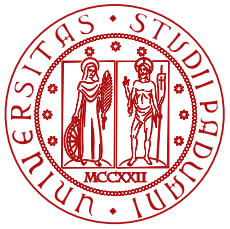

Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Master Thesis in ICT For Internet And Multimedia Engineering

Development of an object detection and mask generation software for dynamic beam projection in automotive pixel lighting applications

Master Candidate

Gulay Erdin

Student ID 2008483

Supervisor

Prof. Federica Battisti

University of Padova

Co-supervisor

Riccardo Zuin

Infineon Techologies Srl

Academic Year
2021/2022

To my dad,
you are always in my heart.

Abstract

Nowadays there are many contributions to the automotive industry and the field is developing fast. This work can be used for some real-time autonomous driving applications. The goal was to add advanced functionality to a standard light source in collaboration with electronic systems. Including advanced features may result in safer and more pleasant driving. The application fields of the work could include glare-free light sources, orientation and lane lights, marking lights, and symbol projection. On a real-time source, object detection and classification with a confidence score is implemented. The best model is obtained by intending to train the model with varying parameters. The most accurate result which is mAP value 0.572 was obtained by distributing the training dataset with learning rate 0.2 and setting the epochs to 300. Moreover, a basic implementation of a glare-free light source was done to avoid the drivers from being blinded by the illumination of the beams. The car and rectangle shape masks were generated as image files and sent as CSV files to the pixel light source device. As a result, the rectangle shaped mask functions more precisely then car shaped.

Contents

List of Figures

List of Tables

List of Code Snippets

List of Acronyms

CSV Comma Separated Values

YOLO You Only Look Once

FP False Positive

DC Duty Cyle

mAP Mean Average Precision

pt PyTorch

CNN Convolutional Neural Networks

DNN Deep Neural Network

ADB Adaptive Light Beam

UART Universal Asynchronous Receiver Transmitter

USB Universal Serial Bus

LAN Local Area Network

NIC Network Interface Cards

CSMA/CD Carrier-sense multiple access with collision detection

LLC Logical Link Control

TF Tensorflow

1 Introduction

This project aimed to make a contribution to the automotive industry. It is beneficial for some real-time autonomous driving applications. In collaboration with electronic systems, the goal was to add advanced functionality to standard light source. Adding advanced features may result in safer driving and a better driving experience.

The state-of-art object detection and classification model YOLOv5 is used. In 2016 Joseph Redmon published the first model YOLOv1. Over time development of the model continued and in 2019 Glenn Jocher published YOLOv5. It is targeted that the electronic systems in the vehicle can see and recognize as the driver sees them and thus contribute to the driving safety and experience. For the object mask computer vision library of python programming language is used. By generating anti-glare images and illuminating the street from the light source, driving safety is increased and the system can be used to improve self-driving vehicles.

Application fields of the project might be: glare-free light source, orientation and lane lights, marking lights and symbol projection.

2

Adaptive Driving Beam

2.1 Introduction

A headlight system called ADB (adaptive driving beam) uses automatic beam forming and a computer to focus lights in various directions. It can illuminate the road directly in front and dim the lights that shine outward and might otherwise blind oncoming drivers.

Adaptive light sources provide the driving environment with a better level of illumination, improving one's capability to see at night or in low light conditions when compared to standard light sources. Adaptive light sources can direct the emitted light in a controlled way compared to regular front beams which have a fixed light cone. These light sources use electronic sensors that can collect data and communicate with other electronic systems in accordance with the guidance of the car.

On each light source, individual LEDs are automatically enabled and disabled by the adaptive high beam for high precision control over light distribution. The system makes use of a camera within the car to accurately detect approaching or preceding vehicles while smoothly illuminating the road ahead. With clear vision and calmer driving conditions, this enables drivers to experience their most brightly illuminated nocturnal drive without having to constantly toggle between high and low beam.

2.1.1 Application Fields of ADB in the Automotive Industry

ADB arises when traditional low and high beam functions are extended with pixel lighting. Pixel lighting is an automotive high definition LED front lighting solution. Pixel lighting applications enhance the driving experience and safety by enabling new use cases such as glare-free light beam, orientation and lane lights and marking lights and symbol projection.

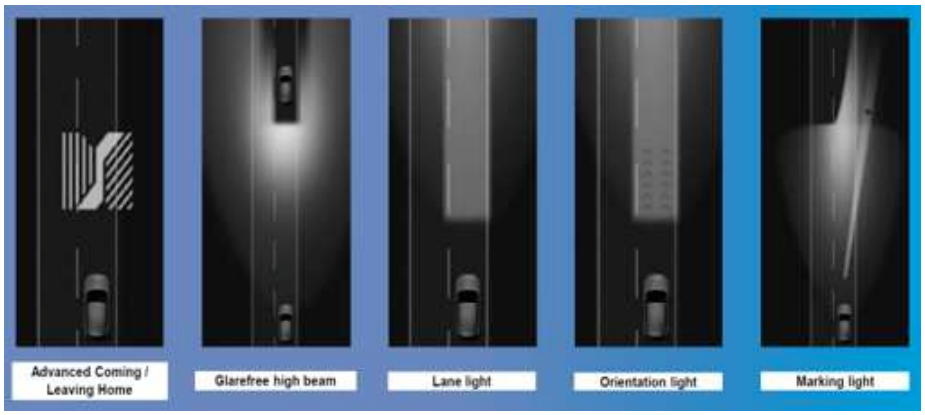

Figure 2.1: Source Huhn, DVN

ADB allows for safer driving. One of the technological advancements in automotive engineering that will increase driving safety is automatic adaptive headlights. These technologies provide autonomous road lighting without the need for driver involvement. The adaptive headlights' ability to avoid accidents is essential to its usefulness.

For example, a car is approaching a curve while driving at night on a two-lane road with no road lights. Even at a safe speed, visibility is compromised. These adaptive headlights make for improved view over the road ahead by turning on and off automatically. While driving at night only constitutes to 25% of overall driving, 50% of traffic deaths happen at night. It doesn't matter whether the road is familiar or not, driving at night is always more dangerous. More than 42,000 people were killed in car crashes in 2020[1]. Adaptive driving lights improve nighttime driving and increase safety.

[1]According to National Safety Council, Injury Facts.

ADB avoids blinding other vehicles or pedestrians. Adaptive headlights are beneficial for both the driver and other vehicles. For instance, the light pattern is adjusted when on a two-way sloped road to avoid glaring of approaching automobiles.

ADB adapts better to driving conditions. High and low beams on automatic adaptive headlights are automatically configured by a system of sensors. Then, the information about road conditions is transmitted to a computer. The computer combines this information together with information of sensors detecting approaching vehicles in front of it. Headlights are automatically turned on based on this data. The driver won't have to worry about blinding other drivers when using either the high beam or the low beam.

ADB improves driving in adverse weather. Road accidents are more likely when it is raining or foggy. Lack of visibility is one of the contributing factors. Fog lights are intended to be used only in fog, thick mist, snow, and other low-visibility conditions. They also point to the right far enough so that the driver may use the clear, white fog line at the road's edge as a guide.

2.2 Micro Pixel Light Source

The main goal of the micro pixel light source is to improve ordinary front lighting functionalities in order to allow the developments for autonomous driving which is explained before.

An active light distribution is made possible by the pixel light source. Only the light pixels that are necessary are turned on; the rest is inactive. As a result, the component is remarkably efficient and optimized for energy-saving solutions. One component can illuminate or shade specific areas by selectively lighting certain pixels. In addition to lighting, these components allow for the projection of patterns, symbols, brand logos, or characters so that data can be visualized. Pixel light sources allow for simultaneous illumination and visualization in a single component while helping drivers to drive safely.

The most efficient method to control the light flux consists in controlling the ratio between the time when the pixel is on compared to the time when the pixel

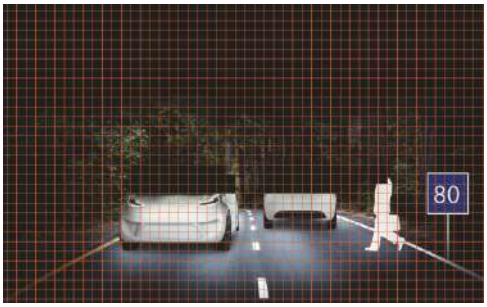

Figure 2.2: Pixel Light Source

is off. This ratio is called duty-cycle and the technique used here borrows this name. Indeed, each pixel of the matrix is driven in on-off mode, so they can be either totally on or totally off. Intermediate shades of light are not permitted unless the ratio between the on and the off period of the LED is varied and this sequence is repeated with really high frequency such that the human eye is not able anymore to distinguish the individual on and off times and senses the average light output. Usually, it is expressed as a percentage or a ratio. In the pixel light source, duty-cycle is the percentage of the ratio of pulse duration, or pulse width (PW) to the total period (T) of the waveform[5].

$$DC = \frac{PW_{ON}}{T} 100 \tag{2.1}$$

The duty-cycle is represented by a percentage. If a digital signal is on for half of the time and off the other half, it has a duty-cycle of 50% and represents an ideal square wave output. If the percentage is higher than 50%, the digital signal spends more time in the on state than in the off state, and vice versa if the duty-cycle is less than 50%. On the figure, these three scenarios are shown:

Micro Pixel light source is a device with 16384 LED matrix which is aimed to be used in automotive front light functions. The LEDs are arranged in a 256x64 matrix with 4:1 aspect ratio. Each LED represents a pixel. The brightness of every single LED can be controlled individually. In order to control the brightness, it is assigned a 10-bit duty-cycle value to every independent pixel. So, every pixel can assume a duty-cycle level between 0 and 1023, where 0 corresponds to 0% and 1023 corresponds to 100%. In other words, 0 stands for always off LED during the total period, while 1023 stands for full light LED during the total period.

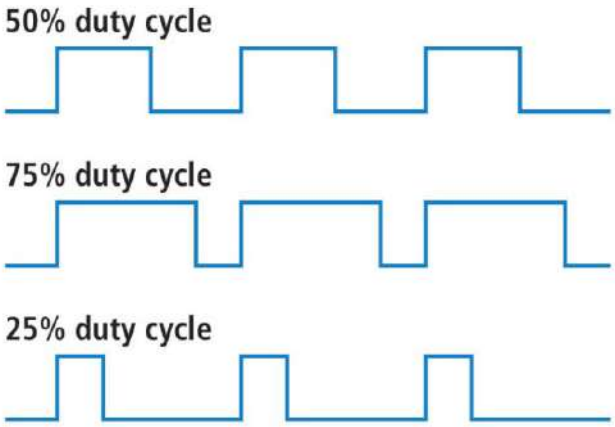

Figure 2.3: 50%, 75%, and 25% duty-cycle Examples

Even if the duty-cycle resolution is 10-bit, communication to assign duty-cycle level to each pixel is made with 8-bit resolution. Micro pixel light source device converts the received data from 8 to 10 bit by using a configurable lookup table. The lookup table associates 256 levels of light intensity to 1024[10-bit] levels of duty-cycle.

8-bit Input	**10-bit Corrected Value**
0	0
1	4
2	8
3	12
...	...
252	1008
253	1012
254	1016
255	1023

Table 2.1: Linear Gamma Correction Lookup Table

Human eye is not linear to perceive the brightness of the light. As an example with 50% of duty-cycle the human eye doesn't perceive 50% of the light that we would have with duty-cycle 100% but much more. The purpose of the lookup table is to associate a value of duty-cycle to each light level in order to compensate the natural non-linear perception of human eye and transform it to a linear scale. An example of human perception non-linearity can be seen on the figure. Further information is given in linearly and exponential dimming and

weber fechner law subsection.

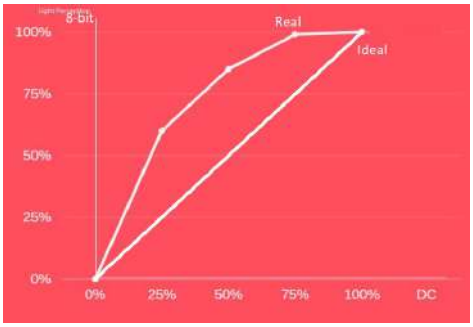

Figure 2.4: Human Light Perception versus Duty-cycle

The Lookup table associates a 8-bit value to 10-bit value. The aim of to use a lookup table is to save data. Resolution is lost because of communication channel. 10-bit data would be a lot of data to be sent. Thus, 8-bit data is sent with lower resolution. Human perception would be linear but it is not. 10-bit duty-cycle does not have to be linear depending on the human perception. In the table, it is linear but in the real case, it is not linear.

2.2.1 Linearly and Exponential Dimming and Weber Fechner Law

The actual process of dimming an LED light described with a basic principle is that to turn the LED totally on and off at a high frequency (usually between 200 and 1000 Hz) such that human eyes can no longer perceive the true on or off status. Changing the duty-cycle might give human eyes the perception that the LED is less bright than full on but also not entirely off (depending on the duty-cycle). A duty-cycle of 100% clearly indicates that the LED is entirely turned on, whilst 0% indicates that the LED is turned off. Assume you want the LED to dim linearly from 0% to 100% and back down to 0% that is, the increase in brightness should be linearly rising (and subsequently decreasing) over time. To accomplish this, just increase and then decrease the duty-cycle linearly.

This will result in a linear increase/decrease in light output, but not in a perceived linear increase/decrease in brightness. The Weber-Fechner law is responsible for this. Weber-Fechner laws are two related ideas in the domain of psycho-physics. Weber's and Fechner's laws are concerned with human

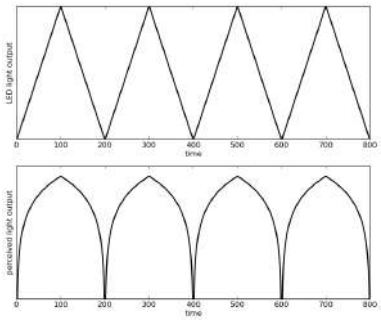

Figure 2.5: Linearly Dimming

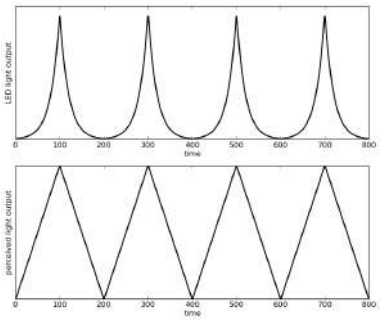

Figure 2.6: Exponential Dimming

perception, especially the relationship between the real change in a physical stimulus and the perceived change. This includes stimuli to all senses such as vision, hearing, taste, touch, and smell. Weber states that, "the minimum increase of stimulus which will produce a perceptible increase of sensation is proportional to the pre-existent stimulus," whereas Fechner's law is an inference from Weber's law (with additional assumptions) which states that the intensity of our sensation increases as the logarithm of an increase in energy rather than as rapidly as the increase. [9]

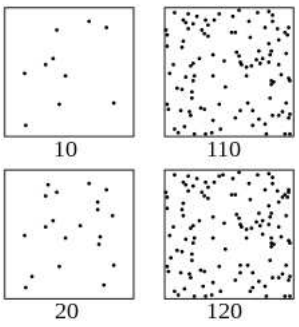

Figure 2.7: An illustration of the Weber Fechner law

As it is seen on the Figure 2.6, the bottom square has 10 more dots on each side than the upper square. However, there is a difference in perception. The distinction between the upper and bottom squares is immediately seen on the left side. The two squares on the right side are virtually identical.

This law describes the observation that human eyes perceive light in a logarithmic manner when it comes to light perception. According to the Weber-Fechner

law, if it have to raise the light output of the LED exponentially rather than linearly to create the illusion that the light is increasing linearly.

2.2.2 Fails

The LED dark and bright failure diagnosis is done by comparing the converted LED forward voltage with the LED diagnosis thresholds. The diagnosis of LED failure is depending on the operating state of the LEDs, bright failures are detected during LED off state and dark failures during LED on state. Regarding the failsafe mode, the device can be configured to deactivate the LED matrix or automatically load a fail safe picture from an external EEPROM or, display a fail safe picture received via the control interface.

2.2.3 Data Transfer Protocols

The universal serial bus and Ethernet protocols are used in order to transmit data from computer to the microcontroller that control the pixel light source. A serial peripheral interface or universal asynchronous receiver transmitter protocols are used in order to provide communication between microcontroller and the pixel light source. The device can communicate with the following integrated data transfer protocols: UART, I2C and SPI.

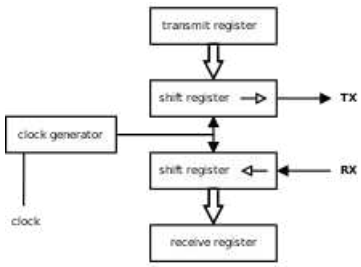

Figure 2.8: Block diagram for a UART

UART

An universal asynchronous receiver transmitter is a computer hardware interface for asynchronous serial communication. In UART communication the data format and transmission speeds are configurable. An UART is usually part of an integrated circuit utilized for serial communications over a peripheral device

serial port. Typically, microcontroller chips include one or more UART peripherals. Automobiles, smart cards, and SIMs all end up making use of specialized UARTs. UART transfers bytes sequentially in a bitwise stream. [15]

I2C

"Inter integrated circuit" protocol is a synchronous, multi controller or multi target, packet switched, single ended, serial communication bus. It is commonly known by the abbreviation I2C or IIC. I2C communication protocol is widely used for attaching lower speed peripheral ICs to processors and microcontrollers in short distance, intraboard communication. Purpose is to promote robustness and interoperability.

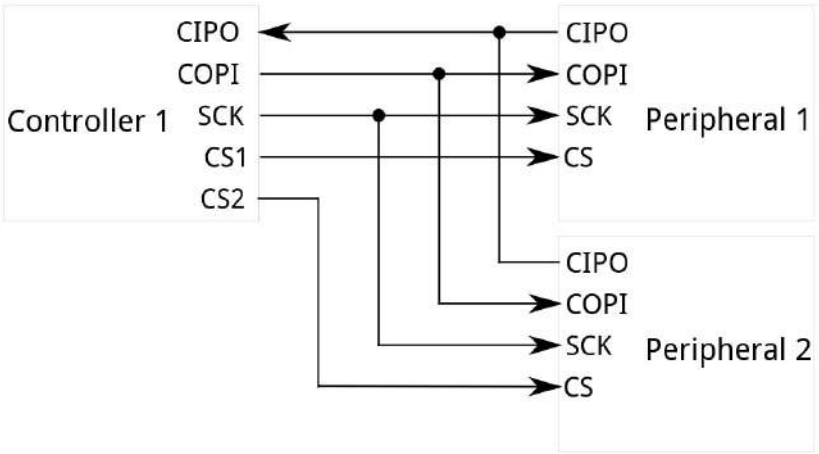

Figure 2.9: I2C Master and Slaves

I2C is a half duplex, synchronous communication protocol. It needs a clock signal to synchronize the data between source and destination. It is a serial communication protocol such as UART. It is a two wire communication protocol. The first line receives and transmits data among multiple nodes. The second one synchronizes data by transmitting the clock signal. Multi masters and multi slaves can be used.

For I2C protocol simplicity and low cost are more important than speed. A particular strength of I2C is the capability of a microchip to control a network of

other devices with just two general purpose I/O pins and software. Many other bus technologies used in similar applications, such as Serial Peripheral Interface Bus (SPI) requires more pins and signals to connect multiple devices.

SPI

The Serial peripheral interface is a synchronous serial communication interface specification used for short-distance communication, primarily in embedded systems. SPI devices communicate in full duplex mode using a master slave architecture usually with a single master.

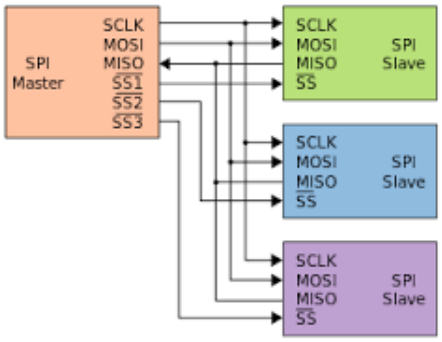

Figure 2.10: SPI Master and Slaves

The master device originates the frame for reading and writing. Multiple slave devices may be supported through selection with individual chip select, sometimes called slave select lines. Sometimes SPI is called a four wire serial bus, contrasting with three, two and one wire serial buses. Further protocols explained in the appendix.

2.2.4 Cyclic Redundancy Check Error Detection Algorithm

Any error detection technique appends redundant bits to the message. These additional bits will enable the receiver to detect whether there is an error or not. In a CRC solution these additional bits are based on a polynomial division.

The protocols that the sender and the receiver mutually agree upon, will never decide the divisor. Thus, the divisor will be known to the sender and the receiver. Therefore, based on the divisor, the CRC is going to be calculated or

the CRC is going to be generated in the sender side and verified in the receiver side. The CRC generation at the sender side involves four steps basically:

1.Find the length of the divisor 'L': 0 bits are appended.

2.Append 'L-1' bits to the original message: Suppose, if the length of the divisor is five, then it is needed to append four bits to the original message. The CRC must be of L-1 bits.

3.Perform binary division operation EXCLUSIVE OR OPERATION(XOR): After performing the binary division, we will get two results. One is the quotient and another is the remainder.

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

4.Remainder of the division is CRC.

The main advantages of the CRC are that it is more efficient than other error detection algorithms even for small bits of data because it is not bigger than the data bits itself. It is simple and low cost. It is scalable up to 64 bits. It encrypts a lot data.

```
#include <inttypes.h> // uint32_t, uint8_t

uint32_t CRC32(const uint8_t data[], size_t data_length) {
  uint32_t crc32 = 0xFFFFFFFFu;

  for (size_t i = 0; i < data_length; i++) {
    const uint32_t lookupIndex = (crc32 ^ data[i]) & 0xff;
    crc32 = (crc32 >> 8) ^ CRCTable[lookupIndex];  // CRCTable is an
   array of 256 32-bit constants
  }

  // Finalize the CRC-32 value by inverting all the bits
  crc32 ^= 0xFFFFFFFFu;
```

```
13   return crc32;
14 }
```

Code 2.1: CRC

3

Object Detection Model

3.1 Object Detection

A computer vision technique called object detection localizes and recognizes items in an image. Due to its adaptability, object detection has been the most widely utilized computer vision technique in recent years.

The terms "object detection" and "object recognition," "object identification," and "image detection" are synonymous. However, object detection is not comparable to other widely used computer vision techniques like classification (which classifies an image into a single category), key-point detection (which locates interesting areas in an image), or semantic segmentation (separates the image into regions via masks).

The object detection task identifies and labels items in an image as belonging to a target class. To do this, object detection models forecast the X1, X2, Y1, and Y2 coordinates as well as the labels for the Object class. Simply enter an image (or video frame) into an object detection model and receive a output with anticipated coordinates and class labels to use object detection in an application.

Object detection models create features from the pixels of the input image in order to generate these predictions.

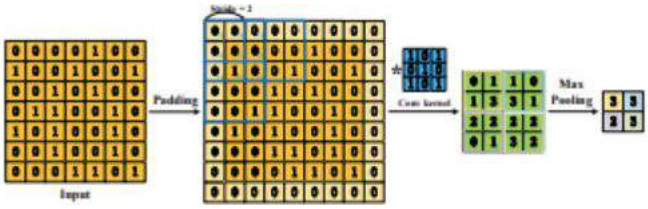

Figure 3.1: Forming features process of two dimensional CNN[1]

Going to follow formation, deep learning network is supplied with image pixel information, and predictions for class and coordinates are formed as offsets from a set of anchor boxes.

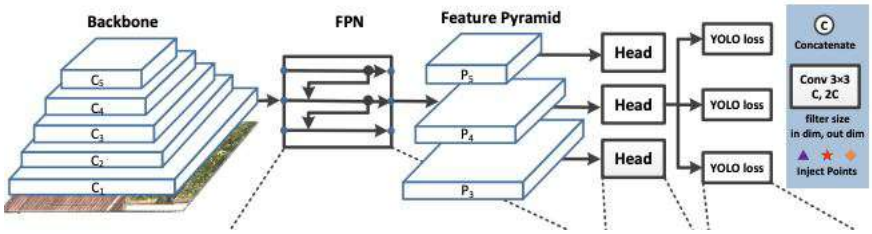

Figure 3.2: An object detection model(YOLOv3)[23]

Data is used to train the object detection model. Therefore, gathering a labeled dataset is essential if it is wanted to train an object detection model to recognize the objects of interest.

Any situation where computer vision is required to locate and recognize items in an image can benefit from object detection. In environments where items and scenery are somewhat similar, object detection flourishes.

Use cases for object detection are available at Roboflow along a wide range of industries. Just a few instances are shown below:

Example use cases for object detection: Electric scooter ID, gas leak detection, document digitization, plant phenotype, flare stack monitoring, resume parsing, augmented reality, weed detection, microscopy, bean counting, garbage cleanup, drone video analysis, conveyor belt debris, traffic counter, pothole identification, soccer player tracker, steelyard throughout, security cam analysis, self driving cars, fish measuring, remote tech support, tennis line tracking, know your customer, endangered species tracking, inventory management, hard hat detection,

pest identification, OCR math, basketball shot tracking, logo identification, satellite imagery, traffic cone finder, airplane maintenance, tumor detection, dd dice counter, plant disease finder, x-ray analysis, roof damage estimator, city bus tracking, board game helpers, dental cavity detection, drought tracking, hog confinements, sushi identifier, oil storage estimator, car wheel finder, license plate reader, exercise counter.

Generally speaking, the following categories may be used to organize object detection use cases: Aerial and Geo-spatial Imagery (e.g. for Agriculture), Drones, Manufacturing Quality Assurance, Anomaly Detection, Safety and Surveillance, Medical Imaging, Object Counting, Self Driving Cars, Retail, E-commerce, Supply Chain, Finance.

An object identification model must be trained on a set of labeled data that includes the objects of interest labeled by bounding boxes. Images can be manually annotated or by using services. It may be necessary to identify as few as 1050 images in order to launch the model. But looking forward, more labeled data will always enhance the performance and generalizability of the models.

It's critical to anticipate potential issues your model could run across when gathering the information: Including several samples of each type of object that wanted to be able to detect. By reducing the environmental variance in the dataset, the work of object detection will be simpler. Labeling a tight box around the object of interest, labeling objects that are partially cutoff on the edge of the image and labeling occluded objects as if the object was fully visible. Before it begins, ontology structure should be considered and confirmed that all of the labels are on the same page.

Generating derivative images from the basic training dataset is a part of data augmentation. This allows you to spend more time for utilizing and improving your object detection model and less time for labeling. Augmentation means that creating more training examples by distorting your input images in order to avoid the model to overfit on specific training examples. For example, it may flip, rotate, blur, or add noise.

It is time to begin training an object detection model after it has a labeled dataset and has done its augmentations. Training involves showing instances of your labeled data to a model in batches and iteratively improving the way the model is mapping images to predictions. Similar to labeling, training and inferring using object detection models may be done in one of two ways: either by training and deploying the model yourself, or by using training and inference services. Since it is the simplest to use comes equipped, YOLOv5 suggested for training.

Object detection is the most technologically advanced computer vision technology locates and recognizes things in images.

3.2 Neural Networks

An artificial neural network (ANN) is a machine learning modeling method that inspired from the human brain to create computer programs that can learn to solve the problems.

To get a basic understanding of the math behind a neural network, it is likely needed to know a little linear algebra for the forward feed method and knowledge of derivatives. It will be using basic partial derivatives, but an understanding of a normal derivative should be good enough. The forward feed method is based on nonlinear activation functions and dot products. Each neural network contains an output layer, a few hidden layers, and a vector of inputs. The inputs may be mistaken for a layer, but since no computations are performed, they are just regarded as a vector or data matrix.

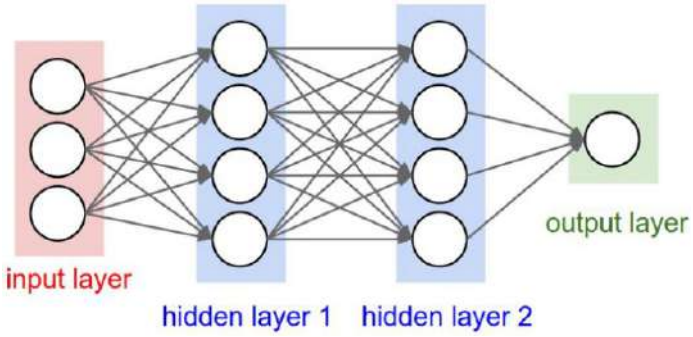

Figure 3.3: Model of a deep neural network[13]

There are several nodes in each layer. Take note that the output layer has 1 node, the first hidden layer has 4 nodes, and the second hidden layer also has 4 nodes. The number of the nudes that each layer has is optional, but as the number of nodes increases, so does the computational power required to train the network. Considered also, how each node is connected to all other nodes on both the previous and next layers by many lines. The parameters we want to adjust in the neural network are these lines, which are referred to as weights.

Let's review some notation before diving into the forward feed formulas for a neural network:

x_i = The ith input into the node

w_i = The ith weight going into the node

b = The bias for that node

z = The output of a node before the activation function is applied

a = The output of a node after the activation function is applied

A representation of a node is shown in the image below.

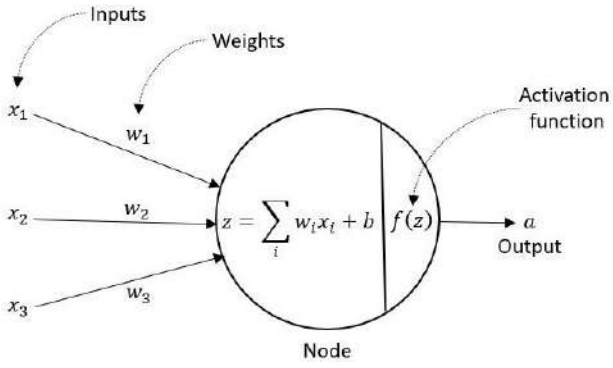

Figure 3.4: A neural network node

Node The linked nodes that constitute an artificial neural network are arranged in layers. The fundamental unit of a neural network is a node, which is similar to a neuron in the human brain. A node gets activations, or inputs, from other nodes' outputs or from data in the input layer. A node's inputs are each linked by a weight (w). The magnitude of a node's influence from an input is indicated by a weight. The output of the node is calculated by computing a weighted sum of the inputs and then applying an activation function.

In a neural network, an activation function is a non-linear conversion of input values which is required to enable modeling of complex tasks. The weighted sum of inputs is a linear transformation. This linear value z is then applied an activation function f(z) which is non-linear. In the field of machine learning, a lot of activation functions are commonly used such as Sigmoid, TanH, ReLU, and Softmax. A sigmoid function which converts linear values to a value between 0 and 1.

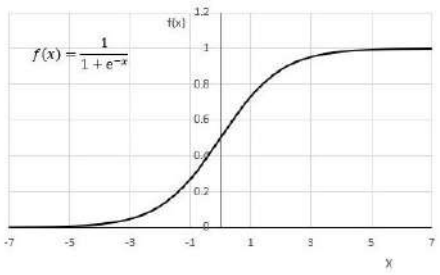

Figure 3.5: Sigmoid Function

Loss Function In a neural network, the loss function calculates the difference between the estimated and actual values. In order to find the optimal model parameters which are weights, the total loss value is minimized. A loss function used depends on the goal of the model. Mean Squared Error(MSE) is a common loss function for predicting outcomes.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

where y is the true value and $\hat{y}$ is the predicted value.

If the goal is classification, a binary cross entropy loss function is used for a binary class, and a cross entropy loss function is used for a multi-class classification.

$$\text{Binary Cross Entropy} = -\left(y \log(\hat{y}) + (1-y) \log(1-\hat{y})\right)$$

$$\text{Cross Entropy} = -\sum_{i=1}^{k} y_i \, log(\hat{y}_i)$$

where k is the number of classes

Neural Network An input layer, a hidden layer, and an output layer are the three layers that compose a simple neural network. Sigmoid activation function and a binary cross entropy loss function will be used if the task is categorization.

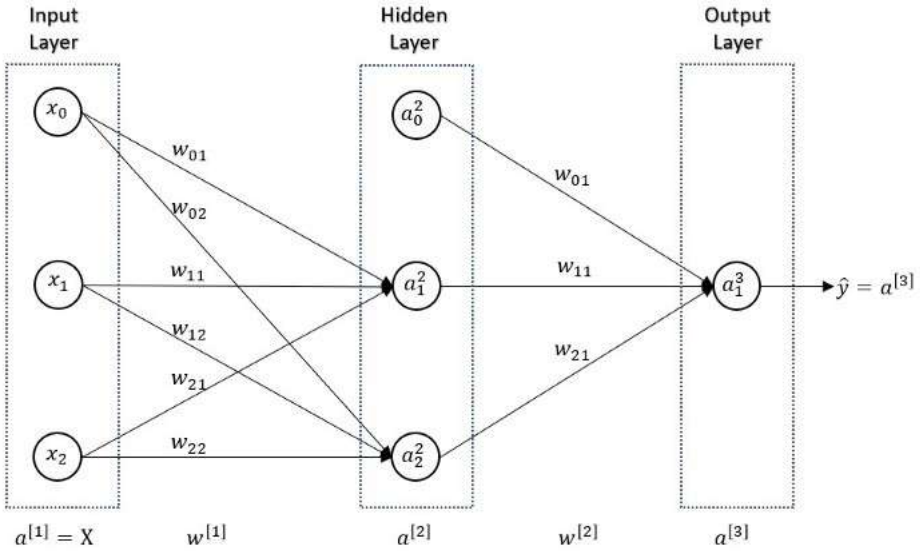

Figure 3.6: Simple neural network

Input Layer Matrix X has two features x_1 and x_2. x_0 represents the bias term which has been assigned a constant value of 1. The resulting X matrix, therefore, has three columns: x_0, x_1, and x_2. It is important to note that the features are columns and the observations are rows. The activation vector $a^{[1]}$ is equal to the input values X in the input layer.

Hidden Layer The hidden layer has two nodes a_2^2 and a_3^2. The superscript refers to the layer number and the subscript indicates the node number. The

weights from the input layer connect to these two nodes only. Node a_2^1 is added in layer 2 in order to represent the bias term. $a^{[2]}$ is the activation vector in the hidden layer.

Output Layer The output layer has only one node since it has a binary class. The activation vector $a^{[3]}$ is equal to estimates $\hat{y}$.

Weights There are two sets of weights. $w^{[1]}$ is the weight matrix between the input and the hidden layer. $w^{[2]}$ is the weight matrix between the hidden and output layer. The rule to determine the dimensions of the weight matrix is excluding the bias node in layer j+1.

j: Number of nodes in layer

j+1: Number of nodes in layer j+1

The following procedures are involved in training a neural network:

1. Initializing the weights: First, the weights should be initialized with small, random numbers between 0 and 1.

2. Forward Propogation In order to complete this process, all nodes in the hidden and output layers that pass through the input and output layers must have their outputs calculated. Each node's computation is divided into two steps:

i. Finding z = α w. The relationship between activation function and weights is linear.

ii. Applying sigmoid activation function on z.

A ReLU function uses the maximum value between 0 and z in the manner described below:

$$\alpha = ReLU(z) = max(0, z) \tag{3.1}$$

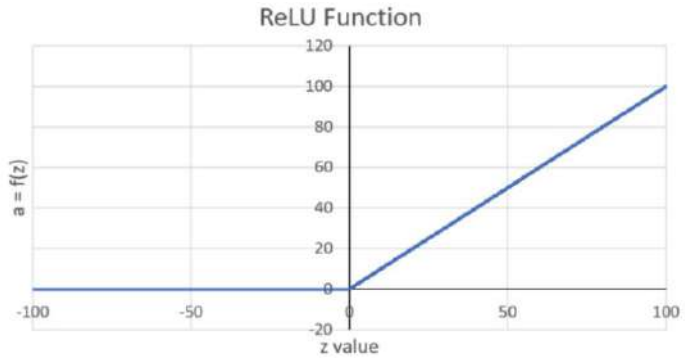

Figure 3.7: ReLU Function

The node calculation is finished after the α value has been calculated, at which point it may be passed on to proceeding layers. The computation that is just performed only applies to one node in a layer. Can be probably anticipate performing this calculation many times for each node in a layer as layers often include more than one node.

The forward feed part of a neural network ends there. Each node in the network follows to the formulas indicated above, and the network is formed of layers that are constructed of nodes.

Dot products are applied instead of a single node to calculate the z value for an entire layer for faster computation times:

$$z = x.w^T + b \tag{3.2}$$

The dot product of the inputs, the transposed weights (the T superscript signifies transpose the values), and the bias for each node may then be used to calculate the z values for a layer. Then, the z values can be applied to a ReLU function. An example of how to use numpy in Python is provided below:

```
#import numpy as np

inputs = np.array([1,2,3,4])
weights = np.array([[1, 0.5, -0.5, 1],
                    [0.5, -1, 1, -0.5],
                    [0, 1, -1, 0.5]])
biases = np.array([10,9,8])

```

```
# Calculate the z value
z = np.dot(inputs, weights.T) + biases

# Calculate the activation value
a = np.maximum(0, z)
```

Code 3.1: Calculation with Python

In this example, the previous layer has 4 nodes which is why the input has 4 values. The layer we are calculating has 3 layers but each node in this layer has to take in 4 inputs. This is why the weights array is a 3x4 array (3 nodes, 4 inputs). There are 3 biases, 1 for each node in this layer.

Loss Function

The loss value is calculated using binary cross entropy as following:

$$L(w) = -\frac{1}{n}\sum_{i=1}^{n}(yi\ log(\hat{y}_i) + (1 - y_i)\ log(1 - \hat{y}_i))$$

The prediction $\hat{y}$ using weights $w^{[1]}$ and $w^{[2]}$ incurred a loss. The objective is to find the weights that provide estimates as close as possible to the true outputs. Neural network achieves this by using gradient descent, an optimization algorithm, that minimizes the loss by updating the weights by an amount proportional to the negative of the gradient (partial derivative of the loss function with respect to the weights).

$$w = w - \alpha\frac{\delta L(w)}{\partial w} \tag{3.3}$$

3. Backward Propagation The partial derivative or the gradient is calculated using a process called backward propagation. First, error contributed by each node called delta calculated by traversing backward from the output layer towards the input layer. The error term is then multiplied with the activation value of the node to determine partial derivative.

δ_j^l = error of node j in layer l

$$\frac{\delta L(w)}{\delta w_{ij}^l} = \alpha_i^l \delta_j^{l+1} \tag{3.4}$$

As a result, now a new set of weights are utilized to compute the prediction as well as the corresponding loss value. The loss converges to a minimal value using an iterative process that includes forward propagation, backpropagation, and weight updates. The corresponding weights are ideal when a minimum loss is obtained.

Every time a new update is done by using the new loss value, found that the loss decreases again and will continue doing so after every consecutive update. This is why backpropagation is so useful and good at optimizing a loss function. The calculations should apply the same math to all other nodes in all other layers.[14]

Below, a Python code that updates the network 100 times. Notice how the loss decreases and approaches 0 which is the lowest the loss value can be.

```
import numpy as np

# The parameters
x1 = 2
x2 = -4
w1 = 0.5
w2 = -0.5
b = 1
alpha = 0.1
# Loop 100 times which is the number of times
# an update will happen to the network
for i in range(0, 100):
    # Forward feed
    z = x1*w1 + x2*w2 + b
    a = max(0, z)
    Loss = a
    print(f"Loss at step #{i+1} : {Loss}")

    # Backpropagation
    dLoss = 1
    da = 1
    dz = (1 if a >= 0.5 else 0)
    dLoss_dz = dz*da
    dx1 = w1
    dw1 = x1
    dx2 = w2
```

```
    dw2 = x2
    db = 1

    dLoss_dx1 = dx1*dz*da
    dLoss_dw1 = dw1*dz*da
    dLoss_dx2 = dx2*dz*da
    dLoss_dw2 = dw2*dz*da
    dLoss_db = db*dz*da

    # Update the parameters
    w1 = w1 - alpha*w1
    w2 = w2 - alpha*w2
    b = b - alpha*b
```

Code 3.2: An example how to update the weights with the loss function

That is the basic aspect of a neural network, more precisely an MLP (multi-layer perception) network. The model we just discussed is the basis for practically all other neural network models. Other types of neural networks include convolutional neural networks (CNN), which deal with images, and recurrent neural networks (RNN), which deal with text.

3.2.1 Convolutional Neural Networks

The capability of artificial intelligence to close the gap between human and machine capabilities has increased significantly. Both professionals and hobbyists focus on many aspects of the field to achieve great results. The field of computer vision is one of several such disciplines. CNNs are the basis of the YOLO algorithm

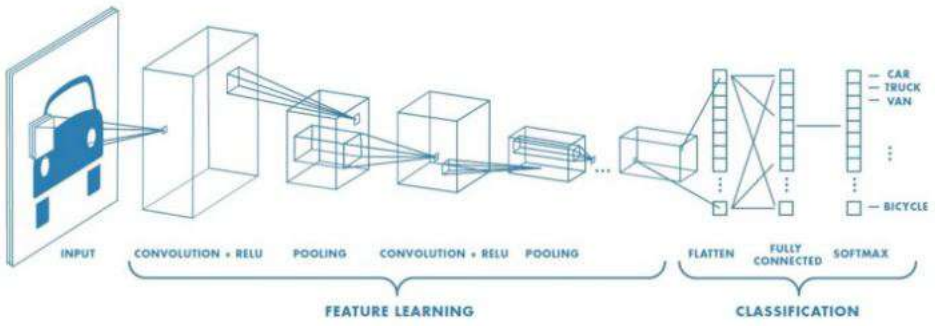

Figure 3.8: Convolutional Neural Networks

The aim of the discipline of neural networks is to give machines the ability to see the environment similarly as humans do, to observe it that way, and to even apply the data for a variety of tasks including Image Video recognition, Image Analysis Classification, Media Recreation, Recommendation Systems, Natural Language Processing, etc.

The breakthroughs in Computer Vision using Deep Learning have been built and improved over time, mostly over one specific algorithm called Convolutional Neural Networks.

A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning method that can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image, and be able to distinguish between them. Comparatively speaking, a CNN requires much less pre-processing than other classification techniques. CNNs have the ability to learn these filters and characteristics, whereas in primitive methods filters are hand-engineered.

A CNN's architecture was influenced by how the Visual Cortex is structured and is similar to the connectivity pattern of neurons in the human brain. Only in this constrained area of the visual field, known as the Receptive Field, could individual neurons react to stimuli. A collection of such fields overlap to cover the entire visual area.

Here is an example:

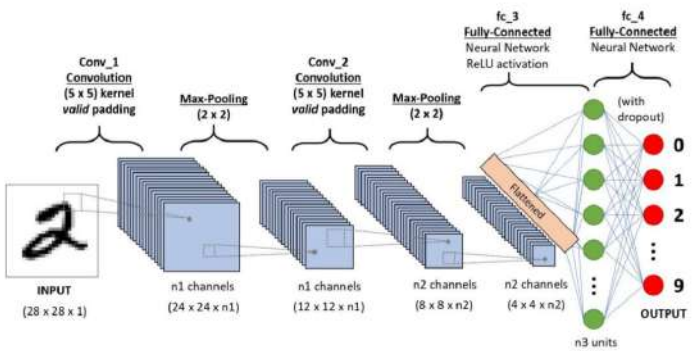

Figure 3.9: A CNN sequence to classify handwritten digits

An image is nothing but a matrix of pixel values. Thus, why not just flatten the image for example converting a 3x3 image matrix into a 9x1 vector and feed it to a Multi-Level Perceptron for classification purposes? Well, this would not be the correct approach.

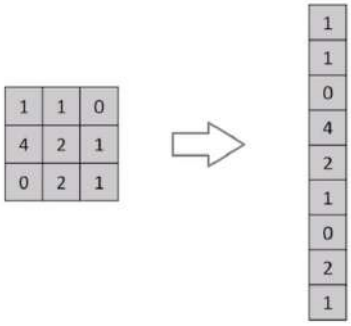

Figure 3.10: Flattening of a 3x3 image matrix into a 9x1 vector

The method might perform class prediction with an average precision score for extremely basic binary images, but it would perform with little to no accuracy for complex images with internal pixel dependencies.

A CNN may effectively capture the spatial and temporal relations in an image by using the appropriate filters. Due to the reduction in the number of parameters needed and the reuse of weights, the architecture achieves a better fitting to the image dataset. In other words, the network may be trained to better comprehend how complex an image is.

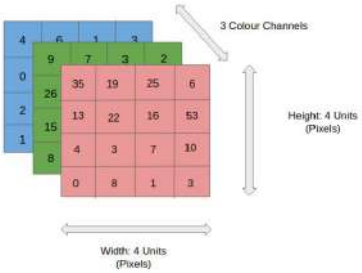

Figure 3.11: 4x4x3 RGB Image

The three color planes of the RGB image (Red, Green, and Blue) have been used to separate it in the image. Images can be found in a variety of different color spaces, including grayscale, RGB, HSV, CMYK, etc.

The CNN's task is to reduce the pictures into a shape that is simpler to analyze without missing features that are essential for obtaining an accurate prediction. This is crucial for creating an architecture that is both scalable to massive data and effective at learning features.

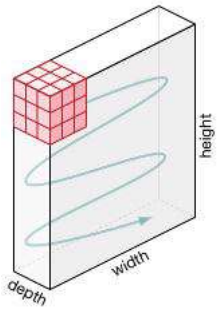

Figure 3.12: Movement of the Kernel

Until the entire width is parsed, the filter travels to the right with a specific Stride Value. Again when the entire image has been traversed, it jumps back up to the image's beginning (on the left) with the same Stride Value.

In the case of images with multiple channels for example RGB, the Kernel has the same depth as that of the input image. Matrix Multiplication is performed and all the results are summed with the bias to give us a squashed one-depth channel Convoluted Feature Output.

The aim of the Convolution Operation is to take the input image's high-level features, like edges, and extract them. There is no restriction that CNNs have only one convolutional layer. Typically, low-level features like edges, color, gradient orientation, etc. are captured by the first Convolutional Layer. With more layers added, the architecture adjusts to High-Level features as well, giving us a network that comprehends the dataset's images completely, much like human do.

There are two different kinds of outcomes from the operation: one in which the dimensionality of the convolved feature is reduced in comparison to the input, and the other in which it is either increased or stays the same. This is accomplished by using Valid Padding in the first scenario or Same Padding in the second.

For example, When augmented the 5x5x1 image into a 6x6x1 image and then apply the 3x3x1 kernel over it, found that the convolved matrix turns out to be of dimensions 5x5x1. Hence the name would be Same Padding. On the contrary, if the same operation performed without padding, presented with a matrix which has dimensions of the Kernel (3x3x1) itself, hence the name would be Valid Padding.

The Pooling layer, like the Convolutional Layer, is in charge of reducing the Convolved Feature's spatial size. By dimensionality reduction, this will reduce the amount of computing power needed to process the data. Additionally, it helps with the extraction of dominant features that are rotational and positional invariant, sustaining the model's successful training process.

Max Pooling and Average Pooling are the two different types of pooling. The largest value from the area of the image that the Kernel has scanned is returned by Max Pooling. On the other hand, the average of all the values from the area of the image captured by the Kernel is what is returned by Average pooling.

Noise Suppression is another function of Max Pooling. Along with performing de-noising and dimensionality reduction, it completely discards the noisy activations. On the other hand, Average Pooling simply performs dimensionality reduction as a noise suppressing mechanism. Therefore, we can say that Max Pooling performs a lot better than Average Pooling.

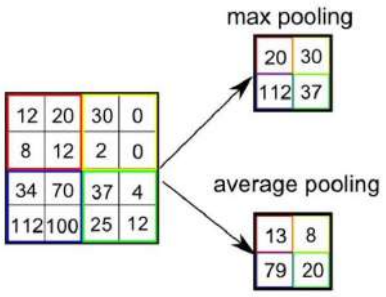

Figure 3.13: Types of pooling

The i-th layer of a convolutional neural network is composed of the convolutional layer and the pooling layer. Depending on the complexities in the images,

the number of such layers may be increased for capturing low-levels details even further, but at the cost of more computational power.

The output of the convolutional layer is a representation of the high-level features, and adding a Fully-Connected layer is a usually a low cost approach to learn non-linear combinations of those features. A potentially non-linear function in that space is being learned by the fully connected layer.

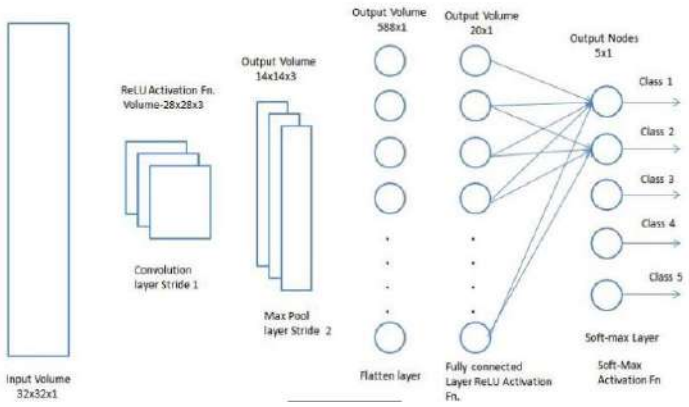

Figure 3.14: Fully connected layer

Converted the input image into a suitable form for our Multi-Level Perceptron, it shall flatten the image into a column vector. The flattened output is fed to a feed-forward neural network and backpropagation applied to each iteration of training. Over a series of epochs, the model is able to distinguish between dominating and certain low-level features in images and classify them using the Softmax Classification method.

There are several CNN designs that may be used, and these architectures have been essential in creating the algorithms that power and will continue to power AI as a whole in the near future. Below is a list of some of them: LeNet, AlexNet, VGGNet, GoogLeNet, ResNet,ZFNet.

3.3 Model Validation

Machine learning depends on model validation because it is the process used to determine whether a model is accurate. The evaluation must be precise and

objective at the same time because it is such a crucial stage in the learning process. Quantitative measures, either information-theoretic based or based on matrix reduction, are utilized to provide a single numerical representation of a model's quality in order to satisfy the objectivity requirement. It is typical to report the error or misclassification rates for this model's quality while also their inverse, accuracy. Performing model evaluation for multi-class problems is not just a simple extension of the binary case. Many techniques, especially those with information theoretic frames, lack multi-class extensions.

3.3.1 mAP (mean Average Precision) for Object Detection

The confusion matrix is being used as a random variable, and the information content is being assessed as such. These methods often ensure a high level of matrix discrimination, which is useful for identifying variations in misclassification rates from associated matrices and advantageous when class distributions are biased toward one class. As seen anecdotally by their scarcity in the multi-class prediction evaluation literature, extensions of these measures are unfortunately difficult to create due to the nature of information theoretic derivations, which strongly rely on non-trivial differential entropy. To get misclassification data, the other branches of measurements rely on confusion matrix reduction and transformation[12] The k times k matrix entries are reduced down to a specific value that represents the classification accuracy using individual elements and sums. Since information loss is unavoidable when condensing a kxk table into a single value, this simplicity frequently comes at a price[4].

The number of instances with the true label I categorized into group j is represented by c_{ij}, and C^k indicates a confusion matrix or the contingency table of actual class labels by their model predictions. A 2x2 confusion matrix creation example is provided in the table.

		Predicted		
		Class 1	Class 2	Total
Actual	Class 1	c_{11}	c_{12}	$c_{11} + c_{12}$
	Class 2	c_{21}	c_{22}	$c_{21} + c_{22}$
	Total	$c_{11} + c_{21}$	$c_{12} + c_{22}$	N

Figure 3.15: 2x2 Confusion matrix

Accuracy can be calculated as the percent of correctly identified observations over all observations. Accuracy is defined as the following in confusion matrix:

$$Accuracy = \frac{c_{11} + c_{22}}{c_{11} + c_{12} + c_{21} + c_{22}} \tag{3.5}$$

Average precision is a common metric used to assess the accuracy of object detectors like Faster R-CNN, SSD, etc.. For recall values greater than or equal to 0, average precision calculates the average precision value. It appears to be complicated but is fact quite straightforward. However, confusion matrix, precision, recall, and IoU will quickly review. The precision of the predictions is measured. With the other words, what percent of the predictions that model made are accurate. Recall rates how well the model remembers all the (true) positives.

$$Precision = \frac{TP}{TP + FP} \tag{3.6}$$

$$Recall = \frac{TP}{TP + FN} \tag{3.7}$$

$$F1 = 2.\frac{Precision * Recall}{Precision + Recall} \tag{3.8}$$

IoU (Intersection over union) calculates how much two borders intersect. It utilizes that to determine the coefficient to which our predicted boundary overlaps the real object boundary (ground truth). In order to categorize whether a prediction is a true positive or a false positive in specific dataset, IoU threshold is predefined for instance 0.5. On the Figure 3.2, the orange bounding box represents the prediction and the blue bounding box represents the ground truth. Here are some definitions in mathematics:

$$IoU = \frac{AreaofOverlap}{AreaofUnion} \tag{3.9}$$

Finding the area beneath the precision-recall curve is the general definition of the Average Precision (AP). Always, precision and recall fall between 0 and 1.

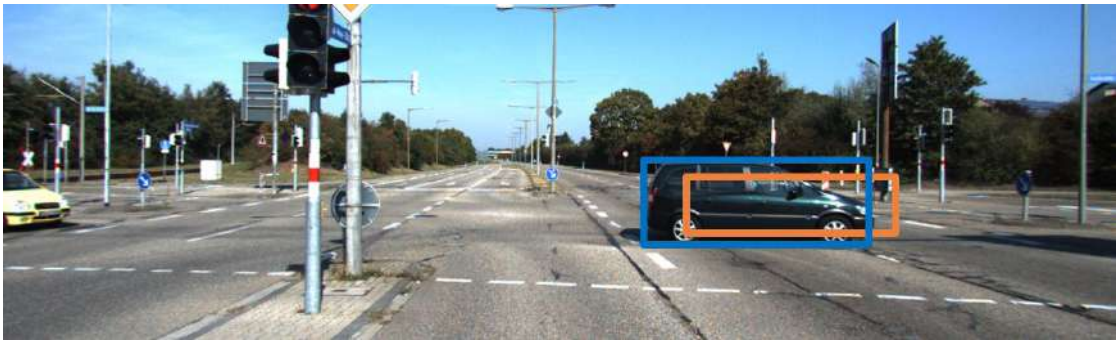

Figure 3.16: An image from KITTI dataset with bounding boxes

Figure 3.17: Intersection over union

Accordingly, AP goes within 0 and 1 also.

$$\int_0^1 p(r)dr \tag{3.10}$$

The average of AP is mAP. In some circumstances, the AP calculated and averaged for each class. While in other circumstances, they have always had the same meaning. For instance, there is no distinction amongst AP and mAP in the COCO dataset:

> *AP is averaged over all categories. Traditionally, this is called mean average precision (mAP). We make no distinction between AP and mAP (and likewise AR and mAR) and assume the difference is clear from context.*

Recent research publications typically solely present the findings for the COCO dataset. The calculation in COCO mAP uses a 101-point interpolated AP definition. AP for COCO is the average over several IoU. (the minimum IoU to consider a positive match). The average AP for IoU from 0.5 to 0.95 with a step size of 0.05 is indicated by AP@[.5:.95]. For the COCO competition, the average point (AP) is computed using 80 categories and 10 IoU levels (AP@[.50:.05:.95]: start at 0.5 and proceed up to 0.95 with a 0.05 step size). The COCO dataset also includes the extra measures.

3.4 History Of YOLO Model

Object detectors may be divided into two types one-stage detectors, and two-stage detectors. In the head, detection takes place. For each bounding box, the tasks of object localization and classification are separated by two-stage detectors. The predictions for object localization and classification are made simultaneously by one-stage detectors. You Only Look Once refers to the one-stage detector.

Joseph Redmon, who is currently retired from computer vision, created the first YOLO (You Only Look Once) using a novel framework called Darknet. The greatest real-time object detectors in computer vision have been created by Darknet, a very flexible research framework implemented in low level languages, including YOLO, YOLOv2, YOLOv3, and recently YOLOv4.

YOLOv1 The first end-to-end differentiable object detection network, YOLO, combined the bounding box drawing and class label identification problems.

YOLOv2 On top of YOLO, YOLOv2 included a number of incremental enhancements like BatchNorm, better resolution, and anchor boxes.

YOLOv3 By including an objectness score into bounding box prediction and adding connections to the backbone network layers to increase performance on smaller objects, YOLOv3 improved upon earlier models.

YOLOv4 Model has a wide variety of computer vision algorithms for object recognition These tested and improved methods have resulted in the best real-time object detector in the game, which is lightweight and simple to use.

YOLOv5

> *Using our system, you only look once (YOLO) at an image to predict what objects are present and where they are.*

Ultralytics published the first official version of YOLOv5 on June 25th. Here explored the revolutionary technologies used in the initial YOLOv5 version

and analyzed preliminary performance results of the new model. In summary, the YOLO model is a quick compact object recognition model that is quite performant compared to its size and has been gradually improving over time.

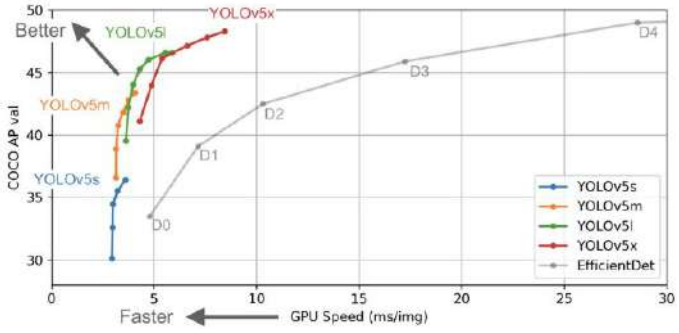

Figure 3.18: YOLOv5 shows promise of state of the art object detection

The aim of the graph is to create an object detector model that is particularly performant (Y-axis) in compared to its inference time (X-axis). Preliminary findings demonstrate that YOLOv5 does exceedingly well to this end relative to other state of the art techniques. In conclusion, YOLOv5 gets the majority of its performance improvement from PyTorch training processes (Yolov4 has darknet framework), while the model architecture remains similar to YOLOv4.

Glenn Jocher's YOLOv5 PyTorch repository is a natural extension of his YOLOv3 PyTorch repository. The YOLOv3 PyTorch repository was a popular location for developers looking to transfer YOLOv3 Darknet weights to PyTorch before moving on to production. Many people (including our Roboflow vision team) appreciated the PyTorch branch's ease of use and would utilize it for deployment.

Ultralytics began to make research improvements alongside repository re-design after fully replicating the model architecture and training procedure of YOLOv3, with the goal of empowering thousands of developers to train and deploy their own custom object detectors to detect any object in the world, a goal we share here at Roboflow.

An object detector is intended to generate features from input images, which are then supplied into a prediction system to construct boxes around objects and

predict their classes.

The YOLO model was the first to link the method of predicting bounding boxes with class labels in an end-to-end differentiable network.

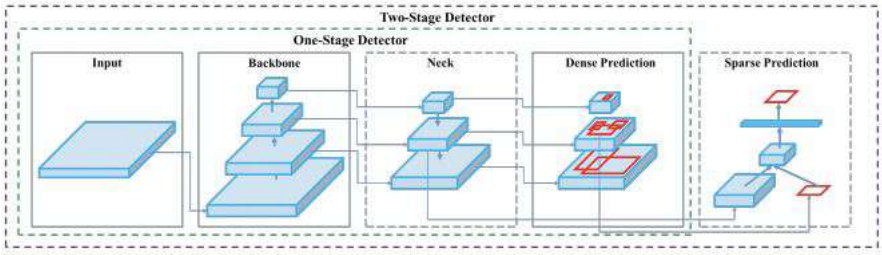

Figure 3.19: YOLOv5 shows promise of state of the art object detection

The YOLO network is composed of three primary components.

Backbone: A convolutional neural network that accumulates and generates image features at various levels of granularity.

Neck: A set of layers used to mix and combine image features before passing them on to prediction.

Head: Consumes neck features and performs box and class prediction procedures.

Of course, there are several techniques of combining various designs at each major component. The contributions of YOLOv4 and YOLOv5 are primarily to combine breakthroughs in other fields of computer vision and demonstrate that, as an unit, they improve YOLO object identification.

On the object detection models, training processes are similarly essential for the overall performance of an object detection system, while getting less emphasis.

Data Augmentation Data augmentation involves transforming the underlying training data in order to expose the model to a wider range of semantic variance than the training set alone.

Loss Calculations YOLO derives a total loss function from the constituent loss functions (IoU, obj, and class losses). These can be properly designed to achieve the goal of maximizing the mean average precision(mAP).

PyTorch Translation for YOLOv5

YOLOv5's most significant contribution is the translation of the Darknet research framework to the PyTorch framework. PyTorch is a popular open source deep learning framework developed by Facebook. It has a focus on accelerating the path from research prototyping to production deployment. The Darknet framework is mostly written in C and provides fine-grained control over the network's activities. Control over the lower level language is beneficial to research in many ways, but it might make it more difficult to incorporate new research discoveries since each new addition requires specialized gradient computations. It takes a lot of effort to translate and exceed the training techniques in Darknet to PyTorch in YOLOv3.

Figure 3.20: PyTorch[17]

Data Augmentation in YOLOv5 YOLOv5 runs training data through a data loader, which augments data online, with each training batch. Scaling, color space changes, and mosaic augmentation are the three types of augmentations performed by the data loader. The most unique is mosaic data augmentation, which mixes four photos into four random ratio tiles. The mosaic data loader is included in the YOLOv3 PyTorch repository, as well as the YOLOv5 repository.

Mosaic augmentation is particularly effective for the widely used COCO object detection reference point, assisting the model in learning to address the well-known "small object problem," in which little objects are not spotted as accurately as bigger ones.

The YOLO network predicts bounding boxes as deviations from a list of anchor box dimensions in order to generate box predictions.

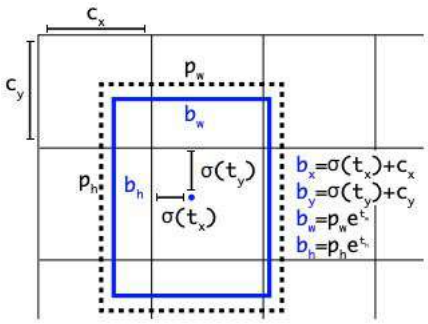

Figure 3.21: Bounding Boxes from YOLOv3

Glenn Jocher presented the idea of learning anchor boxes based on the distribution of bounding boxes in the custom dataset using K-means and genetic learning methods in the YOLOv3 PyTorch repo. This is essential for specific projects since the distribution of bounding box sizes and positions may differ significantly from the COCO dataset's default bounding box anchors. The most radical variation in anchor boxes may arise while attempting to detect giraffes that are very tall and skinny, or manta rays that are very wide and flat.

The PyTorch framework lets developers to reduce floating point precision in training and inference from 32 bit to 16 bit. This considerably speeds up the inference time of the YOLOv5 models. However, at the moment, the speed increases are only offered on the select of GPUs.

Both YOLOv4 and YOLOv5 implement the Cross Stage Partial Networks Bottleneck to create image features, with research credit to WongKinYiu and their paper on Cross Stage Partial Networks for Convolutional Neural Network Backbone. The CSP overcomes duplicate gradient concerns in other bigger ConvNet backbones, resulting in fewer parameters and FLOPS for comparable relevance. This is especially crucial for the YOLO family, where inference speed and minimal model size are critical.

CSP models has the DenseNet base. DenseNet was built to link layers in convolutional neural networks in order to address the vanishing gradient problem (it is difficult to backprop loss signals via a very deep network), improve feature propagation, stimulate feature reuse, and minimize the number of network parameters.

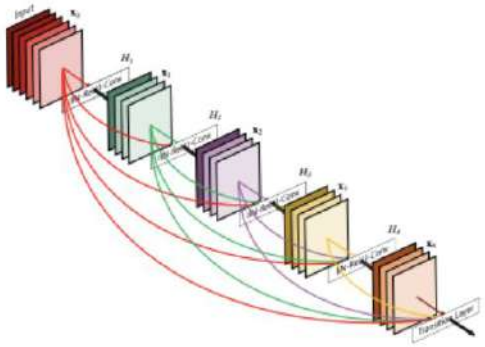

Figure 3.22: DenseNet[6]

The DenseNet has been edited to separate the feature map of the base layer by copying it and sending one copy through the dense block and sending another straight on to the next stage. The idea is to remove computational bottlenecks in the DenseNet and improve learning by passing on an unedited version of the feature map.

The PA-NET neck is used by YOLOv4 and YOLOv5 for feature aggregation.

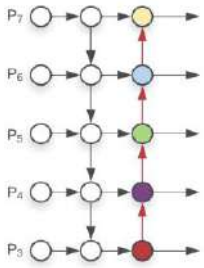

Figure 3.23: PaNet[10]

Above, each one of the P represents a feature layer in the CSP backbone.

YOLOv5 is incredibly simple to use for a developer implementing computer vision technologies into an application in comparison to other object identification frameworks.

Simple Installation: YOLOv5 simply needs PyTorch and a few lightweight Python libraries installed.

Fast Training: As the model developed, the YOLOv5 models train very rapidly, which helps to save cost on experimenting.

Working inference ports: With YOLOv5, you may infer on individual photos, batches of images, streaming video, or webcam ports.

Adaptive Layout While constructing, file folder arrangement is intuitive and straightforward to navigate.

Translation to Mobile: The translation of YOLOv5 from PyTorch weights to ONXX weights to CoreML to iOS is simple.

Similar to YOLO Darknet TXT, the YOLOv5 PyTorch TXT annotation format includes a YAML file with model settings and class values. It is needed to make sure to adapt annotations to work with YOLOv5 based on the annotation tool it is used.27 different labeling formats may be imported via Roboflow and exported for use with YOLOv5.As suggested YOLOv5 labeling tool, Ultralytics, the company that created YOLOv5, collaborates with Roboflow.

YOLOv5's release is reasonably fast, performant, and simple to use. YOLOv5 includes a new PyTorch training and deployment framework that advances the state of the art for object detectors, although it has not yet added innovative model architectural enhancements to the family of YOLO models. Additionally, YOLOv5 is highly user-friendly and is available ready to use on customized objects.

3.4.1 YOLOv1

"YOLO" stands for "You Only Look Once," a model family presented by Joseph Redmon in his May 2016 work, "You Only Look Once: Unified, Real-Time Object Detection."

YOLO is presented as a new approach to object detection. Different than former detection systems, YOLO approaches the object detection as a regression task to bounding boxes which is spatially separated and their classification probabilities that is a prediction of a single network. YOLO can reach double the

mAP of other detectors. Trade off of YOLO model is it makes more localization errors but it's probability to predict FPs is lower.

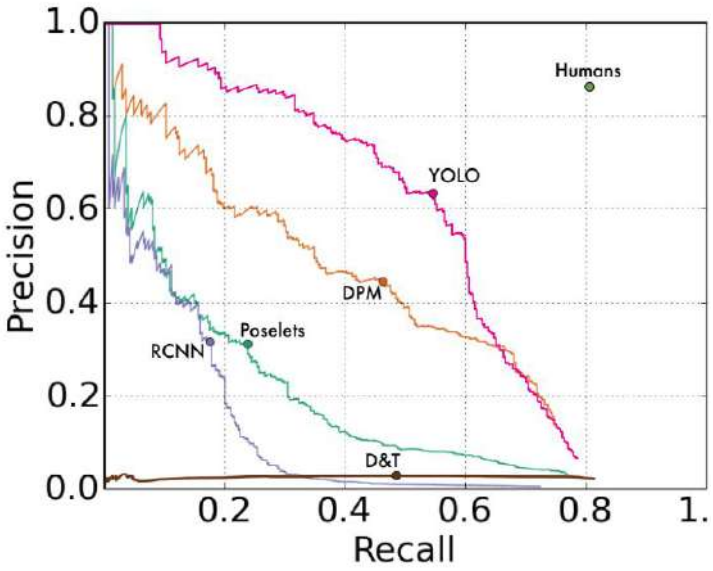

Figure 3.24: Picasso Dataset precision-recall curves comparing to the different object detection models and Human eye

The human visual system is quick and precise, permitting us to carry out difficult activities like driving with little conscious effort. Fast, precise object detection algorithms would open the door to general-purpose, responsive robotic systems, the capability for assistive devices to transmit real-time scene information to human users, and the ability for computers to drive cars without specialized sensors. Deformable parts models (DPM) apply a sliding window technique where the classifier is run across the full image at evenly spaced locations in order to utilize classifiers to perform detection[16]. Instead R-CNN, create probable bounding boxes for an image using region proposal methods before implementing a classifier to the proposed boxes. Then, post-processing is used to optimize the bounding boxes after classification, get rid of duplicate detections, and give the boxes new scores based on other objects in the image[18]. These complex models are slow and challenging to optimize.The particular reason is each component must be trained separately.

YOLO runs pretty fast. It doesn't require a complicated workflow because detection is considered as a regression problem. In order to predict the de-

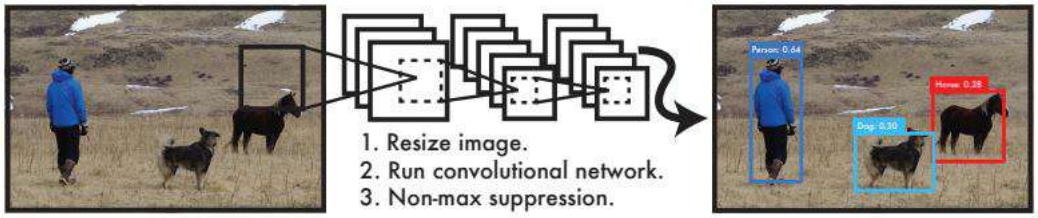

Figure 3.25: YOLO Detection System

tections, we only run our neural network on a raw image at test time. YOLO achieves more than double what other real-time systems' mean average precision is. YOLO uses a broad reasoning approach to make predictions about the image. Because YOLO sees the full image during training and testing, with the exception of sliding window and region proposal-based approaches, it implicitly retains contextual information about classes in addition to their appearance. Compared to Fast R-CNN, YOLO gets lower than half as many background mistakes because it can see larger context [8]. A generalizable model of an object is learned through YOLO. Because of this reason, it is low probability to fail when used in unfamiliar domains or with unexpected inputs.

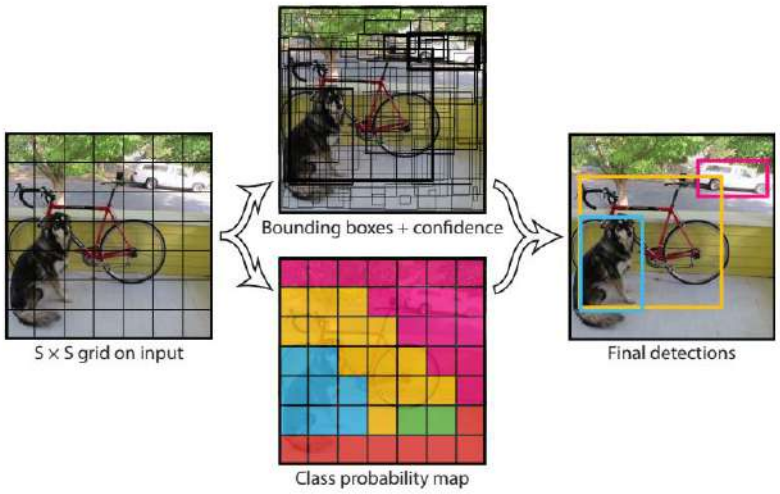

Figure 3.26: YOLO Model Detection

Detection is modeled as a regression task. It creates a S x S grid out of the image and estimates B bounding boxes, confidence in those boxes, and C classification confidence score for each grid cell. These predictions are given by

the tensor S x S x (B * 5 + C).

$$Pr(Class_i|Object) * Pr(Object) * IOU_{pred}^{truth} = Pr(Class_i) * IOU_{pred}^{truth} \quad (3.11)$$

Detection network is made composed of 24 convolutional layers which are followed by two fully-connected layers. The features space from preceding layers is reduced by alternating 1x1 convolutional layers. On the ImageNet classification problem, we pretrain the convolutional layers at half the resolution (224x224 input picture) and then double the resolution for detection. Our network's ultimate prediction output is the 7x7x30 tensor.

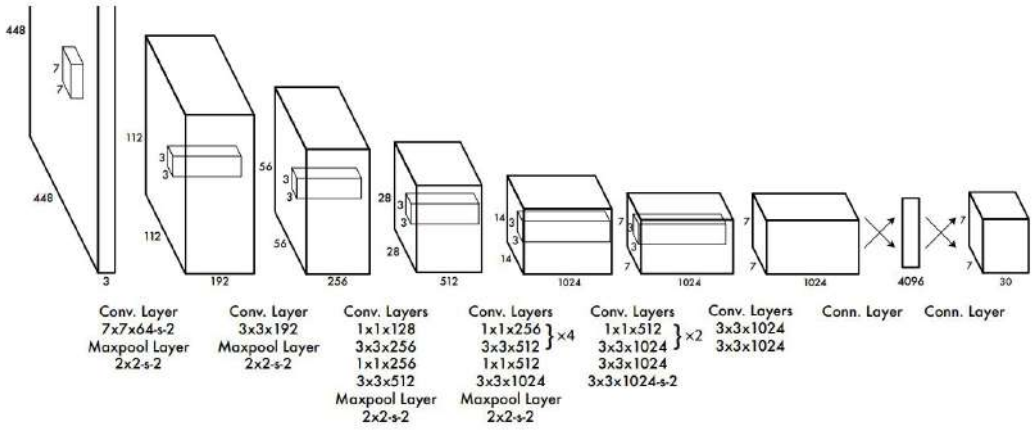

Figure 3.27: Network Design

For the final layer, it is applied a linear activation function, and all other layers utilize the leaky rectified linear activation:

$$\phi = \begin{cases} x, & \text{if } x > 0 \\ 0.1x, & \text{otherwise} \end{cases} \quad (3.12)$$

Optimized for sum-squared error in the output of the model. Sum-squared error used because it is easy to optimize, however it does not perfectly align with the goal which is maximizing average precision. This causes those cells' "confidence" values to drop to zero, frequently overpowering the gradient from cells that do contain objects. This can cause model instability and early training divergence. To address this, the loss raised from bounding box coordinate

predictions while decreasing the loss from confidence predictions for boxes that dont contain objects.This is accomplished by utilizing two parameters, λ_{coord}, and λ_{noobj}. They set $\lambda_{coord} = 5$, and $\lambda_{noobj} = .5$.

Sum-squared error weights errors in large and small boxes equally. Tiny variations in large boxes should matter less than small deviations in small boxes, according to our error measure. To remedy this, we estimate the square root of the bounding box width and height rather than the width and height directly.

Per grid cell, YOLO predicts several bounding boxes. We only want one bounding box predictor to be assigned for each item during training. One predictor is responsible for predicting an object depending on whose prediction has the highest current IOU with the ground truth. This leads to specialization among the bounding box predictors. Each predictor increases in predicting specific sizes, aspect ratios, or object classes, increasing overall recall.

During training multi-part loss function optimized:

$$
\begin{aligned}
&\lambda_{\textbf{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\
&+ \lambda_{\textbf{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] \\
&+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2 \\
&+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2 \\
&+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2
\end{aligned}
$$

Figure 3.28: Loss Function

1_i^{obj} stands for if the object is found in cell i and 1_{ij}^{obj} stands for jth bounding box predictor in cell i is responsible for this prediction.

It should be noted that the loss function penalizes classification error only if an object is found in that grid cell. It additionally penalizes bounding box coordinate error only if the predictor is in charge of the ground truth box.

YOLO is driven by sample artwork and natural pictures. It is pretty much correct, even if it misidentifies one person as an airplane.

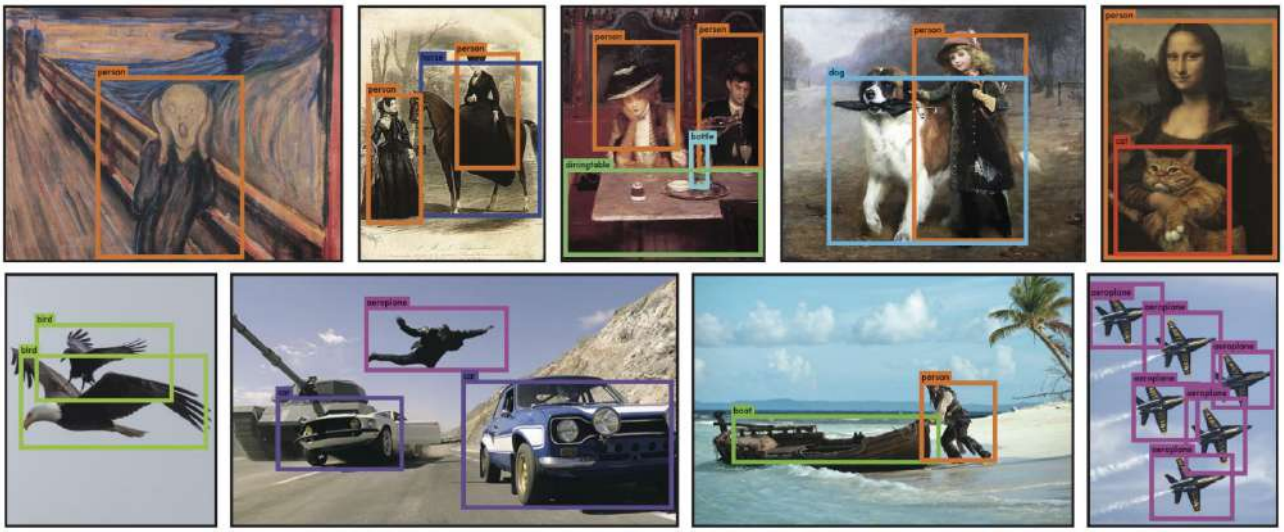

Figure 3.29: Results

Unlike classifier-based systems, YOLO is trained on a loss function that directly correlates to detection performance, and the entire model is learned at the same time. YOLO also generalizes effectively to new domains, making it perfect for demanding applications rapid, robust object detection. It weights localization error and classification error equally, which may not be optimum. Additionally, in every image many grid cells do not contain any object.

3.4.2 YOLOv2

YOLOv2 published as a joint endevor by Joseph Redmon the original author of YOLO and Ali Farhadi. Together they published YOLO9000:Better, Faster, Stronger on 25 December 2016.

YOLO9000 is presented a cutting-edge, real-time object identification system capable of detecting over 9000 item types. Firstly, suggested many innovative

and prior work-based enhancements to the YOLO detection approach. The improved model, YOLOv2, performs admirably on classic detection tasks such as PASCAL VOC and COCO. The same YOLOv2 model may operate at multiple sizes thanks to a breakthrough, multi-scale training strategy, providing a easy compromise between speed and accuracy. At 67 FPS, YOLOv2 gets 76.8 mAP on VOC 2007. At 40 FPS, YOLOv2 gets 78.6 mAP, outperforming state-of-the-art methods like Faster RCNN with ResNet and SSD while still running much faster. Finally, we present a strategy for training on object detection and classification simultaneously. Finally, a strategy presented for training on object identification and categorization simultaneously. This strategy, trained YOLO9000 on both the COCO detection and ImageNet classification datasets at the same time. The collaborative training enables YOLO9000 to predict detections for object classes that lack labelled detection data. On the ImageNet detection task, the technique is validated. Despite only having detection data for 44 of the 200 classes, the YOLO9000 achieves 19.7 mAP on the ImageNet detection validation set. YOLO9000 receives 16.0 mAP on the 156 classes that are not in COCO. YOLO, on the other hand, can detect more than 200 classes and predicts detections for over 9000 distinct object categories. It continues to operate in real-time.

YOLOv2 is cutting-edge and outperforms other detection methods over a wide range of detection datasets. It may also be run at different image sizes to give a smooth trade-off between speed and accuracy. YOLO9000 is a real-time framework for detecting over 9000 object types by optimizing detection and classification simultaneously. WordTree utilized to aggregate data from many sources, and our combined optimization strategy to train on ImageNet and COCO at the same time. The YOLO9000 represents a significant step forward in addressing the dataset size gap between detection and classification.

3.4.3 YOLOv3

YOLOv3 released on 8 April, 2018. YOLOv3 improved on the YOLOv2 paper and both of the original authors contributed, Joseph Redmon and Ali Farhadi. They published YOLOv3: An Incremental Improvement together.

YOLOv3 is an open-source, cutting-edge image recognition model. it is useful to detect the custom objects. Roboflow provided implementations in both

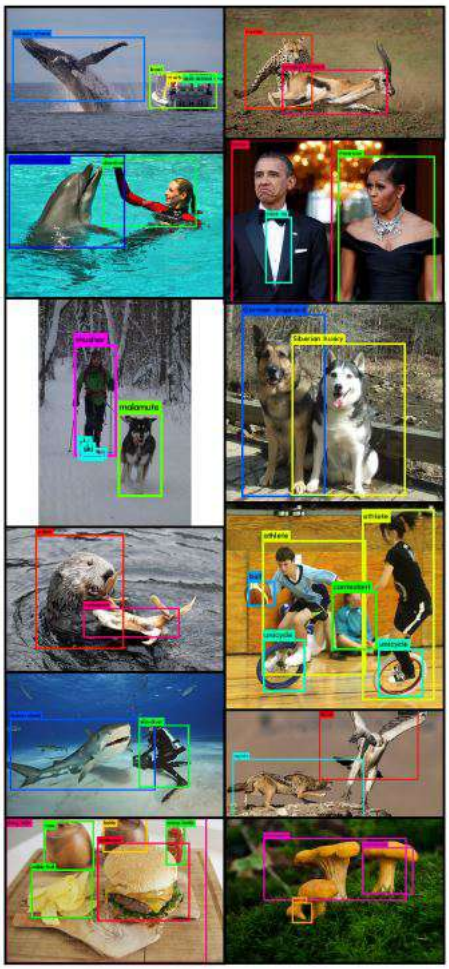

Figure 3.30: YOLO9000[19]

Pytorch and Keras frameworks. YOLOv3 has comparably fast inference times, with each inference taking about 30ms. It takes around 270 megabytes to store the approximately 65 million parameter model. There are various variants of YOLOv3 that can be utilized on Rasberry Pi, such as Tiny-YOLOv3.

Google ultimately published TensorFlow 2.0.0 at the end of September. This is a fantastic achievement for Tensorflow. Nonetheless, a new design does not always imply less suffering for engineers. Yolov3 can be implement in PyTorch, Tensorflow or even MXNet. TensorFlow 2 officially made eager mode a first-tier citizen. Simply said, instead than utilizing TensorFlow-specific APIs to calculate in a graph, you may now execute the graph in dynamic mode using native Python code. There will be no more graph compilation, and debugging and control flow will be considerably easier. Also, the Keras model isn't very

flexible, and the custom training loop is still in its early stages. As a result, the ideal technique for writing YOLO v3 in TF 2 is to start with a simple functional template and progressively add more logic to it. By doing so, it can fail early and fix the bug before it hides too deeply in a giant nested graph.

Making use of this model in production raises the issue of determining the production environment. For example, executing the model in a mobile app, over a remote server, or perhaps on a Raspberry Pi The best technique to store and convert your model's format depends on how it is wanted to utilize. Consider converting to TFLite (for Android and iPhone), converting to CoreML (for iPhone applications), converting for usage on a remote server, or deploying to a Raspberry Pi as next steps dependent on the situation.

YOLOv3 provided the first contribution by framing the object detection problem as a two-step task of first identifying a bounding box as regression problem and then determining the class of that object as classification problem.

Figure 3.31: Two-step problem

Joseph Redmon's YOLOv3, or You Only Look Once, model architecture will be used. This model is a one-shot learner, which means that each image only goes through the network once to generate a prediction, allowing the architecture to be extremely performant, predicting against video streams at up to 60 frames per second. YOLO is essentially a convolutional neural network (CNN) that separates an image into subcomponents and performs convolutions on each of those subcomponents before pooling the results to make a prediction.

YOLOv3 is a lightning-fast model, with inference speeds 100-1000x faster than R-CNN. YOLOv3 was compared to models such as RetinaNet-50 and Retina-Net-101 when it was first introduced. It performed exceptionally well on the COCO dataset in terms of detection speed, inference time, and model size. Some of the results comparing YOLOv3 to traditional models are shown below.

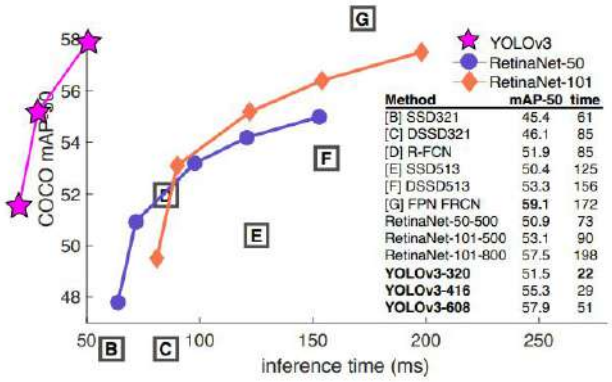

Figure 3.32: Displaying speed/accuracy tradeoff on the mAP at .5 IOU metric. YOLOv3 is good because its very high and far to the left[20]

YOLO v5 in PyTorch training gets much better results than YOLO v3.

YOLO v3 is a masterwork in the rising era of artificial intelligence, as well as a great summation of Convolution Neural Network techniques and tricks in the 2010s. Although there are numerous turn-key solutions available to make the process of creating a detector easier, having hands-on experience coding such complicated detectors is a fantastic learning opportunity for machine learning engineers because simply reading the paper is insufficient.

3.4.4 YOLOv4

With the release of YOLOv4 on April 23, 2020, the real-time object detection space continue to be active and advance. In terms of inference speed, YOLOv4 performs significantly better than other object identification models.

In essence, YOLOv4 is a collection of tiny innovative improvements to well-established computer vision algorithms. The important contribution is to understand how all of these methods may be integrated to effectively and efficiently supplemented for object detection.

Realtime is particularly fundamental for object detection models that rely on video feeds, such as self-driving cars. The other advantage of real-time object detection models is that they are compact and simple for all developers.

Real-time detection is highlighted in YOLOv4 and training is carried out on a single GPU. The goal of the authors is to make it simple for vision engineers and programmers to use their YOLOv4 framework in custom areas.

Each object detector has an image as input and compresses features through the backbone of a convolutional neural network. In image classification, these backbones constitute the network's core nodes, and predictions can be based on them. When detecting objects, the convolutional backbone's feature layers must be mixed and held up against one another in order to generate multiple bounding boxes around pictures while also classifying them. The neck is where the layers of the backbone combine.

Typically, an object detector's backbone network receives pretraining on ImageNet categorization. Pretraining refers to the network's weights being updated for the new task of object detection even when they have already been trained to recognize important features in an image.For the YOLOv4 object detector, the authors took into account the following backbones: CSPResNext50, CSPDarknet53, EfficientNet-B3.

DenseNet serves as the foundation for both the CSPResNext50 and the CSPDarknet53. DenseNet was created to connect layers in convolutional neural networks with the following goals in mind: to improve feature propagation, encourage the network to reuse features, and decrease the number of network parameters; to address the vanishing gradient problem (it is difficult to backprop loss signals through a very deep network). Google Brain created EfficientNet particularly to research the scale issue with convolutional neural networks. When scaling up the CNN, it has a variety of options, including input size, width scaling, depth scaling, and scaling all of the aforementioned options. According to the EfficientNet study, all of these have an ideal location, which they locate through search.

The final YOLOv4 network implements CSPDarknet53 for the backbone network based on their intuition and experimental findings meaning that a lot of experimental data.

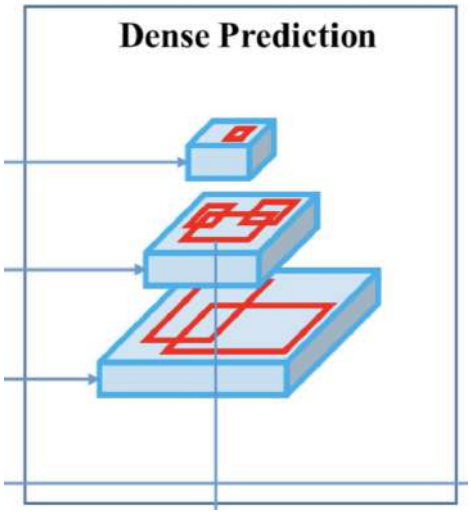

Figure 3.33: Dense Prediction[3]

For the network's feature aggregation, YOLOv4 selects PANet. Since NAS-FPN and BiFPN were built concurrently, they don't write much about the reasoning behind this choice, and this is perhaps an area for further study. Moreover, In order to broaden the receptive area and isolate the most crucial features from the backbone, YOLOv4 adds an SPP block following CSPDarknet53.

With three levels of detection granularity and anchor-based detection stages, YOLOv4 deploys the same YOLO head as YOLOv3.

Bag of Freebies The "Bag of Freebies" used by YOLOv4 boost network performance without increasing production-related inference time. The majority of the freebies in the bag are related to data augmentation. In computer vision, using data augmentation is essential, and we strongly advise doing it to obtain the best performance out of the models.

By using data augmentation, the creators of YOLOv4 were able to expand the size of their training set and expose the model to semantic circumstances that it would not have otherwise observed.

The computer vision community was necessarily aware of many of these techniques; YOLOv4 is only confirming their effectiveness. By combining four photos into one mosaic, the new contribution teaches the model to detect smaller

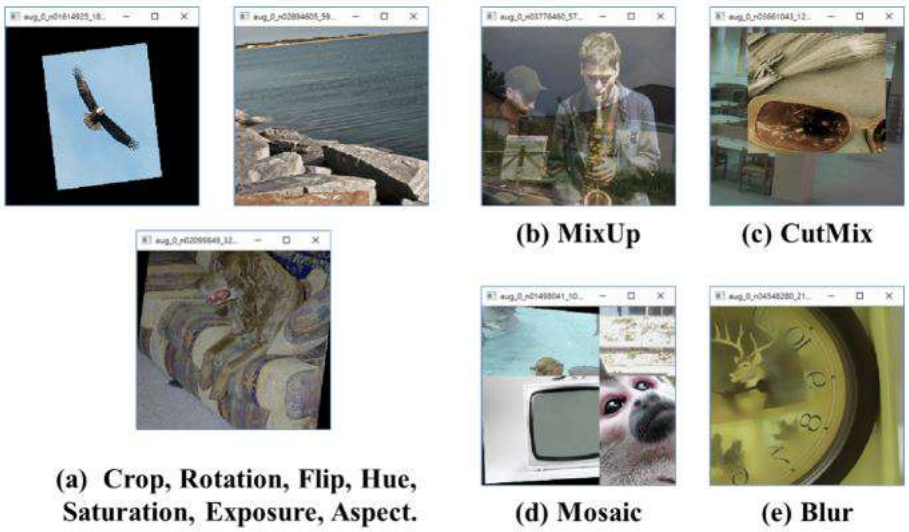

Figure 3.34: Bag of Freebies

objects and pay less attention to surroundings that are not immediately around the object.

Figure 3.35: Freebies

Self-Adversarial Training (SAT) is another original data augmentation contribution made by the authors. The goal of SAT is to determine the portion of the image on which the network depends the most during training. Then, the image is edited to hide this area, pushing the network to generalize to new features that might aid in detection.

A another freebie for editing the loss function is CIoU loss. The way the predicted bounding box overlaps with the ground truth bounding box is the basis for the CIoU loss used by the YOLOv4 authors. Basically, it isn't enough to only consider the overlap; if there isn't any, you need also consider how closely the predicted box was to the ground truth box and urge the network to move it closer to the latter. Of course, a lot of that involves mathematical engineering.

Bag of Specials The "Bag of Specials" techniques used by YOLOv4 ăbecause they only slightly increase inference time but greatly improve performance, making them worthwhile.

The authors test out different activation functions. Features are changed as they go through the network through activation functions. It might be challenging to convince the network to push feature creations toward their optimal point when using traditional activation functions like ReLU. Therefore, research has been conducted to develop features that very slightly enhance this process. A signal-pushing function called Mish is activated to the left and right.

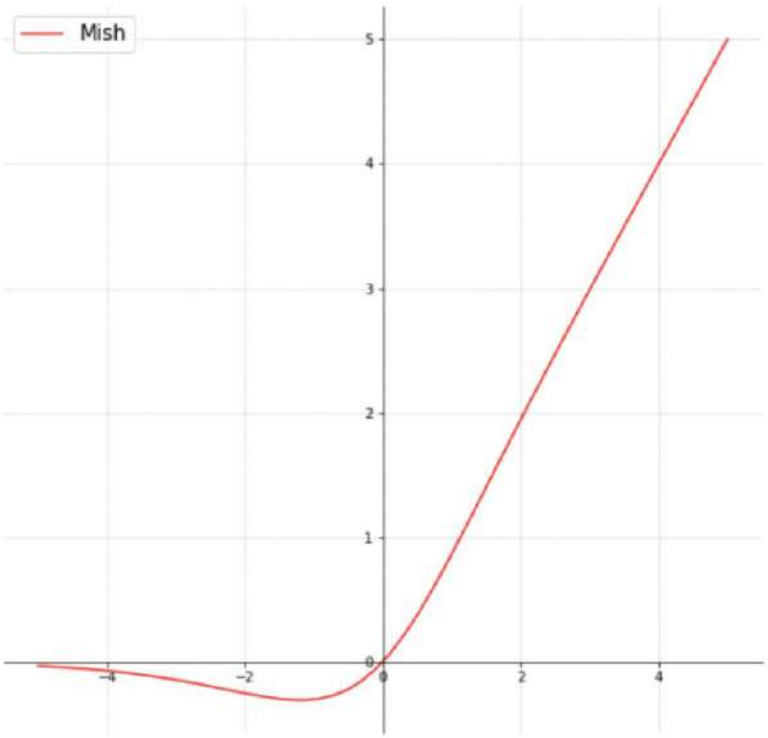

Figure 3.36: Mish Activation Function[11]

To distinguish predicted bounding boxes, the authors applied DIoU NMS. It would be helpful to quickly choose the best bounding box among those that the network may predict over a single object.

Figure 3.37: Using YOLOv3 and it needs some better NMS like YOLOv4 - this cannot be two kinds of jeeps at once (green label is Jeep TJ, brown label is Jeep YJ)

Cross mini-Batch Normalization (CmBN), which can be executed on any GPU, is the method the authors utilize for batch normalization. Multiple GPUs working together are required for many batch normalization approaches.

DropBlock regularization is used in YOLOv4. In DropBlock, sections of the image are hidden from the first layer. DropBlock is a method for making forcing the network to learn features that it may not otherwise rely upon. You may picture a dog whose head is tucked behind under a bush as an illustration. The network need to be able to recognize the dog's body in addition to its head.

The techniques in YOLOv4 were thoroughly proved out via experimentation on COCO dataset. The 80 object classes in COCO are intended to cover a wide range of object detection situations that a detector would run across in the wild. The 80 object classes in COCO are intended to cover a wide range of object detection situations that a detector would run across in the wild.

On the methods used in the paper, YOLOv4 conducts a thorough ablation study. In an ablation research, additions are removed one at a time to demonstrate which additions are better for the network.

Figure 3.38: Dog[21])

In conclusion, YOLOv4 is a brief summary of a broad range of computer vision algorithms for object identification. The greatest real-time object detector in the game is made from these tested and improved approaches, and it is lightweight and easy to use.

3.4.5 YOLOv5

Glenn Jocher from Ultralytics company released a YOLOv5 repository however he did not publish a paper. Above, under Yolov5 subsection the documentation is given based on this repository. Below, the comparison of the Yolov5 with some other models is found (Based on COCO dataset).

COCO dataset is the Microsoft Common Objects in Context dataset contains over 2 million images in 80 classes. The dataset has ranging from "person" to "handbag" to "sink". MS COCO is a standard dataset used to benchmark different models to compare their performance. The annotation method of COCO dataset is use also for other dataset.

However, using COCO dataset for every project would not be convenient. The range of too wide and maybe classification would be out of the scope. There would be too much detection per every frame and this would be confusing. For example, it is not needed to detect an ice cream for a self driving car.

If the Yolov4 and the Yolov5 compared, both models appear to max out their performance on the sample dataset, hence reported similarly accurate perfor-

mance. YOLOv5 trains faster on the sample task, and the batch inference which the implementation utilizes by default delivers real-time results. While YOLOv4 trains slowly, its performance may be adjusted to reach better FPS. Because YOLOv5 is implemented in PyTorch and YOLOv4 is implemented in Darknet, YOLOv5 may be simpler to bring to production, although YOLOv4 is where top-accuracy research may continue. For a computer vision engineer seeking the cutting-edge and not frightened of a bit more customized configuration, YOLOv4 in Darknet remains the most accurate. Moreover, YOLOv5 is a superior alternative for a developer looking to quickly apply near real-time object detection into a project.

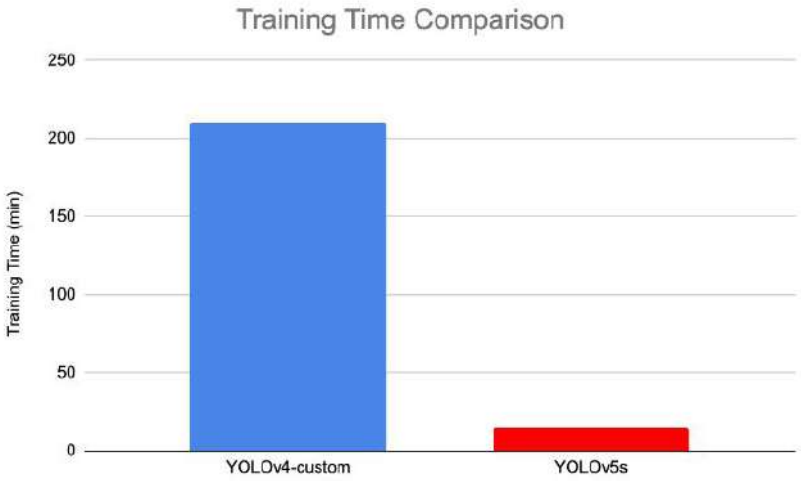

Figure 3.39: Training Time Comparison Example

Traning time comparison of Yolov5 and Yolov4 is explained with an example. The training length in YOLOv4 Darknet is determined by the number of iterations 'max batches' and not epochs. The repository recommends 2000 x num classes for custom objects. On the sample dataset, YOLOv4 Darknet takes a massive 14 hours with this configuration. However, it reaches maximum validation evaluation much sooner; it observed maximum validation evaluation at 1300 iterations, which took roughly 3.5 hours to complete. On the other hand, YOLOv5s, trained on 200 epochs in 14.46 minutes.

Examined the maximum validation mAP @0.5 metric for both networks. They perform similarly in this measure. This does not indicate the performance of

the networks on the COCO dataset. Based on this unique custom dataset, both networks are probably reaching peak performance for this specific custom task.

Figure 3.40: On the task, mAP was similar between the two models since both models reached their maximum.

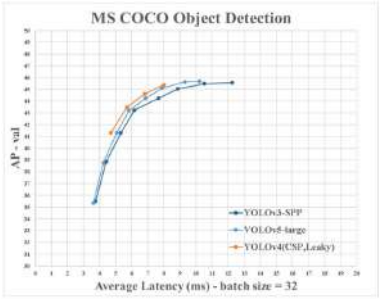

Figure 3.41: mAP on the COCO benchmark

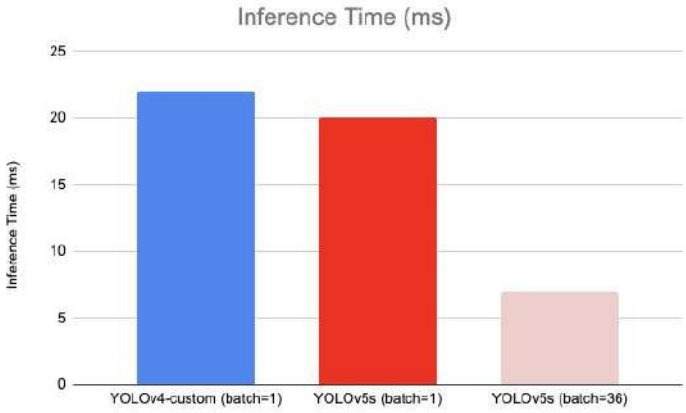

Figure 3.42: Inference times with various batch sizes.

The YOLOv4 and the YOLOv5s inference time compared, YOLOv4 inferences take 22 ms and YOLOv5s inferences take 20 ms on single images that is batch size of 1. When YOLOv5s infers in batch, it infers in 7 ms (140 FPS). Total inference time is divided by the number of images processed in the output. Because the Ultralytics YOLOv5 implementation defaults to this settings, if it is implemented from the repository, batch inference will run at 140 FPS by default. Instead, there is no information available about YOLOv4 to do batch inference.

Roboflow is dedicated to enabling every developer, regardless of domain, to solve issues with computer vision such as autonomous driving, COVID-19 chest scan interpreters, sushi detectors, airplane part maintenance identifiers, and so much more.

3.5 Applications of YOLO

The following fields could use the YOLO algorithm:

Autonomous driving: In autonomous vehicles, the YOLO algorithm can be used to find nearby objects like other cars, pedestrians, and parking signals. Since there is no human driver driving the automobile, object detection is done in autonomous vehicles to prevent collisions.

Wildlife: This method is used to find different kinds of species in forests. Journalists and wildlife rangers both utilize this form of detection to detect animals in still images and videos, both recorded and live. Giraffes, elephants, and bears are a few of the creatures that can be spotted.

Security: In order to impose security in a location, YOLO can also be implemented in security systems. Assume that a particular place has security restrictions prohibiting individuals from entering there. The YOLO algorithm will detect anyone who enters the restricted area, prompting the security staff to take further action.

3.6 Model Selection

YOLOv5 model has wide variety. It is scalable based on training and validation dataset. However, it has speed-performance trade off. It gives the freedom to choose between faster or better perform models.

Larger models like YOLOv5x and YOLOv5x6 will produce better results in nearly all cases, but have more parameters, require more CUDA memory to train, and are slower to run. For mobile deployments we recommend YOLOv5s/m, for cloud deployments we recommend YOLOv5l/x.

Pretrained model checkpoints with the COCO dataset shown on the Figure 3.10 in order to compare.

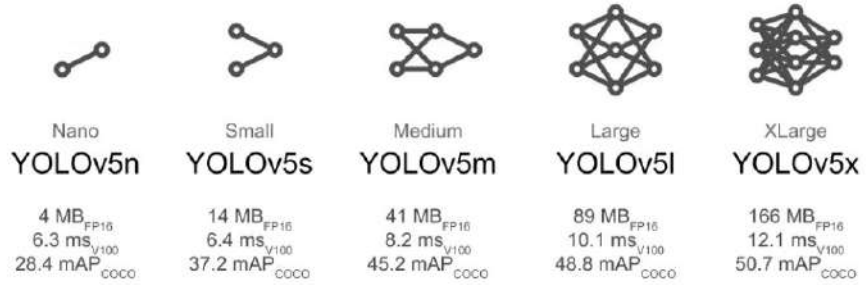

Figure 3.43: YOLO Model Selection

Model	size (pixels)	mAP^{val} 0.5:0.95	mAP^{val} 0.5	Speed CPU b1 (ms)	Speed V100 b1 (ms)	Speed V100 b32 (ms)	params (M)	FLOPs @640 (B)
YOLOv5n	640	28.0	45.7	**45**	**6.3**	**0.6**	1.9	**4.5**
YOLOv5s	640	37.4	56.8	98	6.4	0.9	7.2	16.5
YOLOv5m	640	45.4	64.1	224	8.2	1.7	21.2	49.0
YOLOv5l	640	49.0	67.3	430	10.1	2.7	46.5	109.1
YOLOv5x	640	50.7	68.9	766	12.1	4.8	86.7	205.7

Figure 3.44: Pretrained Checkpoints

Starting from Pretrained weights, recommended for small to medium sized datasets such as VOC, VisDrone, GlobalWheat. Pass the name of the model to the –weights argument. Models download automatically from the latest YOLOv5 release.

```
python train.py --data custom.yaml --weights yolov5s.pt
                                             yolov5m.pt
                                             yolov5l.pt
                                             yolov5x.pt
                                             custom_pretrained.pt
```

Code 3.3: -weights argument

On the other hand, starting from scratch is recommended for the large datasets such as COCO, Objects365, OIv6. Pass the model architecture yaml file which you are interested in, along with an empty –weights ” argument:

```
python train.py --data custom.yaml --weights '' --cfg yolov5s.yaml
                                                      yolov5m.yaml
                                                      yolov5l.yaml
```

```
4                                                    yolov5x.yaml
```

Code 3.4: –weights" –cfg argument

COCO trains at native resolution of –img 640, though due to the high amount of small objects in the dataset it can benefit from training at higher resolutions such as –img 1280. If there are many small objects then custom datasets will benefit from training at native or higher resolution. Best inference results are obtained at the same –img as the training was run at, for example if the training at –img 640, then the detect and test should also at –img 640.

3.7 Dataset To Train The Model

Cars dataset used in order to train the object detection model YOLOv5. Dataset contains 479 images labeled with bounded boxes by Roboflow. Images resized 416x416. Detection labels separated in 4 classes. Car, person, cyclist and bus named with in order; 0, 1, 2, 3. Dataset separated in training 80% (382 images) and 20% validation (97 images).

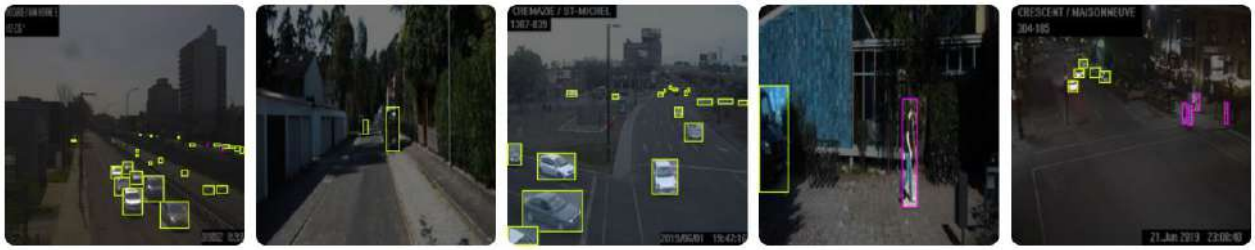

Figure 3.45: Cars Dataset With Bounded Boxes

In the dataset, it is important that images have to have variety. For real-world use cases, it is recommended images from varied times of day, seasons, weather, lighting, perspectives, sources and so on. Labels have to be consistent. Every instance of each class in all image must be annotated. Partial labeling is useless. Labels have to be accurate. Labels must properly surrounding each object. There must be no space between an object and its bounding box. There should be no objects lacking labels. View training batches in order to verify the labels appear correctly for the four of the classes. Background images are images that do not contain any objects and are introduced to a dataset to reduce FPs. To help reduce

FPs, proposed to use 0-10% background images. for reference, COCO dataset has 1000 background images, 1 percent of the total. Background images do not require labels.

Figure 3.46: Train Batch Example

Figure 3.47: Validation Batch Example

3.8 Training The Model

Yolov5s model trained with cars dataset in order to obtain best model for classification between car, person, cyclist and bus. Yolov5s is the small model and optimum for a dataset with 479 images. Hyperparameters set accordingly. The number of epochs to train is defined as 300 above. Image size is 640 and the batch size is 64.

```
RES_DIR = set_res_dir()
if TRAIN:
    !python train.py --data ../data.yaml --weights yolov5s.pt \
    --img 640 --epochs {EPOCHS} --batch-size 64 --name {RES_DIR}
```

Code 3.5: Train the image dataset

Default hyperparameter values indicated on the paper preferred. Values which is used in the training:

```
lr0=0.01, lrf=0.01, momentum=0.937, weight_decay=0.0005,
    warmup_epochs=3.0, warmup_momentum=0.8, warmup_bias_lr=0.1, box
    =0.05, cls=0.5, cls_pw=1.0, obj=1.0, obj_pw=1.0, iou_t=0.2,
    anchor_t=4.0, fl_gamma=0.0, hsv_h=0.015, hsv_s=0.7, hsv_v=0.4,
```

```
degrees=0.0, translate=0.1, scale=0.5, shear=0.0, perspective=0.0,
 flipud=0.0, fliplr=0.5, mosaic=1.0, mixup=0.0, copy_paste=0.0
```

Code 3.6: Hyperparameters train.py

2 weights obtained after training completed; last.pt and best.pt files. Last.pt are the weights from the last epoch of training. Best.pt are the best weights recorded during training. Best.pt obtained always between the 100th and 150th epochs. Therefore, no need to complete all the 300 epochs in order to gain time.

3.9 My Model

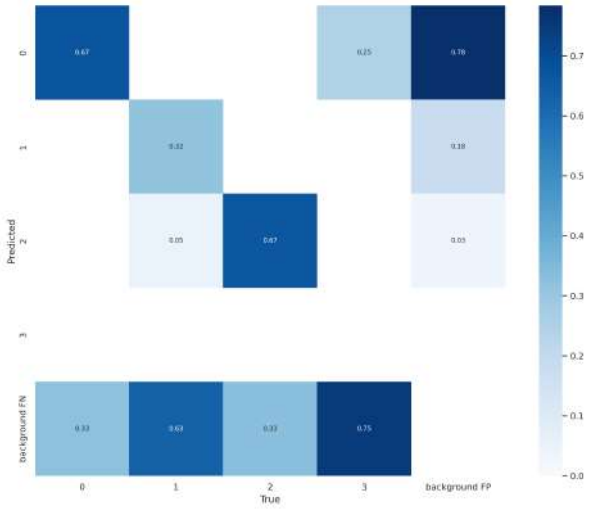

Figure 3.48: Confusion Matrix of best.pt

The training aims the model to learn images and labels. Convolutional neural networks (CNN) are used by the YOLO model to recognize objects in real time. The approach just needs one forward propagation through a neural network to detect objects. This signifies that a single algorithm run is being used to perform prediction all through the entire image. Various classes probabilities and bounding boxes are simultaneously predicted by CNN. The class probabilities of the detected images are provided by the object identification process in YOLO, which is carried out as a regression problem.

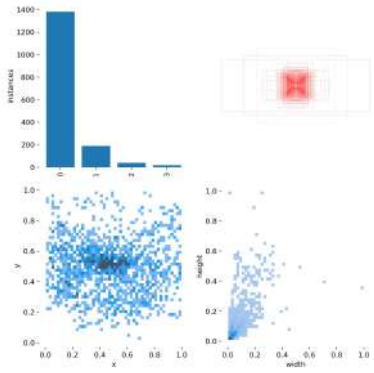

Figure 3.49: Instances

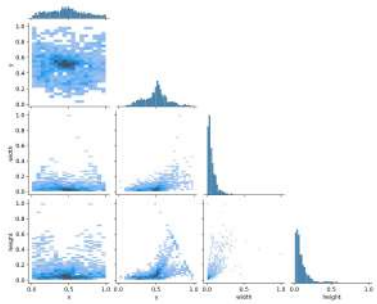

Figure 3.50: Label Correlogram

After the training, best model gained as the output. Best model will be used later in order to detect classes on the given source such as test dataset or webcam.

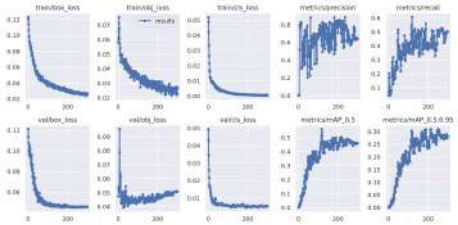

Figure 3.51: Results

Classification results both class by class and as average visualized based on the confusion matrix of the best.pt. It is possible to validate model accuracy by calculating precision, recall and f1 rates by using confusion matrix. PyTorch framework provides TensorBoard tools in order to visualize the model validation.These tools enable you to log PyTorch models and metrics into a directory for display within the TensorBoard UI after installing TensorBoard. For PyTorch models and tensors, embedding visualizations, pictures, histograms, and graphs are all supported.

CUDA is a bridge between the GPU and TensorFlow. It is NVIDIA's method of creating general purpose GPU optimized code. This is how we are able to use GPU devices originally designed for three dimensional games to accelerate neural networks.

```
Model summary: 213 layers, 7020913 parameters, 0 gradients, 15.8 GFLOPs
               Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95:
                 all         48        415      0.809      0.391      0.503      0.309
                   0         48        348      0.841      0.607      0.727      0.429
                   1         48         57      0.732      0.288      0.347      0.152
                   2         48          6      0.663      0.667       0.69      0.434
                   3         48          4          1          0      0.249      0.222
Results saved to runs/train/results_1
```

Figure 3.52: Results best.pt

As it is seen on the results best.pt, CNN consist of 213 layers with 7020913 trainable parameters. The average value of precision is 0,809, recall is 0,391 and mAP is 0,503. While the precision value is relatively high, recall value is relatively low. Interpretation of the results would be less detection, less error. In this case, visualizing the training images is required in order to be sure if the model has sufficient quantity of instances. For this specific circumstance avoiding errors such as FPs can cause the issue of miss the detections.

It is seen on the results that the precision, recall and mAP values are strange for the third class which is bus. It is because even though there was no enough instances, bus is defined as a class. By changing the training dataset or adding more instances to dataset it is possible to have better results for buses. However, bus was not the focus for this project and in the test dataset there was a few buses but sometimes minibus. Thus, bus class is ignored in the work however there was some misclassification error because of the buses.

4 training image visualized randomly. On the images, 0 represents car's class and 1 represents person's class. Bounding boxes placed without any error and classification seems accurate even if class 2 which is cyclist and class 3 which is bus are not visible on the images. Some other results also visualized:

F1 curve reaches 0.45 at 0.53 and precision reaches 1 at 0.97 for all classes.

3.10 Test Dataset

A dataset provided from The KITTI Vision Benchmark Suite used as the test dataset. The KITTI Vision Benchmark Suite is a project of Karlsruhe Institute of Technology and Toyota Technological Institute at Chicago. Novel challenging real-world computer vision benchmarks developed with the help of the autonomous driving platform Annieway. Adapted a typical station wagon with

Figure 3.53: Training Images

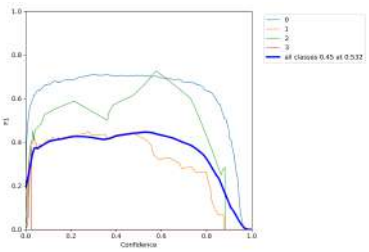

Figure 3.54: F1

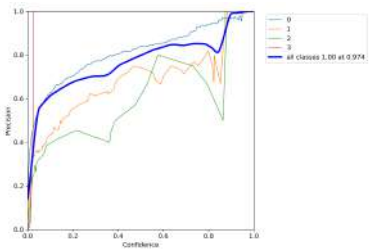

Figure 3.55: Precision

two high-resolution color and grayscale video cameras for this purpose. A Velodyne laser scanner and a GPS localization system give an exact ground truth. The tasks of the interest of this project are stereo, optical flow, visual odometry, 3D object detection and 3D tracking.

The datasets[7] are gathered while driving through rural areas, on highways, around the middle-sized German city of Karlsruhe. For each image, up to 15 autos and 30 pedestrians are observable. The datasets were distributed in accordance with the tasks of interests. By giving the community real-world benchmarks with fresh challenges, the objective is to lessen this bias and supplement the current benchmarks.

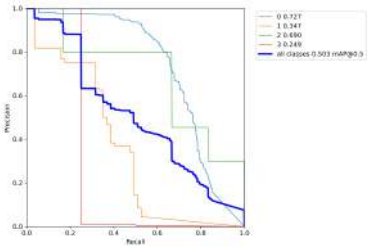

Figure 3.56: Precision Recall

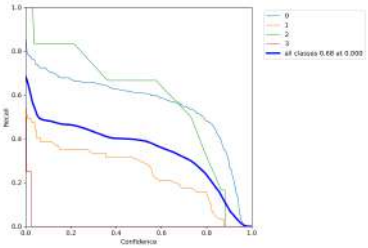

Figure 3.57: Recall

Figure 3.58: Station Wagon

KITTI Raw dataset used as the test dataset. Raw data was distrubuted between categories. The test made with a dataset from Residential category. The data which is collected on 26th of September, 2011, 35th drive. Dataset lenght is 137 frames, 00:13 minutes long.Image resolution is 1392 x 512 pixels. Dataset consists of 23 cars, 6 vans, 2 trucks, 2 pedestrians, 0 sitters, 1 cyclists, 0 trams, 3 misc labels.

3.11 Inference

The inference score is a number between 0 and 1 that indicates confidence that the object was genuinely detected. The closer the number is to 1, the more confident the model is. Inference run on variability of sources such as images, videos, directories, streams, webcam, etc.

```
#Usage - sources:
$ python detect.py --weights yolov5s.pt --source 0               # webcam
                                                 img.jpg # image
                                                 vid.mp4 # video
                                                 path/   # directory
```

Figure 3.59: An image from the mentioned dataset

```
6                                                   path/*.jpg # glob
7                                                   'https://youtu.be/
  Zgi9g1ksQHc'  # YouTube
8                                                   'rtsp://example.com/
  media.mp4'  # RTSP, RTMP, HTTP stream
```

Code 3.7: Add inference to source data

Inference run on variability of formats such as a customized model, PyTorch, dnn, TensorFlow etc.

```
1 Usage - formats:
2 $ python path/to/detect.py --weights yolov5s.pt # PyTorch
3                                       yolov5s.torchscript  #
  TorchScript
4                                       yolov5s.onnx # ONNX Runtime or
  OpenCV DNN with --dnn
5                                       yolov5s.xml # OpenVINO
6                                       yolov5s.engine # TensorRT
7                                       yolov5s.mlmodel # CoreML (macOS-
  only)
8                                       yolov5s_saved_model # TensorFlow
  SavedModel
9                                       yolov5s.pb # TensorFlow GraphDef
10                                      yolov5s.tflite # TensorFlow Lite
11                                      yolov5s_edgetpu.tflite #
  TensorFlow Edge TPU
```

Code 3.8: Add inference to source data

In this case, inference run on KITTI dataset which is saved in the directory and customized model in Pytorch format. Dataset explained before. Customized

model is the best model that is obtained as the output weight of the training before.

```
# Inference on images.
!python detect.py --weights runs/train/{RES_DIR}/weights/best.pt \
--source {data_path} --name {INFER_DIR}
```

Code 3.9: Call detect.py with best model

Run detect.py and it detect the defined classes on the given dataset. Place bounded boxes and write the class and inference score over it.

Figure 3.60

Figure 3.61

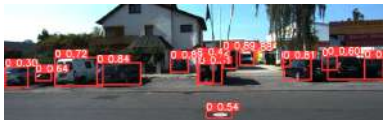

Figure 3.62

Figure 3.63

Figure 3.64

Figure 3.65

Here, it is seen that the detections are good on real-time applications same as the recall and precision results.

3.12 Tests

Mentioned before that KITTI raw dataset are used as the test dataset. Raw dataset divided in variety of categories and drives based on the drive date and the number. In order to observe how the results change on confident scores, the best model tested with different categories, drives of the dataset and learning rate. And one more and test has been done by using greyscale images while the instances are still color images.

Figure 3.66

Figure 3.67

One important thing worth to point out, the images which are used as test results are choosen randomly but in particular, they are the images which has error. This dataset has 130 images and other than the ones which is demonstrated here do not have significant errors. If considered the fact that even the best detector which is human sight/eye might be sometimes incorrect based on the circumstances, the yolo model is doing a very good job.

3.12.1 Test With Given Greyscale Dataset

The test made with a KITTI raw dataset from the Residential category. The data which is collected on 26th of September, 2011, 35th drive. Dataset lenght is 137 frames, 00:13 minutes long. Image resolution is 1392 x 512 pixels. The dataset provides also the greyscale images together with the color images in the dataset. Here, the greyscale images used as the test images:

```
TRAIN = True
# Number of epochs to train for.
EPOCHS = 300
```

Code 3.10: 300 Epoch numbers to train the model

Figure 3.68

Figure 3.69

Figure 3.70

Figure 3.71

The best model obtained after the training with Roboflow dataset and 150 epochs. Learning rate distribution of the training dataset was 70% Training, 20%

Validation, 10% Test. This was the default. However this images reserved for testing were never used. Instead of them, the aforementioned dataset greyscale images were used in order to test the best model. Here are some demonstrations of detections and confidence scores from this greyscale test images:

Figure 3.72

Figure 3.73

On this images it is possible to see a lot of FPs. Also, a pedestrian detected and classified as a car. And a wheel of a car classified as a pedestrian. And some cars classified as bus. And a cyclist detected and classified as a car. There are possible to see way more misclassification error than the color images as test.

The main reason of this error should be the fact that model trained by using color images and instances were all color. For this reason, model having difficulty for detecting objects and classify them correctly. And also, model makes false detections because in the training dataset, there was no greyscale image.

Figure 3.74

Figure 3.75

Figure 3.76

Figure 3.77

There are other tests have been done. However, while some tests made with default learning rate, others made with 0.2. Mostly, experienced how the results change with variety of the epoch numbers. For better understanding and making it possible to compare, the same dataset used for the tests as it will be indicated below. Other than the first test with greyscale dataset above, all the tests done

with color images. Because it is obvious that, it would not be possible to reach a sufficient result with the use of greyscale images. It would be convenient to separate the tests based on their learning rate.

3.12.2 Learning Rate: Default

Learning rate distribution of the training dataset was 70% Training, 20% Validation, 10% Test. Even if the test dataset divided here, it was not used. In this case, test dataset size is not important. It does not have any effect on the model since it was not used for training. This is a real-time application. Test dataset size could be also huge. Test are made by changing the epoch numbers as 50, 150, 300 and 600 and comparing the results. Moreover, test has been done by changing the scale of the YOLOv5 model; small and medium scale models tested based on the number of instances.

Test with 50 Epochs

The test made with a KITTI raw dataset from the Residential category. The data which is collected on 26th of September, 2011, 35th drive. Dataset lenght is 137 frames, 00:13 minutes long. Image resolution is 1392 x 512 pixels. As it is mentioned before dataset contains colour images as well. Colour images used this time. And epoch number set to 50 in order to train the model. And after that the results are obtained as it is seen below:

```
TRAIN = True
# Number of epochs to train for.
EPOCHS = 50
```

Code 3.11: 50 Epoch numbers to train the model

Figure 3.78

Figure 3.79

Even though this results are highly accurate, there are still some error. It is possible to find FPs on some images. But these are very rare. Beside having

good precision and recall results, this model has good confident score together with the classification

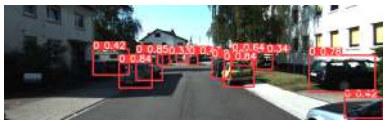

Figure 3.80

Figure 3.81

It is seen here that there FPs. A flower pot detected as a car or background has a fake detection and classified as a car. A car detected and classified as a cyclist. This list goes on and on.

Figure 3.82

Figure 3.83

There are some reasons to explain the reasons of these error and they are not always related to the model. The reasons can be the lighting especially, too dark or too bright, or the images look different then instances which model trained. For example, if the car class doesn't have a variety with car brands and models, then of course detecting and classifying them will create some error.

Figure 3.84

Figure 3.85

Anyway it is possible to see that this model was quite good. Pedestrians are detected and classified in a correct way with a good confident score as it is seen.

In particular, on the last image, even though the cyclist was visible only half, the model was able to detect it correctly and classified it as a cyclist. It might be said model works very well but it still can be improve by increasing the number of iterations. Thus, it has tested with 150 epochs:

Figure 3.86

Figure 3.87

Test with 150 Epochs

This test made with a KITTI raw dataset from the Residential category. The data which is collected on 26th of September, 2011, 35th drive. Dataset length is 137 frames, 00:13 minutes long. Image resolution is 1392 x 512 pixels. Colour images used. As it is mentioned before this is the same dataset as in the former test in order to make it possible to compare the results. And epoch number set to 150 in order to train the model and obtain a improved model. And after that the results are obtained as it is seen below:

Figure 3.88

Figure 3.89

```
TRAIN = True
# Number of epochs to train for.
EPOCHS = 150
```

Code 3.12: 150 Epoch numbers to train the model

150 Epochs mean that now it has more iteration to reach to optimal results. Here, it is already observed a satisfying result. And it is seen that the model has better results this time. First of all it can detect each pedestrian correctly without FPs or FNs and correctly classify.

Figure 3.90

Figure 3.91

Also, this model still can confuse when the pedestrian is not well seen, when it was half or in the shade.

Figure 3.92

Figure 3.93

The model can detect a bus but can not classify correctly on the some frames. It is seen that model classify the bus first as a car and then as a bus. The reason might be that in the first frame the bus was quite far from the source but when it came closer, the problem disappeared.

Figure 3.94

Figure 3.95

There are other misclassification errors. For example here, even though the images are half visible, a cyclist and a car are detected. However, the cyclist classified as a car while the car classified as a bus.

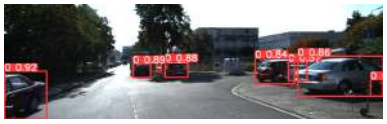

Figure 3.96

Figure 3.97

In order to overcome or at least to decrease the number of errors, it would be better to increase the number or epochs. Moreover, it is seen that when 150 epochs are complected, the model was continuing to improve itself. This means that with more epochs it is possible to obtain a more accurate model. In this case epoch number 300 implemented below:

Test with 300 Epochs

This test made with a KITTI raw dataset from the Residential category. The data which is collected on 26th of September, 2011, 35th drive. Dataset length is 137 frames, 00:13 minutes long. Image resolution is 1392 x 512 pixels. Colour images used. As it is mentioned before this is the same dataset as in the former

test in order to make it possible to compare the results. And epoch number set to 300 in order to train the model. And after that the results are obtained as it is seen below:

```
TRAIN = True
# Number of epochs to train for.
EPOCHS = 300
```

Code 3.13: 300 Epoch numbers to train the model

Figure 3.98

Figure 3.99

It is seen that even if the epochs set to 300, the best model obtained at 153rd epoch and trained stopped. Thus, the training terminates in a shorter time.

Figure 3.100

Figure 3.101

Figure 3.102

Figure 3.103

It is seen that best detection and classification results obtained between 150th and 300th iterations. And the script written in a way that when the improvement of the model stop, iterations will also stop in order to save time. Because even if with more iterations, the model will not go any better after a certain iteration.

There are not many error observed on the test dataset when best model applied. A van was detected as a car in the previous frame because it is too far but this detection is actually a FP. In the next frames, model doesn't detect the van anymore because camera is getting closer. A van can not be detected since no instance defined ever as the van. And a manhole detected as a car.

Figure 3.104

Figure 3.105

Another thing is mention to worth is that minibuses are detected and classified as bus which is not an error.

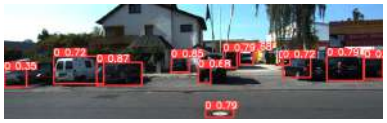

Figure 3.106

Figure 3.107

For this specific circumstances, the tests which have done to understand which scale is the best with the dataset, have done by setting the epoch numbers to 300 as they can be seen below. The mAP value at .5 is 0.595 for all classes. In this point, the MS COCO dataset results used as reference and after the comparison, the results approved that they are good.

```
     Epoch   gpu_mem       box       obj       cls    labels  img_size
   253/299     4.64G   0.02949   0.03125 0.0007873        86       640: 100% 11/11 [00:03<00:00,  3.27it/s]
                Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100% 2/2 [00:00<00:00,  2.19it/s]
                  all         48        434      0.603      0.437      0.498      0.313
Stopping training early as no improvement observed in last 100 epochs. Best results observed at epoch 153, best model saved as best.pt.
To update EarlyStopping(patience=100) pass a new patience value, i.e. `python train.py --patience 300` or use `--patience 0` to disable EarlyStopping.

254 epochs completed in 0.366 hours.
Optimizer stripped from runs/train/results_1/weights/last.pt, 14.5MB
Optimizer stripped from runs/train/results_1/weights/best.pt, 14.5MB

Validating runs/train/results_1/weights/best.pt...
Fusing layers...
Model summary: 213 layers, 7020913 parameters, 0 gradients, 15.8 GFLOPs
                Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100% 2/2 [00:01<00:00,  1.43it/s]
                  all         48        434      0.708      0.537      0.595      0.373
                    0         48        372      0.726      0.704      0.756      0.445
                    1         48         45      0.685      0.533      0.405      0.251
                    2         48         14      0.409      0.286      0.208      0.168
                    3         48          3          1      0.625       0.83      0.626
Results saved to runs/train/results_1
```

Figure 3.108: Results 300 Epochs

It have to be pointed out that this is YOLOv5 small model. But, the YOLO models are scaleable. In order to find the best model, it is better to test also the medium model. Based on the training dataset size and the number of the instances, either small or medium model should be perfect.

Test With 300 Epochs - Medium Model

This test made with a KITTI raw dataset from the Residential category. The data which is collected on 26th of September, 2011, 35th drive. Dataset length

is 137 frames, 00:13 minutes long. Image resolution is 1392 x 512 pixels. Colour images used. As it is mentioned before this is the same dataset as in the former test in order to make it possible to compare the results. And epoch number set to 300 in order to train the model. And after that the results are obtained as it is seen below:

```
TRAIN = True
# Number of epochs to train for.
EPOCHS = 300
```

Code 3.14: 300 Epoch numbers to train the model

Until now all the tests have been done by using YOLOv5s meaning that the small model. But using a size of a model is not mandatory and the training can be done with any size of the model based on the dataset and number of instances.

Figure 3.109

Figure 3.110

The medium model of YOLOv5 is tested here in order to understand it will give better results.

Figure 3.111

Figure 3.112

As it is seen on the images, all the pedestrians could not detected. If the medium model and small model compared, it is possible to see that confident scores of the classifications are slightly lower on the medium model. It is seen here more fake detections.

A van is detected first as a bus and then a car because of some differences between the frames.

Figure 3.113

Figure 3.114

Figure 3.115

Figure 3.116

As it is seen on the images, no improvement observed. This means that for this size of dataset the small model would be the optimal for the training.

Here, the mAP value is 0.606. MS COCO dataset always used as the reference dataset. It possible to see that the result values are also good. But it is important to look at also to precision and recall values. It is seen that in the training there was not many instances of bus but the recall value of this class is 0.667 while the precision is 0.835. This results consequences with a lot of fake detections and misclassifications as bus class.

And this explains the reason why there are many misclassification error as the bus on the images. In order to address this problem, more instances might be added to the model. But actually using the small model would be the most convenient approach.

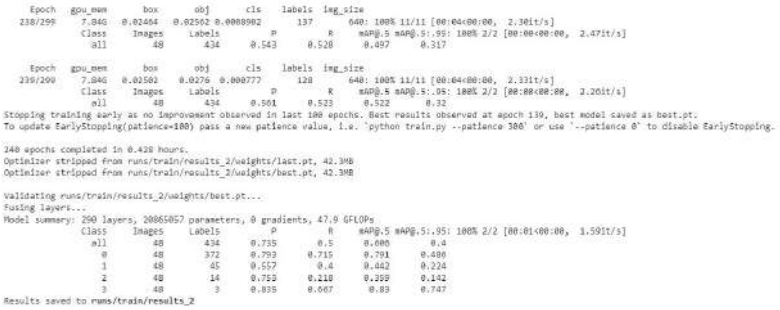

Figure 3.117: Results 300 Epochs - Medium Model

Figure 3.118

Figure 3.119

Test with 600 Epochs

This test by setting the epoch number to 600 has been done just after the test by setting the epoch number to 300. Because it is recommended to set the epoch number to 600 at the beginning for the first test with in particular big models of YOLOv5. Here, wanted to added even though the small model applied.

```
TRAIN = True
# Number of epochs to train for.
EPOCHS = 600
```

Code 3.15: 600 Epoch numbers to train the model

This test made with a KITTI raw dataset from the Residential category. The data which is collected on 26th of September, 2011, 35th drive. Dataset length is 137 frames, 00:13 minutes long. Image resolution is 1392 x 512 pixels. Colour images used. As it is mentioned before this is the same dataset as in the former test in order to make it possible to compare the results. And epoch number set to 600 in order to train the model. And after that the results are obtained as it is seen below:

Here it is seen that the epochs stops at the 102nd epoch. This is quite low comparing to others. And below the mAP value is 0.493 for all classes. It might be said that this is lower than the others. And also the recall values are low.

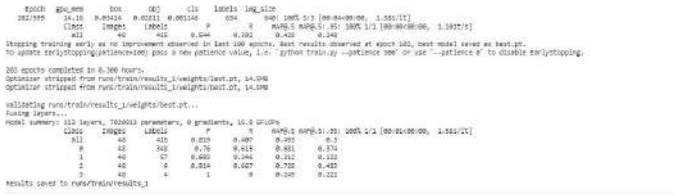

Figure 3.120: Results 600 Epochs

Figure 3.121

Figure 3.122

This test by setting the epoch number to 600 was an extra test, actually. Because it is known that it never reached nor to 300th iteration neither to 600th iteration. This one is only good to see the results and compare with the others. But at the end, it might be said that no significant difference is observed.

Figure 3.123

Figure 3.124

It is observed that there are many fake detections. For example, even if there was 2 pedestrian, they are detected two.

However, most of the detections are correct. Moreover, the minibus is classified as a bus, even though before it often classified as a car. And the cyclist detected and classified correctly even if it is half visible.

It is better to do many tests in order to understand which model would fit the best with the dataset. Until now the default division of the dataset which is mentioned before is used. Below, it is found a custom distribution (80% Training, 20% Validation) of the dataset with some other test:

Figure 3.125

Figure 3.126

3.12.3 Learning Rate: 80% Training, 20% Validation

Learning rate distribution of the training dataset was arranged custom. This is optional. Distribution is 80% Training, 20% Validation. Indicating the learning rate is important for training for better performance with training and validation distribution. However, the test distrubution is not necessary since other sources can be used. In real- time applications source can be the webcam data and in this case the test dataset would be huge.

Test with 200 Epochs

This test made with a KITTI raw dataset from the Residential category. The data which is collected on 26th of September, 2011, 35th drive. Dataset length is 137 frames, 00:13 minutes long. Image resolution is 1392 x 512 pixels. Colour images used. As it is mentioned before this is the same dataset as in the former test in order to make it possible to compare the results. And epoch number set to 200 in order to train the model because usually the best model obtained after 150th epoch. And after that the results are obtained as it is seen below:

```
200 epochs completed in 0.325 hours.
Optimizer stripped from runs/train/results_1/weights/last.pt, 14.5MB
Optimizer stripped from runs/train/results_1/weights/best.pt, 14.5MB

Validating runs/train/results_1/weights/best.pt...
Fusing layers...
Model summary: 213 layers, 7020913 parameters, 0 gradients, 15.8 GFLOPs
               Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100% 1/1 [00:01<00:00,  1.01s/it]
                 all         48        415      0.862      0.388      0.514      0.344
                   0         48        348      0.886      0.583      0.727      0.423
                   1         48         57      0.897      0.304       0.39      0.165
                   2         48          6      0.663      0.667      0.691      0.567
                   3         48          4          1          0      0.248      0.222
Results saved to runs/train/results_1
```

Figure 3.127: Results 200 Epochs

Figure 3.128

Figure 3.129

Here, the mAP value is .514 for all classes. Class by class the precision values higher then before. Thus, also on the images it is seen that the detections and classifications have way less error than before.

Figure 3.130

Figure 3.131

The pedestrians detected and classified correctly. Confidence scores sufficiently high. Fake detections are very rare.

Figure 3.132

Figure 3.133

If this one is compared with previous tests, on the result table, it is seen that recall values of the classes are relatively low. This helps the model in order to detect less but to be more precise meaning that less errors, less fake detections and less misclassification errors.

Figure 3.134

Figure 3.135

In this test, it is realized that, different than default dataset distribution, all the set epochs completed in this custom distribution of the dataset. The reason might be, the model has more instance in the training and the validation dataset. It is important to mention that the best result has just obtained, so far. However, the best model is not yet received even though the epochs completed. In this point, it might be a good approach to set more epoch and let the model to reach the best.

Another test has been done by setting the epoch numbers to 300:

Figure 3.136

Figure 3.137

Test with 300 Epochs

This test made with a KITTI raw dataset from the Residential category. The data which is collected on 26th of September, 2011, 35th drive. Dataset length is 137 frames, 00:13 minutes long. Image resolution is 1392 x 512 pixels. Colour images used. As it is mentioned before this is the same dataset as in the former test in order to make it possible to compare the results.

```
300 epochs completed in 0.469 hours.
Optimizer stripped from runs/train/results_1/weights/last.pt, 14.5MB
Optimizer stripped from runs/train/results_1/weights/best.pt, 14.5MB

Validating runs/train/results_1/weights/best.pt...
Fusing layers...
Model summary: 213 layers, 7020913 parameters, 0 gradients, 15.8 GFLOPs
               Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100% 1/1 [00:01<00:00,  1.45s/it]
                 all         48        416      0.889      0.391      0.503      0.309
                   0         48        348      0.841      0.607      0.727      0.429
                   1         48         57      0.732      0.288      0.347      0.152
                   2         48          6      0.663      0.667       0.69      0.434
                   3         48          4          1          0      0.249      0.222
Results saved to runs/train/results_1
```

Figure 3.138: Results 300 Epochs

Figure 3.139

Figure 3.140

Epoch number set to 300 in order to train the model because the best model was not yet reached in the previous test. It was wanted to see if more epochs can make a difference on the results. And after that the results are obtained as it is seen below:

Here, as it is seen on the result table, the mAP is 0.503 for all classes. This result is more or less same with the previous test. Precision and recall values are also look good.

Some misclassifiacation errors exist. For example, a car detected but misclassified as a bus. Another problem is that a cyclist detected and correctly classified

Figure 3.141

Figure 3.142

but the confidence score is 0.31. This is too low. Also, fake detections observed on the images. For example, a manhole is detected as a car. It is possible to say that this is a common error from YOLOv5 because the detection of the manhole is often seen on the other tests as well. However, this time the confidence score of the detection is 0.54. This value is relatively low. This is actually a good sign that this model is more accurate then others.

Figure 3.143

Figure 3.144

Based on looking at the images, it might be said that the results are quite accurate and adding more numbers of epochs are not needed more than 300 epochs. Also, mAP value may start to decrease. Wanted to avoid from this situation.

Figure 3.145

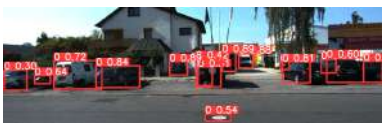

Figure 3.146

Therefore, it is obvious that the best model has been found by considering the training and validation dataset size, number of the instances and the scale of the model. In addition, setting the epoch number to 300 is quite sufficient. For this reason, this model should be used without changing any parameter or the training dataset in order to get the best results with variety of test datasets.

For testing the model, two more test added below with different test dataset. Other than the test dataset, everything remained the same. Thus, it is possible to understand the model completely.

Figure 3.147

Figure 3.148

Test With 300 Epochs - Residential Dataset

This test made with a KITTI raw dataset from the Residential category. The data which is collected on 26th of September, 2011, 19th drive. Dataset length is 487 frames, 00:48 minutes long. Image resolution is 1392 x 512 pixels. Colour images used.

This time the same dataset were not used because the aim here is to test the model on different test dataset.

```
               all         48        415      0.697      0.486      0.471      0.296

     Epoch    gpu_mem       box       obj       cls    labels  img_size
   277/299      12.4G   0.02736   0.02544 0.0005198       641       640: 100% 3/3 [00:05<00:00,  1.75s/it]
                Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100% 1/1 [00:00<00:00,  1.43it/s]
                  all         48        415      0.696      0.485      0.469      0.297
Stopping training early as no improvement observed in last 100 epochs. Best results observed at epoch 177, best model saved as best.pt.
To update EarlyStopping(patience=100) pass a new patience value, i.e. `python train.py --patience 300` or use `--patience 0` to disable EarlyStopping.

278 epochs completed in 0.453 hours.
Optimizer stripped from runs/train/results_1/weights/last.pt, 14.5MB
Optimizer stripped from runs/train/results_1/weights/best.pt, 14.5MB

Validating runs/train/results_1/weights/best.pt...
Fusing layers...
Model summary: 211 layers, 7020913 parameters, 0 gradients, 15.8 GFLOPs
                Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100% 1/1 [00:01<00:00,  1.06s/it]
                  all         48        415      0.831      0.449      0.572      0.356
                    0         48        348       0.84      0.619      0.735      0.437
                    1         48         57      0.774      0.362      0.419      0.195
                    2         48          6       0.71      0.817      0.722      0.486
                    3         48          4          1          0      0.413      0.305
```

Figure 3.149: Results 300 Epochs

Epoch set to 300 and training stopped after 278 epochs. At epoch 177, the best model was observed. The mAP value is 0.572 for all classes. When recall and precision values checked, it was seen that they are the best values, so far.

Figure 3.150

Figure 3.151

As it is seen on the images, the results are very good. Detection works very well also for the small objects which are located away from camera. The classifications are correct and even the confidence scores are high enough.

Figure 3.152

Figure 3.153

Some misclassification error observed. The traffic signs may classify as pedestrians or cyclists. Or the cyclist may be classified as a car. However, the detections and classifications have been done with high accuracy. And, on this dataset it is possible to see four of the classes correctly detected and classified.

Figure 3.154

Figure 3.155

Additionally, another test has been done same as this one meaning that with different dataset below:

Test with 300 Epochs - City Dataset

This test made with a KITTI raw dataset from the City category. The data which is collected on 26th of September, 2011, 13rd drive. Dataset length is 150 frames, 00:15 minutes long. Image resolution is 1392 x 512 pixels. Colour images used. Here are some results:

Figure 3.156

Figure 3.157

300 epochs set in order to train the model. Training stopped at 210th epoch because there was no more improvement. The best model was obtained at 110th epoch. The mAP value was 0.490.

On the images, the detections and classifications looks quite correct other than some false detections. However, a good accuracy reached overall.

Figure 3.158

Figure 3.159

Figure 3.160

Figure 3.161

3.13 Misclassification Error

Self-driving car dataset is missing labels for hundreds of pedestrians or other objects. And that's a problem that is extremely dangerous. Machine learning, the act of training computer algorithms to execute new tasks through example, has the potential to alter industries ranging from agriculture to insurance. But Machine learning techniques can only be as good as the data on which they're trained.

One highly anticipated area where machine learning will bring about societal change is the introduction of self-driving cars. However, with great power comes great responsibility; a poorly trained self-driving car can literally lead to human fatalities. That's why we were astonished and concerned when we discovered that a popular dataset used by thousands of students to develop an open-source self-driving car has crucial flaws and omissions.

Hand-checked that all 15,000 images in the widely used Udacity Dataset 2 and discovered flaws in 4,986 (33%) of them. Thousands of unlabeled vehicles, hundreds of unlabeled pedestrians, and dozens of unlabeled cyclists were among them. Also, discovered that there were several ghost annotations, duplicated bounding boxes, and very large bounding boxes.

Perhaps most gorgeously, 217 (1.4%) of the photos were totally unlabeled yet contained automobiles, trucks, street lights, and/or pedestrians. Red-highlighted annotations were missing in the original dataset:

Figure 3.162: Red-highlighted annotations were missing in the original dataset

Figure 3.163

Figure 3.164

Several example images containing pedestrians that didn't contain any annotations in the original dataset.

Open source datasets are fantastic, but if the public is to put their trust in our community, we must do a better job of ensuring the data we share is comprehensive and correct. If you use public datasets in your projects, please do your diligence and double-check their integrity before releasing them into the community.

3.13.1 Error In The Best Model

So far, we know that the YOLOv5 has performed very well based on looking at the test images. But from time to time, despite these nice recall and precision results, some misclassification detections are encountered.

Detections and classifications are mostly accurate except some misclassifications or FPs. Most significant one is that model think that a manhole cover is a car with 0.54 confidence score. This is because recall value of the model was high. In particular, car class has way more instances. For this specific circumstance, car class has the highest recall and precise values. It is possible to lower the recall value. However, this will cause less detection. Model will not

Figure 3.165

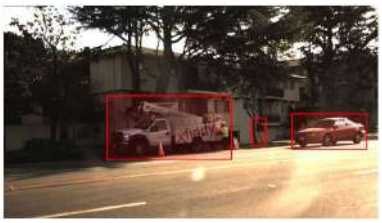

Figure 3.166

be able to detect the objects that is detecting correctly. This trade off have to be considered depending on the purpose of the application of the model.

Misclassification errors, fake detections and other errors which are observed in real-time application mentioned before parallel with the tests.

4

Glare Free Mask

The process of creating a foreground mask specifically, a binary picture including the pixels belonging to moving objects in the scene by using static cameras is known as background subtraction. By subtracting the current frame from a background model that contains the static portion of the scene or, more generally, everything that can be regarded as background given the features of the observed scene, background subtraction produces the foreground mask.

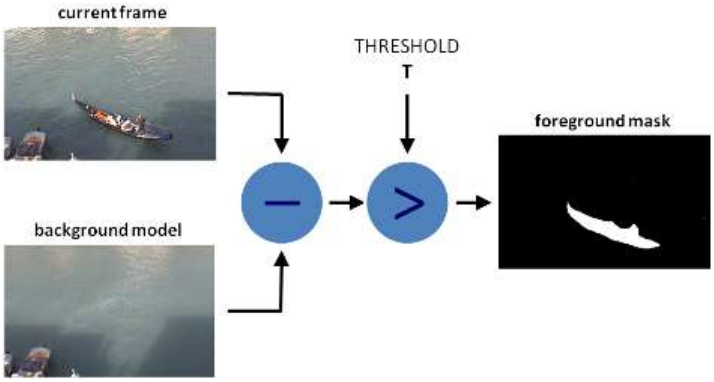

Figure 4.1: Background Mask

Background subtraction modeling consists of two main steps. The first step is background initialization. In the background initialization step, an initial model of the background is computed. The second step is background update. In the background update step model is updated in order to adapt to possible changes in the scene.

The application of background subtraction mask is Glare free high beams. Glare free beams used in order to provide better driver experience and safe drive. Applying a glare free mask to beams avoids the distraction from the road and provides better view to the driver.

KITTI Raw dataset used in order to test the background subtraction mask. Raw data was distributed between categories. The test made with a dataset from City category. The data which is collected on 26th of September, 2011, 17th drive. Dataset length is 120 frames, 00:12 minutes long. Image resolution is 1392 x 512 pixels.

Figure 4.2: An image from KITTI City Dataset

Glare free mask applied on the grayscale images. All the grayscale images available on KITTI web site. Likewise, a code snippet can be used to convert a color image into a grayscale image. OpenCV-Python library used. The OpenCV-Python library of Python extensions was created to address issues with computer vision. Cv2 library has to be imported using import statement in order to use cv2 library. The method cv2.imread() loads an image from the given file. This method produces an empty matrix if the picture cannot be read (due to an unsupported format, a missing file, inappropriate permissions, or another issue). CV2.imread() takes 2 parameters: path and flag. If flag passes integer value 1, it specifies to load a color image. This is the default value. If flag passes integer value 0, it specifies to load a grayscale image. If flag passes integer value -1, it specifies to load an image as such including alpha channel.

```
#importing cv2
import cv2

# Using cv2.imread() method
```

```
# Using 0 to read image in grayscale mode
image = cv2.imread(imagePath, 0)

# Displaying the image
cv2.imshow('image', img)
```

Code 4.1: Load an image grayscale

Figure 4.3: Grayscale Image

In cv2, library threshold() method used in order to separte an object from the background in the image. Four parameters requested to pass to cv2 threshold() method. First, 'src' which is input grayscale image array, Second, 'thresholdValue' which mention that value which is used to classify the pixel values. Third, 'maxVal' which is the value to be given if pixel value is more than (sometimes less than) the threshold value. Four, 'thresholdingTechnique' which is the type of thresholding to be applied. There are five thresholding methods. For this specific case THRESH_BINARY method used.

cv2.THRESH_BINARY: If pixel intensity exceeds the threshold, value set to 255, else set to 0 means black.

Threshold() method is applied on the grayscale images with cv2 library:

```
_, mask = cv2.threshold(image, thresh=10, maxval=255, type=cv2.
    THRESH_BINARY)
im_thresh_gray = cv2.bitwise_not(image, mask)
```

Code 4.2: Mask

Figure 4.4: Mask

After images loaded, a mask applied on the images as it is seen on the Figure 4.4. However, this would not be so useful in order to project glare free beams. Image has too much noise. Difficult to distinguish the objects from each other.

Sometimes to crop an area which has more objects that is looked for can reduce the noise. Region of interest defined to grayscale image in order to consantrate on interested area.

Figure 4.5: Region of Interest

Chosen area defined with [100:850, 400:1000]. The aim is to capture less trees, traffic signs, traffic lights or other irrelevant objects and focus only to the road. After that an background subtractor mask applied on the region of interest.

```
object_detector = cv2.createBackgroundSubtractorMOG2(history=100,
    varThreshold=80)

#Object Detection
    mask= object_detector.apply(roi)
    _, default_mask = cv2.threshold(mask, 254, 255, cv2.THRESH_BINARY
    )
    _, mask = cv2.threshold(mask, 254, 255, cv2.THRESH_BINARY_INV)
```

Code 4.3: Background Subtractor Mask

Background subtractor implemented with cv2 library createBackgroundSubtractorMOG2 method. It has two parameters. They are history and varThreshold. Setting the history value higher helps algorithm to be more precise but algorithm hardly adapt to the changes if the camera is not stable. Setting the varThreshold value higher cause less detections but less FPs.

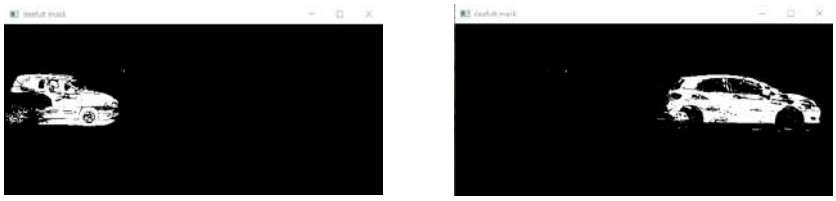

Figure 4.6: Examples of Background Subtractor Mask Applied Region of Interest Area

cv2 library cv2. THRESH_BINARY applies the default mask as it is mentioned before. Instead, cv2. THRESH_BINARY_INV is a mask that removes everything less than defined threshold value. In binary representation 1 stands for colour black and 255 stands for colour white. Inverse Background Subtractor Mask Applied and it removed everything 254 and below from the region of interest area. Thus, everything else will be white.

Figure 4.7: Examples of Inverse Background Subtractor Mask Applied Region of Interest Area

As it is seen on the Figure 4.7, Cars are black and all the rest is white. This is proper for 8bits or 10bits representation of the image. Image file can easily converted to a comma separated format with the help of the duty cycles from 1 to 255. In order to project it with mPLS, this mask requested.

Later, another dataset used in order to apply a mask. KITTI Raw dataset used in order to test the background subtraction mask. Raw data was distributed between categories. The test made with a dataset from City category. The data which is collected on 26th of September, 2011, 18th drive. Dataset length is 276 frames, 00:27 minutes long. Image resolution is 1392 x 512 pixels.

Figure 4.8: Color image

Same as before, images opened as a grey-scaled images and cropped a region of interest. Then, an object mask generated also for this dataset in order to subtract the background.

Figure 4.9: Object Mask

It is seen that to implement this script also on other dataset is possible. Here is the script:

```
!pip install imutils
from imutils import paths
import numpy as np
import cv2
import matplotlib.pyplot as plt
%matplotlib inline
#jupyter notebook
import numpy as np
import os
import PIL
```

```
import PIL.Image
import tensorflow as tf

#from tracker import*

import math


class EuclideanDistTracker:
    def __init__(self):
        # Store the center positions of the objects
        self.center_points = {}
        # Keep the count of the IDs
        # each time a new object id detected, the count will increase by one
        self.id_count = 0


    def update(self, objects_rect):
        # Objects boxes and ids
        objects_bbs_ids = []

        # Get center point of new object
        for rect in objects_rect:
            x, y, w, h = rect
            cx = (x + x + w) // 2
            cy = (y + y + h) // 2

            # Find out if that object was detected already
            same_object_detected = False
            for id, pt in self.center_points.items():
                dist = math.hypot(cx - pt[0], cy - pt[1])

                if dist < 100:
                    self.center_points[id] = (cx, cy)
                    print(self.center_points)
                    objects_bbs_ids.append([x, y, w, h, id])
                    same_object_detected = True
                    break

            # New object is detected we assign the ID to that object
            if same_object_detected is False:
                self.center_points[self.id_count] = (cx, cy)
```

```
                objects_bbs_ids.append([x, y, w, h, self.id_count])
                self.id_count += 1

        # Clean the dictionary by center points to remove IDS not
    used anymore
        new_center_points = {}
        for obj_bb_id in objects_bbs_ids:
            _, _, _, _, object_id = obj_bb_id
            center = self.center_points[object_id]
            new_center_points[object_id] = center

        # Update dictionary with IDs not used removed
        self.center_points = new_center_points.copy()
        return objects_bbs_ids




#imagePaths = list(paths.list_images("data1"))
imagePaths = list(paths.list_images("drive0018")) #drive0018 dataset
#imagePaths = list(paths.list_images("data11092635"))



object_detector = cv2.createBackgroundSubtractorMOG2(history=100,
    varThreshold=50)

#Display each images grayscale
for imagePath in imagePaths:
    image = cv2.imread(imagePath, 0) #0 makes grayscale
    cv2.imshow("Frame", image)
    cv2.waitKey(0) #frame sequence (in order to use enter button
    change 30 w 0)

    _, mask = cv2.threshold(image, thresh=10, maxval=255, type=cv2.
    THRESH_BINARY)
    _, mask_i = cv2.threshold(image, thresh=10, maxval=255, type=cv2.
    THRESH_BINARY_INV)
    #im_thresh_gray = cv2.bitwise_not(image, mask)

    cv2.imshow("mask", mask)
```

```
91      #cv2.imshow("mask INVERSE", mask_i)
92      #cv2.imshow("im_thresh_gray", im_thresh_gray)
93
94      #save images
95      #cv2.imwrite("save_frame.png", image)
96      #cv2.imwrite(imagePaths, image)
97      #cv2.imwrite("save_mask.png", mask)
98      #cv2.imwrite(imagePath, mask)
99
100
101     #Extract Region of interest
102     #roi = image[150:720, 300:800]
103     roi = image[100:850, 300:1100]
104
105
106
107     #Object Detection
108     mask= object_detector.apply(roi)
109     _, mask = cv2.threshold(mask, thresh=10, maxval=255, type=cv2.
    THRESH_BINARY)
110
111     #cv2.THRESH_BINARY ->DEFAULT SUBSTRACTED BACKGROUND MASKED IMG
    cv2.THRESH_BINARY_IVY INVERSED PROPER FOR CONVERING TO CSV AND
    PROJECING HPLD2
112     _, default_mask = cv2.threshold(mask, 254, 255, cv2.THRESH_BINARY
    )
113     _, mask_inv = cv2.threshold(mask, 254, 255, cv2.THRESH_BINARY_INV
    ) #(1->black, 255->white) we remove everything 254 and below and
    all th erest will be white
114     #_, mask_trunc = cv2.adaptiveThreshold(mask, 255, cv2.
    ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 11);
115     contours, _ = cv2.findContours(mask, cv2.RETR_TREE, cv2.
    CHAIN_APPROX_SIMPLE)
116     detections = []
117     for cnt in contours:
118         #Calculate area and remove small elements
119         area = cv2.contourArea(cnt)
120         if area > 300: #pixels
121             #cv2.drawContours(roi, [cnt], -1, (0, 255, 0), 2)
122             x, y, w, h = cv2.boundingRect(cnt)
123             #print(x, y, w, h)
124             detections.append([x, y, w, h])
125
```

```
126             #add recktangles on detected cars
127             #label = str("Car detected!")
128             #img_rect = cv2.rectangle(mask_inv,(x,y),(x+w+5,y+h+5)
    ,(0,255,0),-1)
129             #cv2.putText(img_rect, label, (x, y + 20), 3, 1,
    (255,0,0), 1)
130     #cv2.imshow("image", img_rect)
131     #cv2.imwrite(imagePath, img_rect)
132
133
134
135
136
137     print(detections)
138     cv2.imshow("default mask", default_mask)
139     cv2.imshow("roi", roi)
140     cv2.imshow("Frame", image)
141     cv2.imshow("Mask Inverse", mask_inv)
142     #v2.imshow("Mask Trunc", mask_trunc)
143
144     #save the final mask img
145     cv2.imwrite(imagePath, mask_inv)
146
147 key = cv2.waitKey(30)
148 if key == 27:
149     break
150
151 cap.release()
152
153
154 cv2.destroyAllWindows()
```

Code 4.4: Background Subtractor Mask

As it is seen on the script, to the dataset another mask is implemented. This masks add rectangles on the cars in order to obtain a more consistent mask. Because on the some images, it is seen time to time that the mask faded and illuminated. This observed mostly on the glasses of the car because glasses reflects the light. In this case it is not possible to recognize a car and apply a mask on it. And this also valid for the metal on the car that reflex the the light in the same way. Because the is to avoid to blind the drivers with the pixel light source. In order to do that, applying a rectangle shape mask might be more

efficient based on the comparing the two different masks below:

Figure 4.10: Real shape of the car mask and rectangle masks demonstrated

Figure 4.11: Real shape of the car mask and rectangle masks demonstrated

Figure 4.12: Real shape of the car mask and rectangle masks demonstrated

Moreover, on the code script it is seen that, it is possible to add some text on to the rectangle mask. This would be replaced with some symbols or the logo of the brands, etc.

4.1 Transferring Images to Pixel Light Source Device

Micro pixel light source device used in order to project the generated images. As it is mentioned before it has 16k individually controlled LEDs on it. It is important to mention that before I start to work on this project, pixel light source involved in some other projects with includes some stress tests. At the end of this tests, some of the LEDs received always off error. They does not work, basically. This could cause some noise and because of that some confusion on the illuminated part. However, looking at the software generated images before

they projected on the device might address this issue and could help to get precise results.

Masked images generated and saved as PNG files but sending them to the pixel light source as they as is not possible, in this point. Data have to be converted in a format that the micro controller Infineon XMC4700 Relax Kit[2] and micro pixel light source electronics could read. The firmware of the micro controller is able to read CSV files. All the images converted to the CSV files in order to send them without an issue. Here is the python script that converts PNG files to CSV files:

```
# import required libraries
import numpy as gfg
import matplotlib.image as img
import pandas as pd
import cv2
import csv

im = cv2.imread("drive0018_mask_rect/0000000062.png")       # Read
    image
imS = cv2.resize(im, (256, 64))       # Resize image
#imS = cv2.resize(im, (400, 300))       # Resize image
cv2.imshow("output", imS)                       # Show image
cv2.waitKey(0)

# read an image
print("Image shape:", imS.shape)

# if image is colored (RGB)
if(imS.shape[2] == 3):

  # reshape it from 3D matrice to 2D matrice
  imS_reshape = imS.reshape(imS.shape[0], -1)
  imS_reshape = cv2.resize(imS_reshape, (256, 64))
  print("Reshaping to 2D array:", imS_reshape.shape)

# if image is grayscale
else:
  # remain as it is
  imS_reshape = imS

# converting it to dataframe.
```

```
mat_df = pd.DataFrame(imS_reshape)

# exporting dataframe to CSV file.
mat_df.to_csv('drive18_rect-00000000062.csv',  sep = ';', header =
    None, index = None)

# retrieving dataframe from CSV file
loaded_df = pd.read_csv('drive18_rect-0000000062.csv', sep = ';',
    header = None)
# getting matrice values.
loaded_2D_mat = loaded_df.values

# reshaping it to 3D matrice
loaded_mat = loaded_2D_mat.reshape(loaded_2D_mat.shape[0],
    loaded_2D_mat.shape[1] // imS.shape[2], imS.shape[2])
print("Image shape of loaded Image :", loaded_mat.shape)
```

Code 4.5: PNG to CSV Converter

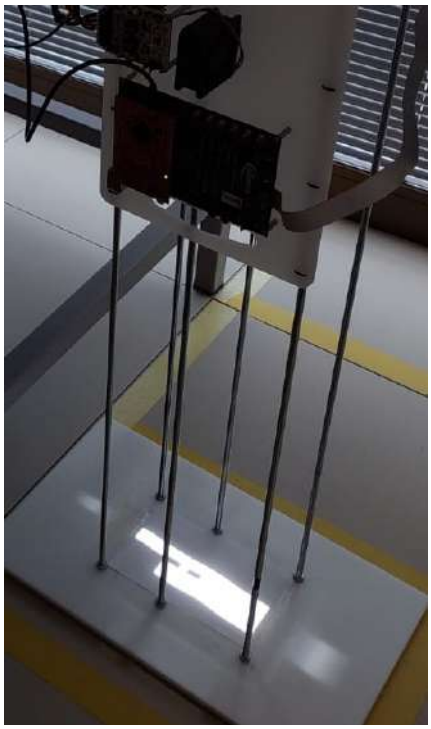

Figure 4.13: Micro Pixel Light Source Device

Some screen shots demonstrated that a colour frame, a background subtracted mask and a rectangle mask merged together in a real-time application. A video

Figure 4.14: Masks projected from Pixel Light Source

generated 10 seconds long. Below is some images captured from the video:

It is important to mentioned that generated images projected via device. When an image projected, it is flipped horizontally different than original image. Thus, the original image is flipped horizontally in order to merge them all in the same way.

Figure 4.15: Masks projected from Pixel Light Source

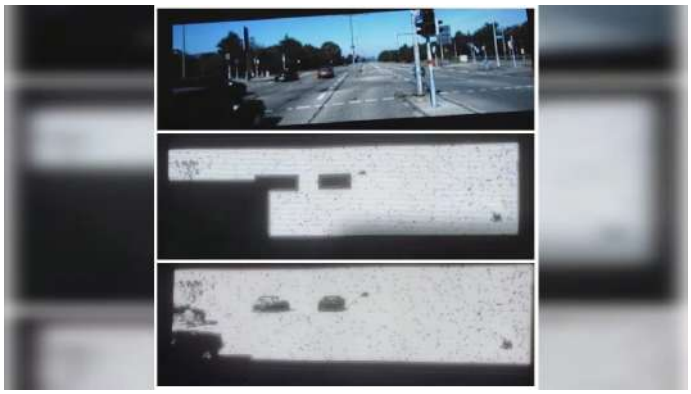

Figure 4.16: A screenshot of the merged video

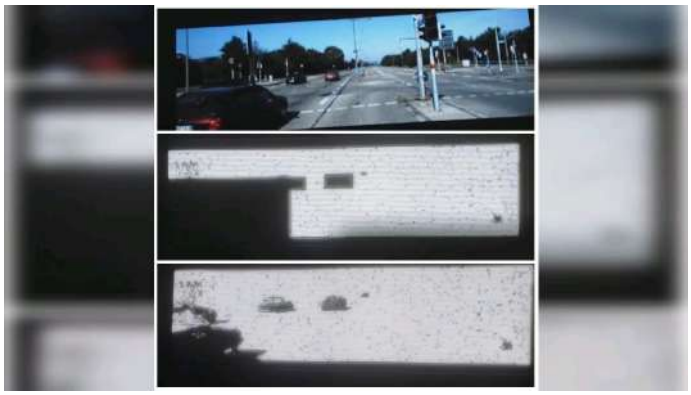

Figure 4.17: A screenshot of the merged video

Figure 4.18: A screenshot of the merged video

5

Conclusion

This work aimed to contribute in automotive field. It might be helpful for some autonomous driving applications in real-time. The collaboration with the electronic systems, wanted to add advance functionalities to standard pixel light source. Adding advance functionalities could help safer driving and better driving experience.

Object detection and classification with a confidence score is applied on a real-time source. By trying to train the model with different parameters the best model is obtained. Distribution of training dataset with the learning 0.2 and setting the epochs to 300 gave the mAP value 0.572 as the most accurate result.

Worked on the glare-free pixel light source in order to avoid the drivers from blinding with the illumination of the beams. Shape of the car and rectangle shaped masked generated as image files and sent to micro pixel light source device as CSV files. As the result, the rectangle shape mask works more precise.

Therefore, the new advanced pixel light source provides a better and safer driving experience to both the driver and other driver who is driving from the opposite direction. Moreover, this is a contribution to autonomous driving and in the future even more functionalities can be added. Object detection and classification model might be extended with more classes such as recognizing the traffic signs, white strips, barriers besides the car and etc. And segmentation model could be added for a car to recognize the street, the lines, boarders

and zebra lines. Also, a three dimensional model would be trained in order to make it possible for an autonomous car to distinguish between a real car and advertisement of a car printed on a billboard as two dimensional. These developments will be crucial in the future also for self driving cars.

6 Appendix

6.1 Data Transfer Protocols

USB

The Universal Serial Bus (USB) is an industry standard that specifies cables, connections, and protocols for connecting, communicating, and powering computers, peripherals, and other computers.[22] It is synchronous. A broad variety of USB hardware exist, including 14 different connector types of which USB-C is the most recent connector. The connector include 4 pins. Pin 1 is the +5V, pin 2 is the Data-, pin 3 is the Data+ and pin 4 is the GND.

Figure 6.1: USB logo on the head of a standard USB-A plug

USB was designed to standardize the connection of peripherals to personal computers, both to communicate with and to supply electric power. It has largely replaced interfaces such as serial ports and parallel ports and has become common place on a wide range of devices. Example of peripherals that are connected with via USB include computer keyboards and mouse, video cameras,

printers, portable media players, mobile(portable) digital telephones, disk drives and network adapters.

ETHERNET

In a local area network, Ethernet is the protocol of choice. A local area network is essentially a group of interconnected devices that are positioned relatively close together in a limited area. However, three characteristics distinguish LANs from wide area networks (WANs). First, physical proximity with a narrower geographic range, but also the resources to run at high speed data rates. Bandwidth is abundant, and it typically goes anywhere from 100MB/s to the 1 GB/s and 10 GB/s that we see in today's network. Third and perhaps the most important one, is that they do not need a lease line or a telecom provider or service provider in order to interconnect the devices. A LAN can be as small as a simple office, or even a teleworker's home office and from there to a full campus with multiple buildings and fiber connections between the buildings.

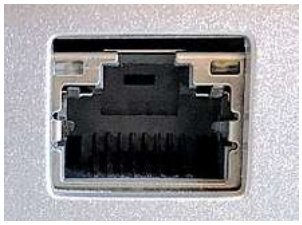

Figure 6.2: Ethernet port

The most popular and oldest LAN technology is Ethernet protocol, so it is more frequently used in LAN environments which is used in almost all networks like offices, homes, public places, enterprises, and universities. Ethernet has gained huge popularity because of its maximum rates over longer distances using optical media.

Ethernet protocol uses a star topology or a linear bus which is the foundation of the IEEE 802.3 standard. The main reason to use Ethernet widely is simple to understand, maintain, implement, provides flexibility, and permits less cost network implementation.

In the OSI Network Model, Ethernet protocol operates at the first two layers like the physical and data Link Layers but Ethernet separates the Data Link Layer into two different layers called the Logical Link Control layer and the Medium Access Control Layer..

The physical layer in the network mainly focuses on the elements of hardware like repeaters, cables network interface cars(NIC). The data link layer in the network system mainly addresses the way that data packets are transmitted from one type of node to another. Ethernet uses an access method called CSMA/CD. This is the system where every computer listens to the cable before transmitting anything through the network.

The two layers in the above Ethernet protocol block diagram deal with the physical network structure where the network devices can transmit data from one device to another on a network. Certainly, the most popular set of protocols used for both the physical data link layer is known as Ethernet. Ethernet is available in different forms where the current Ethernet can be defined through the IEEE 802.3 standard.

Ethernet protocols are available in different flavors and operates at various speeds by using different types of media. But, all the Ethernet versions are well-matched through each other. These versions can mixmatch on a similar network with the help of different network devices like hubs, switches, bridges to connect the segments of the network that utilize different types of media.

The Ethernet protocol's actual transmission speed can be measured in Mbps (millions of bits for each second). The speed versions of Ethernet are available in three different types 10Mbps, called standard Ethernet; 100Mbps called fast Ethernet and 1000 Mbps called as Gigabit Ethernet. The transmission speed of the network is the maximum speed that can be attained over the network in ideal conditions. The output of the Ethernet network rarely achieves this highest speed.

It works mainly in the first two layers in the OSI network model like data link and physical. Ethernet at the first layer uses signals, bit streams that move

on the media, physical components that situate signals on media and different topologies.

Ethernet plays a key role at L1 in the communication that occurs between different devices, however every function of this has some limitations. The sub-layers of data link give significantly to technological compatibility and computer communications. The MAC sublayer is anxious through the physical components that will be utilized to converse the information and arrange the data for communication over the media. The logical link control sublayer will stay independent of the physical equipment that will be utilized for the process of communication. Ethernet protocol simply divides the data link layer functions into two separate sublayers like the logical link control sublayer and the media access control sublayer.

The functions of the data link layer in the OSI model are allocated to both the sub layers like LLC and MAC. For Ethernet protocol, the IEEE 802.2 std simply explains the functions of the LLC sub-layer and the 803.3 std explains the functions of MAC and the physical layer.

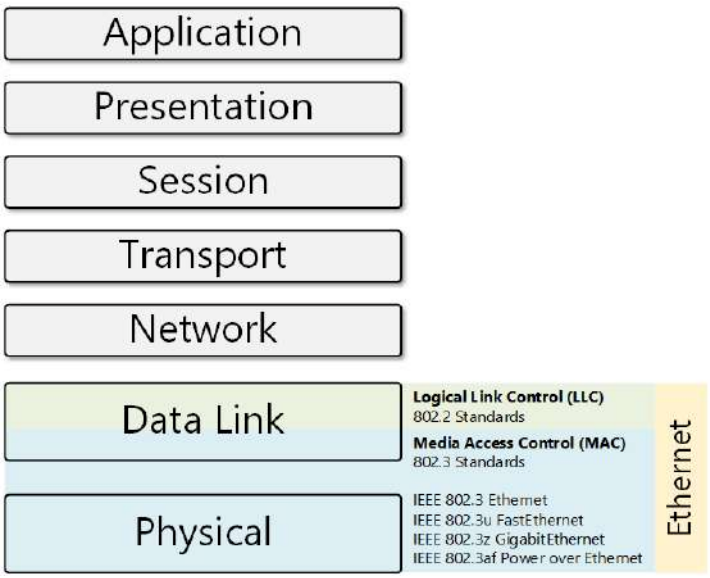

Figure 6.3: Ethernet OSI Layers

The logical link control (LLC) handles the communication between the upper layers and lower layers. The LLC layer uses the data of network protocol like

IPv4 packet and adds control data to help in delivering the packet toward the destination node. The second layer (L2) interacts through the higher layers using LLC which can be implemented within software and its implementation is independent of the physical devices.

In a computer system, the logical link control can be considered the driver software of the Network Interface Card. So, the driver software of NIC is a program that directly communicates through the hardware on the network interface card to transmit data in between the two layers of MAC and Media.

References

[1] In: https://doi.org/10.48550/arXiv.2004.02806.

[2] In: https://www.infineon.com/cms/en/product/microcontroller/32-bit-industrial-microcontroller-based-on-arm-cortex-m/32-bit-xmc4000-industrial-microcontroller-arm-cortex-m4/xmc4700/.

[3] Hong-Yuan Mark Liao Alexey Bochkovskiy Chien-Yao Wang. "YOLOv4: Optimal Speed and Accuracy of Object Detection". In: 23 Apr 2020.

[4] Chauvin. In: 2000.

[5] "Electric Motors". In: *Machine Design*. Retrieved March 23, 2011.

[6] Laurens van der Maaten Gao Huang Zhuang Liu. "Densely Connected Convolutional Networks". In: 28 Jan 2018.

[7] Andreas Geiger et al. "Vision meets Robotics: The KITTI Dataset". In: 2013.

[8] R. B. Girshick. "Fast R-CNN". In: 2015, pp. 80–83.

[9] James Jeans. "Science Music". In: *Dover Publications*. 1968/1937, pp.222 224.

[10] Quoc V. Le Mingxing Tan Ruoming Pang. "EfficientDet: Scalable and Efficient Object Detection". In: 27 Jul 2020.

[11] Diganta Misra. "Mish: A Self Regularized Non-Monotonic Activation Function". In: 13 Aug 2020.

[12] Moreno and Albacete. In: 2010.

[13] Martin Jan Musiol. "Speeding up". In: Jan 2016.

[14] Andrew Ng. "Deep L-Layer Neural Network Deep Neural Networks". In: Coursera. URL: www.coursera.org/learn/neural-networks-deep-learning/lecture/7dP6E/deep-l-layer-neural-network.

[15] Adam Osborne. “An Introduction to Microcomputers Volume 1: Basic Concepts”. In: *Osborne-McGraw Hill Berkeley California USA*. 1980, pp. 116–126.

[16] R. B. Girshick P. F. Felzenszwalb. “Object detection with discriminatively trained part based models”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 2010, 32(9):1627–1645.

[17] Adam Paszke et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[18] J. Donahue R. Girshick. “Rich feature hierarchies for accurate object detection and semantic segmentation”. In: *In Computer Vision and Pattern Recognition (CVPR)*. 2014, pp. 580–587.

[19] J. Redmon and A. Farhadi. “YOLO9000:Better, Faster, Stronger”. In: 25 Dec 2016.

[20] J. Redmon and A. Farhadi. “Yolov3: An incremental improvement”. In: 2018.

[21] Seong Joon Oh Sangdoo Yun Dongyoon Han. “CutMix: Regularization Strategy to Train Strong Classifiers with Localizable Features”. In: 7 Aug 2019.

[22] Simson. “USB deserves more support”. In: *Business. Boston Globe Online*. 1 December 1995. Archived from the original on 6 April 2012. Retrieved 12 December 2011.

[23] Guanzhong Wang Xiang Long Kaipeng Deng. “PP-YOLO: An Effective and Efficient Implementation of Object Detector”. In: 3 Aug 2020.

Acknowledgments

Abstract

I wish to extend my special thanks to following people for helping me finalize the project: my manager Andrea Morici to helped me to start this great journey in Infineon Technologies and all the help for my thesis and also for my career, my tutor Riccardo Zuin to explain me everything related to pixel light source from basic to advance and electronics and supporting me for every single step. And thanks for the support to everyone in Infineon Italy Automotive Body Power Technical Marketing and Software Department. Moreover, thanks to Professor Federica Battisti for encouraging me to work in this field and for the support.