DE-37 (DC&SE)

NUST COLLEGE OF
ELECTRICAL AND MECHANICAL ENGINEERING

Vehicle Tagging Using Deep Learning

PROJECT REPORT

DE-37 (DC&SE)

Submitted by

NS AMMAR AMJAD

NS TAIMUR IBRAHIM

PC AZIZ UR REHMAN

NS SYED MOHAMMAD ALI

BACHELORS

IN

COMPUTER ENGINEERING

YEAR

2019

PROJECT SUPERVISORS

DR. USMAN AKRAM

DR. SAJID GUL KHAWAJA

COLLEGE OF

ELECTRICAL AND MECHANICAL ENGINEERING

YEAR 2019

Declaration

We hereby declare that no portion of the work referred to in this Project Thesis has been submitted in support of an application for any other degree or qualification of this for any other university. If any act of plagiarism found, we are fully responsible for every disciplinary action taken against us depending upon the seriousness of the proven offence.

Copyright Statement

Acknowledgments

We would like to acknowledge everyone who played a role in our academic achievements.

First, our parents who loved and supported us in throughout our lives especially through our academic endeavors. Without their constant support we would not have been so fortunate.

Secondly, the faculty and especially our supervisors who guided us on every wrong turn and helped us see the error in our ways.

The vast experience of our teachers and their consistent advice helped us prevent errors which could not have foreseen throughout the research process and the implementation phase.

Thank you all for your unwavering support.

This page is intentionally left blank

Table of Contents

Abstract

Traditional road monitoring or security systems usually employ people to manually monitor the camera feeds and note the license plate numbers of the vehicles of interest. These types of models have two main disadvantages. Firstly, they require humans to be working constantly to watch the footage and note the numbers and secondly the video needs to be transmitted from the street cameras to the central offices where the humans would usually be monitoring the footage. Thus, requiring a complete video transmission infrastructure which can be costly.

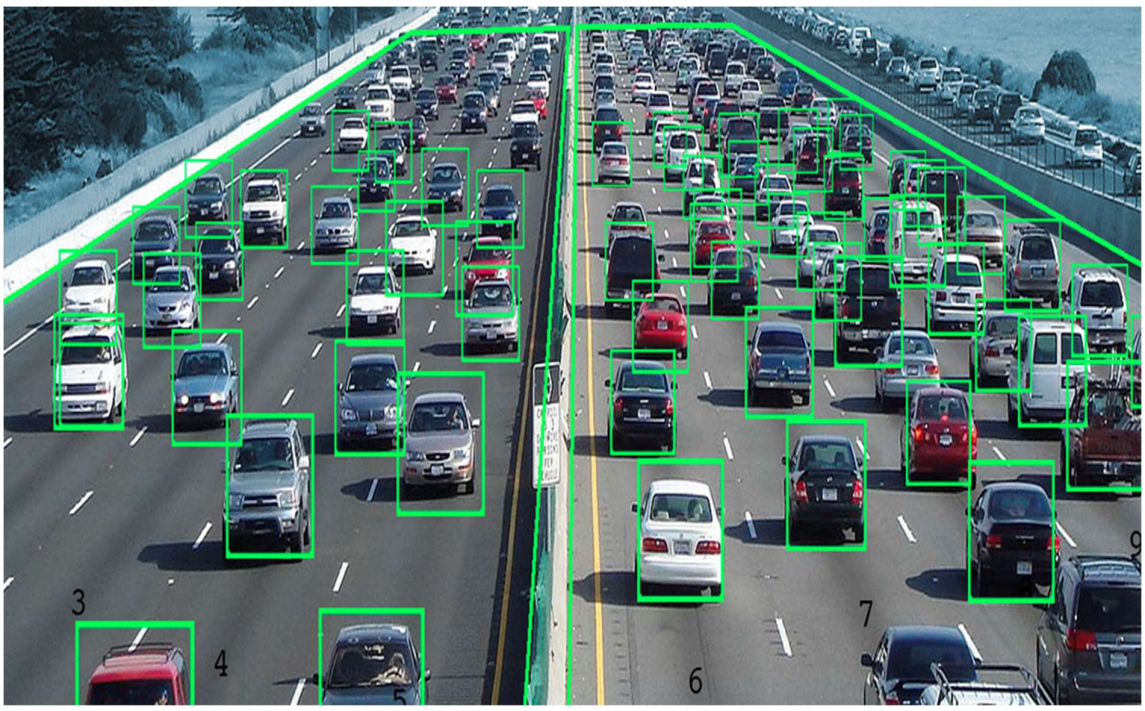
This project aims to solve both the problems. By using Deep Learning to detect the license plates and performing Optical Character Recognition (OCR) for finally detecting the numbers, there is no need for humans to manually note the numbers. Secondly, by performing the inference of the plates on the cameras along with OCR and only transmitting the license plate numbers in text format reduces the costs of transmission network to a fraction of what is needed for video transmission.

Introduction

Traffic surveillance is becoming an increasingly complex task due to the explosive growth in the number of vehicles on roads. It is no longer possible for humans alone to effectively manage such systems.

Meanwhile, progresses in deep learning are advancing at a tremendous pace and it is becoming easier to replace most of the seemingly difficult tasks for humans with deep learning solution.

Deep learning models can classify multiply cars as well as other things in the field of view simultaneously and even at runtime if given appropriate computing power. One such example is given below, about how a camera security system can detect cars in real-time.



We intend to use the same concept but tailor our application to specifically license plate detection and then run it on a mobile

setup that will allow us to perform the detections on the cameras without transmitting the video to a central location and finally attempt to achieve a near real-time performance.

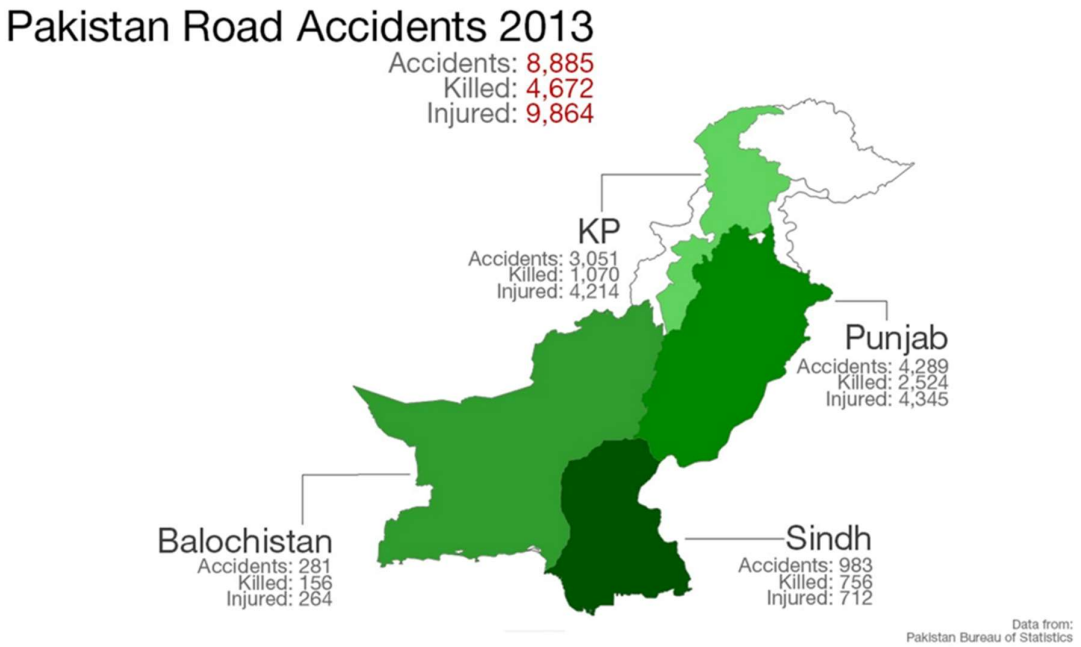
Potential Impact

The Punjab Safe Cities Authority is currently in the process of upgrading the traffic surveillance system in Punjab and is installing CCTV cameras to monitor traffic. The PPIC3 is pictured below.

It employs over 300 operators who spend a large amount of time going over surveillance footage to identify any violations.

Our system has the potential to directly benefit them by performing this analysis autonomously, freeing up the staff to do more productive work.

The following diagram shows the status of road accidents all over Pakistan in 2013. The numbers have been constantly growing before this and expected to grow immensely in the reports from later years as well.



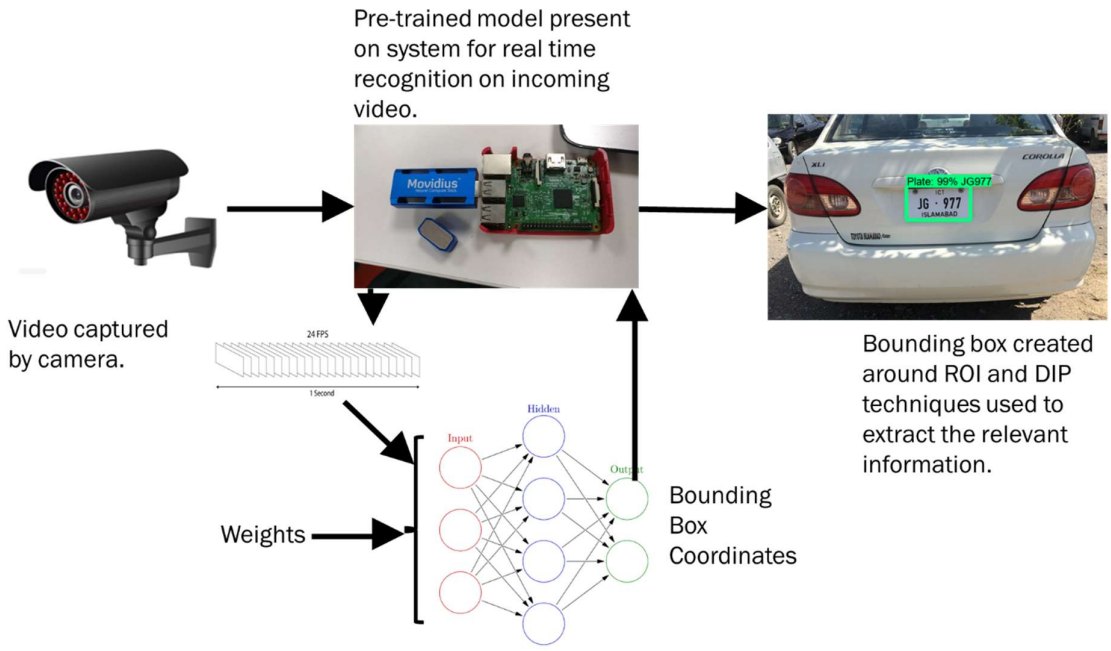
This problem can be solved by better regulation, however with the number of cars on roads constantly on the rise, automation is the way to go. Using an autonomous solution has the potential to save a lot of live.

Objectives

- Gather Data and Label it for Training the Deep Learning Model.
- Train a model and achieve a good accuracy.
- Perform Optical Character Recognition to detect numbers.
- Run the system on a mobile setup, on a Raspberry Pi in our case.

System Diagram

The high level working of the system is shown below.



The process starts by a video camera that captures video feed. This arrives at the Raspberry Pi in the form of individual frames and the Raspberry Pi can select the current incoming frame for processing. During this processing the frame is passed through a deep learning network for object detection. The weights are provided to the network that are trained beforehand on a GPU separately on our dataset of pictures of license plates of Pakistan.

This object detection process is accelerated by the Intel Movidius Neural Compute Stick connected to the Raspberry Pi. Open Vino toolkit is installed on the Raspberry Pi that allows the supported weights to be used with the Neural Compute Stick. Once the Object detection is completed and the coordinates of detected license plates are received, the relevant image is cropped and passed to an OCR program for character detection. Finally, the result is displayed by overlaying the information on image.

Dataset

The dataset was manually collected by capturing images of cars from different angles to get a diverse set of images for training.

Different types of vehicles were included as well

Images containing different lighting and visibility conditions were also included in the dataset such as at night and viewing through glass.

The dataset in total consisted of 413 images. From these 82 images were randomly selected as testing data and the rest 331 images were used as training data. Hence resulting in an 80-20 training-testing split.

Each image was labelled manually, resulting in 413 xml files containing the labels and their coordinates.

Labelling the Dataset

The process of labelling involved using a tool called "labelimg", and opening each image in it, selecting the region of license plate by drawing a bounding box and saving it by the label "Plate".
Examples are shown below:

After each image was labelled and saved, pressing next button will bring the next image in the directory and it also must go through the same process till all the images are labelled. For each labelled image, an xml file is generated with same name as the image, containing the coordinates and names of the labels. The screenshot of the folder shows images and their respective label files generated.

14.xml	4/17/2018 4:09 AM	XML Document	1 KB
15.jpg	4/14/2018 5:28 PM	JPG File	162 KB
15.xml	4/17/2018 4:09 AM	XML Document	1 KB
16.jpg	4/14/2018 5:28 PM	JPG File	151 KB
16.xml	4/17/2018 4:09 AM	XML Document	1 KB
17.jpg	4/14/2018 5:28 PM	JPG File	166 KB
17.xml	4/17/2018 4:09 AM	XML Document	1 KB
18.jpg	4/14/2018 5:28 PM	JPG File	179 KB
18.xml	4/17/2018 4:09 AM	XML Document	1 KB
19.jpg	4/14/2018 5:29 PM	JPG File	159 KB
19.xml	4/17/2018 4:09 AM	XML Document	1 KB
20.jpg	4/14/2018 5:29 PM	JPG File	146 KB
20.xml	4/17/2018 4:09 AM	XML Document	1 KB
21.jpg	4/14/2018 5:29 PM	JPG File	166 KB
21.xml	4/17/2018 4:09 AM	XML Document	1 KB
22.jpg	4/14/2018 5:29 PM	JPG File	179 KB
22.xml	4/17/2018 4:09 AM	XML Document	1 KB
23.jpg	4/14/2018 5:29 PM	JPG File	187 KB
23.xml	4/17/2018 4:09 AM	XML Document	1 KB
24.jpg	4/14/2018 5:29 PM	JPG File	161 KB
24.xml	4/17/2018 4:09 AM	XML Document	1 KB
29.jpg	4/14/2018 3:47 AM	JPG File	211 KB
29.xml	4/17/2018 4:09 AM	XML Document	1 KB

The contents of the label xml file are as follows:

```
<?xml version="1.0"?>
- <annotation>
    <folder>imaag</folder>
    <filename>umZ7MgA.jpg</filename>
    <path>C:\dataset\umZ7MgA.jpg</path>
  - <source>
      <database>Unknown</database>
    </source>
  - <size>
      <width>771</width>
      <height>446</height>
      <depth>3</depth>
    </size>
    <segmented>0</segmented>
  - <object>
      <name>Plate</name>
      <pose>Unspecified</pose>
      <truncated>0</truncated>
      <difficult>0</difficult>
    - <bndbox>
        <xmin>322</xmin>
        <ymin>267</ymin>
        <xmax>459</xmax>
        <ymax>334</ymax>
      </bndbox>
    </object>
</annotation>
```

The coordinates are specified as exact pixel locations. This however means that if image had to be resized later, the label file would no longer remain useful and the image will have to be relabeled. This was quite a frequent problem since different deep learning

networks had different memory requirements and to fit larger networks in RAM, it was required to shrink the dimensions of the images which meant relabeling them again after dimension reduction.

Training

After preparation of the dataset, next step is to train a deep learning network using the labelled dataset. Initially a YOLO V2 network, built using Keras was trained and it gave a good, above 90% accuracy on testing data. However, it later became apparent that our objectives will play a major role in deciding the network we want to go forward with.

Selecting the Right Network

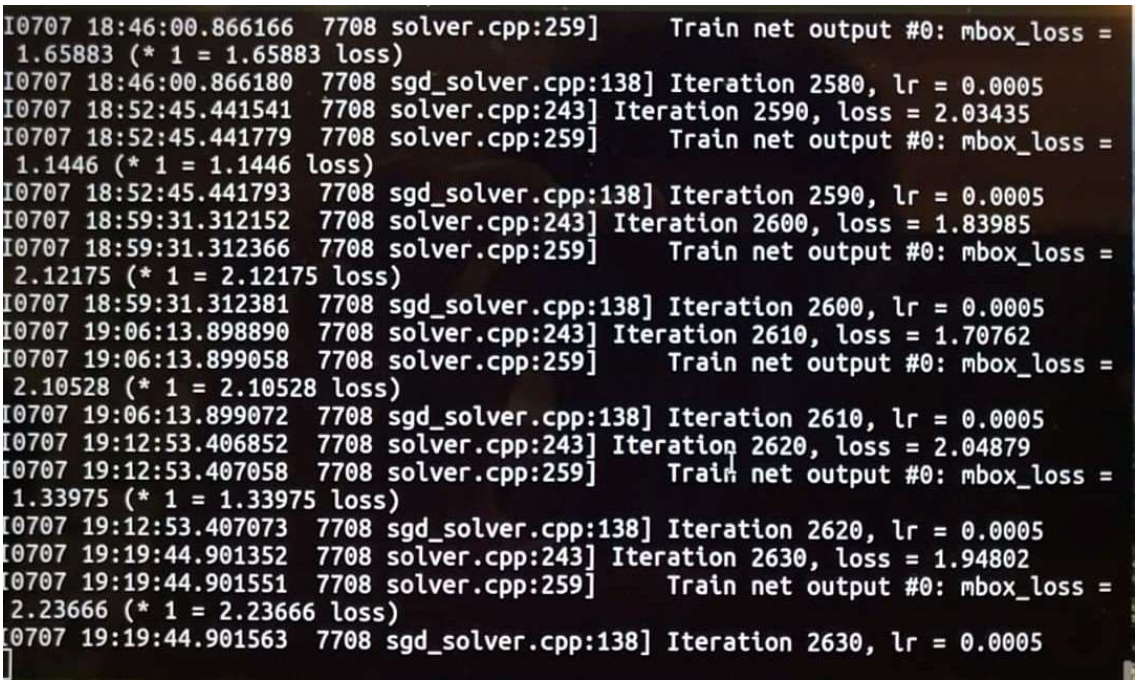
Our end goal was to run the inference on Raspberry Pi and accelerate it using Intel Movidius Neural Compute Stick (NCS). However, the NCS does not support all the deep learning frameworks and especially not Keras, the one used by the network we trained earlier.

The two main frameworks that NCS did support were Caffe and TensorFlow. So, we set out to train a Resnet Inception V2 model on TensorFlow, and it achieved an even better accuracy of 99% in license plate detection. However, Resnet model is quite computationally expensive and not suitable for our application. Moreover, TensorFlow models required extensive reworking to reduce their input dimension to 1, which is what the NCS required for it to be compatible with them.

Thus, we finally decided to select a MobileNet SSD network (details in next section) since it was the least computationally expensive

network available and trained it on the Caffe framework, the other framework supported by NCS. This solved both the problems experienced before.

The Caffe model took several iterations to train, reducing the loss with each iteration as shown below.



Using the Trained Weights

After the training process is completed, the weights are generated containing a ".caffemodel" extension file and a ".prototxt" extension file.

Both are needed to do the inference using the NCS. There are two main ways to achieve this. Initially when the NCS was launched, intel also released a Neural Compute Stick SDK (NCSDK) that would allow conversion of supported network weights into an inference graph file with ".graph" extension. This would then run directly on the NCS.

However, this was not a robust process and the conversion could fail depending on a lot of reasons. Fortunately, intel released OpenVino Toolkit later which allows the weights to be used directly without needing any conversions. This made the whole process much easier and robust than before and it is what we ended up using to run the trained network on Raspberry Pi, accelerated by NCS.

After successful installation of OpenVino Toolkit, the following line can be used in Python code to take as inputs the two weights files generated after training.

```
net= cv2.dnn.readNetFromCaffe(args["prototxt"], args["model"])
```

Then we can select the OpenVino version of Open CV as:

```
net.setPreferableTarget(cv2.dnn.DNN_TARGET_MYRIAD)
```

Finally, the input frame can be passed for detections using the lines shown below:

```
blob = cv2.dnn.blobFromImage(frame, 0.007843, (300, 300), 127.5)
net.setInput(blob)
detections = net.forward()
```

MobileNet SSD Setup

To perform classifications, we need a network. There are two types of deep neural networks that we are using, a base network and a detection network. A base network provides high level features for classifications. To perform classifications using these high-level features we need a fully connected layer at the end. However, an even better approach is to replace the fully connected layer with a detection network.

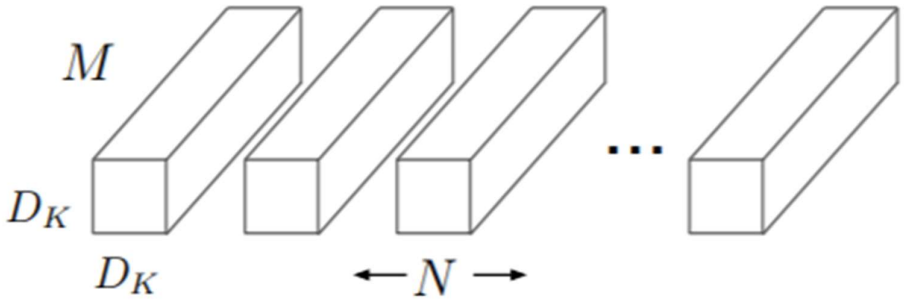
So, our network consists of MobileNet as the base network and Single Shot MultiBox Detector (SSD) as the detection network. The main reason for selecting this combination was to achieve the least latency in the network so that we can get the best performance while using limited computation power and get as close to real-time performance as possible.

MobileNet Base Network

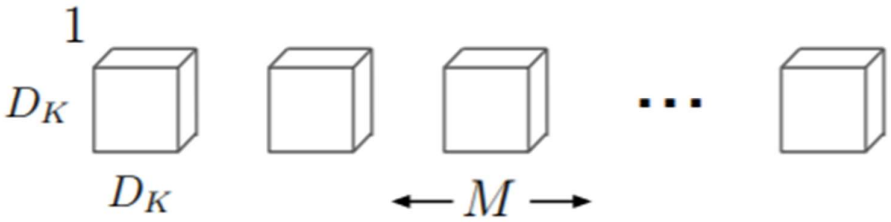
With deep learning networks, the main advances are usually related to higher accuracy, but this comes at a compromise of efficiency with respect to size and speed.

To solve this, the researchers at Google have come up with small and low latency models that can work fast on mobile devices and in limited computational power environments. This exactly matches our use case and hence we have selected this base network, MobileNet.

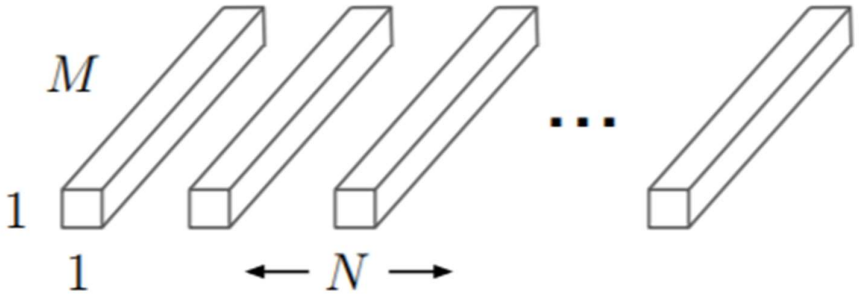
The speed comes from depth wise separable convolutions which is a form of factorized convolutions which factorize a standard convolution into a depth wise convolution and a 1×1 convolution called a pointwise convolution as shown in the figure below:



(a) Standard Convolution Filters



(b) Depthwise Convolutional Filters



(c) 1×1 Convolutional Filters called Pointwise Convolution in the context of Depthwise Separable Convolution

Figure 2. The standard convolutional filters in (a) are replaced by two layers: depthwise convolution in (b) and pointwise convolution in (c) to build a depthwise separable filter.

The MobileNet network has 28 layers if we count depth wise and pointwise convolutions as separate layers. 2 more layers in the end,

average pooling and fully connected are used if this network is used by itself. However, we would be replacing them with SSD network.

The complete 30 layers are shown below, our setup includes all except the last 2 layers:

Table 1. MobileNet Body Architecture

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

Single Shot MultiBox Detector Network:

Like the situation with MobileNet, before the introduction of SSD, most of the object detection systems used the approach of hypothesize bounding boxes, resample pixels or features for each box and apply a high-quality classifier. While this was accurate, it was too computationally expensive and not good for real-time applications.

The SSD network solves this by eliminating bounding box proposals and subsequent pixel or feature resampling stage.

Comparison between the SSD and YOLO models in shown below and why the SSD is faster.

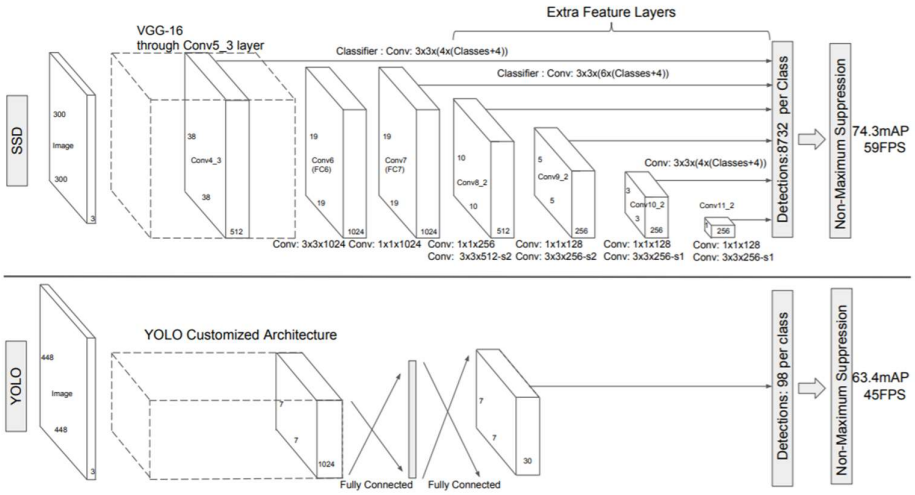


Fig. 2: A comparison between two single shot detection models: SSD and YOLO [5].

SSD network allows prediction of the offsets to default boxes of different scales and aspect ratios and their associated confidences. It adds several feature layers to the end of a base network.

Tesseract Optical Character Recognition (OCR)

The MobileNet SSD network detects the plates and the detected region is cropped from the image. It now has to pass through an Optical Character Recognition system to get the digits on the license plates.

Tesseract, while not accurate is one of the best OCR solutions available and being opensource makes it free for use. Low accuracy in OCR is a problem with almost all the solutions available and we did not have any better alternative to choose instead.

The first step is to do a connected component analysis which allows the outlines of the components to be stored. This has a big advantage that by inspecting the nesting of outlines and the number of child and grandchild outlines it is simple to detect inverse text as well along with black on white text. Du to this Tesseract was the first OCR engine to handle white-on-black text.

The outlines are gathered into blobs by nesting. Blobs are analyzed for fixed pitch or proportional text. Then it passes through a two-step process where in first part an attempt is made to recognize words in turn. The words that are correct are passed as training data into an adaptive classifier. This gives it a chance to recognize the text later more accurately.

Since this would cause better detection at the bottom of the page as opposed to the top, a second pass is initiated so that any words that were not detected before can be detected now.

Finally, the fuzzy spaces are resolved, and alternative hypotheses are checked to locate small-cap text.

Tesseract also uses a “Line Finding” algorithm and it is will recognize a skewed page without fixing it which would have caused a loss in image quality. The main part of this process is blob filtering along with line construction.

Hardware Setup

The hardware setup is used as shown on the previous page. Raspberry Pi 3 is used (any Raspberry Pi device would work) as a base device to manage all the things. It will execute the Python code that will use the weights of MobileNet SSD network trained on our dataset using Caffe.
Peripherals like mouse, keyboard and webcam for video input are also connected to the Raspberry Pi and it will manage the I/O for these devices. The Intel Movidius Neural Compute Stick, shown in the diagram as an enlarged blue USB stick is also connected as a peripheral to the Raspberry Pi and it will help to accelerate the computation specific to object detection which in our use case is the license plate detection. After the plate is detected, the OCR is performed using Tesseract which is not supported by Movidius and hence using it drops the frame rate by a lot as it will be running on the Raspberry Pi CPU.

The display out if needed is also given by connecting the HDMI cable to the Raspberry Pi. Finally, the last thing needed is a micro USB cable connected to the Raspberry Pi to power the whole setup.

The whole package is small and light enough to be used anywhere on the go or alongside the highway security cameras, to do inference on the edge.

Results

The plate detection is a relatively easier task than optical character recognition (OCR) and it shows clearly in the results.

We achieved above 95% accuracy in license plate detections but after successfully detecting license plates, the accuracy for correct optical character recognition as well is only about 50-60%.

Following pictures show the results when both work correctly:


Plate: 99% LF401
ICT
LF · 401
ISLAMABAD


Plate: 99% ABU081
ABU·081
ICT-ISLAMABAD

These pictures show that when both the plate detection and OCR work then the results are exactly what we want. However, often there are times when the camera is at an angle, causing skewness in the plate image. This is when plate detection still works, however the optical character recognition fails.

When using the system in real-time on the Raspberry Pi, accelerated by the Neural Compute Stick V1, we get performance of 4 to 5 frames per second while only in plate detection mode. If OCR is performed as well then, the framerate dips to 0.6 frames per second, not maintaining real-time performance anymore. This is because the Tesseract OCR cannot be accelerated on the Neural Compute Stick and will use the Raspberry Pi CPU instead.

The performance of the system can be increased by using the

Neural Compute Stick V2 which has the potential to take the plate detection performance to about 6-8 frames per second. Moreover, the performance of OCR can be increased by using a more powerful board instead of Raspberry Pi.

References

1. SSD: Single Shot MultiBox Detector [Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, Alexander C. Berg]
2. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications [Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, Hartwig Adam]
3. An Overview of the Tesseract OCR Engine [Ray Smith]
4. Intel OpenVino Toolkit
5. Caffe Deep Learning Framework
6. Neural Compute Application Zoo (NC App Zoo)

Turnitin Originality Report

Processed on: 08-Jul-2019 00:59 PKT
ID: 1149785657
Word Count: 2866
Submitted: 3

fyp1 By Syed Mohammad Ali

Similarity Index	Similarity by Source	
11%	Internet Sources:	7%
	Publications:	7%
	Student Papers:	10%

2% match (Internet from 21-Mar-2019)
https://storage.googleapis.com/pub-tools-public-publication-data/pdf/44872.pdf

1% match (student papers from 14-May-2018)
Submitted to University of Surrey on 2018-05-14

1% match (publications)
R. Smith. "An Overview of the Tesseract OCR Engine", Ninth International Conference on Document Analysis and Recognition (ICDAR 2007) Vol 2, 2007

1% match (student papers from 29-Jun-2019)
Submitted to University of South Australia on 2019-06-29

1% match (Internet from 19-Oct-2018)
http://repository.sustech.edu/bitstream/handle/123456789/15281/chapter%202.pdf?seq=

1% match (student papers from 11-Feb-2016)
Submitted to University of Northampton on 2016-02-11

1% match (student papers from 09-Dec-2018)
Submitted to Swinburne University of Technology on 2018-12-09

< 1% match (publications)
Xinggan Peng, Long Wen, Diqiu Bai, Bei Peng. "Chapter 22 Reformative Vehicle License Plate Recognition Algorithm Based on Deep Learning", Springer Science and Business Media LLC, 2019

< 1% match (student papers from 04-Feb-2014)
Submitted to MCAST on 2014-02-04

< 1% match (Internet from 05-May-2019)
https://www.chaj.com/post/176985543051/opencv-people-counter

< 1% match (student papers from 10-May-2018)
Submitted to Nottingham Trent University on 2018-05-10

< 1% match (publications)
"Computer Vision", Springer Science and Business Media LLC, 2017

< 1% match (student papers from 02-Jun-2019)

Submitted to Loughborough University on 2019-06-02

< 1% match (student papers from 28-Apr-2018)
Submitted to Clemson University on 2018-04-28

< 1% match (publications)
Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, Alexander C. Berg. "Chapter 2 SSD: Single Shot MultiBox Detector", Springer Nature, 2016

< 1% match (publications)
"Computer Vision – ECCV 2016", Springer Science and Business Media LLC, 2016

< 1% match (publications)
Yeong-Kang Lai, Chu-Ying Ho, Yu-Hau Huang, Chuan-Wei Huang, Yi-Xian Kuo, Yu-Chieh Chung. "Intelligent Vehicle Collision-Avoidance System with Deep Learning", 2018 IEEE Asia Pacific Conference on Circuits and Systems (APCCAS), 2018

< 1% match (student papers from 17-Jul-2013)
Submitted to Higher Education Commission Pakistan on 2013-07-17

DE-37 (DC&SE) NUST COLLEGE OF ELECTRICAL AND MECHANICAL ENGINEERING Vehicle Tagging Using Deep Learning PROJECT REPORT DE-37 (DC&SE) Submitted by NS AMMAR AMJAD NS TAIMUR IBRAHIM PC AZIZ UR REHMAN NS SYED MOHAMMAD ALI BACHELORS IN COMPUTER ENGINEERING YEAR 2019 PROJECT SUPERVISORS DR. USMAN AKRAM YEAR 2019 DR. SAJID GUL KHAWAJA COLLEGE OF ELECTRICAL AND MECHANICAL ENGINEERING Declaration Copyright Statement Acknowledgments LEFT BLANK Table of Contents Declaration

..... 29

Abstract Traditional road monitoring or security systems usually employ people to manually monitor the camera feeds and note the license plate numbers of the vehicles of interest. These types of models have two main disadvantages. Firstly, they require humans to be working constantly to watch the footage and note the numbers and secondly the video needs to be transmitted from the street cameras to the central offices where the humans would usually be monitoring the footage. Thus, requiring a complete video transmission infrastructure which can be costly. This project aims to solve both the problems. By using Deep Learning to detect the license plates and performing Optical Character Recognition (OCR) for finally detecting the numbers, there is no need for humans to manually note the numbers. Secondly, by performing the inference of the plates on the cameras along with OCR and only transmitting the license plate numbers in text format reduces the costs of transmission network to a fraction of what is needed for video transmission. Introduction Traffic surveillance is becoming an increasingly complex task due to the explosive growth in the number of vehicles on roads. It is no longer possible for humans alone to effectively manage such systems. Meanwhile, progresses in deep learning are advancing at a tremendous pace and it is becoming easier to replace most of the seemingly difficult tasks for humans with deep learning solution. Deep learning models can classify multiply cars as well as other things in the field of view simultaneously and even at runtime if given appropriate computing power. One such example is given below, about how a camera security system can detect cars in real- time. We intend to use the same concept but tailor our application to specifically license plate detection and then run it on a mobile setup that will allow us to perform the detections on the cameras without transmitting the video to a central location and finally attempt to achieve a near real-time performance. Potential Impact The Punjab Safe Cities Authority is currently in the process of upgrading the traffic surveillance system in Punjab and is installing CCTV cameras to monitor traffic. The PPIC3 is pictured below. It employs over 300 operators who spend a large amount of time going over surveillance footage to identify any violations. Our system has the potential to directly benefit them by performing this analysis autonomously, freeing up the staff to do more productive work. The following diagram shows the status of road accidents all over Pakistan in 2013. The numbers have been constantly growing before this and expected to grow immensely in the reports from later years as well. This problem can be solved by better regulation, however with the number of cars on roads constantly on the rise, automation is the way to go. Using an autonomous solution has the potential to save a lot of live. Objectives ? Gather Data and Label it for Training the Deep Learning Model. ? Train a model and achieve a good accuracy. ? Perform Optical Character Recognition to detect numbers. ? Run the system on a mobile setup, on a Raspberry Pi in our case. System Diagram The high level working of the system is shown below. The process starts by a video camera that captures video feed. This arrives at the Raspberry Pi in the form of individual frames and the Raspberry Pi can select the current incoming frame for processing. During this processing the frame is passed through a deep learning network for object detection. The weights are provided to the network that are trained beforehand on a GPU separately on our dataset of pictures of license plates of Pakistan. This object detection process is accelerated by the Intel Movidius Neural Compute Stick connected to the Raspberry Pi. Open Vino toolkit is installed on the Raspberry Pi that allows the supported weights to be used with the Neural Compute Stick. Once the Object detection is completed and the coordinates of detected license plates are received, the relevant image is cropped and passed to an OCR program for character detection. Finally, the result is displayed by overlaying

the information on image. Dataset The dataset was manually collected by capturing images of cars from different angles to get a diverse set of images for training. Different types of vehicles were included as well Images containing different lighting and visibility conditions were also included in the dataset such as at night and viewing through glass. The dataset in total consisted of 413 images. From these 82 images were randomly selected as testing data and the rest 331 images were used as training data. Hence resulting in an 80-20 training- testing split. Each image was labelled manually, resulting in 413 xml files containing the labels and their coordinates. Labelling the Dataset The process of labelling involved using a tool called "labelimg", and opening each image in it, selecting the region of license plate by drawing a bounding box and saving it by the label "Plate". Examples are shown below: After each image was labelled and saved, pressing next button will bring the next image in the directory and it also must go through the same process till all the images are labelled. For each labelled image, an xml file is generated with same name as the image, containing the coordinates and names of the labels. The screenshot of the folder shows images and their respective label files generated. The contents of the label xml file are as follows: The coordinates are specified as exact pixel locations. This however means that if image had to be resized later, the label file would no longer remain useful and the image will have to be relabeled. This was quite a frequent problem since different deep learning networks had different memory requirements and to fit larger networks in RAM, it was required to shrink the dimensions of the images which meant relabeling them again after dimension reduction. Training After preparation of the dataset, next step is to train a deep learning network using the labelled dataset. Initially a YOLO V2 network, built using Keras was trained and it gave a good, above 90% accuracy on testing data. However, it later became apparent that our objectives will play a major role in deciding the network we want to go forward with. Selecting the Right Network Our end goal was to run the inference on Raspberry Pi and accelerate it using Intel Movidius Neural Compute Stick (NCS). However, the NCS does not support all the deep learning frameworks and especially not Keras, the one used by the network we trained earlier. The two main frameworks that NCS did support were Caffe and TensorFlow. So, we set out to train a Resnet Inception V2 model on TensorFlow, and it achieved an even better accuracy of 99% in license plate detection. However, Resnet model is quite computationally expensive and not suitable for our application. Moreover, TensorFlow models required extensive reworking to reduce their input dimension to 1, which is what the NCS required for it to be compatible with them. Thus, we finally decided to select a MobileNet SSD network (details in next section) since it was the least computationally expensive network available and trained it on the Caffe framework, the other framework supported by NCS. This solved both the problems experienced before. The Caffe model took several iterations to train, reducing the loss with each iteration as shown below. Using the Trained Weights After the training process is completed, the weights are generated containing a ".caffemodel" extension file and a ".prototxt" extension file. Both are needed to do the inference using the NCS. There are two main ways to achieve this. Initially when the NCS was launched, intel also released a Neural Compute Stick SDK (NCSDK) that would allow conversion of supported network weights into an inference graph file with ".graph" extension. This would then run directly on the NCS. However, this was not a robust process and the conversion could fail depending on a lot of reasons. Fortunately, intel released OpenVino Toolkit later which allows the weights to be used directly without needing any conversions. This made the whole process much easier and robust than before and it is what we ended up using to run the trained network on Raspberry Pi, accelerated by NCS. After successful installation of

OpenVino Toolkit, the following line can be used in Python code to take as inputs the two weights files generated after training. net= cv2.dnn.readNetFromCaffe(args[" prototxt"], args[" model"]) Then we can select the OpenVino version of Open CV as: net.setPreferableTarget(cv2.dnn.DNN_TARGET_MYRIAD) Finally, the input frame can be passed for detections using the lines shown below: blob = cv2.dnn.blobFromImage(frame, 0.007843, (300, 300), 127.5) net.setInput(blob) detections = net.forward() MobileNet SSD Setup To perform classifications, we need a network. There are two types of deep neural networks that we are using, a base network and a detection network. A base network provides high level features for classifications. To perform classifications using these high-level features we need a fully connected layer at the end. However, an even better approach is to replace the fully connected layer with a detection network. So, our network consists of MobileNet as the base network and Single Shot MultiBox Detector (SSD) as the detection network. The main reason for selecting this combination was to achieve the least latency in the network so that we can get the best performance while using limited computation power and get as close to real-time performance as possible. MobileNet Base Network With deep learning networks, the main advances are usually related to higher accuracy, but this comes at a compromise of efficiency with respect to size and speed. To solve this, the researchers at Google have come up with small and low latency models that can work fast on mobile devices and in limited computational power environments. This exactly matches our use case and hence we have selected this base network, MobileNet. The speed comes from depth wise separable convolutions which is a form of factorized convolutions which factorize a standard convolution into a depth wise convolution and a 1×1 convolution called a pointwise convolution as shown in the figure below: The MobileNet network has 28 layers if we count depth wise and pointwise convolutions as separate layers. 2 more layers in the end, average pooling and fully connected are used if this network is used by itself. However, we would be replacing them with SSD network. The complete 30 layers are shown below, our setup includes all except the last 2 layers: Single Shot MultiBox Detector Network: Like the situation with MobileNet, before the introduction of SSD, most of the object detection systems used the approach of hypothesize bounding boxes, resample pixels or features for each box and apply a high-quality classifier. While this was accurate, it was too computationally expensive and not good for real-time applications. The SSD network solves this by eliminating bounding box proposals and subsequent pixel or feature resampling stage. Comparison between the SSD and YOLO models in shown below and why the SSD is faster. SSD network allows prediction of the offsets to default boxes of different scales and aspect ratios and their associated confidences. It adds several feature layers to the end of a base network. Tesseract Optical Character Recognition (OCR) The MobileNet SSD network detects the plates and the detected region is cropped from the image. It now has to pass through an Optical Character Recognition system to get the digits on the license plates. Tesseract, while not accurate is one of the best OCR solutions available and being opensource makes it free for use. Low accuracy in OCR is a problem with almost all the solutions available and we did not have any better alternative to choose instead. The first step is to do a connected component analysis which allows the outlines of the components to be stored. This has a big advantage that by inspecting the nesting of outlines and the number of child and grandchild outlines it is simple to detect inverse text as well along with black on white text. Du to this Tesseract was the first OCR engine to handle white-on-black text. The outlines are gathered into blobs by nesting. Blobs are analyzed for fixed pitch or proportional text. Then it passes through a two- step process where in

first part an attempt is made to recognize words in turn. The words that are correct are passed as training data into an adaptive classifier. This gives it a chance to recognize the text later more accurately. Since this would cause better detection at the bottom of the page as opposed to the top, a second pass is initiated so that any words that were not detected before can be detected now. Finally, the fuzzy spaces are resolved, and alternative hypotheses are checked to locate small-cap text. Tesseract also uses a "Line Finding" algorithm and it is will recognize a skewed page without fixing it which would have caused a loss in image quality. The main part of this process is blob filtering along with line construction. Hardware Setup The hardware setup is used as shown on the previous page. Raspberry Pi 3 is used (any Raspberry Pi device would work) as a base device to manage all the things. It will execute the Python code that will use the weights of MobileNet SSD network trained on our dataset using Caffe. Peripherals like mouse, keyboard and webcam for video input are also connected to the Raspberry Pi and it will manage the I/O for these devices. The Intel Movidius Neural Compute Stick, shown in the diagram as an enlarged blue USB stick is also connected as a peripheral to the Raspberry Pi and it will help to accelerate the computation specific to object detection which in our use case is the license plate detection. After the plate is detected, the OCR is performed using Tesseract which is not supported by Movidius and hence using it drops the frame rate by a lot as it will be running on the Raspberry Pi CPU. The display out if needed is also given by connecting the HDMI cable to the Raspberry Pi. Finally, the last thing needed is a micro USB cable connected to the Raspberry Pi to power the whole setup. The whole package is small and light enough to be used anywhere on the go or alongside the highway security cameras, to do inference on the edge. Results The plate detection is a relatively easier task than optical character recognition (OCR) and it shows clearly in the results. We achieved above 95% accuracy in license plate detections but after successfully detecting license plates, the accuracy for correct optical character recognition as well is only about 50-60%. Following pictures show the results when both work correctly: These pictures show that when both the plate detection and OCR work then the results are exactly what we want. However, often there are times when the camera is at an angle, causing skewness in the plate image. This is when plate detection still works, however the optical character recognition fails. When using the system in real-time on the Raspberry Pi, accelerated by the Neural Compute Stick V1, we get performance of 4 to 5 frames per second while only in plate detection mode. If OCR is performed as well then, the framerate dips to 0.6 frames per second, not maintaining real-time performance anymore. This is because the Tesseract OCR cannot be accelerated on the Neural Compute Stick and will use the Raspberry Pi CPU instead. The performance of the system can be increased by using the Neural Compute Stick V2 which has the potential to take the plate detection performance to about 6-8 frames per second. Moreover, the performance of OCR can be increased by using a more powerful board instead of Raspberry Pi. References 1. SSD: Single Shot MultiBox Detector [Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, Alexander C. Berg] 2. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications [Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, Hartwig Adam] 3. An Overview of the Tesseract OCR Engine [Ray Smith] 4. Intel OpenVino Toolkit 5. Caffe Deep Learning Framework 6. Neural Compute Application Zoo (NC App Zoo)