AUTOMATED HANDLING OF PORT CONTAINERS USING MACHINE LEARNING

A Thesis

BY

MD. ASHIFUR RAHMAN

STUDENT NO.: 1312029

ABU SALEH MD. ARMAN BUHUIYAN

STUDENT NO.: 1312038

&

BHUBON COSTA

STUDENT NO.: 1312045

Submitted to the

Department of NAME, BUET

In Partial Fulfillment of the Requirements for the

Degree of Bachelor in Science in Naval Architecture & Marine Engineering

Under the Supervision of

Professor Dr. Md. Mashud Karim

DEPARTMENT OF NAVAL ARCHITECTURE & MARINE ENGINEERING

BANGLADESH UNIVERSITY OF ENGINEERING & TECHNOLGY

DHAKA-1000, BANGLADESH. OCTOBER, 2018

DECLARATION

We declare that this thesis is a presentation of our original research work except for quotations and citations, which have been duly acknowledged and the thesis has not been presented in any other university for consideration of any certification. Whenever contributions of others are involved, every effort is made to indicate it clearly, with due reference to the literature and acknowledgement of collaborative research and discussions. The work was done under the guidance of Professor Dr. Mashud Karim, at the Bangladesh University of Engineering & Technology.

_____

Ashifur Rahman

_____

Abu Saleh MD. Arman Bhuiyan

_____

Bhubon Costa

ACKNOWLEDGEMENTS

The authors would like to express with due respect their deepest gratitude to their supervisor Dr. Mashud Karim, Professor, Department of Naval Architecture & Marine Engineering, BUET, whose guidance and valuable directives was encouraging at all stages of this research.

The authors are also grateful to the Library of Department of Naval Architecture & Marine Engineering, BUET for providing resources required for the successful completion of this work.

Finally they thank all the individuals who shared their knowledge through World Wide Web.

ABSTRACT

Roughly ninety percent of the world's goods are transported by sea with over seventy percent **as** containerized cargo. Most of the containers in our only sea port are handled manually. Port containers handled by human resources cost the shipping industry a lot of valuable time and resources.

Automation of port cargo handling using artificial intelligence will result in reduction of human power, less consumption of time and will be cost effective.

To handle port containers autonomously, object recognition can be used which is one type of supervised machine learning. Supervised learning, in the context of artificial intelligence (AI) and machine learning, is a type of system in which both input and desired output data are provided. Input and output data are labeled for classification to provide a learning basis for future data processing.

To detect objects or more specifically containers in this case, Google TensorFlow object detection API with python 3 as an interface is used. TensorFlow is an open-source software library for dataflow programming across a range of tasks. It is a symbolic math library, and is also used for machine learning applications such as neural networks. It is used for both research and production at Google, often replacing its closed-source predecessor, DistBelief. Several python package such as Numpy, Matplotlib, OpenCV etc. are used to attain the desired goal.

Trained Artificially intelligent model is able to detect customized container prototype with an accuracy above 98%. An automated container handling prototype was made based on this research which was able to handle container autonomously.

TABLE OF CONTENTS

List of Tables

List of Figures

NOTATIONS

t_f = Tensorflow Classification Function

t_f.get_default_graph= Tensor Default Graph Calling Function

tensor.slice= Tensor Flow Image Slicing Function

detection_boxes= Classification confidence function

image.shape[]= Images Array After Shaping

ReLU= Object Weight Factor Function

ACRONYMS AND ABBREVIATIONS

XML= Image Data Containing File
CSV= Image Dimension File
Numpy= Python Math Package
Jupyter=Python Notebook Management Package
Mathplotlib=Python Math Plotter Package
Pip= Python Package Management Tool
OpenCV=Python Video Capture Package

CHAPTER ONE

INTRODUCTION

1.1 General

Artificial intelligence (AI), sometimes called machine intelligence, is intelligence demonstrated by machines, in contrast to the natural intelligence displayed by humans and other animals[1]. In computer science AI research is defined as the study of "intelligent agents": any device that perceives its environment and takes actions that maximize its chance of successfully achieving its goals[1]. Colloquially, the term "artificial intelligence" is applied when a machine mimics "cognitive" functions that humans associate with other human minds, such as "learning" and "problem solving"[1]. The scope of AI is disputed: as machines become increasingly capable, tasks considered as requiring "intelligence" are often removed from the definition, a phenomenon known as the AI effect, leading to the quip, "AI is whatever hasn't been done yet." For instance, optical character recognition is frequently excluded from "artificial intelligence", having become a routine technology[2]. Modern machine capabilities generally classified as AI include successfully understanding human speech, competing at the highest level in strategic game systems (such as chess and Go), autonomously operating cars, and intelligent routing in content delivery networks and military simulations[2].

Artificial intelligence was founded as an academic discipline in 1956, and in the years since has experienced several waves of optimism, followed by disappointment and the

loss of funding (known as an "AI winter"), followed by new approaches, success and renewed funding. For most of its history, AI research has been divided into subfields that often fail to communicate with each other. These sub-fields are based on technical considerations, such as particular goals (e.g. "robotics" or "machine learning"), the use of particular tools ("logic" or artificial neural networks), or deep philosophical differences. Subfields have also been based on social factors (particular institutions or the work of particular researchers).

In the twenty-first century, AI techniques have experienced a resurgence following concurrent advances in computer power, large amounts of data, and theoretical understanding; and AI techniques have become an essential part of the technology industry, helping to solve many challenging problems in computer science, software engineering and operations research.

Artificial Intelligence or Machine learning tasks are typically classified into several broad categories:

- Supervised learning: The computer is presented with example inputs and their desired outputs, given by a "teacher", and the goal is to learn a general rule that maps inputs to outputs. As special cases, the input signal can be only partially available, or restricted to special feedback[3].
- Semi-supervised learning: The computer is given only an incomplete training signal: a training set with some (often many) of the target outputs missing[3].
- Active learning: The computer can only obtain training labels for a limited set of instances (based on a budget), and also has to optimize its choice of objects to acquire labels for. When used interactively, these can be presented to the user for labeling[3].
- Unsupervised learning: No labels are given to the learning algorithm, leaving it on its own to find structure in its input. Unsupervised learning can be a goal in itself (discovering hidden patterns in data) or a means towards an end (feature learning)[3].

- Reinforcement learning: Data (in form of rewards and punishments) are given only as feedback to the program's actions in a dynamic environment, such as driving a vehicle or playing a game against an opponent[3].

To achieve our objective, we worked with supervised machine learning. Supervised learning is the machine learning task of learning a function that maps an input to an output based on example input-output pairs[4]. It infers a function from *labeled training data* consisting of a set of *training examples*. In supervised learning, each example is a *pair* consisting of an input object (typically a vector) and a desired output value (also called the *supervisory signal*)[7]. A supervised learning algorithm analyzes the training data and produces an inferred function, which can be used for mapping new examples. An optimal scenario will allow for the algorithm to correctly determine the class labels for unseen instances. This requires the learning algorithm to generalize from the training data to unseen situations in a "reasonable" way

One of the important segments of supervised learning is object detection. Object detection in computer vision. Object detection is the process of finding instances of real-world objects such as faces, bicycles, and buildings in images or videos. Object detection algorithms typically use extracted features and learning algorithms to recognize instances of an object category.

Containers in ships can be detected using object detection algorithm. For this, algorithm has to be trained to detect containers on a ship. To automate this system fully, the design of containers have to be changed slightly. When the containers are stacked upon one another, the top part of the upper most containers should have two sign attached on it:

Number of containers on that stack and the products the containers are carrying. Based on that data and using object detection algorithm, automated port crane will handle the containers based on free space available.

1.2 Objective of the study

There are several objectives which have been followed in this study. The main objectives of the study are as follows:

- To automate port containers loading-unloading based on the free space available on port using machine learning object detection API.
- To categorize port containers based on the products inside the containers using machine learning
- Determine the further applicability of machine learning in shipping industry such as determining cost efficient route and increasing security using face recognition.

1.3 Scope of the Thesis

In this research work feasibility of automated port cargo handling is checked by comparing the process with manual containers handling and the evaluation of this process based on a real time trained model. The objective of this thesis is not concerned with the applicability of automation in port cargo handling. But this research can be used to determine the further applicability of machine learning and object detection in port cargo handling.

1.4 Organization of the thesis

The thesis has been organized in five chapters. In chapter one, a general introduction to the research needs along with its objectives and scope of the study are provided.

In chapter two of this dissertation, a brief review on available literature regarding characteristics and types of available neural network techniques used in the artificial intelligence field is presented. In addition, disadvantages of manual handling of containers also discussed in this chapter.

In chapter three, the process of training and building of self-learned automated port system along with machine learning technologies are discussed.

In chapter four, Log loss of different learning system are discussed and compared with one another to find out the system with minimum log loss which will detect object with maximum confidence.

In chapter five, conclusions emerged from the study are discussed. Recommendations for future research are also presented.

CHAPTER TWO

PRELEMINARIES AND LITERATURE REVIEW

2.1 General

To fulfill the objective of our thesis we are going to TensorFlow object detection API which is developed by Google. TensorFlow was developed by the Google Brain team for internal Google use. It was released under the Apache 2.0 open source license on November 9, 2015. TensorFlow will help us train our machine using Mobilenet V2 SSD model which is 35% faster than Mobilenet V1 SSD. By using this model we are going to train our machine based on around 500 images of containers prototypes that we have built for training purpose. We will train this machine to build a model which will give us an idea about automated port cargo handling which will be further used to evaluate it against manual cargo handling. In the coming section, we will discuss about the introduction of containers in shipping industry, disadvantages of manual handling of containers and brief literature review on various aspects of object detection algorithm.

2.2 Introduction of Containers

Before containerization, goods were usually handled manually as break bulk cargo. Typically, goods would be loaded onto a vehicle from the factory and taken to a port warehouse where they would be offloaded and stored awaiting the next vessel.

Containerization has its origins in early coal mining regions in England beginning in the late 18th century. In 1766 James Brindley designed the box boat 'Starvationer' with 10

wooden containers, to transport coal from Worsley Delph (quarry) to Manchester by Bridgewater Canal. In 1795, Benjamin Outram opened the Little Eaton Gangway, upon which coal was carried in wagons built at his Butterley Ironwork. The horse-drawn wheeled wagons on the gangway took the form of containers, which, loaded with coal, could be transshipped from canal barges on the Derby Canal, which Outram had also promoted. During World War II, the Australian Army used containers to help more easily deal with various breaks of gauge in the railroads. These non-stackable containers were about the size of the later 20-foot ISO container and perhaps made mainly of wood. During the same time, the United States Army started to combine items of uniform size, lashing them onto a pallet, unitizing cargo to speed the loading and unloading of transport ships. In April 1951, at Zürich Tiefenbrunnen railway station, the Swiss Museum of Transport and *Bureau International des Containers* (BIC) held demonstrations of container systems, with the aim of selecting the best solution for Western Europe. Present were representatives from France, Belgium, the Netherlands, Germany, Switzerland, Sweden, Great Britain, Italy and the United States.

2.3 Disadvantages of manual handling of containers

Manual handling of containers is time and resource consuming and often considered risky. The main bottleneck of manual handling lies on its poor handling of containers which results in a lot of time consumption as a ship has to wait on a port for very long time to be fully discharged. Another problem is obviously huge loss of resources such as too much wastage of money as this process involved too much human resources. According to Chittagong port authority to handle one container port authority charges the ship owner 65

dollars which is too much costly for a single container. This cost increases more when the charges of various agencies are also considered.

2.4 Introduction to automation: Neural Networking

An Artificial Neural Network (ANN) is an information processing paradigm that is inspired by the way biological nervous systems, such as the brain, process information. The key element of this paradigm is the novel structure of the information processing system. It is composed of a large number of highly interconnected processing elements (neurones) working in unison to solve specific problems. ANNs, like people, learn by example[3]. An ANN is configured for a specific application, such as pattern recognition or data classification, through a learning process[3]. Learning in biological systems involves adjustments to the synaptic connections that exist between the neurones. This is true of ANNs as well[4].

2.5 Architecture of Neural Networking

There are two types of Neural Networking:

- Feed-Forward Networks
- Feedback Networks

2.5.1 Feed-forward networks

Feed-forward ANNs allow signals to travel one way only; from input to output. There is no feedback (loops) i.e. the output of any layer does not affect that same layer. Feed-forward ANNs tend to be straight forward networks that associate inputs with outputs. They are extensively used in pattern recognition. This type of organization is also referred to as bottom-up or top-down[4].

2.5.2 Feedback network:

Feedback networks can have signals travelling in both directions by introducing loops in the network. Feedback networks are very powerful and can get extremely complicated. Feedback networks are dynamic; their 'state' is changing continuously until they reach an equilibrium point. They remain at the equilibrium point until the input changes and a new equilibrium needs to be found. Feedback architectures are also referred to as interactive or recurrent, although the latter term is often used to denote feedback connections in single-layer organizations[4].

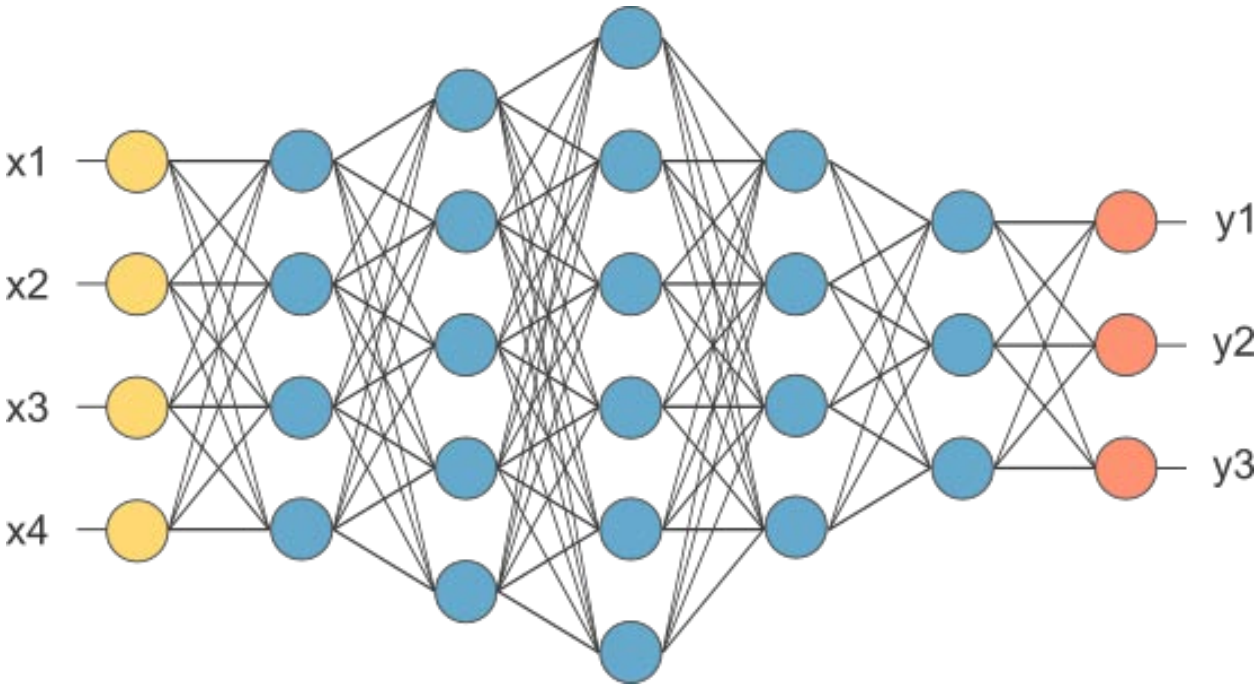

Fig 2.1: An example of a simple feedforward network

2.5.3 Network layers:

The commonest type of artificial neural network consists of three groups, or layers, of units: a layer of "input" units is connected to a layer of "hidden" units, which is connected to a layer of "output" units.

- The activity of the input units represents the raw information that is fed into the network[5].
- The activity of each hidden unit is determined by the activities of the input units and the weights on the connections between the input and the hidden units[5].

- The behavior of the output units depends on the activity of the hidden units and the weights between the hidden and output units[5].

This simple type of network is interesting because the hidden units are free to construct their own representations of the input. The weights between the input and hidden units determine when each hidden unit is active, and so by modifying these weights, a hidden unit can choose what it represents[8].

We also distinguish single-layer and multi-layer architectures. The single-layer organisation, in which all units are connected to one another, constitutes the most general case and is of more potential computational power than hierarchically structured multi-layer organisations. In multi-layer networks, units are often numbered by layer, instead of following a global numbering[8].

2.6 The Learning Process

The memorization of patterns and the subsequent response of the network can be categorized into two general paradigms:

- **Associative mapping** in which the network learns to produce a particular pattern on the set of input units whenever another particular pattern is applied on the set of input units. The associative mapping can generally be broken down into two mechanisms
    - *Auto-association*: an input pattern is associated with itself and the states of input and output units coincide. This is used to provide

pattern competition, i.e. to produce a pattern whenever a portion of it or a distorted pattern is presented. In the second case, the network actually stores pairs of patterns building an association between two sets of patterns[9].

- *hetero-association*: is related to two recall mechanisms:
    - *nearest-neighbor* recall, where the output pattern produced corresponds to the input pattern stored, which is closest to the pattern presented, and
    - *Interpolative* recall, where the output pattern is a similarity dependent interpolation of the patterns stored corresponding to the pattern presented. Yet another paradigm, which is a variant associative mapping is classification, i.e. when there is a fixed set of categories into which the input patterns are to be classified[9].

❖ **Regularity detection** in which units learn to respond to particular properties of the input patterns. Whereas in associative mapping the network stores the relationships among patterns, in regularity detection the response of each unit has a particular 'meaning'. This type of learning mechanism is essential for feature discovery and knowledge representation[10].

Every neural network possesses knowledge which is contained in the values of the connections weights. Modifying the knowledge stored in the network as a function of experience implies a learning rule for changing the values of the weights[10].

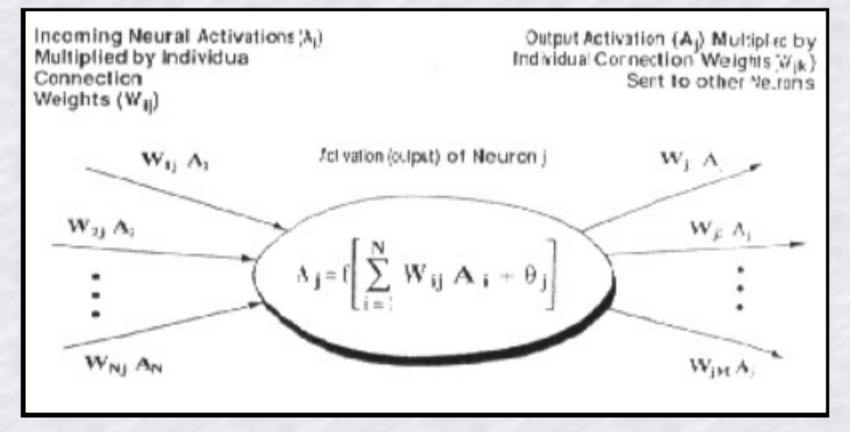

Fig 2.2: Function of a Neural Network

2.7 The Back-Propagation Algorithm

In order to train a neural network to perform some task, we must adjust the weights of each unit in such a way that the error between the desired output and the actual output is reduced. This process requires that the neural network compute the error derivative of the weights (**EW**)[4]. In other words, it must calculate how the error changes as each weight is increased or decreased slightly. The back propagation algorithm is the most widely used method for determining the **EW**[4].

The back-propagation algorithm is easiest to understand if all the units in the network are linear. The algorithm computes each **EW** by first computing the **EA**, the rate at which the error changes as the activity level of a unit is changed. For output units, the **EA** is simply the difference between the actual and the desired output[4]. To compute the **EA** for a hidden unit in the layer just before the output layer, we first identify all the weights between that hidden unit and the output units to which it is connected. We then multiply those weights by the **EA**s of those output units and add the products. This sum equals the **EA** for the chosen hidden unit. After calculating all the **EA**s in the hidden layer just before the output layer, we can compute in like fashion the **EA**s for other layers, moving from layer to layer in a direction opposite to the way activities propagate through the network. This is what gives back propagation its name. Once the **EA** has been computed for a unit, it is straight forward to compute the **EW** for each incoming connection of the unit. The **EW** is the product of the EA and the activity through the incoming connection[4].

2.8 Convolutional Neural Network

Convolutional Neural Networks are very similar to ordinary Neural Networks; they are made up of neurons that have learnable weights and biases. Each neuron receives some inputs, performs a dot product and optionally follows it with a non-linearity[4]. The whole network still expresses a single differentiable score function: from the raw image pixels on one end to class scores at the other. And they still have a loss function (e.g.

SVM/Softmax) on the last (fully-connected) layer and all the methods for learning regular Neural Networks still apply[4].

2.9 Architecture Overview

A simple ConvNet is a sequence of layers,and every layer of a ConvNet transforms one volume of activations to another through a differentiable function. We use three main types of layers to build ConvNet architectures: Convolutional Layer, Pooling Layer, and Fully-Connected Layer. We will stack these layers to form a full ConvNet architecture Computers read images as pixels and it is expressed as matrix (N ×N ×3) (height by width by depth)[5]. Images make use of three channels (rgb), so that is why we have a depth of 3. The Convolutional Layer makes use of a set of learnable filters. A filter is used to detect the presence of specific features or patterns present in the original image (input). It is usually expressed as a matrix (M×M×3) ,with a smaller dimension but the same depth as the input file. This filter is convolved (slided) across the width and height of the input file, and a dot product is computed to give an activation map[5][8].

Different filters which detect different features are convolved on the input file and a set of activation maps is outputted which is passed to the next layer in the CNN[6][7].

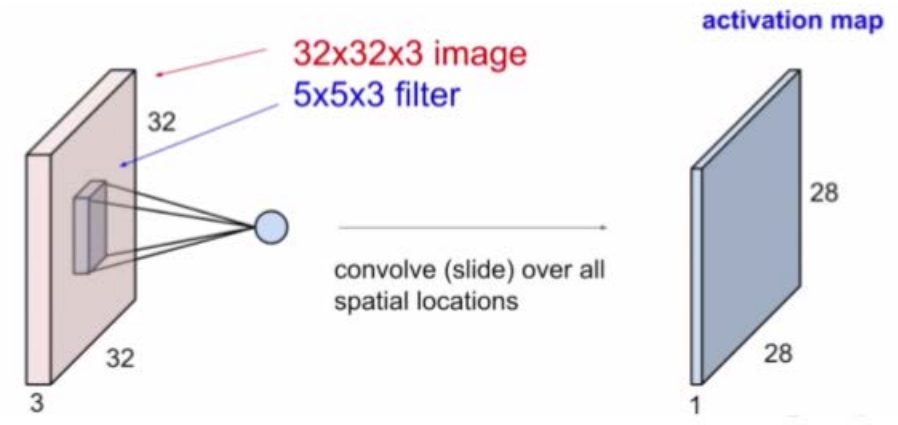

Figure 2.3: The use of filters to get Activation Maps

Activation function is a node that is put at the end of or in between Neural Networks. They help to decide if the neuron would fire or not[8].

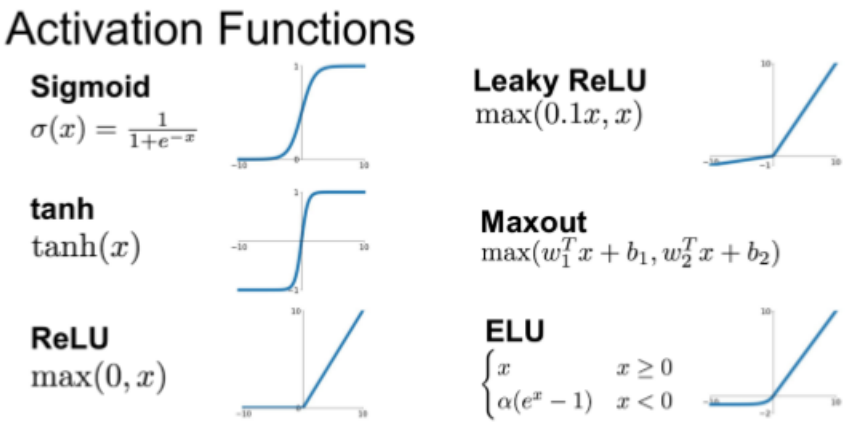

Figure 2.4: Activation functions

ReLU function is the most widely used activation function in neural networks today. One of the greatest advantages ReLU has over other activation functions is that it does not activate all neurons at the same time. From the image for ReLU function above, well notice that it converts all negative inputs to zero and the neuron does not get activated[6][7]. This makes it very computational efficient as few neurons are activated per time. It does not saturate at the positive region. In practice, ReLU converges six times faster than tanh and sigmoid activation functions[9].

The Pooling layer can be seen between Convolution layers in CNN architecture. This layer basically reduces the amount of parameters and computation in the network, controlling over fitting by progressively reducing the spatial size of the network. There are two operations in this layer; Average pooling and maximum pooling[7]. Max-pooling,

like the name states; will take out only the maximum from a pool. This is actually done with the use of filters sliding through the input; and at every stride, the maximum parameter is taken out and the rest is dropped[7]. This actually down-samples the network. Unlike the convolution layer, the pooling layer does not alter the depth of the network, the depth dimension remains unchanged[10].

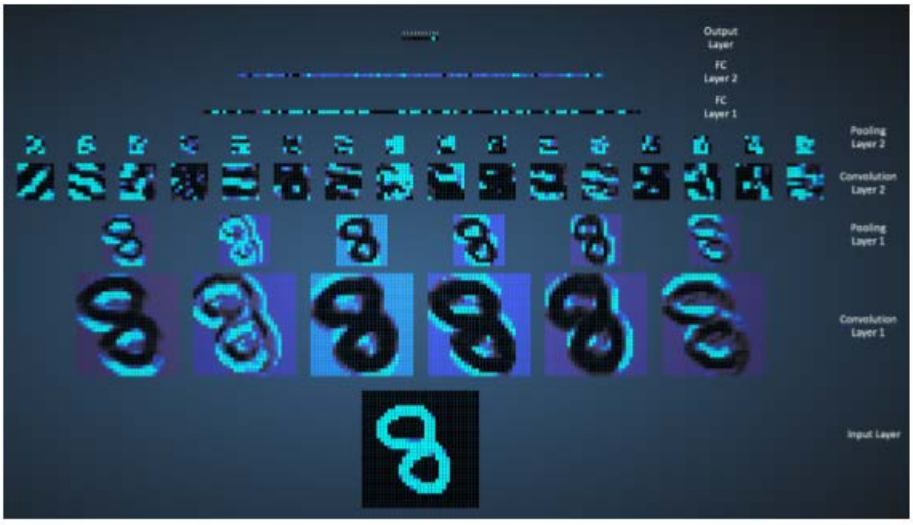

Fig 2.6: All layers of convolutional neural network demonstration

CHAPTER THREE

CLASSIFICATION AND DATA TRAINING PROCESS

3.1 Introduction

As discussed earlier, TensorFlow object detection API is used to train the model. For programming purpose we have used Python 3.7. We have also used several python packages to fulfill our objective. Besides of using TensorFlow Api, TensorFlow python package also used to integrate it with python. Numpy package is used to make image array. Matplotlib is used to plot data and opencv to capture video feed. In this chapter we will discuss about these features and the process of training the machine.

3.2 TensorFlow object detection API

Creating accurate machine learning models capable of localizing and identifying multiple objects in a single image remains a core challenge in computer vision. The TensorFlow Object Detection API is an open source framework built on top of TensorFlow that makes it easy to construct, train and deploy object detection models. It is used for both research and production at Google. TensorFlow was developed by the Google Brain team for internal Google use. It was released under the Apache 2.0 open source license on November 9, 2015.

3.2.1 Working procedure of TensorFlow

Basics of TensorFlow are that first we create a model which is called a computational graph with TensorFlow objects then we create a TensorFlow session in which we start running all the computation[8]. Libraries like TensorFlow are not simply deep learning libraries; they are libraries for deep learning. They are actually just number-crunching libraries, much like Numpy is. The difference is, however, a package like TensorFlow allows us to perform specific machine learning number-crunching operations like derivatives on huge matrices with large efficiency[8]. We can also easily distribute this processing across our CPU cores, GPU cores, or even multiple devices like multiple GPUs. But that's not all! We can even distribute computations across a distributed network of computers with TensorFlow. So, while TensorFlow is mainly being used with machine learning right now, it actually stands to have uses in other fields, since really it is just a massive array manipulation library[5].

What is a tensor? Up to this point in the machine learning series, we've been working mainly with vectors (numpy arrays), and a tensor can be a vector. Most simply, a tensor is an array-like object, and, as you've seen, an array can hold your matrix, your vector, and really even a scalar[6].

At this point, we just simply need to translate our machine learning problems into functions on tensors, which is possible with just about every single ML algorithm[8]. Consider the neural network. What does a neural network break down into?

We have data (X), weights (w), and thresholds (t). Are all of these tensors? X will be the dataset (an array), so that's a tensor. The weights are also an array of weight values, so they're tensors too. Thresholds? Same as weights. Thus, our neural network is indeed a function of X,w, and t, or f(Xwt), so we are all set and can certainly use TensorFlow, but how?

TensorFlow works by first defining and describing our model in abstract, and then, when we are ready, we make it a reality in the session. The description of the model is what is known as your "Computation Graph" in TensorFlow terms. Let's play with a simple example. First, let's construct the graph:

```
import tensorflow as tf

# creates nodes in a graph
# "construction phase"
x1 = tf.constant(5)
x2 = tf.constant(6)
```

```
result = tf.mul(x1,x2)
print(result)
```

Fig 3.1: TensorFlow graph construction algorithm

So we have some values. Now, we can do things with those values, such as multiplication.

Notice that the output is just an abstract tensor still. No actual calculations have been run, only operations created. Each operation, or "op," in our computation graph is a "node" in the graph[9].

To actually see the result, we need to run the session. Generally, you build the graph first, then you "launch" the graph:

```
# defines our session and launches graph
sess = tf.Session()
# runs result
print(sess.run(result))
```

Fig 3.2: TensorFlow Launching Graph

3.2.2 Installation procedure of TensorFlow

TensorFlow is installed with Python's *pip* package manager. Official packages available for Ubuntu, Windows, macOS, and the Raspberry Pi.

```
# Current release for CPU-only
$ pip install tensorflow

# Nightly build for CPU-only (unstable)
$ pip install tf-nightly

# GPU package for CUDA-enabled GPU cards
$ pip install tensorflow-gpu

# Nightly build with GPU support (unstable)
$ pip install tf-nightly-gpu
```

Fig 3.3: TensorFlow Installation

3.3 Python and pip3

Python is an interpreted high-level programming language for general-purpose programming. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace. Python 3.7 has been downloaded from its official site and installed on local machine.

pip is a package management system used to install and manage software packages written in Python. Many packages can be found in the default source for packages and their dependencies — Python Package Index (PyPI). pip is a recursive acronym for "**Pip** Installs Packages".

3.4 Installation of Python Packages

We have used several python packages to build a successful interface between Python 3.7 and TensorFlow object detection API. All these packages have own working and installation procedure.

3.4.1 Jupyter Notebook

Jupyter Notebook is used to organize our project. It works Like an IDE. The Jupyter Notebook is an open-source web application that allows one to create and share documents that contain live code, equations, visualizations and narrative text. Uses include: data cleaning and transformation, numerical simulation, statistical modeling,

data visualization, machine learning, and much more. Python is a prerequisite for python. We have installed Jupyter with python pip3 command line.

```
python3 -m pip install --upgrade pip
python3 -m pip install jupyter
```

Fig 3.4: Jupyter Installation with pip3

3.4.2 Numpy

Numpy is used to make image array. Images that we have taken for our training and testing purposes are categorized into array using Numpy. NumPy is the fundamental package for scientific computing with Python. It contains among other things:

- a powerful N-dimensional array object
- sophisticated (broadcasting) functions
- tools for integrating C/C++ and Fortran code
- useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

3.4.3 Matplotlib

Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib

can be used in Python scripts, the Python and IPython shells, the Jupyter notebook, web application servers, and four graphical user interface toolkits.

```
python -m pip install -U pip
python -m pip install -U matplotlib
```

Fig 3.5: Matplotlib installation with pip3

3.4.4 OpenCV

OpenCV is a library of programming functions mainly aimed at real-time computer vision. Originally developed by Intel, it was later supported by Willow Garage then Itseez. The library is cross-platform and free for use under the open-source BSD license.

OpenCV was designed for computational efficiency and with a strong focus on real-time applications. Written in optimized C/C++, the library can take advantage of multi-core processing. Enabled with OpenCL, it can take advantage of the hardware acceleration of the underlying heterogeneous compute platform.

3.5 Training Process

For Training Process we have taken around 500 photos of our model from different angle. After that we labeled the container in the image to make it detectable for TensorFlow. For this purpose we used tzutalin/levelimg. Using tzutalin/levelimg we have made border around our container boxes in our images and give the labeled images a class. We use this procedure to make the container detectable to TensorFlow object

detection API. After leveling our image, Levelimg package make a .xml file containing matching annotation information for each image. We have used protoc-3.6.0-win32 to serialize our data. All of TensorFlow's file formats are based on Protocol Buffers. Protocol buffers are Google's language-neutral, platform-neutral, extensible mechanism for serializing structured data – think XML, but smaller, faster, and simpler. One can define how he wants his data to be structured once, and then he can use special generated source code to easily write and read his structured data to and from a variety of data streams and using a variety of languages.

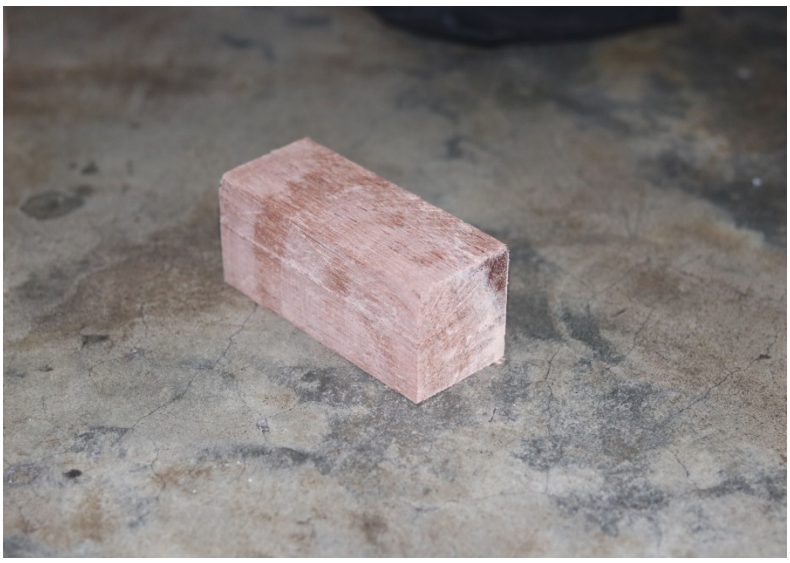

Fig 3.6: Wooden container model used for training purposes

We have split our image files along with their respective .xml files in two folders: train and test.10% of images with their respective xml files are put into test folder and the rest

90% are put into train folder. Training process requires a large amount of data as the neural network learns from different view point.

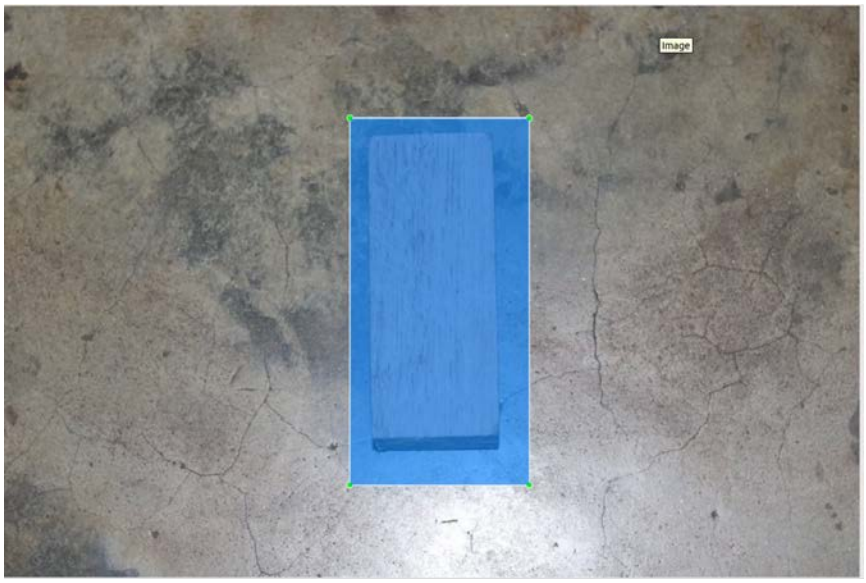

Fig 3.7: Leveled images using tzutalin/levelimg

We have converted our .xml files to two.csv file ***train_levels.csv*** and ***test_levels.csv*** using ***xml_to_csv.py*** file. A CSV is a comma separated values file which allows data to be saved in a table structured format. CSVs look like a garden-variety spreadsheet but with a .csv extension. Traditionally they take the form of a text file containing information separated by commas, hence the name. After converting .xml to .csv, we have converted .csv files into .tfr file using ***generate_tfrecord.py***. Produced ***train.record*** was used to train the machine.

If someone is working with large datasets, using a binary file format for storage of data can have a significant impact on the performance of import pipeline and as a consequence on the training time of model. Binary data takes up less space on disk, takes less time to copy and can be read much more efficiently from disk. This is especially true if data is stored on spinning disks, due to the much lower read/write performance in comparison with SSDs. However, pure performance isn't the only advantage of the TFRecord file format. It is optimized for use with TensorFlow in multiple ways. To start with, it makes it easy to combine multiple datasets and integrates seamlessly with the data import and preprocessing functionality provided by the library. Especially for datasets that are too large to be stored fully in memory this is an advantage as only the data that is required at the time (e.g. a batch) is loaded from disk and then processed. Another major advantage of TFRecords is that it is possible to store sequence data—for instance, a time series or word encodings—in a way that allows for very efficient and (from a coding perspective) convenient import of this type of data.

	A	B	C	D	E	F	G	H
1	filename	width	height	class	xmin	ymin	xmax	ymax
2	8.JPG	1152	768	container	298	154	591	582
3	8.JPG	1152	768	container	705	167	961	566
4	9.JPG	1152	768	container	251	148	710	511

Fig 3.8: Generated CSV file

We have used ***ssd_mobilenet_v1_coco_11_06_2017*** as a training model because it is faster than other model. so, it takes less time to train model. After, configuring all our

files, we started to train our machine to detect our container model. As we don't have a dedicated GPU and our machine only run on a CPU. Our machine is not trained properly. After training process our total loss was below 2. That means our machine can detect within the range of 98% accuracy. For a decent training, it would take 10,000 iteration but we took only around 200 iteration.

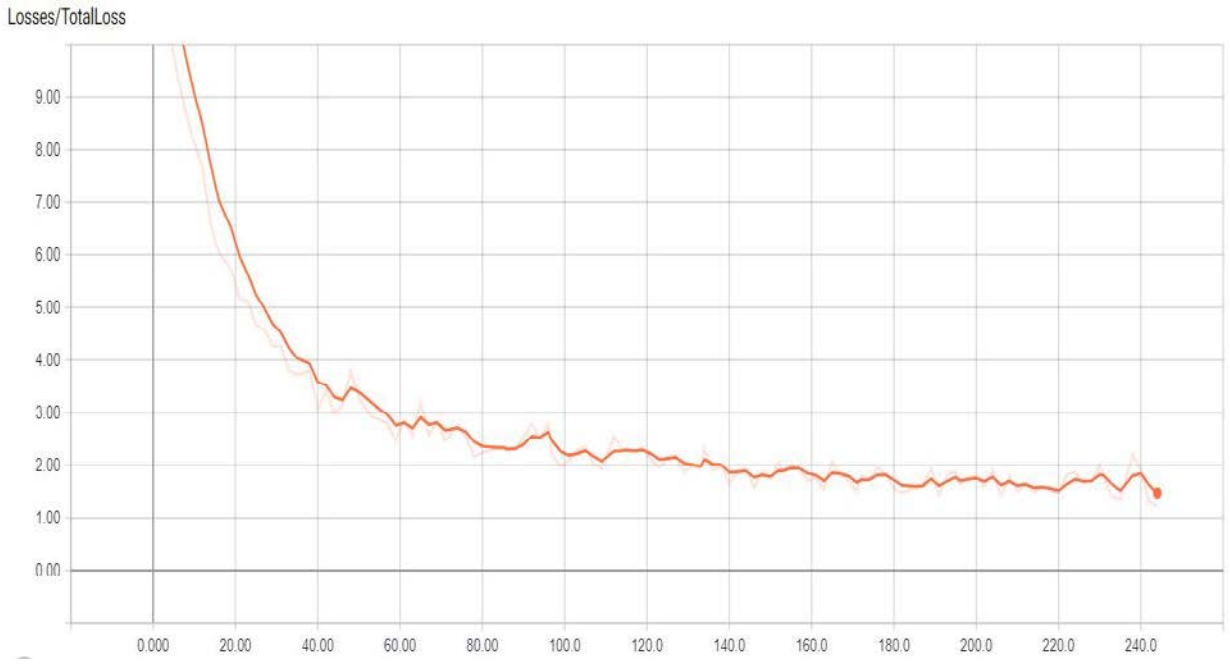

Fig 3.9: Learning Curve

3.6 Testing process

To test our machine we have used two types of data: Static test images and live video feed from camera. The purpose of using two types of data is to observe if our machine could detect container in real world situation.

```
def run_inference_for_single_image(image, graph):
  with graph.as_default():
    with tf.Session() as sess:
      # Get handles to input and output tensors
      ops = tf.get_default_graph().get_operations()
      all_tensor_names = {output.name for op in ops for output in op.outputs}
      tensor_dict = {}
      for key in [
          'num_detections', 'detection_boxes', 'detection_scores',
          'detection_classes', 'detection_masks'
      ]:
        tensor_name = key + ':0'
        if tensor_name in all_tensor_names:
          tensor_dict[key] = tf.get_default_graph().get_tensor_by_name(
              tensor_name)
      if 'detection_masks' in tensor_dict:
        # The following processing is only for single image
        detection_boxes = tf.squeeze(tensor_dict['detection_boxes'], [0])
        detection_masks = tf.squeeze(tensor_dict['detection_masks'], [0])
        # Reframe is required to translate mask from box coordinates to image coordinates and fit the image size.
        real_num_detection = tf.cast(tensor_dict['num_detections'][0], tf.int32)
        detection_boxes = tf.slice(detection_boxes, [0, 0], [real_num_detection, -1])
        detection_masks = tf.slice(detection_masks, [0, 0, 0], [real_num_detection, -1, -1])
        detection_masks_reframed = utils_ops.reframe_box_masks_to_image_masks(
            detection_masks, detection_boxes, image.shape[0], image.shape[1])
        detection_masks_reframed = tf.cast(
            tf.greater(detection_masks_reframed, 0.5), tf.uint8)
        # Follow the convention by adding back the batch dimension
        tensor_dict['detection_masks'] = tf.expand_dims(
            detection_masks_reframed, 0)
      image_tensor = tf.get_default_graph().get_tensor_by_name('image_tensor:0')

      # Run inference
      output_dict = sess.run(tensor_dict,
                             feed_dict={image_tensor: np.expand_dims(image, 0)})

      # all outputs are float32 numpy arrays, so convert types as appropriate
      output_dict['num_detections'] = int(output_dict['num_detections'][0])
      output_dict['detection_classes'] = output_dict[
          'detection_classes'][0].astype(np.uint8)
      output_dict['detection_boxes'] = output_dict['detection_boxes'][0]
      output_dict['detection_scores'] = output_dict['detection_scores'][0]
      if 'detection_masks' in output_dict:
        output_dict['detection_masks'] = output_dict['detection_masks'][0]
  return output_dict
```

Fig 3.10: Portion of Object detection algorithm

3.6.1 Testing process using static images

We have separated 10% of our captured images to test our trained machine. As our loss is less than 2% after 2400 epoch so our machine has an ability to detect containers in

static images with at least 98% accuracy. Our trained machine did not learn to identify or classify any other objects than containers. But the static images have an advantages than live video feed. In static images possibility of presence of other objects is too small and it takes less CPU and GPU memory to classify the desired object which in our case is container.

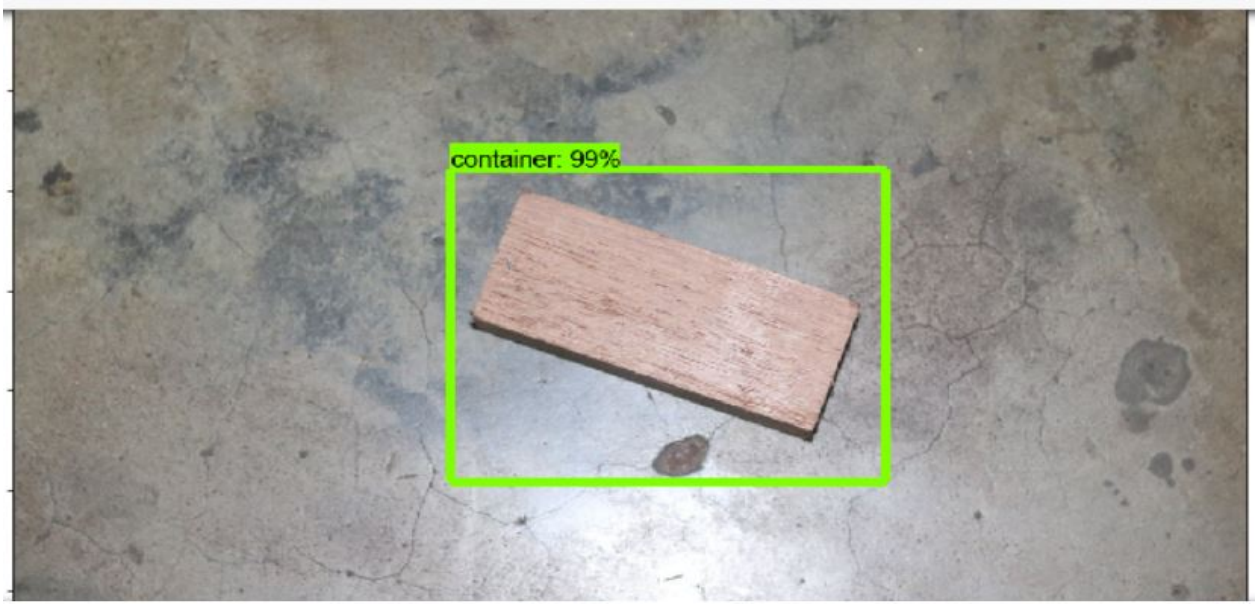

Fig 3.11: Detection of single container in an Image

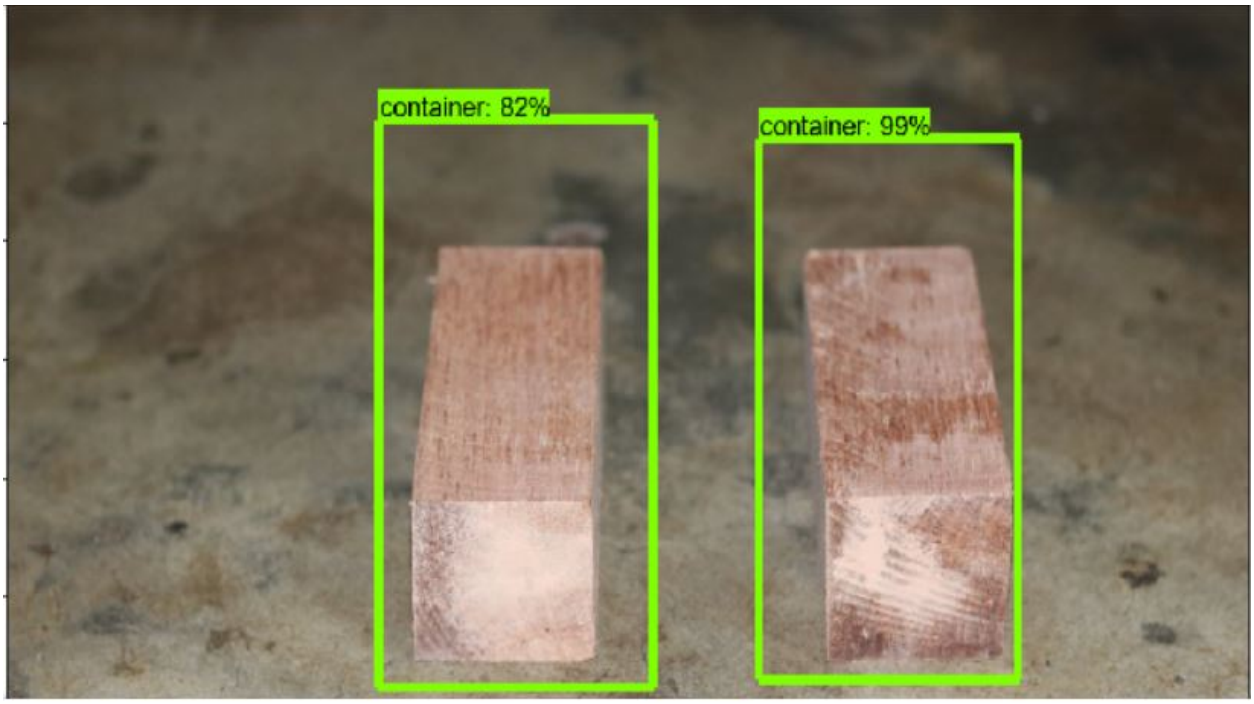

Fig 3.12: Detection of multiple container in an Image

3.6.2 Testing process using live video feed

One of the biggest challenges of our research was detecting container from live video feed. We have used OpenCV for this purpose. OpenCV captured video feed from our default web cam and then pass it to our trained machine to detect container from it. As we only run on CPU, detection process was quite slow. Another challenge of our research was calculating the position of detected containers. For this purpose we have subdivided our screen into two grids and developed an algorithm that will send a specific message when a container is detected on a grid. There is a bottleneck of detecting containers from live video feed. As our machine is not yet trained to detect all objects it can encounter in

a port area, so it will classify or detect any other unfamiliar objects as container with higher rate of accuracy. We have solved this problem in two steps. We have increased the

Fig 3.13: Real time multiple object detection from live video feed

threshold of accuracy from 70% to 90%. The second step is to take snaps from live video feed so that the machine will run the object detection algorithm only for a short time. This will result in a less memory consumption of CPU and GPU and the machine will have to handle less objects.

3.7 Prototype of automated port handling mechanism

We have tried to build a prototype to demonstrate how automated port handling mechanism. We have used two stepper motors to mimic the process. Arduino UNO is used to make an interface between hardware and object detection API. For this we have used Python Pyserial package to enable the objection detection API to communicate with arduino. We have sent signals when a container is detected on a grid. The signals vary depend on which grid has the container. Based on the position of container on a specific grid, Pyserial package sends a specific signal based on the location where the container is detected to the serial port which enables the arduino to rotate the servo motor to a specific rotation number which in turns send the crane to the specific location of the containers and the crane handles the containers based on free space available.

```
on_right, on_left = False, False
for i in range(len(output_dict['detection_boxes'])):
  if(output_dict['detection_scores'][i] >  CONTAINER_THRESHOLD):
    rect = output_dict['detection_boxes'][i]
    mid_x = (rect[1] + rect[3]) / 2
    print(rect)
    if(mid_x < .5):
      on_left = True
    else:
      on_right = True
    print(on_right, on_left)

x=1

if(on_left and x==1):
  cv2.putText(image_np,"Container on the Left Grid", (100, 100), cv2.FONT_HERSHEY_SIMPLEX, 0.6, 255)
  ser.write(str.encode('2'))
  x=0

if(on_right and x==1):
  cv2.putText(image_np,"Container on the Right Grid", (330, 100), cv2.FONT_HERSHEY_SIMPLEX, 0.6, 255)
  ser.write(str.encode('1'))
  x=0

image_np[:, 320 - 2 : 320 + 2] = [255, 0, 0]
cv2.imshow('object detection', cv2.resize(image_np, (640, 480)))
cv2.imwrite('container_detection.jpg', cv2.resize(image_np, (640, 480)))
else:
  cv2.imshow('object detection', image_np)
  print('Standby ')
if cv2.waitKey(25) & 0xFF == ord('q'):
  cv2.destroyAllWindows()
  break
```

Fig 3.14: Detection API signal Algorithm

```
  if(Serial.available() > 0)       // Send data only when you receive data:
   {
      data = Serial.read();         //Read the incoming data & store into data
      Serial.print(data);           //Print Value inside data in Serial monitor
      Serial.print("\n");
      if(data == '1') // Checks whether value of data is equal to 1
      {

   // Enables the motor to move in a particular direction
   //Makes 200 pulses for making one full cycle rotation

  delay(1000); // One second delay

  digitalWrite(dirPin,HIGH); //Changes the rotations direction
  // Makes 400 pulses for making two full cycle rotation
  for(int x = 0; x < 8000; x++) {
    digitalWrite(stepPin,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin,LOW);
    delayMicroseconds(500);
  }
   delay(2000);
for(int x = 0; x < 8000; x++) {
  digitalWrite(dirPin,LOW);
    digitalWrite(stepPin,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin,LOW);
    delayMicroseconds(500);
  }

   }
```

Fig 3.15: Arduino signal manipulation algorithm

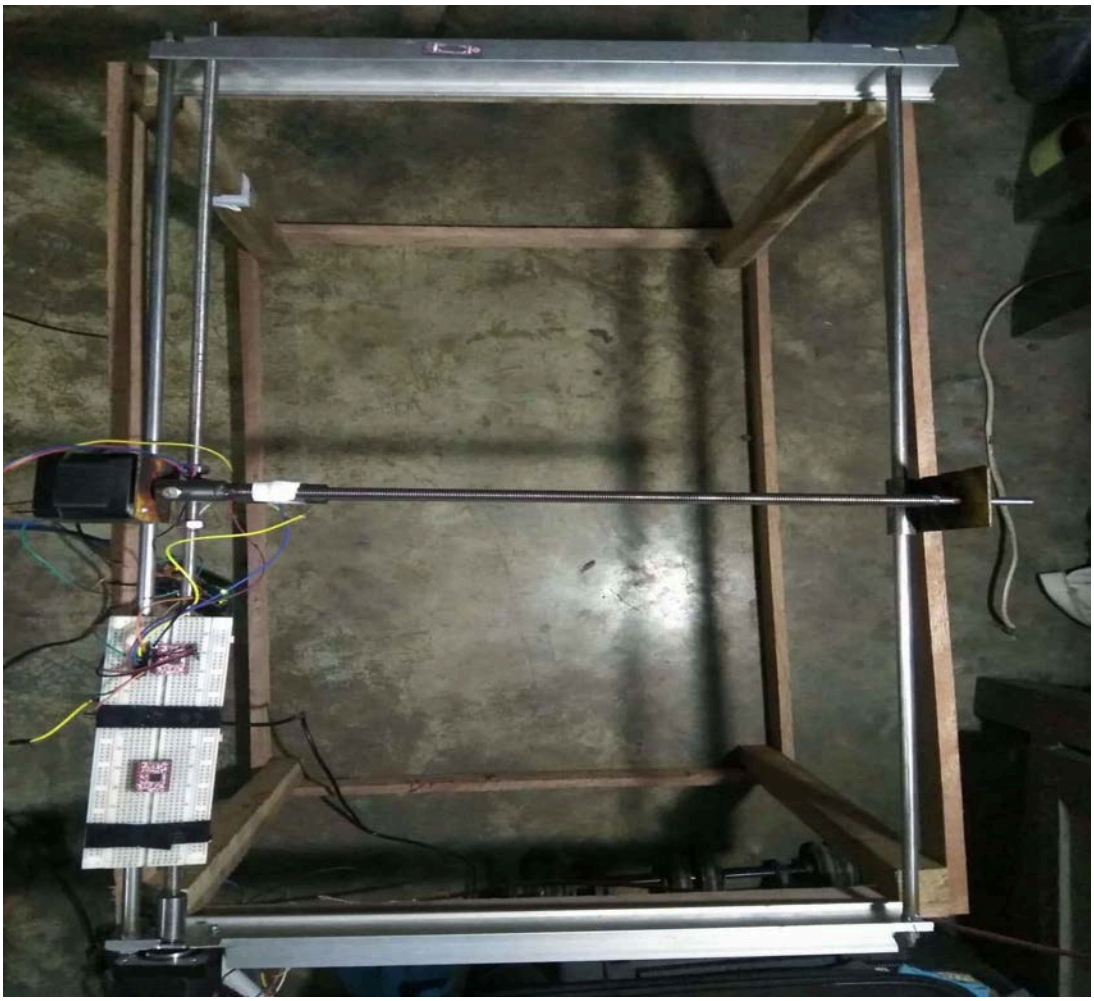

Fig 3.16: Prototype of automated port cargo handler

CHAPTER FOUR

EXPERIMENTAL RESULTS

4.1 Introduction

We have discussed the procedure to obtain our objective last chapter. As we have not any dedicated GPU and high end CPU, we have failed to train our model with 100% efficiency. To train the machine we have used

- An Intel Core i5 7th Gen CPU
- NVIDIA GEFORCE 2GB GPU

4.2 Total loss

We have got ***Loss/BoxClassifierLoss/classification_loss/mul_1*** as below 2. Loss for the classification of detected objects into various classes: Cat, Dog, Airplane etc. In our case it will be loss for the classification of detected containers into one single class which is container. As our loss is slightly higher than 1, so our machine will detect the object with a confidence percentage of minimum 98%. The loss has been measured using TensorBoard graph API.

4.3 Total loss comparison for different models

Table 4.1: Experimental Results

Input Image	Model	Logloss
Raw Image	CNN	.78
Raw Image	Resnet 50	.70
Threshold segmented nodules	Resnet50	2.91
Unet segmented nodules	Resnet50	1.27
Raw Image	Resnet50, xgboost	.61
Threshold segmented nodules	Resnet50, xgboost	1.09
Unet segmented nodules	Resnet50, xgboost	.55

4.4 Conclusion

The lowest Log Loss in our experiment is .55. Although this is a good Log Loss, but it is not the best. The lowest Log Loss that has been reported for this particular dataset is .39. So there are a lot of rooms for optimization. We have plan for continuing to work on this problem. In the next chapter we'll discuss our future plan which might decrease the Log Loss even further.

CHAPTER FIVE

CONCLUSIONS

5.1 Introduction

As we've seen in our experimental result that our result isn't the best result out there. But there are a lot of improvements. Below is some improvements that can be done but we couldn't do it due to limited time and resources.

- The accuracy of the built machine is more than 98% for detecting containers. However, it is possible to increase the accuracy of the prototype machine up to 100% using high quality equipment and machine parts.
- Using this system cargo handling of port can automated and time and cost of handling can be reduced.
- One big improvement in the result should be seen if we could use ResNet101 or maybe ResNet152. But that will take so much time and resources to train on such huge data which we can't afford at this moment.
- Working on 3D blobs rather than 2D blobs.
- Gathering more data for training.
- We have failed to classify the containers based on the project inside and one of the drawback of our research is our trained model can work for only one container i.e., it can handle only one container at a time but it

cannot handle a stack of containers continuously. So, there is a scope of improving this research based on these shortcomings.

- As this system is automated, a closed loop control system can be introduced with controller function (G_c), process function(G), Sensor function (H). All the disturbance should be taken into account.

Hopefully in near future we'll get enough resource to train and improve the ideas that we couldn't test this time.

REFERENCES

1. Canziani, A. Paszke, and E. Culurciello, “An analysis of deep neural network models for practical applications,” arXiv preprint arXiv: 1605.07678, 2016

2. S. Lawrence, C. L. Giles, A. C. Tsoi, and A. D. Back, “Face recognition: A convolutional neural-network approach,” IEEE transactions on neural networks, vol. 8, no. 1, pp. 98–113, 1997.

3. O. Ronneberger, P. Fischer, and T. Brox, “U-net: Convolutional networks for biomedical image segmentation,” CoRR, vol. abs/1505.04597, 2015.

4. K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” CoRR, vol. abs/1512.03385, 2015.

5. Statistical Decision Making for Optimal Budget Allocation in Crowd Labeling
Xi Chen, Qihang Lin, Dengyong Zhou; 16(Jan):1−46, 2015.

6. M. A. Hearst, S. T. Dumais, E. Osuna, J. Platt, and B. Scholkopf, “Support vector machines,” IEEE Intelligent Systems and their applications, vol. 13, no. 4, pp. 18–28, 1998.

7. T. Chen and C. Guestrin, “Xgboost: A scalable tree boosting system,” in Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining, pp. 785–794, ACM, 2016.

8. D.A.vanDykandX.-L.Meng,“Theartofdataaugmentation,”JournalofComputational and Graphical Statistics, vol. 10, no. 1, pp. 1–50, 2001.

9. O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al., “Imagenet large scale visual recognition challenge,” International Journal of Computer Vision, vol. 115, no. 3, pp. 211–252, 2015.

10. D. M. Hawkins, “The problem of overfitting,” Journal of chemical information and computer sciences, vol. 44, no. 1, pp. 1–12, 2004.

11. R. E. Schapire, “The boosting approach to machine learning: An overview,” in Nonlinear estimation and classification, pp. 149–171, Springer, 2003.

12. T. Hastie, S. Rosset, J. Zhu, and H. Zou, “Multi-class adaboost,” Statistics and its Interface, vol. 2, no. 3, pp. 349–360, 2009.

APPENDIX

Source Code:

```
model {
  ssd {
    num_classes: 1
    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
      }
    }
    similarity_calculator {
      iou_similarity {
      }
    }
    anchor_generator {
      ssd_anchor_generator {
        num_layers: 6
        min_scale: 0.2
        max_scale: 0.95
        aspect_ratios: 1.0
        aspect_ratios: 2.0
        aspect_ratios: 0.5
        aspect_ratios: 3.0
        aspect_ratios: 0.3333
      }
    }
    image_resizer {
      fixed_shape_resizer {
        height: 300
        width: 300
      }
    }
    box_predictor {
      convolutional_box_predictor {
        min_depth: 0
        max_depth: 0
        num_layers_before_predictor: 0
        use_dropout: false
```

Fig A.1: Source Code-1

```
import numpy as np
import os
import six.moves.urllib as urllib
import sys
import tarfile
import tensorflow as tf
import zipfile

from collections import defaultdict
from io import StringIO
from matplotlib import pyplot as plt
from PIL import Image
import cv2
import serial
cap=cv2.VideoCapture(1)

#serial communication ----------
ser=serial.Serial('com4',9600)


# This is needed since the notebook is stored in the object_detection folder.
sys.path.append("..")
from object_detection.utils import ops as utils_ops

if tf.__version__ < '1.4.0':
  raise ImportError('Please upgrade your tensorflow installation to v1.4.* or later!')
```

Fig A.2: Source Code-2

```
import tensorflow as tf
from google.protobuf import text_format
from object_detection import exporter
from object_detection.protos import pipeline_pb2

slim = tf.contrib.slim
flags = tf.app.flags

flags.DEFINE_string('input_type', 'image_tensor', 'Type of input node. Can be '
                    'one of [`image_tensor`, `encoded_image_string_tensor`, '
                    '`tf_example`]')
flags.DEFINE_string('input_shape', None,
                    'If input_type is `image_tensor`, this can explicitly set '
                    'the shape of this input tensor to a fixed size. The '
                    'dimensions are to be provided as a comma-separated list '
                    'of integers. A value of -1 can be used for unknown '
                    'dimensions. If not specified, for an `image_tensor, the '
                    'default shape will be partially specified as '
                    '`[None, None, None, 3]`.')
flags.DEFINE_string('pipeline_config_path', None,
                    'Path to a pipeline_pb2.TrainEvalPipelineConfig config '
                    'file.')
flags.DEFINE_string('trained_checkpoint_prefix', None,
                    'Path to trained checkpoint, typically of the form '
                    'path/to/model.ckpt')
flags.DEFINE_string('output_directory', None, 'Path to write outputs.')
flags.DEFINE_string('config_override', '',
                    'pipeline_pb2.TrainEvalPipelineConfig '
                    'text proto to override pipeline_config_path.')
tf.app.flags.mark_flag_as_required('pipeline_config_path')
tf.app.flags.mark_flag_as_required('trained_checkpoint_prefix')
tf.app.flags.mark_flag_as_required('output_directory')
FLAGS = flags.FLAGS


def main(_):
  pipeline_config = pipeline_pb2.TrainEvalPipelineConfig()
  with tf.gfile.GFile(FLAGS.pipeline_config_path, 'r') as f:
    text_format.Merge(f.read(), pipeline_config)
  text_format.Merge(FLAGS.config_override, pipeline_config)
  if FLAGS.input_shape:
    input_shape = [
        int(dim) if dim != '-1' else None
        for dim in FLAGS.input_shape.split(',')
    ]
  else:
    input_shape = None
  exporter.export_inference_graph(FLAGS.input_type, pipeline_config,
                                  FLAGS.trained_checkpoint_prefix,
                                  FLAGS.output_directory, input_shape)


if __name__ == '__main__':
  tf.app.run()
```

Fig A.3: Source Code-3

```
import os
import glob
import pandas as pd
import xml.etree.ElementTree as ET


def xml_to_csv(path):
    xml_list = []
    for xml_file in glob.glob(path + '/*.xml'):
        tree = ET.parse(xml_file)
        root = tree.getroot()
        for member in root.findall('object'):
            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)
    return xml_df


def main():
  for directory in ['train','test']:
    image_path = os.path.join(os.getcwd(), 'images/{}'.format(directory))
    xml_df = xml_to_csv(image_path)
    xml_df.to_csv('data/{}_labels.csv'.format(directory), index=None)
    print('Successfully converted xml to csv.')


main()
```

Fig A.4: Source Code-4

```
Serial.print("Forward");

digitalWrite(dirPin1,HIGH); //Changes the rotations direction
// Makes 400 pulses for making two full cycle rotation
for(int x = 0; x < 32000; x++) {
  digitalWrite(stepPin1,HIGH);
  delayMicroseconds(500);
  digitalWrite(stepPin1,LOW);
  delayMicroseconds(500);
}
Serial.print("CNTAINER HANDING DEAY.....");
delay(2000);
    //for Y Axis

  digitalWrite(6,HIGH); //Changes the rotations direction
// Makes 400 pulses for making two full cycle rotation
for(int x = 0; x < 12000; x++) {

  digitalWrite(stepPin2,HIGH);
  delayMicroseconds(500);
  digitalWrite(stepPin2,LOW);
  delayMicroseconds(500);

}
 delay(2000);
  digitalWrite(6,LOW); //Changes the rotations direction
// Makes 400 pulses for making two full cycle rotation
for(int x = 0; x < 12000; x++) {
  digitalWrite(stepPin2,HIGH);
  delayMicroseconds(500);
  digitalWrite(stepPin2,LOW);
  delayMicroseconds(500);

}
```

Fig A.5: Source Code-5

How to Operate the Autonomous Equipment:

1. A PC with a minimum 2 GB GPU will be required to run the machine.
2. The PC must have TensorFlow API and Python 3 installed on it along with necessary Python packages described on this document.
3. A web camera or any sort of visioning device will be required to enable the visioning capability of the equipment.
4. An Arduino UNO and necessary programming IDE will be required.
5. To enable the machine to classify customized object necessary command should be run in specific directory from command prompt. Such as: ***python3 obeject_detection_container.py***