilities will help to ensure that all tasks are finished quickly and effectively.

11.3 Budget

- Setting a budget for the project and keeping an eye on spending to make sure it stays within it.
- Calculating the project's cost and developing a budget that accounts for all relevant costs, such as those related to staff, equipment, and travel.
- Keeping an eye on the project's development to make sure the budget is not exceeded.

11.4 Tasks

- Breaking the project down into smaller tasks and distributing them to team members are tasks. creating a project plan that details the project's goals, deadlines, and tasks.
- Recognizing and defining each work that must be carried out as a part of the project.
- Assembling together a project schedule that details the tasks, dependencies, and due dates.
- Monitoring the process and making sure that every work is finished quickly and well.

11.5 Schedule

- Creating a timeline for the project and tracking its progress will help to guarantee that it stays on course and within budget.
- Putting together a thorough project schedule that lists all of the tasks, objectives, and due dates.

- Keeping an eye on the project's development to make sure it stays on track.
- Locating and resolving any delays or problems that might delay the project's completion.
- The project management should also take into account the moral and legal ramifications of gathering and using photos of people, as well as any potential social effects of the technology on things like civil liberties and privacy.

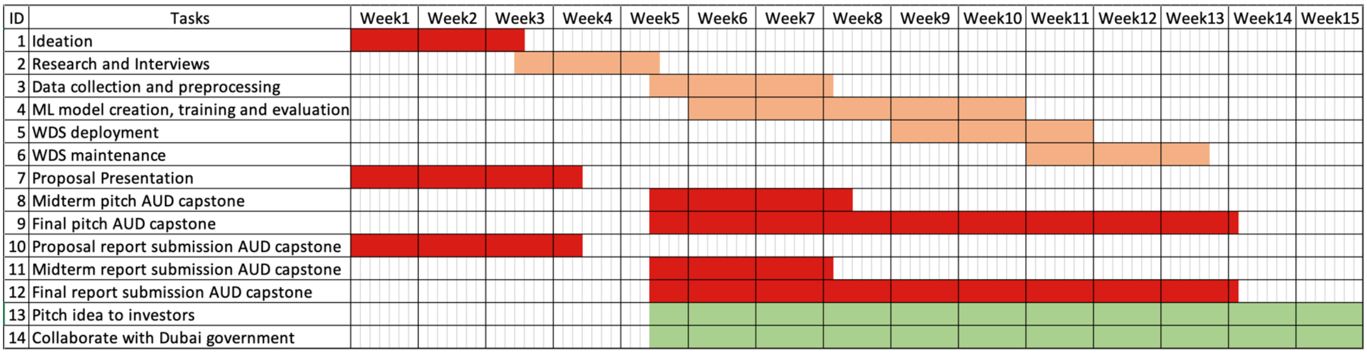


Figure 11.1: Gantt chart showing the schedule of each task

ID	Tasks	Start	End	Duration	Predecessor	Priority
1	Ideation	1/15/2023	2/2/2023	18		High
2	Research and Interviews	2/2/2023	2/16/2023	14		Medium
3	Data collection and preprocessing	2/16/2023	3/6/2023	18		Medium
4	ML model creation, training and evaluation	2/20/2023	3/26/2023	34		Medium
5	WDS deployment	3/13/2023	4/2/2023	20		Medium
6	WDS maintenance	3/27/2023	4/14/2023	18		Medium
7	Proposal Presentation	1/15/2023	2/8/2023	24		High
8	Midterm pitch AUD capstone	2/16/2023	3/8/2023	20	7	High
9	Final pitch AUD capstone	2/16/2023	4/17/2023	60	8	High
10	Proposal report submission AUD capstone	1/15/2023	2/8/2023	24		High
11	Midterm report submission AUD capstone	2/16/2023	3/6/2023	18	10	High
12	Final report submission AUD capstone	2/16/2023	4/17/2023	60	11	High
13	Pitch idea to investors	2/16/2023	4/30/2023	73		Low
14	Collaborate with Dubai government	2/16/2023	4/30/2023	73		Low

Figure 11.2: Gantt table showing the priority and duration of each task

11.6 Risk Management

Identifying potential risks to the project and taking steps to reduce such risks.

11.7 Communication

Establishing frequent team meetings and channels of communication to update everyone on the status of the project and to handle any issues that may occur.

11.8 Quality Assurance

Establishing a quality assurance procedure will help to verify the model is dependable, accurate, and robust.

11.9 Ethical and Professional Responsibility

- Making sure that any rules governing the use of technology, such as privacy and data protection legislation, are followed.
- The project management should also take into account the moral and legal ramifications of gathering and using photos of people,as well as any potential social effects of the technology on things like civil liberties and privacy. They should also have a strategy for handling and protecting the data, as well as a method for evaluating how the model's choice will affect society.

  The project management should take into account the moral and legal ramifications of gathering and using photos of people, as well as any potential social effects of the technology on things like civil liberties and privacy. They should also have a strategy for handling and protecting the data, as well as a method for evaluating how the model's choice will affect society.

  The developers and users of the weapon detection system have a professional responsibility to ensure that their technology is designed and used in a way that aligns with the code of ethics. The code of ethics for weapon detection systems is a set of principles and guidelines that informs the design, development, and use of such systems ethically and responsibly. This code is necessary because weapon detection systems, such as those using drones, have the potential to affect the lives and rights of individuals and communities. This code includes principles such as prioritizing respect for human rights, transparency, accountability, avoidance of harm and discrimination, ethical use, data protection, fairness and accuracy, and continuous evaluation and improvement. Additionally, professionals working with weapon detection systems should maintain a commitment to professional conduct, including honesty, integrity, and objectivity. They should also prioritize the safety and security of individuals and communities, and work to uphold legal and regulatory requirements. By adhering to a code of ethics and professional responsibilities, weapon detection system professionals can demonstrate their commitment to upholding ethical standards and promoting the well-being of society.

Chapter 12

Data set

Our data set is from the Andalusian Research Institute in Data Science and Computer Intelligence [10]. The weapon detection data set is for hand held guns and knives. We focused on the object detection subset of the data base rather than the object classification section. The latter, classification, is an AI task where the input is a clean image with only one object and the output whole image is classified as a specific object. The former, object detection, handles real world environments with busy backgrounds and many objects in frame. Detection identities if the target objects are in the image and their location in the image. We have 3000 images for pistol detection and 2078 for knife detection.

Thus, our data set includes images with pistols and knifes in many different environments and conditions with corresponding annotations of their location on the images. The location annotations are stored in XML flies with the same name as their corresponding image, such as the one below. The region of pixels which holds the target object, and its class(weapon type) is clearly labeled in the file.


```
<?xml version="1.0" ?>
<annotation>
	<folder>Reflex</folder>
	<filename>DSC_0001.JPG</filename>
	<path>../JPEGImages/DSC_0001.JPG</path>
	<source>
		<database>Unknown</database>
	</source>
	<size>
		<width>1920</width>
		<height>1280</height>
		<depth>3</depth>
	</size>
	<segmented>0</segmented>
	<object>
		<name>knife</name>
		<pose>Unspecified</pose>
		<truncated>0</truncated>
		<difficult>0</difficult>
		<bndbox>
			<xmin>873</xmin>
			<ymin>651</ymin>
			<xmax>1079</xmax>
			<ymax>758</ymax>
		</bndbox>
	</object>
</annotation>
```


Figure 12.1: Knife image and corresponding annotation

The images used were taken under a variety of environments, so our solution is more generalized. For instance, have images taken both indoors 12.2 and outdoors 12.3, thus our systems can be used in a variety of lighting conditions and background. There was especially focus on using CCTV footage 12.4 as this will be a common use case. We also have images where the weapons are partly covered by hands 12.5 to replicate their use in real life scenarios. Images also capture weapons at varying distances and angles 12.6, which is also helps generalize our solution. Furthermore, the data set does not only include one or a few models of weapons for each class. There is a wide variety of shape, colour, size and material 12.7 Thus, we are not training our model to detect a particular gun or knife but rather guns and knives in general.

Figure 12.2: Data set sample with weapon indoors

Figure 12.3: Data set sample with weapon outdoors

Figure 12.4: Data set sample with CCTV

Figure 12.5: Data set sample with covered weapon

Figure 12.6: Data set sample with differing angles

Figure 12.7: Data set sample with weapons of differing model

Chapter 13

Implementation and Data analysis

We have created 4 separate modules during our experimentation. Models 1,2 and 3 have been trained with the base model "ssd mobilenet v2 fpnlite 320x320" and batch sizes of 16. Model 4 used the base model "ssd mobilenet v2 fpnlite 640x640", which takes in more pixels for input images, and used a batch size of 8 due to computational limitations. Batch size is how many images are taken as input per step. Additionally, model 1 has had been trained on a 3/4 training data split. Model 2,3 and 4 have been trained on a 85/100 training data split. Model 1 trained for 6,000 training steeps, model 2 for 50,000 steps, model 3 for 200,000 steps and model 4 for 46,000 steps. Each training step is an update to the model weights.

Coco metrics were used to evaluate our models based on their test data set. Evaluation can be done on the final model to get its current metrics or can be run during training to see the metrics of the model over time (evaluations are done every 1000 training steps). Model 1 was on evaluated at the end of training. Model 2 and 3 were exported from the same training session and were evaluated starting at 14,000 steps. Model 4 was evaluated continuously.

Mean Average Precision (mAP) is metric used to evaluate object detection models. It's value can range from 0 to 1. As its names suggests is the mean of the Average Precision(AP) across all classes.

To further understand AP, we must consider the precision-recall curve . Precision is how many of our guesses are correct(True Positive/(True Positive+ False Positive) and recall is have we guessed every time we should have (True Positive/(True Positive+False negative). The precision recall curve is a plot which shows how recall changes with precision or vice versa. In the below figure 13.1 we can see how the area under curve (AUC) for precision-recall is similar to AP. AP is the weighted average of mean precision per increase in recall.

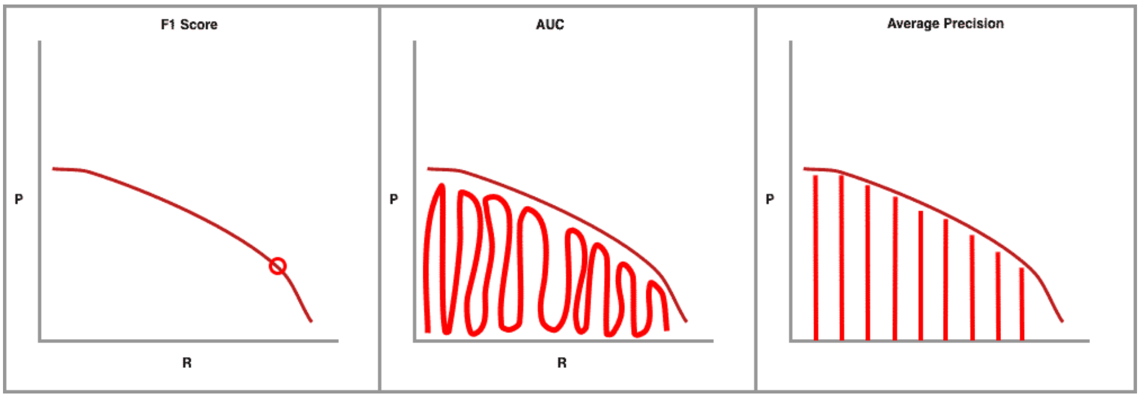


Figure 13.1: Relation of Precision-Recall cure to Average Precision

Intersection over Union(IoU) is a comparison of the model output bounding box to the test (ground truth) bounding box. As the name suggest the area of the intersect of these two these boxes is taken over their union - refer to below figure13.2. The threshold of IoU we consider for a correct detection can change the shape of the precision recall curve as seen in figure13.3.

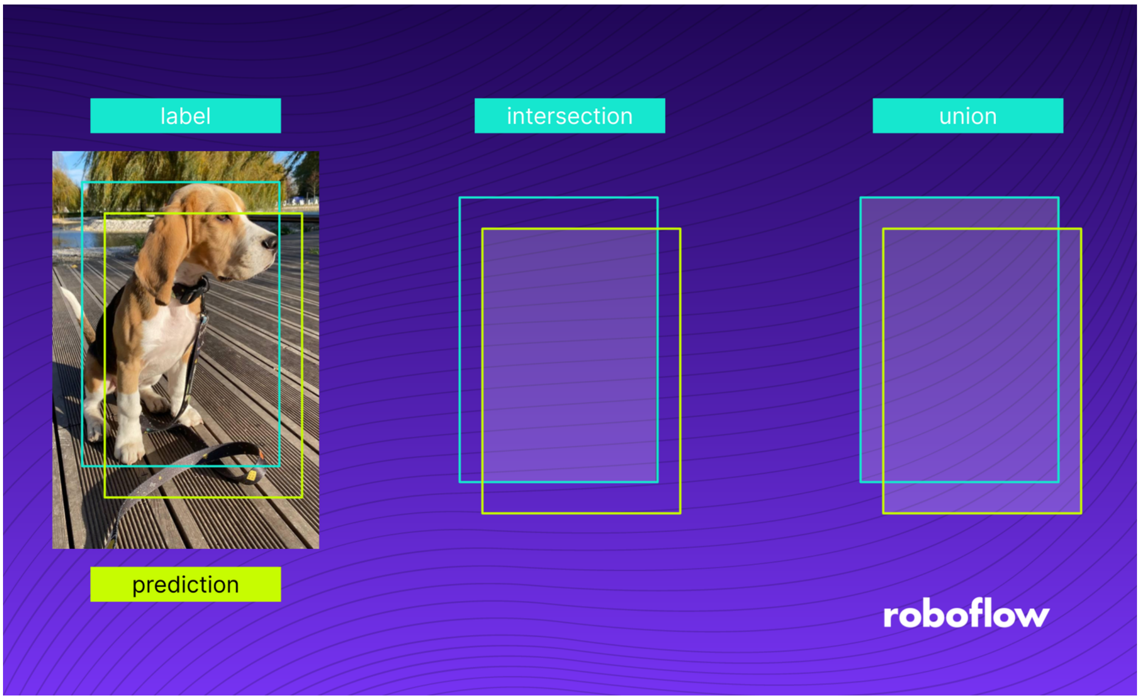


Figure 13.2: Intersection over Union(IoU)

The mAP score in COCO metrics takes the mean AP scores of precision-recall curves with IOU(Intersection over Union) thresholds from 0.50 to 0.95, with increases of 0.05, and then averages the results across all classes. [11]

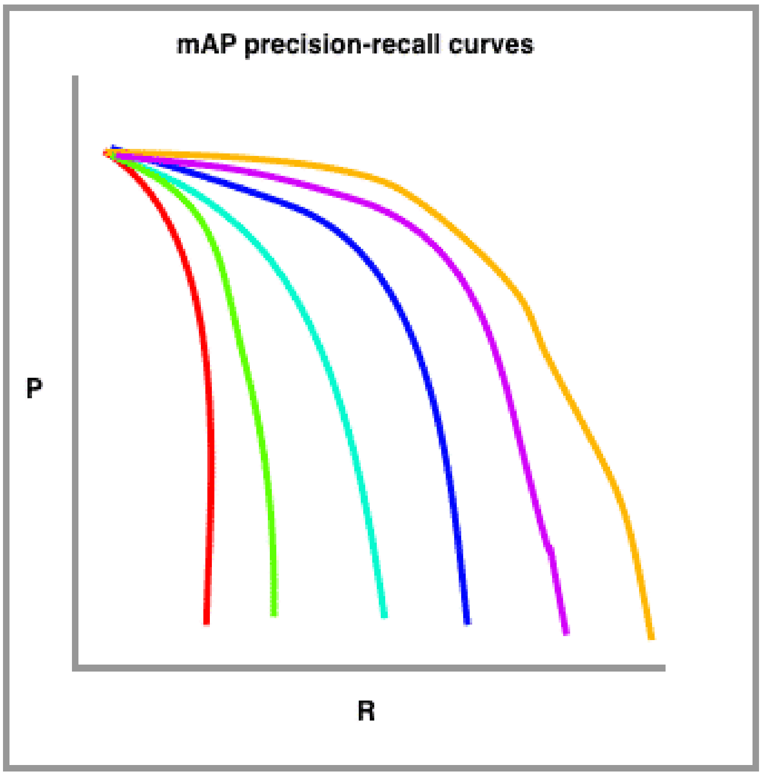


Figure 13.3: Person-Recall curves with different Intersection over Union(IoU) threshold

To discuss the mAP scores, model 113.5 is much lower at around than others 13.23 13.40 which are more similar. In all models the larger objects have the best mAP score(13.6,13.24, 13.41), medium sized objects slightly worse (13.7,13.25, 13.42) and small object are have the worst mAP score (13.8,13.26, 13.43). This is to be expected as there are less features to recognize as an object becomes smaller. Similarly the mAP at 0.5IoU (13.9,13.27, 13.44) are higher than at 0.75 IoU (13.10,13.28, 13.45). The high scores at 0.5 IoU suggest that the model is practically detecting the objects very well. We believe discrepancy may be do to how different researches have labeled the data - some have the hand include the hand in the bounding box and some haven't. For our task with small hand held objects of guns and knives half the object can be held in the hand. Thus we believe a threshold of 0.5 IoU is reasonable and therefore our models perform well.

For recall our models behaved as predicted. AR@X refers to the average recall at X detection per image. The higher the value of X the higher the score.For instance AR@100(13.13,13.31, 13.48) is better than AR@10(13.12,13.30, 13.47) which is still better than AR@1 (13.11,13.29, 13.46). Like with mAP larger objects (13.14,13.32, 13.49) are easier to recall than medium sized objects (13.15,13.33, 13.50), which are easier than small (13.16,13.34, 13.51).

Our loss functions (13.20,13.37, 13.55)reduced over time as expected showing us that our model is learning and converging on a "solution".

To visualize how a model has learned over time we can reference 13.57 where model 4 has been trained for 1,000 steps, 13.58 14,000 steps and 13.59 46,000. Even at 6,000 steep model 1 began correctly detecting objects 13.22.

Based on our analysis model 2 is the best choice for our task. For our task we are prioritizing computational speed over absolutely higher metrics. It is important to note that even when skipping every other frame we are still analyzing over 15 frames per second so there are many chances for our model to detect the weapon(s). Whats important is the our model can create an output in less than a 10th of second so that it can keep up in real time. That is why we are choosing model 2 as its has close enough scores to model 4 but would be nearly twice as fast and thus more compatible with more hardware. Model 1 has no computational advantage and by every metric is significantly worse. Model 3 is very similar in scores to model 2 and has risk of having been over trained.

The bellow graphs have been smoothed by 0.6. The graphs have an x-axis of training steps.

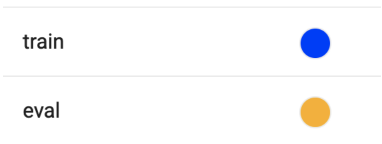


Figure 13.4: Legend of graphs

13.1 Model 1 graphs

As mentioned previously model 1 used the base model "ssd mobilenet v2 fpnlite 320x320", a batch size of 16, 75/100 of the data was split for training and it took 6,000 training steps.

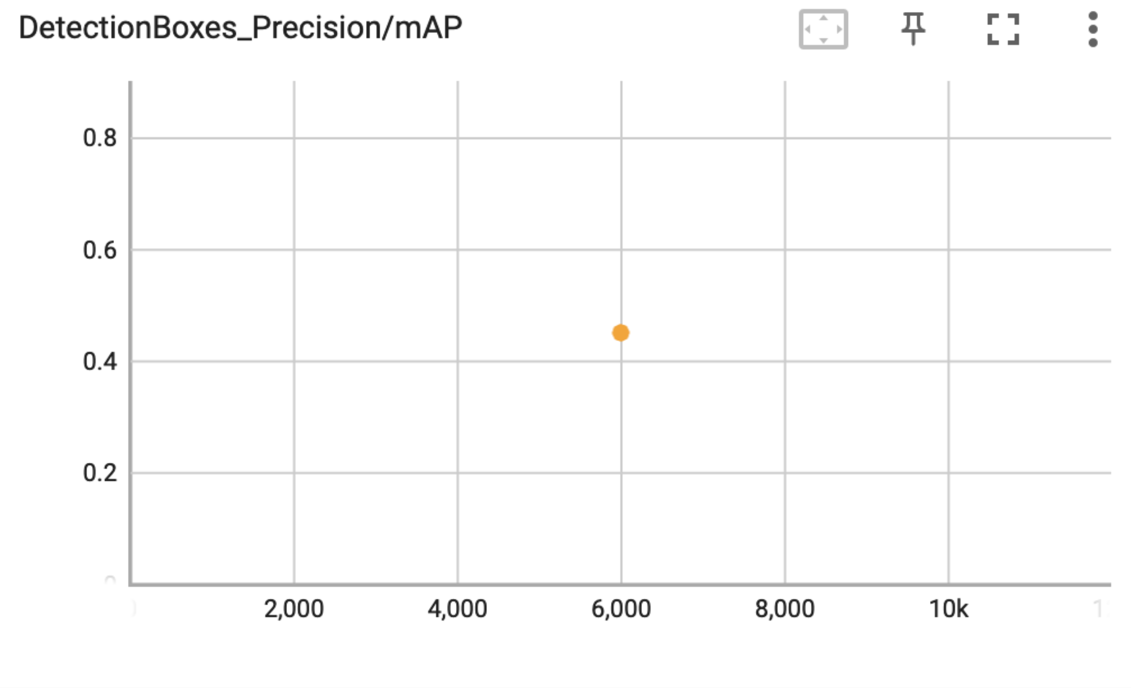


Figure 13.5: Model 1: mAP score (evaluated at end of training)

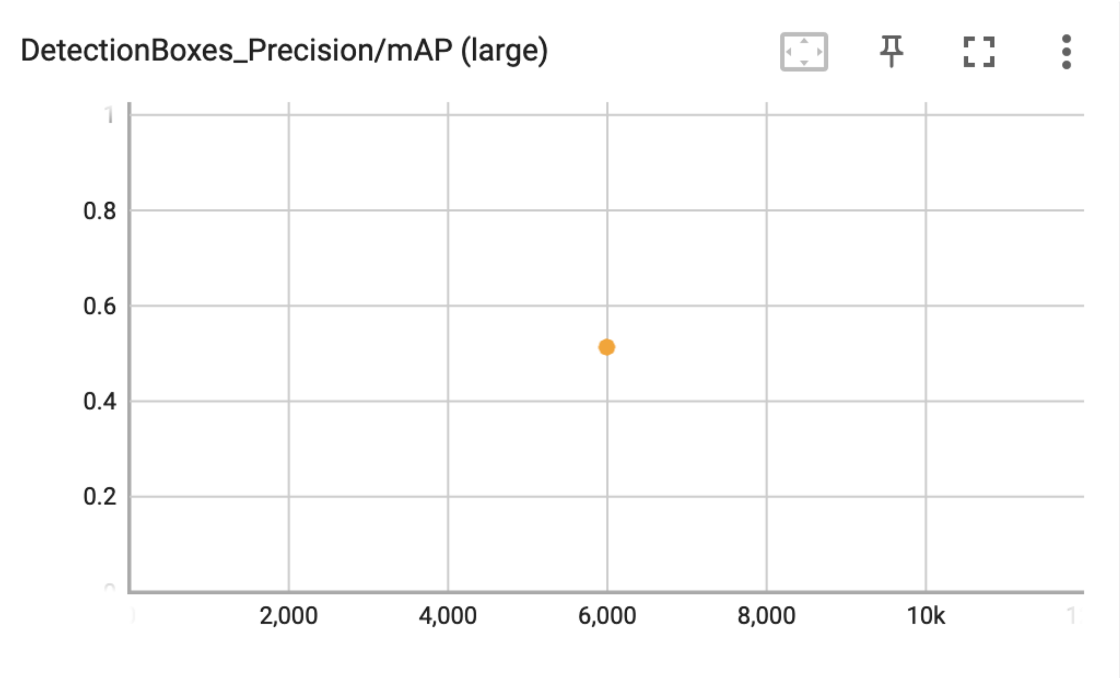


Figure 13.6: Model 1: mAP large score (evaluated at end of training)

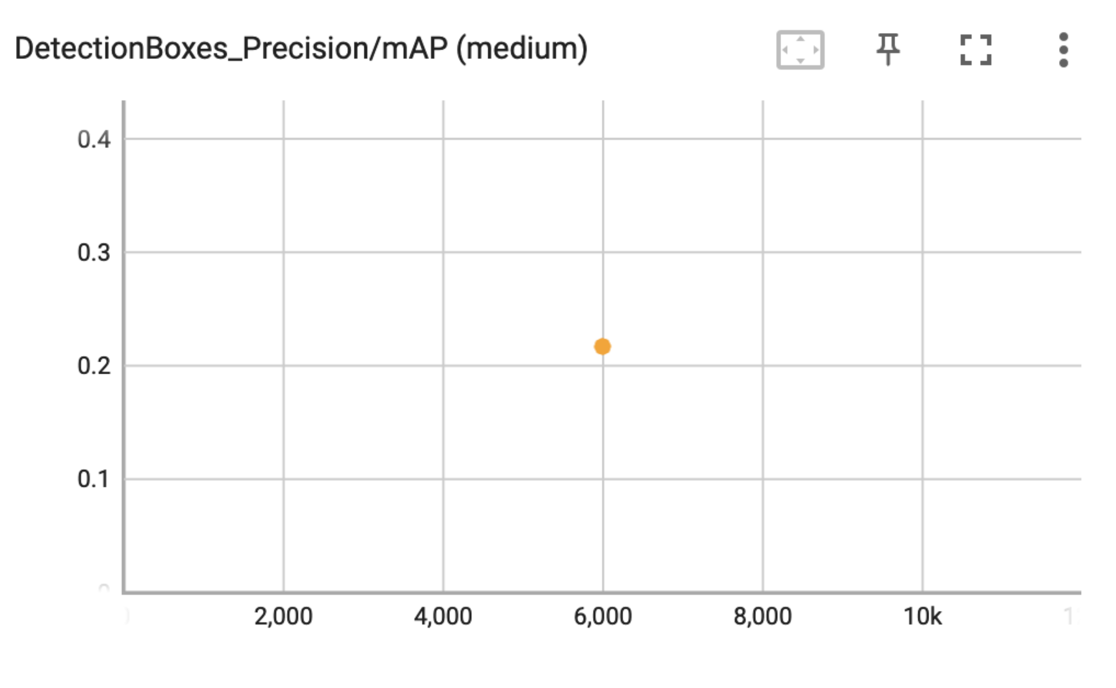


Figure 13.7: Model 1: mAP medium score (evaluated at end of training)

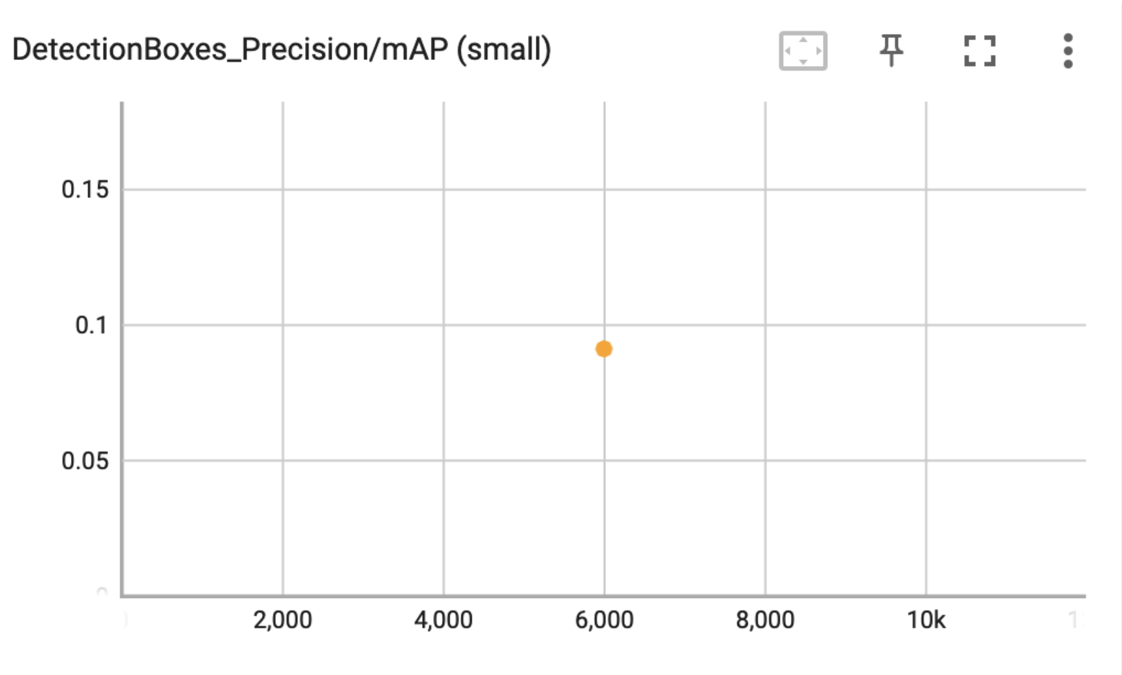


Figure 13.8: Model 1: mAP small score (evaluated at end of training)

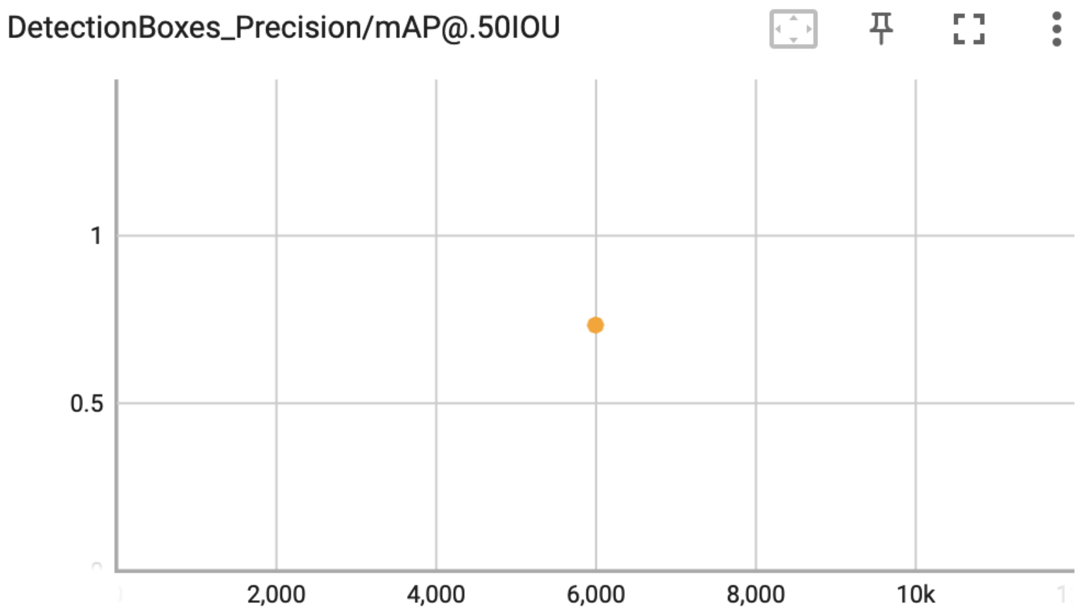


Figure 13.9: Model 1: mAP @.50IOU score (evaluated at end of training)

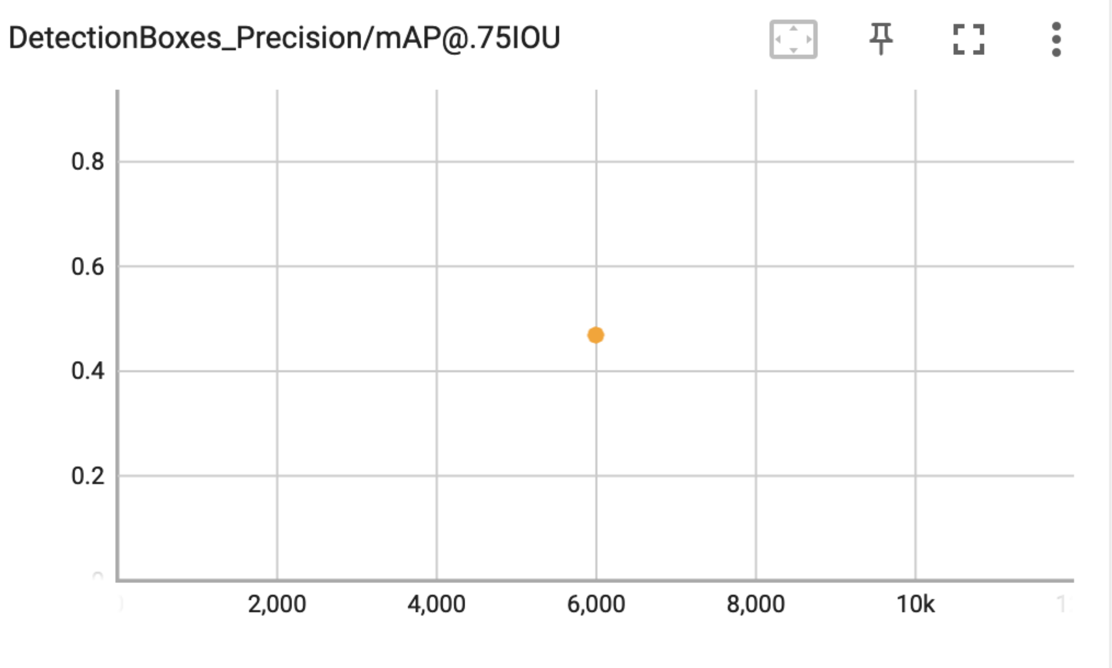


Figure 13.10: Model 1: mAP @.75IOU score (evaluated at end of training)

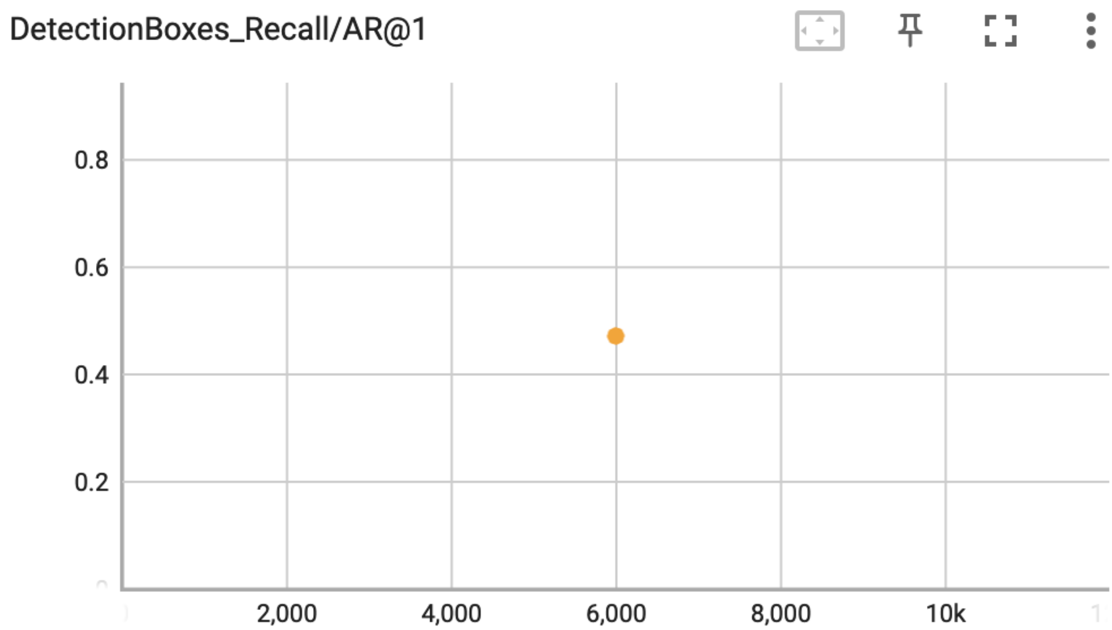


Figure 13.11: Model 1: recall AR@1 score (evaluated at end of training)

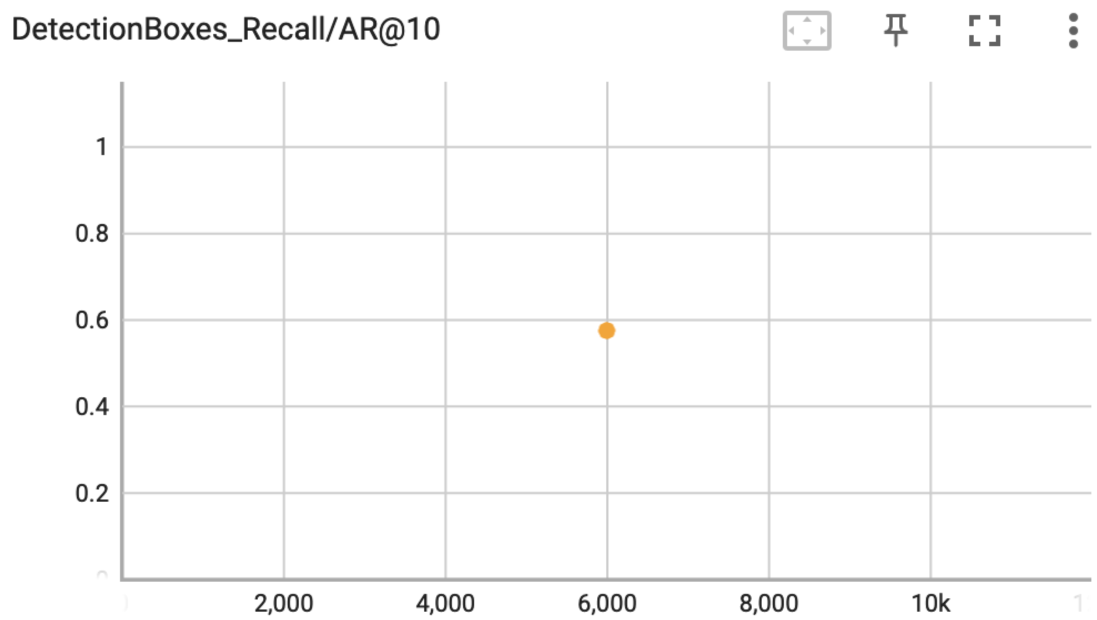


Figure 13.12: Model 1: recall AR@10 score (evaluated at end of training)

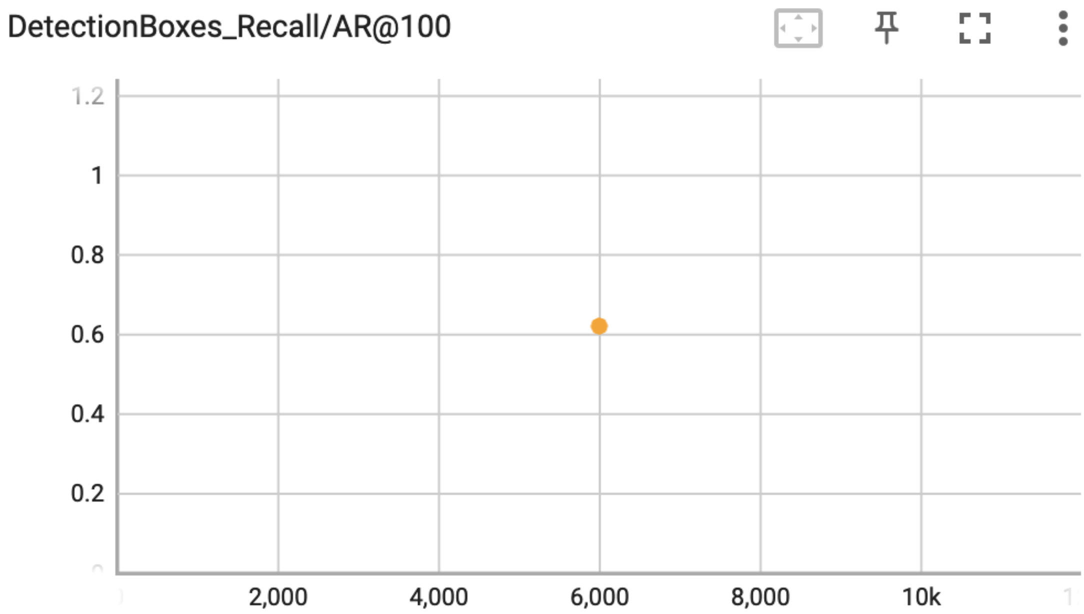


Figure 13.13: Model 1: recall AR@100 score (evaluated at end of training)

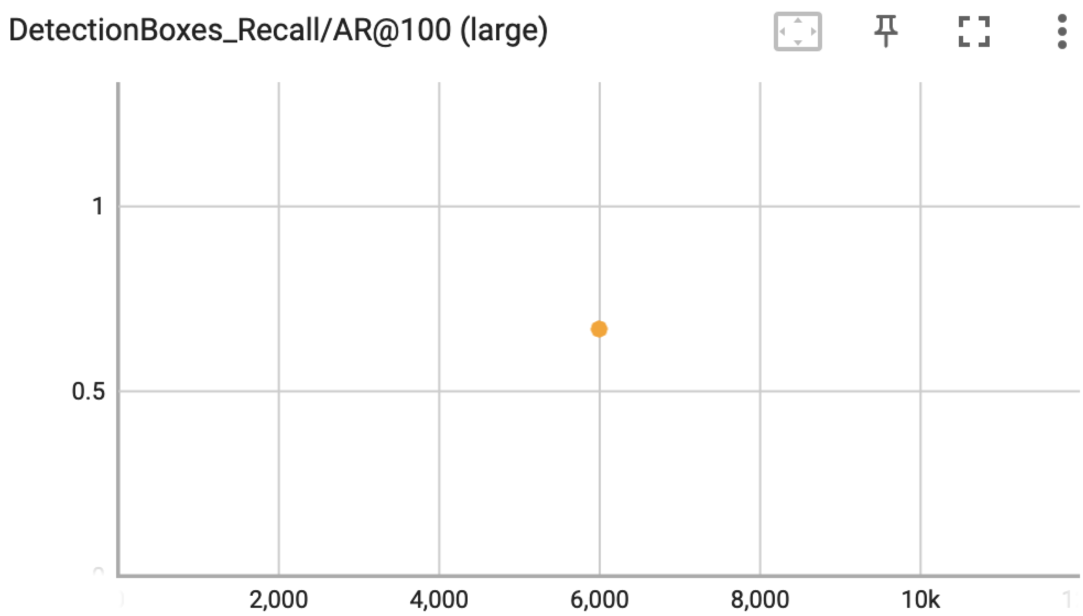


Figure 13.14: Model 1: recall AR@100 large score (evaluated at end of training)

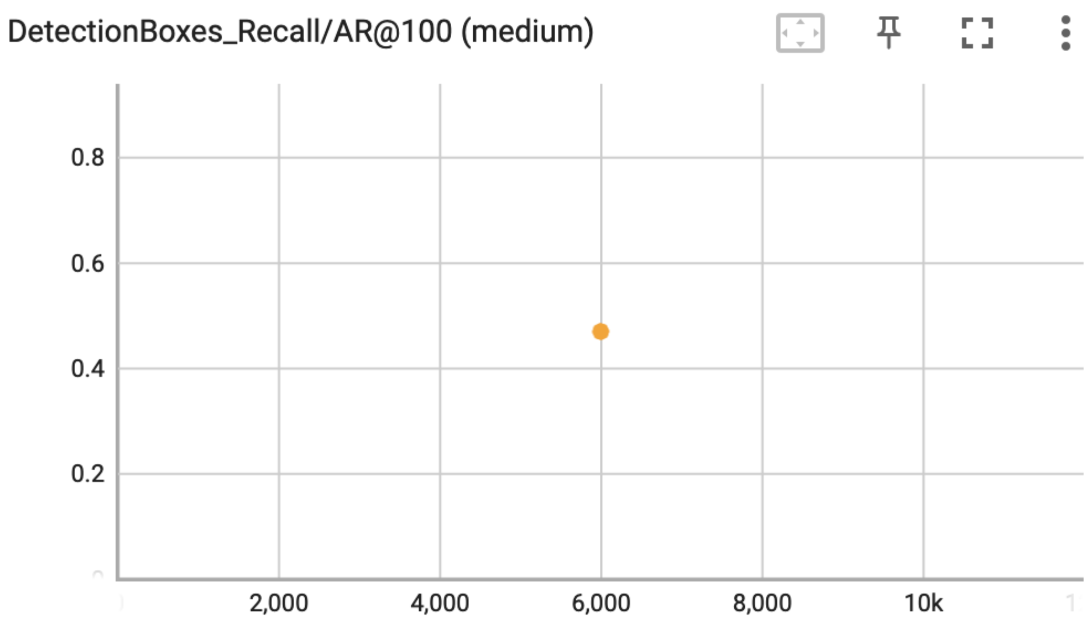


Figure 13.15: Model 1: recall AR@100 medium score (evaluated at end of training)

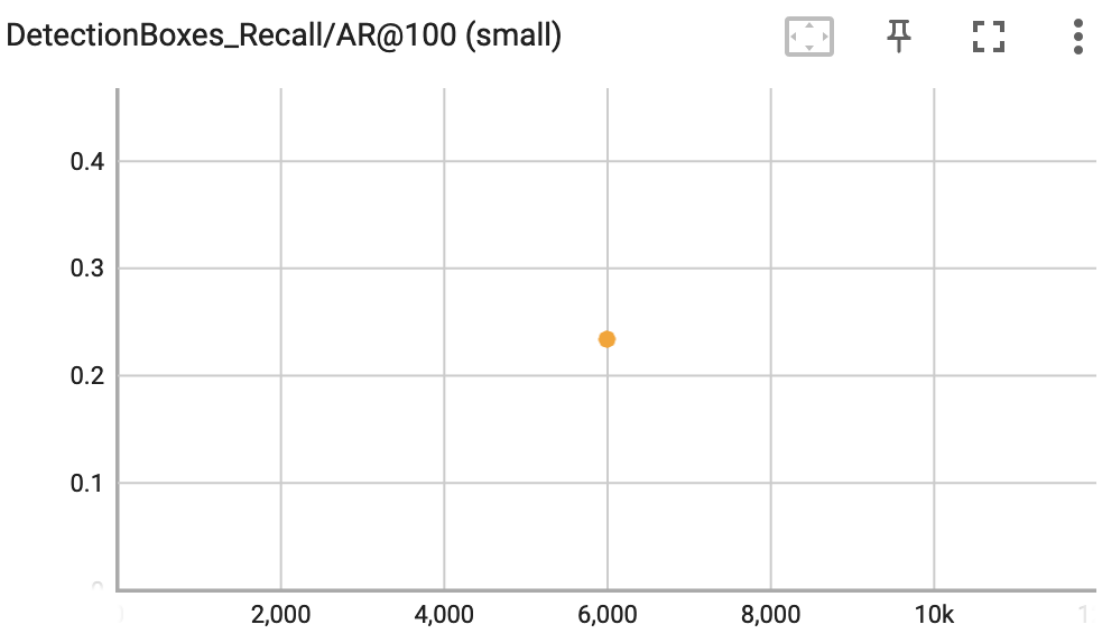


Figure 13.16: Model 1: recall AR@100 small score (evaluated at end of training)

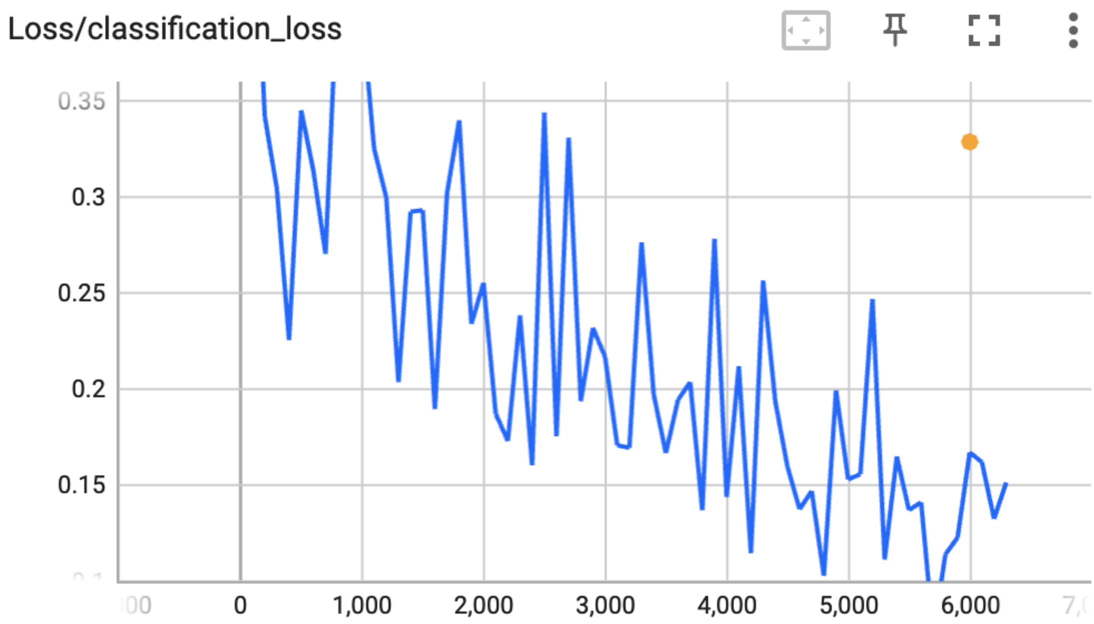


Figure 13.17: Model 1: classification loss (evaluated at end of training)

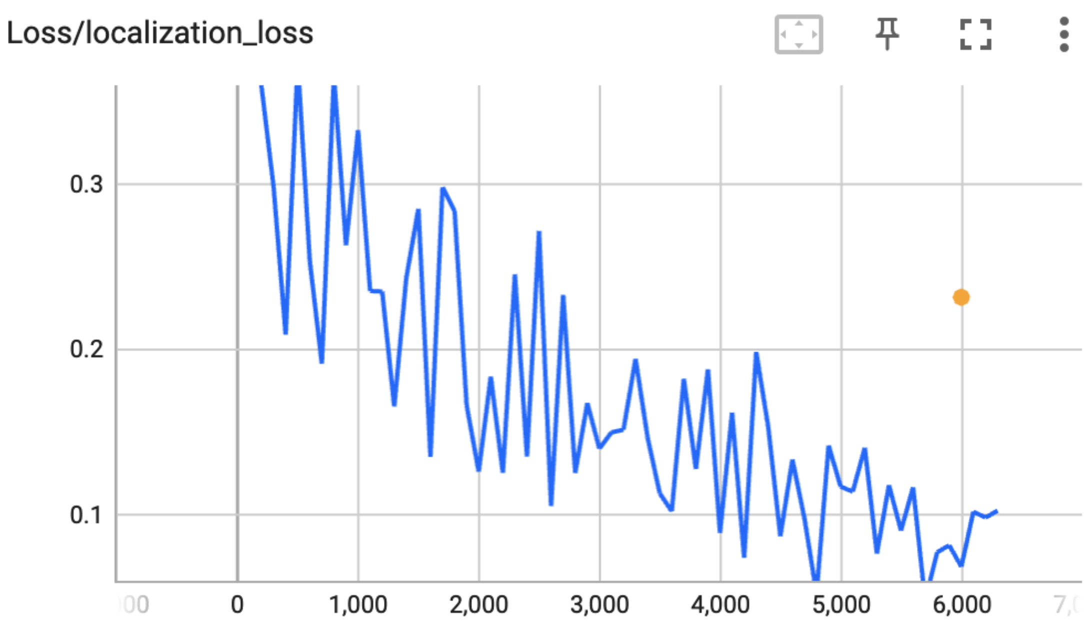


Figure 13.18: Model 1: localization loss (evaluated at end of training)

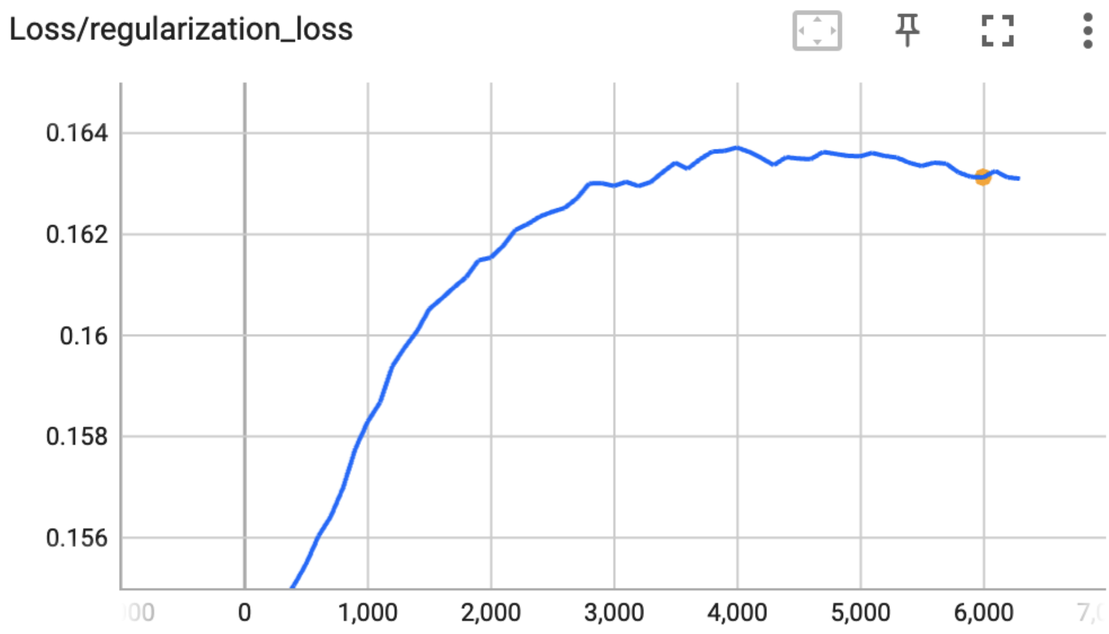


Figure 13.19: Model 1: regularization loss (evaluated at end of training)

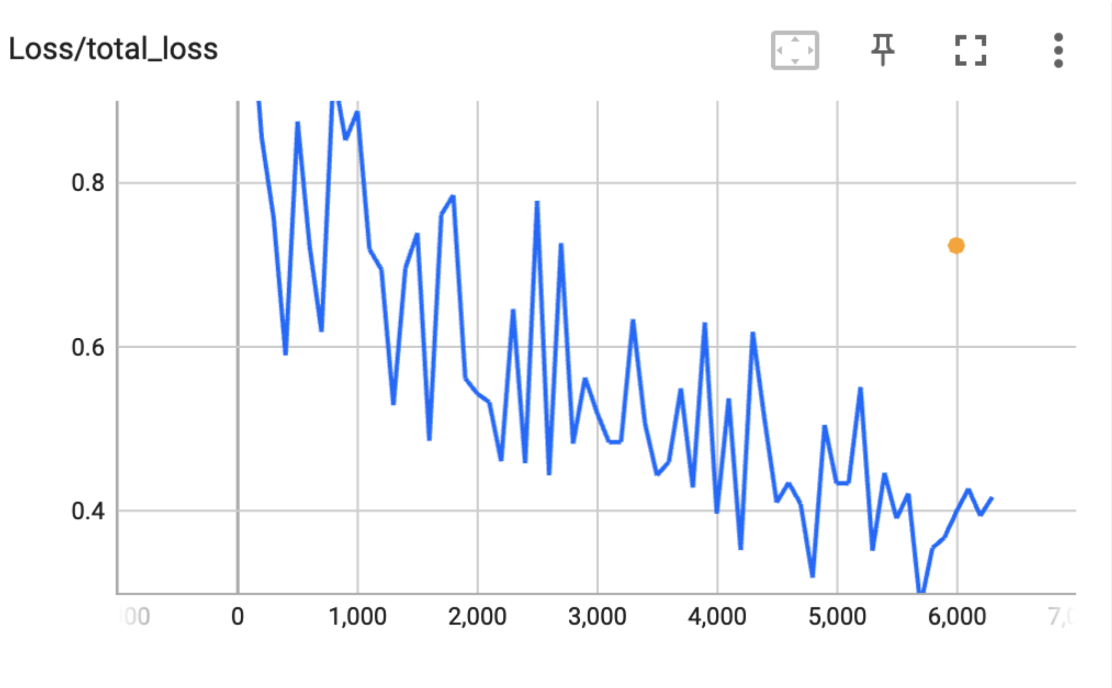


Figure 13.20: Model 1: total loss (evaluated at end of training)

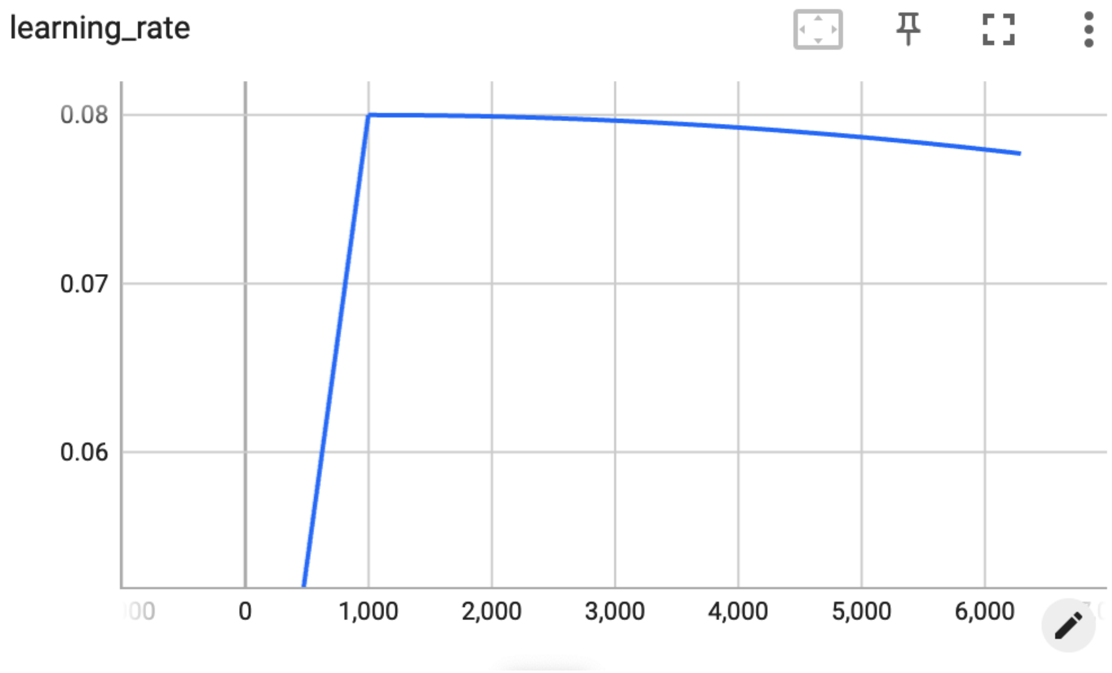


Figure 13.21: Model 1: learning rate

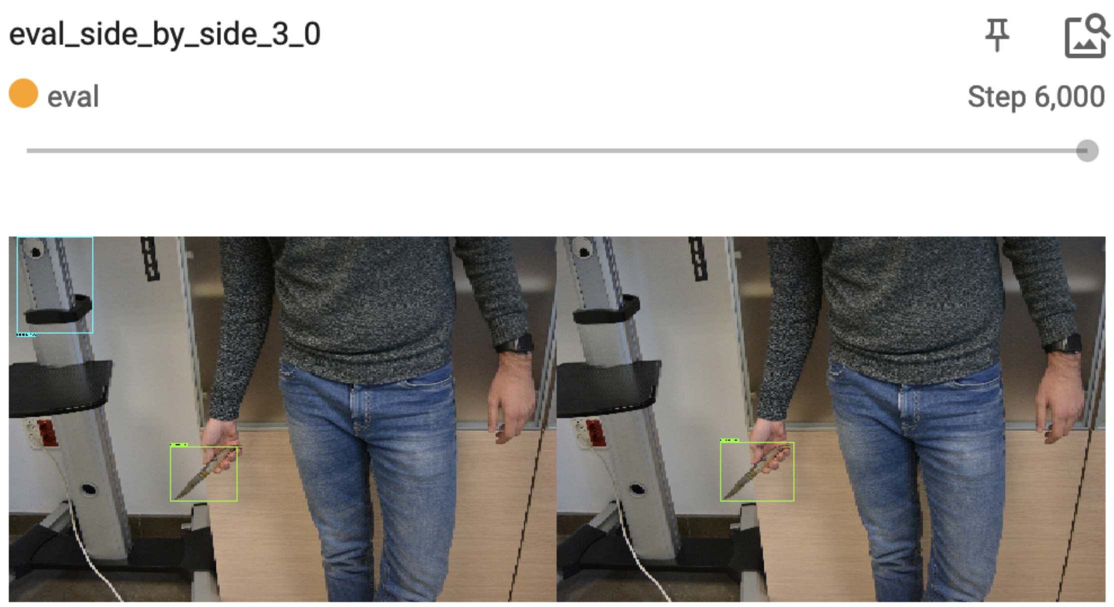


Figure 13.22: Model 1: Eval3-left is model output, right is test label

13.2 Model 2 and 3 graphs

As mentioned previously model 23 used the base model "ssd mobilenet v2 fpnlite 320x320", a batch size of 16 and 85/100 of the data was split for training. They were a part of the same training session and so share the same graphs. Model 2 was exported at 50,000 steps and model 3 at 200,000 steps.

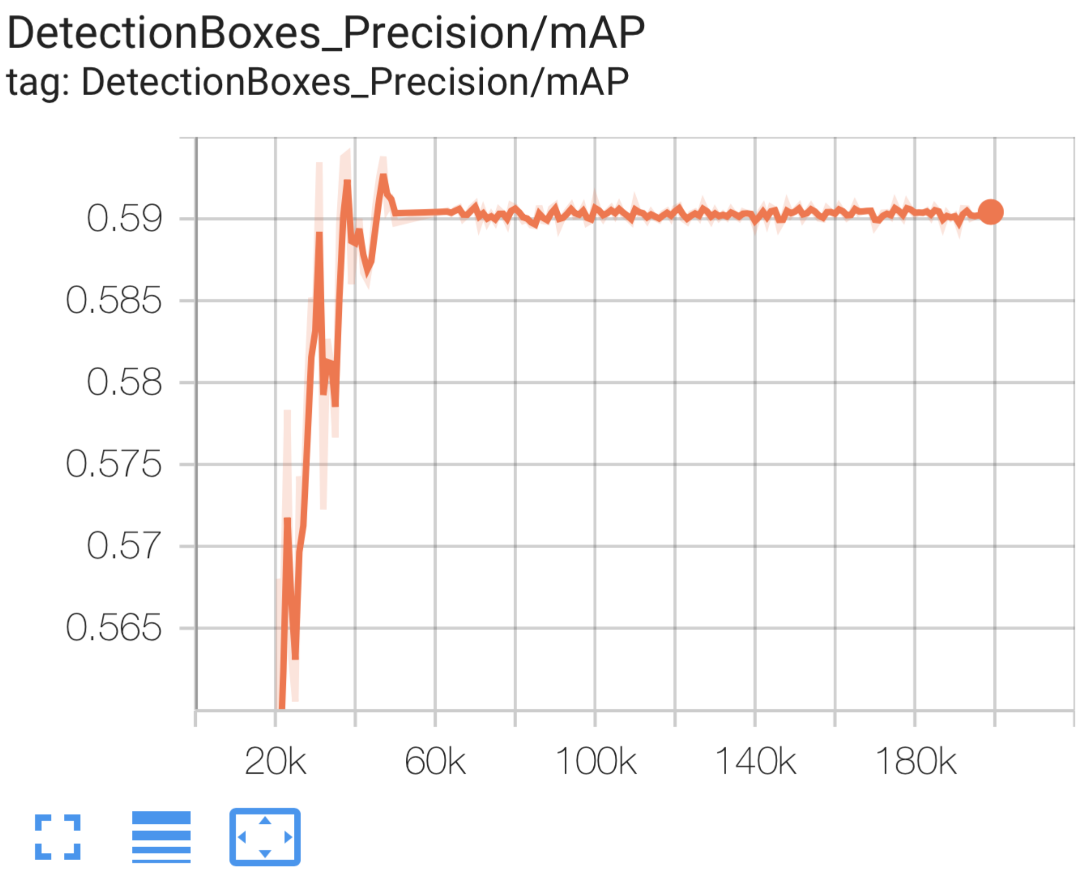


Figure 13.23: Model 2 and 3: mAP score (evaluation started at 14,000 steps)

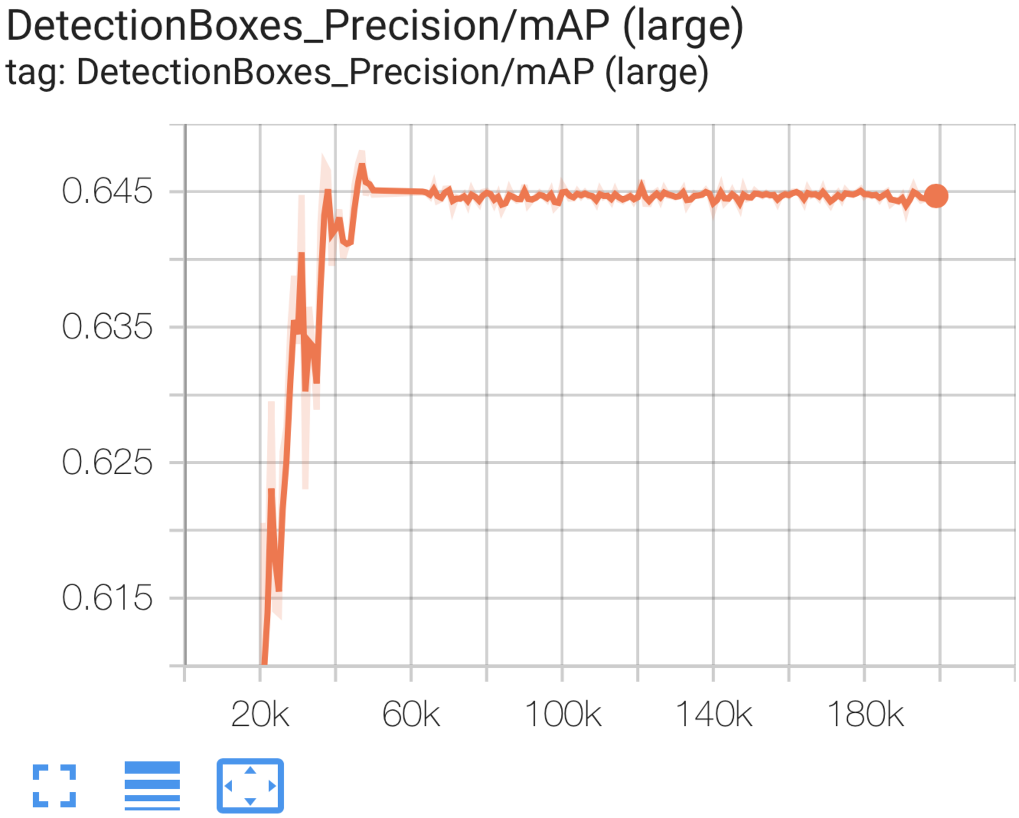


Figure 13.24: Model 2 and 3: mAP large score (evaluation started at 14,000 steps)

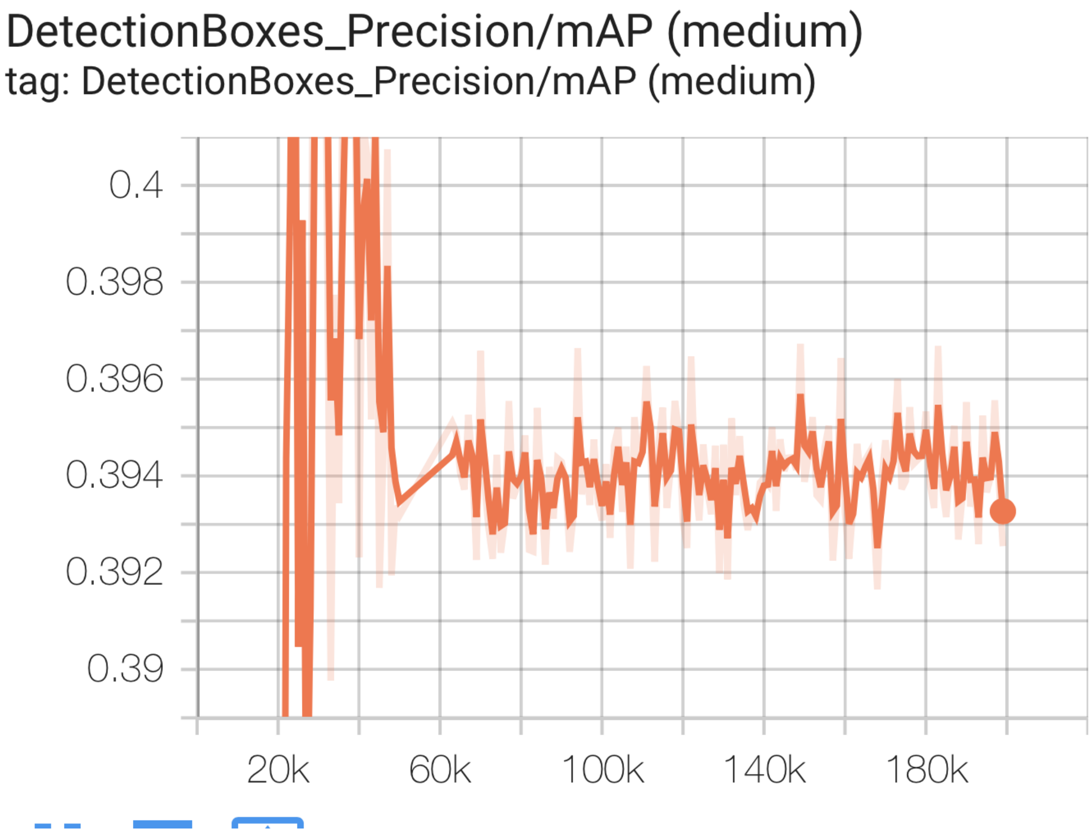


Figure 13.25: Model 2 and 3: mAP medium score (evaluation started at 14,000 steps)

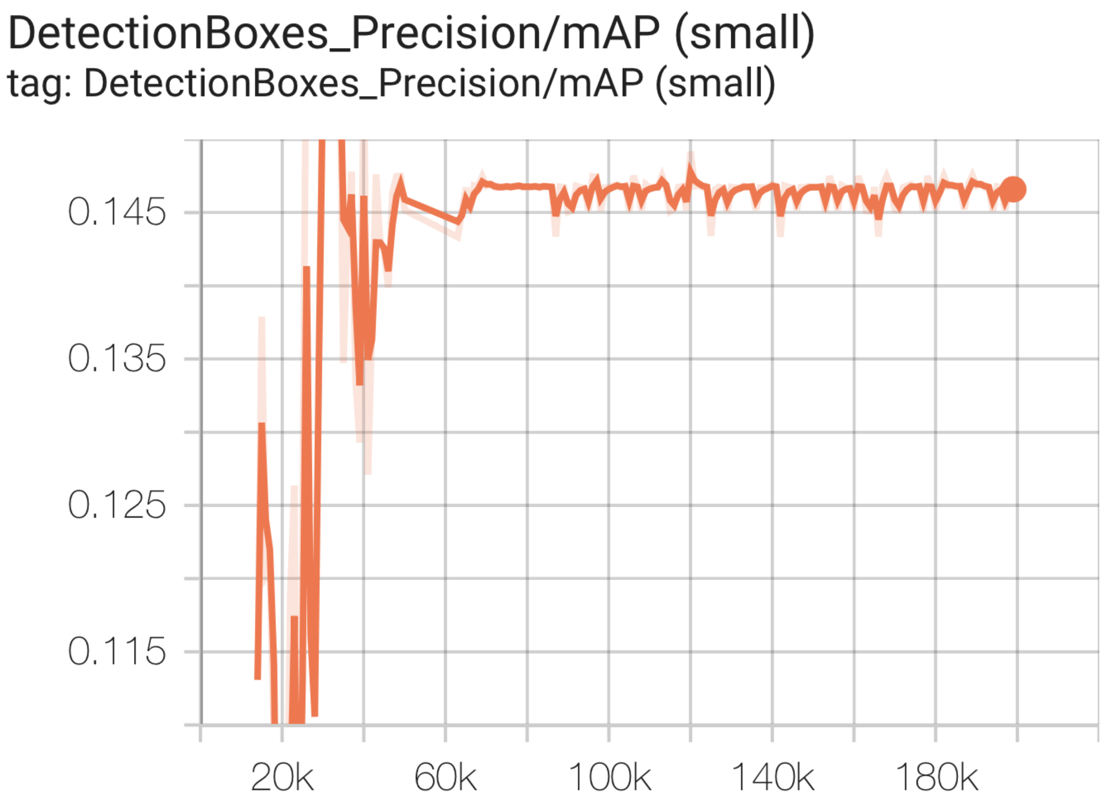


Figure 13.26: Model 2 and 3: mAP small score (evaluation started at 14,000 steps)

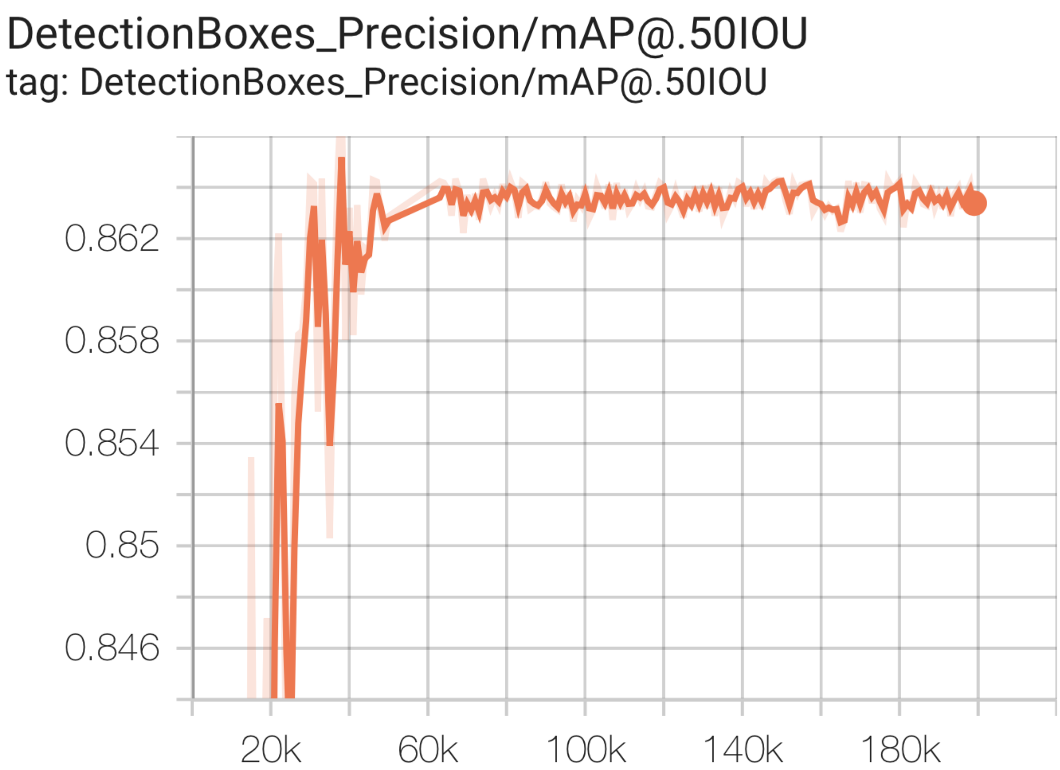


Figure 13.27: Model 2 and 3: mAP @.50IOU score (evaluation started at 14,000 steps)

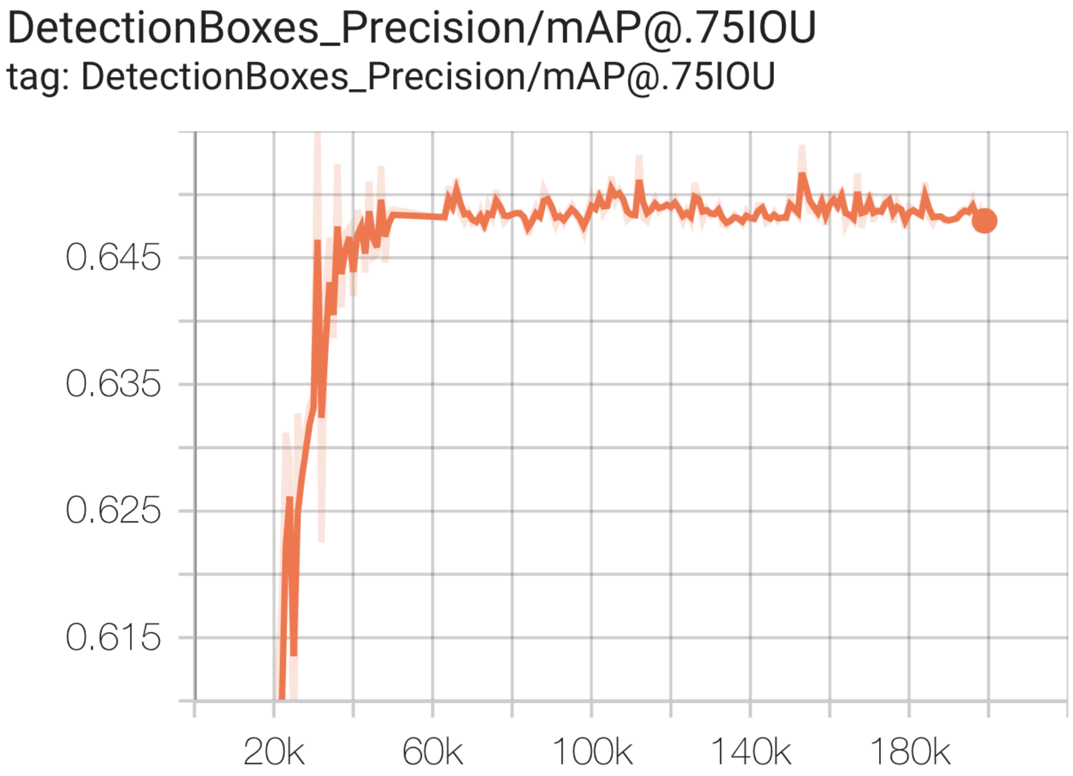


Figure 13.28: Model 2 and 3: mAP @.75IOU score (evaluation started at 14,000 steps)

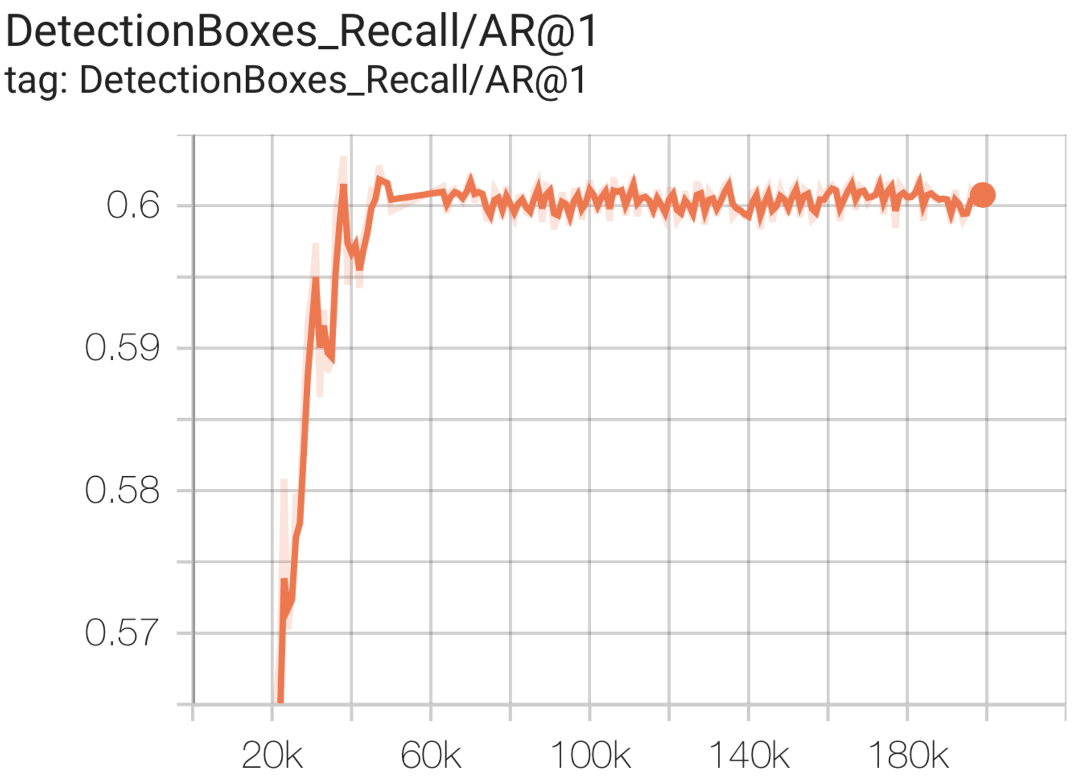


Figure 13.29: Model 2 and 3: recall AR@1 score (evaluation started at 14,000 steps)

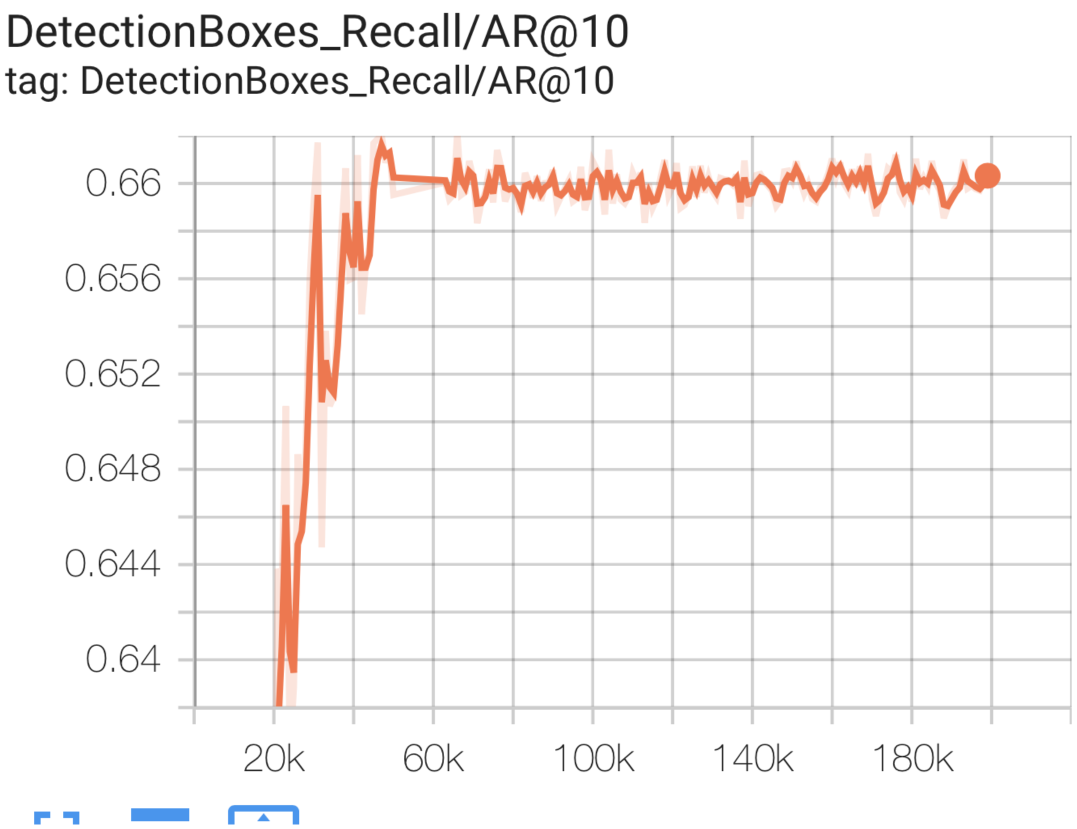


Figure 13.30: Model 2 and 3: recall AR@10 score (evaluation started at 14,000 steps)

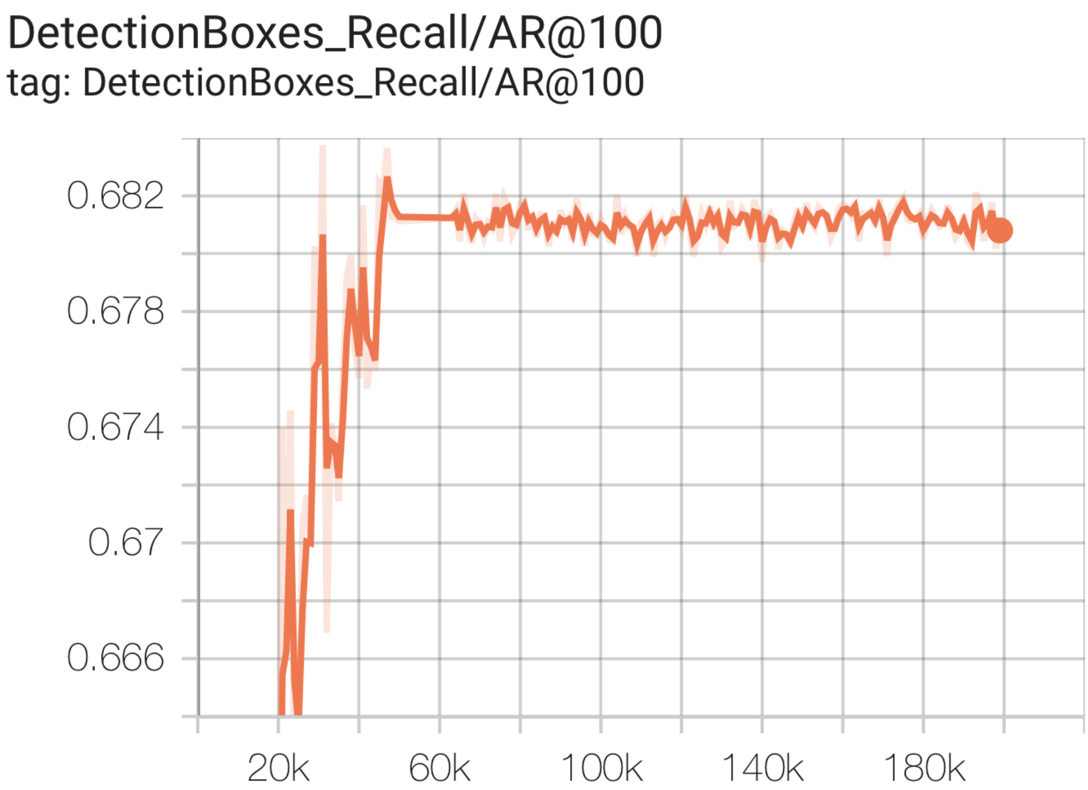


Figure 13.31: Model 2 and 3: recall AR@100 score (evaluation started at 14,000 steps)

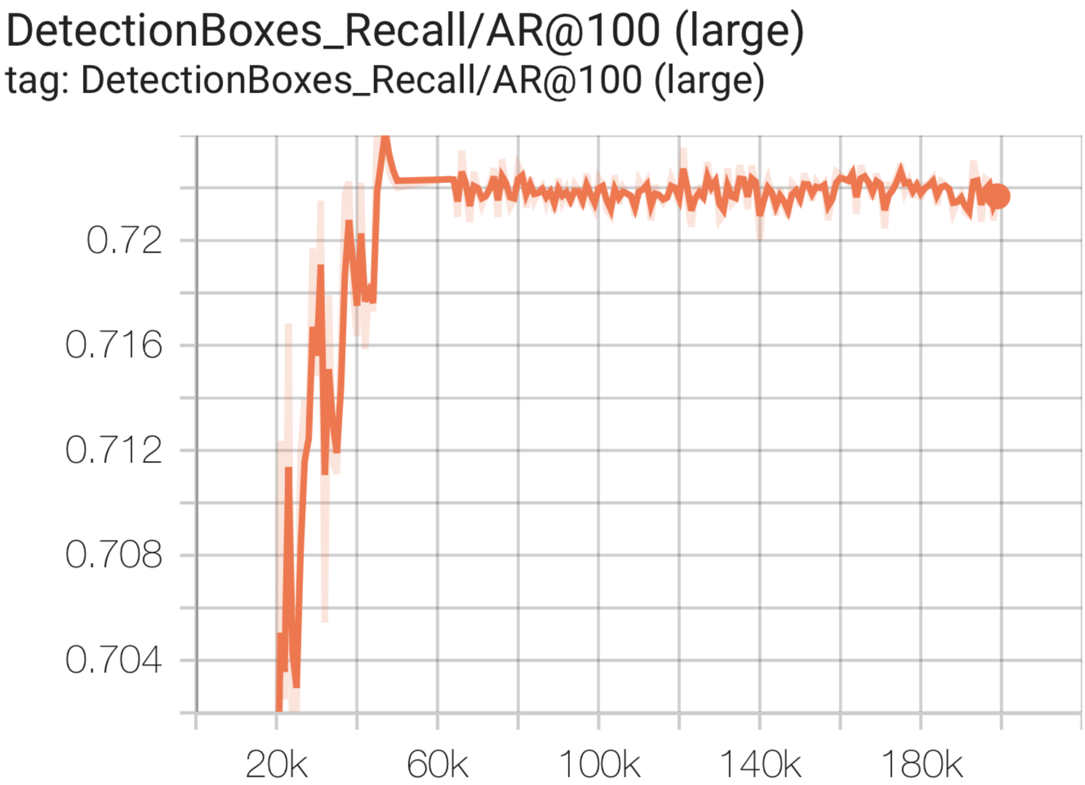


Figure 13.32: Model 2 and 3: recall AR@100 large score (evaluation started at 14,000 steps)

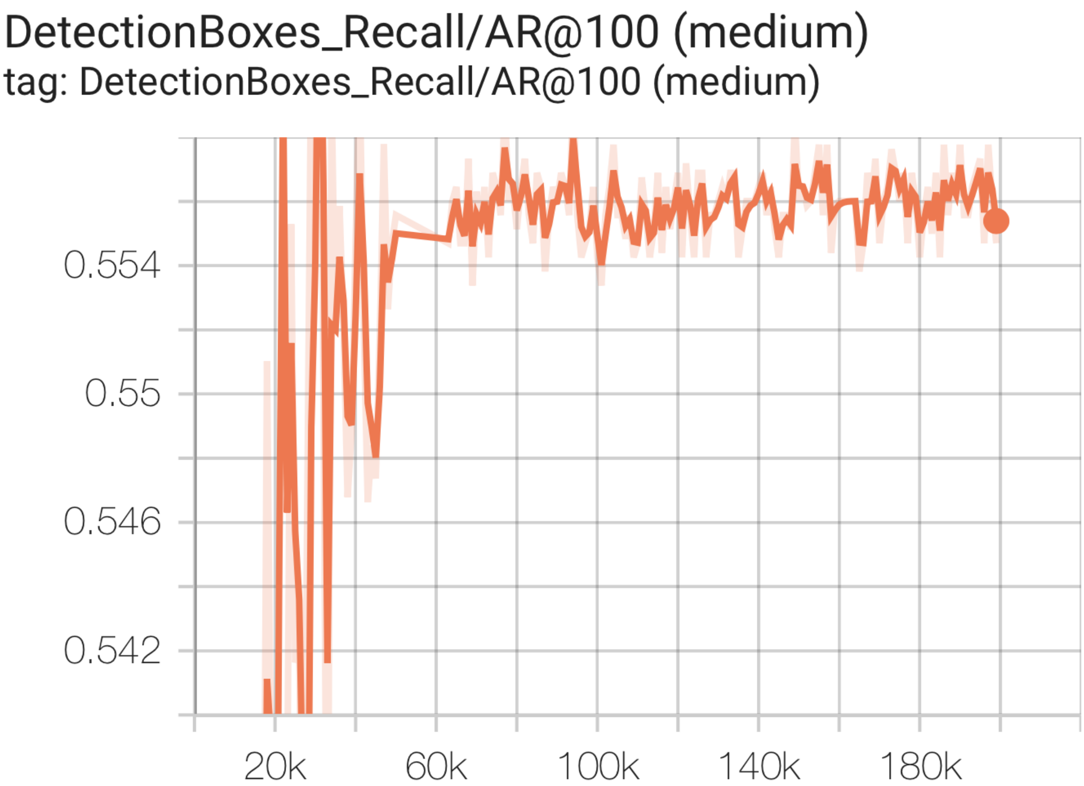


Figure 13.33: Model 2 and 3: recall AR@100 medium score (evaluation started at 14,000 steps)

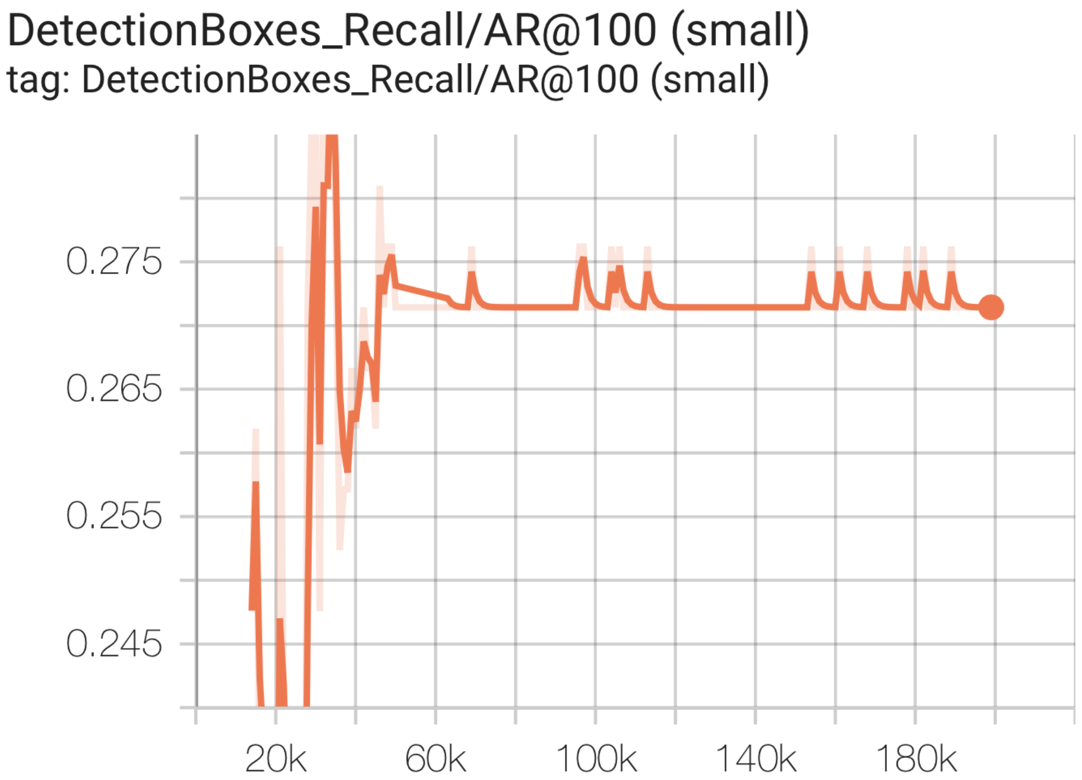


Figure 13.34: Model 2 and 3: recall AR@100 small score (evaluation started at 14,000 steps)

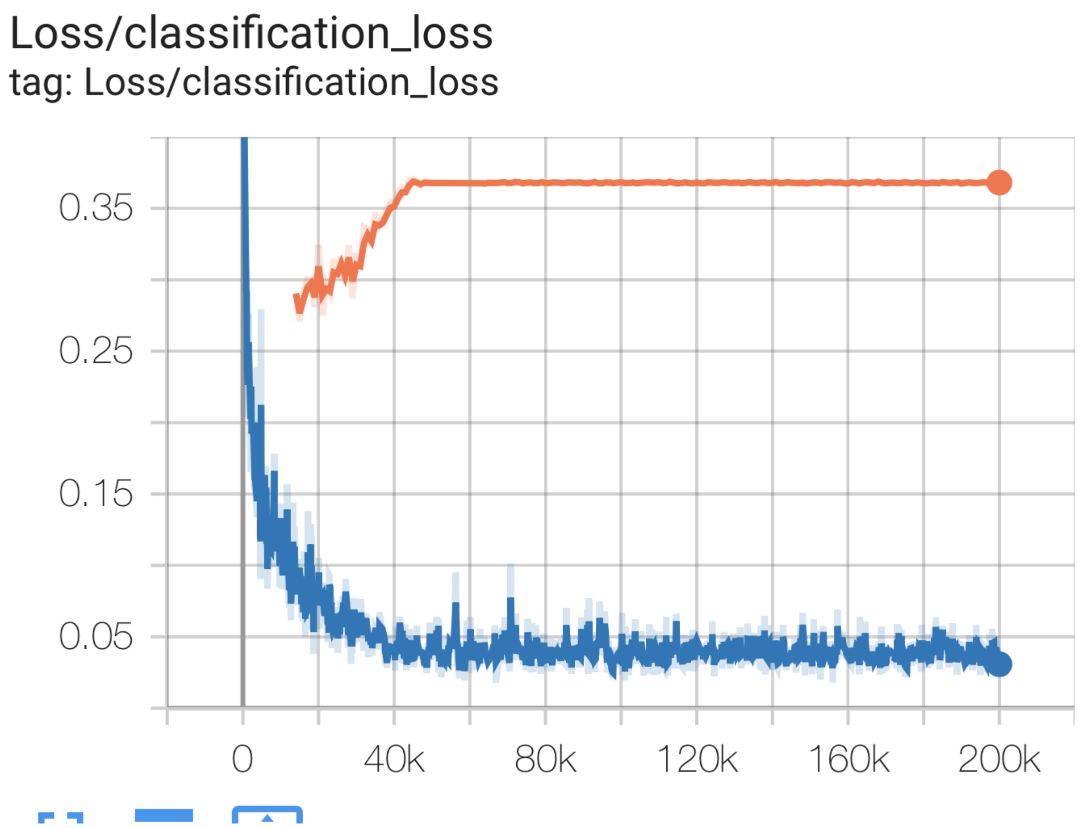


Figure 13.35: Model 2 and 3: classification loss (evaluation started at 14,000 steps)

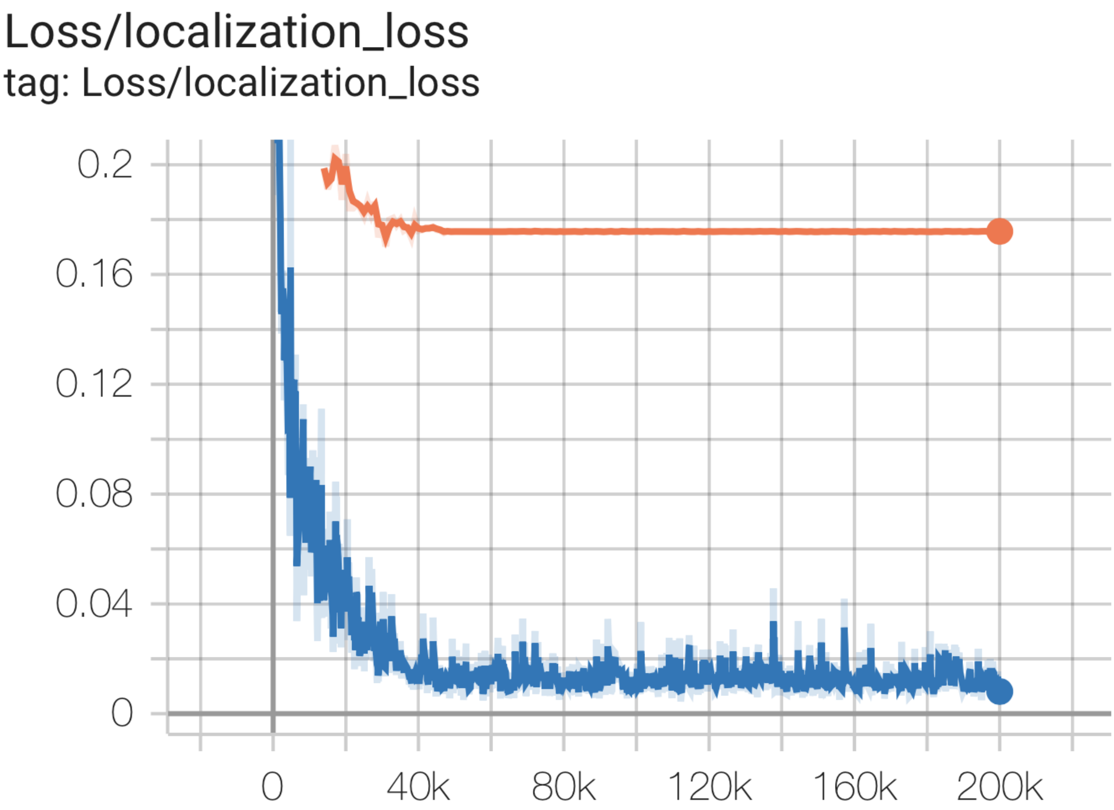


Figure 13.36: Model 2 and 3: localization loss (evaluation started at 14,000 steps)

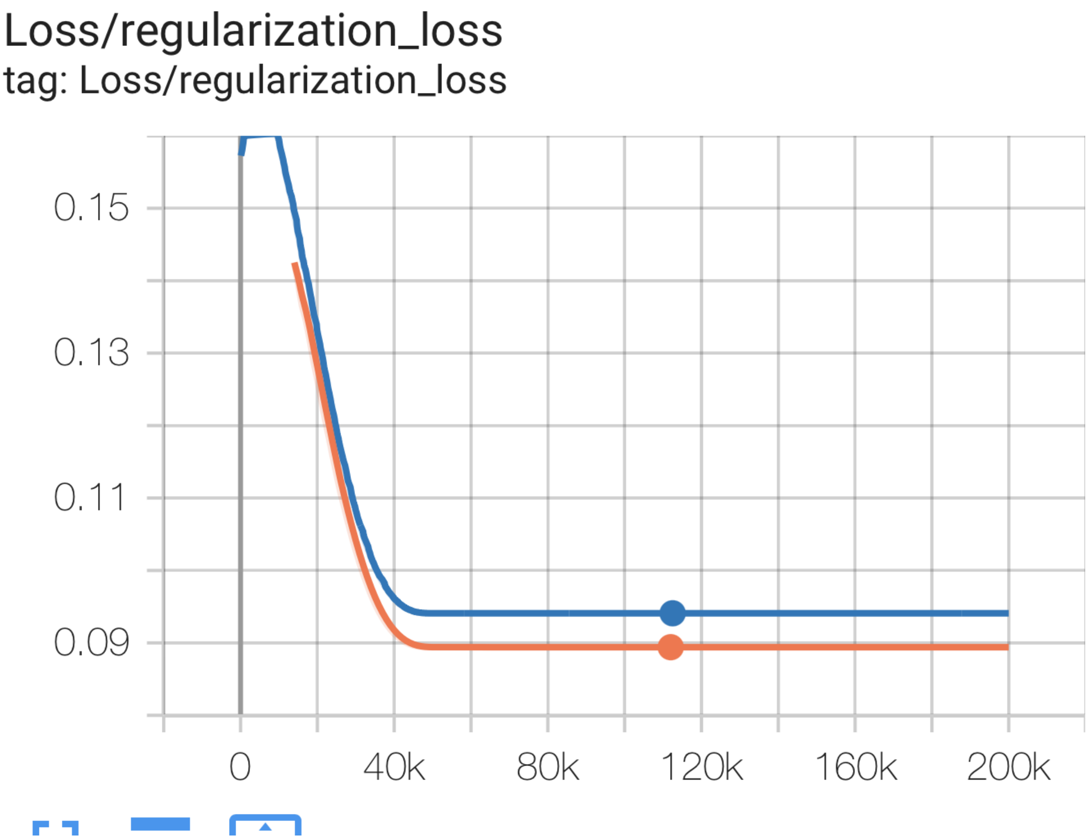


Figure 13.37: Model 2 and 3: regularization loss (evaluation started at 14,000 steps)

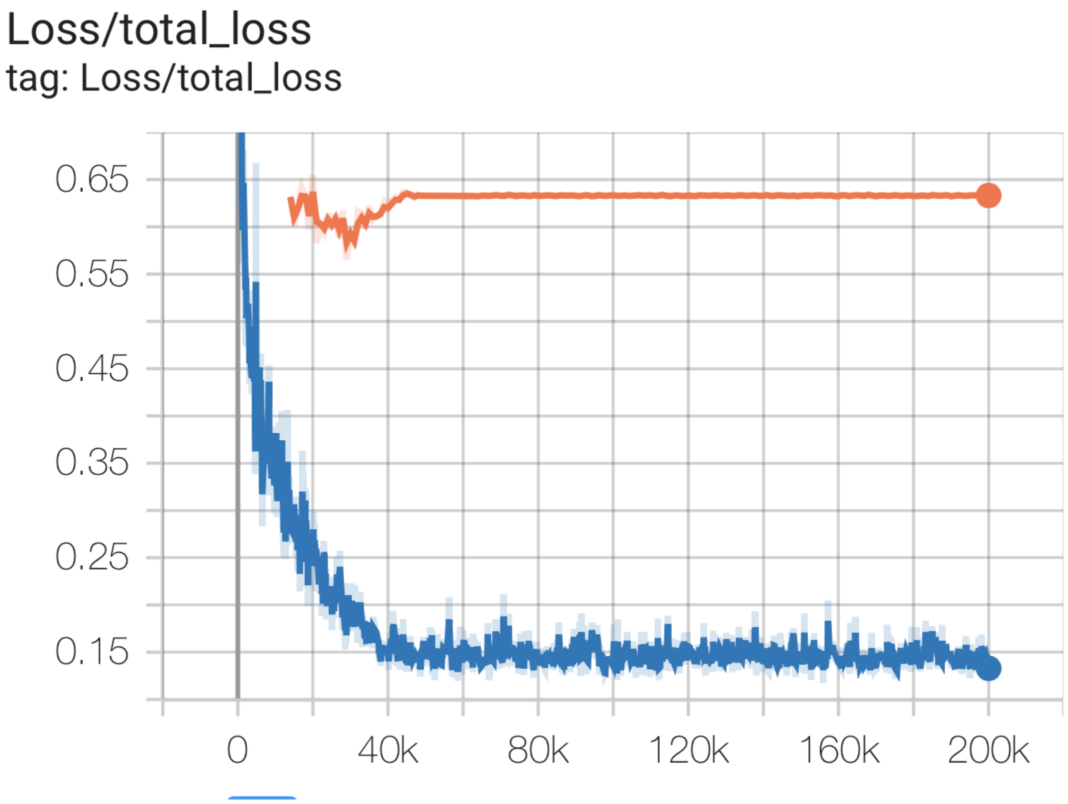


Figure 13.38: Model 2 and 3: total loss (evaluation started at 14,000 steps)

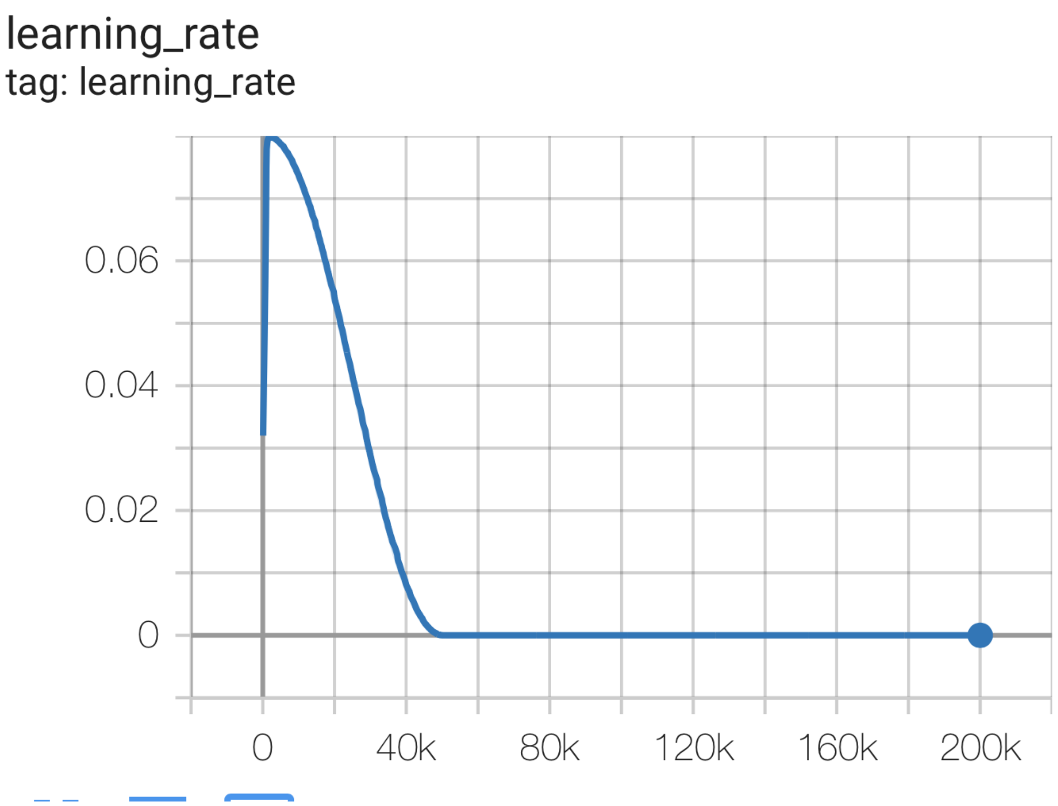


Figure 13.39: Model 2 and 3: learning rate

13.3 Model 4 graphs

As mentioned previously model 4 used the base model "ssd mobilenet v2 fpnlite 640x640", a batch size of 8, 85/100 of the data was split for training and was exported at at 46,000 steps.

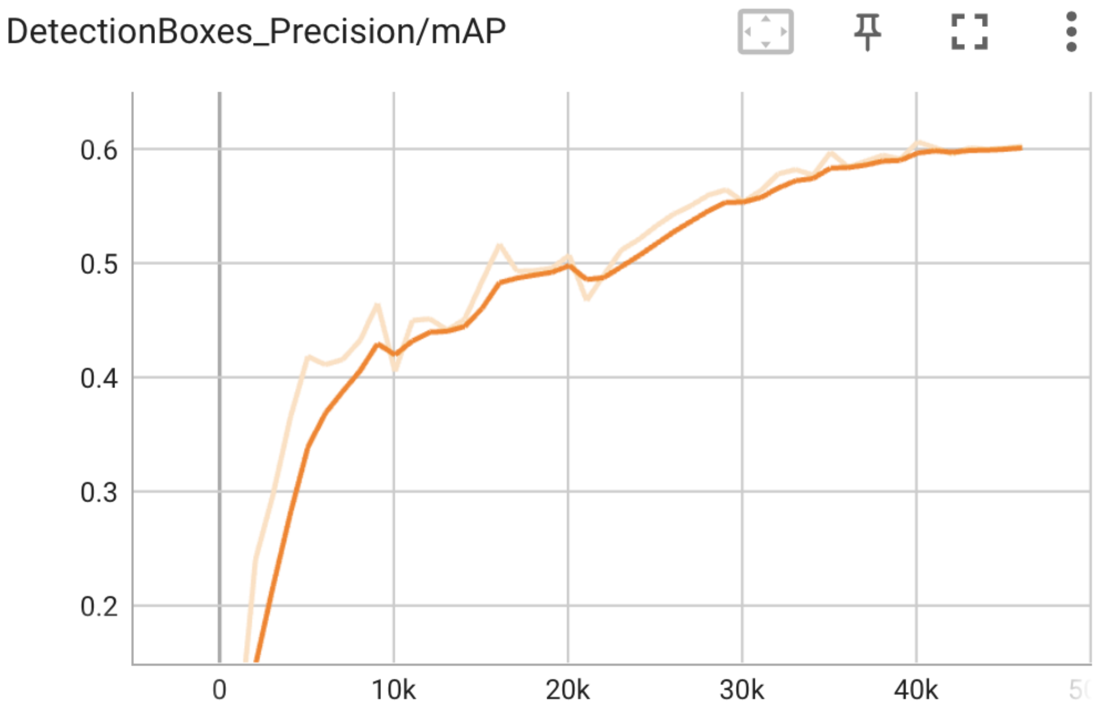


Figure 13.40: Model 4: mAP score

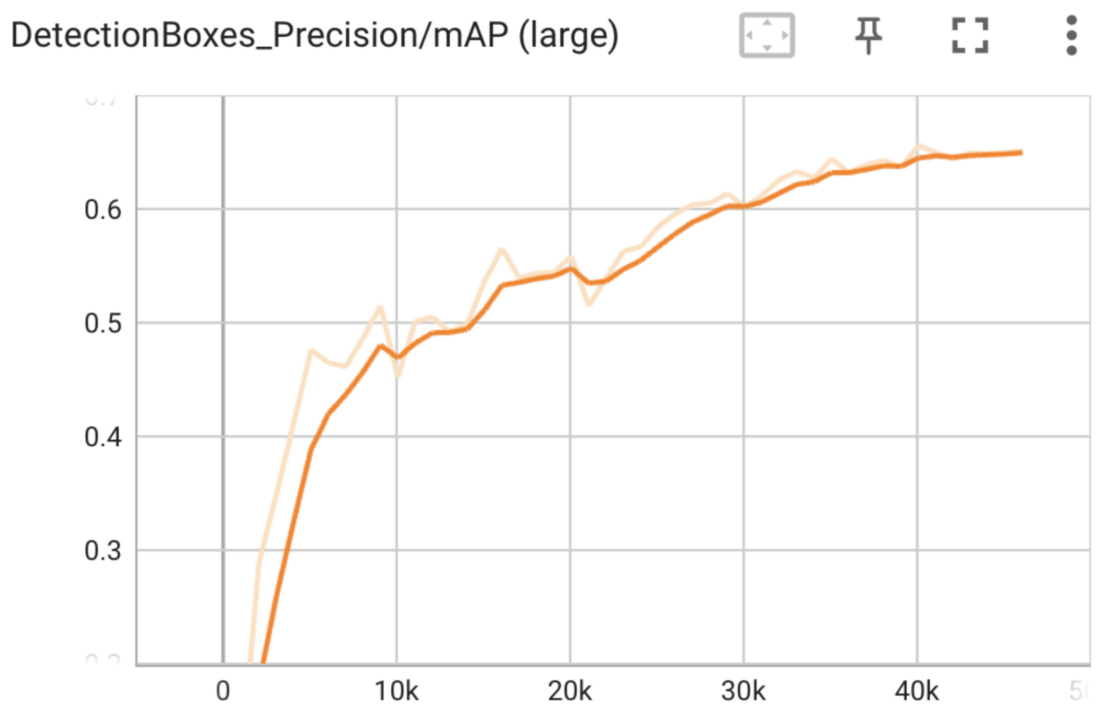


Figure 13.41: Model 4: mAP large score

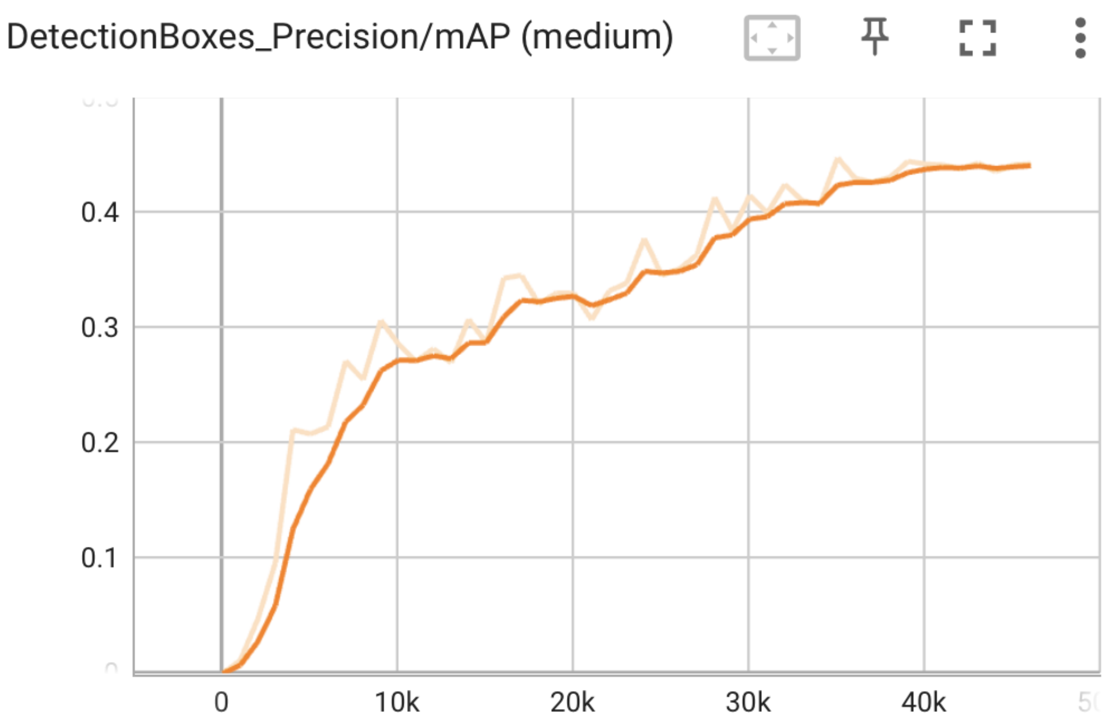


Figure 13.42: Model 4: mAP medium score

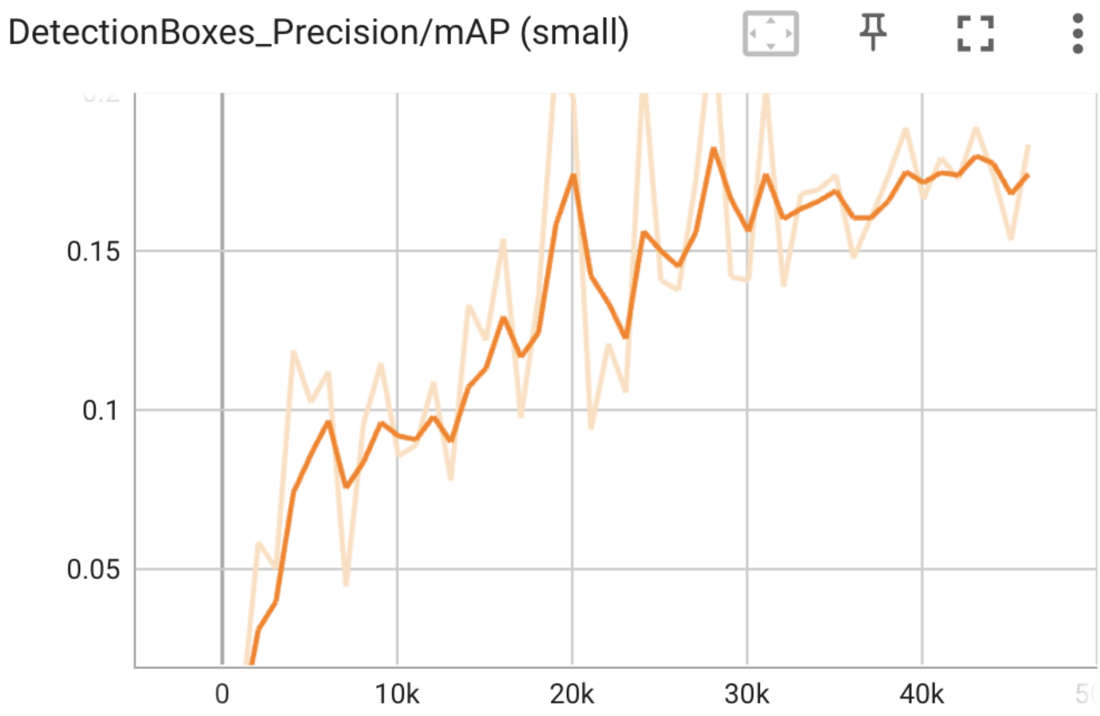


Figure 13.43: Model 4: mAP small score

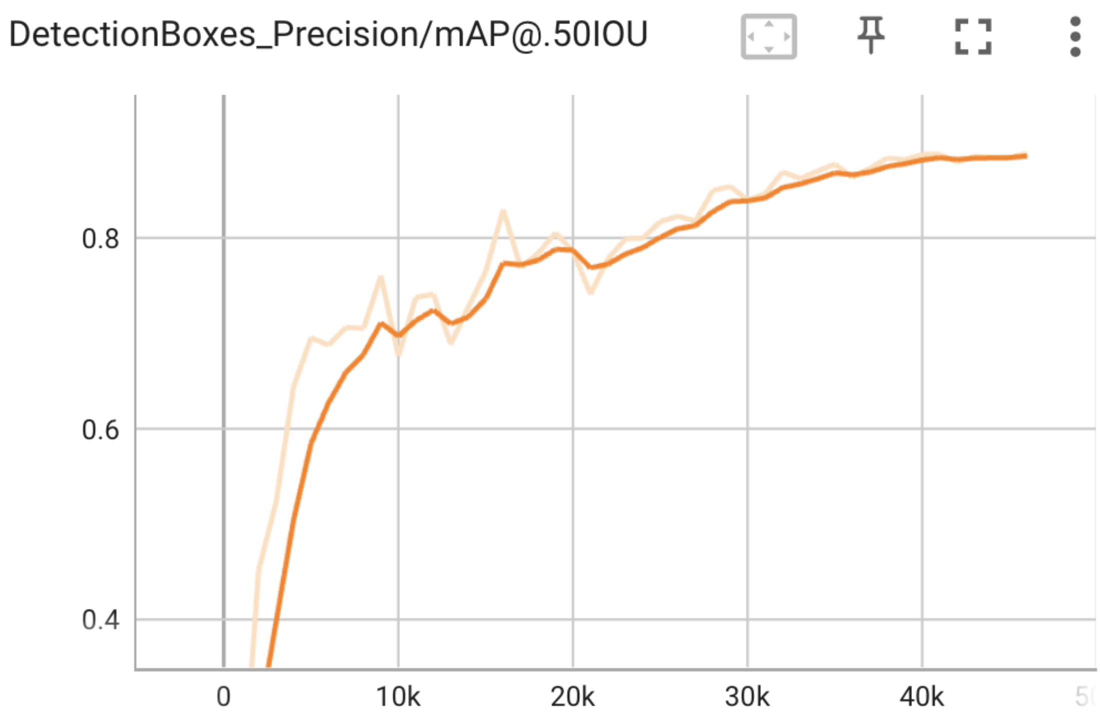


Figure 13.44: Model 4: mAP @.50IOU score

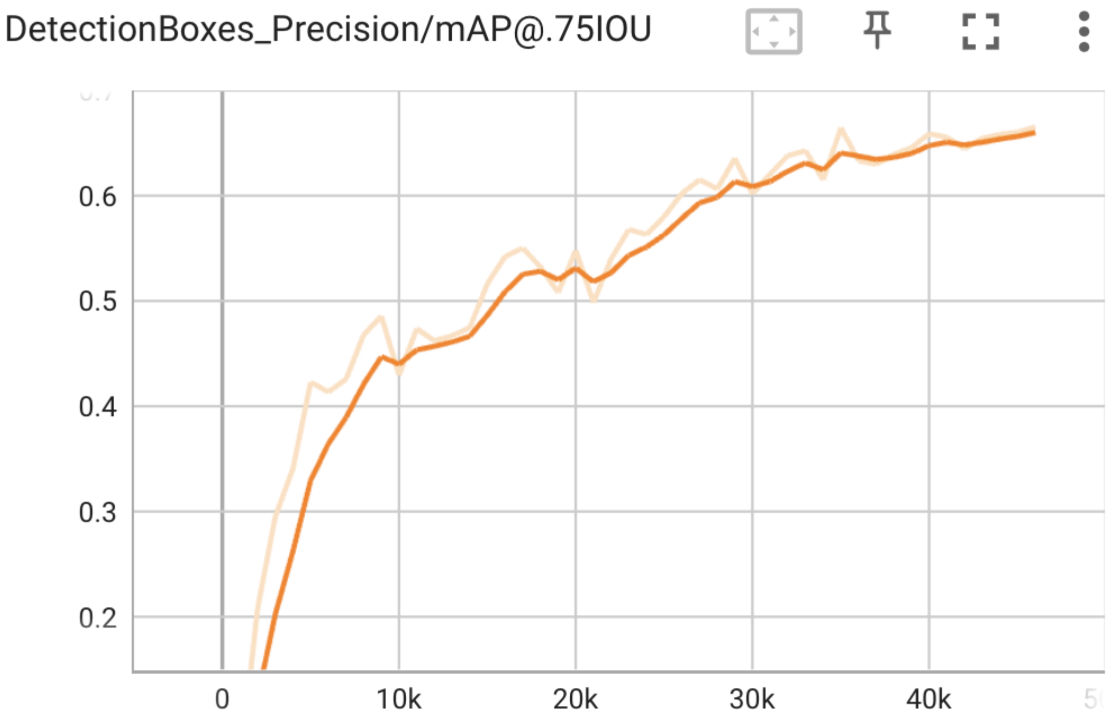


Figure 13.45: Model 4: mAP @.75IOU score

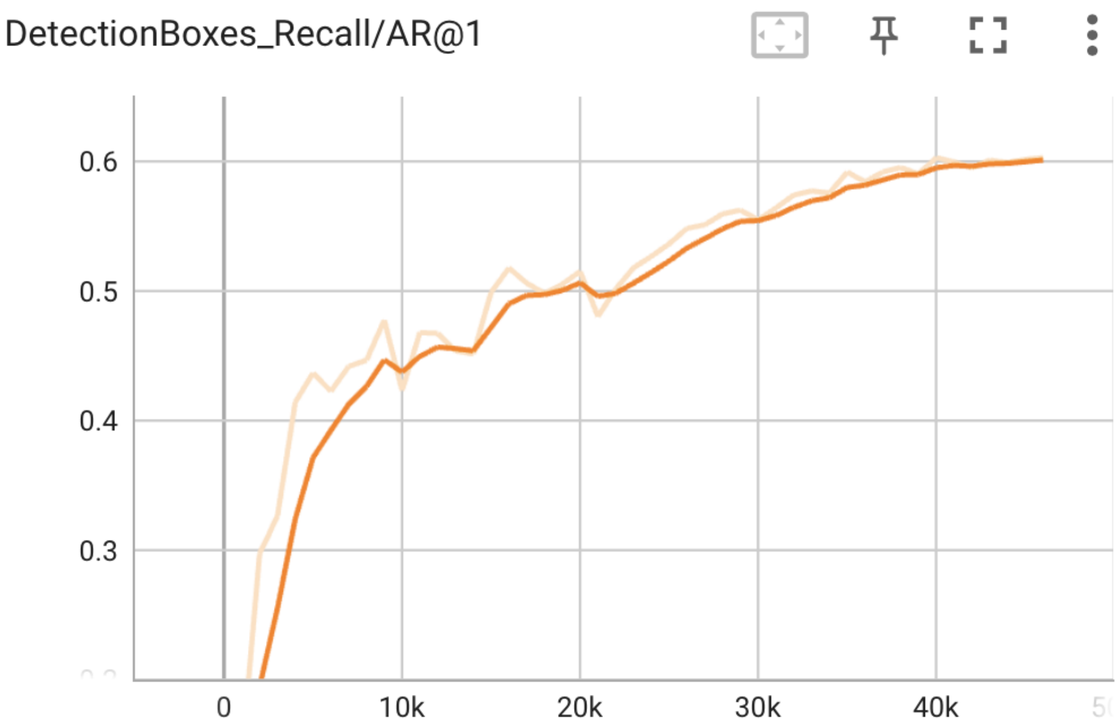


Figure 13.46: Model 4: recall AR@1 score

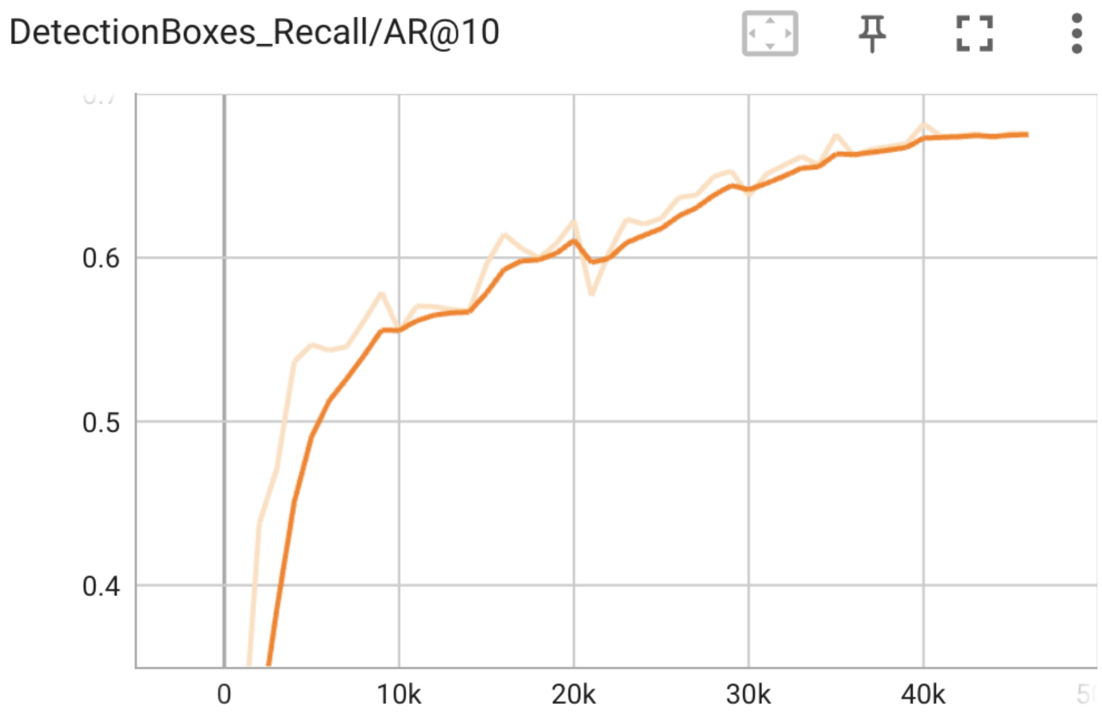


Figure 13.47: Model 4: recall AR@10 score

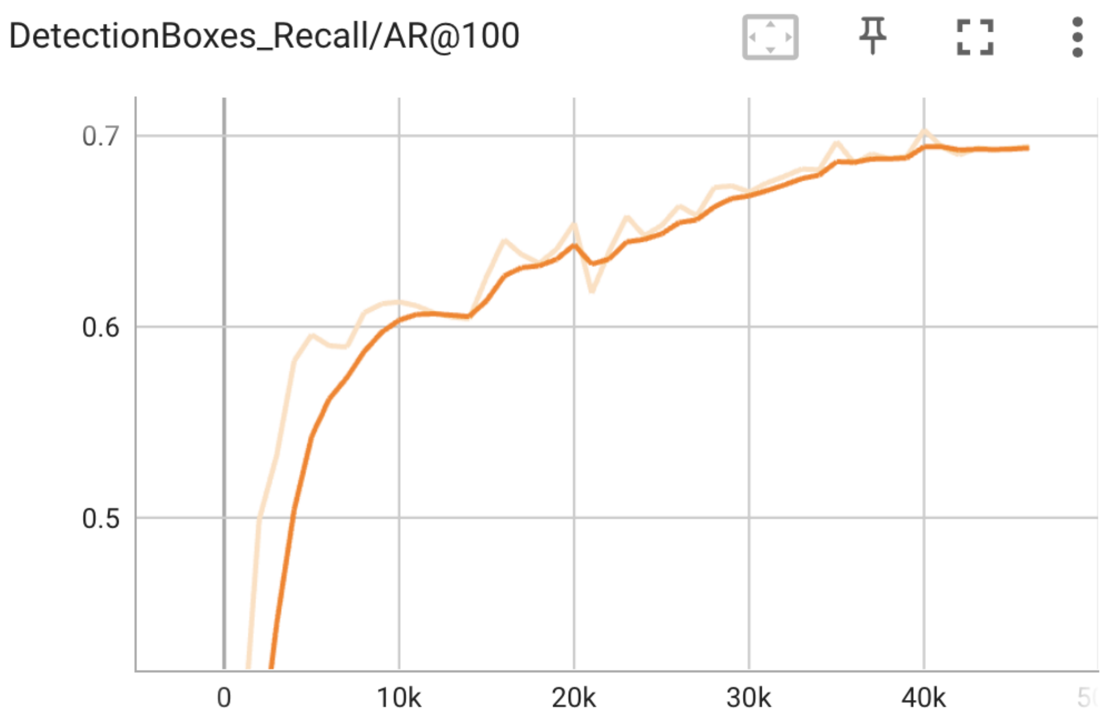


Figure 13.48: Model 4: recall AR@100 score

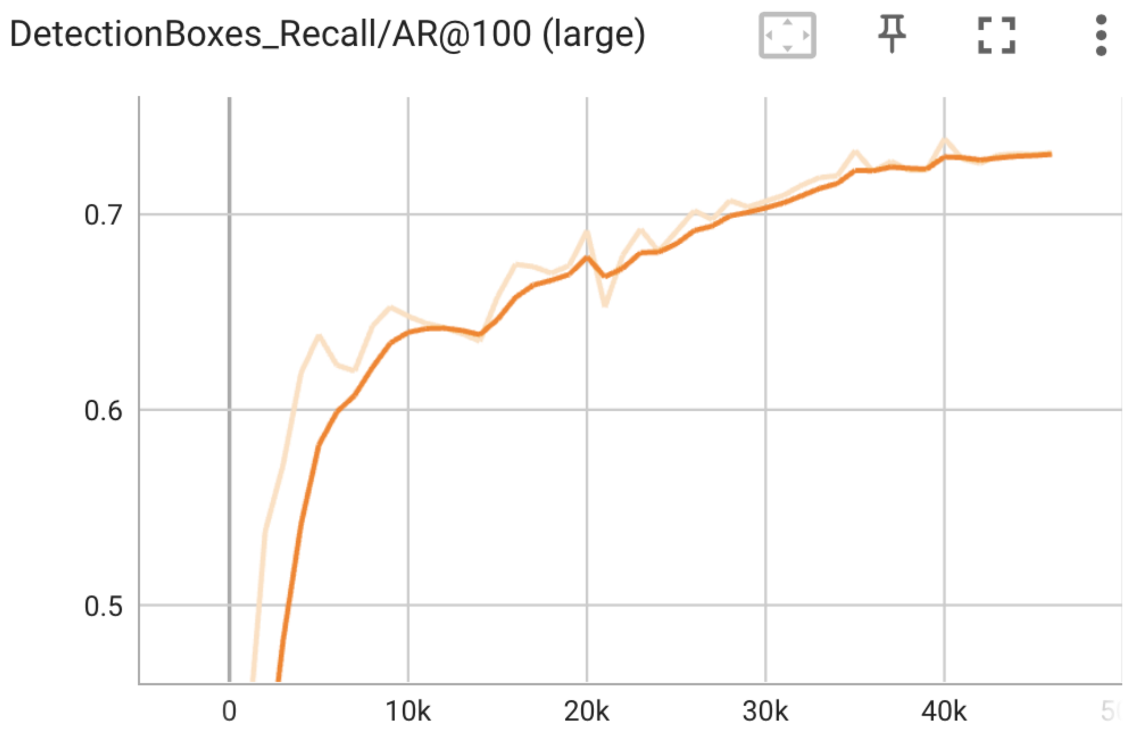


Figure 13.49: Model 4: recall AR@100 large score

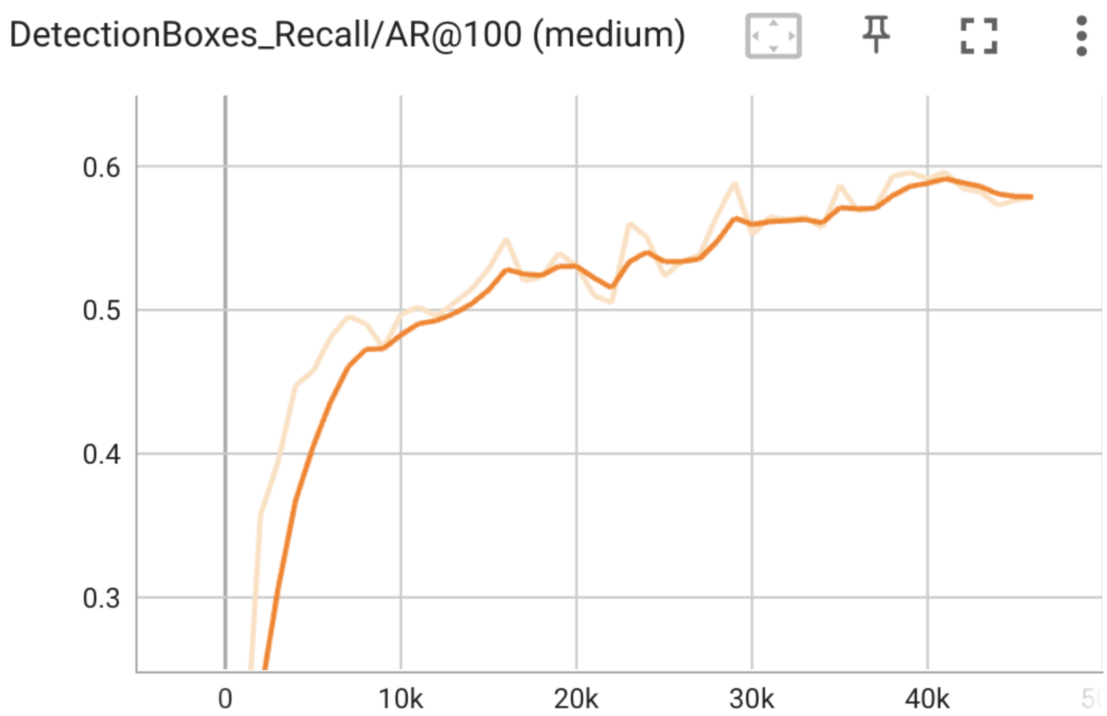


Figure 13.50: Model 4: recall AR@100 medium score

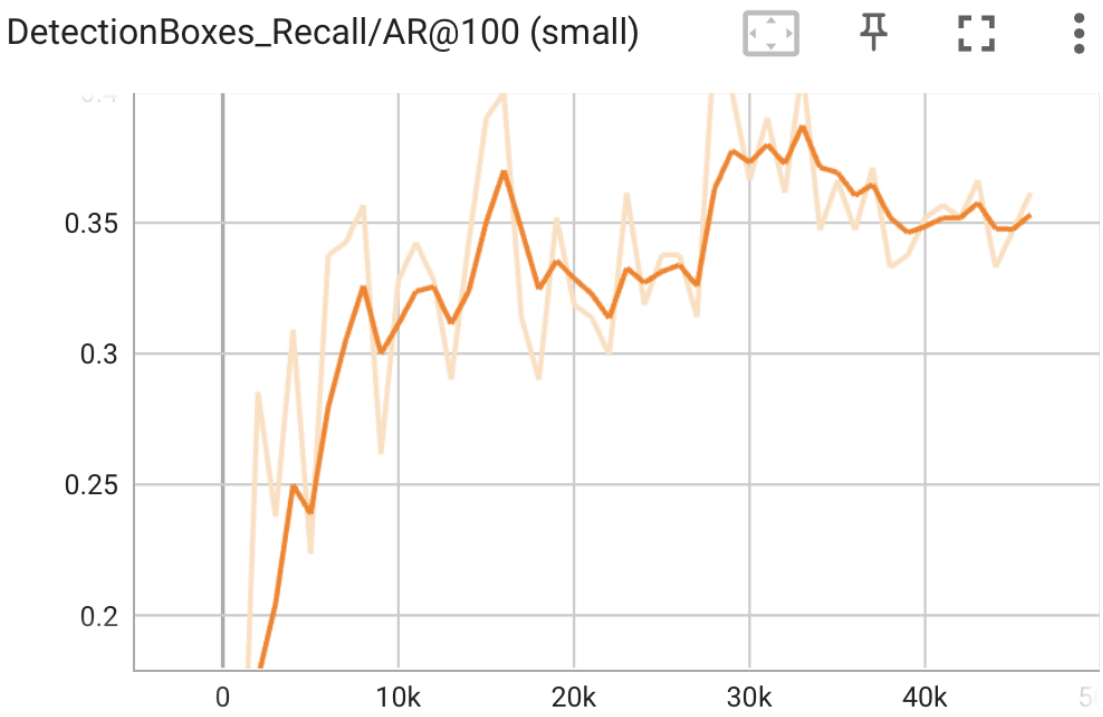


Figure 13.51: Model 4: recall AR@100 small score

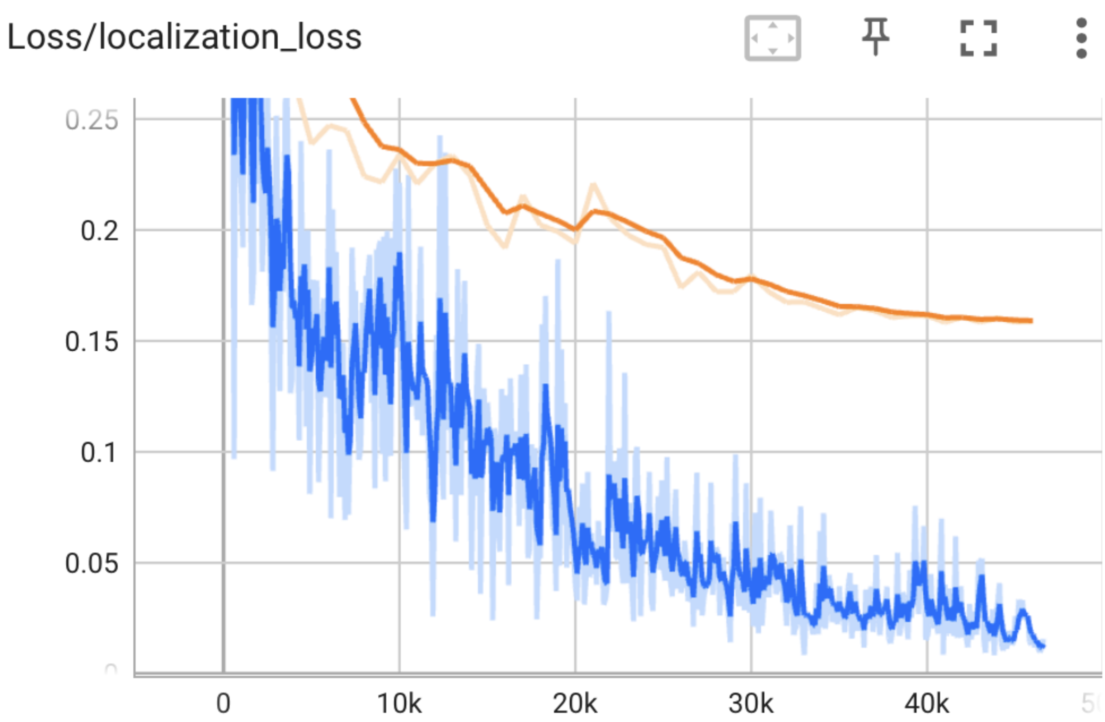


Figure 13.52: Model 4: classification loss

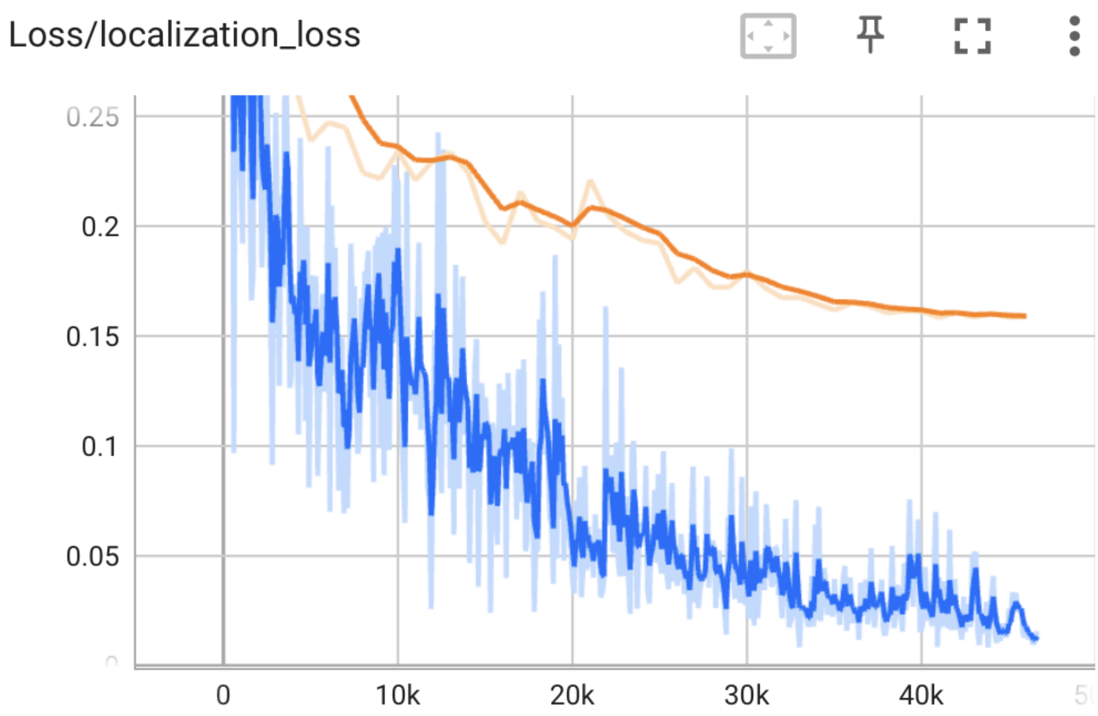


Figure 13.53: Model 4: localization loss

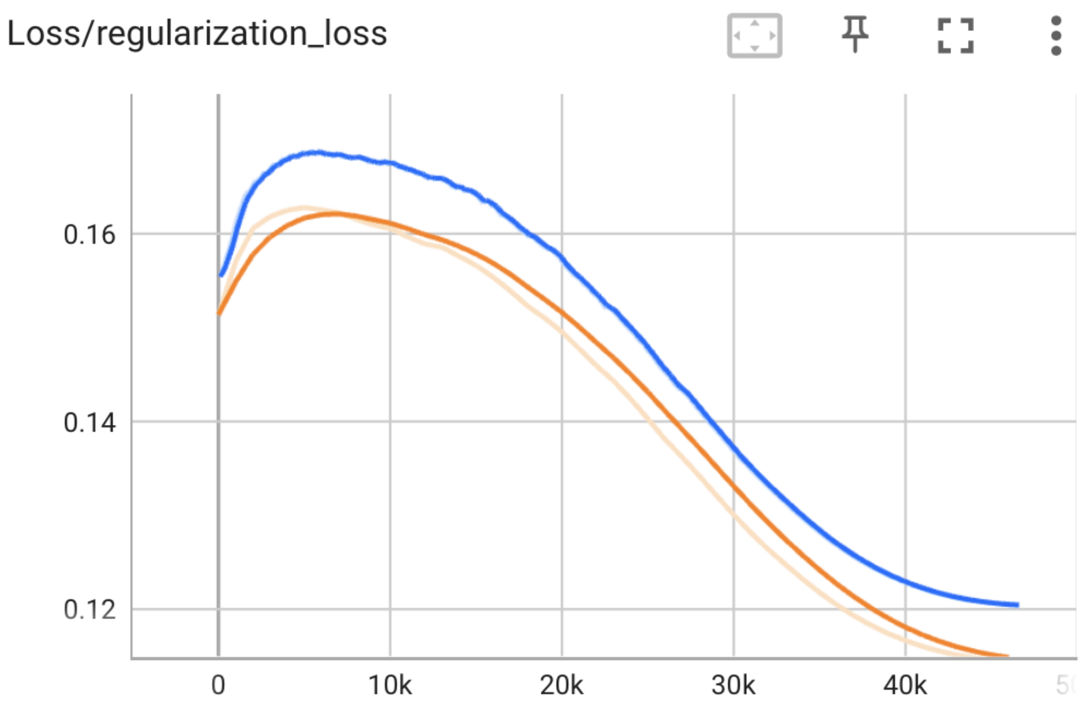


Figure 13.54: Model 4: regularization loss

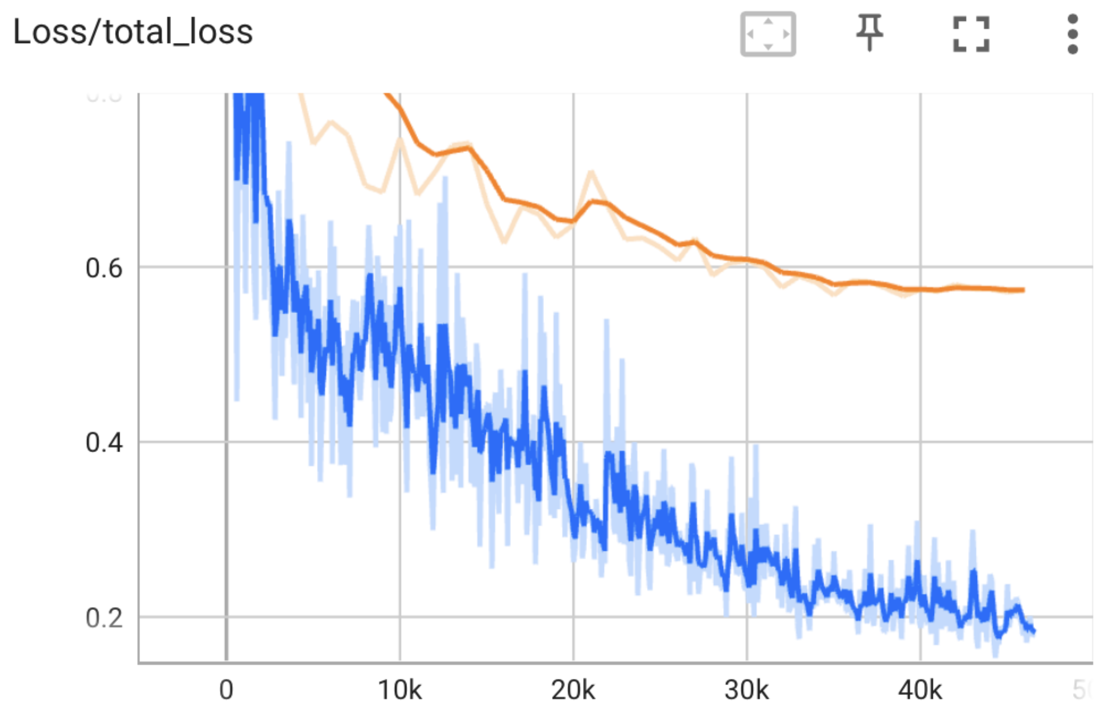


Figure 13.55: Model 4: total loss

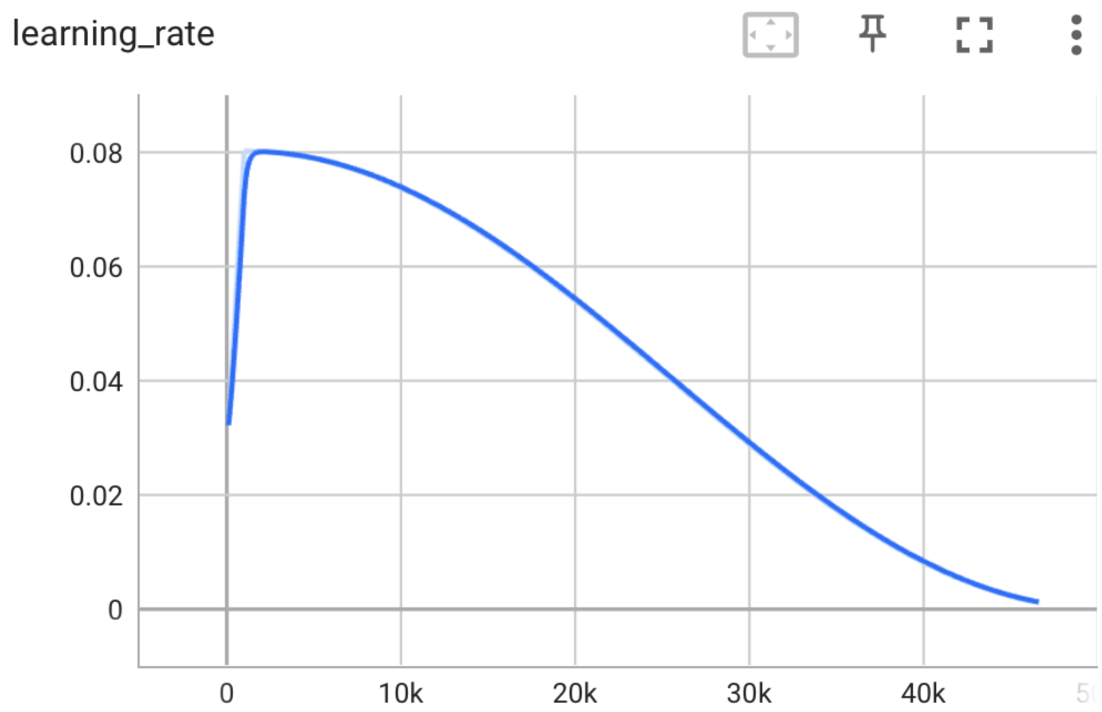


Figure 13.56: Model 4: learning rate

Figure 13.57: Model 4: Eval0 at 1,000 steps-left is model output, right is test label

Figure 13.58: Model 4: Eval0 at 14,000 steps-left is model output, right is test label

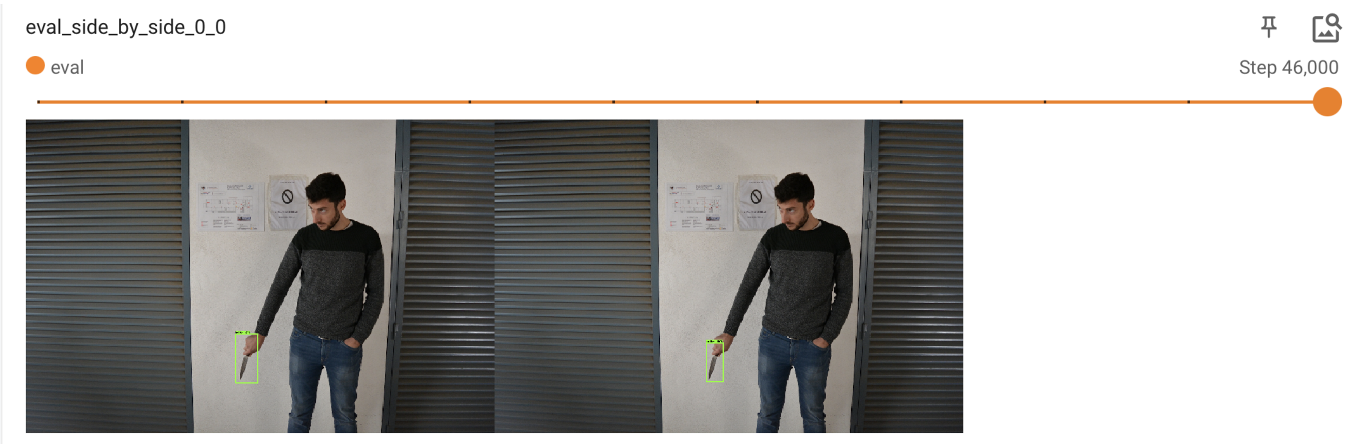


Figure 13.59: Model 4: Eval0 at 46,000 steps-left is model output, right is test label

Chapter 14

Conclusion

The project's primary goal is to reduce the aforementioned events using real-time weapon detection. Using an early warning system, these situations can be stopped by notifying the operators and relevant authorities so that urgent action can be done. Additionally, these detectors offer a wide range of security applications for the protection of human life that will allow authorities to take action right away before a serious crisis occurs. Therefore, the development of weapon detector applications may be a significant contribution to society and make a city considerably safer. Our goal is to provide residents, visitors, and investors with a safe and secure environment.

Chapter 15

Appendix

15.1 Creating model

FOR MAC: tensorflow-metal

Accelerate the training of machine learning models with TensorFlow right on your Mac

```
!python --version
```

```
#CREATING VIRTUAL ENVIRONMENT

#make sure kerranl and ternimal match!

#create
!python3 -m venv ~/MyEnvo/venv-metal

#actiavte
!source ~/MyEnvo/venv-metal/bin/activate
!python -m pip install -U pip
```

```
%pip install  protobuf==3.20.3
```

```
#base tensorfloe for mac os
#Tensorflow: libary for machine learning/CNN
!python -m pip install tensorflow-macos==2.12.0
```

```
#To utilize gpu
!python -m pip install tensorflow-metal
```

```
#verififaction of mac gpu
# can also check activity monitor
import tensorflow as tf

cifar = tf.keras.datasets.cifar100
(x_train, y_train), (x_test, y_test) = cifar.load_data()
model = tf.keras.applications.ResNet50(
    include_top=True,
    weights=None,
    input_shape=(32, 32, 3),
    classes=100,)

loss_fn = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
model.compile(optimizer="adam", loss=loss_fn, metrics=["accuracy"])
model.fit(x_train, y_train, epochs=5, batch_size=64)
```

Figure 15.1: Creating model:1

General

```
#REDEFINE to use easily on differnt computers, define to path this file is

#path for our github repositiory on local computer
rep_path='/Users/Aida/Documents/GitHub/Rep2'

#Check if path correct & working in this path
%cd $rep_path
```

```
# python >=3.8 for tensorflow 2
!python --version
```

```
# defining dictory of project
#Make sure file is created in correct location
%cd $rep_path
#creating directory for Weapon Dectection Syatem
!mkdir WDS
#checking path & reassining working directory
%cd WDS
```

```
##ONLY NEED DO DO ONCE PER COMPUTER/ENVIROMENT

#pip used to install packges, de facto standard
!python get-pip.py
```

Figure 15.2: Creating model:2

Instaling packages

```
#verify install of tf2
!python -c "import tensorflow as tf;print(tf.reduce_sum(tf.random.normal([1000, 1000])))"
```

```
#checking tf version
!tensorflow --version
```

```
#library used in building a custom object detector using Tensorflow Object Detection
%pip install Cython
%pip install pillow
%pip install lxml
%pip install jupyter
%pip install matplotlib
%pip install pycocotools
%pip install opencv-python #for cv2: computer vision
%pip install pandas
%pip install contextlib2
%pip install typing_extensions
%pip update typing-extensions

# pycocotools is a Python API that. # assists in loading, parsing and visualizing the annotations in COCO.
%pip install -qq pycocotools
```

```
!protoc --version  #compiler version>=3 for tf 2
```

Figure 15.3: Creating model:3

GENERAL IMPORTS

```
# Will change the / operator to mean true division throughout the module.
# A command line option will enable run-time warnings for classic division applied to int or long arguments;
#another command line option will make true division the default.
from __future__ import division

# Stop package-internal modules from shadowing top-level modules
# If you run the file as part of the pkg package, the package's internal files stop showing up as top-level.
from __future__ import absolute_import

import pandas as pd
import numpy as np

# creates table-like custom objects from the items in CSV files
import csv

#  A regular expression (or RE) specifies a set of strings that matches it; the functions in this module
# let you check if a particular string matches a given regular expression (or if a given regular expression
# matches a particular string, which comes down to the same thing)
import re

# Unofficial pre-built CPU-only OpenCV packages for Python
import cv2

#lets the user interact with the native OS Python is currently running on
import os

#search CSV files and for text in files.
import glob

# his function takes an XML data string (xml_data) or a file path or file-like object (from_file) as input,
# converts it to the canonical form, and writes it out using the out file(-like) object, if provided, or returns
# it as a text string if not
import xml.etree.ElementTree as ET

#The io module provides Python’s main facilities for dealing with various types of I/O.
#There are three main types of I/O: text I/O, binary I/O and raw I/O.
import io

import tensorflow as tf

# Creates a copy of an image memory from pixel data in a buffer
# In its simplest form, this function takes three arguments (mode, size, and unpacked pixel data).
from PIL import Image

# import namedtuple; namedtuple() is a factory function available in collections
# It allows you to create tuple subclasses with named fields
#You can access the values in a given named tuple using the dot notation and the field names, like in obj

#import OrderedDict; An OrderedDict is a data type in the collections module.
# It is a dict subclass that tracks the order in which items were added.
from collections import namedtuple, OrderedDict

#support file copying and removal.
import shutil

#defines functions and classes which help in opening URLs (mostly HTTP) in a complex world –
# basic and digest authentication, redirections, cookies and more
import urllib.request

# read and write tar archives
import tarfile
```

Figure 15.4: Creating model:4

Orgniazing database/ DONT RUN UNLESS MANULY RESET LABELS

For more information on dataset: https://dasci.es/transferencia/open-data/24705/

Using pistol and knife detection

```
#making sure in right place
%cd $rep_path/WDS

#creating data directory
!mkdir data
#creating sub directories
!mkdir data/images data/train_labels data/test_labels
```

Pistol and knife images in WDS/data/images

https://github.com/ari-dasci/OD-WeaponDetection/trunk/Pistol detection/Weapons/

https://github.com/ari-dasci/OD-WeaponDetection/tree/master/Knife_detection/Images

BEEFORE SPLIT PUT ALL THESE FILES WDS/data/train_labels

#pistol annaotions/bounding box https://github.com/ari-dasci/OD-WeaponDetection/tree/master/Pistol detection/xmls

#knife annaotions/bounding box https://github.com/ari-dasci/OD-WeaponDetection/tree/master/Knife_detection/annotations

```
#DONT RERUN!!!!
#WARNING!!!!!!!!!! DONT RUN WITHOUT HAVING ALL LABELS IN TRAIN LABELS FIRST AND CLEARING TEST FOLDER

#spliting data 75%train 25%test
#3000 pistol+2078knife=5078
#25% => 1270 rounded

#spliting data 85%train 15%test
#3000 pistol+2078knife=5078
#15% => 762 rounded

#ASSUMING anotations are initially in train labels
#start with train as most data will be in trian
#random slipt of data
%cd $rep_path/WDS

!ls $rep_path/WDS/data/train_labels/* | sort -R | head -762 | xargs -I{} mv {} $rep_path/WDS/data/test_labels
#!ls $rep_path/WDS/data/train_labels/* | sort -R | head -1270 | xargs -I{} mv {} $rep_path/WDS/data/test_labels
```

```
#checking if we have right ammount of labels for traning and test
!ls -1 $rep_path/WDS/data/train_labels/ | wc -l
!ls -1 $rep_path/WDS/data/test_labels/ | wc -l
```

Figure 15.5: Creating model:5

preprocessing

```
# again, ensure it is running in right lcoation
%cd $rep_path/WDS/data

# set image extension type
images_extension = 'jpg'

# convert xml files into a csv file that contains the following:
# image name, width, height, class, xmin, ymin, xmax, ymax.
def xml_to_csv(path):
    classes_names = []
    xml_list = []

    for xml_file in glob.glob(path + '/*.xml'):
        tree = ET.parse(xml_file)
        root = tree.getroot()
        for member in root.findall('object'):
            classes_names.append(member[0].text)
            aString=root.find('filename').text
            if aString.startswith("armas"):
                value = (root.find('filename').text + '.' + images_extension,
                         int(root.find('size')[0].text),
                         int(root.find('size')[1].text),
                         member[0].text,
                         int(member[4][0].text),
                         int(member[4][1].text),
                         int(member[4][2].text),
                         int(member[4][3].text))
            else:
                value = (root.find('filename').text,
                         int(root.find('size')[0].text),
                         int(root.find('size')[1].text),
                         member[0].text,
                         int(member[4][0].text),
                         int(member[4][1].text),
                         int(member[4][2].text),
                         int(member[4][3].text))
            xml_list.append(value)
    column_name = ['filename', 'width', 'height', 'class', 'xmin', 'ymin', 'xmax', 'ymax']
    xml_df = pd.DataFrame(xml_list, columns=column_name)
    classes_names = list(set(classes_names))
    classes_names.sort()
    return xml_df, classes_names
```

Figure 15.6: Creating model:6

```python
# for the train_labels and test_labels csv files, run xml_to_csv()
for labelpath in ['train_labels', 'test_labels']:
    image_path = os.path.join(os.getcwd(), labelpath)
    xml_df, classes = xml_to_csv(labelpath)
    xml_df.to_csv(f'{labelpath}.csv', index=None)
    print(f'Successfully converted {labelpath} xml to csv.')

# Creating `label_map.pbtxt`
label_map_path = os.path.join("label_map.pbtxt")
pbtxt_content = ""

# Making a pbtxt file which has the class names
for i, class_name in enumerate(classes):
        # display_name is optional
        pbtxt_content = (
            pbtxt_content
            # + "item {{\n    id: {0}\n    name: '{1}'\n    display_name: 'Knife'\n }}\n\n".format(i + 1, class_name)
            #+ "item {{\n    id: {1}\n    name: '{2}'\n    display_name: 'Pistol'\n }}\n\n".format(i + 1, class_name)
            + "item {{\n    id: {0}\n    name: '{1}'\n  }}\n\n".format(i + 1, class_name)
        )
pbtxt_content = pbtxt_content.strip()
with open(label_map_path, "w") as f:
        f.write(pbtxt_content)
```

```
#checking pbtxt
%cd $rep_path/WDS/data
!cat label_map.pbtxt
```

```
#checking pbtxt
%cd $rep_path/WDS/data
!ls
```

Figure 15.7: Creating model:7

```python
# ensures the images' box position is within the image borders
# images path
images_path=rep_path+'/WDS/data/images/'

# returns image name which had an error with the error causing trait
# eg: xmin, ymin, xmax, ymax.
for CSV_FILE in ['train_labels.csv', 'test_labels.csv']:
  with open(CSV_FILE, 'r') as fid:
      print('Checking file:', CSV_FILE)
      file = csv.reader(fid, delimiter=',')
      first = True
      cnt = 0
      error_cnt = 0
      error = False
      for row in file:
          if error == True:
              error_cnt += 1
              error = False
          if first == True:
              first = False
              continue
          cnt += 1
          name, width, height, xmin, ymin, xmax, ymax = row[0], int(row[1]), int(row[2]), int(row[4]), int(row[5]), int(row[6]), int(row[7])
          #path = os.path.join(images_path, name)
          path = images_path+ name
          img = cv2.imread(path)
          if type(img) == type(None):
              error = True
              print('Could not read image', img)
              continue
          org_height, org_width = img.shape[:2]
          if org_width != width:
              error = True
              print('Width mismatch for image: ', name, width, '!=', org_width)
          if org_height != height:
              error = True
              print('Height mismatch for image: ', name, height, '!=', org_height)
          if xmin > org_width:
              error = True
              print('XMIN > org_width for file', name)
          if xmax > org_width:
              error = True
              print('XMAX > org_width for file', name)
          if ymin > org_height:
              error = True
              print('YMIN > org_height for file', name)
          if ymax > org_height:
              error = True
              print('YMAX > org_height for file', name)
          if error == True:
              print('Error for file: %s' % name)
              print()
      print('Checked %d files and realized %d errors' % (cnt, error_cnt))
```

Figure 15.8: Creating model:8

```
#removing the image with error causing box position
%cd $rep_path/WDS/data/
!rm images/'armas (2815).jpg'
```

```
#reading test_labels
df = pd.read_csv(rep_path+'/WDS/data/test_labels.csv')
# removing image
df = df[df['filename'] != 'armas (2815).jpg']
#reseting index
df.reset_index(drop=True, inplace=True)
#saving df
df.to_csv(rep_path+'/WDS/data/test_labels.csv')
df = pd.read_csv(rep_path+'/WDS/data/train_labels.csv')
# removing armas (2815).jpg
df = df[df['filename'] != 'armas (2815).jpg']
#reseting the index
df.reset_index(drop=True, inplace=True)
#saving the df
df.to_csv(rep_path+'/WDS/data/train_labels.csv')
```

Figure 15.9: Creating model:9

Preparing tensor flow object detection

```
#checking tensor flow version
print(tf.__version__)

#CHANGES HOW WE SET UP TENSOR FLOW
```

```
#API form Model Garden for TensorFlow, needed for object detection API
#only need to run once
%cd $rep_path/WDS
!git clone https://github.com/tensorflow/models/
```

```
#for metrics: COCOAPI
%cd $rep_path
!git clone https://github.com/cocodataset/cocoapi.git
%cd cocoapi/PythonAPI
!make
!cp -r pycocotools $rep_path/WDS/models/research/
```

```
%cd $rep_path/WDS/models/research

#proto buffer compilation, Protobufs used to conifure modele and trainning parapemetres
!protoc object_detection/protos/*.proto --python_out=.
#!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
# You MUST open a new Terminal for the changes in the environment variables to take effect.
```

```
%cd $rep_path/WDS/models/research

#installing objeect dection api
%cp object_detection/packages/tf2/setup.py . #tf2

!python -m pip install .
```

Figure 15.10: Creating model:10

```
%cd $rep_path/WDS/models/research

#appending directories to system path, run every kernel

research_path=rep_path+'/WDS/models/research/'
slim_path=rep_path+'/WDS/models/research/slim/'


import sys
sys.path.append(research_path)
sys.path.append(slim_path)


print(sys.path)
```

```
!cd $rep_path/WDS/models/research/slim

# Slim is a light wieght & highlvl Tensorflow API: to define, train, & evaluate image classification models
#Verify installation of slim
!python -c "from nets import cifarnet; mynet = cifarnet.cifarnet"
```

```
%cd $rep_path/WDS/models/research/
#testing if tensor flow object dection api is corectly installed

!python object_detection/builders/model_builder_tf2_test.py
```

Figure 15.11: Creating model:11

Making tensor flow records

\+ Code

```
from object_detection.utils import dataset_util
%cd $rep_path/WDS/models/

data_path = rep_path +'/WDS/data/'
img_path=data_path +'images/'

#defing our different classes
def class_text_to_int(row_label):
    if row_label == 'knife':
        return 1
    if row_label == 'pistol':
        return 2
    else:
        None

#defining slipt
def split(df, group):
    data = namedtuple('data', ['filename', 'object'])
    gb = df.groupby(group)
    return [data(filename, gb.get_group(x)) for filename, x in zip(gb.groups.keys(), gb.groups)]

#defining create tf example
def create_tf_example(group, path):
    with tf.io.gfile.GFile(os.path.join(path, '{}'.format(group.filename)), 'rb') as fid:
            encoded_jpg = fid.read()
    encoded_jpg_io = io.BytesIO(encoded_jpg)
    image = Image.open(encoded_jpg_io)
    width, height = image.size

    filename = group.filename.encode('utf8')
    image_format = b'jpg'
    xmins = []
    xmaxs = []
    ymins = []
    ymaxs = []
    classes_text = []
    classes = []

    for index, row in group.object.iterrows():
        xmins.append(row['xmin'] / width)
        xmaxs.append(row['xmax'] / width)
        ymins.append(row['ymin'] / height)
        ymaxs.append(row['ymax'] / height)
        classes_text.append(row['class'].encode('utf8'))
        classes.append(class_text_to_int(row['class']))

    tf_example = tf.train.Example(features=tf.train.Features(feature={
        'image/height': dataset_util.int64_feature(height),
        'image/width': dataset_util.int64_feature(width),
        'image/filename': dataset_util.bytes_feature(filename),
        'image/source_id': dataset_util.bytes_feature(filename),
        'image/encoded': dataset_util.bytes_feature(encoded_jpg),
        'image/format': dataset_util.bytes_feature(image_format),
        'image/object/bbox/xmin': dataset_util.float_list_feature(xmins),
        'image/object/bbox/xmax': dataset_util.float_list_feature(xmaxs),
        'image/object/bbox/ymin': dataset_util.float_list_feature(ymins),
        'image/object/bbox/ymax': dataset_util.float_list_feature(ymaxs),
        'image/object/class/text': dataset_util.bytes_list_feature(classes_text),
        'image/object/class/label': dataset_util.int64_list_feature(classes),
    }))
    return tf_example
```

Figure 15.12: Creating model:12

```
%cd $rep_path/WDS/models/

#creating tf records from CSV

for csv in ['train_labels', 'test_labels']:
  #training tf record
  writer = tf.io.TFRecordWriter(data_path + csv + '.record')
  path = os.path.join(img_path)

  #cvs to tf record
  sample = pd.read_csv(data_path + csv + '.csv')
  grouped = split(sample, 'filename')

  #writing records
  for group in grouped:
    tf_sample = create_tf_example(group, path)
    writer.write(tf_sample.SerializeToString())

  writer.close()
  out_path = os.path.join(os.getcwd(), data_path + csv + '.record')
  print('Finisheed making the TFRecords: {}'.format(data_path +csv + '.record'))
```

```
#showing whats in data folder
%cd $rep_path/WDS/models/
!ls -l ../data/
```

Figure 15.13: Creating model:13

Base models

```
#For TF2
#Base models used

# max Number of training steps.
num_steps = 100000
# Number of evaluation steps.
num_eval_steps = 50

selected_model = 'SSD_MobileNet_V2_FPNLite_640x640'


# Some models to train on
MODELS_CONFIG = {
    'SSD_MobileNet_V2_FPNLite_320x320': {
        'model_name': 'ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8',
        'pipeline_file': 'pipeline.config',
        'batch_size': 16
    },
    'EfficientDet_D1_640x640': {
        'model_name': 'efficientdet_d1_coco17_tpu-32',
        'pipeline_file': 'pipeline.config',
        'batch_size': 16
    },
    'CenterNet_Resnet101_V1_FPN_512x512': {
        'model_name': 'centernet_resnet101_v1_fpn_512x512_coco17_tpu-8',
        'pipeline_file': 'pipeline.config',
        'batch_size': 16
    },
    'SSD_ResNet101_V1_FPN_640x640_(RetinaNet101)': {
        'model_name': 'ssd_resnet101_v1_fpn_640x640_coco17_tpu-8',
        'pipeline_file': 'pipeline.config',
        'batch_size': 16
    },
    'SSD_MobileNet_V2_FPNLite_640x640': {
        'model_name': 'ssd_mobilenet_v2_fpnlite_640x640_coco17_tpu-8',
        'pipeline_file': 'pipeline.config',
        'batch_size': 8
    }

}
```

Figure 15.14: Creating model:14

CONFIGURING THE THE TRAINING PIPELINE

```
#--------------------------------------- CONFIGURING THE THE TRAINING PIPELINE -------------------------------------------


# This part is configuring the training pipeline for a weapon detection system using the TensorFlow Object Detection API.
# It is writing a configuration file for the SSD-MobileNet-V2 model, which is a popular object detection model.

# The %%writefile command is a Jupyter notebook-specific command that writes the following code to a file.
# In this case, the code is being written to the file "file path"

# The code is divided into three main sections: model, train_config, and train_input_reader.

# Model section:

# The model section specifies the architecture of the model, including the type of object detection model (ssd),
# the number of classes being detected (num_classes: 1),
# the box coder used for encoding bounding box coordinates,
# and the anchor generator used to generate anchor boxes for the model.


# The image_resizer specifies the input image size for the model.
# In this case, the input images will be resized to 300x300 pixels.
# The box_predictor specifies the architecture of the box predictor network,
# which predicts the class and location of each object in the image.
# The feature_extractor specifies the architecture of the feature extractor network,
# which extracts features from the input image.
# The loss specifies the loss function used to train the model.


# Train_config section:

# The batch_size specifies the number of images used in each training batch.
# The optimizer specifies the optimization algorithm used to train the model.
# In this case, the model is trained using the RMSProp optimizer.
# The fine_tune_checkpoint specifies the path to the pre-trained model checkpoint used for transfer learning.
# The num_steps specifies the number of training steps to run.
# The data_augmentation_options specify the data augmentation techniques used during training.



# Train_input_reader section:

# The tf_record_input_reader specifies the path to the TFRecords file containing the training data.
# The label_map_path specifies the path to the label map file, which maps the class names to integer labels.
```

Figure 15.15: Creating model:15

```
%%writefile $rep_path/WDS/our_models/our_SSD_MobileNet_V2_FPNLite_320x320/pipeline.config
model {
  ssd {
    num_classes: 2
    image_resizer {
      fixed_shape_resizer {
        height: 320
        width: 320
      }
    }
    feature_extractor {
      type: "ssd_mobilenet_v2_fpn_keras"
      depth_multiplier: 1.0
      min_depth: 16
      conv_hyperparams {
        regularizer {
          l2_regularizer {
            weight: 3.9999998989515007e-05
          }
        }
        initializer {
          random_normal_initializer {
            mean: 0.0
            stddev: 0.009999999776482582
          }
        }
        activation: RELU_6
        batch_norm {
          decay: 0.996999979019165
          scale: true
          epsilon: 0.0010000000474974513
        }
      }
      use_depthwise: true
      override_base_feature_extractor_hyperparams: true
      fpn {
        min_level: 3
        max_level: 7
        additional_layer_depth: 128
      }
    }
```

Figure 15.16: Creating model:16

```
box_coder {
  faster_rcnn_box_coder {
    y_scale: 10.0
    x_scale: 10.0
    height_scale: 5.0
    width_scale: 5.0
  }
}
matcher {
  argmax_matcher {
    matched_threshold: 0.5
    unmatched_threshold: 0.5
    ignore_thresholds: false
    negatives_lower_than_unmatched: true
    force_match_for_each_row: true
    use_matmul_gather: true
  }
}
similarity_calculator {
  iou_similarity {
  }
}
box_predictor {
  weight_shared_convolutional_box_predictor {
    conv_hyperparams {
      regularizer {
        l2_regularizer {
          weight: 3.9999998989515007e-05
        }
      }
      initializer {
        random_normal_initializer {
          mean: 0.0
          stddev: 0.009999999776482582
        }
      }
      activation: RELU_6
      batch_norm {
        decay: 0.996999979019165
        scale: true
        epsilon: 0.0010000000474974513
      }
    }
    depth: 128
    num_layers_before_predictor: 4
    kernel_size: 3
    class_prediction_bias_init: -4.599999904632568
    share_prediction_tower: true
    use_depthwise: true
  }
}
```

Figure 15.17: Creating model:17

```
    anchor_generator {
      multiscale_anchor_generator {
        min_level: 3
        max_level: 7
        anchor_scale: 4.0
        aspect_ratios: 1.0
        aspect_ratios: 2.0
        aspect_ratios: 0.5
        scales_per_octave: 2
      }
    }
    post_processing {
      batch_non_max_suppression {
        score_threshold: 9.99999993922529e-09
        iou_threshold: 0.6000000238418579
        max_detections_per_class: 100
        max_total_detections: 100
        use_static_shapes: false
      }
      score_converter: SIGMOID
    }
    normalize_loss_by_num_matches: true
    loss {
      localization_loss {
        weighted_smooth_l1 {
        }
      }
      classification_loss {
        weighted_sigmoid_focal {
          gamma: 2.0
          alpha: 0.25
        }
      }
      classification_weight: 1.0
      localization_weight: 1.0
    }
    encode_background_as_zeros: true
    normalize_loc_loss_by_codesize: true
    inplace_batchnorm_update: true
    freeze_batchnorm: false
  }
}
```

Figure 15.18: Creating model:18

```
train_config {
  batch_size: 16
  data_augmentation_options {
    random_horizontal_flip {
    }
  }
  data_augmentation_options {
    random_crop_image {
      min_object_covered: 0.0
      min_aspect_ratio: 0.75
      max_aspect_ratio: 3.0
      min_area: 0.75
      max_area: 1.0
      overlap_thresh: 0.0
    }
  }
  sync_replicas: true
  optimizer {
    momentum_optimizer {
      learning_rate {
        cosine_decay_learning_rate {
          learning_rate_base: 0.07999999821186066
          total_steps: 50000
          warmup_learning_rate: 0.026666000485420227
          warmup_steps: 1000
        }
      }
      momentum_optimizer_value: 0.8999999761581421
    }
    use_moving_average: false
  }
  fine_tune_checkpoint: "/Users/Aida/Documents/GitHub/Rep2/WDS/pre-trained-models/ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8/checkpoint/ckpt-0"
  num_steps: 50000
  startup_delay_steps: 0.0
  replicas_to_aggregate: 8
  max_number_of_boxes: 100
  unpad_groundtruth_tensors: false
  fine_tune_checkpoint_type: "detection"
  fine_tune_checkpoint_version: V2
}
train_input_reader {
  label_map_path: "/Users/Aida/Documents/GitHub/Rep2/WDS/data/label_map.pbtxt"
  tf_record_input_reader {
    input_path: "/Users/Aida/Documents/GitHub/Rep2/WDS/data/train_labels.record"
  }
}
eval_config {
  metrics_set: "coco_detection_metrics"
  use_moving_averages: false
}
eval_input_reader {
  label_map_path: "/Users/Aida/Documents/GitHub/Rep2/WDS/data/label_map.pbtxt"
  shuffle: false
  num_epochs: 1
  tf_record_input_reader {
    input_path: "/Users/Aida/Documents/GitHub/Rep2/WDS/data/test_labels.record"
  }
}
```

Figure 15.19: Creating model:19

```
%%writefile $rep_path/WDS/our_models/our_SSD_MobileNet_V2_FPNLite_640x640_B/pipeline.config
model {
  ssd {
    num_classes: 2
    image_resizer {
      fixed_shape_resizer {
        height: 640
        width: 640
      }
    }
    feature_extractor {
      type: "ssd_mobilenet_v2_fpn_keras"
      depth_multiplier: 1.0
      min_depth: 16
      conv_hyperparams {
        regularizer {
          l2_regularizer {
            weight: 3.9999998989515007e-05
          }
        }
        initializer {
          random_normal_initializer {
            mean: 0.0
            stddev: 0.009999999776482582
          }
        }
        activation: RELU_6
        batch_norm {
          decay: 0.996999979019165
          scale: true
          epsilon: 0.0010000000474974513
        }
      }
      use_depthwise: true
      override_base_feature_extractor_hyperparams: true
      fpn {
        min_level: 3
        max_level: 7
        additional_layer_depth: 128
      }
    }
    box_coder {
      faster_rcnn_box_coder {
        y_scale: 10.0
        x_scale: 10.0
        height_scale: 5.0
        width_scale: 5.0
      }
    }
```

Figure 15.20: Creating model:20

```
    matcher {
      argmax_matcher {
        matched_threshold: 0.5
        unmatched_threshold: 0.5
        ignore_thresholds: false
        negatives_lower_than_unmatched: true
        force_match_for_each_row: true
        use_matmul_gather: true
      }
    }
    similarity_calculator {
      iou_similarity {
      }
    }
    box_predictor {
      weight_shared_convolutional_box_predictor {
        conv_hyperparams {
          regularizer {
            l2_regularizer {
              weight: 3.9999998989515007e-05
            }
          }
          initializer {
            random_normal_initializer {
              mean: 0.0
              stddev: 0.009999999776482582
            }
          }
          activation: RELU_6
          batch_norm {
            decay: 0.996999979019165
            scale: true
            epsilon: 0.0010000000474974513
          }
        }
        depth: 128
        num_layers_before_predictor: 4
        kernel_size: 3
        class_prediction_bias_init: -4.599999904632568
        share_prediction_tower: true
        use_depthwise: true
      }
    }
```

Figure 15.21: Creating model:21

```
    anchor_generator {
      multiscale_anchor_generator {
        min_level: 3
        max_level: 7
        anchor_scale: 4.0
        aspect_ratios: 1.0
        aspect_ratios: 2.0
        aspect_ratios: 0.5
        scales_per_octave: 2
      }
    }
    post_processing {
      batch_non_max_suppression {
        score_threshold: 9.99999993922529e-09
        iou_threshold: 0.6000000238418579
        max_detections_per_class: 100
        max_total_detections: 100
        use_static_shapes: false
      }
      score_converter: SIGMOID
    }
    normalize_loss_by_num_matches: true
    loss {
      localization_loss {
        weighted_smooth_l1 {
        }
      }
      classification_loss {
        weighted_sigmoid_focal {
          gamma: 2.0
          alpha: 0.25
        }
      }
      classification_weight: 1.0
      localization_weight: 1.0
    }
    encode_background_as_zeros: true
    normalize_loc_loss_by_codesize: true
    inplace_batchnorm_update: true
    freeze_batchnorm: false
  }
}
```

```
train_config {
  batch_size: 8
  data_augmentation_options {
    random_horizontal_flip {
    }
  }
  data_augmentation_options {
    random_crop_image {
      min_object_covered: 0.0
      min_aspect_ratio: 0.75
      max_aspect_ratio: 3.0
      min_area: 0.75
      max_area: 1.0
      overlap_thresh: 0.0
    }
  }
  sync_replicas: true
  optimizer {
    momentum_optimizer {
      learning_rate {
        cosine_decay_learning_rate {
          learning_rate_base: 0.07999999821186066
          total_steps: 50000
          warmup_learning_rate: 0.026666000485420227
          warmup_steps: 1000
        }
      }
      momentum_optimizer_value: 0.8999999761581421
    }
    use_moving_average: false
  }
  fine_tune_checkpoint: "/Users/Aida/Documents/GitHub/Rep2/WDS/pre-trained-models/ssd_mobilenet_v2_fpnlite_640x640_coco17_tpu-8/checkpoint/ckpt-0"
  num_steps: 50000
  startup_delay_steps: 0.0
  replicas_to_aggregate: 8
  max_number_of_boxes: 100
  unpad_groundtruth_tensors: false
  fine_tune_checkpoint_type: "detection"
  fine_tune_checkpoint_version: V2
}
train_input_reader {
  label_map_path: "/Users/Aida/Documents/GitHub/Rep2/WDS/data/label_map.pbtxt"
  tf_record_input_reader {
    input_path: "/Users/Aida/Documents/GitHub/Rep2/WDS/data/train_labels.record"
  }
}
eval_config {
  metrics_set: "coco_detection_metrics"
  use_moving_averages: false
}
eval_input_reader {
  label_map_path: "/Users/Aida/Documents/GitHub/Rep2/WDS/data/label_map.pbtxt"
  shuffle: false
  num_epochs: 1
  tf_record_input_reader {
    input_path: "/Users/Aida/Documents/GitHub/Rep2/WDS/data/test_labels.record"
  }
}
```

Figure 15.23: Creating model:23

```
# to verify write
pipeline_fname=rep_path+'/WDS/our_models/our_SSD_MobileNet_V2_FPNLite_640x640_B/pipeline.config'
```

✓ 0.0s

```
# Checking the config file after editing
!cat {pipeline_fname}
```

Figure 15.24: Creating model:24

tensor board

```
#tensor board is used to track model while training
%pip install -U tensorboard
```

```
#strting tensor board
#run in ternimal parelle to traning to see progress hwile training
%cd $rep_path/WDS
model_dir=rep_path+'/WDS/our_models/our_SSD_MobileNet_V2_FPNLite_640x640_B'
!tensorboard --logdir=model_dir

#For upload to website to view remotly
!tensorboard dev upload --logdir \
    'model_dir'
```

Figure 15.25: Creating model:25

training

```
#defining model directory
model_dir=rep_path+'/WDS/our_models/our_SSD_MobileNet_V2_FPNLite_640x640_B'
model_dir
```

```
#defining model directory

pipeline_config_path=rep_path+'/WDS/our_models/our_SSD_MobileNet_V2_FPNLite_640x640_B/pipeline.config'
pipeline_config_path
```

```
%cd $rep_path/WDS/models/research
#start training
!python3 /Users/Aida/Documents/GitHub/Rep2/WDS/models/research/object_detection/model_main_tf2.py  \
    --pipeline_config_path={pipeline_config_path} \
    --model_dir={model_dir} \
    --alsologtostderr \
    --num_train_steps={num_steps} \
    --num_eval_steps={num_eval_steps}
```

Figure 15.26: Creating model:26

Evaluation

```
#dependancy of cocoapi
%pip install chardet
```

```
#run in terminal while training to evaluate over time
#using cocoapi metrics
%cd $rep_path/WDS
!python $rep_path/WDS/models/research/object_detection/model_main_tf2.py \
    --model_dir={model_dir}\
    --pipeline_config_path={pipeline_config_path} \
    --checkpoint_dir={model_dir}
```

Figure 15.27: Creating model:27

15.2 Implementing model

```
#REDEFINE to use easily on differnt computers, define to path this file is

#path for our github repositiory on local computer
rep_path='/Users/User1/Documents/GitHub/Rep2'

#Check if path correct & working in this path
%cd $rep_path
```

```
#select camera
cam_varr=0 #webcam
#cam_varr="rtmp://live.restream.io/live/re_6588559_f67e5f9e60fe5b825de6" #https://app.restream.io/
```

Figure 15.28: Implementing model:0

Package to install. Once per envrionment

```
#dependancy of face recognition
%pip install CMake
%pip install dlib
```

```
#installing open cv face recognition api
!pip3 install face_recognition
```

```
#API form Model Garden for TensorFlow, needed for object detection API
#only need to run once
%cd $rep_path/WDS
!git clone https://github.com/tensorflow/models/
%cd $rep_path/WDS/models/research

#proto buffer compilation, Protobufs used to conifure modele and trainning parapemetres
!protoc object_detection/protos/*.proto --python_out=.

#installing objeect dection api
%cp object_detection/packages/tf2/setup.py . #tf2

!python -m pip install .
```

Figure 15.29: Implementing model:1

Imports. Every kerrnel

```
#IMPORTING MODEL AND STARTING WEBCAM


import numpy as np
import time
import cv2  #real-time computer vision
import os
import face_recognition #libary for face rec

#to supperress tf logging
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
import tensorflow as tf
from object_detection.utils import label_map_util
from object_detection.utils import config_util
from object_detection.utils import visualization_utils as viz_utils
from object_detection.builders import model_builder
#supress tf logging
tf.get_logger().setLevel('ERROR')

#for email
import smtplib
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart
```

```
#confirm cv2 version
import cv2
print(cv2.__version__)
```

```
print(cv2.getBuildInformation())
```

Figure 15.30: Implementing model:2

Set up

```
#learning face from sample image
#excpetion for when face couldnt be detected in sample image
try:
    gaurd01_pic = face_recognition.load_image_file("gaurd_01.jpg")
    gaurd01_encoding = face_recognition.face_encodings(gaurd01_pic)[0]
except IndexError:
    print("Couldn't find face in sample image")
    quit()
```

Figure 15.31: Implementing model:3

```
#array of face encodings
known_gaurd_encodings = [
    gaurd01_encoding
]

#array of face labels
known_gaurd_label = [
    "gaurd01"
]

#intalize varribles
face_locations = []
face_encodings = []
face_label = []

#for analyzing every other frame for speed preformance (face)
analyze_this_frame = True

#skping frames for performance (weapon)
frames_per_cycle_w=2
frame_count_w=1 #offset from face

###defining paths of model##
label_path=rep_path+'/WDS/data/label_map.pbtxt'
configuration_path=rep_path+'/WDS/our_exported-models/our_model_2/pipeline.config'
checkpoint_path=rep_path+'/WDS/our_exported-models/our_model_2/checkpoint'

print(label_path)
print(configuration_path)
print(checkpoint_path)
```

Figure 15.32: Implementing model:4

```
####loading model####

#using GPU dynamic memory alloaton
gpus = tf.config.experimental.list_physical_devices('GPU')
for gpu in gpus:
    tf.config.experimental.set_memory_growth(gpu, True)

#builging dection model and loading pipleine configuration
configs = config_util.get_configs_from_pipeline_file(configuration_path)
model_config = configs['model']
detection_model = model_builder.build(model_config=model_config, is_training=False)

#restoring checkpoint
ckpt = tf.compat.v2.train.Checkpoint(model=detection_model)
ckpt.restore(os.path.join(checkpoint_path, 'ckpt-0')).expect_partial()

#definig detection function too find wepaons in video
@tf.function
def detect_fn(image):

    image, shapes = detection_model.preprocess(image)
    prediction_dict = detection_model.predict(image, shapes)
    detections = detection_model.postprocess(prediction_dict, shapes)

    return detections, prediction_dict, tf.reshape(shapes, [-1])

#for ploting loading labbel map dta
cat_index = label_map_util.create_category_index_from_labelmap(label_path,use_display_name=True)
```

Figure 15.33: Implementing model:5

```python
#initial for messagee
last_email_sent_time = 0
current_time = time.time()

# Define the send_email function
def send_email(subject, body, from_email, to_email, password):
    # Create a MIMEMultipart message
    message = MIMEMultipart()
    message["From"] = from_email
    message["To"] = to_email
    message["Subject"] = subject

    # Add body to email
    message.attach(MIMEText(body, "plain"))

    # Create SMTP session
    server = smtplib.SMTP('smtp.gmail.com', 587)
    server.starttls()
    server.login(from_email, password)

    # Send email
    text = message.as_string()
    server.sendmail(from_email, to_email, text)
    server.quit()
```

Figure 15.34: Implementing model:6

```python
####starting camera###

vid = cv2.VideoCapture(cam_varr) #video feed webcame

#geting frame size
frame_width = int(vid.get(3))
frame_height = int(vid.get(4))
size = (frame_width, frame_height)

#for saving video
#result = cv2.VideoWriter('WDSrecording.avi', cv2.VideoWriter_fourcc(*'MJPG'),10, size)

while(vid.isOpened()): #video capture is initizalized
    startTime=time.time()
    ret, frame=vid.read() #reads video frame: ret=bool if available frame, frame=image array
    frame_expanded = np.expand_dims(frame, axis=0)#match feed to deimisons of model
```

Figure 15.35: Implementing model:7

```python
######DETECTING ELEMENTS##########
##########DETECTING FACE######

#for analyzing every other frame for better speed preformance
if analyze_this_frame:
    #to reducee frame size for speed perforomancee
    tiny_frame = cv2.resize(frame, (0, 0), fx=0.25, fy=0.25)

    # RGB(facee rec uses) <-- BGR(open cv)
    rgb_tiny_frame = cv2.cvtColor(tiny_frame, cv2.COLOR_BGR2RGB)

    # find locations of faces in frame
    face_locations = face_recognition.face_locations(rgb_tiny_frame)

    #getting encodings of located faces
    face_encodings = face_recognition.face_encodings(rgb_tiny_frame, face_locations)

    face_label = []
    for face_encoding in face_encodings:
        # comparing facee encodings found iin frame to our face encodings
        matches = face_recognition.compare_faces(known_gaurd_encodings, face_encoding)
        name = "Unknown"

        #assingingg to the most similar face
        face_distances = face_recognition.face_distance(known_gaurd_encodings, face_encoding)
        best_match_index = np.argmin(face_distances)
        if matches[best_match_index]:
            name = known_gaurd_label[best_match_index]
        face_label.append(name)

#Alternatng frames for face
analyze_this_frame = not analyze_this_frame
##########END DETECTING FACE######

##########DETECTING WEAPON######
if (frame_count_w%frames_per_cycle_w)==0:
  in_tensor = tf.convert_to_tensor(np.expand_dims(frame, 0), dtype=tf.float32)
  detections, predictions_dict, shapes = detect_fn(in_tensor)
  label_id_offset = 1

#to skip certian frames
frame_count_w=frame_count_w+1
##########END DETECTING WEAPON######

#######END DETECTING ELEMENTS##########
```

Figure 15.36: Implementing model:8

```
# Showing results of face
for (top, right, bottom, left), name in zip(face_locations, face_label):
    # with scale of orginal size
    top *= 4
    right *= 4
    bottom *= 4
    left *= 4

    # create box
    cv2.rectangle(frame, (left, top), (right, bottom), (0, 0, 255), 2)

    # labeling box
    cv2.rectangle(frame, (left, bottom - 35), (right, bottom), (0, 0, 255), cv2.FILLED)
    ##setting font
    font = cv2.FONT_HERSHEY_DUPLEX
    ##adding name
    cv2.putText(frame, name, (left + 6, bottom - 6), font, 1.0, (255, 255, 255), 1)

# Check if any weapons are detected
if  current_time - last_email_sent_time > 300:
    # Check if it's been more than 5 minutes since the last email was sent
    if detections['detection_scores'][0].numpy().any()>=0.5:
        # Send an email alert
        subject = "Weapon detected!"
        body = "A weapon has been detected in the building. Please investigate."
        from_email = "seniordesignaud2023@gmail.com"
        to_email = "eng.abdullahmourad@gmail.com"
        password = "ctvlttuynqzxtmbb"
        send_email(subject, body, from_email, to_email, password)
        last_email_sent_time = current_time

#dRAWING BOX OF DEDETED OBJECT(weaapon)
viz_utils.visualize_boxes_and_labels_on_image_array(
    #with_detec,
    frame,
    detections['detection_boxes'][0].numpy(),
    (detections['detection_classes'][0].numpy() + label_id_offset).astype(int),
    detections['detection_scores'][0].numpy(),
    cat_index,
    use_normalized_coordinates=True,
    max_boxes_to_draw=300,
    min_score_thresh=.20,
    agnostic_mode=False)
```

Figure 15.37: Implementing model:9

```
# Showing results of face
for (top, right, bottom, left), name in zip(face_locations, face_label):
    # with scale of orginal size
    top *= 4
    right *= 4
    bottom *= 4
    left *= 4

    # create box
    cv2.rectangle(frame, (left, top), (right, bottom), (0, 0, 255), 2)

    # labeling box
    cv2.rectangle(frame, (left, bottom - 35), (right, bottom), (0, 0, 255), cv2.FILLED)
    ##setting font
    font = cv2.FONT_HERSHEY_DUPLEX
    ##adding name
    cv2.putText(frame, name, (left + 6, bottom - 6), font, 1.0, (255, 255, 255), 1)

# Check if any weapons are detected
if  current_time - last_email_sent_time > 300:
    # Check if it's been more than 5 minutes since the last email was sent
    if detections['detection_scores'][0].numpy().any()>=0.5:
        # Send an email alert
        subject = "Weapon detected!"
        body = "A weapon has been detected in the building. Please investigate."
        from_email = "seniordesignaud2023@gmail.com"
        to_email = "eng.abdullahmourad@gmail.com"
        password = "ctvlttuynqzxtmbb"
        send_email(subject, body, from_email, to_email, password)
        last_email_sent_time = current_time

#dRAWING BOX OF DEDETED OBJECT(weaapon)
viz_utils.visualize_boxes_and_labels_on_image_array(
    frame,
    detections['detection_boxes'][0].numpy(),
    (detections['detection_classes'][0].numpy() + label_id_offset).astype(int),
    detections['detection_scores'][0].numpy(),
    cat_index,
    use_normalized_coordinates=True,
    max_boxes_to_draw=300,
    min_score_thresh=.20,
    agnostic_mode=False)
```

Figure 15.38: Implementing model:10

```
    #Displaying footage to user
    if not ret:
        print("Exting. Didnt get frame")
        break
    else:
        #result.write(frame) #saving footage

        #display video with results, resize to orginal ratio
        cv2.imshow('WDS footage', cv2.resize(frame, size))

        #press q to quit
        if cv2.waitKey(1) == ord('q'):
            break

###ending webcam###
vid.release()
cv2.destroyAllWindows()
```

Figure 15.39: Implementing model:11

Bibliography

[1] AWS Editor, "What Is A Neural Network?," *Amazon,* 1978. [Online]. Available: https://aws.amazon.com/what-is/neural-network/. [Accessed: 29-Jan-2023].

[2] IBM Editor, "What are convolutional neural networks?," *IBM.* [Online]. Available: https://www.ibm.com/topics/convolutional-neural-networks. [Accessed: 29-Jan-2023].

[3] IBM Editor, "What is machine learning?," *IBM.* [Online]. Available: https://www.ibm.com/topics/machine-learning. [Accessed: 29-Jan-2023].

[4] K. Haciefendioglu, H. B. Basaga, and G. Demir, "Automatic detection of earthquake-induced ground failure effects through faster R-CNN deep learning-based object detection using satellite images," *Natural Hazards,* vol. 105, no. 1, pp. 383-403, 2020.

[5] L. Hardesty, "Explained: Neural networks," *Massachusetts Institute of Technology,* 14-Apr-2017. [Online]. Available: https://news.mit.edu/2017/explained-neural-networks-deep-learning-0414. [Accessed: 29-Jan-2023].

[6] Bhatti, Muhammad Khan, Muhammad Gufran Aslam, Masood Fiaz, Muhammad. (2021). "Weapon Detection in Real-Time CCTV Videos using Deep Learning". IEEE Access. PP. 1-1. 10.1109/ACCESS.2021.3059170.

[7] Ahmed, S., Bhatti, M. T., Khan, M. G., LÃ¶vstrÃ¶m, B., Shahid, M. (2022). Development and Optimization of Deep Learning Models for Weapon Detection in Surveillance Videos. Applied Sciences, 12(12), 5772. MDPI AG. Retrieved from http://dx.doi.org/10.3390/app12125772

[8] "Concealed Weapons Detection (CÂ» Avigilon," www.avigilon.com, Jun. 09, 2021. https://www.avigilon.com/products/motorola-solutions-ecosystem/concealed-weapons-detection

[9] "AI Gun Detection Technology - How It Works," ZeroEyes. https://zeroeyes.com/technology/

[10] "Weapon Detection â DaSCI,â dasci.es. https://dasci.es/transferencia/open-data/24705/

[11] J. Solawetz, âMean average precision (MAP) in object detection,â Roboflow Blog, 25-Nov-2022. [Online]. Available: https://blog.roboflow.com/mean-average-precision/.