Augustine Igbinidu-Uwuigbe

Agnet Object-Detection and Alert System with TensorFlow-Serving and Agnet-API

Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Bachelor’s Thesis

30 August 2022

Abstract

Author:	Augustine Igbinidu-Uwuigbe
Title:	Agnet Object-Detection and Alert System with TensorFlow-Serving and Agnet-API
Number of Pages:	45 pages
Date:	30 August 2022
Degree:	Bachelor of Engineering
Degree Programme:	Information Technology
Professional Major:	Internet of Things and Cloud Computing
Supervisors:	Serge Delmas, System Integration Specialist Marko Uusitalo, Senior Lecturer

Recently, Machine Learning has played a major role in the field of science and technology. Object detection in Computer vision systems has gained a lot of use in many industries and is still being developed for many use cases today. It is now an essential technology for many monitoring systems, especially for detecting threats or tracking items.

Agnet as a secure end-to-end communication solution for Airbus can be used as a monitoring system for cameras and drones. Having an integrated and deployable smart monitoring system powered by computer vision technology for risk reporting will bring value to Agnet.

This thesis aimed to investigate the usage of Computer Vision to create a smart monitoring and risk-reporting system for Agnet. To achieve these objectives the theoretical structure will include every step taken in building this computer vision and alerting system: acquisition, processing, model training, model deployment, inference, and risk reporting. This will give an extensive perspective of the key parts and their application. This is followed by the description of Agnet-API, which is the application programming interface for Agnet. Finally, a proof-of-concept Computer Vision and riskreporting system to demonstrate its practicality in a production environment.

Based on the studies illustrated in this paper, it can be concluded that Computer Vision through Agnet-API is a viable and cheap smart-monitoring solution for Organizations. The desired objectives were fulfilled and the applicability of this solution to several communication systems is provided.

Keywords: Computer Vision, Agnet-API, Smart-monitoring, Risk-reporting

Contents

List of Abbreviations

SMS:	Smart Monitoring System. A Computer-vision based system that acts as a monitoring solution for video devices.
API:	Application Programming Interface.
ML:	Machine Learning. The act of machines learning from past data in other to make smart decisions without being implicitly programmed.
CV:	Computer Vision. Defines the tracking and detecting of objects within video sources or Images.
TFS:	TensorFlow Serving. A Rest-API-based service offered by TensorFlow for easy Model deployment.
NN:	Neural Networks. Network of neurons and nodes that transmit information through signals.
REST:	Representational State Transfer. Communication protocol for API services based on HTTP (Hypertext Transfer Protocol)
TPU:	Tensor Processing Unit. Dedicated unit built by Google developers specifically for neural network machine learning. Performs optimally on Google’s TensorFlow software.
CPU:	Central Processing Unit. A core processing unit that performs basic operations within a computer.

GPU:	Graphics Processing Unit. An electronic circuit designed to render graphic images at a more rapid rate. It is meant to process several data simultaneously which makes it useful for machine learning.
VM:	Virtual Machine. Allows the execution of an Operating system that acts as a separate computer system within an application window on a desktop or host. Several of these instances can be run on a single host with each acting as a separate computer system.
MC:	Mission Critical. This describes all necessary services or operations that are required for normal operations to be executed either in businesses or in any mission.
MCPTT:	Mission Critical Push to Talk. Mission-critical voice and audio transmission using the Push to Talk service.
MCDATA:	Mission Critical Data. Data services (multimedia, message, video, etc.) for Mission-critical use.
SSD:	Single Shot Detector. A Machine Learning model architecture.
TP:	True Positives. The number of correct detections identified by a model.
FP:	False Positives. The number of correct detections not identified by a model.
MaP:	Mean Average Precision. This is the mean value for the average precision of each class of object to be detected.
AP:	Average Precision. This means how well a Machine Learning model can identify True Positives (TP) out of all positive predictions across a range of different classes.

1 introduction

Machine Learning (ML) has been very active in the technology industries and every day more innovative ideas are developed to further advance the potential it brings to modern-day technology. Deep learning as an aspect of ML has made it possible for machines to learn similarly to how the brain works.

Deep Learning (DL) technology has made it possible to build Computer Vision solutions which are essential for many monitoring systems, especially for detecting threats or tracking items. Having an integrated and deployable smart monitoring system powered by machine learning algorithms for risk reporting can be a relevant asset for large enterprises.

Agnet as a secure end-to-end Mission-critical broadband solution could benefit from having an integrated Smart Monitoring System (SMS), which utilizes Computer Vision Technology.

The objective of this thesis is as follows: (1) Explore the different components of a simple computer vision application (2) Detail how TensorFlow and OpenCV can be used in Machine Learning, deep learning, and Neural Networks to create a smart monitoring and alerting system (3) Detail and Demonstrate how to Integrate a computer vision system into other solutions using Rest-API.

2 Background Knowledge

The development of the Agnet SMS (Smart Monitoring System) would be impossible without the necessary theoretical knowledge of the technologies involved. Some of the major topics involved are discussed in the chapter as follows.

2.1 Machine-Learning

Machines generally are not intelligent. They were originally designed to perform a designated task without being able to learn from their experience.

Machine Learning is the aspect of artificial intelligence that aims to enable machines to do designated tasks and learn from the mistakes made without being explicitly programmed. They can learn using intelligent software. This intelligent software comprises algorithms and models that define the structure or patterns at which the machine learns or behaves.[3]

ML gives computers the ability to learn from data.

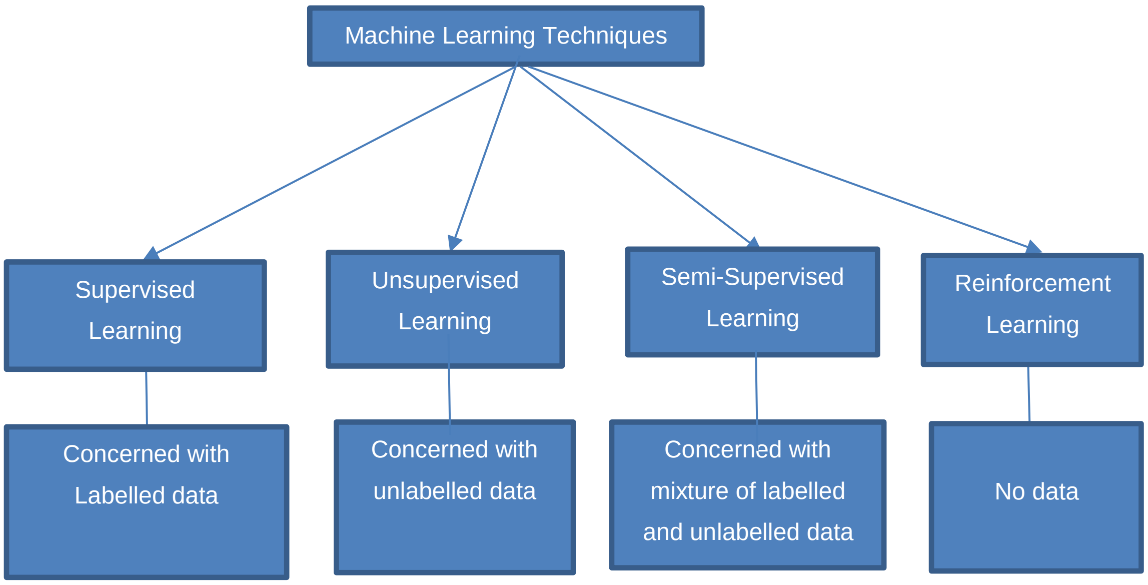

Machine Learning Techniques
Supervised Learning
Unsupervised Learning
Semi-Supervised Learning
Reinforcement Learning
Concerned with Labelled data
Concerned with unlabelled data
Concerned with mixture of labelled and unlabelled data
No data

Figure 1. Different machine learning models and their required data [3]

2.1.1 Supervised-Learning

Supervised Learning as the name implies requires the supervision of the ML processes involved when training the model. This involves feeding the model with past data and providing the recorded outcomes(labels) of those data. The model function formed will be able to predict future outcomes based on the patterns it learned from the previously labeled data. This is the most common method of ML that involves predicting certain outcomes based on inputs.

2.1.2 Unsupervised-Learning

Unsupervised Learning deals with the aspect of ML, where the algorithm is only provided with data only. In this process, there is no teacher to instruct the model, and no desired output is provided during the training phase. The algorithm is meant to extract knowledge from the data. A common use case is in clustering of data with similar features.

2.1.3 Semi-supervised Learning

This is a type of ML that is between supervised and unsupervised learning. It deals with problems where only a few labeled data are available, with a large portion of unlabelled data. It is mostly used when the data with labeling examples are difficult or expensive to produce.

2.1.4 Reinforcement-Learning

This is an area of ML where the algorithm involves a factor that must determine the best course of action in a situation or environment. These series of decisions come with a reward or punishment. This form of ML helps decide the best series of actions to maximize reward. Like Unsupervised Learning, it does not require any teaching or instruction.

2.2 Neural networks

Neural Networks (NN) include a common way to represent (slightly complex) algorithms connecting input data to output data. It involves several similar units called neurons, performing simple mathematical tasks, and aligned in connected layers. Each neuron takes as input different outputs from the previous layer.
Figure 2 illustrates the transmission of signals in the human cell.

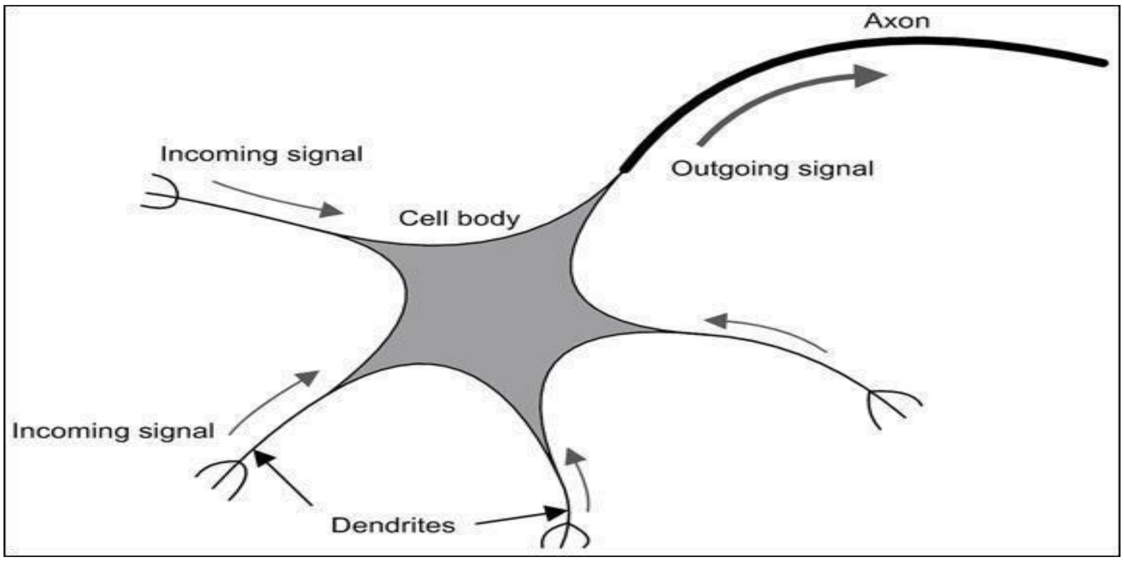


Figure 2. Nerve cells receive incoming signals and produce output signals through the axon [16]

As displayed in Figure 2, the human cell consists of an axon and several dendrites. The dendrites receive a signal and the axon transmit the output to other cell body.
In Artificial Neural Networks the nodes are the dendrites while the Neuron is the Axon. Several inputs are passed into the neuron where the Neuron produces output based on those inputs using some mathematical function(algorithm).

The figure below shows how a simple Artificial Neural Network model receives several inputs and produces some output.

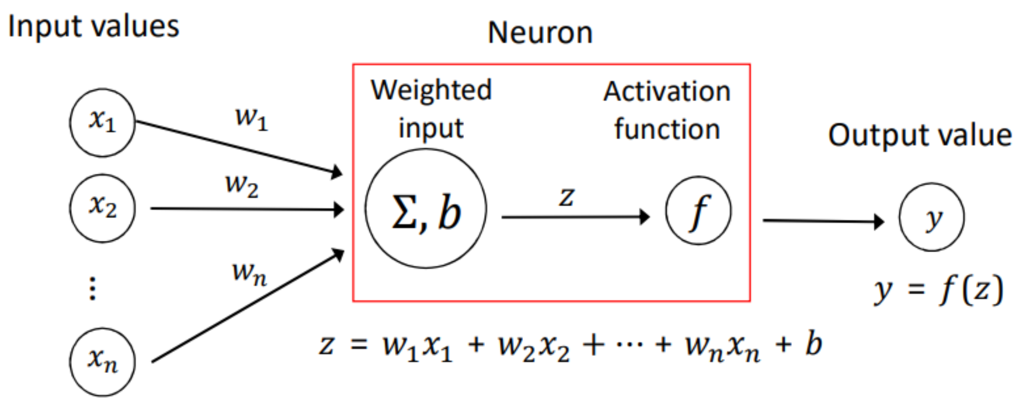


Figure 3. Artificial Neuron with several inputs and a single output [16]

2.3 Deep Learning

DL (Deep Learning) consists of an advanced form of neural network that involves a more complex network of interconnected layers that comprises different nodes. In a Neural Network, thousands to millions of neurons in the hidden layer of a network are broken down and processed into fewer output neurons. This form of NN (Neural Network) is called Deep Learning. [3]
The figure below shows the structure of a Deep Neural Network.

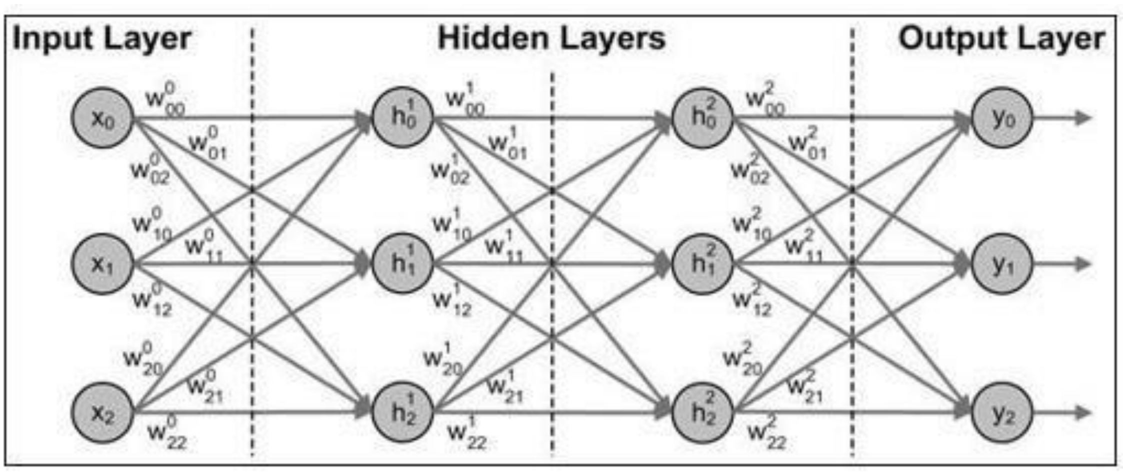

Input Layer
Hidden Layers
Output Layer
x_0
x_1
x_2
h^1_0
h^1_1
h^1_2
h^2_0
h^2_1
h^2_2
y_0
y_1
y_2
w^0_{00} w^0_{01} w^0_{02} w^0_{10} w^0_{11} w^0_{12} w^0_{20} w^0_{21} w^0_{22}
w^1_{00} w^1_{01} w^1_{02} w^1_{10} w^1_{11} w^1_{12} w^1_{20} w^1_{21} w^1_{22}
w^2_{00} w^2_{01} w^2_{02} w^2_{10} w^2_{11} w^2_{12} w^2_{20} w^2_{21} w^2_{22}

Figure 4. Structure of a Deep NN (Neural Network) [16]

A Neural Network with Three or more layers is usually referred to as DL (Deep Learning) or Deep Neural Network.

2.4 Computer Vision

CV (Computer Vision) utilizes deep learning technology to extract valuable information from images, videos, and other visual inputs.
This technology is used in Image recognition, object detection, object tracking, and more.
A simple form of Computer Vision is the detection of objects in images. This technology utilizes a process called ‘Feature Extraction’ to understand objects within an image.

Steps Involved in a Basic Computer Vision System

- Image-Acquisition
- Pre-Processing
- Feature Extraction
- Classification

2.4.1 Image-Acquisition

As the name implies, this stage involves gathering necessary image Data required for training and evaluating the classifier which would be used for detecting objects or classifying images.
The quality of the Data used in training a classifier is essential for its performance. The Data used in the Image-Acquisition process is referred to as Training-Data.

2.4.2 Pre-Processing

This stage involves all the modifications performed on the Data before any other form of data processing (e.g., Feature-Extraction) to boost performance of the Machine Learning Process. This is to avoid using garbage data which would result to a bad ML model. The type of Technique performed during this stage depends on the type of data and the kind of model that will be used. [3] Some of the Modifications during this phase are

- Image labeling
- Image-Augmentation
- Resizing
- Grayscale

2.4.3 Feature Extraction

During the processing of an image, a series of conversions is performed on an image to extract the important features that the algorithm will use for making

predictions. It is the transformation of raw data into numerical values that can be processed while retaining the information contained in the original data.
The figure below shows an Image and its processed form showing some extracted features.

2.4.4 Classification

This is the final step of the computer vision system process. Several computations involving algorithms and patterns are applied to the processed data. These algorithms and patterns will generate outputs called Models. The model generated is based on the meaningful information gathered by the algorithm which is used to classify future unseen data (Images).

2.5 TensorFlow

TensorFlow is an Open-Source framework that provides various researchers and Engineers, with easy access to ML tools and services to enable them to build their ML-based applications. It helps form an easy Model-Building and Robust Model deployment to Enable powerful experimentation. TensorFlow is built on the Tensor Framework [8].
This framework was released by Google and backed by google engineers. TensorFlow takes data in multi-dimensional arrays called tensors, which makes for faster computation on systems with high-computation power.

2.5.1 What are Tensors?

A tensor is a multi-dimensional array of data used in mathematical models that expresses neural networks. It is usually a higher-dimension generalization of matric or vector.[7] It is a more complex form of NumPy arrays with a variable number of dimensions.
The figure below suitably describes tensors.

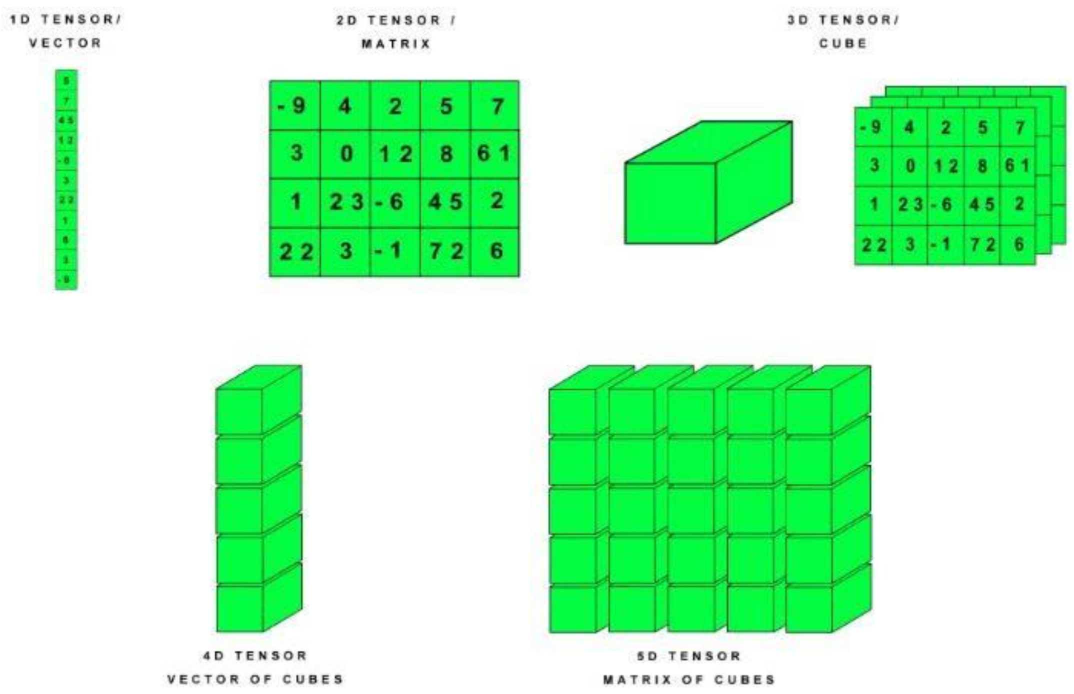


Figure 5. Different representations of tensors [9]

2.5.2 Why Choose TensorFlow?

- **Convenient API**: TensorFlow has an easy-to-use API architecture that enables python developers to use TensorFlow raw, low-level API, or the core API in developing their models. They also can use high-level API libraries for built-in models.[7]

- **Flexible architecture**: One of the best parts of using TensorFlow is the ability to distribute models across CPU, GPU, or TPU processors with minimal code alteration. The framework allows for developers to not only provide large-scale distributed training and inference but also test with other ML models and increase the performance of existing models.[7]

- **Computational Graph model**: TensorFlow uses computational graphs (also called graphs) to represent a series of operations arranged into a graph of nodes to help developers visualize what is going on within the neural network layers. It simply means a combination of nodes that represents the operations being carried out by a model. [7]

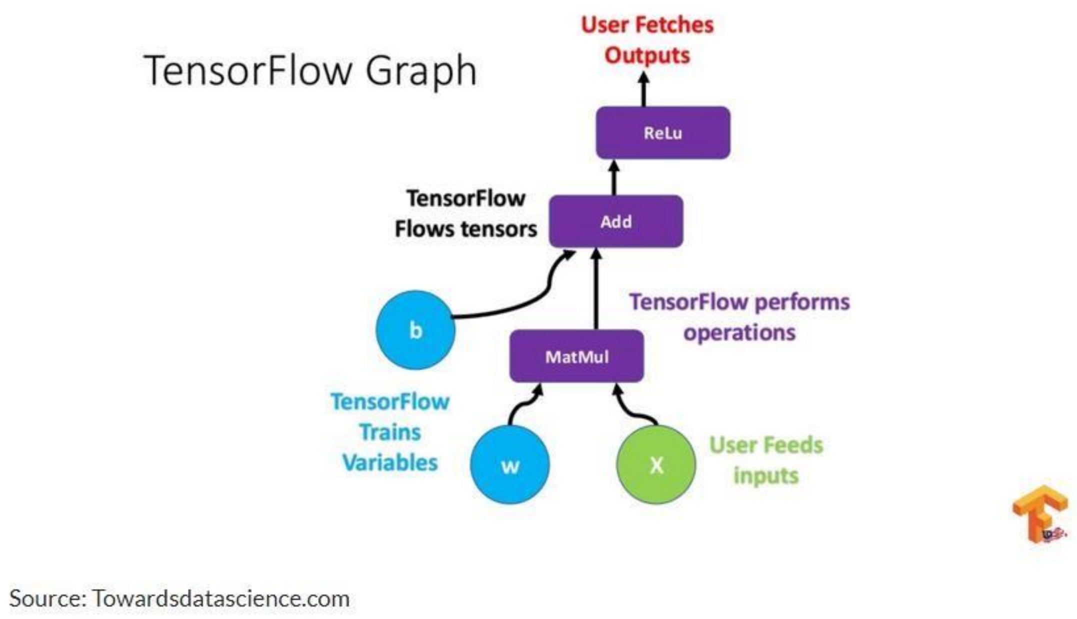


Figure 6: TensorFlow computational Graph [8]

- **Shared Processing**: TensorFlow was designed to support the distribution of processes on different servers. This will help reduce the load by transferring processes from a system with small computational power to another system with higher computational power. Parts of a computational graph can be shared across a cluster of servers with GPUs or with larger memory all on the same network. [7]

- **Performance**: The performance of a deep learning framework is dependent on the fundamental hardware to run optimally at its top performance with lower energy costs. Usually, the original development platform of any framework would obtain the best optimization.

  TensorFlow performs best on Google TPUs, but it also obtains high performance on other platforms ranging from servers, desktops, mobile devices, and embedded systems. [7]

2.6 Rest-API

The Restful API is an application programming interface that allows data exchange between a client and a server using commonly HTTP protocol. REST (Representational State Transfer) API follows the architecture of a web service by utilizing the Client-Server protocol.

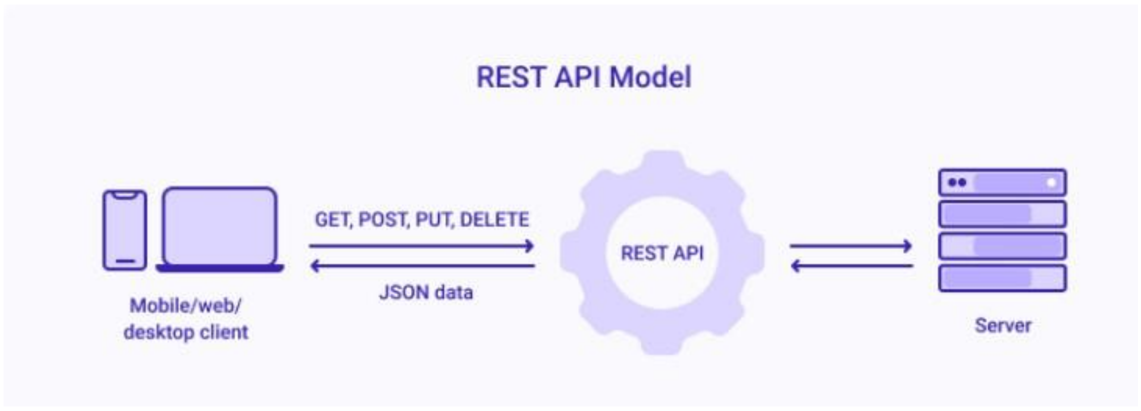


Figure 7. Common REST API Model [17]

As shown in the Figure above, the Web browser or application makes requests for data or commands using the REST-API protocol and receives a response from the server.

2.7 Docker

Docker is an open-source platform that allows for building, deploying, and managing containerized applications (Docker Containers) [11]. The Docker tool packages several applications and services into a containerized environment.

2.7.1 What are Containers?

Containers are a form of virtual machines that consists of isolated processes/resources built into the Linux kernel. Some of the features that make containers virtual-like machines are:

- Control groups - For allocating resources among several processes.
- Namespace – For limiting process access or visibility into other areas or resources within the system. Enabling multiple application elements to distribute the resources of a single instance of the host operating system

(the same way as a hypervisor allows multiple virtual machines to share the host CPU)

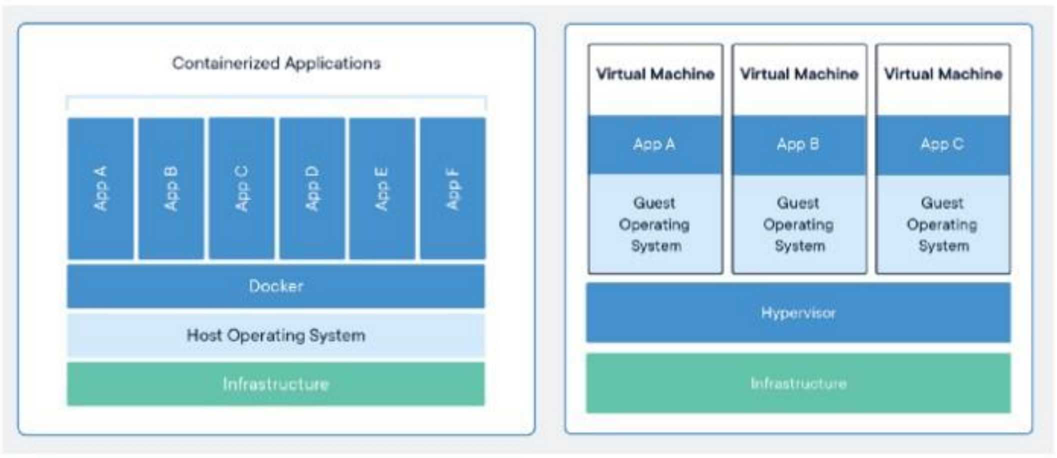


Figure 8: Comparison of containers and virtual machines [12]

2.7.2 Advantages of Docker Containers

1. **Lighter weight:** Containers do not have full-blown Operating systems, unlike Virtual machines. They encompass only the Operating System processes and dependencies required to execute the code. The size of containers is measured in Megabytes (vs. gigabytes in VMS). Hence containers have faster runtime and have optimized start-up times.
2. **Larger Resource Efficiency:** Due to its lightweight, several instances of an application can run simultaneously on the same hardware(host) with minimal resource intensity than VMs. This helps for better efficiency.
3. **Improved Productivity:** Compared to Virtual Machines (VMs), docker containers are faster, and easier to deploy, provision, and restart. These make development operations easier and more agile. It is ideal for the continuous integration and deployment practiced by development engineers.[9]

2.7.3 Components of Docker

1. **Docker-File:** The docker file is a simple text file that defines a list of instructions used in building a docker image.
2. **Docker Images:** This is a read-only executable file that contains all the instructions for creating a docker container. It is like a snapshot in the VM world. It includes all source code as well as the tools and libraries that the application needs to run as a container.
3. **Docker daemon:** This is a service that runs on the operating system. It listens for commands or requests and performs operations accordingly. It is responsible for managing docker objects like docker images, containers, networks, etc.
4. **Docker Hub:** This is a public repository that stores docker images. It is arguably the largest platform for container images.
5. **Docker Compose:** This is a tool that is used to run multi-container docker applications. It allows the creation and running of multiple containers defined within a YAML file. These containers can share resources and communicate with each other. (Dependent containers). [9]

2.7.4 Docker Deployment and Orchestration

Containers can be managed and deployed by several tools, but there are a few tools that are more viable and used by developers.

Docker-compose allows for managing and deploying several docker containers on the same host. While this might solve the challenge of multi-container operations, it does not help in deploying and managing containers across several servers.

While docker has its tool(docker-swarm) used for container orchestration across many servers, Kubernetes has proven to be a more viable solution than dockerswarm. What is Kubernetes?

Kubernetes: This is an open-source container orchestration platform that automates the management, scaling, and deployment of several applications [13].

3 Implementation

The theoretical process that was undertaken before building this computer vision project (proof of concept) has been addressed in the preceding chapters. This chapter aims to show how this knowledge was implemented and to give a better view of the development of the project.

The figure below represents the entire architecture and workflow of this project.

Project-Architecture

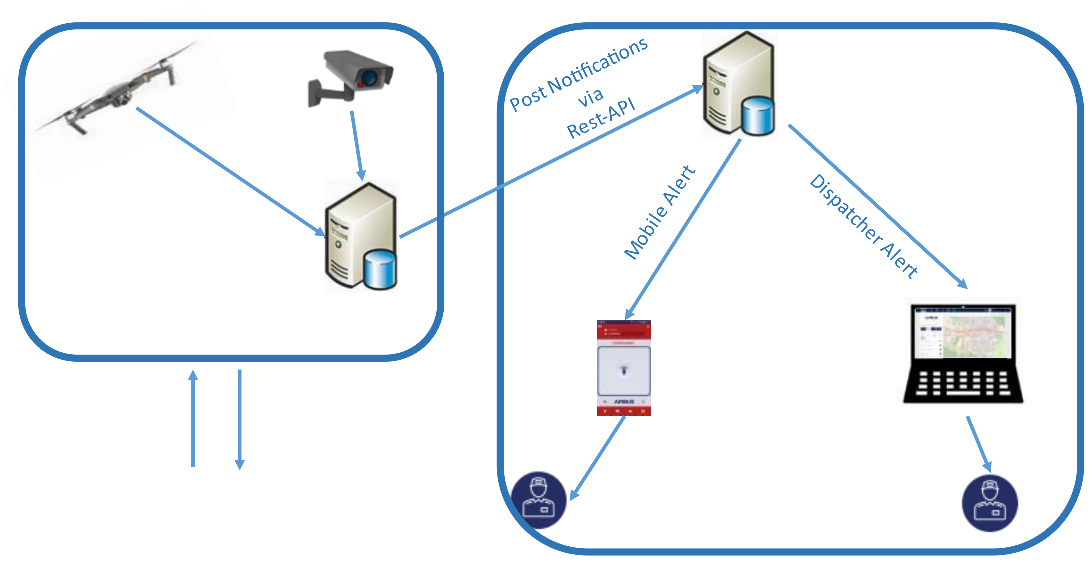


Figure 9. Agnet Computer Vision System Architecture

The project includes the development of a computer vision application that can interact with other applications via API.

Building a CV system follows the same process as many other software systems. There are a few questions that were considered before building an Agnet CV system.

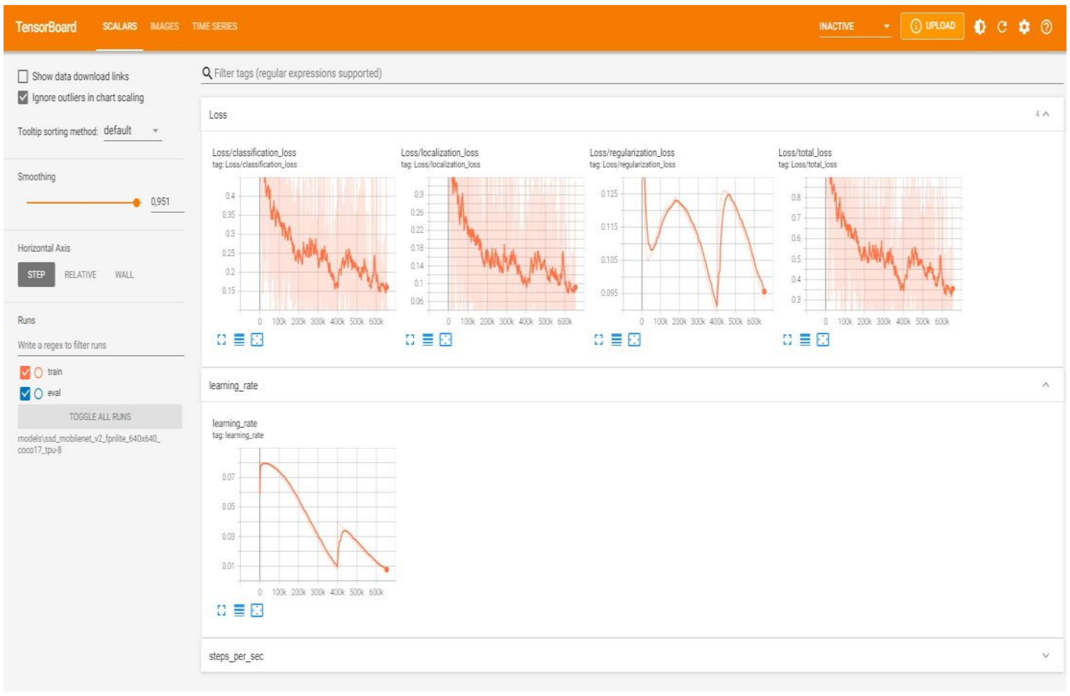
The table below answers some of these questions.

Question	Answer

What is the purpose of the software?	To detect Threats like weapons and send alerts based on detected threats.
What technologies would be used?	Technologies like TensorFlowGPU, OpenCV, Docker, REST-API, etc.
• What environment will this software be built on?	The prototype will be built on UBUNTU Linux and future prototypes would use other platforms like Raspberry-Pi, RHEL8+(To reduce cost of production and facilitate easy deployment)

3.1 Implementation Tools

- TensorFlow-GPU: TensorFlow supports the use of a Graphics card for additional computation power in the model training phase. TensorFlow GPU was installed on the hardware used for training the model. The hardware specifications can be found in section 3.2.1.3.
- Tensor board: This tool is used to help provide detailed measurements and make visualizations during the machine learning workflow. It is used to measure metrics like accuracy and loss. It provides all the necessary tools needed for ML experimentation and model evaluation. Below is a snippet of the Tensor board during model training.



Snippet 1. TensorFlow Visualization Board

- Docker-compose: This tool is used to manage and deploy both the server application (TensorFlow serving) and the client application.

3.2 Solution Implementation

This project consists of Three major steps defined below.

1. Building the Model
2. Developing the application
3. Integrating the application with Agnet

A detailed analysis of each of these steps will be provided as this chapter progresses.

3.2.1 Building The Model

Many Computer vision systems being developed today follow Similar processes. The various steps applied when building this application are described in the following subsections.

Image Acquisition

As described in the earlier chapters of this document, any ML or CV application uses a model. This model requires specific data (Training data), which it will learn from.

The Images used to train this computer vision system were acquired from Kaggle. Kaggle is an online platform that consists of a community of machine learning enthusiasts and data scientists to collaborate and share ideas.[1]

Since the Computer vision system aims for detecting threats like weapons (Pistols, Assault-rifles, etc.) therefore, the necessary Images needed for training this system will consist of fire and guns.

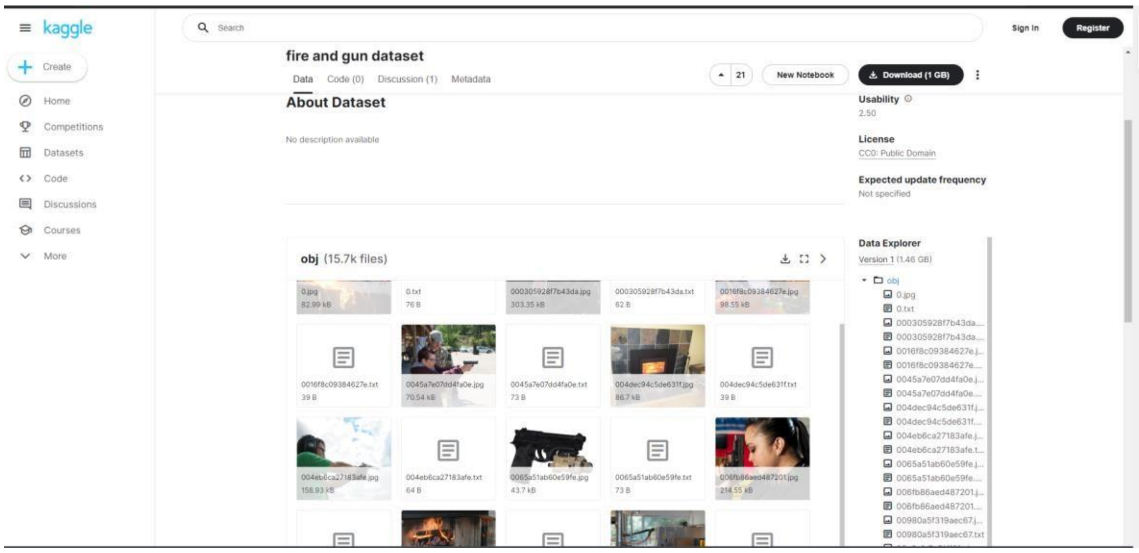


Figure 10. Fire-Gun Dataset Kaggle [1]

The dataset consisted of 7784 images and their respective labels making about 15700 files.

Image Processing

The Dataset were, however, labeled with text files. Since the computer vision system model is going to be based on TensorFlow, some series of conversions and pre-processing was needed to achieve the right data format.

An online ML tool called Robo-flow is used to achieve this.[2]

Roboflow allowed for converting and creating XML labels for each image as shown in the figure below.

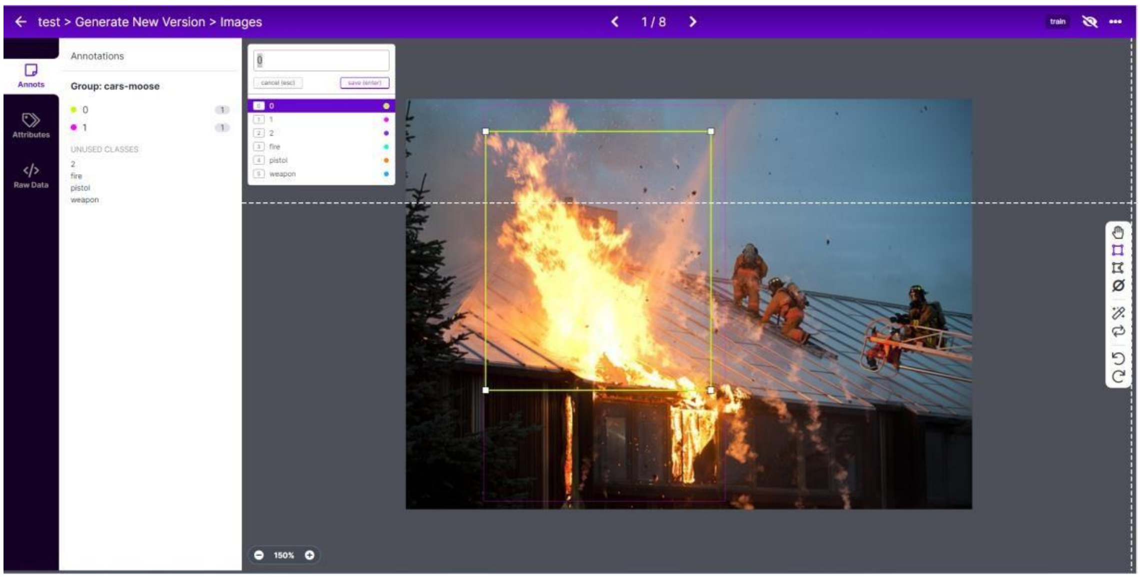
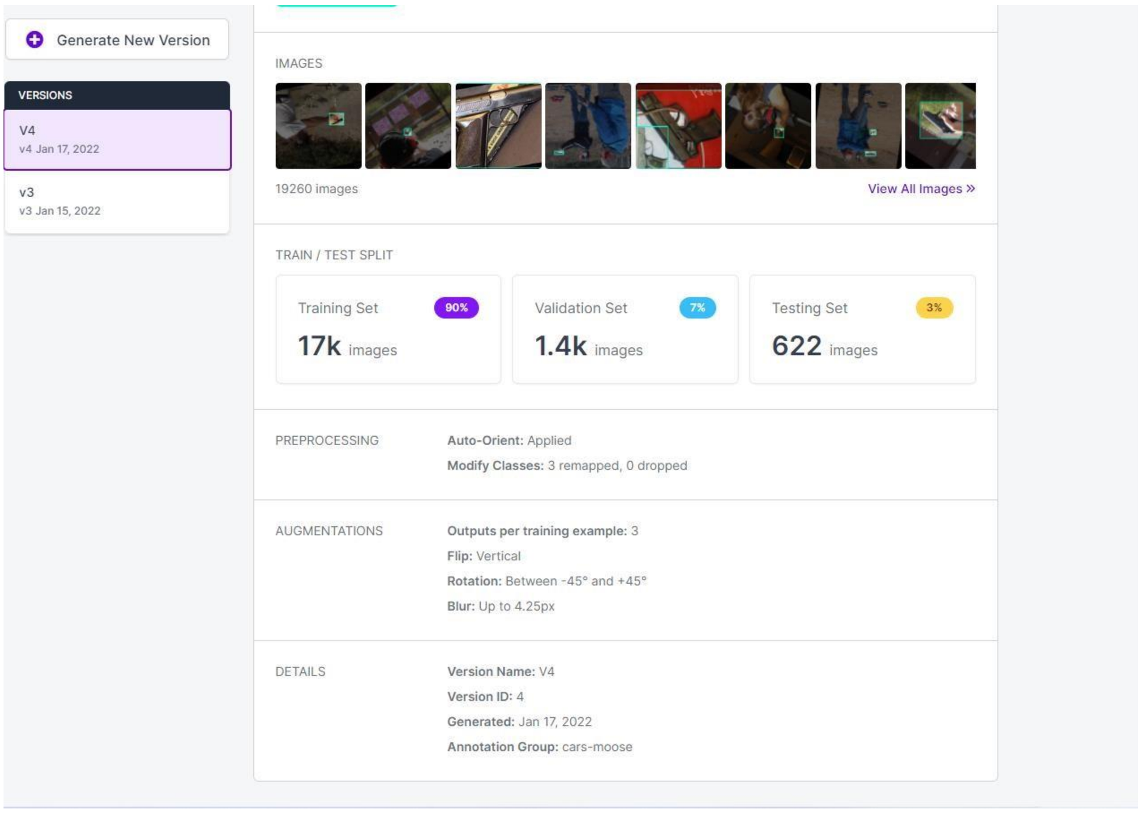


Figure 11. Bounding box images and their respective XML labels [2]

After creating the labels for the Images, the data was split into 75% Training set, 15% Validation set, and 10% Test set. Some data annotations were done to the Training data to increase the number of Images from 5800 to 17000 as shown in figure 12 below.


Generate New Version

VERSIONS

V4
v4 Jan 17, 2022

v3
v3 Jan 15, 2022

IMAGES

19260 images

View All Images »

TRAIN / TEST SPLIT

Training Set 90%
17k images

Validation Set 7%
1.4k images

Testing Set 3%
622 images

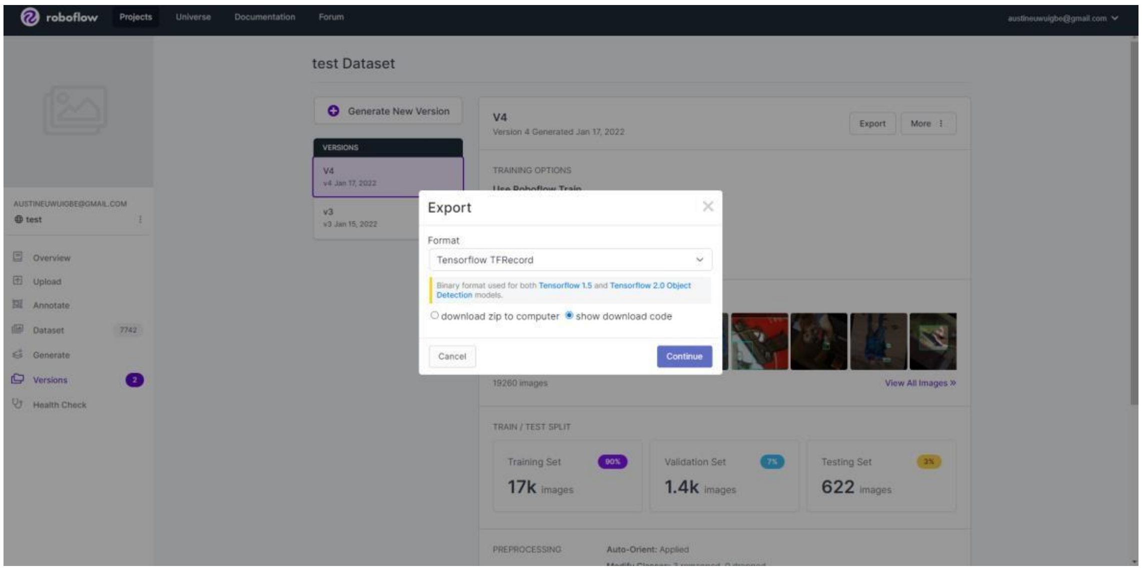
PREPROCESSING
Auto-Orient: Applied
Modify Classes: 3 remapped, 0 dropped

AUGMENTATIONS
Outputs per training example: 3
Flip: Vertical
Rotation: Between -45° and +45°
Blur: Up to 4.25px

DETAILS
Version Name: V4
Version ID: 4
Generated: Jan 17, 2022
Annotation Group: cars-moose

Figure 12. Image Pre-processing and Augmentation with Roboflow [2] As seen in figure 12 several pre-processing and augmentation techniques were used to provide about 3 times more training data like rotation, vertical flip, blur, etc.

These newly generated data are then exported as TensorFlow records (TFRecord), which would be used to train the TensorFlow model as shown in figure 13 below.



Export

Format: Tensorflow TFRecord

Binary format used for both Tensorflow 1.5 and Tensorflow 2.0 Object Detection models.

download zip to computer [✓ show download code]

Figure 13. Exporting TensorFlow records via Roboflow [2]

Model Training

To give a more detailed and accurate description, it is important to Describe what kind of hardware was used for building the models used for this project.

The table below describes the Hardware specifications for the machine used in training the model.

Specification	Details
Hardware Name	ROG-Strix-G15-G513 Laptop
Model	G513QM-HF225T
Processor	AMD-Ryzen™ 7 5800H Mobile Processor (8-core/16-thread, 20MB cache, up to 4.4 GHz max boost)

Graphics card	NVIDIA® GeForce RTX™ 3060 Laptop GPU With ROG Boost up to 1802MHz at 115W (130W with Dynamic Boost) 6GB GDDR6

Two of the fastest single-shot detector (640 x 640 pixeled) models in terms of response time were chosen for the application. The models are.

- SSD MobileNet V1 FPN 640X640
- SSD ResNet50 V1 FPN 640X640

Why is Single-Shot detector?

The SSD model uses the VG-16 model as the feature map extractor and then series of convolutional filters in addition for the object detection. The single-shot detector aims to optimize speed and accuracy. It runs several convolutional layers of different scales, which makes it able to detect objects of different sizes.

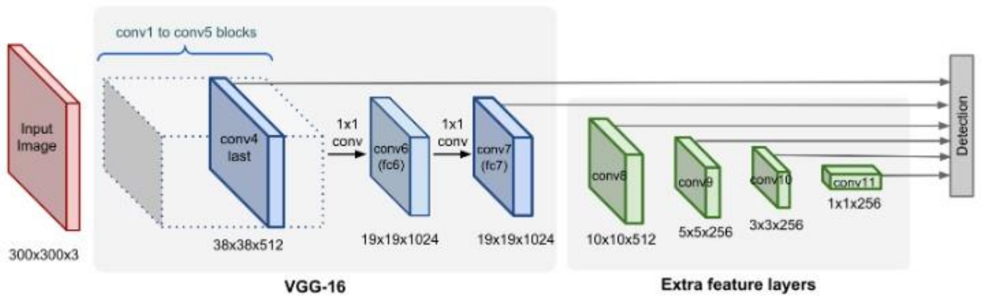


Figure 14. SSD Model Architecture [12]

What is Single-Shot Detection?

This means that the task for classifying and localizing objects within an image is done with just one pass across the network. Hence its speed compared to other model architectures.

<u>Model 1: SSD MobileNet FPNlite.</u>

MobileNet as the name implies is a lightweight convolutional network designed to run on mobile devices. It uses the depth wise separable convolutional layer instead of the standard convolutional layer. Since the SSD model is independent of its base model, the MobileNet architecture is used as its base model in this case.

It consists of Three Layers as shown in the figure below.

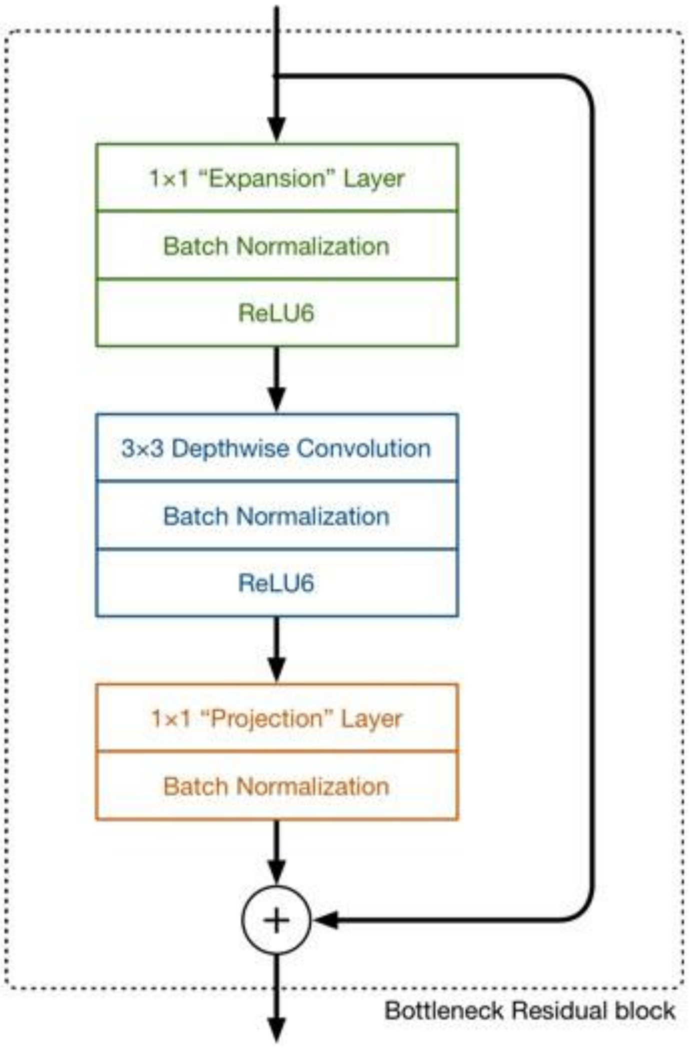


Figure 15. MobileNet V2 Architecture [14]

Like the MobileNet version 1 it maintains the 3x3 DepthWise Convolution layer, but the Projection Layer reduces the dimension of the input channel unlike the Pointwise Layer of the Version 1 which does the opposite.[14]

This model template was found in the official TensorFlow GitHub repo.

Training the model

The model takes images of size 640 x 640 pixels with a batch size of 2 due to computing power limitation. A batch size of 3 or more resulted in memory allocation error. After training this model for over 1.9 million steps, some inference graphs were derived. Four accuracy metrics were used to determine the performance of the model.

1. Classification Loss: This describes how well the model correctly classifies the images. If 2000 images were processed, how many images are correctly classified, and how many were wrongly classified?
2. Localization Loss: This describes how well the model correctly locates the object within the image. A bounding box is constructed to precisely point out the object within the processed frame.
3. Total Loss: This combines the loss derived from classification and localization. It enables better inference and evaluation of the model.
4. Mean Average Precision: This is the mean value for the average precision of each class of object to be detected. The precision of a model simply means how well the model can identify True Positives (TP) out of all positive predictions. It is given by Precision = (TP)/ (TP + FP) where FP is False Positives and TP is True Positives.

The graph showing how the model has performed during the training process is displayed below.

The vertical axis describes the loss value ranging from 0 to 1

The Horizontal axis describes the number of steps ranging from 0 to 1.9 million.

These graphs are displayed in the figures below.

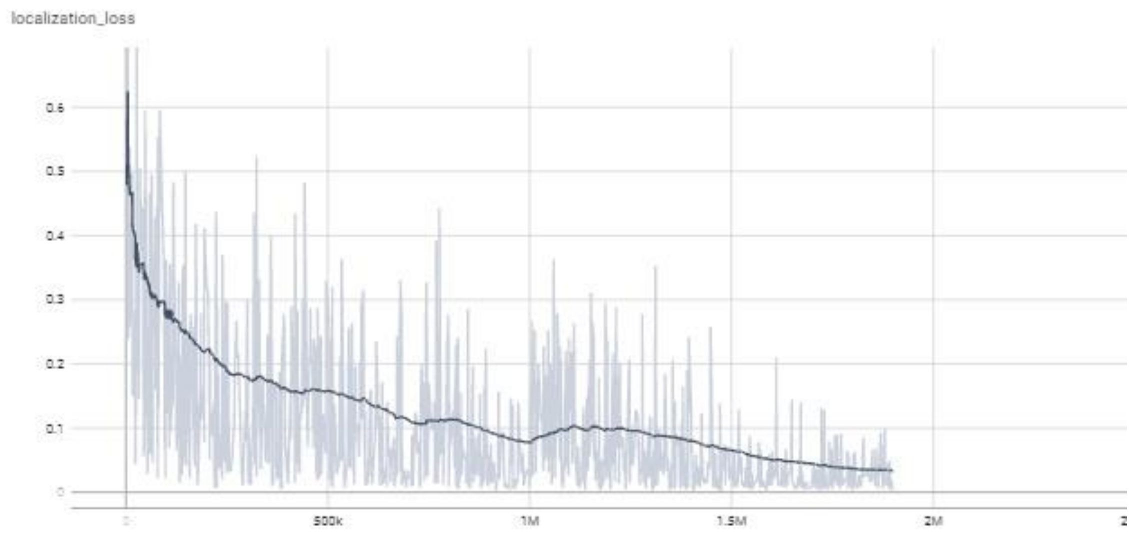


Figure 16. Localization Loss Graph

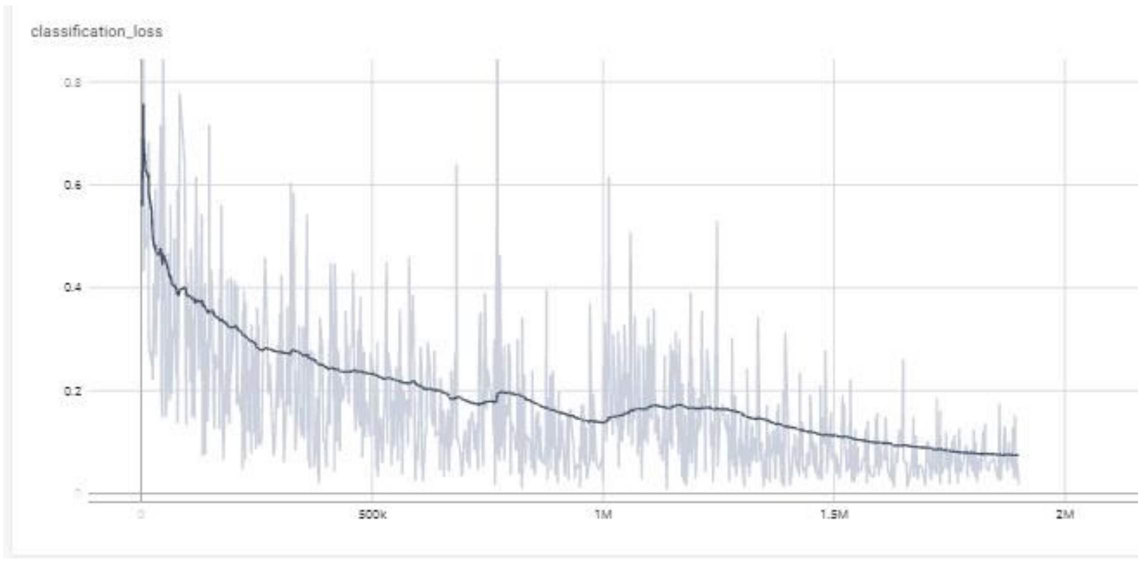


Figure 17. Classification Loss Graph

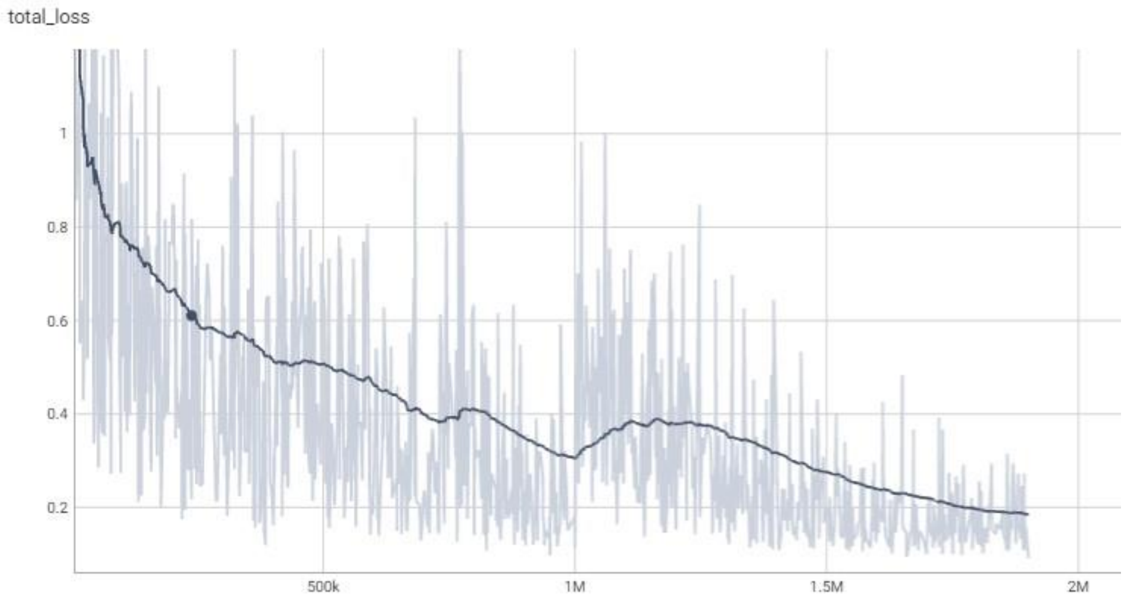
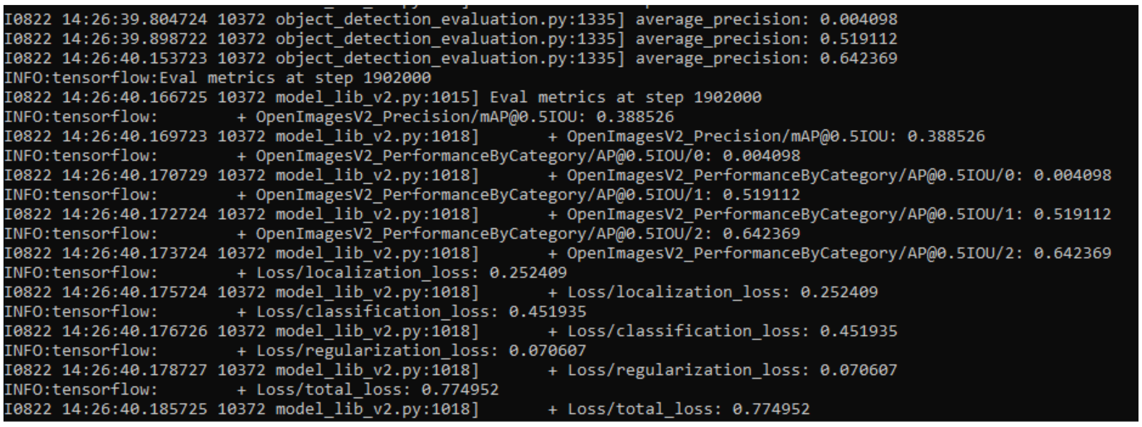


Figure 18. Total Loss Graph

As seen from the graphs above the loss gradually reduced over time with the fastest learning curve between step 0 to step 1000000(one million). The total time taken to train this model was 75.23 hours (3.14 days).

Model Evaluation

There were 622 test images used to evaluate the model. After running the evaluation, the following results were obtained as displayed in the snippet below.



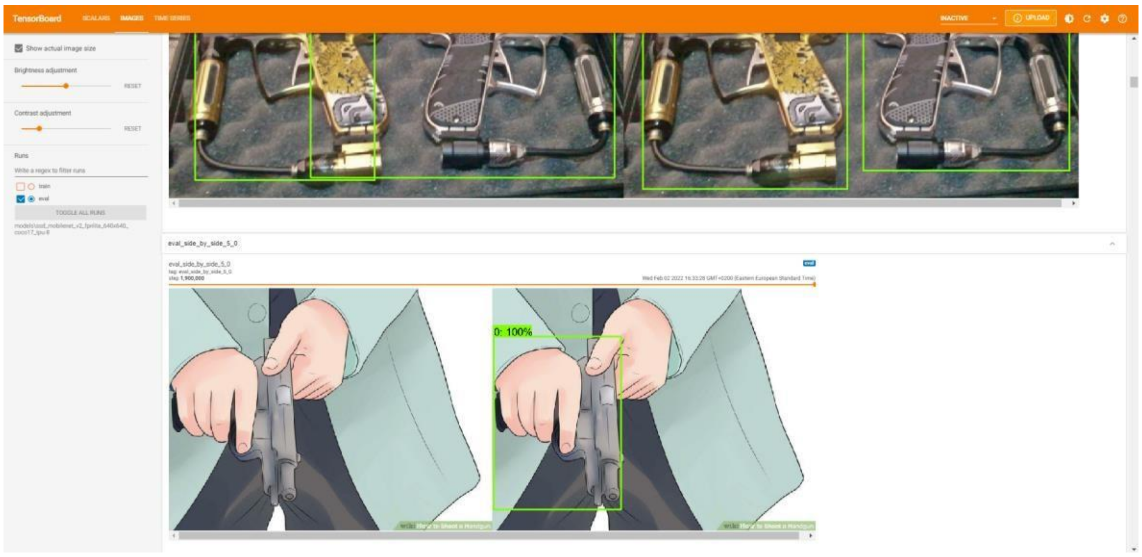
```
I0822 14:26:39.804724 10372 object_detection_evaluation.py:1335] average_precision: 0.004098
I0822 14:26:39.898722 10372 object_detection_evaluation.py:1335] average_precision: 0.519112
I0822 14:26:40.153723 10372 object_detection_evaluation.py:1335] average_precision: 0.642369
INFO:tensorflow:Eval metrics at step 1902000
I0822 14:26:40.166725 10372 model_lib_v2.py:1015] Eval metrics at step 1902000
INFO:tensorflow:        + OpenImagesV2_Precision/mAP@0.5IOU: 0.388526
I0822 14:26:40.169723 10372 model_lib_v2.py:1018]       + OpenImagesV2_Precision/mAP@0.5IOU: 0.388526
INFO:tensorflow:        + OpenImagesV2_PerformanceByCategory/AP@0.5IOU/0: 0.004098
I0822 14:26:40.170729 10372 model_lib_v2.py:1018]       + OpenImagesV2_PerformanceByCategory/AP@0.5IOU/0: 0.004098
INFO:tensorflow:        + OpenImagesV2_PerformanceByCategory/AP@0.5IOU/1: 0.519112
I0822 14:26:40.172724 10372 model_lib_v2.py:1018]       + OpenImagesV2_PerformanceByCategory/AP@0.5IOU/1: 0.519112
INFO:tensorflow:        + OpenImagesV2_PerformanceByCategory/AP@0.5IOU/2: 0.642369
I0822 14:26:40.173724 10372 model_lib_v2.py:1018]       + OpenImagesV2_PerformanceByCategory/AP@0.5IOU/2: 0.642369
INFO:tensorflow:        + Loss/localization_loss: 0.252409
I0822 14:26:40.175724 10372 model_lib_v2.py:1018]       + Loss/localization_loss: 0.252409
INFO:tensorflow:        + Loss/classification_loss: 0.451935
I0822 14:26:40.176726 10372 model_lib_v2.py:1018]       + Loss/classification_loss: 0.451935
INFO:tensorflow:        + Loss/regularization_loss: 0.070607
I0822 14:26:40.178727 10372 model_lib_v2.py:1018]       + Loss/regularization_loss: 0.070607
INFO:tensorflow:        + Loss/total_loss: 0.774952
I0822 14:26:40.185725 10372 model_lib_v2.py:1018]       + Loss/total_loss: 0.774952
```

Snippet 2. Model 1 Evaluation Results

From the Image above the model produced a MaP of 0.388526 based on 3 classes (Fire, pistol, and Weapon). However, one of the classes had an Average

precision Value of 0.0041 which is very much lower than the other classes which produced an AP of 0.52 and 0.642 respectively.
This means the model fails to recognize a lot of True Positives for class 0 compared to other classes. It is Important to mention that the amount of Image data for class 0 was much higher than other classes (Imbalanced Training Data).

The quality of the Images was poor in some cases which affects the evaluation performance.
The image below shows the quality of images that was occasionally tested and how the model fails to detect objects in this image.

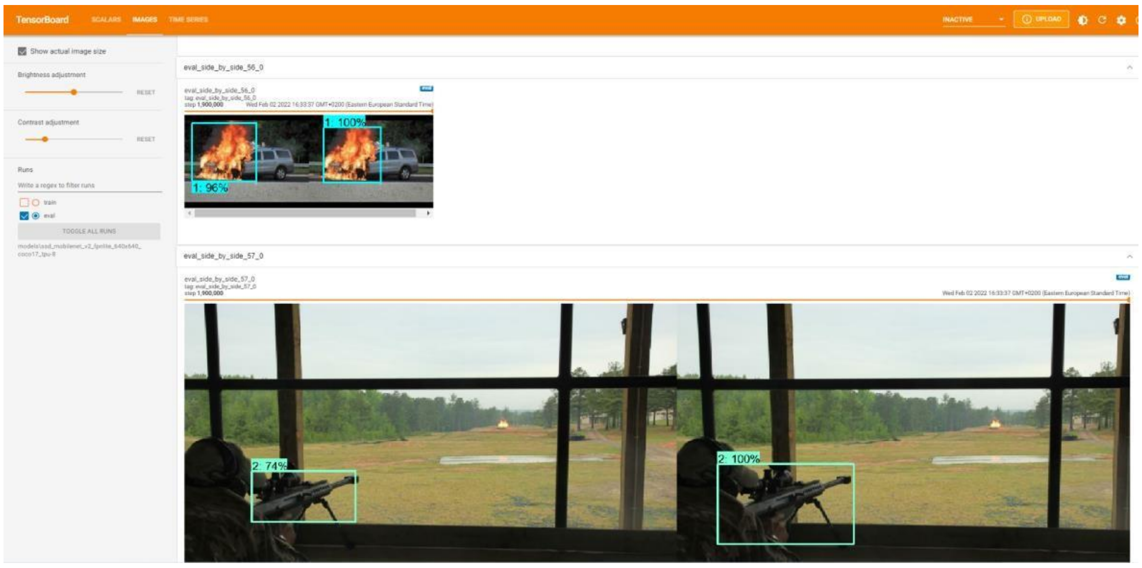


models\ssd_mobilenet_v2_fpnlite_640x640_coco17_tpu-8

eval_side_by_side_5_0
tag: eval_side_by_side_5_0
step 1,900,000

Figure 19. Model Performance on Animated Images

As seen in the Figure above, the image with no detection appears to be an animated image, which is not viable for our test as the solution is to be used for real-life cases.
However, in the image below it is visible that the detection was made on an image captured by a camera.



models\ssd_mobilenet_v2_fpnlite_640x640_coco17_tpu-8

eval_side_by_side_56_0

eval_side_by_side_56_0
tag: eval_side_by_side_56_0
step 1,900,000 Wed Feb 02 2022 16:33:37 GMT+0200 (Eastern European Standard Time)

eval_side_by_side_57_0

eval_side_by_side_57_0
tag: eval_side_by_side_57_0
step 1,900,000 Wed Feb 02 2022 16:33:37 GMT+0200 (Eastern European Standard Time)

Figure 20. Model Performance on Normal Images

<u>Model 2: SSD ResNet50 Fpn.</u>

Residual Networks 50 or ResNet50 is a Convolutional network that is 50 layers deep. The pretrained ResNet50 is used as base model for the feature extractor layer performing the object detection (SSD layer).

The batch size of the images used by the algorithm was also reduced to 2 due to the computing power limitation. The total training time for this model after 1.7 million steps was 107.216 hours (4.47 days)

After training the following accuracy graphs below were derived for evaluation purposes.

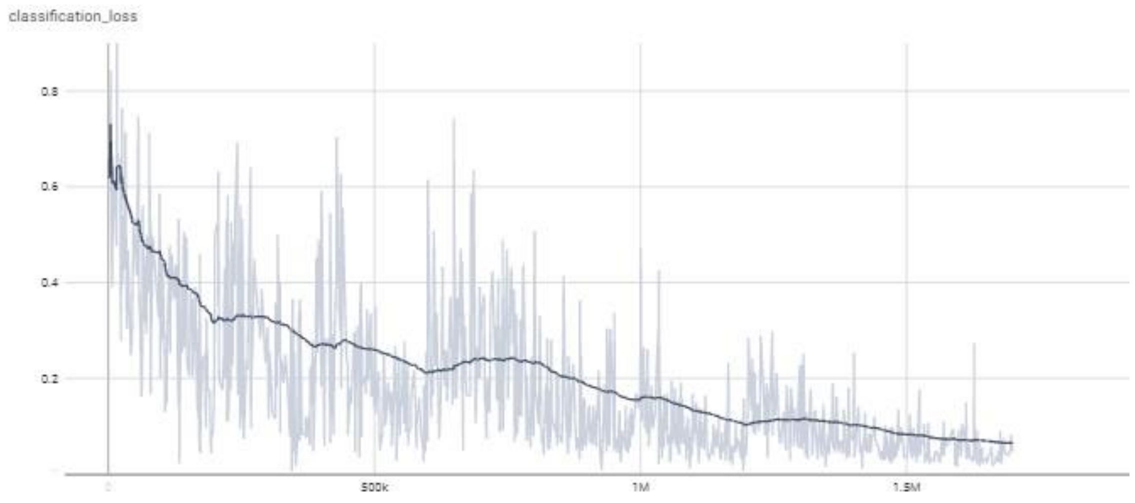


Figure 21. Resnet-Model Classification Loss Graph

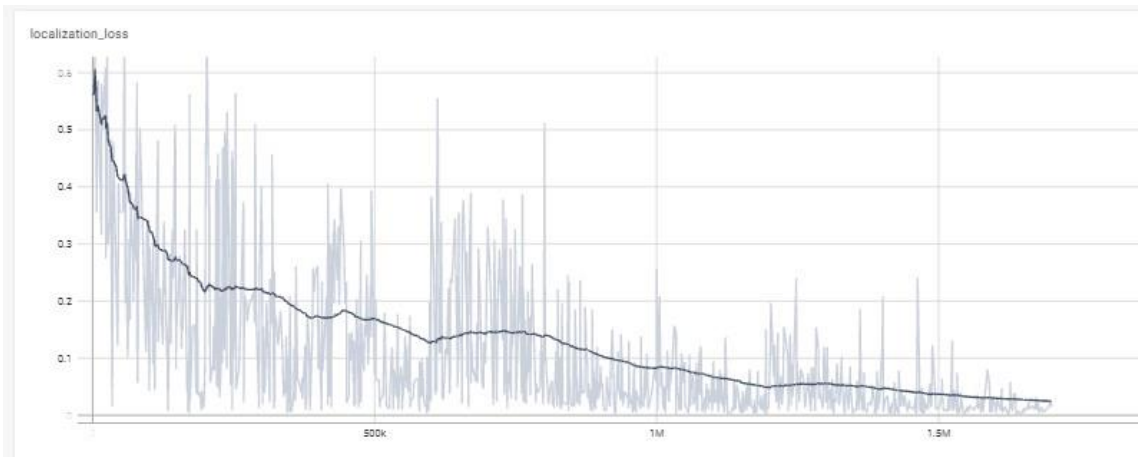


Figure 22. Resnet-Model Localization Loss Graph

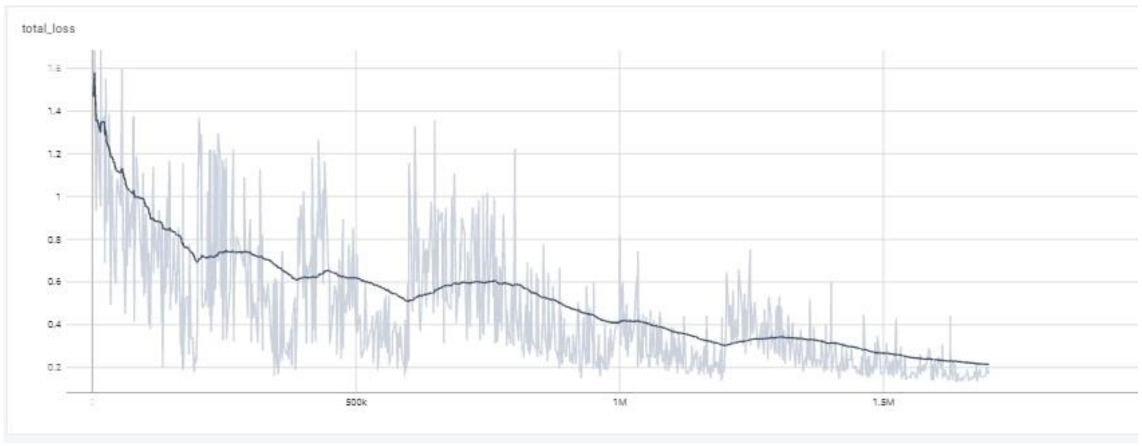
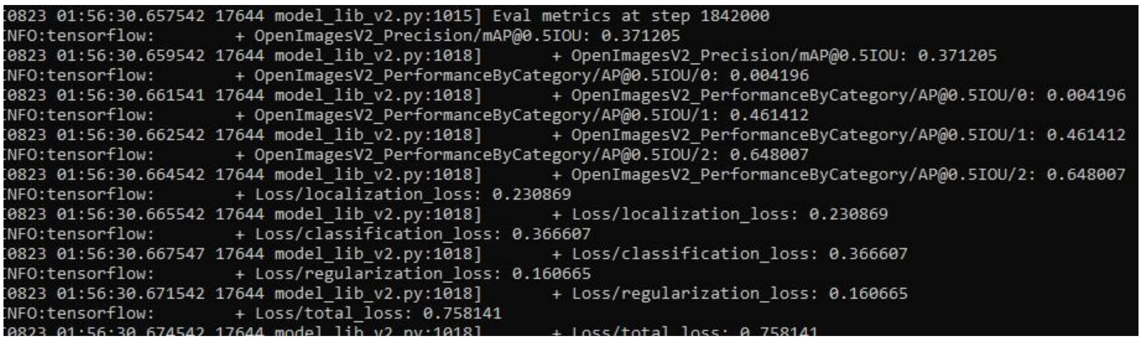


Figure 23. Resnet-Model Total Loss Graph

Model Evaluation

After running the evaluation for the 622 test Images, the following results for this model were obtained as displayed in the snippet below.



```
0823 01:56:30.657542 17644 model_lib_v2.py:1015] Eval metrics at step 1842000
NFO:tensorflow:         + OpenImagesV2_Precision/mAP@0.5IOU: 0.371205
0823 01:56:30.659542 17644 model_lib_v2.py:1018]        + OpenImagesV2_Precision/mAP@0.5IOU: 0.371205
NFO:tensorflow:         + OpenImagesV2_PerformanceByCategory/AP@0.5IOU/0: 0.004196
0823 01:56:30.661541 17644 model_lib_v2.py:1018]        + OpenImagesV2_PerformanceByCategory/AP@0.5IOU/0: 0.004196
NFO:tensorflow:         + OpenImagesV2_PerformanceByCategory/AP@0.5IOU/1: 0.461412
0823 01:56:30.662542 17644 model_lib_v2.py:1018]        + OpenImagesV2_PerformanceByCategory/AP@0.5IOU/1: 0.461412
NFO:tensorflow:         + OpenImagesV2_PerformanceByCategory/AP@0.5IOU/2: 0.648007
0823 01:56:30.664542 17644 model_lib_v2.py:1018]        + OpenImagesV2_PerformanceByCategory/AP@0.5IOU/2: 0.648007
NFO:tensorflow:         + Loss/localization_loss: 0.230869
0823 01:56:30.665542 17644 model_lib_v2.py:1018]        + Loss/localization_loss: 0.230869
NFO:tensorflow:         + Loss/classification_loss: 0.366607
0823 01:56:30.667547 17644 model_lib_v2.py:1018]        + Loss/classification_loss: 0.366607
NFO:tensorflow:         + Loss/regularization_loss: 0.160665
0823 01:56:30.671542 17644 model_lib_v2.py:1018]        + Loss/regularization_loss: 0.160665
NFO:tensorflow:         + Loss/total_loss: 0.758141
0823 01:56:30.674542 17644 model_lib_v2.py:1018]        + Loss/total_loss: 0.758141
```

Snippet 3. Model2 Evaluation results

From the Image above the model produced a MaP of 0.371 based on 3 classes (Fire, pistol, and Weapon). The same behaviour was noticed as one of the classes has an Average precision Value of 0.0041 which is very much lower than the other classes which produced an AP of 0.46 and 0.648 respectively.
This can confirm that our Training data is imbalanced.

Comparing Results of Model1 and Model2

Model	**Batch-Size**	**Number of steps**	**Total time**	**Evaluation MaP (623 images)**	**Model-Size**
Model 1	2	1900000	3.14 days	0.389	17.9 Megabytes
Model 2	2	1700000	4.47 days	0.371	128 Megabytes

From the above table, model 1 yielded a better result and was the best model of the two.

3.2.2 The Computer Vision system

This system is a TensorFlow serving object detection model using REST-API for risk identification and reporting.

The application consists of two docker containers as follows.

- **Server:** This container runs the TensorFlow Service that makes the object detection model available via a network for inference purposes. The model is accessed by clients that have access to the network.
- **Client:** This container runs the script which periodically scans a video source and passes each frame (Image) to the model on the server application. The client application receives a response from the server about the predicted item.

Below is an architecture of this Computer Vision system built with Docker.

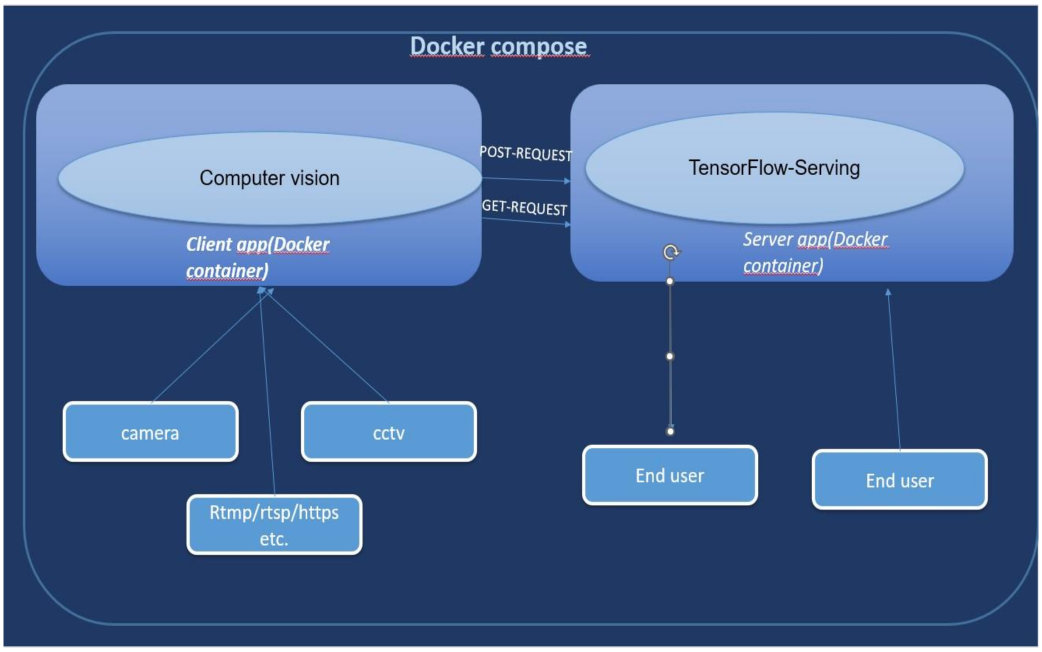


Figure 24. Computer Vision system architecture within Docker-compose

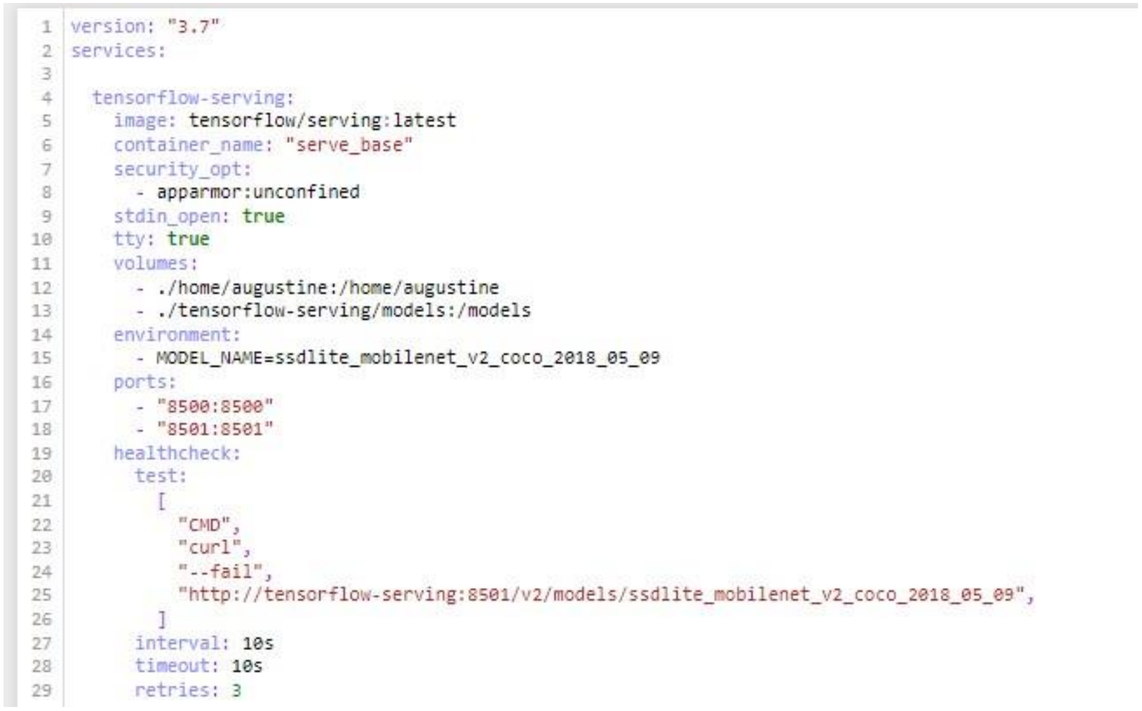
From the figure above, it is visible that there are two main docker containers.

The Server container runs the TensorFlow serving tool which takes the trained ML model and makes it available via the REST-API.

The Client container uses all necessary libraries like OpenCV and Pillow to process videos from various sources as individual frames and send each frame to the server via a POST request for processing.

The server returns a response to the client app containing possible detections. All these automated processes are managed with the dockercompose tool.

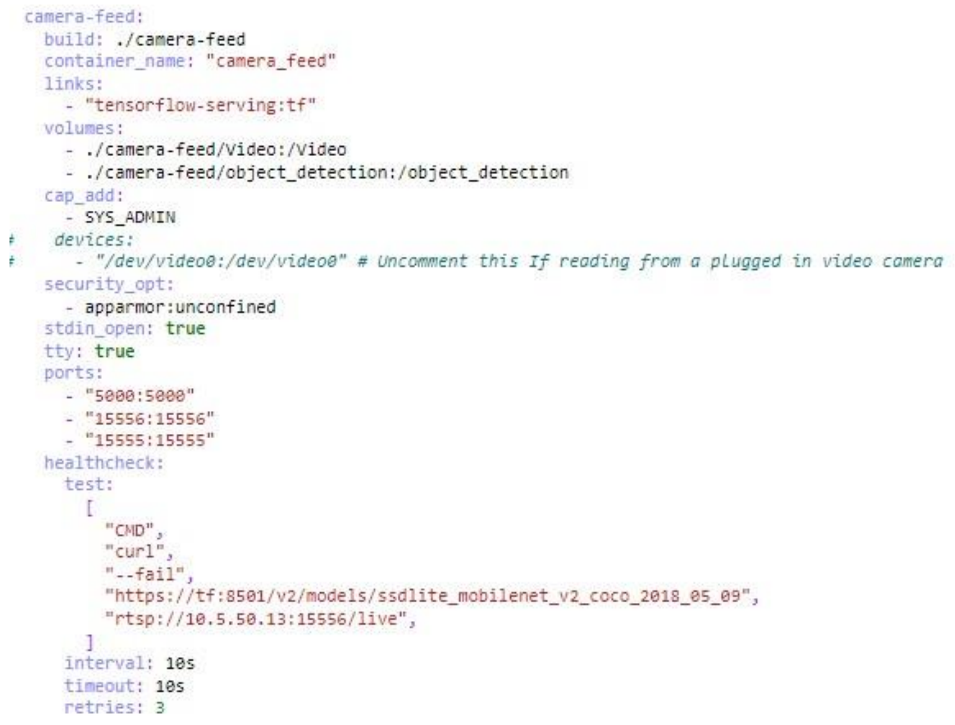
The snippet below shows the code used in building both docker containers.



```yaml
version: "3.7"
services:

  tensorflow-serving:
    image: tensorflow/serving:latest
    container_name: "serve_base"
    security_opt:
      - apparmor:unconfined
    stdin_open: true
    tty: true
    volumes:
      - ./home/augustine:/home/augustine
      - ./tensorflow-serving/models:/models
    environment:
      - MODEL_NAME=ssdlite_mobilenet_v2_coco_2018_05_09
    ports:
      - "8500:8500"
      - "8501:8501"
    healthcheck:
      test:
        [
          "CMD",
          "curl",
          "--fail",
          "http://tensorflow-serving:8501/v2/models/ssdlite_mobilenet_v2_coco_2018_05_09",
        ]
      interval: 10s
      timeout: 10s
      retries: 3
```

Snippet 4. Definition of Server and Client applications in docker-compose YAML file.

- **The TensorFlow-Serving docker image:** This image is provided by TensorFlow and can be pulled directly from the docker repository.
- **The trained ML models:** This model is mounted to the server application so that the container can provide it to the client application.



```
  camera-feed:
    build: ./camera-feed
    container_name: "camera_feed"
    links:
      - "tensorflow-serving:tf"
    volumes:
      - ./camera-feed/Video:/Video
      - ./camera-feed/object_detection:/object_detection
    cap_add:
      - SYS_ADMIN
#     devices:
#       - "/dev/video0:/dev/video0" # Uncomment this If reading from a plugged in video camera
    security_opt:
      - apparmor:unconfined
    stdin_open: true
    tty: true
    ports:
      - "5000:5000"
      - "15556:15556"
      - "15555:15555"
    healthcheck:
      test:
        [
          "CMD",
          "curl",
          "--fail",
          "https://tf:8501/v2/models/ssdlite_mobilenet_v2_coco_2018_05_09",
          "rtsp://10.5.50.13:15556/live",
        ]
      interval: 10s
      timeout: 10s
      retries: 3
```

Snippet 5. Custom model mounted into client docker application.

- **Object detection folder:** The necessary files have been mounted into the container using docker volumes.
  The client application uses OpenCV to process frames and send each frame to the TensorFlow server for prediction.

- **Docker link:** This is a way to link the network namespace of one container with another. When a container is built, it creates its network environment and cannot be accessed by the outside world or from other containers. The network environment can be accessed by opening a port between the external network to the container network or by linking several containers with each other.

3.2.3 Integrating The App with Agnet

The goal of the project is to detect objects within a video source and then send this information to the Agnet Server. It is therefore important to know what Agnet means. The below subsection will further explain what Agnet means.

What is Agnet?

Agnet is a secure group collaboration solution. It brings secure professional communications to your smart devices and offers the capability to bring your radio and smart device users to the same groups. Hence, smart device users become part of the professional world. Voice, data, video, and location services are all at hand with the reliability and security that professional users expect. [15]

Tactilon Agnet extends flexibly from a simple push-to-talk (PTT) to an extensive group collaboration solution that makes use of smart device capabilities in a secure and controlled way. It offers a full variety of different communication methods from instant PTT to video and location sharing, and the ability to professionally manage users and groups.

It meets the requirements of public safety organizations as well as transport, utility, and industry users. The data necessary for an operation can be easily and securely accessed, even when using different devices and technologies.

Features of Agnet

- Push to talk group calls (MCPTT)
- Private Voice Call
- Messaging (Multimedia)
- End-to-end encryption
- MCDATA: Secure messaging and data collaboration, also from 3rd party systems.
- MC Video: video calls and streaming
- Emergency calls
- Lifeguard
  (advanced man-down)
- Real-time location tracking
- Operational status

The figure below gives a brief overview of the features of Agnet

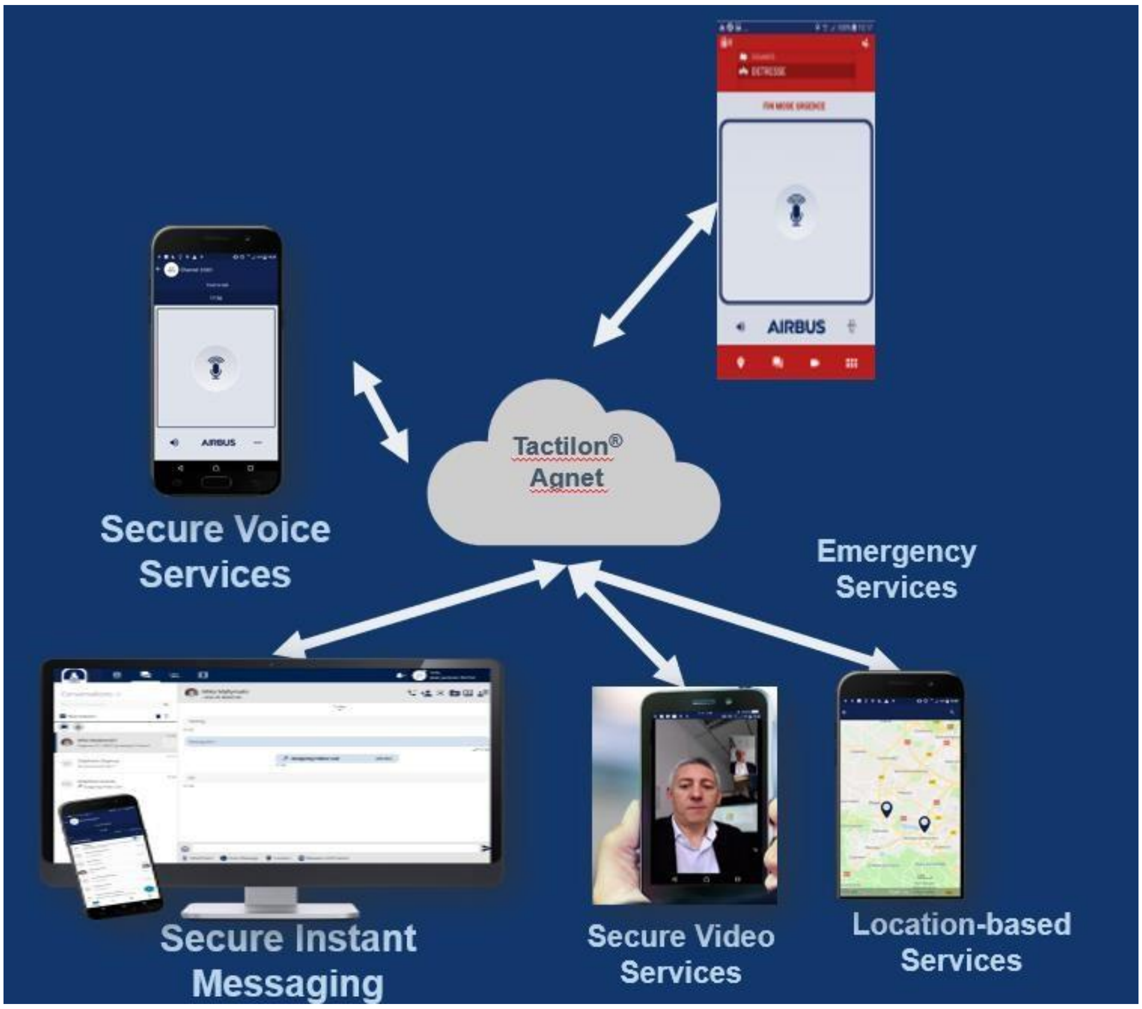


Figure 25. Features of Agnet.

Agnet-Api

This is a developer program for applications and accessories to help extend the functionalities of Agnet using innovative solutions. It acts as an API gateway for integrating new solutions to the agnet backend.

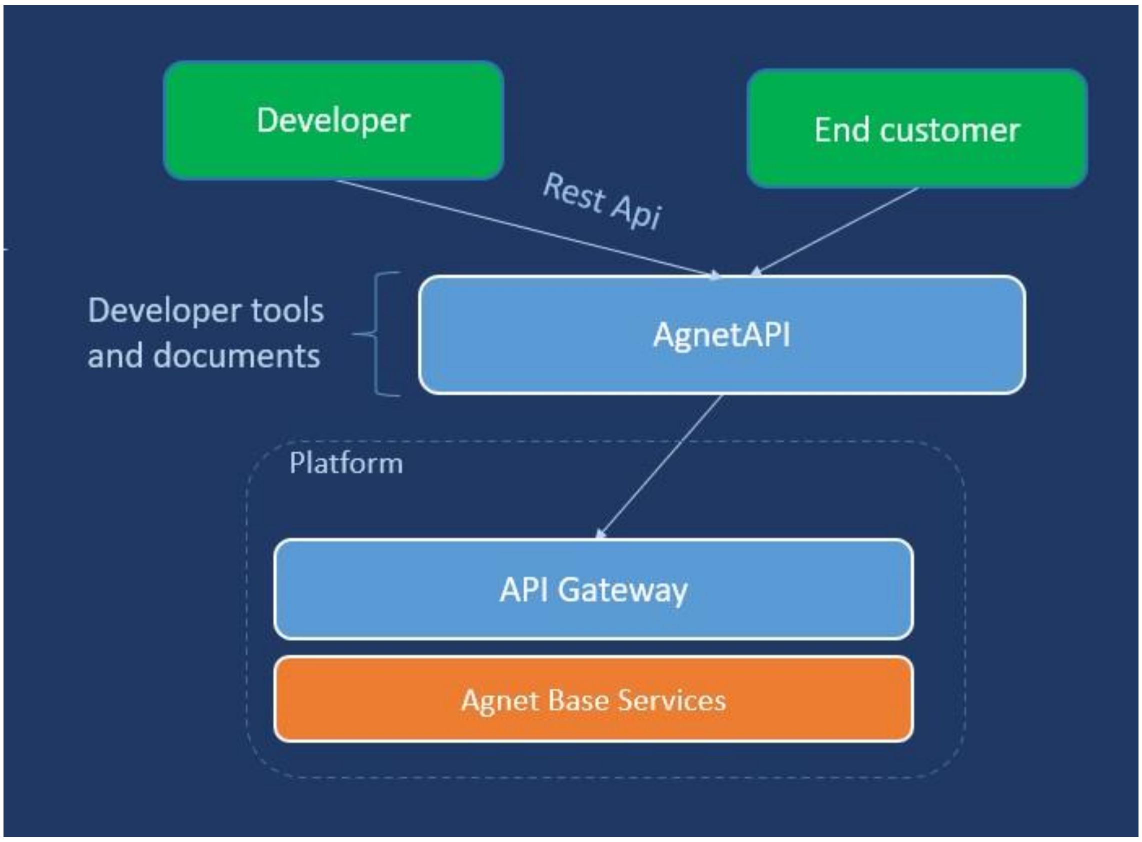
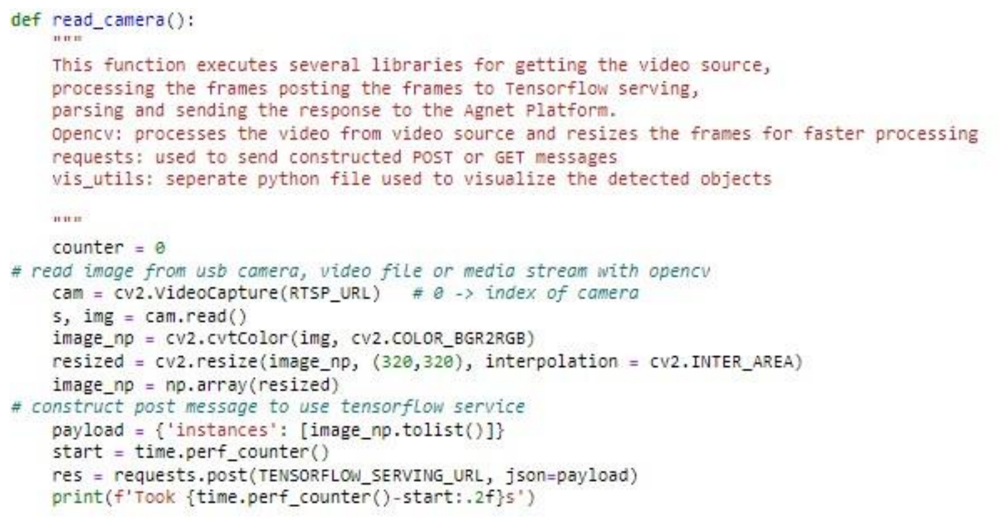


Figure 26. Agnet-API Service model from Third Party Applications.

As seen from the figure above, the AgnetAPI service can be reached via the REST API protocol. This service can then trigger a series of actions on the Agnet Platform. Actions like Emergency message triggers can alert the end-user of incidents.
The Agnet-API program will be used to integrate the computer vision system into the Agnet platform.

The client application of the Computer Vision system contains most of the code and data orchestration from Image processing to constructing post messages to Agnet containing detected objects.
There are 3 main steps involved within the client application from image processing to triggering of alerts on the Agnet platform. These steps are listed below.

- Video capture and processing: The video is captured with OpenCV and is processed frames by frames. These frames are then parsed as JSON objects and posted to the TensorFlow Service for inference. The below code snippet shows how the video is being processed with OpenCV

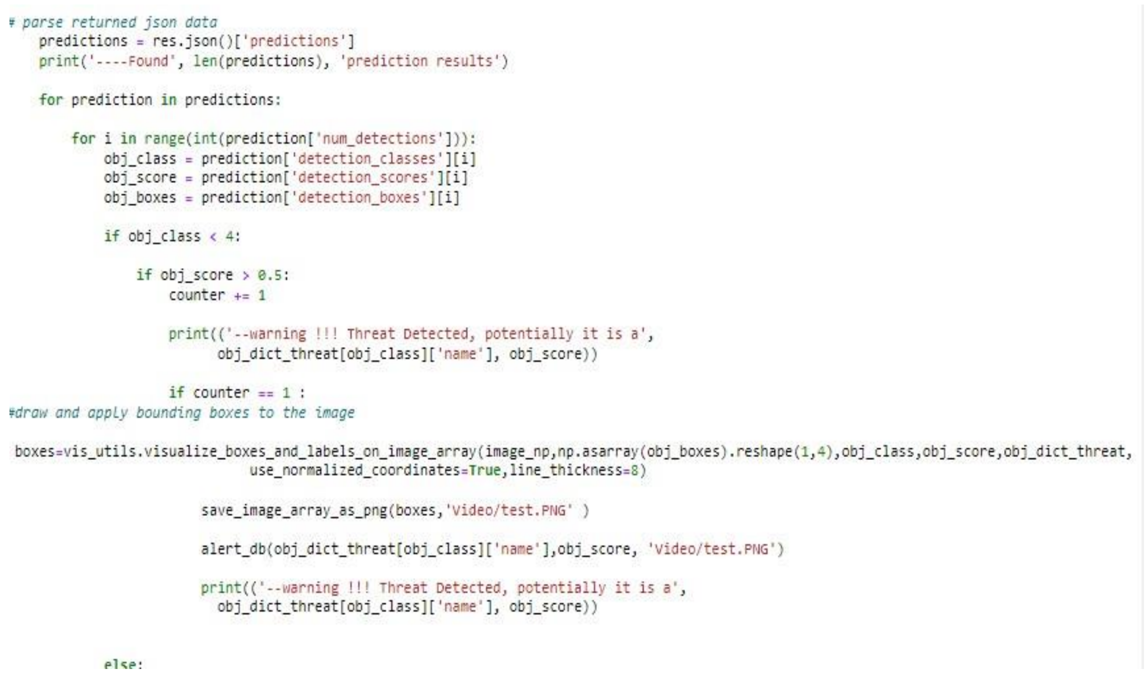


```python
def read_camera():
    """
    This function executes several libraries for getting the video source,
    processing the frames posting the frames to Tensorflow serving,
    parsing and sending the response to the Agnet Platform.
    Opencv: processes the video from video source and resizes the frames for faster processing
    requests: used to send constructed POST or GET messages
    vis_utils: seperate python file used to visualize the detected objects

    """
    counter = 0
# read image from usb camera, video file or media stream with opencv
    cam = cv2.VideoCapture(RTSP_URL)   # 0 -> index of camera
    s, img = cam.read()
    image_np = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    resized = cv2.resize(image_np, (320,320), interpolation = cv2.INTER_AREA)
    image_np = np.array(resized)
# construct post message to use tensorflow service
    payload = {'instances': [image_np.tolist()]}
    start = time.perf_counter()
    res = requests.post(TENSORFLOW_SERVING_URL, json=payload)
    print(f'Took {time.perf_counter()-start:.2f}s')
```

Snippet 6. Capturing frames from video source and sending it to TensorFlow-Server for inference.

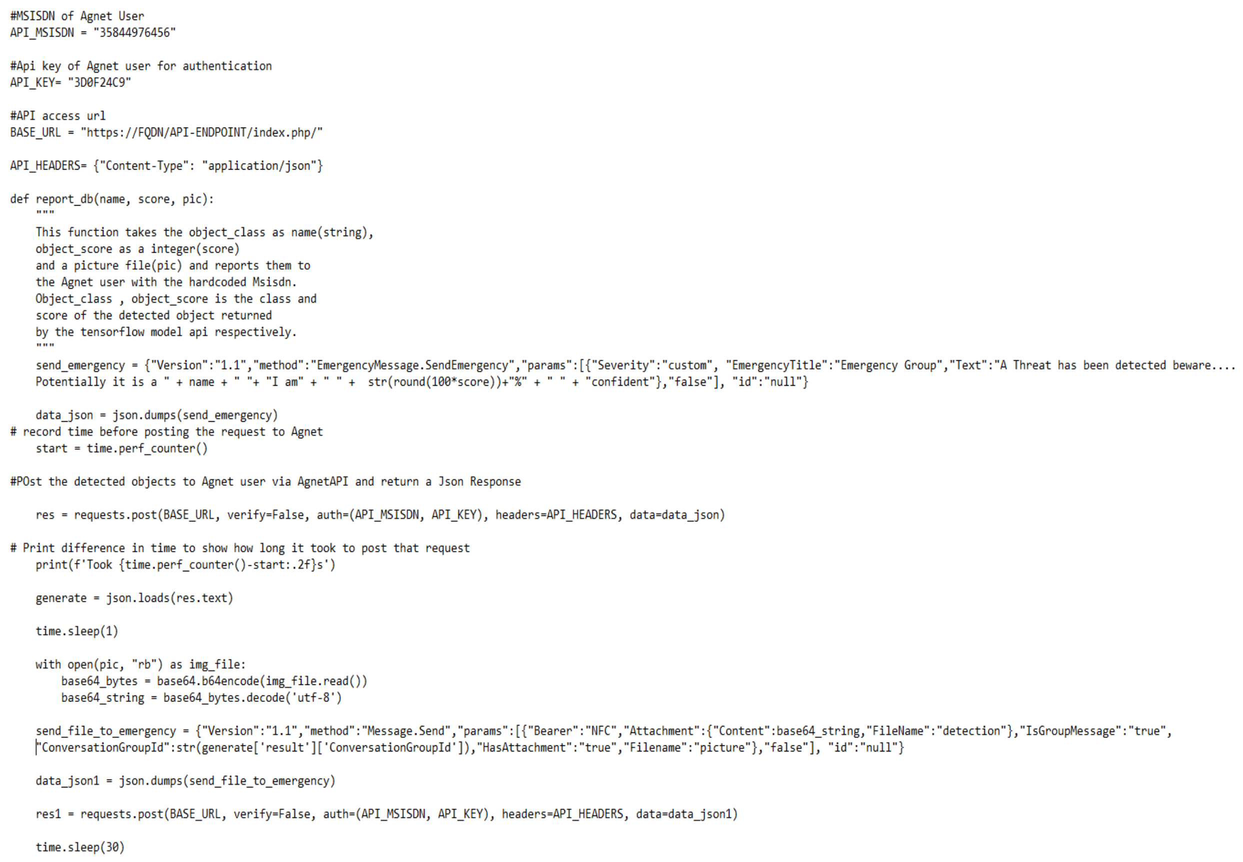
- TensorFlow object detection: The TensorFlow service which runs the custom-trained model will take these frames as input and return a JSON object containing predictions. Snippet 6 shows the frames being sent as post messages to the TensorFlow Service.

- Event trigger based on detection: The predictions are further passed into a function that constructs a posted message to the Agnet services. An emergency message event is triggered and the Agnet client user receives an image with the detected item. This post message uses the REST-API. The snippets below show how this returned JSON data is parsed and used to trigger events via the Agnet-API.



```
# parse returned json data
    predictions = res.json()['predictions']
    print('----Found', len(predictions), 'prediction results')

    for prediction in predictions:

        for i in range(int(prediction['num_detections'])):
            obj_class = prediction['detection_classes'][i]
            obj_score = prediction['detection_scores'][i]
            obj_boxes = prediction['detection_boxes'][i]

            if obj_class < 4:

                if obj_score > 0.5:
                    counter += 1

                    print(('--warning !!! Threat Detected, potentially it is a',
                          obj_dict_threat[obj_class]['name'], obj_score))

                    if counter == 1 :
#draw and apply bounding boxes to the image

 boxes=vis_utils.visualize_boxes_and_labels_on_image_array(image_np,np.asarray(obj_boxes).reshape(1,4),obj_class,obj_score,obj_dict_threat,
                              use_normalized_coordinates=True,line_thickness=8)

                        save_image_array_as_png(boxes,'Video/test.PNG' )

                        alert_db(obj_dict_threat[obj_class]['name'],obj_score, 'Video/test.PNG')

                        print(('--warning !!! Threat Detected, potentially it is a',
                          obj_dict_threat[obj_class]['name'], obj_score))


            else:
```

Snippet 7. Visualizing the detected objects and passing it to the Database function.



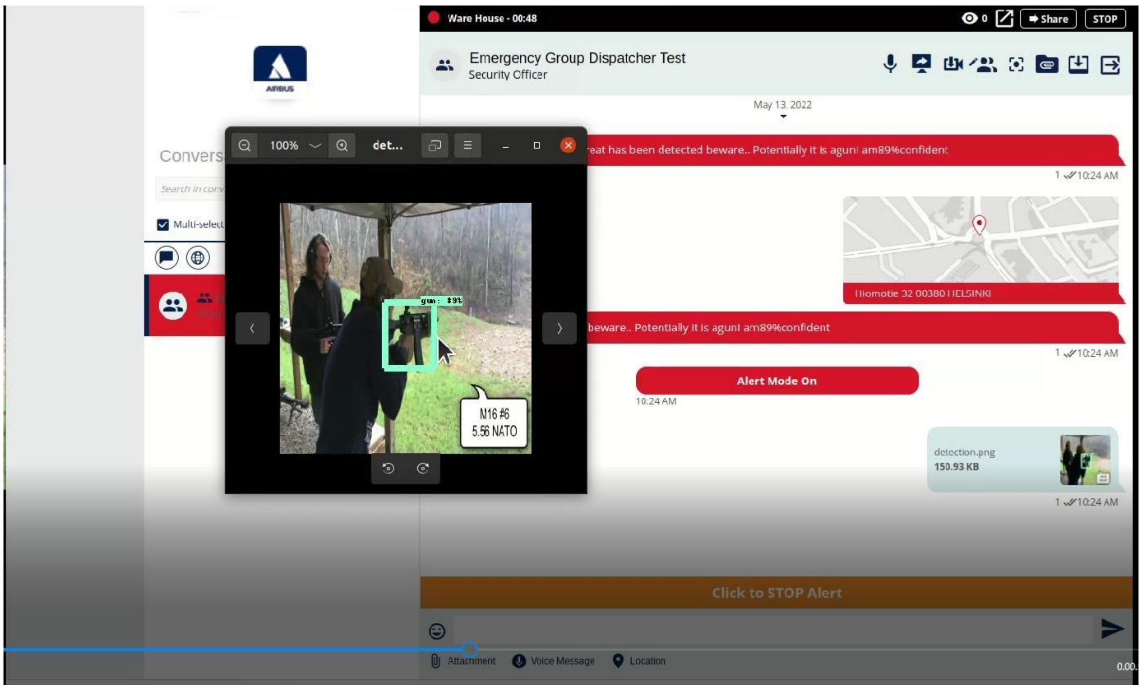
```
#MSISDN of Agnet User
API_MSISDN = "35844976456"

#Api key of Agnet user for authentication
API_KEY= "3D0F24C9"

#API access url
BASE_URL = "https://FQDN/API-ENDPOINT/index.php/"

API_HEADERS= {"Content-Type": "application/json"}

def report_db(name, score, pic):
    """
    This function takes the object_class as name(string),
    object_score as a integer(score)
    and a picture file(pic) and reports them to
    the Agnet user with the hardcoded Msisdn.
    Object_class , object_score is the class and
    score of the detected object returned
    by the tensorflow model api respectively.
    """
    send_emergency = {"Version":"1.1","method":"EmergencyMessage.SendEmergency","params":[{"Severity":"custom", "EmergencyTitle":"Emergency Group","Text":"A Threat has been detected beware....
    Potentially it is a " + name + " "+ "I am" + " " +  str(round(100*score))+"%" + " " + "confident"},"false"], "id":"null"}

    data_json = json.dumps(send_emergency)
# record time before posting the request to Agnet
    start = time.perf_counter()

#POst the detected objects to Agnet user via AgnetAPI and return a Json Response

    res = requests.post(BASE_URL, verify=False, auth=(API_MSISDN, API_KEY), headers=API_HEADERS, data=data_json)

# Print difference in time to show how long it took to post that request
    print(f'Took {time.perf_counter()-start:.2f}s')

    generate = json.loads(res.text)

    time.sleep(1)

    with open(pic, "rb") as img_file:
        base64_bytes = base64.b64encode(img_file.read())
        base64_string = base64_bytes.decode('utf-8')

    send_file_to_emergency = {"Version":"1.1","method":"Message.Send","params":[{"Bearer":"NFC","Attachment":{"Content":base64_string,"FileName":"detection"},"IsGroupMessage":"true",
    "ConversationGroupId":str(generate['result']['ConversationGroupId']),"HasAttachment":"true","Filename":"picture"},"false"], "id":"null"}

    data_json1 = json.dumps(send_file_to_emergency)

    res1 = requests.post(BASE_URL, verify=False, auth=(API_MSISDN, API_KEY), headers=API_HEADERS, data=data_json1)

    time.sleep(30)
```

Snippet 8. A report Database function that communicates with Agnet via the API interface.

4 Performance

The application aimed to trigger emergency messages, geo-location, and multimedia messages (containing Images of Detected Items from video sources).

The below figures display images of the events being triggered and sent to the Agnet users.



Emergency Group Dispatcher Test
Security Officer

May 13, 2022

reat has been detected beware.. Potentially it is aguni am89%confident

1 10:24 AM

Hiomotie 32 00380 HELSINKI

beware.. Potentially it is aguni am89%confident

1 10:24 AM

Alert Mode On

10:24 AM

detection.png
150.93 KB

1 10:24 AM

Click to STOP Alert

Figure 27: Emergency message to Agnet Dispatcher (Agnet web application client).

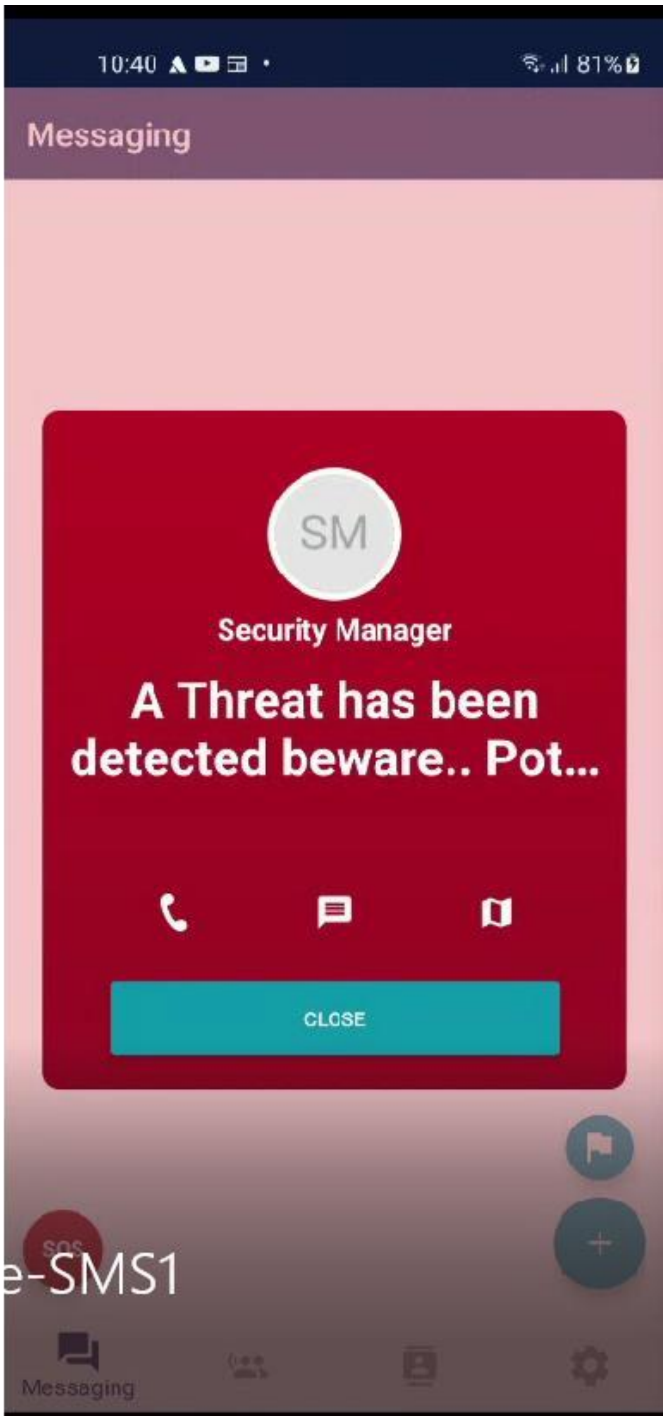

10:40
81%
Messaging
SM
Security Manager
A Threat has been detected beware.. Pot...

Figure 28: Emergency message to Agnet Mobile Client.

The performance of the project was measured on several metrics listed below.

- Response time: The time between the object’s appearance on the video footage to the trigger of the emergency message is in milliseconds. This shows that the application is very time-critical and can make detections faster than the human eye.

- Bandwidth required to run the application: The application does not perform any heavy job. It reads the frame from an already existing video server or video source and performs detection on it. This makes the network requirement much less demanding. The application would run on a 3G, 4G LTE network. The application also runs behind a Secure VPN (Virtual Private Network).
- System requirements: The application is lightweight and will run on any 64bit ubuntu system. A basic ubuntu system requires 1 Gigahertz CPU, 1 Gigabyte RAM (Random Access Memory), and 2.5gigabytes of storage.

5 Conclusion

The project builds a complete object detection system that reads video sources (camera, cctv etc.) and triggers emergency alerts via the Agnet platform. The entire system runs on docker ubuntu. This was achieved by constructing two applications using docker-compose. The server application utilizes TensorFlow-Serving for deploying the model and the client application utilizes Agnet-API for triggering alerts. The application can be easily deployed to several systems efficiently using the docker-compose tool. The Goal of this Thesis project was successfully achieved. This means the object-detection Software was integrated into the Agnet Platform in other to build a smart monitoring solution Proof of Concept for Public Safety. The project application does prove that Agnet can be integrated with any third-party software through Agnet-API.

Further developments for this project include running the system on smaller computing units like the raspberry pi 3b+.

References

1. Atulya Kumar. Fire and Gun Dataset. June 2020.
URL:https://www.kaggle.com/datasets/atulyakumar98/fire-and-gunhttps://www.kaggle.com/datasets/atulyakumar98/fire-and-gun-datasetdataset
2. Bharath Ramsundar and Reza Bosagh Zadeh. TensorFlow for Deep Learning. O'Reilly Media, Inc. March 2018. URL:https://learning.oreilly.com/library/view/tensorflowfordeep/9781491980446/
3. Eihab Mohammed Bashier, Muhammad Badruddin Khan, Mohssen Mohammed. Machine Learning. O'Reilly Media, Inc. August 2016.
URL:https://learning.oreilly.com/library/view/machinelearning/9781315354415/x HTMLch01
4. Nick McClure. TensorFlow Machine Learning Cookbook. O'Reilly Media, Inc. February 2017.
URL:https://learning.oreilly.com/library/view/TensorFlow+Machine+Learning+Co okbook/9781786462169/ch01s02.htm
5. Samaya Madhavan, Sidra Ahmed, Vinay Rao, Anto John. Compare Deep Learning Frameworks. IBM Developer. March 2021.
URL:https://developer.ibm.com/articles/compare-deep-learninghttps://developer.ibm.com/articles/compare-deep-learning-frameworks/frameworks/
6. Pranshu Sharma. A basic introduction to TensorFlow in Deep Learning Analytics. Vidhya. March 2022.
URL: https://www.analyticsvidhya.com/blog/2022/03/a-basic-introductionhttps://www.analyticsvidhya.com/blog/2022/03/a-basic-introduction-to-tensorflow-in-deep-learning/to-tensorflow-in-deep-learning/
7. Sayak Paul. Investigating Tensors with Pytorch. DataCamp. September 2018.
URL: https://www.datacamp.com/tutorial/investigating-tensors-pytorch
8. Scott Carey. What is Docker? The Spark for the container revolution. InfoWorld. August 2021.
URL: https://www.infoworld.com/article/3204171/what-is-docker-thespark-for-the-container-revolution.html
9. IBM Cloud Education. Docker. IBM. June 2021. URL: https://www.ibm.com/cloud/learn/docker

10. Burak Karakan. What exactly is Docker? January 2020.
URL: https://medium.com/swlh/what-exactly-is-docker-1dd62e1fde38

11. IBM Cloud Education. Kubernetes. IBM. July 2021.
URL: https://www.ibm.com/cloud/learn/kubernetes

12. Lilian Weng. Object Detection:Fast Detection Models. December 27 2018.
URL: https://lilianweng.github.io/posts/2018-12-27-object-recognitionpart-4/

13. Eddie Forson. Understanding SSD MultiBox: Realtime Detection Learning.
November 18 2017. URL:
https://towardsdatascience.com/understanding-ssd-multibox-real-

14. Matthijs Hollemans. MobileNet Version 2. April 22 2018.
URL: https://machinethink.net/blog/mobilenet-v2/

15. Airbus SLC. Tactilon-Agnet. August 30 2022.
URL: https://www.securelandcommunications.com/tactilo

16. Scarpino, Matthew. TensorFlow for Dummies. April 2018.
timeobject-detection-in-deep-learning-495ef744fab

URL:https://learning.oreilly.com/library/view/tensorflowhttps://learning.oreilly.com/library/view/tensorflow-for-dummies/9781119466215/fordummies/9781119466215/

17. Rahul, Panchal. What is REST API vs. Web API (vs SOAP API)? August 2021.
URL: https://www.rlogical.com/blog/what-is-rest-api-vs-web-api-vs-soap-api