Faculty of engineering cairo university
Communications and electronics departement

Car detection and depth estimation using mono camera

Based on machine and deep learning algorithms
For self driving cars

Supervisors :

Dr.Hassan Mostaf DR. Hossam Hassan

Prepared By:

Ibrahim Mohamed Ibrahim

Mahmoud Ahmed Saad

Mostafa Mohamed Mostafa

Rania Mahmoud Hassan

Contents

List of Figures

Chapter 1

Deep learning

1.1 Why Deep Learning over Traditional Machine Learning?

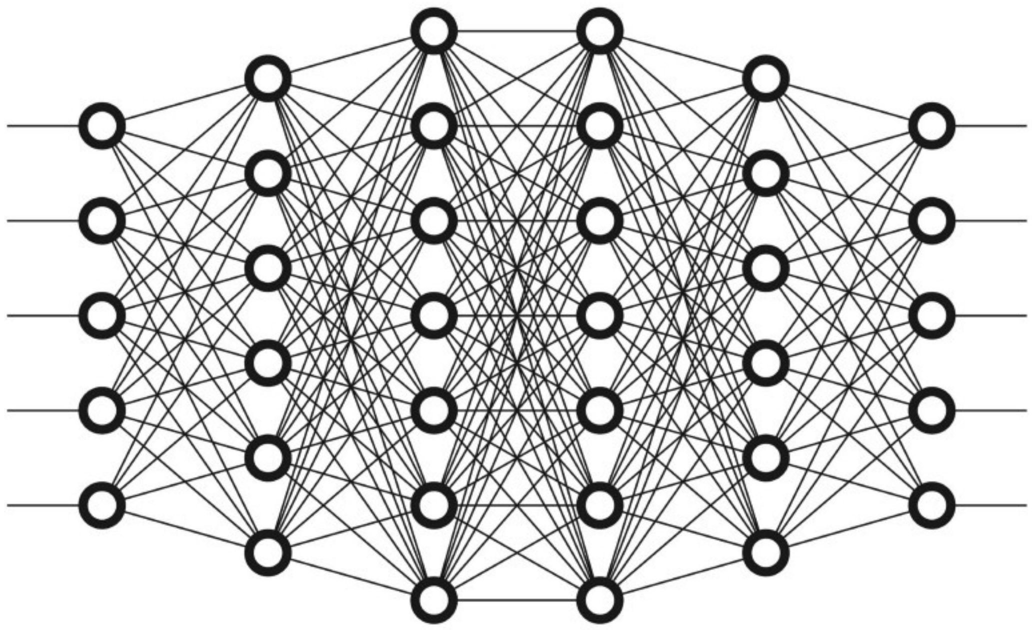


Figure 1.1: Neural Network

Artificial Intelligence boils down to two very popular concepts Machine Learning and Deep Learning. But lately, Deep Learning is gaining much popularity due to its supremacy in terms of accuracy when trained with a huge amount of data.

Here is the Google trend for the keyword deep learning:

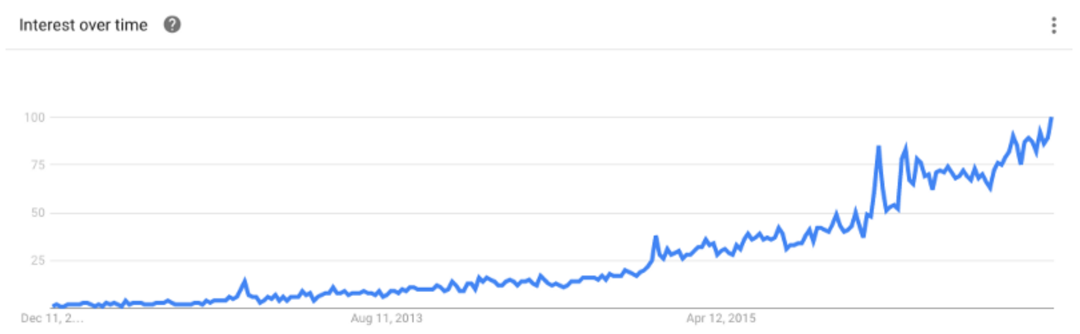


Figure 1.2: Trend of "Deep Learning" in google

The software industry nowadays moving towards machine intelligence. Machine Learning has become necessary in every sector as a way of making machines intelligent. In a simpler way, Machine Learning is a set of algorithms that parse data, learn from them, and then apply what they've learned to make intelligent decisions.

Examples of Machine Learning are everywhere. It's how Netflix knows which show you'll want to watch next or how Facebook recognizes your friend's face in a digital photo.

The thing about traditional Machine Learning algorithms is that as complex as they may seem, they're still machine-like. They need a lot of domain expertise, human intervention only capable of what they're designed for; nothing more, nothing less. For AI designers and the rest of the world, that's where deep learning holds a bit more promise.

1.2 What is Deep Learning ?

Practically, Deep Learning is a subset of Machine Learning that achieves great power and flexibility by learning to represent the world as a nested hierarchy of concepts, with each concept defined in relation to simpler concepts, and more abstract representations computed in terms of less abstract ones.

Elaborately, a deep learning technique learns categories incrementally through its hidden layer architecture, defining low-level categories like letters first then little higher level categories like words and then higher level categories like sentences. In the example of image recognition, it means identifying light/dark areas before categorizing lines and then shapes to allow face recognition. Each neuron or node in the network represents one aspect of the whole and together they provide a full representation of the image. Each node or hidden layer is given a weight that represents the strength of its relationship with the output and as the model develops the weights are adjusted.

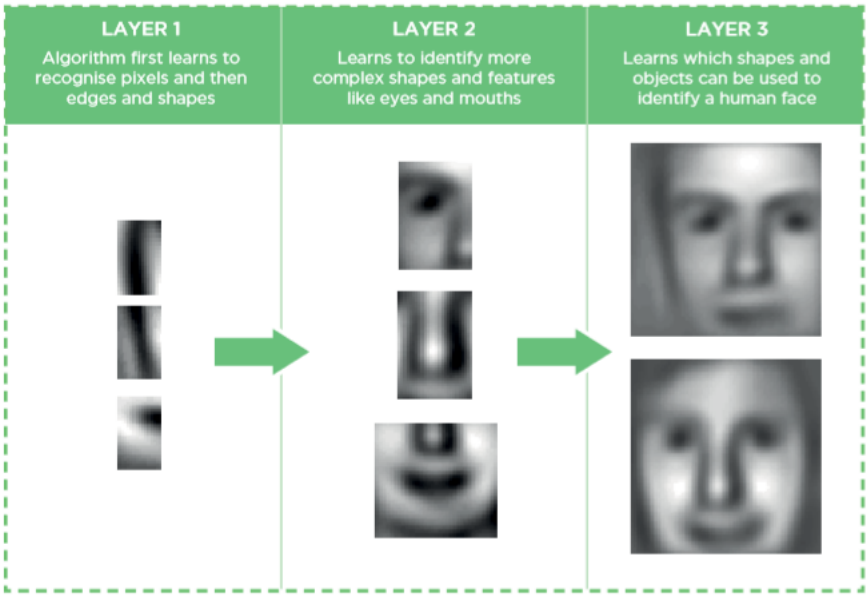

LAYER 1
Algorithm first learns to recognise pixels and then edges and shapes
LAYER 2
Learns to identify more complex shapes and features like eyes and mouths
LAYER 3
Learns which shapes and objects can be used to identify a human face

Figure 1.3: Deep Learning Architecture

1.3 Distinctive Features Of Deep Learning

A big advantage with deep learning and a key part in understanding why it's becoming popular is that it's powered by massive amounts of data. The "Big Data Era" of technology will provide huge amounts of opportunities for new innovations in deep learning. As per Andrew Ng, the chief scientist of China's major search engine Baidu and one of the leaders of the Google Brain Project, "The analogy to deep learning is that the rocket engine is the deep learning models and the fuel is the huge amounts of data we can feed to these algorithms."

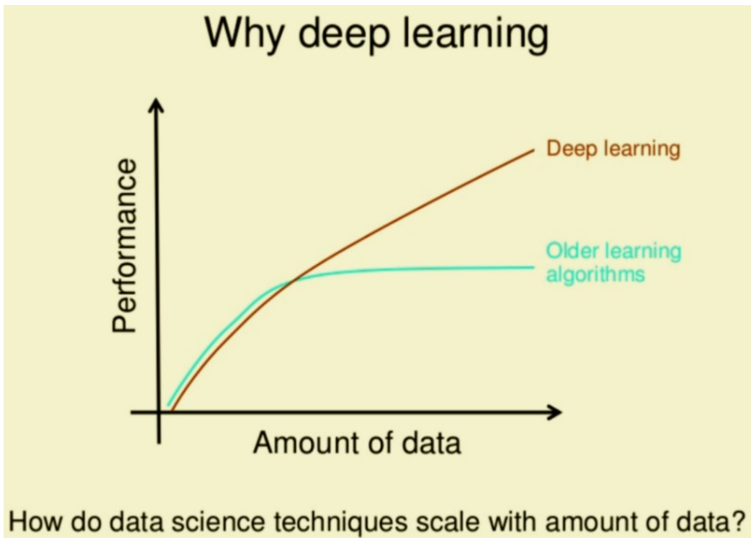


Figure 1.4: Amount of data VS. deep and machine learning techniques

Deep Learning requires high-end machines contrary to traditional Machine Learning algorithms. GPU has become an integral part now to execute any Deep Learning algorithm.

In traditional Machine learning techniques, most of the applied features need to be identified by a domain expert in order to reduce the complexity of the data and make patterns more visible to learning algorithms to work. The biggest advantage Deep Learning algorithms as discussed before are that they try to learn high-level features from data in an incremental manner. This eliminates the need for domain expertise and hard core feature extraction.

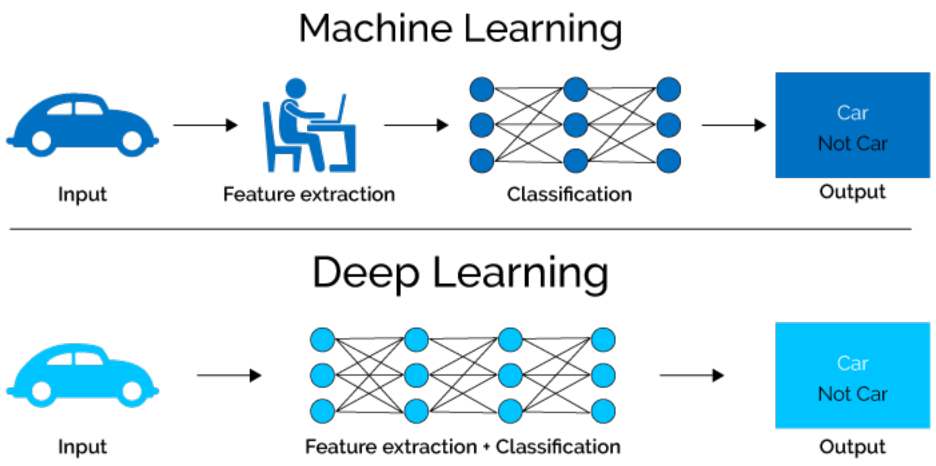


Figure 1.5: Machine learning Vs. Deep learning

Another measure difference between Deep Learning and Machine Learning technique is the problem-solving approach. Deep Learning techniques tend to solve the problem end to end, whereas Machine learning techniques need the problem statements to break down to different parts to be solved first and then their results to be combined at the final stage. For example for a multiple object detection problem, Deep Learning techniques like Yolo net take the image as input and provide the location and name of objects at the output. But in usual Machine Learning algorithms like SVM, a bounding box object detection algorithm is required first to identify all possible objects to have the HOG as input to the learning algorithm in order to recognize relevant objects.

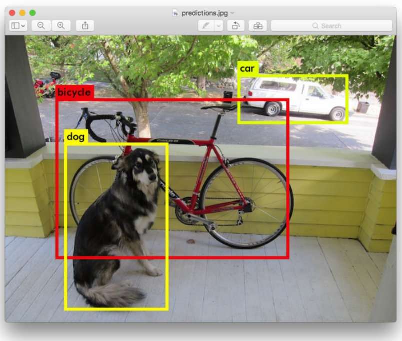


Figure 1.6: YOLO's output

Usually, a Deep Learning algorithm takes a long time to train due to a large number of parameters. Popular ResNet algorithm takes about two weeks to train completely from scratch. Whereas traditional Machine Learning algorithms take a few seconds to few hours to train. The scenario is completely reverse in the testing phase. At test time, Deep Learning algorithm takes much less time to run. Whereas, if it is compared with k-nearest neighbors (a type of machine learning algorithm), test time increases on increasing the size of data. Although this is not applicable on all machine learning algorithms, as some of them have small testing times too.

Interpretability is the main issue why many sectors using other Machine Learning techniques over Deep Learning. if deep learning is used to calculate the relevance score of a document. The performance it gives is quite excellent and is near the human performance. But there's is an issue. It does not reveal why it has given that score. Indeed mathematically you can find out which nodes of a deep neural network were activated, but we don't know what their neurons were supposed to model and what these layers of neurons were doing collectively. So we fail to interpret the results. Which is not in case of Machine Learning algorithms like decision trees, logistic regression etc.

1.4 When to use Deep Learning or not over others?

1. Deep Learning outperforms other techniques if the data size is large. But with small data size, traditional Machine Learning algorithms are preferable.

2. Deep Learning techniques need to have the high-end infrastructure to train in reasonable time.

3. When there is a lack of domain understanding for feature introspection, Deep Learning techniques outshines others as we have to worry less about feature engineering.

4. Deep Learning really shines when it comes to complex problems such as image classification, natural language processing, and speech recognition.

1.5 Different between deep learning and neural network

Today, People tend to view that one implies the other. Indeed, Deep Neural Networks are the showcase for Deep Learning. Some argue that Decision trees and Hierarchical temporal memory qualify as non-neural deep learning models. The point is that the concept of Deep Learning is not tied to Neural Networks but is very well demonstrated through them.

Brieflly :
Deep learning = deep artificial neural networks + other kinds of deep models.

Deep artificial neural networks = artificial neural networks with more than 1 layer.

1.6 Types of neural networks

1.6.1 Feedforward Neural Network – Artificial Neuron

This neural network is one of the simplest forms of ANN, where the data or the input travels in one direction. The data passes through the input nodes and exit on the output nodes. This neural network may or may not have the hidden layers. In simple words, it has a front propagated wave and no backpropagation by using a classifying activation function usually.
Below is a Single layer feed forward network. Here, the sum of the products of inputs and weights are calculated and fed to the output. The output is considered if it is above a certain value i.e threshold(usually 0) and the neuron fires with an activated output (usually 1) and if it does not fire, the deactivated value is emitted (usually -1).

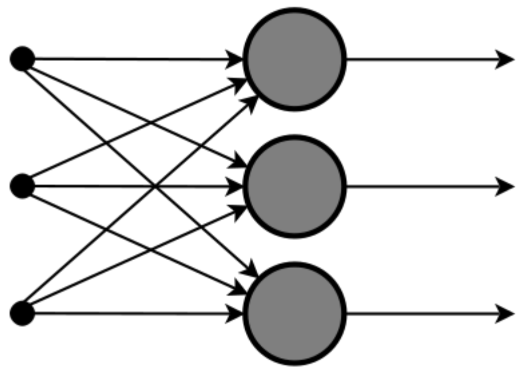


Figure 1.7: Single layer feedforward neural network

Application of Feedforward neural networks are found in computer vision and speech recognition was classifying the target classes are complicated. These kind of Neural Networks are responsive to noisy data and easy to maintain.

1.6.2 Radial basis function Neural Network

Radial basic functions consider the distance of a point with respect to the center. RBF functions have two layers, first where the features are combined with the Radial Basis Function in the inner layer and then the output of these features are taken into consideration while computing the same output in the next time-step which is basically a memory.
Below is a diagram which represents the distance calculating from the center to a point in the plane similar to a radius of the circle. Here, the distance measure used in euclidean, other distance measures can also be used. The model depends on the maximum reach or the radius of the circle in classifying the points into different categories. If the point is in or around the radius, the likelihood of the new point

begins classified into that class is high. There can be a transition while changing from one region to another and this can be controlled by the beta function.

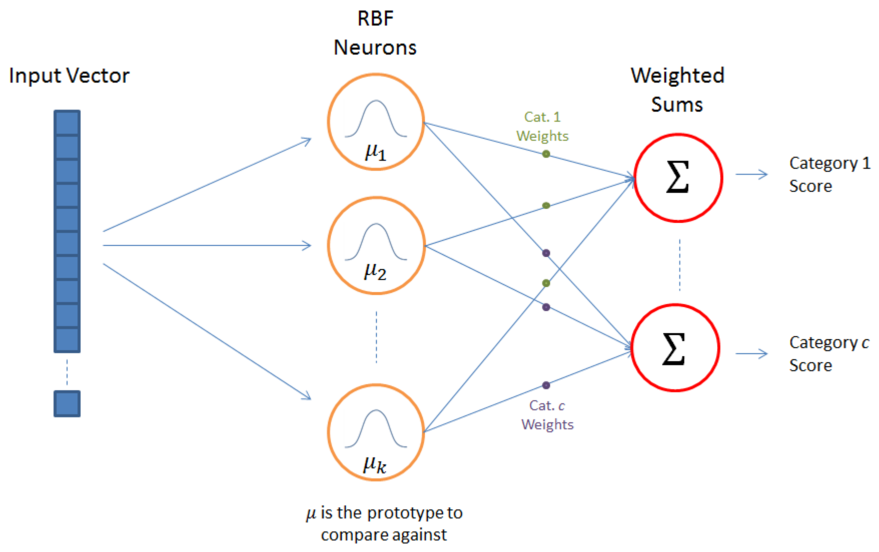


Figure 1.8: Radial basis function Neural Network:

1.6.3 Kohonen Self Organizing Neural Network

The objective of a Kohonen map is to input vectors of arbitrary dimension to discrete map comprised of neurons. The map needs to be trained to create its own organization of the training data. It comprises either one or two dimensions. When training the map the location of the neuron remains constant but the weights differ depending on the value. This self-organization process has different parts, in the first phase, every neuron value is initialized with a small weight and the input vector. In the second phase, the neuron closest to the point is the 'winning neuron' and the neurons connected to the winning neuron will also move towards the point like in the graphic below. The distance between the point and the neurons is calculated by the euclidean distance, the neuron with the least distance wins. Through the iterations, all the points are clustered and each neuron represents each kind of cluster. This is the gist behind the organization of Kohonen Neural Network.

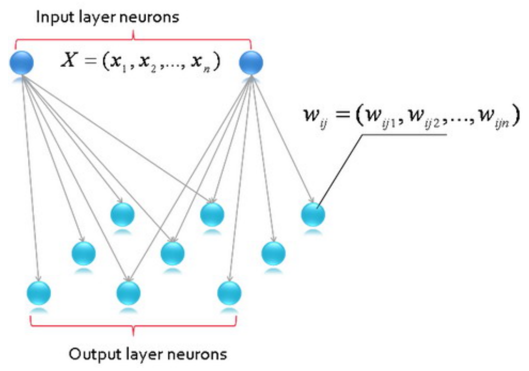

Input layer neurons
$X = (x_1, x_2, \dots, x_n)$
$w_{ij} = (w_{ij1}, w_{ij2}, \dots, w_{ijn})$
Output layer neurons

Figure 1.9: Kohonen Self Organizing Neural Network

1.6.4 Recurrent Network(RNN)

Recurrent nets are a powerful set of artificial neural network algorithms especially useful for processing sequential data such as sound, time series (sensor) data or written the natural language. A version of recurrent networks was used by DeepMind in their work playing video games with autonomous agents.
Recurrent nets differ from feedforward nets because they include a feedback loop, whereby the the output from step n-1 is fed back to the net to affect the outcome of step n, and so forth for each subsequent step. For example, if a net is exposed to a word letter by letter, and it is asked to guess each following letter, the first letter of a word will help determine what a recurrent net thinks the second letter will be, etc.
This differs from a feedforward network, which learns to classify each handwritten numeral of MNIST independently according to the pixels it is exposed to form a single example, without referring to the preceding example to adjust its predictions. Feedforward networks accept one input at a time and produce one output. Recurrent nets don't face the same one-to-one constraint.
While some forms of data, like images, do not seem to be sequential, they can be understood as sequences when fed into a recurrent net. Consider an image of a handwritten word. Just as recurrent nets process handwriting, converting each cursive image into a letter, and using the beginning of a word to guess how that word will end, so nets can treat part of any image like letters in a sequence. A neural net roving over a large picture may learn from each region what the neighboring regions are more likely to be.
Recurrent nets and feedforward nets both "remember" something about the world, in a loose sense, by modeling the data they are exposed to. But they remember in very different ways. After training, feedforward net produces a static model of the data it has been shown, and that model can then accept new examples and accurately classify or cluster them.
In contrast, recurrent nets produce dynamic models – i.e. models that change over time – in ways that yield accurate classifications dependent on the context of the examples they're exposed to.
To be precise, recurrent models include the hidden state that determined the previous classification in a series. In each subsequent step, that hidden state is combined with the new step's input data to produce a new hidden state and then a new classification. Each hidden state is recycled to produce its modified successor.
Human memories are also context-aware, recycling an awareness of previous states to properly interpret new data. a hidden state affected by their short-term memories and preceding sensations.
Different short-term memories should be recalled at different times, in order to assign the right meaning to the current input. Some of those memories will have been forged recently, and other memories will have been forged many time steps before they are needed. The recurrent net that effectively associates memories and inputs remote in time is called a Long Short-Term Memory (LSTM), as much as that sounds like an oxymoron.

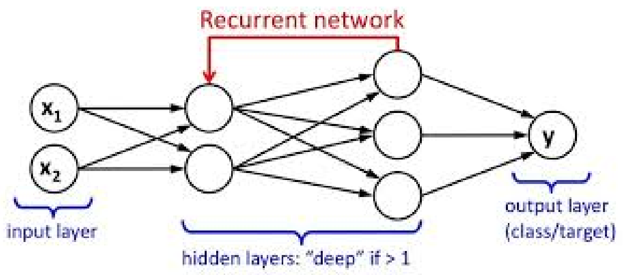


Figure 1.10: Recurrent Network(RNN)

1.6.5 Convolutional Neural Network

Convolutional neural networks are similar to feedforward neural networks, where the neurons have learn-able weights and biases. Its application has been in signal and image processing which takes over OpenCV in the field of computer vision.
Below is a representation of a ConvNet, in this neural network, the input features are taken in batch wise like a filter. This will help the network to remember the images in parts and can compute the operations. These computations involve the conversion of the image from RGB or HSI scale to Gray-scale. Once we have this, the changes in the pixel value will help in to detect the edges and images can be classified into different categories.
ConvNet are applied in techniques like signal processing and image classification techniques. Computer vision techniques are dominated by convolutional neural networks because of their accuracy in image classification. The technique of image analysis and recognition, where the agriculture and weather features are extracted from the open source satellites like LSAT to predict the future growth and yield of a particular land are being implemented.

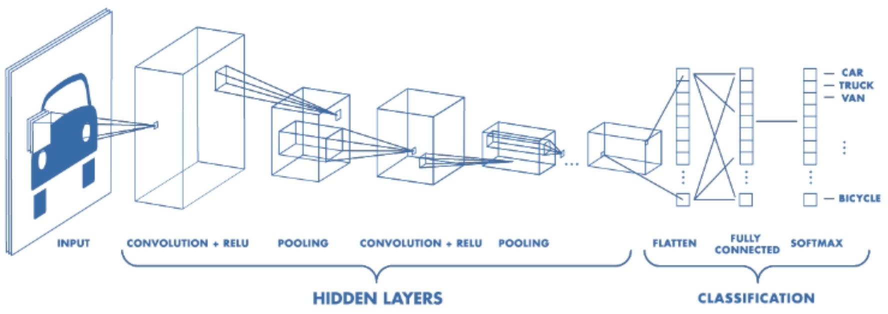


Figure 1.11: CNN

1.6.6 Modular Neural Network

Modular Neural Networks have a collection of different networks working independently and contributing towards the output. Each neural network has a set of inputs which are unique compared to other networks constructing and performing sub-tasks. These networks do not interact or signal each other in accomplishing the tasks. The advantage of a modular neural network is that it breakdowns a large computational process into smaller components decreasing the complexity. This breakdown will help in decreasing the number of connections and negates the interaction of these network with each other, which in turn will increase the computation speed. However, the processing time will depend on the number of neurons and their involvement in computing the results.

Modular Neural Networks (MNNs) is a rapidly growing field in artificial Neural Networks research. This paper surveys the different motivations for creating MNNs: biological, psychological, hardware, and computational. Then, the general stages of MNN design are outlined and surveyed as well, viz., task decomposition techniques, learning schemes, and multi-module decision-making strategies.

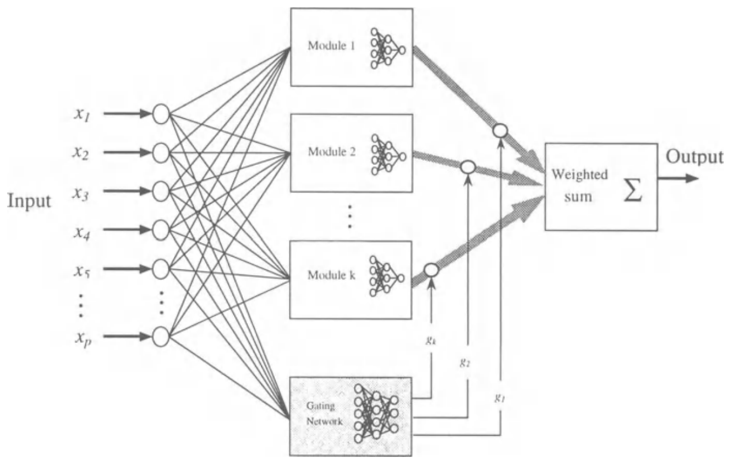


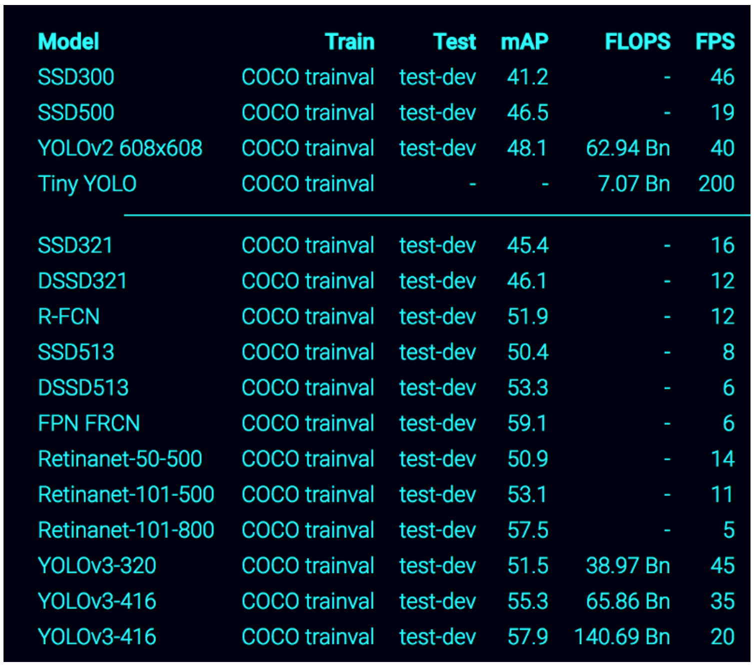
Figure 1.12: Modular Neural Network

Chapter 2

YOLO

2.1 YOLO and its versions

You only look once (YOLO) is an object detection system targeted for real-time processing. Here is the accuracy and speed comparison provided by the YOLO web site.



Model	Train	Test	mAP	FLOPS	FPS
SSD300	COCO trainval	test-dev	41.2	-	46
SSD500	COCO trainval	test-dev	46.5	-	19
YOLOv2 608x608	COCO trainval	test-dev	48.1	62.94 Bn	40
Tiny YOLO	COCO trainval	-	-	7.07 Bn	200
SSD321	COCO trainval	test-dev	45.4	-	16
DSSD321	COCO trainval	test-dev	46.1	-	12
R-FCN	COCO trainval	test-dev	51.9	-	12
SSD513	COCO trainval	test-dev	50.4	-	8
DSSD513	COCO trainval	test-dev	53.3	-	6
FPN FRCN	COCO trainval	test-dev	59.1	-	6
Retinanet-50-500	COCO trainval	test-dev	50.9	-	14
Retinanet-101-500	COCO trainval	test-dev	53.1	-	11
Retinanet-101-800	COCO trainval	test-dev	57.5	-	5
YOLOv3-320	COCO trainval	test-dev	51.5	38.97 Bn	45
YOLOv3-416	COCO trainval	test-dev	55.3	65.86 Bn	35
YOLOv3-416	COCO trainval	test-dev	57.9	140.69 Bn	20

Figure 2.1: Comparison between yolo, its versions and other pre-trained detection networks

2.2 How it works.

YOLO divides the input image into an SxS grid. Each grid cell predicts only one object. For example, the yellow grid cell below tries to predict the “person” object whose center (the blue dot) falls inside the grid cell.

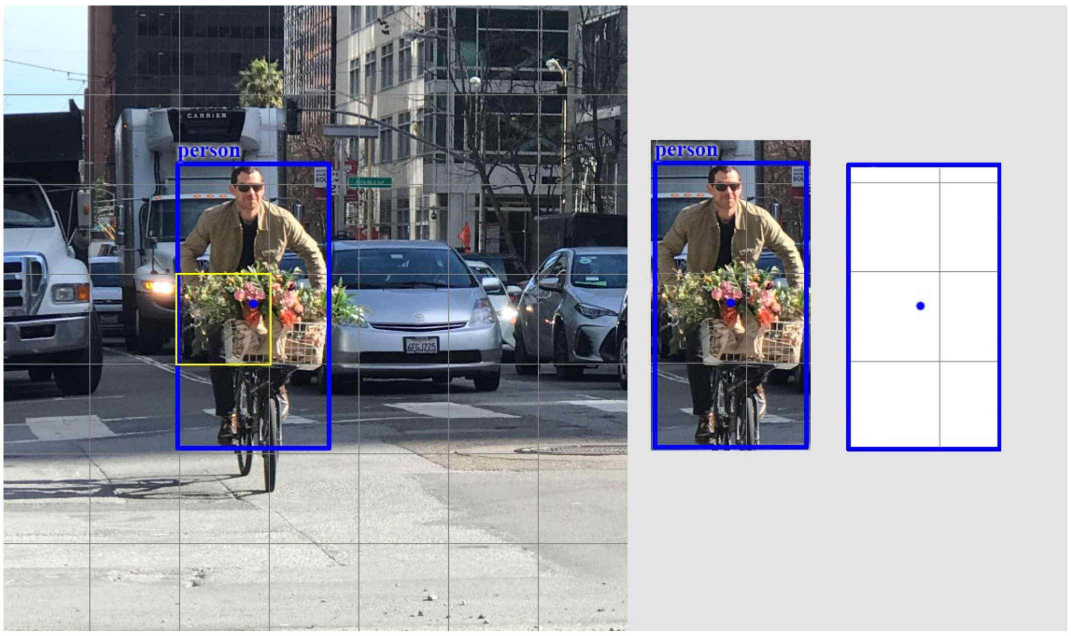


Figure 2.2: Each grid cell detects only one object.

Each grid cell predicts a fixed number of boundary boxes. In this example, the yellow grid cell makes two boundary box predictions (blue boxes) to locate where the person is.

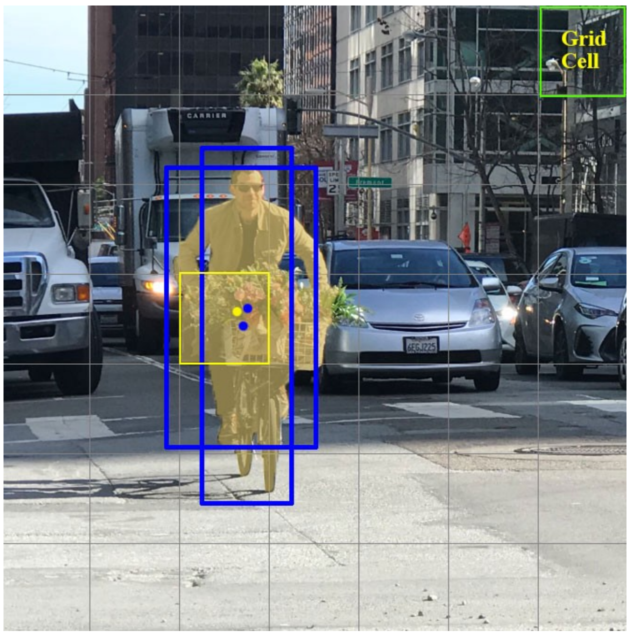


Figure 2.3: Each grid cell make a fixed number of boundary box guesses for the object.

However, the one-object rule limits how close detected objects can be. For that, YOLO does have some limitations on how close objects can be. For the picture below, there are 9 Santas in the lower left corner but YOLO can detect 5 only.

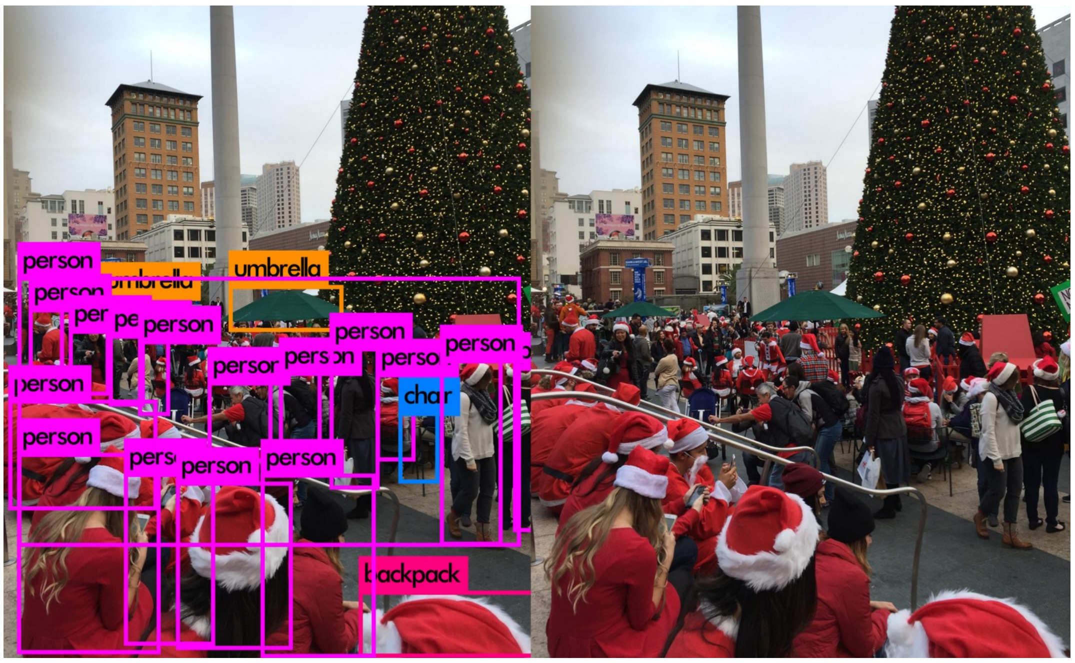


Figure 2.4: YOLO may miss objects that are too close.

For each grid cell,
• it predicts B boundary boxes and each box has one box confidence score,
• it detects one object only regardless of the number of boxes B,
• it predicts C conditional class probabilities (one per class for the likeliness of the object class).

To evaluate PASCAL VOC, YOLO uses 7x7 grids (SxS), 2 boundary boxes (B) and 20 classes (C).

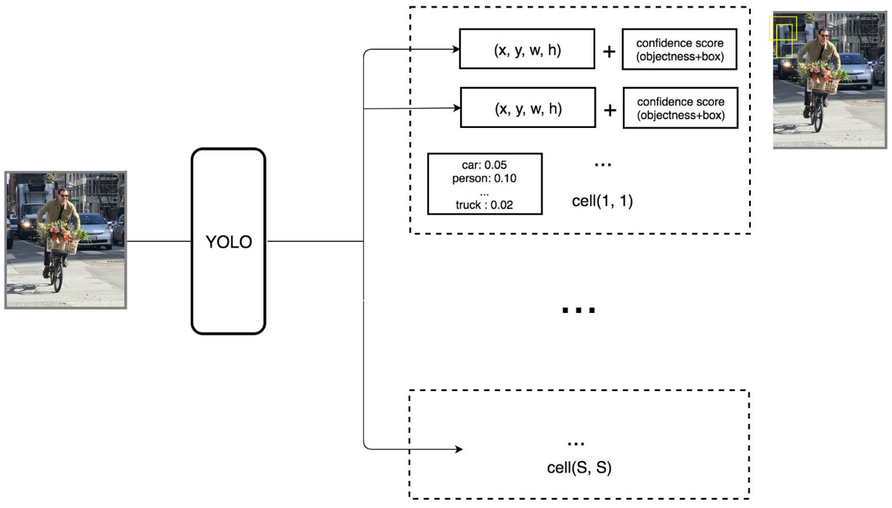


Figure 2.5: YOYO make SxS predictions with B boundary boxes.

Each boundary box contains 5 elements: (x, y, w, h) and a box confidence score. The confidence score reflects how likely the box contains an object (objectness) and how accurate is the boundary box. YOLO normalize the bounding box width w and height h by the image width and height. x and y are offsets to the corresponding cell. Hence, x, y, w, and h are all between 0 and 1. Each cell has 20 conditional class probabilities. The conditional class probability is the probability that the detected object belongs to a particular class (one probability per category for each cell). So, YOLO's prediction has a shape of (S, S, Bx5 + C) = (7, 7, 2x5 + 20) = (7, 7, 30).

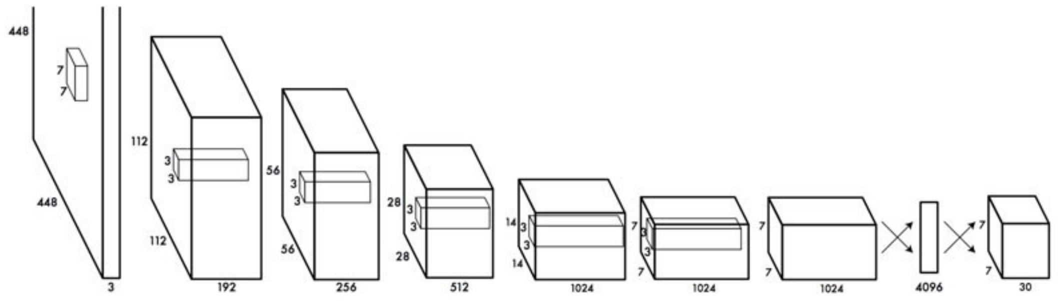


Figure 2.6: Layers of YOLO

The major concept of YOLO is to build a CNN network to predict a (7, 7, 30) tensor. It uses a CNN network to reduce the spatial dimension to 7x7 with 1024 output channels at each location. YOLO performs a linear regression using two fully connected layers to make 7x7x2 boundary box predictions (the middle picture below). To make a final prediction, we keep those with high box confidence scores (greater than 0.25) as our final predictions (the right picture).

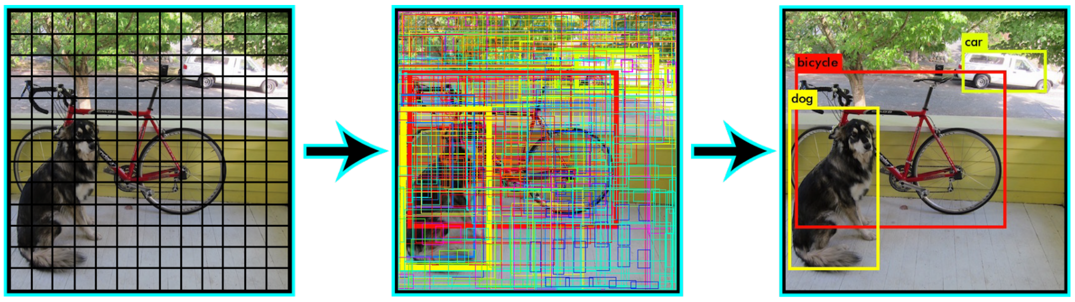
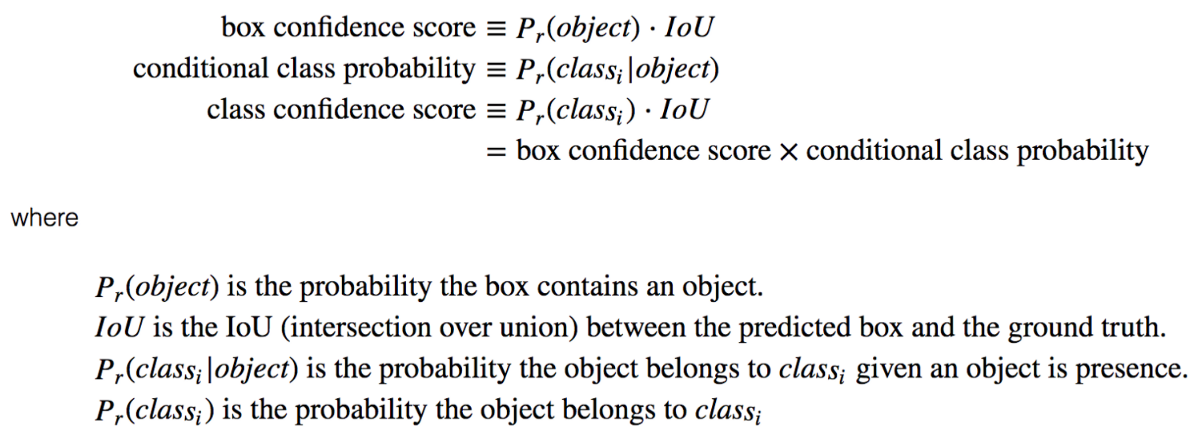


Figure 2.7: Detection steps

The class confidence score for each prediction box is computed as:

$$\text{class confidence score} = \text{box confidence score} \times \text{conditional class probability}$$

It measures the confidence in both the classification and the localization (where an object is located).



$$\begin{aligned}\text{box confidence score} &\equiv P_r(object)\cdot IoU\\ \text{conditional class probability} &\equiv P_r(class_i|object)\\ \text{class confidence score} &\equiv P_r(class_i)\cdot IoU\\ &= \text{box confidence score}\times\text{conditional class probability}\end{aligned}$$

where

$P_r(object)$ is the probability the box contains an object.
IoU is the IoU (intersection over union) between the predicted box and the ground truth.
$P_r(class_i|object)$ is the probability the object belongs to $class_i$ given an object is presence.
$P_r(class_i)$ is the probability the object belongs to $class_i$

Figure 2.8: the mathematical definitions for some expresions

2.3 Network design

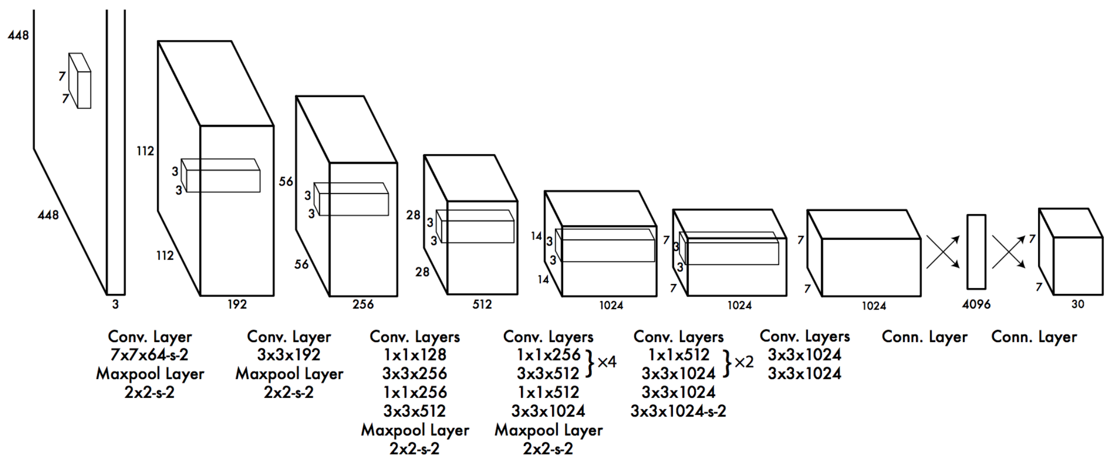


Figure 2.9: Network design

YOLO has 24 convolutional layers followed by 2 fully connected layers (FC). Some convolution layers use 1x1 reduction layers alternatively to reduce the depth of the features maps. For the last convolution layer, it outputs a tensor with shape (7, 7, 1024). The tensor is then flattened. Using 2 fully connected layers as a form of linear regression, it outputs 7x7x30 parameters and then reshapes to (7, 7, 30), i.e. 2 boundary box predictions per location.
A faster but less accurate version of YOLO, called Fast YOLO, uses only 9 convolutional layers with shallower feature maps.

2.4 Loss function

YOLO predicts multiple bounding boxes per grid cell. To compute the loss for the true positive, YOLO only wants one of them to be responsible for the object. For this purpose, YOLO selects the one with the highest IoU (intersection over union) with the ground truth. This strategy leads to specialization among the bounding box predictions. Each prediction gets better at predicting certain sizes and aspect ratios.

YOLO uses sum-squared error between the predictions and the ground truth to calculate loss. The loss function composes of:
• the classification loss.
• the localization loss (errors between the predicted boundary box and the ground truth).
• the confidence loss (the objectness of the box).

2.4.1 Classification loss

If an object is detected, the classification loss at each cell is the squared error of the class conditional probabilities for each class:

$$\sum_{i=0}^{S^2} \mathbb{1}_i^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2$$

where

$\mathbb{1}_i^{obj} = 1$ if an object appears in cell i, otherwise 0.

$\hat{p}_i(c)$ denotes the conditional class probability for class c in cell i.

Figure 2.10: Classification loss

2.4.2 Localization loss

The localization loss measures the errors in the predicted boundary box locations and sizes. We only count the box responsible for detecting the object.

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$$
$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i}\right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i}\right)^2 \right]$$

where

$\mathbb{1}_{ij}^{obj} = 1$ if the j th boundary box in cell i is responsible for detecting the object, otherwise 0.

λ_{coord} increase the weight for the loss in the boundary box coordinates.

Figure 2.11: Localization loss

YOLO does not want to weight absolute errors in large boxes and small boxes equally. i.e. a 2-pixel error in a large box is the same for a small box. To partially address this, YOLO predicts the square root of the bounding box width and height instead of the width and height. In addition, to put more emphasis on the boundary box accuracy, YOLO multiplies the loss by λ coord (default= 5).

2.4.3 Confidence loss

If an object is detected in the box, the confidence loss (measuring the objectness of the box) is:

$$\sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2$$

where

$\hat{C}_i$ is the box confidence score of the box j in cell i.

$\mathbb{1}_{ij}^{obj} = 1$ if the j th boundary box in cell i is responsible for detecting the object, otherwise 0.

Figure 2.12: The confidence loss in case of an object was detected

If an object is not detected in the box, the confidence loss is:

$$\lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2$$

where

$\mathbb{1}_{ij}^{noobj}$ is the complement of $\mathbb{1}_{ij}^{obj}$.

$\hat{C}_i$ is the box confidence score of the box j in cell i.

λ_{noobj} weights down the loss when detecting background.

Figure 2.13: The confidence loss in case of an object was not detected

Most boxes do not contain any objects. This causes a class imbalance problem, i.e. we train the model to detect background more frequently than detecting objects. To remedy this, we weight this loss down by a factor λnoobj (default= 0.5).

2.4.4 Loss

The final loss adds localization, confidence and classification losses together.

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

Figure 2.14: Loss

2.5 Inference: Non-maximal suppression

YOLO can make duplicate detections for the same object. To fix this, YOLO applies non-maximal suppression to remove duplications with lower confidence. Non-maximal suppression adds 2- 3% in mAP.
Here is one of the possible non-maximal suppression implementation:
• Sort the predictions by the confidence scores.
• Start from the top scores, ignore any current prediction if we find any previous predictions that have the same class and IoU ¿ 0.5 with the current prediction.
• Repeat step 2 until all predictions are checked.

2.6 Benefits of YOLO

• Fast and Good for real-time processing.
• Predictions (object locations and classes) are made from one single network. Can be trained end-to-end to improve accuracy.
• YOLO is more generalized. It outperforms other methods when generalizing from natural images to other domains like artwork.

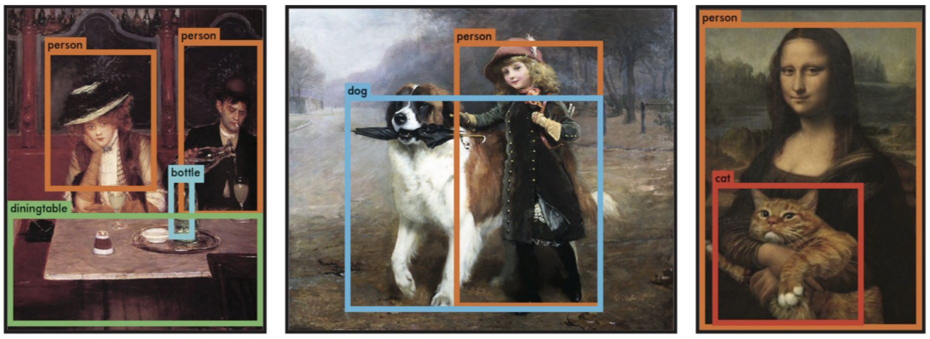


Figure 2.15: YOLO predictions for some artwork

• Region proposal methods limit the classifier to the specific region. YOLO accesses to the whole image in predicting boundaries. With the additional context, YOLO demonstrates fewer false positives in background areas.
•YOLO detects one object per grid cell. It enforces spatial diversity in making predictions.

2.7 YOLO version 2

SSD is a strong competitor for YOLO which at one point demonstrates higher accuracy for real-time processing. Comparing with region-based detectors, YOLO has higher localization errors and the recall (measure how good to locate all objects) is lower. YOLOv2 is the second version of the YOLO with the objective of improving the accuracy significantly while making it faster.

2.7.1 Accuracy improvements

1-Batch normalization

Add batch normalization in convolution layers. This removes the need for dropouts and pushes mAP up 2%.

2-High-resolution classifier

The YOLO training composes of 2 phases. First, YOLO trains a classifier network like VGG16. Then replaces the fully connected layers with a convolution layer and retrains it end-to-end for the object detection. YOLO trains the classifier with 224 x 224 pictures followed by 448 x 448 pictures for the object detection. YOLOv2 starts with 224 x 224 pictures for the classifier training but then retune the classifier again with 448 x 448 pictures using much fewer epochs. This makes the detector training easier and moves mAP up by 4%.

3-Convolutional with Anchor Boxes

As indicated in the YOLO paper, the early training is susceptible to unstable gradients. Initially, YOLO makes arbitrary guesses on the boundary boxes. These guesses may work well for some objects but badly for others resulting in steep gradient changes. In early training, predictions are fighting with each other on what shapes to specialize on.

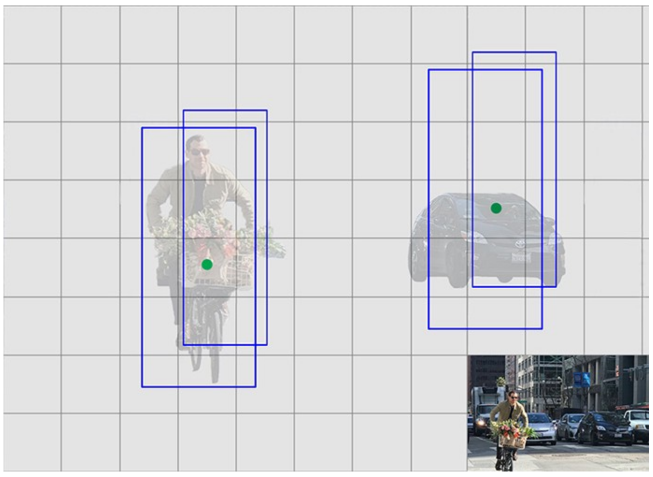


Figure 2.16: Non-diverse boxes

In the real-life domain, the boundary boxes are not arbitrary. Cars have very similar shapes and pedestrians have an approximate aspect ratio of 0.41.

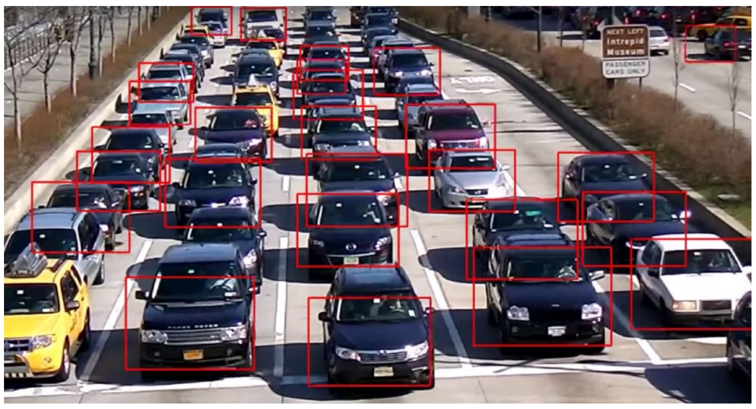


Figure 2.17: Boxes of cars are almost the same

Since we only need one guess to be right, the initial training will be more stable if YOLO starts with diverse guesses that are common for real-life objects.

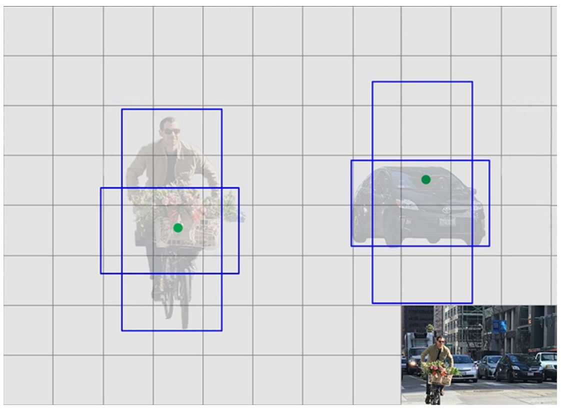


Figure 2.18: Anchors

For example, we can create 5 anchor boxes with the following shapes.

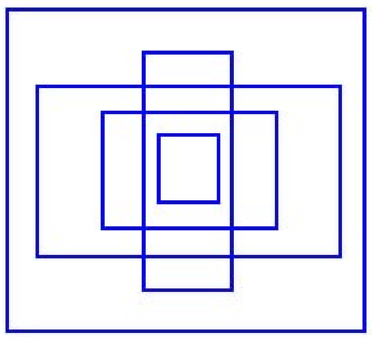


Figure 2.19: 5 shapes of anchors

Instead of predicting 5 arbitrary boundary boxes, YOLO predicts offsets to each of the anchor boxes above. If YOLO constrains the offset values, YOLO can maintain the diversity of the predictions and have each prediction focuses on a specific shape. So the initial training will be more stable.In the paper, anchors are also called priors.

Here are the changes YOLO makes to the network:

• Remove the fully connected layers responsible for predicting the boundary box.

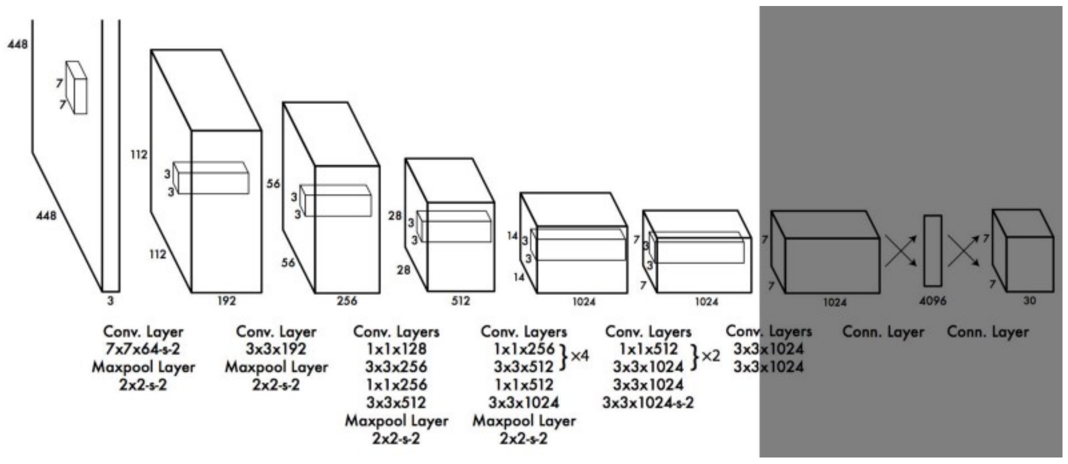

448
7
7
448
3
112
3
3
112
192
56
3
3
56
256
28
3
3
28
512
14
3
3
14
1024
7
3
3
7
1024
7
7
1024
4096
7
7
30
Conv. Layer 7x7x64-s-2 Maxpool Layer 2x2-s-2
Conv. Layer 3x3x192 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x128 3x3x256 1x1x256 3x3x512 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x256 3x3x512 }×4 1x1x512 3x3x1024 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x512 3x3x1024 }×2 3x3x1024 3x3x1024-s-2
Conv. Layers 3x3x1024 3x3x1024
Conn. Layer
Conn. Layer

Figure 2.20: Edits were done to the netowrk in YOLOv2

• YOLO moves the class prediction from the cell level to the boundary box level. Now, each prediction includes 4 parameters for the boundary box, 1 box confidence score (objectness) and 20 class probabilities. i.e. 5 boundary boxes with 25 parameters: 125 parameters per grid cell. Same as YOLO, the objectness prediction still predicts the IOU of the ground truth and the proposed box.

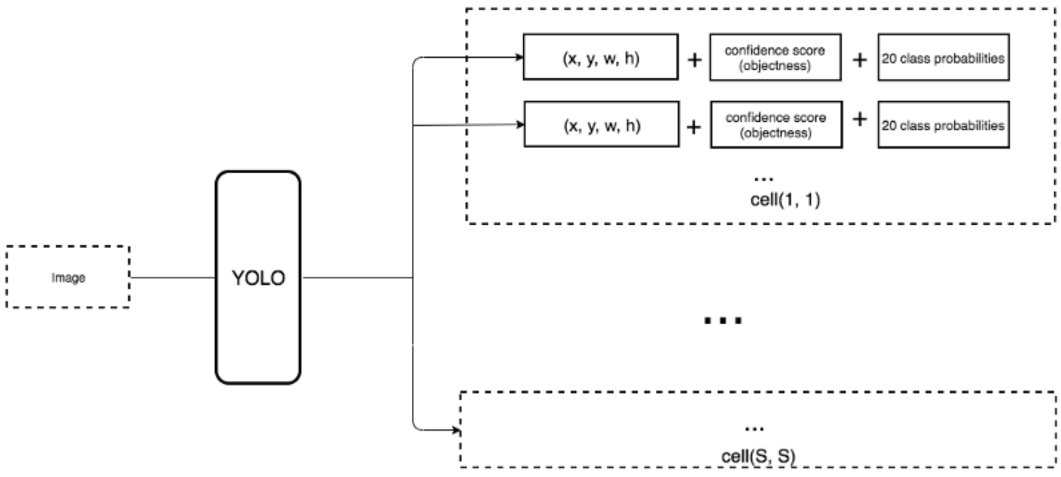


Figure 2.21: New sequence of detections in YOLO v2

• To generate predictions with a shape of 7 x 7 x 125, we replace the last convolution layer with three 3 x 3 convolutional layers each outputting 1024 output channels. Then we apply a final 1 x 1 convolutional layer to convert the 7 x 7 x 1024 output into 7 x 7 x 125.
• Change the input image size from 448 x 448 to 416 x 416. This creates an odd number spatial dimension (7x7 v.s. 8x8 grid cell). The center of a picture is often occupied by a large object. With an odd number grid cell, it is more certain on where the object belongs.

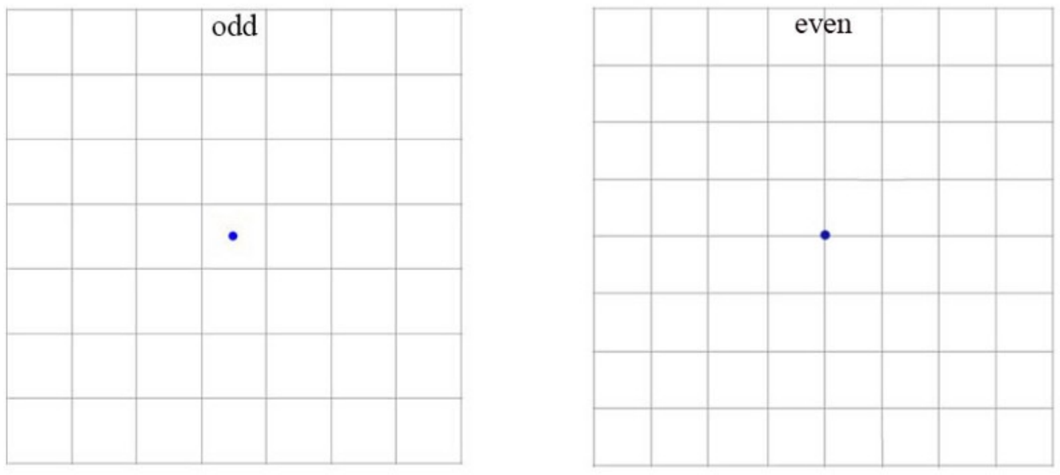


Figure 2.22: Odd number spatial dimension

• Remove one pooling layer to make the spatial output of the network to 13x13 (instead of 7x7).

Anchor boxes decrease mAP slightly from 69.5 to 69.2 but the recall improves from 81% to 88%. i.e. even the accuracy is slightly decreased but it increases the chances of detecting all the ground truth objects.

Dimension Clusters

In many problem domains, the boundary boxes have strong patterns. For example, in the autonomous driving, the 2 most common boundary boxes will be cars and pedestrians at different distances. To identify the top-K boundary boxes that have the best coverage for the training data, YOLO runs K-means clustering on the training data to locate the centroids of the top-K clusters.

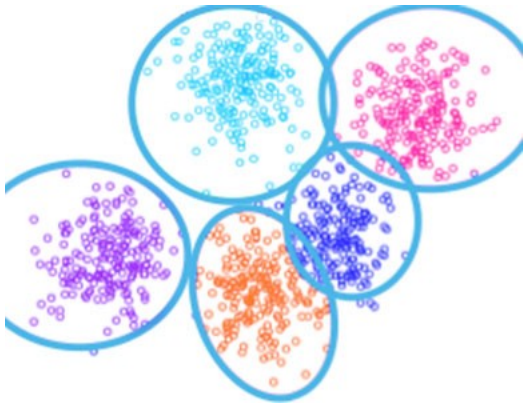
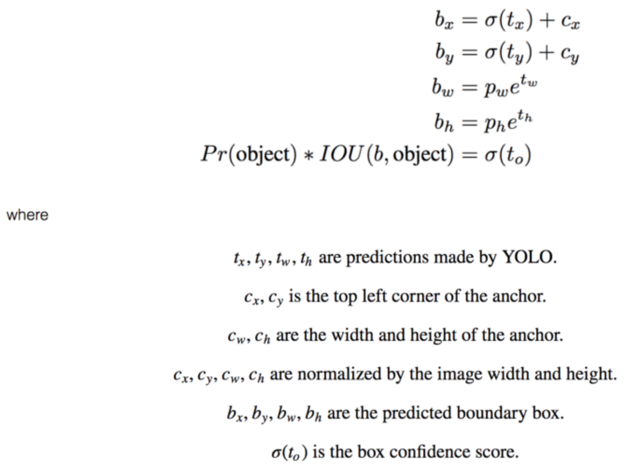


Figure 2.23: k-means cluster

Direct location prediction

YOLO makes predictions on the offsets to the anchors. Nevertheless, if it is unconstrained, YOLO's guesses will be randomized again. YOLO predicts 5 parameters (tx, ty, tw, th, and to) and applies the sigma function to constraint its possible offset range.



$$
\begin{aligned}
b_x &= \sigma(t_x) + c_x \\
b_y &= \sigma(t_y) + c_y \\
b_w &= p_w e^{t_w} \\
b_h &= p_h e^{t_h} \\
Pr(\text{object}) * IOU(b, \text{object}) &= \sigma(t_o)
\end{aligned}
$$

where

t_x, t_y, t_w, t_h are predictions made by YOLO.

c_x, c_y is the top left corner of the anchor.

c_w, c_h are the width and height of the anchor.

c_x, c_y, c_w, c_h are normalized by the image width and height.

b_x, b_y, b_w, b_h are the predicted boundary box.

$\sigma(t_o)$ is the box confidence score.

Figure 2.24: Direct location predections

Here is the visualization. The blue box below is the predicted boundary box and the dotted rectangle is the anchor.

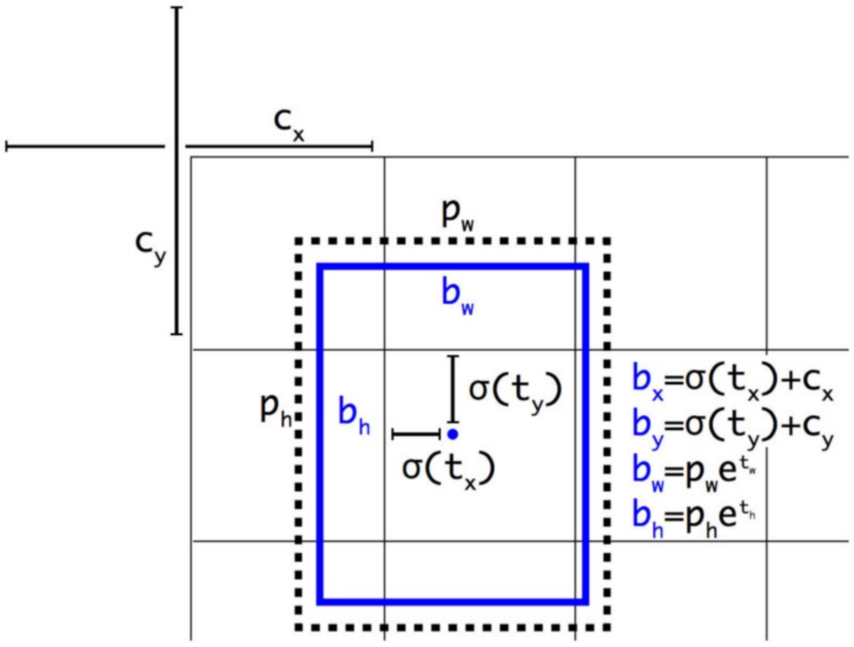


Figure 2.25: Visualizing of the direct location predections

With the use of k-means clustering (dimension clusters) and the improvement mentioned in this section, mAP increases 5%.

Fine-Grained Features

Convolution layers decrease the spatial dimension gradually. As the corresponding resolution decreases, it is harder to detect small objects. Other object detectors like SSD locate objects from different layers of feature maps. So each layer specializes at a different scale. YOLO adopts a different approach called passthrough. It reshapes the 28 x 28 x 512 layer to 14 x 14 x 2048. Then it concatenates with the original 14 x 14 x1024 output layer. Now we apply convolution filters on the new 14 x 14 x 3072 layer to make predictions.

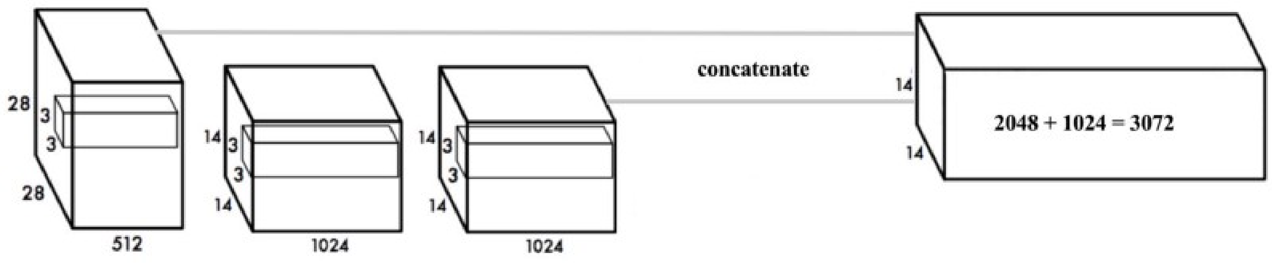


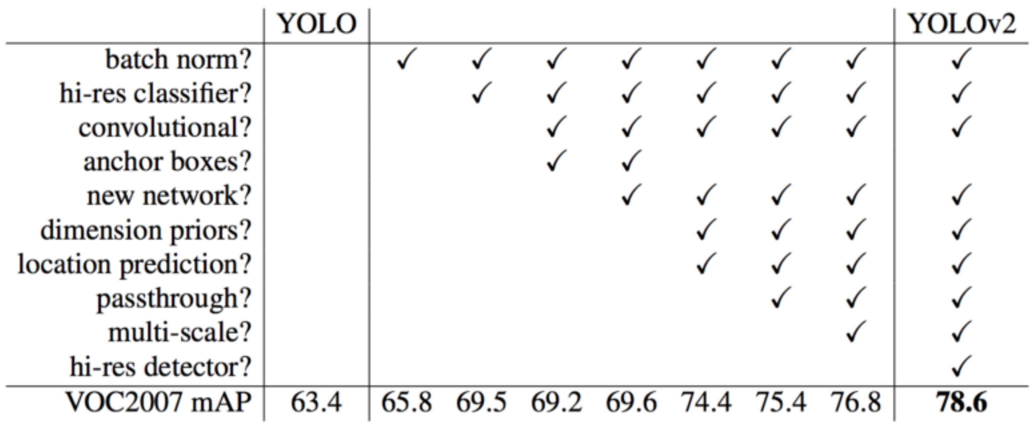
Figure 2.26: Fine-Grained features

Multi-Scale Training

After removing the fully connected layers, YOLO can take images of different sizes. If the width and height are doubled, YOLO is just making 4x output grid cells and therefore 4x predictions. Since the YOLO network downsamples the input by 32, YOLO just needs to make sure the width and height is a multiple of 32. During training, YOLO takes images of size 320x320, 352x352, ... and 608x608 (with a step of 32). For every 10 batches, YOLOv2 randomly selects another image size to train the model. This acts as data augmentation and forces the network to predict well for different input image dimension and scale. In additional, YOLO can use lower resolution images for object detection at the cost of accuracy. This can be a good trade-off for speed on low GPU power devices. At 288 x 288, YOLO runs at more than 90 FPS with mAP almost as good as Fast R-CNN. At high-resolution YOLO achieves 78.6 mAP on VOC 2007.

Accuracy

Here are the accuracy improvements after applying the techniques discussed so far:



	YOLO								YOLOv2
batch norm?		✓	✓	✓	✓	✓	✓	✓	✓
hi-res classifier?			✓	✓	✓	✓	✓	✓	✓
convolutional?				✓	✓	✓	✓	✓	✓
anchor boxes?				✓	✓				
new network?					✓	✓	✓	✓	✓
dimension priors?						✓	✓	✓	✓
location prediction?						✓	✓	✓	✓
passthrough?							✓	✓	✓
multi-scale?								✓	✓
hi-res detector?									✓
VOC2007 mAP	63.4	65.8	69.5	69.2	69.6	74.4	75.4	76.8	**78.6**

Figure 2.27: Accuracy comparison

2.7.2 Training

YOLO is trained with the ImageNet 1000 class classification dataset in 160 epochs: using stochastic gradient descent with a starting learning rate of 0.1, polynomial rate decay with a power of 4, weight decay of 0.0005 and momentum of 0.9. In the initial training, YOLO uses 224 x 224 images, and then retune it with 448 x 448 images for 10 epochs at a learning rate of $10^{-}3$. After the training, the classifier achieves a top-1 accuracy of 76.5% and a top-5 accuracy of 93.3 %.
Then the fully connected layers and the last convolution layer is removed for a detector. YOLO adds three 3 x 3 convolutional layers with 1024 filters each followed by a final 1 x 1 convolutional layer with 125 output channels. (5 box predictions each with 25 parameters) YOLO also add a passthrough layer. YOLO trains the network for 160 epochs with a starting learning rate of $10^{-}3$, dividing it by 10 at 60 and 90 epochs. YOLO uses a weight decay of 0.0005 and momentum of 0.9.

2.7.3 Classification

Datasets for object detection have far fewer class categories than those for classification. To expand the classes that YOLO can detect, YOLO proposes a method to mix images from both detection and classification datasets during training. It trains the end-to-end network with the object detection samples while backpropagates the classification loss from the classification samples to train the classifier path. This approach encounters a few challenges:
• How do we merge class labels from different datasets? In particular, object detection datasets and different classification datasets uses different labels.
• Any merged labels may not be mutually exclusive, for example, Norfolk terrier in ImageNet and dog in COCO. Since it is not mutually exclusive, we can not use softmax to compute the probability.

Hierarchical classification

YOLO combines labels in different datasets to form a tree-like structure WordTree. The children form an is-a relationship with its parent like biplane is a plane. But the merged labels are now not mutually exclusive.

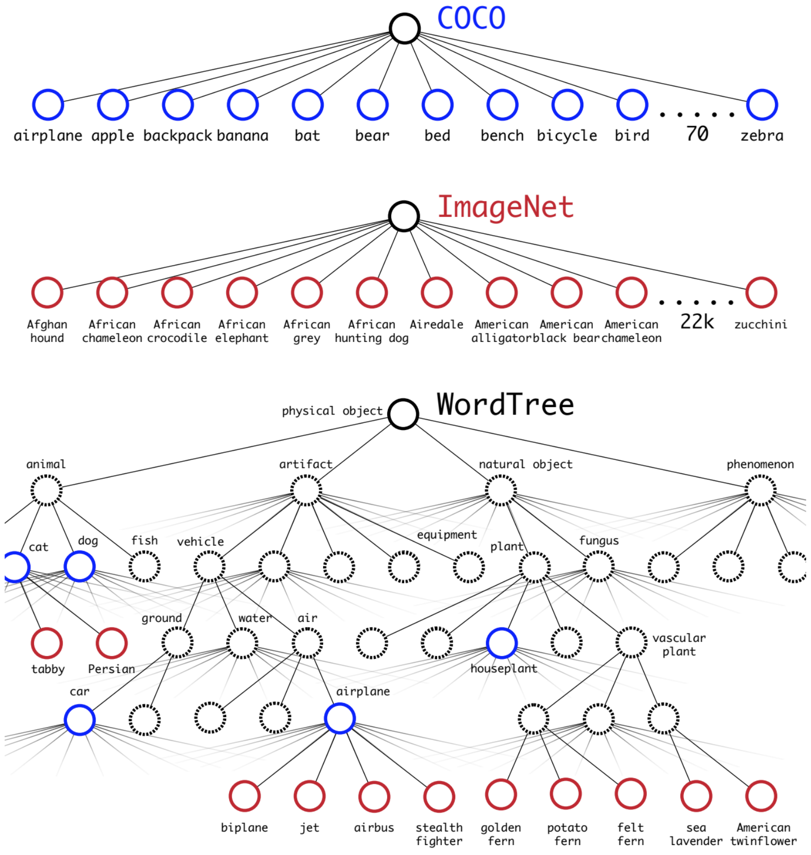

COCO
airplane apple backpack banana bat bear bed bench bicycle bird 70 zebra
ImageNet
Afghan hound African chameleon African crocodile African elephant African grey African hunting dog Airedale American alligator American black bear American chameleon 22k zucchini
WordTree
physical object
animal artifact natural object phenomenon
cat dog fish vehicle equipment plant fungus
tabby Persian ground water air houseplant vascular plant
car airplane
biplane jet airbus stealth fighter golden fern potato fern felt fern sea lavender American twinflower

Figure 2.28: Combining COCO and ImageNet labels to a hierarchical WordTree

Using the 1000 class ImageNet. Instead of predicting 1000 labels in a flat structure, YOLO creates the corresponding WordTree which has 1000 leave nodes for the original labels and 369 nodes for their parent classes. Originally, YOLO predicts the class score for the biplane. But with the WordTree, it now predicts the score for the biplane given it is an airplane.

When YOLO sees a classification image, it only back propagates classification loss to train the classifier. YOLO finds the bounding box that predicts the highest probability for that class and it computes the classification loss as well as those from the parents. (If an object is labeled as a biplane, it is also considered to be labeled as airplane, air, vehicle…) This encourages the model to extract features common to them. So even YOLO has never trained a specific class of objects for object detection, it can still make such predictions by generalizing predictions from related objects.

2.8 YOLO 9000

YOLO9000 extends YOLO to detect objects over 9000 classes using hierarchical classification with a 9418 node WordTree. It combines samples from COCO and the top 9000 classes from the ImageNet. YOLO samples four ImageNet data for every COCO data. It learns to find objects using the detection data in COCO and to classify these objects with ImageNet samples.
During the evaluation, YOLO test images on categories that it knows how to classify but not trained directly to locate them, i.e. categories that do not exist in COCO. YOLO9000 evaluates its result from the ImageNet object detection dataset which has 200 categories. It shares about 44 categories with COOC. Therefore, the dataset contains 156 categories that have never been trained directly on how to locate them. YOLO extracts similar features for related object types. Hence, YOLO can detect those 156 categories by simply from the feature values.
YOLO9000 gets 19.7 mAP overall with 16.0 mAP on those 156 categories. YOLO9000 performs well with new species of animals not found in COCO because their shapes can be generalized easily from their parent classes. However, COCO does not have bounding box labels for any type of clothing so the test struggles with categories like "sunglasses".

2.9 YOLOv3

2.9.1 Class prediction

Most classifiers assume output labels are mutually exclusive. It is true if the output is mutually exclusive object classes. Therefore, YOLO applies a softmax function to convert scores into probabilities that sum up to one. YOLOv3 uses multi-label classification. For example, the output labels may be "pedestrian" and "child" which are not non-exclusive. (the sum of output can be greater than 1 now.) YOLOv3 replaces the softmax function with independent logistic classifiers to calculate the likeliness of the input belongs to a specific label. Instead of using mean square error in calculating the classification loss, YOLOv3 uses binary cross-entropy loss for each label. This also reduces the computation complexity by avoiding the softmax function.

2.9.2 Bounding box prediction and cost function calculation

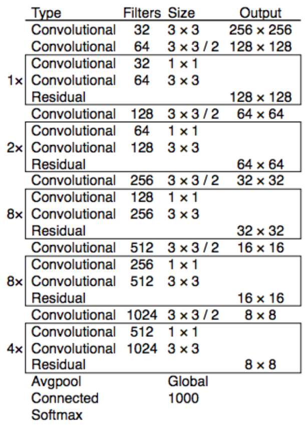
YOLOv3 predicts an objectness score for each bounding box using logistic regression. YOLOv3 changes the way in calculating the cost function. If the bounding box prior (anchor) overlaps a ground truth object more than others, the corresponding objectness score should be 1. For other priors with overlap greater than a predefined threshold (default 0.5), they incur no cost. Each ground truth object is associated with one boundary box prior only. If a bounding box prior is not assigned, it incurs no classification and localization lost, just confidence loss on objectness. We use tx and ty (instead of bx and by) to compute the loss.

2.9.3 Feature Pyramid Networks (FPN) like Feature Pyramid

YOLOv3 makes 3 predictions per location. Each prediction composes of a boundary box, a objectness and 80 class scores, i.e. N x N x [3 x $(4 + 1 + 80)$] predictions.
YOLOv3 makes predictions at 3 different scales (similar to the FPN):
1- In the last feature map layer.
2- Then it goes back 2 layers back and upsamples it by 2. YOLOv3 then takes a feature map with higher resolution and merge it with the upsampled feature map using element-wise addition. YOLOv3 apply convolutional filters on the merged map to make the second set of predictions.
3- Repeat 2 again so the resulted feature map layer has good high-level structure (semantic) information and good resolution spatial information on object locations.

2.9.4 Feature extractor

A new 53-layer Darknet-53 is used to replace the Darknet-19 as the feature extractor. Darknet-53 mainly composed of 3 x 3 and 1x1 filters with skip connections like the residual network in ResNet. Darknet-53 has less BFLOP (billion floating point operations) than ResNet-152 but achieves the same classification accuracy at 2x faster.

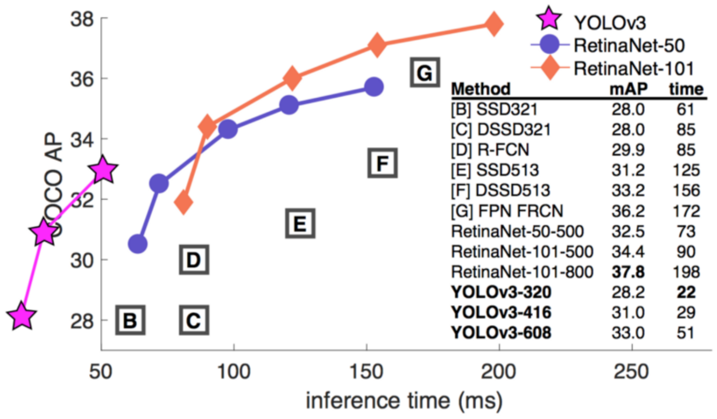


	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
	Convolutional	32	1 × 1	
1×	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
	Convolutional	64	1 × 1	
2×	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
	Convolutional	128	1 × 1	
8×	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
	Convolutional	256	1 × 1	
8×	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
	Convolutional	512	1 × 1	
4×	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Figure 2.29: Darknet 53

YOLOv3 performance

YOLOv3's COCO AP metric is on par with SSD but 3x faster. But YOLOv3's AP is still behind RetinaNet. In particular, AP at IoU=.75 drops significantly comparing with RetinaNet which suggests YOLOv3 has higher localization error. YOLOv3 also shows significant improvement in detecting small objects.
YOLOv3 performs very well in the fast detector category when speed is important.



Method	mAP	time
[B] SSD321	28.0	61
[C] DSSD321	28.0	85
[D] R-FCN	29.9	85
[E] SSD513	31.2	125
[F] DSSD513	33.2	156
[G] FPN FRCN	36.2	172
RetinaNet-50-500	32.5	73
RetinaNet-101-500	34.4	90
RetinaNet-101-800	**37.8**	198
YOLOv3-320	28.2	**22**
YOLOv3-416	31.0	29
YOLOv3-608	33.0	51

Figure 2.30: YOLOv3 VS. Retina netork

Chapter 3

Training, Validation and Testing

3.1 Datasets and Data Preparation

The purpose of this chapter is to illustrate the nature of the two datasets we've used in our project, how we've obtained them, split them and how the data preparation went through.

We've used two open source datasets for this project, one for training and validation, and the other for testing. Since handling the two datasets went quite differently, we've decided to provide to major sections for each dataset with the consequent sections explaining the methodology.

3.1.1 Udacity's Open Source Vehicle Dataset

Udacity is a famous E-Learning platform. They have numerous courses that cover a variety of topics. They also have what they call a "Nano Degree", a collection of courses on a specific topic that covers a huge syllabus whose aim is to turn the student taking the degree into an expert on the topic. They provide practical projects and walk through, video lectures, assignments and external resources.

Recently Udacity started a Nano degree on autonomous vehicles, and with that announcement, they published a massive dataset on vehicles for the students to train on. However, they decided to open source the dataset as well for the public, which in turn came into our favour.

The dataset they've published covers urban areas with cars, traffic lights, pedestrians, cyclists, and motorists. We carefully surveyed the contents of the datasets and chose the most appropriate one for our purpose.

Udacity's dataset is publicly available at their Github repository. To download the dataset we visited https://github.com/udacity/self-driving-car/tree/master/annotations , Chose the second dataset because it had much more images, downloaded the dataset and the annotated CSV file.

To prepare the dataset, we extracted the archive we've downloaded and parsed the annotations from the CSV file format into the Darknet format. Darknet takes annotations differently than most other frameworks. The format that the CSV file had the number of objects, the object class, the x-min, y-min, x-max, y-max coordinates of the bounding box.

However, Darknet takes its annotations in the following format:
• $< object - class >< x >< y >< width >< height >$
• $< object - class > - integernumberofobjectfrom0to(classes - 1)$
• $< x >< y >< width >< height >$- float values relative to width and height of image, it can be equal from 0.0 to 1.0
• $< x >< y > - arecenterofrectangle(arenottop - leftcorner)$

for example:

$< x >=< absolute_x > / < image_width > or < height >=< absolute_height >< image_height >$

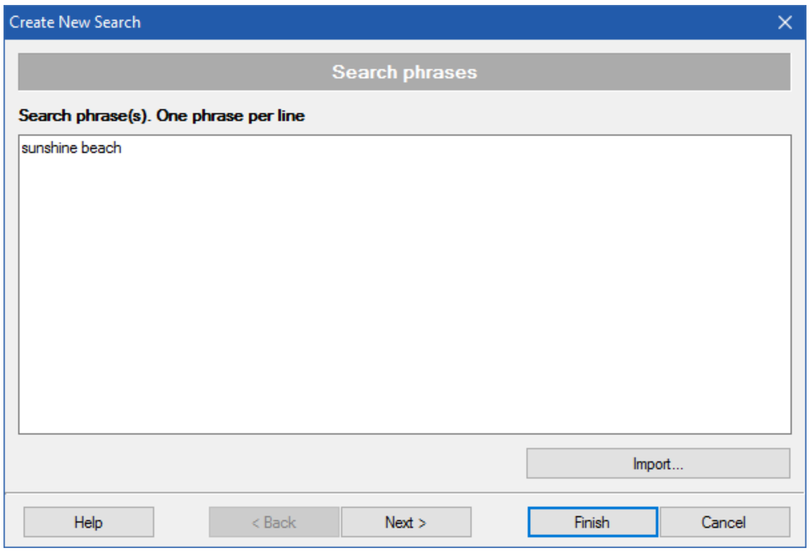
The reason for that is that Darknet, or YOLO to be precise, resizes the image much time and performs various translations, rotations, and stretching operations on the image in order to ensure position variability and scale invariance. Thus, it makes more sense to take the coordinates of the bounding box of the annotated image relative to the image width and height which alleviates various calculations to be needed later.

The dataset we've chosen comprised of 13,000 annotated images. The annotations were in a CSV format, however, Darknet has a specific way of reading annotations, so You have to change that using a Python script.

3.1.2 Image Net

Extreme Picture Finder is a software for Windows which offers the easiest way to download Google Images search results to your PC automatically. All you have to do is to enter your search keyword or phrase and hit the Start button. The program will go through all the search result and deliver the full-size images right to your PC very fast.

Here is the quick-start Guide. First of all - download the most recent version of Extreme Picture Finder using the button above. Install and start the program. In the program, menu selects the Search-New search item or click the button to open the New Search Wizard window.



Search phrase(s). One phrase per line

sunshine beach

Figure 3.1: create new search

Now type your search keyword or phrase and click Finish button. That's it! That's all you have to do to save the search results from Google, Bing, Ask.com, and Flickr.com.And you can view all downloaded images in the built-in image viewer immediately.Getting the Images you need to annotate it using annotating tool like " BBOX-TOOL" will be explained later.

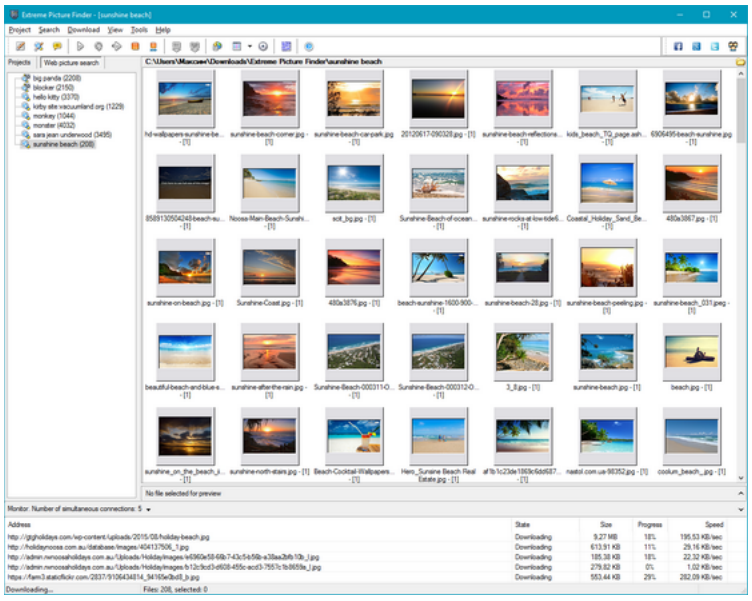

Extreme Picture Finder - [sunshine beach]
Project Search Download View Tools Help

Figure 3.2: Getting the Pictures

then after getting the annotations, we need to convert it to Dark-net Format using convert.Py script " On Our GitHub "

3.1.3 Cars Dataset

The Cars DataSet contains 16,185 images of 196 classes of cars. The data is split into 8,144 training images and 8,041 testing images, where each class has been split roughly in a 50-50 split. Classes are typically at the level of Make, Model, Year, e.g. 2012 Tesla Model S or 2012 BMW M3 Coupe.A devkit, including class labels for training images and bounding boxes for all images.

run annot.py to get annotations transformed from .mat to files .jpg
run tojpeg.py to replace the extension from .jpg to .txt
run annot2.py to make annotations in the form of bbox tool
run convert.py to convert to darknet format
“ supported on our GitHub “

3.1.4 Getting Your Own Data Set

if you want to have a big data set you need to record at least 2 Hours of cars And that what we had done, we fixed an iPhone 6S + on the front mirror facing the windshield to recode the road we set the camera on “ 720 HD at 60 frames per seconds “.
We had taken care of the windshield to be so transparent and we put a black piece of cloth to prevent the reflection of the Car dashboard on the windshield. And we had to drive on medium speed to prevent the oscillation on the mirror. Next, we needed to get images from the video to use it as a data set.

We installed FFMPEG via a terminal using “sudo apt-get install ffmpeg” and that as about all we needed to do to obtain the tool.

for example:

$$ffmpeg - iIMG_7259.MOV - r4/1 < filename.JPEG >$$

4/1 this part where we get the numbers of frames per seconds

Preparation of annotations
once we had unannotated images we need to get the annotations for the training so we used BBox Tool

A simple tool for labeling object bounding boxes in images, implemented with Python Tkinter.

Usage

• The current tool requires that the images that will be labeled reside in /Images/001, /Images/002, etc. You will need to modify the code if you want to label images elsewhere.
• Input a folder number (e.g, 1, 2, 5...), and click Load. The images in the folder, along with a few example results will be loaded.
• To create a new bounding box, left-click to select the first vertex. Moving the mouse to draw a rectangle, and left-click again to select the second vertex.
• To cancel the bounding box while drawing, just press Esc.
• To delete an existing bounding box, select it from the list box, and click Delete. To delete all existing bounding boxes in the image, simply click ClearAll.

• After finishing one image, click Next to advance. Likewise, click Prev to reverse. Or, input an image id and click Go to navigate to the specified image.Be sure to click Next after finishing an image, or the result won't be saved. After Annotating

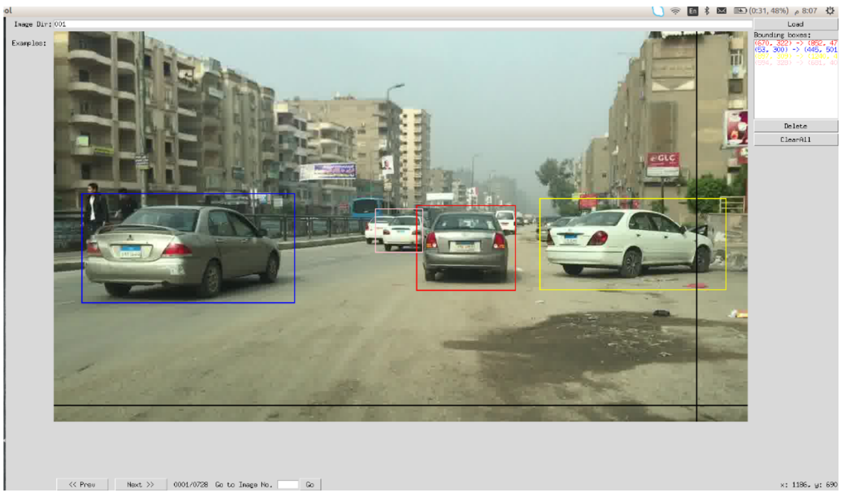


Figure 3.3: annotaion using bbox tool

you will have images that have no cars with 0 annotations, these images affect the training process so we need to delete them to do that you have to do 2 steps • Opening the annotations folder and sort the files by size then get all files that have 2 bytes size cut them and paste them in the images folder • run deletingfiles.py script this will delete the images and its annotations.

Then we used Convert.py script " on our GitHub" to convert these annotations to Darknet Format, then we used to jpg.py script " on our GitHub" to convert the images to.JPG.

After we had our annotated Dataset we need to Put the images.JPG and the annotations.txt in the same folder then we used Process.Py script to have Our " train.txt" and "Test.txt" these two files have the directory of all the images and it annotations divided into 90% training data and 10% of testing data

After we had our annotated Dataset we need to Put the images.JPG and the annotations.txt in the same folder then we used Process.Py script to have Our " train.txt" and "Test.txt" these two files have the directory of all the images and it annotations divided into 90% training data and 10% of testing data

3.2 Training, Validating, Optimizing and Testing YOLO

This chapter highlights the most important part of our project. It aims to show the training and validation of our CNN of choice, YOLO. It also shows the optimizations and modifications we've done both to the network and the Jetson TX2 to accommodate the processing power required to obtain our results. We conclude the chapter by showing our results.

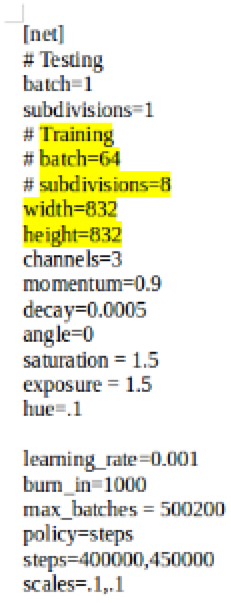
3.2.1 Training YOLO

Training YOLO to detect the custom object, in our case cars, required us to install the previously mentioned frameworks and tools, and obtain the datasets that match our requirements. All of our training was done offline on a desktop PC running Ubuntu 16.04LTS, a Nvidia GTX1080Ti with 8GBs of DDR3 memory for the video card, 10GBs for the RAM and an Intel i7-4670K 8 core CPU.
Since we planned to run the system on the Jetson TX2, we chose the Tiny-YOLO architecture for our system as it requires much less memory to load and much less processing power. The Tiny architecture can be loaded easily into the 32GB memory of the Jetson, hence it is the optimal choice.

To train YOLO on custom objects, we needed to create several files. YOLO essentially needs the following list of files to be created in order to train:

• Configuration file: this file contains the network architecture. Once you've decided to work on 3 network architectures, So we should create a .cfg file containing the parameters for our network. In our case, we created and modified an exact match of the

yolov2-tiny.cfg
yolov3-tiny.cfg and
yolov2.cfg file.



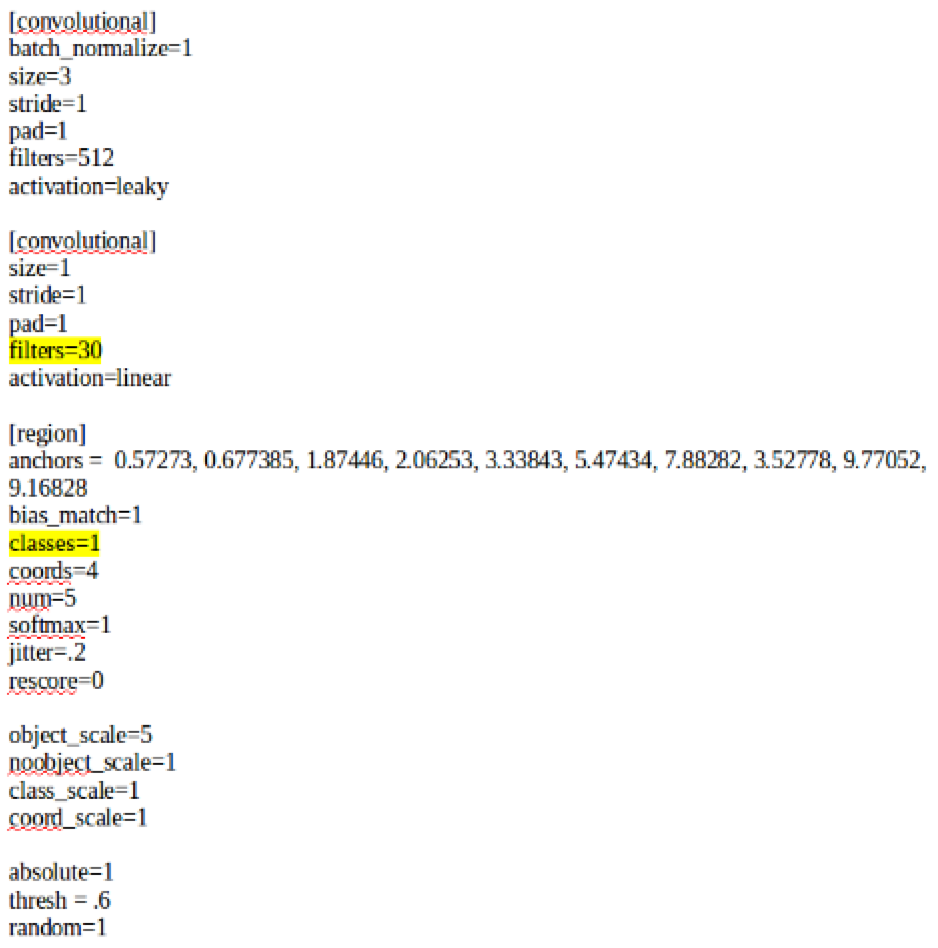
```
[net]
# Testing
batch=1
subdivisions=1
# Training
# batch=64
# subdivisions=8
width=832
height=832
channels=3
momentum=0.9
decay=0.0005
angle=0
saturation = 1.5
exposure = 1.5
hue=.1

learning_rate=0.001
burn_in=1000
max_batches = 500200
policy=steps
steps=400000,450000
scales=.1,.1
```

Figure 3.4: yolov2.cfg

We've edited the batch size (the number of images per iteration) into 64. We've also edited the number of subdivision (the number of images that get loaded into memory at once) into 8. We also increased the width and height of the input image to 832x832 instead of 416x416. Increasing the resolution of the image gives us better results during testing, and the same weights obtained can be used for any resolution later on.

3.2.2 yolov2-tiny.cfg and yolov2.cfg

We changed the number of classes to 1 since we're training on only one custom object which is "car". Finally, we edited the number of filters in the last layer according to the equation filters=(classes + 5)*5, in which case gave us 30 filters.

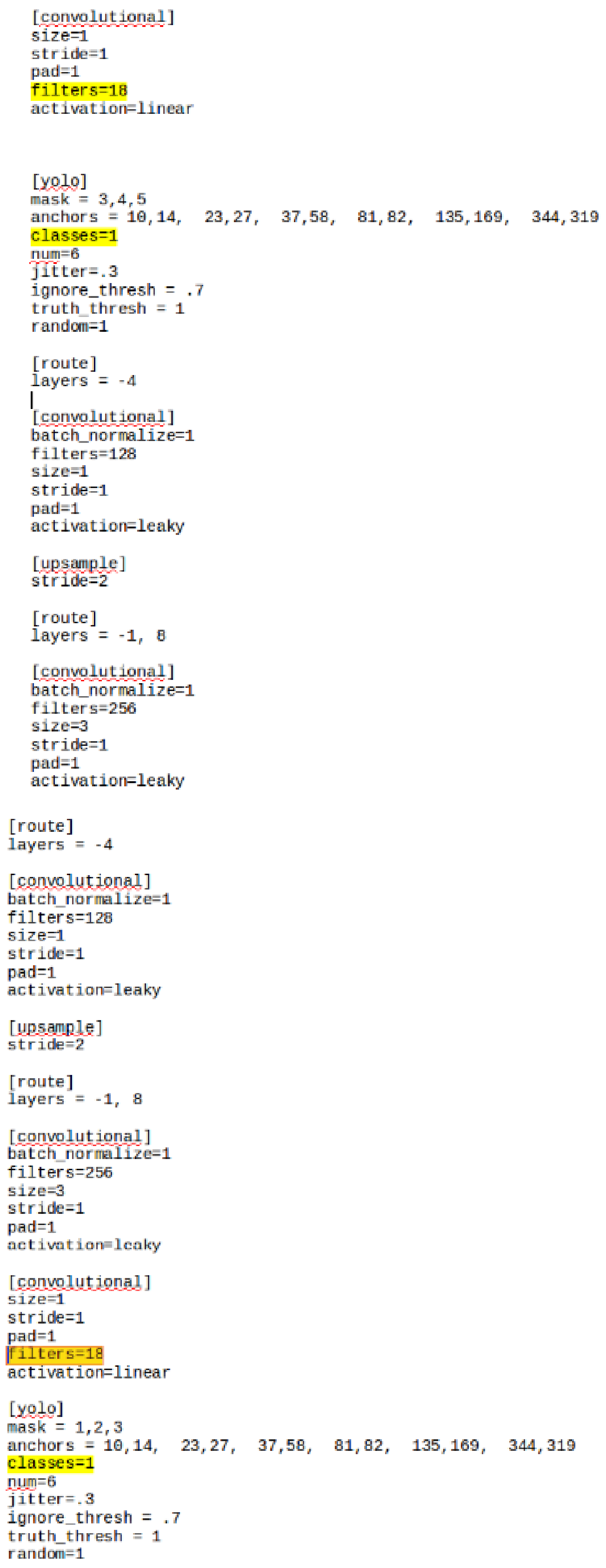


```
[convolutional]
batch_normalize=1
size=3
stride=1
pad=1
filters=512
activation=leaky

[convolutional]
size=1
stride=1
pad=1
filters=30
activation=linear

[region]
anchors =  0.57273, 0.677385, 1.87446, 2.06253, 3.33843, 5.47434, 7.88282, 3.52778, 9.77052,
9.16828
bias_match=1
classes=1
coords=4
num=5
softmax=1
jitter=.2
rescore=0

object_scale=5
noobject_scale=1
class_scale=1
coord_scale=1

absolute=1
thresh = .6
random=1
```

Figure 3.5: yolov2-tiny.cfg

3.2.3 yolov3-tiny.cfg

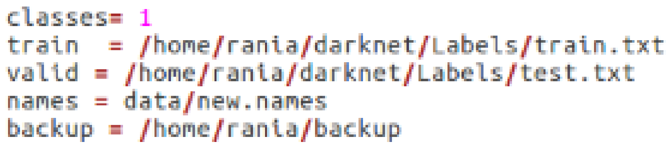
We changed the number of classes to 1 since we're training on only one custom object which is "car". Finally, we edited the number of filters in the last layer according to the equation filters=(classes + 5)*3, in which case gave us 18 filters.



```
  [convolutional]
  size=1
  stride=1
  pad=1
  filters=18
  activation=linear


  [yolo]
  mask = 3,4,5
  anchors = 10,14,  23,27,  37,58,  81,82,  135,169,  344,319
  classes=1
  num=6
  jitter=.3
  ignore_thresh = .7
  truth_thresh = 1
  random=1

  [route]
  layers = -4

  [convolutional]
  batch_normalize=1
  filters=128
  size=1
  stride=1
  pad=1
  activation=leaky

  [upsample]
  stride=2

  [route]
  layers = -1, 8

  [convolutional]
  batch_normalize=1
  filters=256
  size=3
  stride=1
  pad=1
  activation=leaky

[route]
layers = -4

[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=leaky

[upsample]
stride=2

[route]
layers = -1, 8

[convolutional]
batch_normalize=1
filters=256
size=3
stride=1
pad=1
activation=leaky

[convolutional]
size=1
stride=1
pad=1
filters=18
activation=linear

[yolo]
mask = 1,2,3
anchors = 10,14,  23,27,  37,58,  81,82,  135,169,  344,319
classes=1
num=6
jitter=.3
ignore_thresh = .7
truth_thresh = 1
random=1
```

Figure 3.6: yolov3-tiny.cfg

• Object names: This .names file contains the names of the classes we're training on. The order of classes must match the number used in the annotation file per image to match the classes together, our objects were only 1, car.
• A data file: this .data file contains 4 parameters
(train : points to the location where the file containing a list of images to be trained on resides, valid: points to the location where the file containing list of images to be validated on resides, names: points to the location of the names file, backup: points to the location where the weights are stored after training).



```
classes= 1
train  = /home/rania/darknet/Labels/train.txt
valid = /home/rania/darknet/Labels/test.txt
names = data/new.names
backup = /home/rania/backup
```

Figure 3.7: coco.data

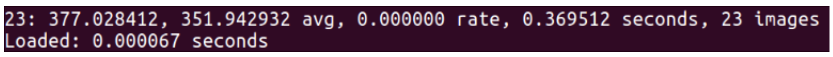
The final step was to place the images and their annotations in the same folder and download the pre-trained weights file to begin the actual training.
To train the network, we used the following command:

```
./darknet detector train data/Car.data cfg/car-tinyV2.cfg darknet53.conv.74
./darknet detector train data/Car.data cfg/car-tinyV3.cfg darknet53.conv.74
./darknet detector train data/Car.data cfg/car-V2.cfg darknet53.conv.74
```

./darknet : the executable file resulting from building Darknet.
detector: indicates which object file to use after building Darknet.
train: indicates the operating we'd like to invoke
data/ Car.data: indicates the location of the data file to use.
cfg/car-tinyV2.cfg , car-tinyV3.cfg and car-V2.cfg: indicates the location of the configuration file to use.
darknet53.conv.74: indicates the weights file with which the weights in the network are initialized.

Darknet saves the weights initially after every 100 iterations in the backup folder. Once it reaches the 1000th iteration, it saves them after every 1000 iterations. The good thing is that we could pause the training any time and resume it later with the latest saved weights file. We can also validate each weights file separately.

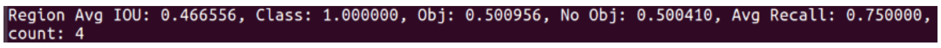
OutPut Batch



```
23: 377.028412, 351.942932 avg, 0.000000 rate, 0.369512 seconds, 23 images
Loaded: 0.000067 seconds
```

Figure 3.8: batch output 1

• 23 indicates the current training iteration/batch.
• 377.028412 is the total loss.
• 351.942932 avg is the average loss error, which should be as low as possible. As a rule of thumb, once this reaches below 0.060730 avg, you can stop training.
• 0.00000 rate represents the current learning rate, as defined in the .cfg file.
• 0.369512 seconds represents the total time spent to process this batch.
• The 23 images at the end of the line is nothing more than 9000 * 64, the total amount of images used during training so far.



```
Region Avg IOU: 0.466556, Class: 1.000000, Obj: 0.500956, No Obj: 0.500410, Avg Recall: 0.750000,
count: 4
```

Figure 3.9: batch output 2

• Region Avg IOU: 0.466556 is the average of the IOU of every image in the current subdivision. A 46,65% overlap, in this case, this model still requires further training.

• The Avg Recall: 0.7500 is defined in the code as recall/count, and thus a metric for how many positives YOLOv2 detected out of the total amount of positives in this subdivision. In this case, only one of the eight positives was correctly detected.

• Count: 4 is the number of positives (objects to be detected) present in the current subdivision of images (subdivision with size 8 in our case). Looking at the other lines in the log, you'll see there is also subdivision that only has 6 or 7 positives, indicating there are images in that subdivision that do not contain an object to be detected.

3.2.4 Validation

According to the users of YOLO, usually 2000 iterations suffice per class, however, we went for 6,000 iterations For YOLO-tinyV2, 4,000 iterations For YOLO-tinyV3 and 4,000 iterations For YOLO-V2, and picked the one where we obtained the best measures on the validation set. Our choice was at the 2000th iteration For YOLO-tinyV2 because the network seemed to suffer from overfitting after this iteration, where it performed well on the training set but on the validation set it gave worse results than it should.

The measure for validation in YOLO is the Intersection Over Union or IOU. The reason for that is that IOU takes into account both the size of the bounding box from the network and the size of the actual bounding box from the ground truth as illustrated in the next figure.

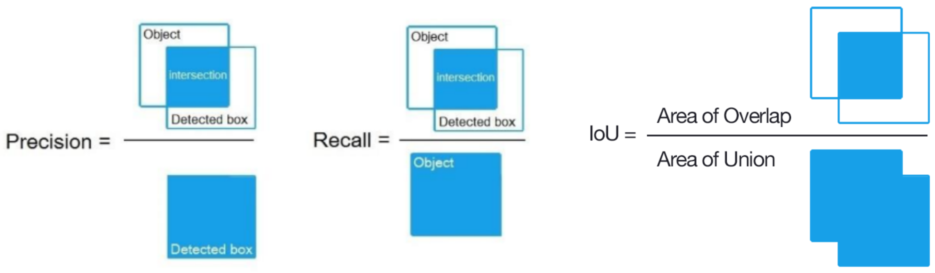


Figure 3.10: Comparsion between precision, recall, and IoU methods

To validate YOLO on our validation part of the dataset, we invoke the same command as the one needed for training but with a minor difference:

we used the following script to run all the weights in series to get the iou
#binbash
for ((i=0;i¡=5;i++))
do

$ .darknet detector recall cfgcoco1.data cfgcars_v2.cfg backup cars_v2_2000.weights

done

Note: Go for /darknet/examples
Open Detector.c
paste this :
list*plist = get_paths(”test.txt”)

In line 496 instead of the existing line
The following graph shows the results of the best-performing weights for the tiny model obtained at all iterations. They show the Intersection over union and the recall rates. To plot this graph we wrote our own python scripts to handle the output from the terminal and plot.

3.3 Testing

To test our trained Tiny-YoloV2 model we record a new video “ 720 HD at 30 fps “ to be used as a test data. After installation the kit is ready for the test we downloaded YOLO on it then we put our suitable weights, cfg, data and names files.

We run this command
$./darknet$ detector demo $cfg/coco1.data$ $cfg/cars_v2.cfg$ 11.weights $< filename >$
“ the video will be on out git hub “ to run the real-time detection on the kit’s camera you need **Install OpenCV for Jetson TX2**

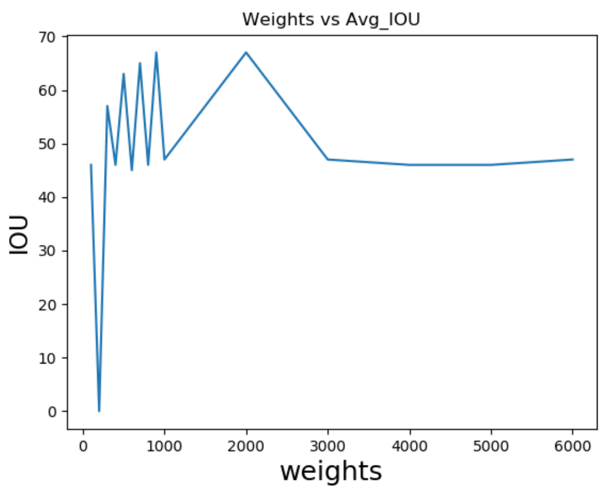


Figure 3.11: Weights VS IOU

3.4 Jetson TX2

The Jetson TX2 Developer Kit gives you a fast, easy way to develop hardware and software for the Jetson TX2 AI supercomputer on a module. It exposes the hardware capabilities and interfaces of the developer board come with design guides and other documentation and are pre-flashed with a Linux development environment. It also supports NVIDIA Jetpack—a complete SDK that includes the BSP, libraries for deep learning, computer vision, GPU computing, multimedia processing, and much more.

You will find all the spices in the following:
https://developer.nvidia.com/embedded/buy/jetson-tx2-devkit

3.5 JetPack

NVIDIA JetPack SDK is the most comprehensive solution for building AI applications. Use the JetPack installer to flash your Jetson Developer Kit with the latest OS image, to install developer tools for both the host PC and Developer Kit, and to install the libraries and APIs, samples, and documentation needed to jumpstart your development environment.
You will find all the documentation about the installation in the following Nvidia docs JetPack

3.5.1 OpenCV Installation

To download the source and build OpenCV:

```
$ git clone https://github.com/jetsonhacks/buildOpenCVTX2.git
$ cd buildOpenCVTX2
$ ./buildOpenCV.sh
```

Once finished building, you are ready to install.As explained in OpenCv navigate to the build directory to install the newly built libraries:

```
$ cd  /opencv/build
$ sudo make install
```

Once you have generated the build files, you can use the ccmake tool to examine the different options and modules available.Remember to setup you OpenCV library paths correctly.

Test
To test the installation. Open the terminal and type

```
$ python
>> import cv2
>>
```

Notes
• This is meant to be a template for building your own custom version of OpenCV, pick and choose your own modules and options

• Most people do NOT have both OpenCV4Tegra and the source built OpenCV on their system. Some people have noted success using both, however, check the forums.

• Sometimes the make tool does not build everything. Experience dictates to go back to the build directory and run make again, just to be sure

• Different modules and setting may require different dependencies, make sure to look for error messages when building.

• After building, you should run the tests. The build script includes the testing options. All tests may not pass.

• The build script adds support for Python 2.7

• The compiler assumes that the Jetson TX2 aarch64 (ARMv8) architecture is NEON enabled, therefore you do not have to enable the NEON flag for the build

3.6 Open the onboard camera in Jetson TX2

It was a challenge to open the onboard camera as the TX2 camera's configuration is not the same as the laptop built-in camera. The first task is to open the onboard camera. Camera access is through a GStreamer pipeline:

```
cap = cv2.VideoCapture("nvcamerasrc ! video/x-raw(memory:NVMM), width=(int)1280, height=(int)720,format=(string)I420, framerate=(fraction)30/1 ! nvvidconv flip-method=0 ! video/x-raw, format=(string)BGRx ! videoconvert ! video/x-raw, format=(string)BGR ! appsink")
```

This will enable you to open the onboard camera and use it. You may test it running the Canny Edge Detetor

3.7 YOLO Demo with the onboard camera

To use the onboard camera for a demo using YOLO, Type the same command for the demo and add the last part that opens the camera.

```
$ ./darknet detector demo cfg/coco.data cfg/yolov3.cfg yolov3.weights cv2.VideoCapture(”nvcamera
! video/x-raw(memory:NVMM), width=(int)1280, height=(int)720,format=(string)I420,
framerate=(fraction)30/1 ! nvvidconv flip-method=0 ! video/x-raw, format=(string)BGRx
! videoconvert ! video/x-raw, format=(string)BGR ! appsink”)
```

Chapter 4

Depth estimation

4.1 Introduction

the main purpose is to estimate the distance between our car and surrounding cars to get enough data to let the car drive itself and avoid any accident may be happened ,to get the distance of an object there many approaches we start a search for the suitable solution for our case Different approaches.

4.1.1 LIDAR

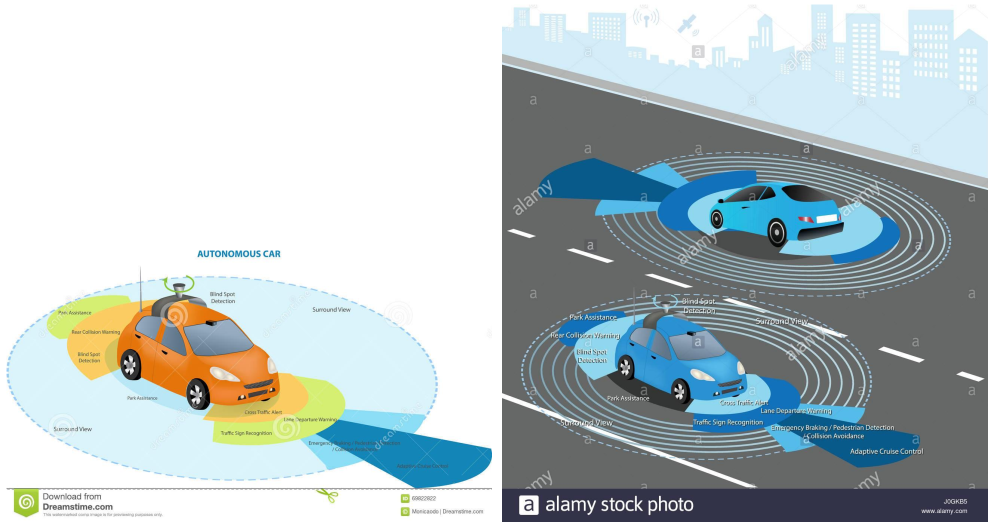


Figure 4.1: Lidar

LIDAR is considered the best solution to get an accurate distance for surrounding objects but wont be suitable for our case as it very expensive and need a government permission so we move to camera direction
how to get the distance using only camera as we won't use any extra hardware above the main purpose of the project and to make the whole system independent module work on its own , that will help in future work and commercial purposes.

4.1.2 Stereo vision systems

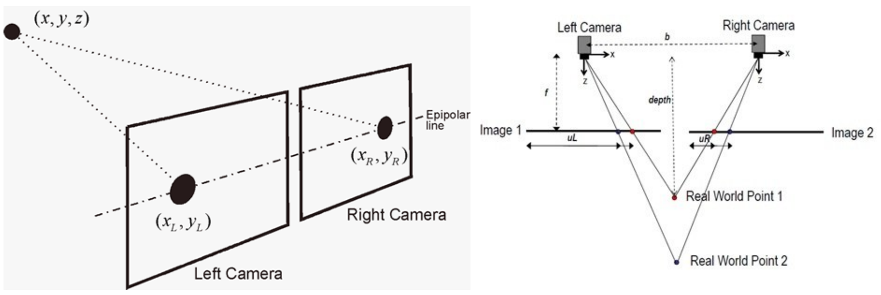


Figure 4.2: Stereo Approach

Several stereo algorithms have been proposed in recent years to solve the problem of finding the correspondence of the right and left image. Simple methods employ the measure of absolute or squared differences of the pixels intensities, to measure the similarity between the images (Sunyoto et al., 2004). Other methods, in order to increase accuracy, employ window-based matching, where a cost function is evaluated around the pixel of interest to find the best match. These methods usually do not consider occlusions and present problems in regions displaying little or repetitive textures, leading to similar cost functions and being unable to find the proper match (Darabiha et al., 2003; Silva et al., 2003; Cox et al., 1996).

4.1.3 Hardware

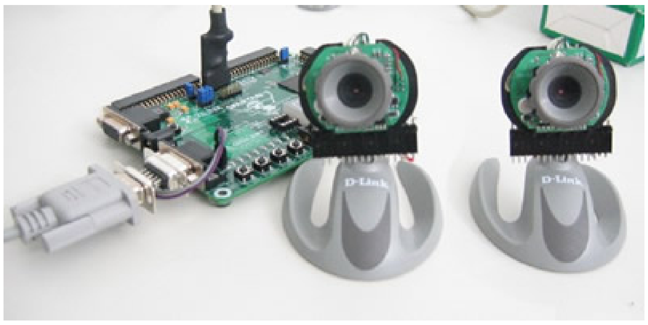


Figure 4.3: Stereo Hardware

But it needs a high and complex processing power, Thus using this method won't apply for us as frame per second (FPS) is one of the critical parameters we want to maintain in this scope we can use ZED camera

Figure 4.4: ZED Camera

which consists of 2 cameras and internal processing unit it can give the distance of each pixel but it also too expensive, so our approach is coming to find a way to get the distance using only a mono camera without any extra hardware

4.1.4 Deep learning and Heat map

Using deep learning to get the depth of a pixel by training enough data the output be like the figure below a heat map corresponding to the distance.

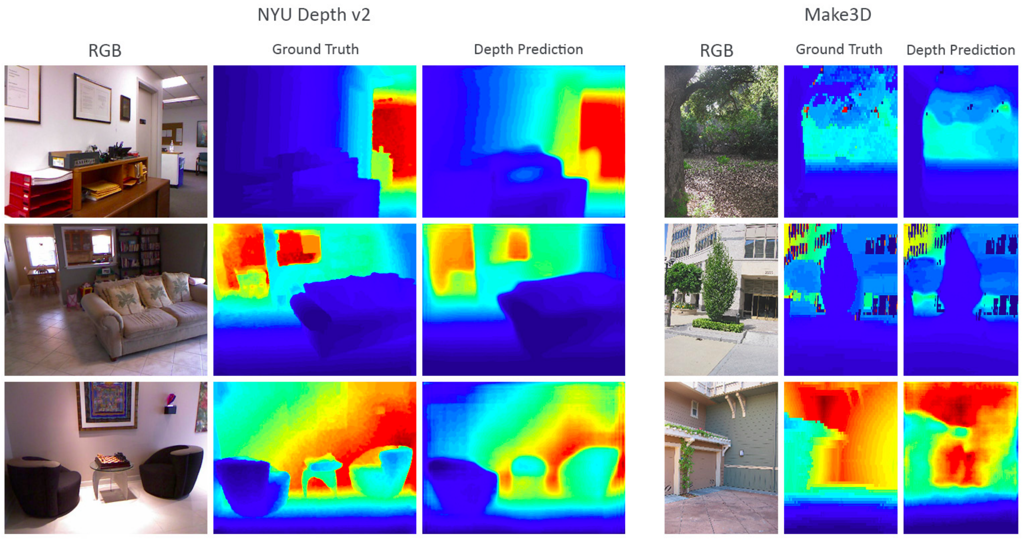


Figure 4.5: Heat Map

but the map changed with every picture and affected by a lot of another parameter so it won't be the best solution for our case as the milliseconds mean life in such a project.

4.1.5 Our approach

we come up with an idea, using the boundary box that our neural network draw over the object, and calculate the actual distance so we can find a relation between them.

Figure 4.6: Test Picture

First, we took over 100 pictures of cars with constant focal length and resolution and calculate the distance between the camera and the cars then we pass them throw our neural network and get the width of each car then we get the different relationships between them using Regression technique as we don't know the output form of the equation which should be used in our case we divided our data into 70 % train and 30 % test, so each time we apply different method we calculate the accuracy to get the best an equation that describes our case.
By applying different regression models we needed to know the goodness of fit for each model. That's why we're using R-squared.

4.1.6 R-squared

R-squared is a statistical measure of how close the data are to the fitted regression line. It is also known as the coefficient of determination, or the coefficient of multiple determination for multiple regression.

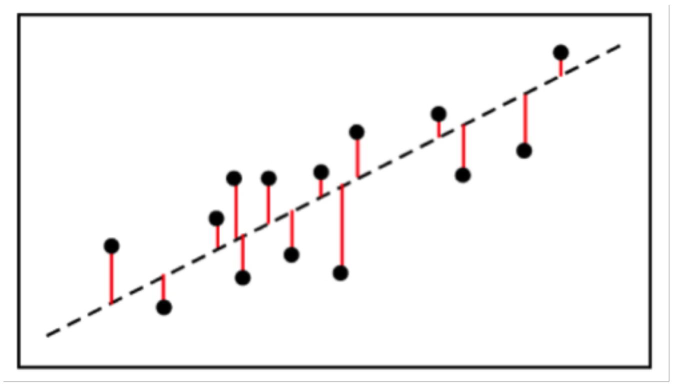


Figure 4.7: R-squared Calcuation

$$R - squared = 1 - \frac{first\ sum\ of\ errors}{second some of errors} \tag{4.1}$$

First, we use the line of best fit equation to predict y values on the chart based on the corresponding x values. Once the line of best fit is in place, we can create an error squared equation to keep the errors within a relevant range. Once we have a list of errors, you can add them up and run them through the R-squared formula.

4.2 Results of Width Approach

First, we use the line of best fit equation to predict y values on the chart based on the corresponding x values. Once the line of best fit is in place, we can create an error squared equation to keep the errors within a relevant range. Once we have a list of errors, you can add them up and run them through the R-squared formula. A python script was created to get the regression for different regression model and to calculate the R-Squared and the average error.

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2.$$

4.3 Different Models

4.3.1 Simple Linear Regression

In statistics, linear regression is a linear approach to modelling the relationship between a scalar response (or dependent variable) and one or more explanatory variables (or independent variables). The case of one explanatory variable is called simple linear regression

The very simplest case of a single scalar predictor variable x and a single scalar response variable y is known as simple linear regression. The Equation of linear Model is

$$ax + b$$

The fitting returns the parameters of a and b. Thus the equation turns to be

$$-4.36403157 \cdot 10^{-1}x + 1.46100620 \cdot 10^{3}$$

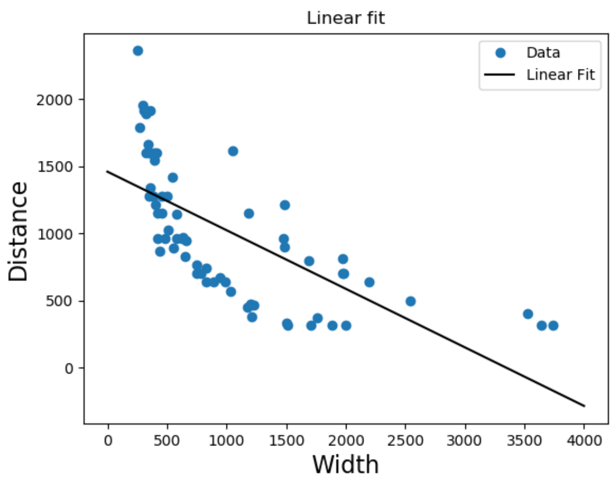


Figure 4.8: Simple Linear Regression

As shown above, the linear fitting is not a good choice to express our data curve. To prove it R-Squared is calculated to see the goodness of the fitting.

Simple linear regression R-Squared =46.57%

4.3.2 Quadratic Regression

A quadratic regression is the process of finding the equation of the parabola that best fits a set of data. As a result, we get an equation of the form:

$$y = ax^2 + bx + c$$

The best way to find this equation manually is by using the least squares method. That is, we need to find the values of a,b,and c such that the squared vertical distance between each point (xi,yi) and the quadratic curve is minimal. The fitting returns the parameters of a, b and c. Thus the equation turns to be

$$y = 2.25629607 \cdot 10^{-4}x^2 - 1.19047614x + 1.84189804 \cdot 10^3 \tag{4.2}$$

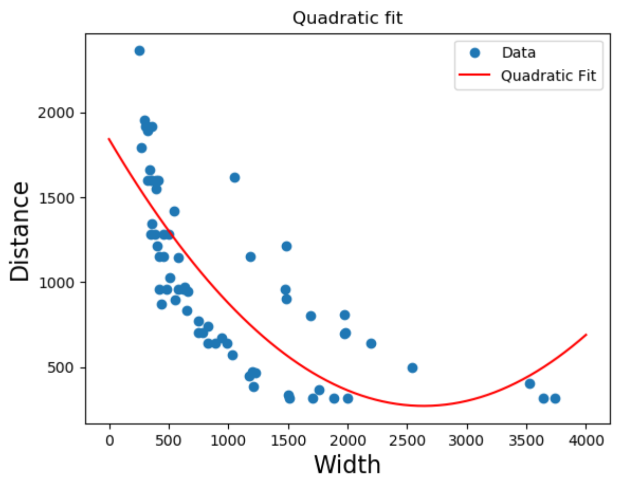


Figure 4.9: Quadratic Regression

However, the quadratic fitting is considered to be better than the simple linear regression, it still not a good choice to perfectly express our data curve. To prove it R-Squared is calculated to see the goodness of the fitting.

Quadratic regression R-Squared =61.9%

4.3.3 Cubic Regression

Cubic regression is an approach to modelling the relationship between a scalar response (or dependent variable) and one explanatory variable (or independent variable). As a result, we get an equation of the form:

$$y = ax^3 + bx^2 + cx + d$$

The best way to find this equation manually is by using the least squares method. That is, we need to find the values of a,b, c and d such that the squared vertical distance between each point (xi,yi) and the cubic curve is minimal. The fitting returns the parameters of a, b, c and d. Thus the equation turns to be

$$y = -2.65465233 \cdot 10^{-7} x^3 + 1.70655105 \cdot 10^{-3} x^2 - 3.29419599 x + 2.51580606 \cdot 10^3 \quad (4.3)$$

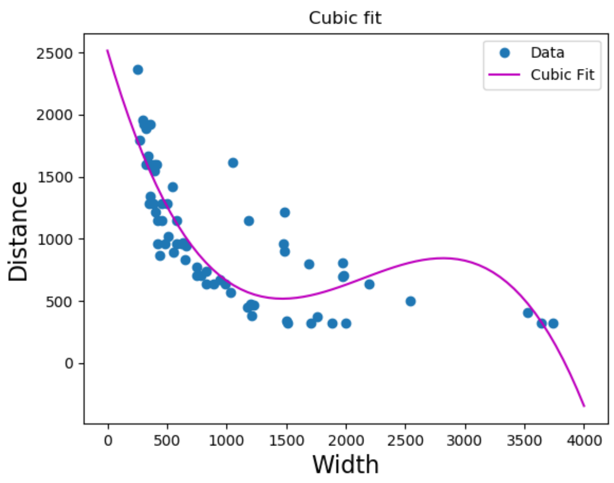


Figure 4.10: Cubic Regression

However, the cubic fitting is considered to be better than both the simple linear regression and the quadritic regression, it has oly one problem which is that it goes to the negative distance once the input width exceeds 3839.50 and that happens alot in cars.Although the calculated R-Squared = 72.66%

4.3.4 Logarthmic Fit

From the scatter plot of the data, logarthmic equation would be simillar to the required curve. Thus, logarthmic fitting was used to get the parameters of its equation and to calculate its fitting goodness to be compared with pervious models.It still describes the relationship between a scalar response (or dependent variable) and one explanatory variable (or independent variable). As a result, we get an equation of the form:

$$y = a + b \log 10\,(x)$$

The fitting returns the parameters of a and b. Thus the equation turns to be

$$y = 4933.66141883 - 1356.58125959 \log 10\,(x) \tag{4.4}$$

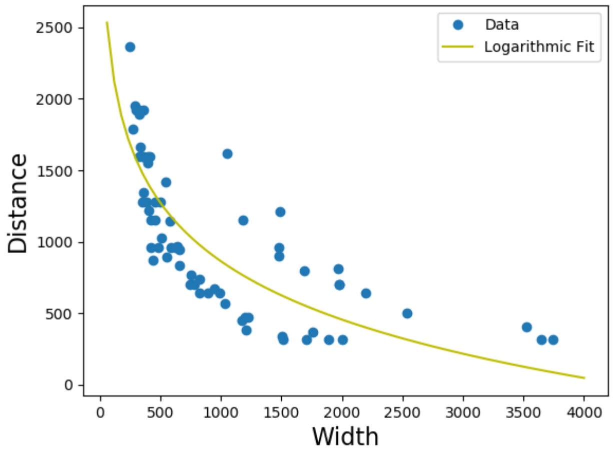


Figure 4.11: Logarthmic Regression

For this non-linear equation the fitting is too much better than before. But still has the same problem of the cubic regression which is that at a certain value it intersects with the x axis then goes to the negative part. Logarthmic equation is going to the negative distance (at width = 433.346) more quickely than the cubic fit does. Which mean it will be unvalid for any car has a width exceeding the threshold value as it will be seen at a negative distance which is not real. The calculated R-Squared = 67.45%

4.3.5 Power Fit

Since cubic regression is the closest approach to model our data but has the negative distance problem. Thus, we thought about another function that never goes to the negative axis and can describ the relationship between a scalar response (or dependent variable) and one explanatory variable (or independent variable). As a result, we get the power fit which has an equation of the form:

$$y = c \cdot x^m$$

The fitting returns the parameters of a and b. Thus the equation turns to be

$$y = 1.09669465 \cdot 10^5 \cdot x^{-7.24335644 \cdot 10^{-1}} \tag{4.5}$$

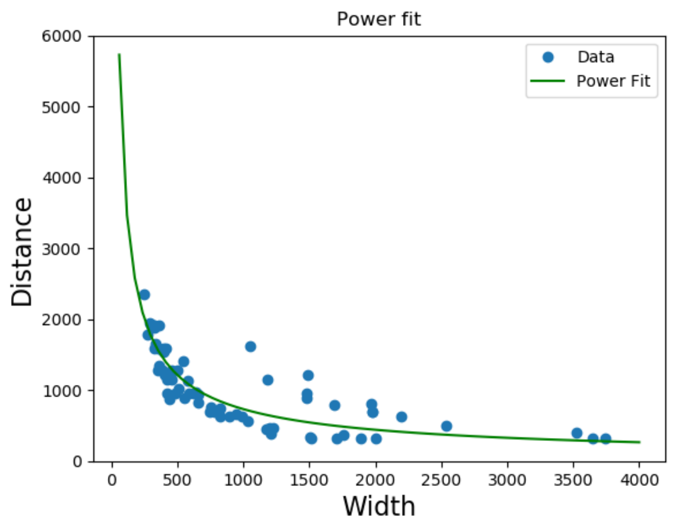


Figure 4.12: Power Regression

Power fit produces the required curve for fitting our data and it explains very well the relation between the distance and the width of the car. Moreover, it has the highest R-Squared = 72.66%

Which means it can fit most of our data (72.66% of it) and can predict the distance while the width is given from the detected car.

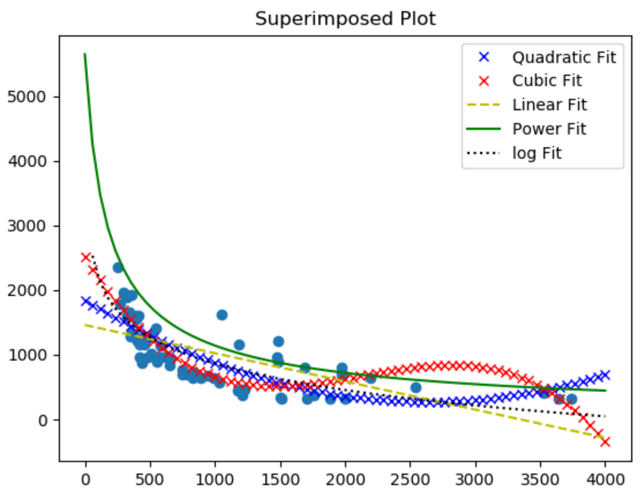


Figure 4.13: All Model Output

Fitness of linear fit= 46.57%
Fitness of quadratic fit= 61.988%
Fitness of Cubic fit= 72.66%
Fitness of Lograthmitc fit= 67.45%
Fitness of power fit= 76.109%

4.3.6 Modifing YOLO to estimate the distance

The obtained equation is added to our neural network (YOLO) to be processed in real time detection to know how far cars are going whenever they are moving. the distance will be estimated for each detected car and the user could see the value in meter above the boundry box that bounding the detected car.

To do that, the source code of YOLO was modified by opening

$$/darknet/src/image.c \tag{4.6}$$

then go to the (draw_detections) function which is used to draw the boundry box over the detected car and to type its class name after detecting it.

Add the obtained equation in line 260 and print its value, then type "Make" at the terminal and test it. That will print the distance only at the terminal but not printed at the showen detected object.

To print the distance to be showen beside the class name, we modify the same source file by adding these two lines:

```
char L[200]=", Distance = "
sprintf(L "D = %.2f" d)
strcat(labelstr,L)
```

The first line will creat array of characters with a fixed size to hold the word distance and the value of it as YOLO is implemented in C not C++ thus, there is no string only array of characters. The third line will concatenate the distance with the label string (which perviously had the class name) to be showen togather. Only one thing left, you need to covert the number calculated from the equation from the integer type to char to be concatenated with the label stringm this can be done by using the second line. Type "Make" and test it !

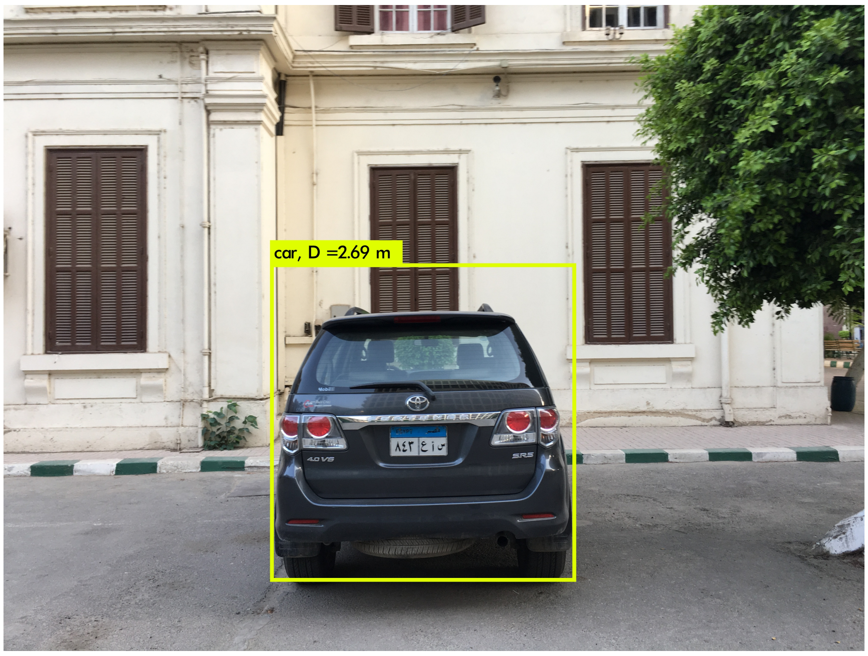


Figure 4.14: Width Output

4.4 Problems of width approach

This approach was perfect as long as the car is captured by its back but because our approach is based on the width of the car , then we have a fatal error occurs when the car is captured by its side not its back. This is seen by the relation as a very close car as it's having a very large width which is not the real thing. Here is an example below.

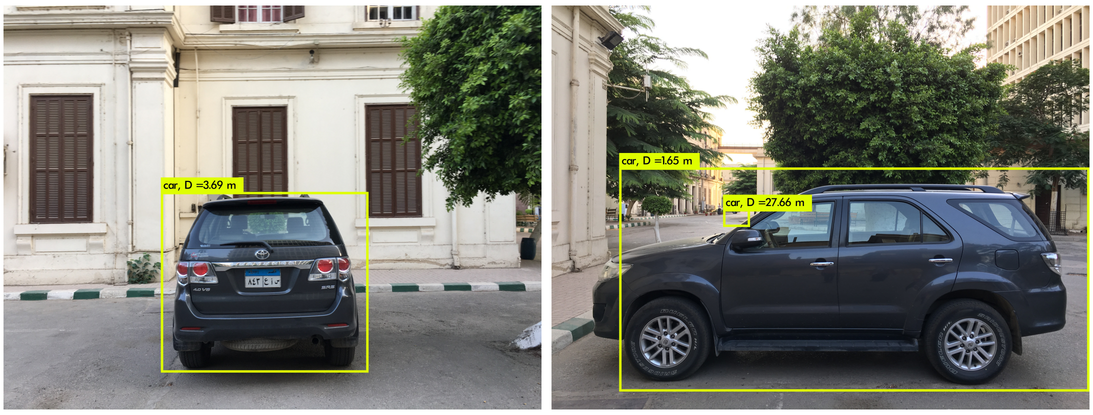


Figure 4.15: Width Problem

Two problems were found in this method:
• Large sized cars have large boundary boxes, so they appear closer than small sized cars with small boundary boxes.
• The width changing with car orientation as we it gets bigger as the car move to a

side and we need the parameter only depend on the car itself, not its orientation.
At the previous example the car's actual distance was 4.30 m when it was detected by its back it was seen at 3.69 m but when it was detected by its side it was seen at 1.65 m which is totally wrong. That's why we've turned to make our approach based on height not width.

4.5 Results of Height Approach

Similar to the width approach, we use the line of best fit equation to predict y values on the chart based on the corresponding x values. Once the line of best fit is in place, we can create an error squared equation to keep the errors within a relevant range. Once we have a list of errors, you can add them up and run them through the R-squared formula.
Another python script was created to get the regression for different regression model and to calculate the R-Squared and the average error.

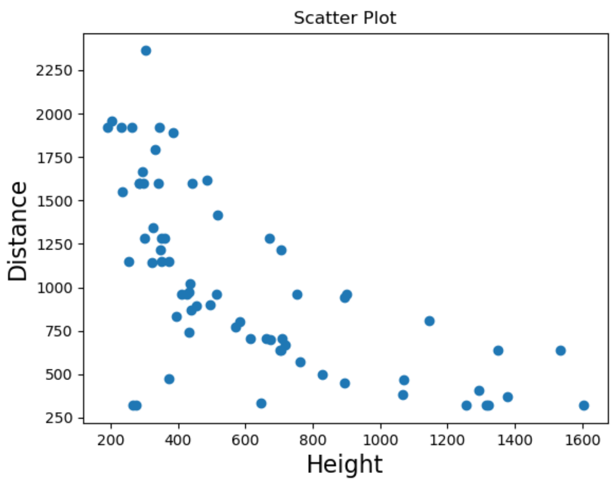


Figure 4.16: ScatterPlot

4.6 Different Models

4.6.1 Simple Linear Regression

In statistics, linear regression is a linear approach to modelling the relationship between a scalar response (or dependent variable) and one or more explanatory variables (or independent variables). The case of one explanatory variable is called simple linear regression

The very simplest case of a single scalar predictor variable x and a single scalar response variable y is known as simple linear regression. The Equation of linear Model is

$$ax + b$$

The fitting returns the parameters of a and b. Thus the equation turns to be

$$-9.52033163 \cdot 10^{-1}x + 1.60025939 \cdot 10^{3}$$

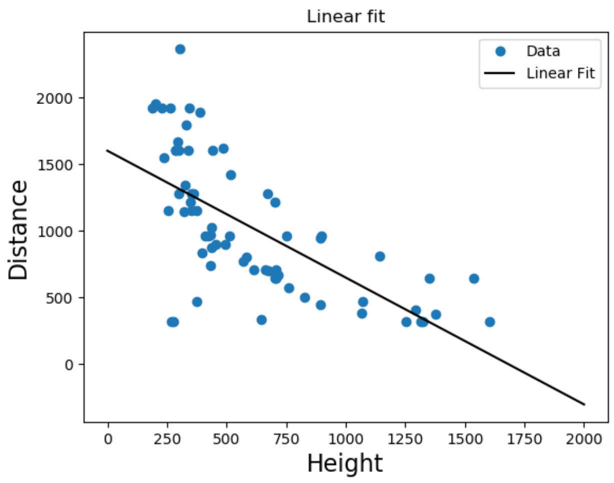


Figure 4.17: Simple Linear Regression

As shown above, the linear fitting is not a good choice to express our data curve. To prove it R-Squared is calculated to see the goodness of the fitting.

Simple linear regression R-Squared =44.10%

4.6.2 Quadratic Regression

A quadratic regression is the process of finding the equation of the parabola that best fits a set of data. As a result, we get an equation of the form:

$$y = ax^2 + bx + c$$

The best way to find this equation manually is by using the least squares method. That is, we need to find the values of a,b,and c such that the squared vertical distance between each point (xi,yi) and the quadratic curve is minimal. The fitting returns the parameters of a, b and c. Thus the equation turns to be

$$y = 1.01243832 \cdot 10^{-3}x^2 - 2.59374421x + 2.09112024 \cdot 10^3 \tag{4.7}$$

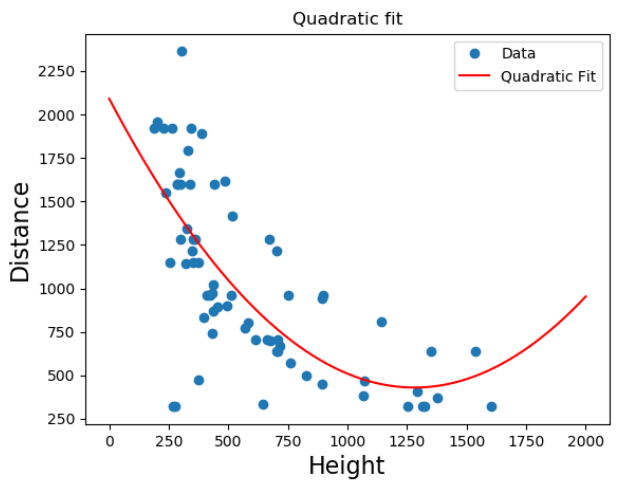


Figure 4.18: Quadratic Regression

However, the quadratic fitting is considered to be better than the simple linear regression, it still not a good choice to perfectly express our data curve. To prove it R-Squared is calculated to see the goodness of the fitting.

Quadratic regression R-Squared =50.65%

4.6.3 Cubic Regression

Cubic regression is an approach to modelling the relationship between a scalar response (or dependent variable) and one explanatory variable (or independent variable). As a result, we get an equation of the form:

$$y = ax^3 + bx^2 + cx + d$$

The best way to find this equation manually is by using the least squares method. That is, we need to find the values of a,b, c and d such that the squared vertical distance between each point (xi,yi) and the cubic curve is minimal. The fitting returns the parameters of a, b, c and d. Thus the equation turns to be

$$y = --8.78623075 \cdot 10^{-7}x^3 + 3.22119947 \cdot 10^{-3}x^2 - 4.17561758x + 2.40148100 \cdot 10^3 \tag{4.8}$$

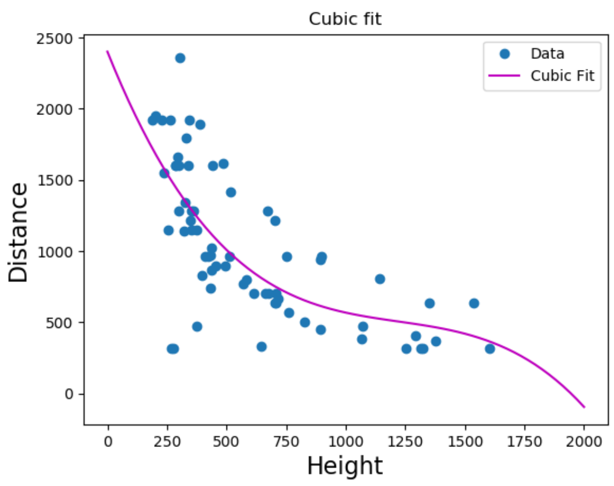


Figure 4.19: Cubic Regression

However, the cubic fitting is considered to be better than both the simple linear regression and the quadritic regression, it has oly one problem which is that it goes to the negative distance once the input width exceeds 1954.549 and that happens alot in cars.Although the calculated R-Squared = 51.3%

4.6.4 Logarthmic Fit

From the scatter plot of the data, logarthmic equation would be simillar to the required curve. Thus, logarthmic fitting was used to get the parameters of its equation and to calculate its fitting goodness to be compared with pervious models.It still describes the relationship between a scalar response (or dependent variable) and one explanatory variable (or independent variable). As a result, we get an equation of the form:

$$y = a + b \log 10\,(x)$$

The fitting returns the parameters of a and b. Thus the equation turns to be

$$y = -6.91721311 \cdot 10^{-1} + 7.20855715 \cdot 10^{4} \log 10\,(x) \tag{4.9}$$

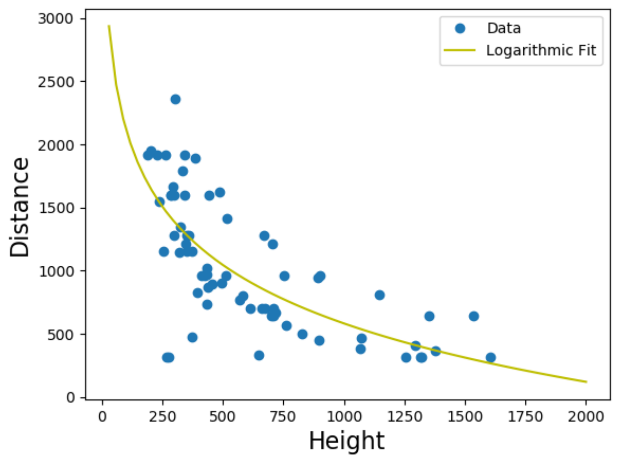


Figure 4.20: Logarthmic Regression

For this non-linear equation the fitting is too much better than before. But still has the same problem of the cubic regression which is that at a certain value it intersects with the x axis then goes to the negative part. Logarthmic equation is going to the negative distance (at Height = 239.705) more quickely than the cubic fit does. Which mean it will be unvalid for any car has a width exceeding the threshold value as it will be seen at a negative distance which is not real. The calculated R-Squared = 50.7749%

4.6.5 Power Fit

Since cubic regression is the closest approach to model our data but has the negative distance problem. Thus, we thought about another function that never goes to the negative axis and can describ the relationship between a scalar response (or dependent variable) and one explanatory variable (or independent variable). As a result, we get the power fit which has an equation of the form:

$$y = c \cdot x^m$$

The fitting returns the parameters of a and b. Thus the equation turns to be

$$y = 7.20855715 \cdot 10^4 \cdot x^{-6.91721311 \cdot 10^{-1}} \tag{4.10}$$

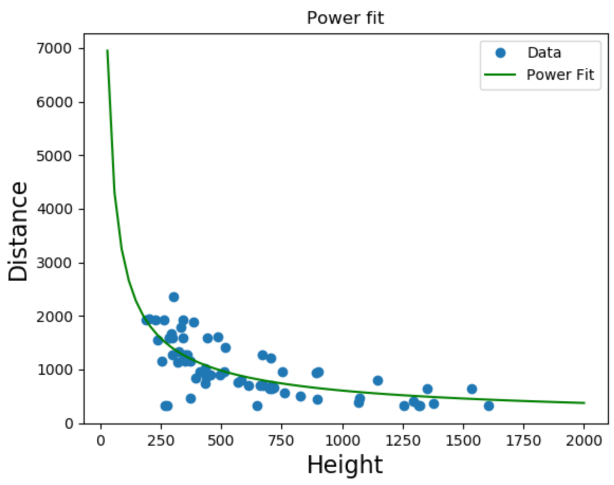


Figure 4.21: Power Regression

Power fit produces the required curve for fitting our data and it explains very well the relation between the distance and the width of the car. Moreover, it has the highest R-Squared = 51.404%
which means it can fit most of our data (72.66% of it) and can predict the distance while the width is given from the detected car.

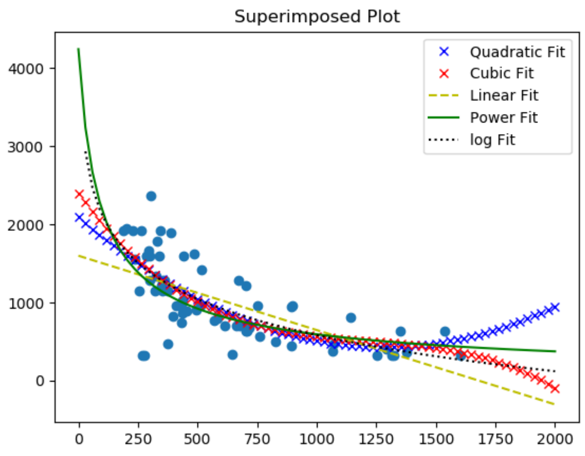


Figure 4.22: All Model Output

Fitness of linear fit= 44.100%
Fitness of quadratic fit= 50.654%
Fitness of Cubic fit= 51.30%
Fitness of Lograthmitc fit= 50.77%
Fitness of power fit= 51.404%

4.6.6 Check the main problem

After adding the height power fit equation instead of the width and testing the same case. The results are not dependant on the width anymore, thus, the car is seen as a fixed height from both the side view and the front view.

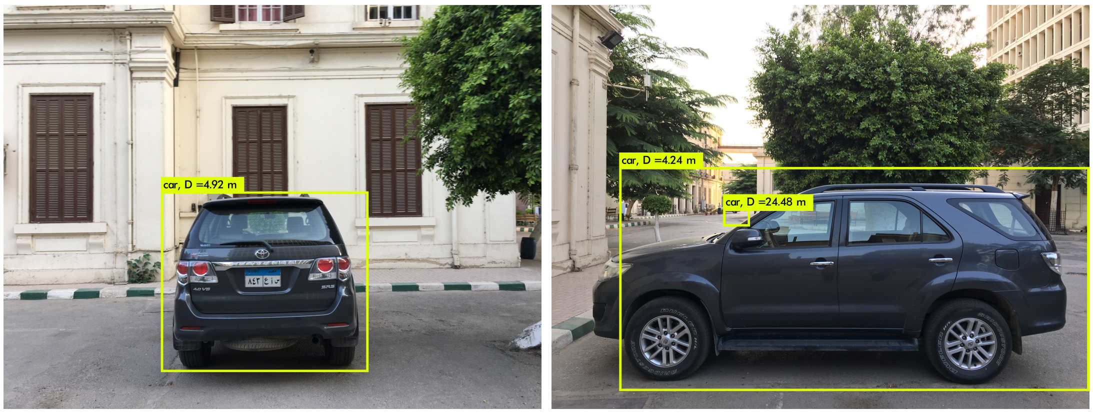


Figure 4.23: Height Results

Average Error in Height approach
The height approach gives 25%
or more error which is not good as expected

Test Case Problem in Height approach
If an only part from the car is captured, the boundary box will appear smaller and the distance will be falsely far.

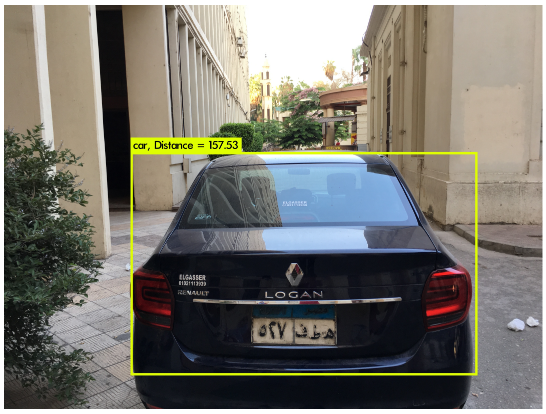


Figure 4.24: Partial Problem

The solution to the problems:

we still suffer from the partial problem so we come up with idea use the two variable to help each other in different cases as if we get a partial hight of the car we take the width the main parameter in our calculation and visa versa so we think to improve our model and use Multi Regression. We used two statistical tools called STATA and MiniTab.

4.7 Multiple-Regression

Linear regression models the relationship between a dependent variable and one or more explanatory variables using a linear function. If two or more explanatory variables have a linear relationship with the dependent variable, the regression is called a multiple linear regression. Multiple regression, on the other hand, is a broader class of regressions that encompasses linear and nonlinear regressions with multiple explanatory variables.

Regression analysis is a common way to discover a relationship between dependent and explanatory variables. However, this statistical relationship does not mean that the explanatory variables cause the dependent variable; it rather speaks of some significant association in the data. Linear regression attempts to draw a line that comes closest to the data by finding the slope and intercept that define the line and minimize regression errors. However, many relationships in data do not follow a straight line, so statisticians use nonlinear regression instead.

4.7.1 Height and Width Multiple Regression Model

The statistical model used is a multiple regression made to indicate the relationship between the width, height and distance. To reach the functional form of the regression model two fitting tool were used MiniTab and STATA tools

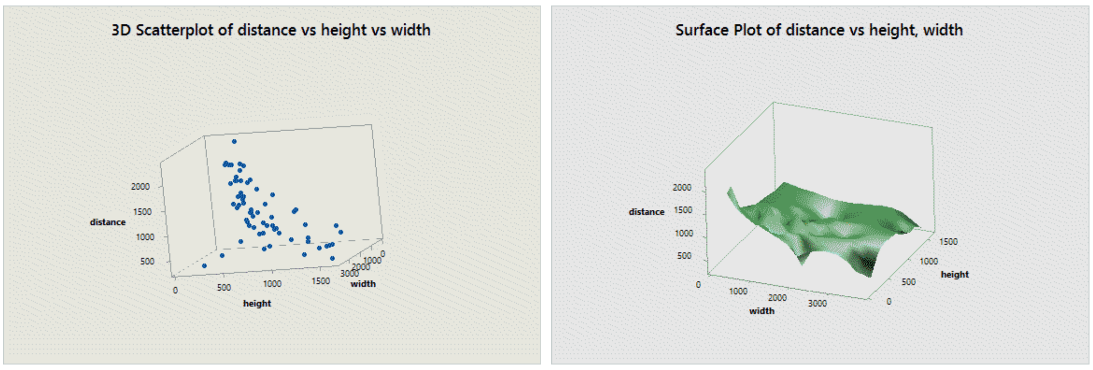
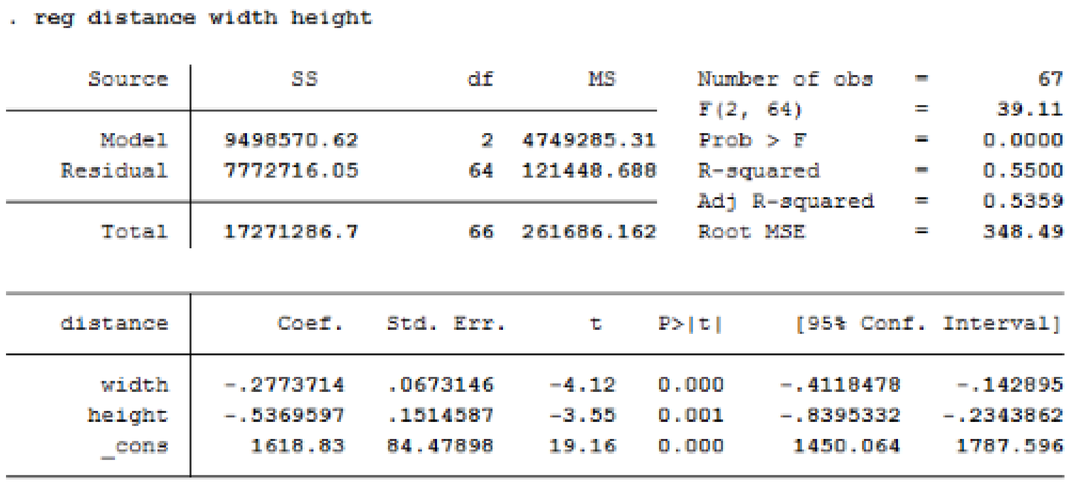


Figure 4.25: 3D scatter plot

Multiple Regression Linear Model

To have the Linear Model, MiniTab tool was used to deterime its equation. First insert your data from your excel sheet by copying each column into the data section. Then go to stat choose regression and fit line regression. Then this table will be showen



```
. reg distance width height

      Source |       SS           df       MS      Number of obs   =        67
-------------+----------------------------------   F(2, 64)        =     39.11
       Model |  9498570.62         2  4749285.31   Prob > F        =    0.0000
    Residual |  7772716.05        64  121448.688   R-squared       =    0.5500
-------------+----------------------------------   Adj R-squared   =    0.5359
       Total |  17271286.7        66  261686.162   Root MSE        =    348.49

------------------------------------------------------------------------------
    distance |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
       width |  -.2773714   .0673146    -4.12   0.000    -.4118478    -.142895
      height |  -.5369597   .1514587    -3.55   0.001    -.8395332   -.2343862
       _cons |    1618.83   84.47898    19.16   0.000     1450.064    1787.596
------------------------------------------------------------------------------
```

Figure 4.26: Model summery

The model equation is

$$Distance = 1618.83 - 0.2773714 \cdot width - 0.5369597 \cdot height \tag{4.11}$$

Having a 55% R-squared
and 53% adjusted R-squared

R-squared and adjusted R-squared

One major difference between R-squared and the adjusted R-squared is that R-squared supposes that every independent variable in the model explains the variation in the dependent variable. It gives the percentage of explained variation as if all independent variables in the model affect the dependent variable. Adjusted R-squared, on the other hand, gives the percentage of variation explained by only those independent variables that in reality affect the dependent variable. R-squared cannot verify whether the coefficient ballpark figure and its predictions are prejudiced. It also does not show if a regression model is satisfactory; it can show an R-squared figure for a good model, or a high R-squared figure for a model that doesn't fit.

The adjusted R-squared compares the descriptive power of regression models that include diverse numbers of predictors. **Every predictor added to a model increases R-squared and never decreases it.** Thus, a model with more terms may seem to have a better fit just for the fact that it has more terms, while the adjusted R-squared compensates for the addition of variables and only increases if the new term enhances the model above what would be obtained by probability and decreases when a predictor enhances the model less than what is predicted by chance. In an overfitting condition, an incorrectly high value of R-squared, which leads to a decreased ability to predict, is obtained. This is not the case with the adjusted R-squared.

The adjusted R-squared is a modified version of R-squared for the number of predictors in a model. The adjusted R-squared can be negative, but isn't always, while an R-squared value is between 0 and 100 and shows the linear relationship in the sample of data even when there is no basic relationship.

The adjusted R-squared is the best estimate of the degree of relationship in the basic population. To show correlation of models with R-squared, pick the model with the highest limit, but the best and easiest way to compare models is to select one with the smaller adjusted R-squared. Adjusted R-squared is not a typical model for comparing nonlinear models, but multiple linear regressions.

4.7.2 Testing the Multiple Regression linear model on YOLO

Putting the model equation in YOLO and testing it, it was found that it wasn't good as it's acting more likely as the width equation.

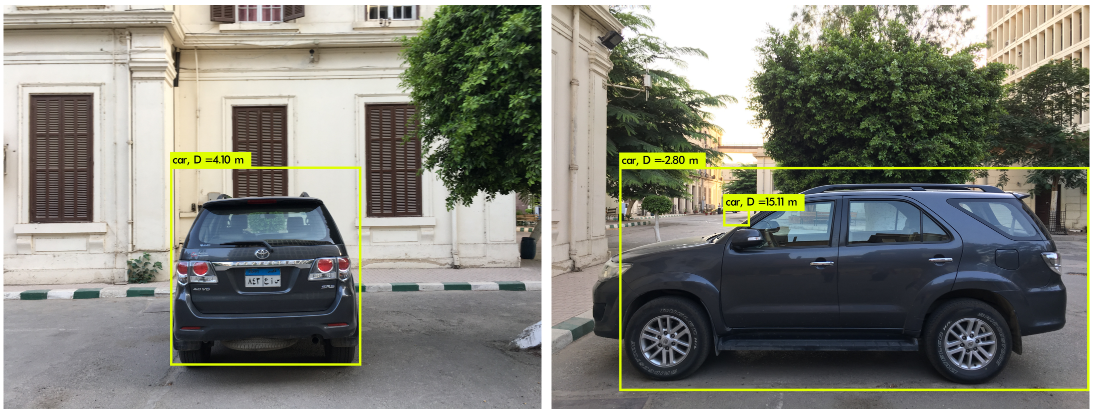


Figure 4.27: Width Problem

4.7.3 Multiple Regression Non Linear Model

Model 1

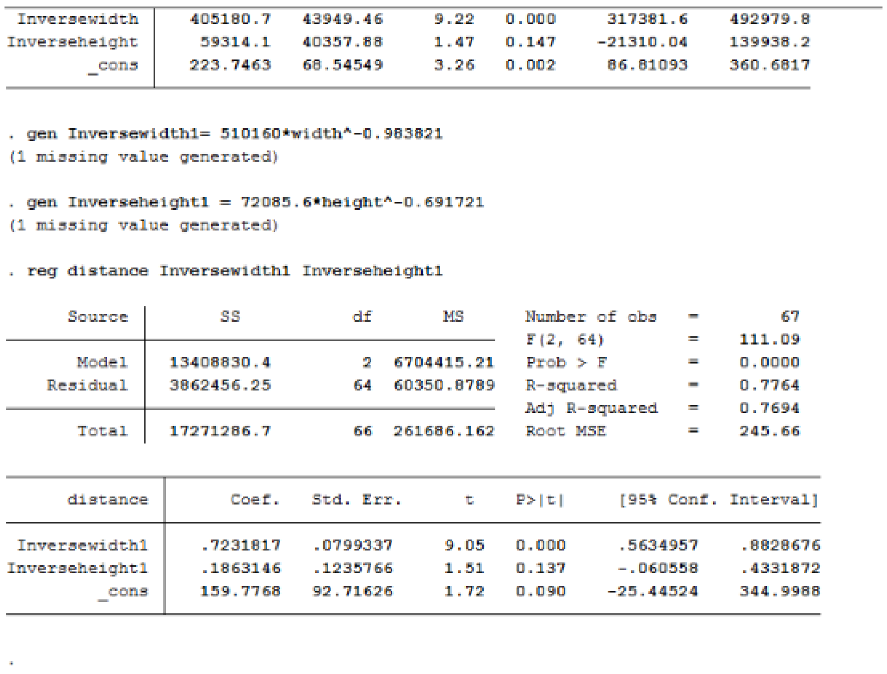
The most appropriate function describing the data is a power function and that was assured by looking at the following rectangle hyperbola shape of the scatter plot between each of the height and width with distance. we put as input the output of our previous work the power function.

The regression model variables that best represent the data and has the highest R seq among other trials: The independent variables

$$inversewidth = 510160 \cdot width^{-0.983821} \tag{4.12}$$

$$inverseheight = 72085.6 \cdot height^{-0.691721} \tag{4.13}$$

the dependent variable = Distance. The error term is mu



```
 Inversewidth │   405180.7   43949.46     9.22   0.000     317381.6    492979.8
Inverseheight │    59314.1   40357.88     1.47   0.147    -21310.04    139938.2
        _cons │   223.7463   68.54549     3.26   0.002     86.81093    360.6817

. gen Inversewidth1= 510160*width^-0.983821
(1 missing value generated)

. gen Inverseheight1 = 72085.6*height^-0.691721
(1 missing value generated)

. reg distance Inversewidth1 Inverseheight1

      Source │       SS           df       MS      Number of obs   =        67
─────────────┼──────────────────────────────────   F(2, 64)        =    111.09
       Model │  13408830.4         2  6704415.21   Prob > F        =    0.0000
    Residual │  3862456.25        64  60350.8789   R-squared       =    0.7764
─────────────┼──────────────────────────────────   Adj R-squared   =    0.7694
       Total │  17271286.7        66  261686.162   Root MSE        =    245.66

──────────────┬────────────────────────────────────────────────────────────────
     distance │      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
──────────────┼────────────────────────────────────────────────────────────────
 Inversewidth1 │   .7231817   .0799337     9.05   0.000     .5634957    .8828676
Inverseheight1 │   .1863146   .1235766     1.51   0.137    -.060558    .4331872
         _cons │   159.7768   92.71626     1.72   0.090    -25.44524    344.9988
──────────────┴────────────────────────────────────────────────────────────────

.
```

Figure 4.28: Model1 summery

The model equation is

$$Distance = 0.72318 \cdot inversewidth + 0.1863 \cdot inverseheight + 159.7768 \tag{4.14}$$

Dependant variable (B1,B2): B1 and Bb2 are positive, thus there is a positive relationship between the power function of width , height and distance .which indicate a negative relationship between distance and both height and width., as STATA automatically run a hypothesis test that assumes that the margin effects of the independent variables on the dependent variable are zero, so there is no correlation observed in the model.

P value : the maximum tolerable level of committing the error of rejecting a correct hypothesis, of each coefficient is examined against an alpha level of 5% .that is we could be a 95% sure of our result, thus accept the null hypothesis, that is here refers to no relationship. While concluding that there is strong statistical evidence of a correlation between the depended variable and independent variables, rejecting the null hypothesis, the p-value of the coefficient should be less than 5% if not we cannot make inference about future data or the population of all data could be generated based on this result of this exact sample.

In our model, the p-value of inverse width coefficient = .005 meaning that there is strong statistical evidence that there is a positive correlation between it and distance. On the other hand, the relationship is not statically significant for B2, inverse height, as its p-value= .137 is higher than .05 at significance level 95% the p-value of B2 is.

Result of R Seq and adjusted R Seq we found that 77% of the variations in distance is explained through the height and width .while adj R2 indicates that the model fits the data by 76% .the two measures refers to a good model that fit the model significantly and explain the variations of distance using the inverse height and inverse width.

Testing the Multiple Regression non-linear model1 on YOLO

Putting the model equation in YOLO and testing it, it was found that it wasn't good as it's acting more likely as the width equation.

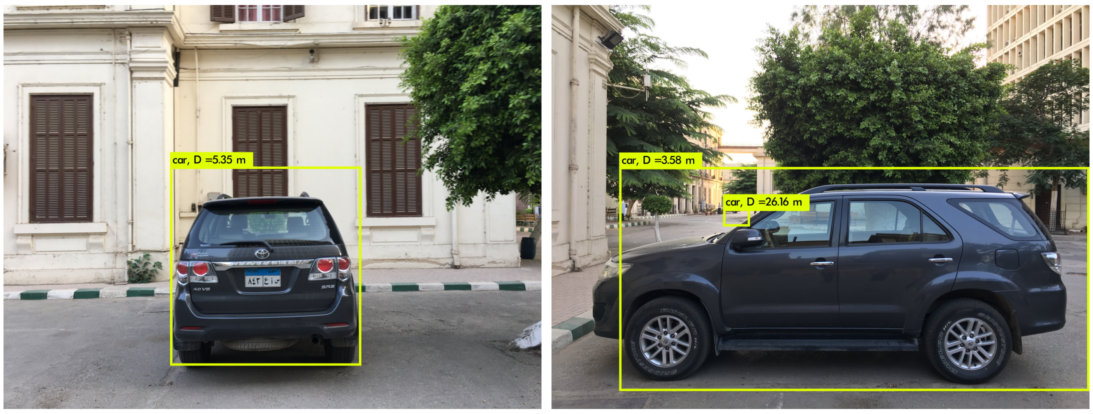
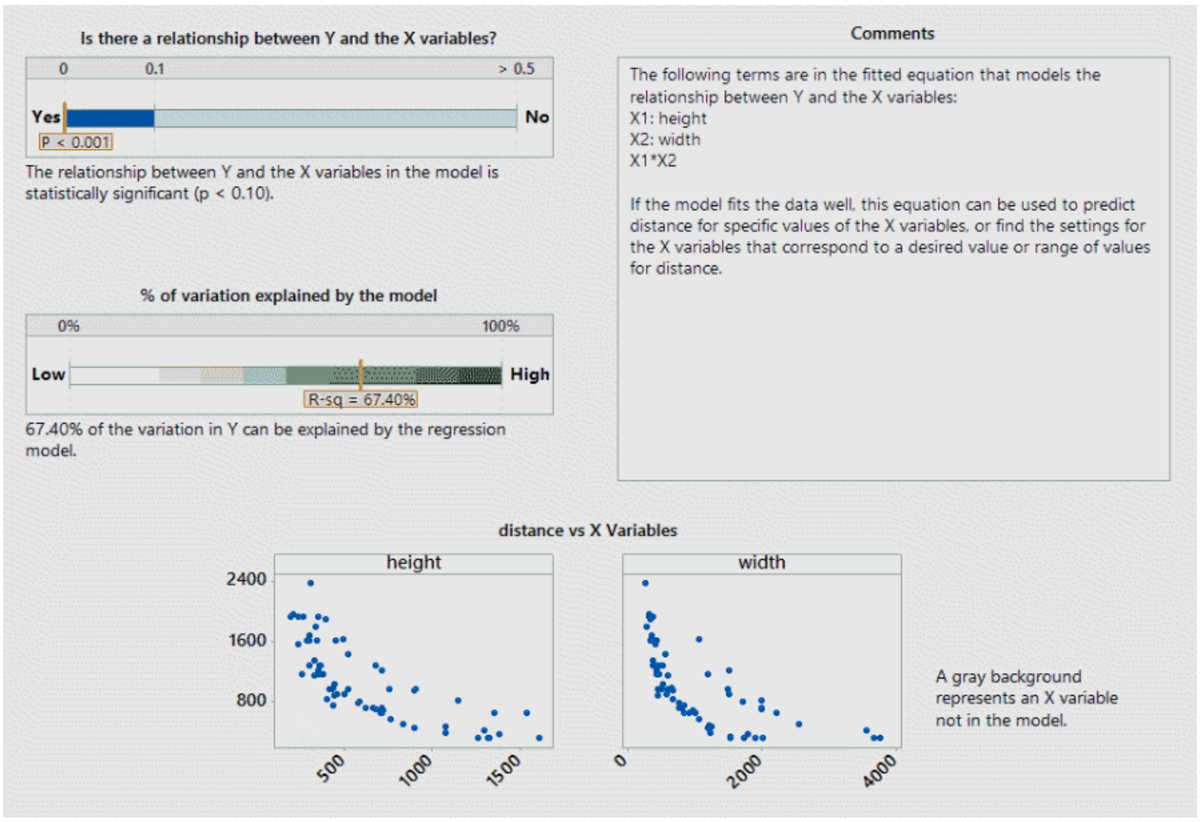


Figure 4.29: Model 1 Results

Model 2

This model was recommeded by the tool itsel, to get the recommedation model, press on "Assistant" choose "Multiple Regression" then "Fit model". The P value for this model is statistically significant. This model uses the height term and the width one in addation to the Interpolation term which is the height multiplied by the width. In the next equation you'll notice that the height has the biggest weight and the interpolation term has the lowest one. That's because this equation tries to fit more than 60 precent of the real data.



The relationship between Y and the X variables in the model is statistically significant (p < 0.10).

67.40% of the variation in Y can be explained by the regression model.

Comments

The following terms are in the fitted equation that models the relationship between Y and the X variables:
X1: height
X2: width
X1*X2

If the model fits the data well, this equation can be used to predict distance for specific values of the X variables, or find the settings for the X variables that correspond to a desired value or range of values for distance.

A gray background represents an X variable not in the model.

Figure 4.30: Model2 Summery Report

The model equation is

$$Distance = 2048 - 1.312 \cdot height - 0.6160 \cdot width) + 0.000490 \cdot hieght \cdot width \quad (4.15)$$

Having a 67.40% R-squared
and 66.30% adjusted R-squared.

This model was built based on the change of adding terms and delete ones till reaching the best fit that describes the curve very well. the two measures refers to a good model that fit the model significantly and explain the variations of distance using the width and height and the interpolation term.
Testing the Multiple Regression non-linear model1 on YOLO

Putting the model equation in YOLO and testing it, it was found that it was great at the special test case that we've experinced before and it's acting more likely as the height equation.

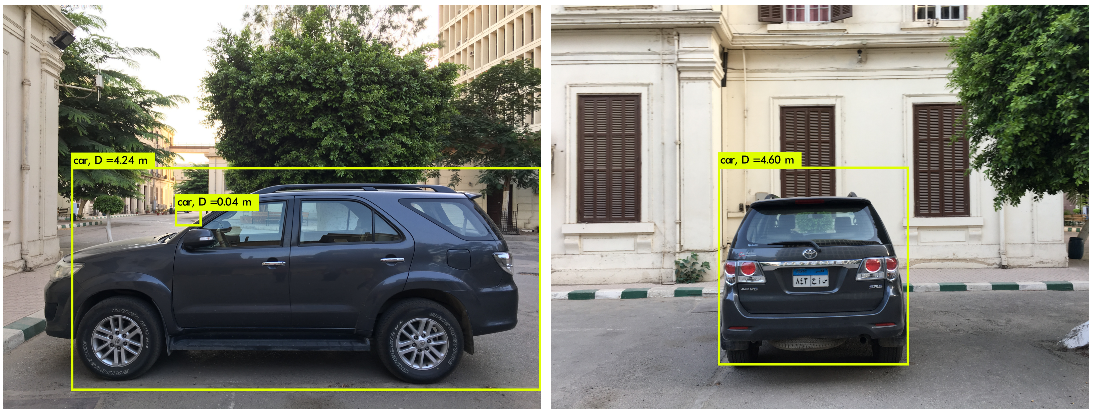


Figure 4.31: Model 2 Results

4.7.4 Mean Square Error

Predictor

If $\hat{\boldsymbol{Y}}$ is a vector of predictions generated from a sample of n data points on all variables, and $\boldsymbol{Y}$ is the vector of observed values of the variable being predicted, then the within-sample MSE of the predictor is computed as

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2.$$

This is an easily computable quantity for a particular sample (and hence is sample-dependent).

Estimator

The MSE of an estimator $\hat{\boldsymbol{\theta}}$ with respect to an unknown parameter $\boldsymbol{\theta}$ is defined as

$$\mathrm{MSE}(\hat{\theta}) = \mathrm{E}_{\hat{\theta}}\left[(\hat{\theta} - \theta)^2\right].$$

This definition depends on the unknown parameter, and the MSE in this sense is a property of an estimator. Since an MSE is an expectation, it is not a random variable. That being said, the MSE could be a function of unknown parameters, in which case any estimator of the MSE based on estimates of these parameters would be a function of the data and thus a random variable. If the estimator is derived from a sample statistic and is used to estimate some population statistic, then the expectation is with respect to the sampling distribution of the sample statistic.

The MSE can be written as the sum of the variance of the estimator and the squared bias of the estimator, providing a useful way to calculate the MSE and implying that in the case of unbiased estimators, the MSE and variance are equivalent.

4.8 MSE Problem

The mean square error measured for the **width approach** was

Linear error= 24.13
Quadrtic error= 13.93
Cubic error= 8.85
Log error= 11.41
Power error= 5.45

The mean square error measured for the **Height approach** was

Linear error= 16.13
Quadrtic error= 10.386
Cubic error= 9.247
Log error= 9.966
Power error= 8.24
The mean square error measured for the **Multiple Regression** was

Linear multiple regression= 16.44
Model1 non-linear multiple regression= 4.506
Model2 non-linear multiple regression= 9.364

The error is considered to be an important measure to see how close these approaches to the real life and after rejecting the width approach due to its problems, only height approach was left to be used. To improve the error and the R-squared values for this approach. More data must be taken so that the function is described better and the R-squared gets better which leads to the minimum error.

4.9 New More Data for height approach

After careful observation, we found that the closest distances are not covered very well, thus it needs more data at this region to cover this area. A 35 new image with distance measured was added to the pervious data to have a total 92 images for regression and 35 for testing.

4.9.1 Removing the Outliers

An outlier is an observation that lies an abnormal distance from other values in a random sample from a population. In a sense, this definition leaves it up to us to decide what will be considered abnormal. Before abnormal observations can be singled out, it is necessary to characterize.

Two activities are essential for characterizing a set of data:

• Examination of the overall shape of the graphed data for important features, including symmetry and departures from assumptions.
• Examination of the data for unusual observations that are far removed from the

mass of data. These points are often referred to as outliers. Two graphical techniques for identifying outliers, scatter plots and box plots.

After previewing the scatter plot of the data, outliers were removed from the data to characterize the fitting equation.

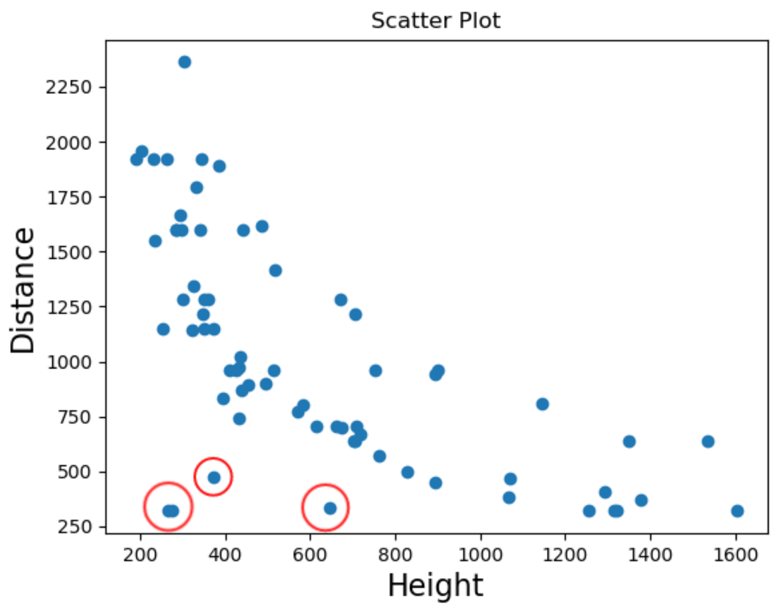


Figure 4.32: Hight Scatter data

4.9.2 Recalculation of R-Squared and MSE

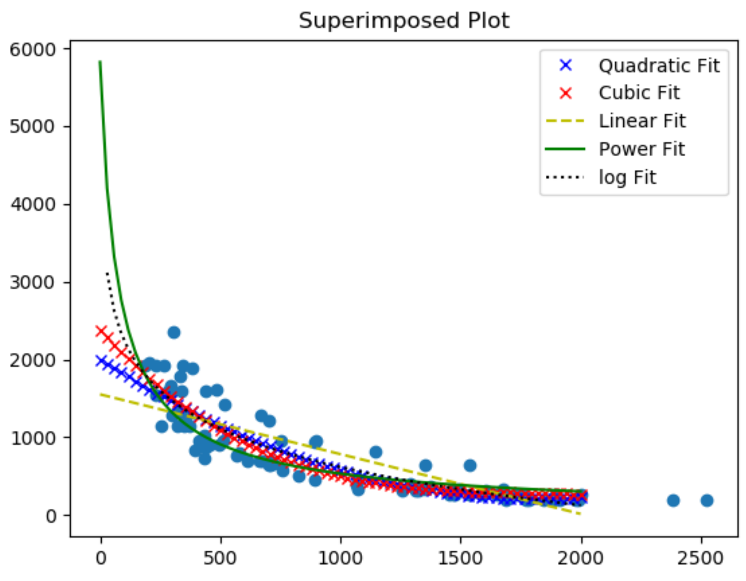


Figure 4.33: R-square Recalcuate

The R_squared measured for the new data was
R-Squared for the linear regression =70.415%
R-Squared for the quadratic regression =79.93%
R-Squared for the Cubic regression=81.917%
R-Squared for the log regression=81.2789%
R-Squared for the power regression=81.38%

Although the cubic fit has the highest fitting, it was rejected due to the negative problem mentioned previously. The second high r_squared was for power fit. For the second time the power regression was chosen.

The mean square error measured was

Linear error= 17.76
Quadrtic error= 9.765
Cubic error= 6.766
Log error= 7.9545
Power error= 4.519

The power equation parameters were completely changed to fit the new curve. Thus, the old equation was discarded and replaced by a new one

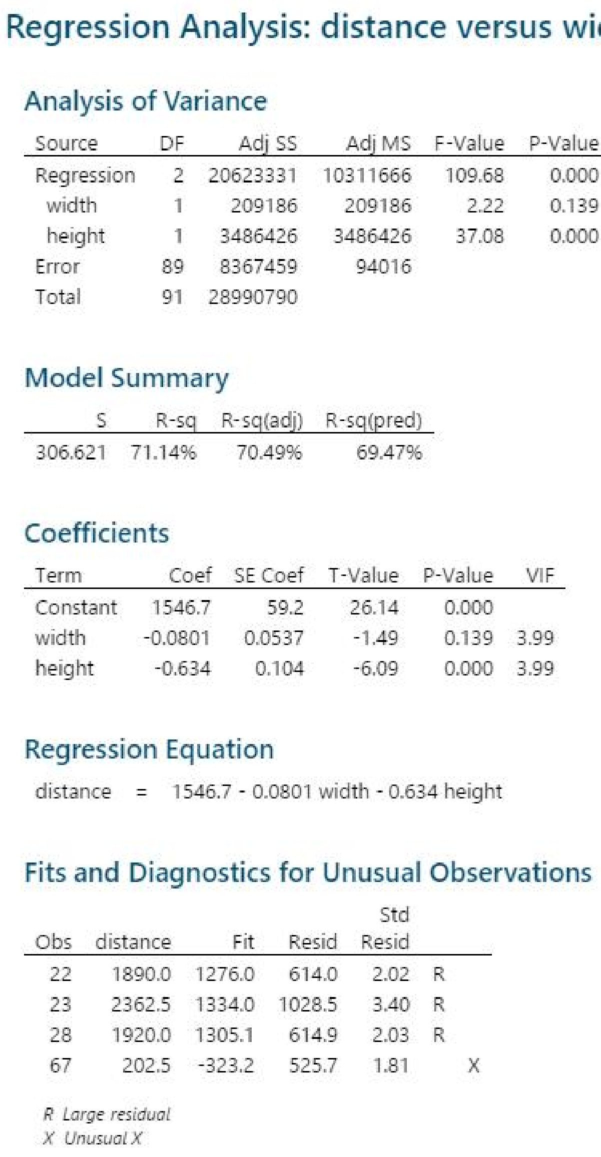
$$y = 1.81665897 \cdot 10^5 \cdot x^{-8.40291565 \cdot 10^{-1}} \tag{4.16}$$

4.10 More Data For Multiple Regression

The new data were a 35 new image with distance measured was added to the pervious data to have a total 92 images for regression and 35 for testing. For the multiple regression models, three models were extracted this time.

4.10.1 Multiple Regression Linear Model

As previously explained to have the Linear Model, MiniTab tool was used to deterime its equation. First insert your data from your excel sheet by copying each column into the data section. Then go to stat choose regression and fit line regression. Then this table will be showen



Regression Analysis: distance versus width, height

Analysis of Variance

Source	DF	Adj SS	Adj MS	F-Value	P-Value
Regression	2	20623331	10311666	109.68	0.000
width	1	209186	209186	2.22	0.139
height	1	3486426	3486426	37.08	0.000
Error	89	8367459	94016		
Total	91	28990790			

Model Summary

S	R-sq	R-sq(adj)	R-sq(pred)
306.621	71.14%	70.49%	69.47%

Coefficients

Term	Coef	SE Coef	T-Value	P-Value	VIF
Constant	1546.7	59.2	26.14	0.000	
width	-0.0801	0.0537	-1.49	0.139	3.99
height	-0.634	0.104	-6.09	0.000	3.99

Regression Equation

distance = 1546.7 - 0.0801 width - 0.634 height

Fits and Diagnostics for Unusual Observations

Obs	distance	Fit	Resid	Std Resid	
22	1890.0	1276.0	614.0	2.02	R
23	2362.5	1334.0	1028.5	3.40	R
28	1920.0	1305.1	614.9	2.03	R
67	202.5	-323.2	525.7	1.81	X

R Large residual
X Unusual X

Figure 4.34: Model Result

The model equation is

$$Distance = 1546.7 - 0.0801 \cdot width - 0.634 \cdot height \tag{4.17}$$

As observed above. Width is insignificant and from the fits and diagonstics table. An unusual obersvations and large residuals were detected. This model will not be used.

4.10.2 Multiple Regression Non-Linear Models

Model 1

This model was recommeded by the tool itsel, to get the recommedation model, press on "Assistant" choose "Multiple Regression" then "Fit model". The P value for this model is statistically significant. The equation suposed to hae both the width term and the height term. Unusually, the recommended model only had one term which is the **height term** as the recommendation showed that width is not effective as height. The euation is descriped as

$$Distance = 2001.1 - 1.961 \cdot height + 0.000536 \cdot height^2 \tag{4.18}$$

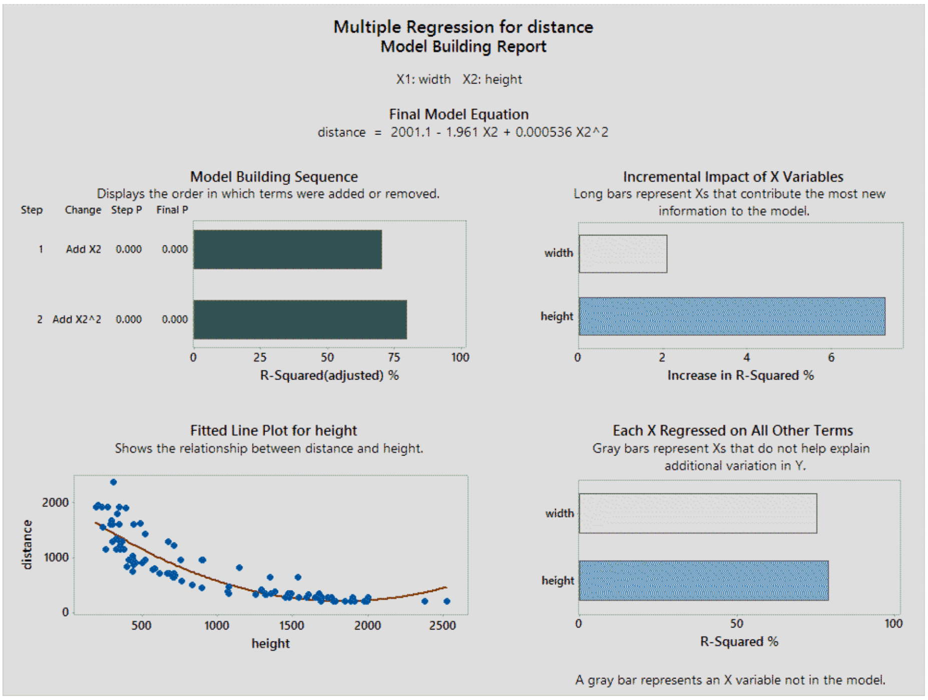


Figure 4.35: Model Result

Model 2

Another non-linear model was found to be:

$$Distance = -0.4893 \cdot width - 1.106 \cdot height + 0.00032 \cdot height \cdot width + 1951.751 \tag{4.19}$$

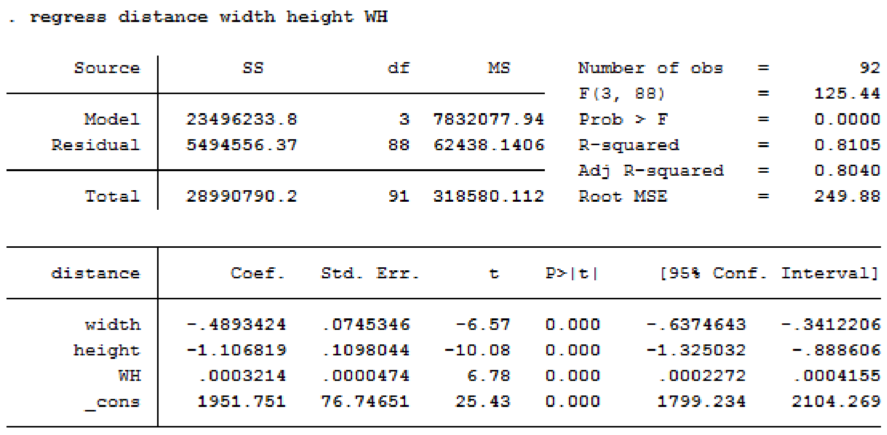
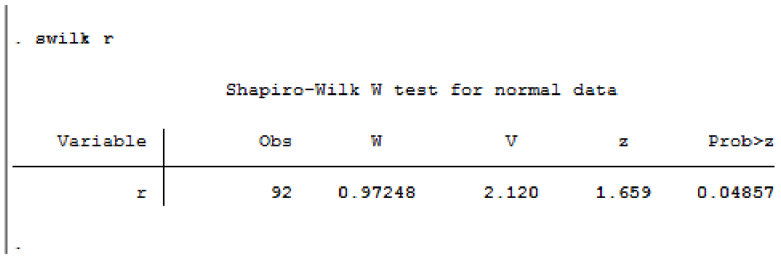

```
. regress distance width height WH

      Source |       SS           df       MS      Number of obs   =        92
-------------+----------------------------------   F(3, 88)        =    125.44
       Model |  23496233.8         3  7832077.94   Prob > F        =    0.0000
    Residual |  5494556.37        88  62438.1406   R-squared       =    0.8105
-------------+----------------------------------   Adj R-squared   =    0.8040
       Total |  28990790.2        91  318580.112   Root MSE        =    249.88

------------------------------------------------------------------------------
    distance |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
       width |  -.4893424   .0745346    -6.57   0.000    -.6374643   -.3412206
      height |  -1.106819   .1098044   -10.08   0.000    -1.325032    -.888606
          WH |   .0003214   .0000474     6.78   0.000     .0002272    .0004155
       _cons |   1951.751   76.74651    25.43   0.000     1799.234    2104.269
------------------------------------------------------------------------------
```

Figure 4.36: Model Result

From the linear model, as residuals were not normally distributed. A transformation needed which is to use inverse data.



```
. swilk r

                   Shapiro-Wilk W test for normal data

    Variable |        Obs       W           V         z       Prob>z
-------------+------------------------------------------------------
           r |         92    0.97248      2.120     1.659    0.04857

.
```

Figure 4.37: Model Result

The test rejects the hypothesis of normality when the p-value is less than or equal to 0.05

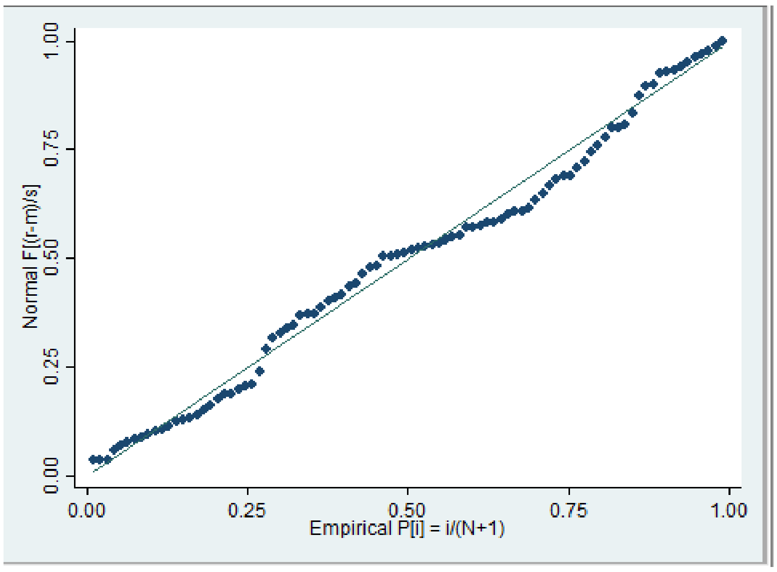
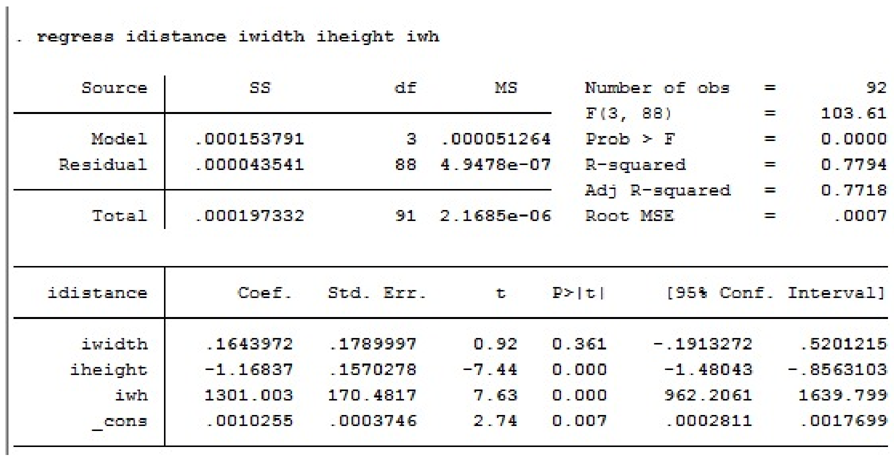


Figure 4.38: S shape error

Transformation to inverse



```
. regress idistance iwidth iheight iwh

      Source |       SS           df       MS      Number of obs   =        92
-------------+----------------------------------   F(3, 88)        =    103.61
       Model |  .000153791         3  .000051264   Prob > F        =    0.0000
    Residual |  .000043541        88  4.9478e-07   R-squared       =    0.7794
-------------+----------------------------------   Adj R-squared   =    0.7718
       Total |  .000197332        91  2.1685e-06   Root MSE        =     .0007

------------------------------------------------------------------------------
   idistance |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
      iwidth |   .1643972   .1789997     0.92   0.361    -.1913272    .5201215
     iheight |   -1.16837   .1570278    -7.44   0.000     -1.48043   -.8563103
         iwh |   1301.003   170.4817     7.63   0.000     962.2061    1639.799
       _cons |   .0010255   .0003746     2.74   0.007     .0002811    .0017699
------------------------------------------------------------------------------
```

Figure 4.39: Inverse Model

This model explained 77.94% of the variations in the data (R-squared = 77.94% And adjusted R-squared= 77.18%). With 95% confidence level.

$$Distance = 0.1643 \cdot iwidth - 1.1683 \cdot iheight + 1301.003 \cdot iwh + 0.0010255 \quad (4.20)$$

As shown from the output, the model has 4 parameters (the intercept, width's coefficient, height's coefficient, the interaction term between width and height's coefficient). The only insignificant one which has no significantly difference effect from the intercept is the width.

Taking into consideration that the inverse of the data is used to ensure validation of regression model (it's assumed to have normally distributed residuals)

Chapter 5

Future Work

5.1 Camera parameters

5.1.1 Focal Length

Focal length, usually represented in millimeters (mm), is the basic description of a photographic lens. It is not a measurement of the actual length of a lens, but a calculation of an optical distance from the point where light rays converge to form a sharp image of an object to the digital sensor or 35mm film at the focal plane in the camera. The focal length of a lens is determined when the lens is focused at infinity.

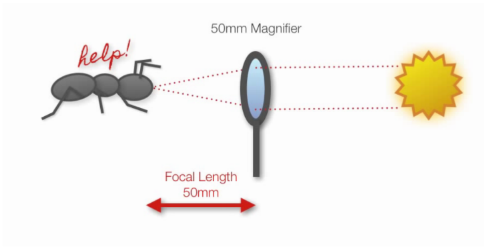


Figure 5.1: Focal Length

When a lens is used to collect light coming from an object, it will create an image. The size and location of the image depending on the location of the object and the focal length of the lens. Equation 1 defines the relationship between the object distance (o), the focal length (f), and the image distance (i). The image distance is simply the distance from the object to the thin lines. The image distance is the distance from the thin lens to the image. The focal length is a characteristic of the thin lens, and it specifies the distance at which parallel rays come to a focus.

$$1/o + 1/i = 1/f \tag{5.1}$$

The focal length tells us the angle of view—how much of the scene will be captured—and the magnification—how large individual elements will be. The longer the focal length, the narrower the angle of view and the higher the magnification. The shorter the focal length, the wider the angle of view and the lower the magnification. as shown in figure 2

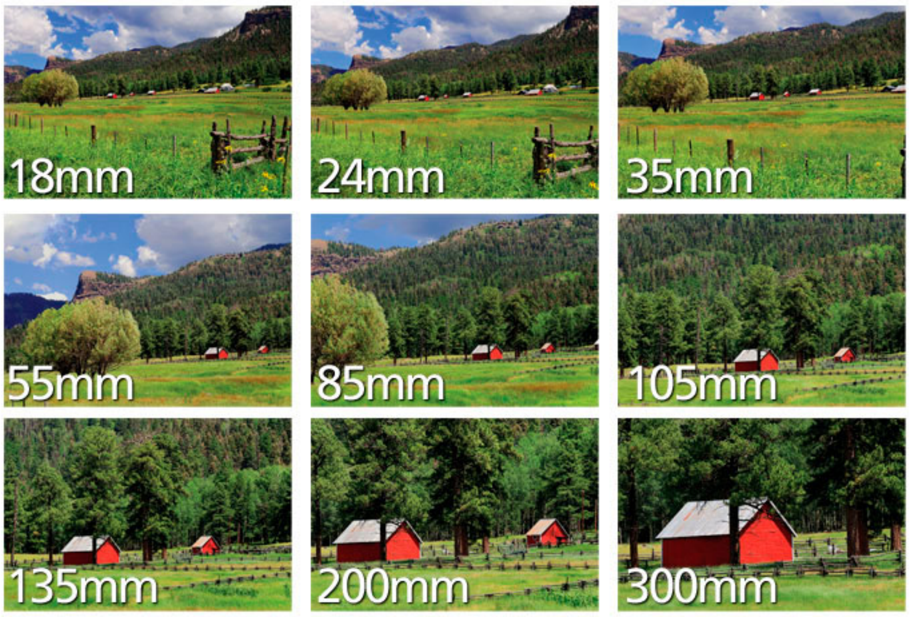


Figure 5.2: diffrent focal lengths

When objects move away from infinity the image moves away from the focal point. Figure 3 demonstrates the object-image relationship for a positive lens. In the figure, two rays are traced from the object. One ray is traveling parallel to the optical, which hits the lens and is bent so that it passes through the focal point of the lens. A second ray is traveling from the object to the center of the lens, which will pass straight through the lens. The point at which the two points intersect is where the light comes to a focus, and that is why axis re the image is formed.

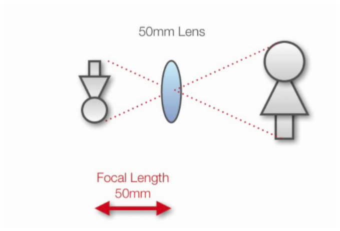


Figure 5.3: Object being imaged by a lens

to get a better zoom without losing the resolution. for example a face . to get a nice picture of it we need a bigger focal length. Figure 4 demonstrates how we get a bigger zoom when using a bigger focal length at the same distance.

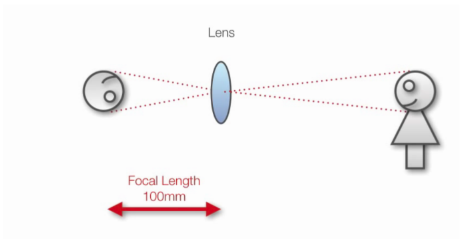


Figure 5.4: Object being imaged by a 100mm lens

and for the opposite case to get the wider view of people and a landscape then we need a smaller focal length as shown in figure 5

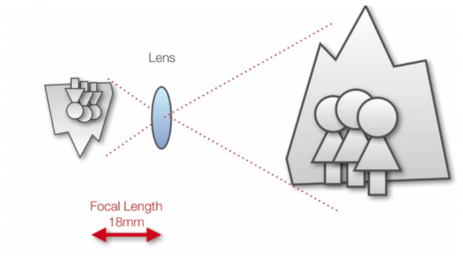


Figure 5.5: Object being imaged by an 18mm lens

5.1.2 Angel Of View

A Fixed Focal Length Lens, also known as a conventional or entocentric lens, is a lens with a fixed Angular Field of View (AFOV). By focusing the lens for different working distances, differently sized Fields of View (FOV) can be obtained, though the viewing angle is held constant. AFOV is typically specified as the full angle (in degrees) associated with the horizontal dimension (width) of the sensor that the lens is to be used with.

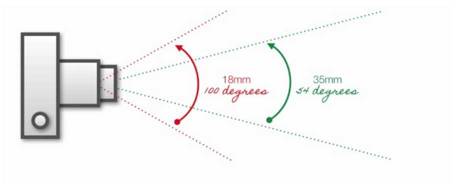


Figure 5.6: Angle of view

The focal length of a lens defines the lens's angular field of view. For a given sensor size, the shorter the focal length, the wider the angular field of the lens. Additionally, the shorter the focal length of the lens, the shorter the distance needed to obtain the same FOV compared to a longer focal length lens.
For a simple, thin convex lens, the focal length is the distance from the back of the lens to the plane of the image formed of an object placed infinitely far in front of the lens. From this definition, it can be shown that the angular field of view

of a lens is related to the focal length (Equation 2), where f is the focal length in millimeters and h is the horizontal dimension of the sensor in millimeters (Figure 6).

$$AFOV = 2 * tan^{-1}(h/2f) \tag{5.2}$$

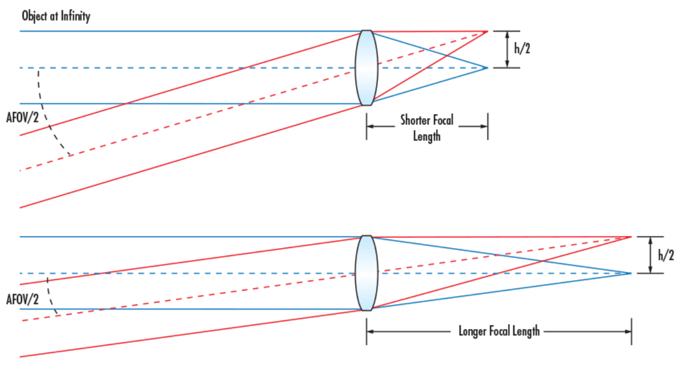


Figure 5.7: For a given Sensor Size, h, Shorter Focal Lengths produce Wider AFOV's

5.1.3 Crop factor

Crop factor describes the size difference between a 35mm film frame and your camera's sensor. For example, if your camera has a crop factor of 2, it means that a 35mm film frame is twice as large as your camera's sensor.

Modern digital cameras are fitted with sensors of varying size. The best digital SLRs have sensors which are the same size as 35mm film, so they have a crop factor of 1 (this is known as "full-frame"). At the other end of the scale, digital compact cameras have very small sensors and high crop factors of 5 of 6. The higher the crop factor, the more noticeable the "zooming in" effect for a given focal length.

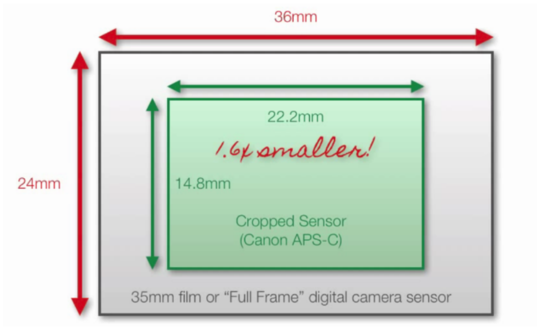


Figure 5.8: crop factor

Effective Focal Length

If you multiply a lens's focal length by the camera's crop factor, you get the "equivalent focal length", which is the focal length needed to produce the same angle of view on a 35mm camera. This is why you might also hear crop factor referred to as the "focal length multiplier" (or "FLM").

For example, a 50mm lens on a 1.6 crop factor camera has an effective focal length of 80mm, because 50 x 1.6 = 80. If you fitted a 50mm to a 1.6 crop factor camera, you'd need a 35mm lens to get the same field of view.

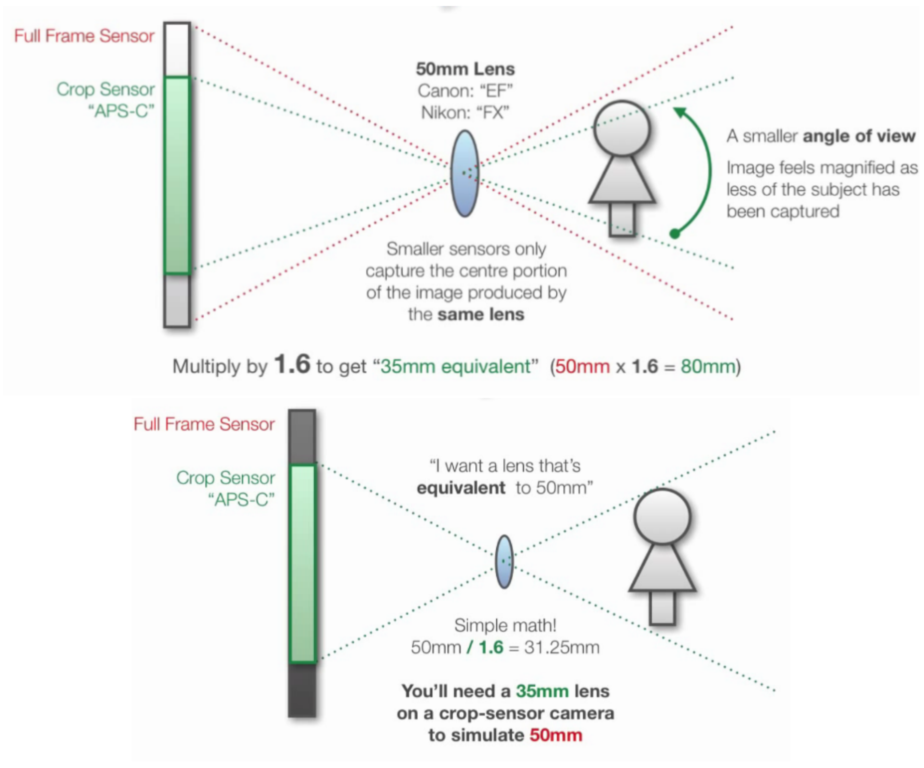

Full Frame Sensor
Crop Sensor "APS-C"
50mm Lens
Canon: "EF"
Nikon: "FX"
A smaller angle of view
Image feels magnified as less of the subject has been captured
Smaller sensors only capture the centre portion of the image produced by the same lens
Multiply by 1.6 to get "35mm equivalent" (50mm x 1.6 = 80mm)
Full Frame Sensor
Crop Sensor "APS-C"
"I want a lens that's equivalent to 50mm"
Simple math!
50mm / 1.6 = 31.25mm
You'll need a 35mm lens on a crop-sensor camera to simulate 50mm

Figure 5.9: Effective Focal Length

5.2 Speed

introduction
We want to make a speed indicator system for surrounding cars in order to avoid any accident may happen

Our idea is to calculate the relative speed of surrounding cars, by knowing our speed and the relative speed of other cars and the distance between them, we can predict the dangerous situation if the cars won't be able to stop in the right time, and warn the driver
Algorithm

we get relative speed by calculating the difference in distance of the sounding car in the time of a single frame

$$V = dD/dT$$

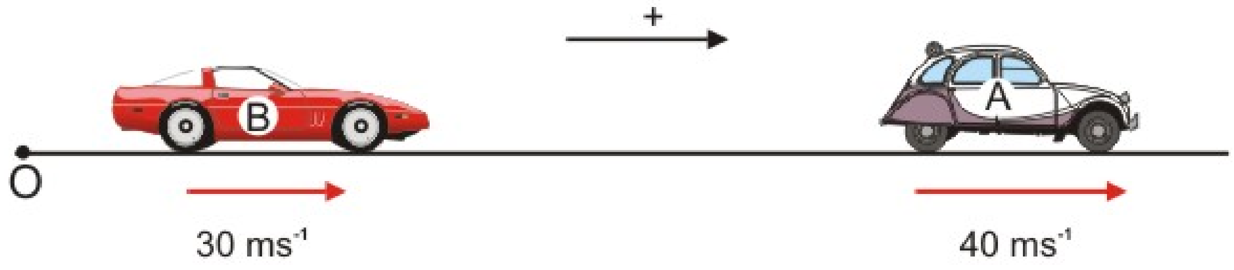


Figure 5.10: Relative Speed

$$\vec{v}_{B|A} = \vec{v}_B - \vec{v}_A.$$

Thus, we get the speed of each car on the road we detected and calculate its speed by referring to ours, and by knowing the difference in time of each frame in our system 30 fps we make the calculation easier

But we still have a problem, as in yolo the detection accrued per frames, each frame is separated from the others so we need to make a tracking algorithm to maintain tracking the car through frames and calculate its relative speed as we mentioned above

5.2.1 Traking

we start to sort the cars in each frame by its closeness to ours and by knowing the change in time of each frame is so small ,and also by observing we found the change in distance ¡.1 so we compare each frame by a through to make sure that the same car in each frame and start sort again and so on

5.2.2 Accdidnet avoiding system

our vision for simple accident warning system to help the driver , after we get the distance and relative speed of surrounding cars we can predict if there an accident may be happened, by knowing my speed and the up head car speed and the distance between us , we can know if there's enough time to stop or maneuver or not ,the closer the object is the more dangerous the situation is so we give the driver different warning base on dangerous level .

5.3 Future Work

• **Camera fusion between 4 cameras around the car** you need to collect a separate data set for each side of the cars and to train each side alone to get a higher accuracy . and to make a surrounding view of the car on a single screen .

- **Lane Detection Fusion** you can make the car and the lane detection of the same project and software so that you can start working on giving signals to the actuators like breaking and steering to break or to avoid the lane .

- **Increasing The classes of Detection** you can start working on different types of classes that relative to vehicles like Bus, Motorcycles, bicycle, people, traffic lights and traffic signs Especially the traffic lights so that you could take actions on the color of the light and the breaking lights of the others vehicles.

- **Implementing The Whole Neural Network** you can start working on re Implementing the layer of the Neural network so that you could increase the process and decrease the numbers of layers and neurons .

- **Distance Algorithm** if you followed our algorithm you need to do more training for the regression and to add different parameters like those in the **Camera parameters** section and to Add the **Depth Of Fields** and the **Aperture** to make the estimation more generic to any camera. Or to create a new algorithm for prediction using CNN .

- **Car's Speed Estimation** according to the algorithm of the relative speed estimation in the **Speed** Section you need to implement an **effective Tracking Algorithm** for the cars to get its relative speed and if you have your speed then you can get the absolute speed of the surrounding cars .

- **Sensors Fusion** starting interfacing the Detection algorithm with other Sensors Like ultrasonics for the short Distances and Radars for the long Distances and IRs for the surrounding heated objects for better accuracy.

Chapter 6

Conclusion

At the beginning we started by discovering the field of deep learning through a lot of online courses, by the time we started to understand the whole picture of the project "Real Time Car Detection" and how to implement it, we implemented the common Neural Network (NN) such as (Resnet,VGG16,VGG19,...etc) to classify objects . We spent the first early months in learning and try to test this famous pre-trained Network

We get to the point knowing the suitable architecture used in real time object detection even used in industry and that was "you look only once" "YOLO" neural network, which is considered to be one of the best NNs based on deep learning to detect an object in real time, we started to understand its paper, know the difference between its different versions and the Network architecture

After that we started to prepare for the main stage which is modifying the Network to meet our needs and try to make it even better. We started to get enough data for cars and its annotations through trying almost every possible way , such as the online dataset from different websites as (Udacity ,Voc pascal .... etc) but we faced a problem which was the output format of each one is different from our neural network format so we started to make a python and bash scripts to filter the data and transform its format to darknet format , we also get thousands of pictures from the street by recording a real time video, crop it into frames and manually annotate them by BBox tool

Modifying Yolo was the next step. To be able to obtain our purpose to detect only cars , we noticed that the Version2 wasn't real time enough as it was around 7 Frames Per Second and even after modifying it we made it goes up 9 FPS, the output of detection wasn't bad it was about 70 % accurate to detect cars

Moving to the hardware part. Nivida jetson tx2 was used. We installed the suitable jetpack, the essential libraries and other programs to make it run the code and connect it with its internal camera

Yolo version3 was released and we started to understand the changes, this time the difference was huge, after we modified it to our custom object we could achieve up to 20 FPS and we trained the model on a bigger data this time. The final output

was much better the accuracy around 90%
We finally achieve the main target which was the real-time and high accuracy, we moved to add more features

Depth estimation, as the main target of this project is to build a module in a self-driving car project, so we need to get the distance of surrounding cars ,we start searching for different approaches that we can use to get the distance, through different approaches we found, using an extra sensor or an expensive stereo camera but we found all these approaches want to be suitable for our case so we try to think in a different way, we create new algorithm to detect the cars using only mono camera, we use the boundary box that generated from our NN by finding the relation between the distance and the dimension of the boundary box

We use the regression technique and try to fit the distance with the width of the car with different models (linear, quadratic , cubic, logarithmic , power) we found the power function is the best with 11% error, but we face a problem of the width changing with car orientation as it gets bigger as the car moves to the side view.

We needed the parameter that only depends on the car itself, not its orientation

Thus, we tried the height and repeat the process. It was much better but still there a problem If an only part from the car is captured, the boundary box will appear smaller and the distance will be falsely far.
Thus, we try to make a multiple-regression model using width and height by using STATE and Minitab

We create new models for the different cases and chose the best one, the result was much better and doesn't suffer from partial problem

Moreover, we wanted to add a new feature, as we already got the distance of the surrounding cars. We can also get their relative speed with respect to ours, by calculating the difference in distances over frames, then we create a simple tracking algorithm by sorting the cars in each frame by knowing through. To make sure that is the same car we calculated its speed

Besides, this project is developed to be able to help the driver to avoid accidents, as we already calculate the speed and the distance. We can predict if there was something wrong, as if there's not enough distance to be able to stop in the right time or if I get close for the upheld cars too much ,that may cause an accident, so we divided the whole situation into three regions of dangerous and as the driver get much closer to the forward car it gives a warning

At the end, we could achieve a real-time detection with accuracy above 90% and can run in 20 FPS, we add new features as distance estimation with only mono camera and get accuracy 93% ,we also could generate from all above an accident warning system by calculating speed of surrounding cars ,we still hope to develop this project more and more in future as self-driving cars and deep learning is the nearly future we are going to .

Chapter 7

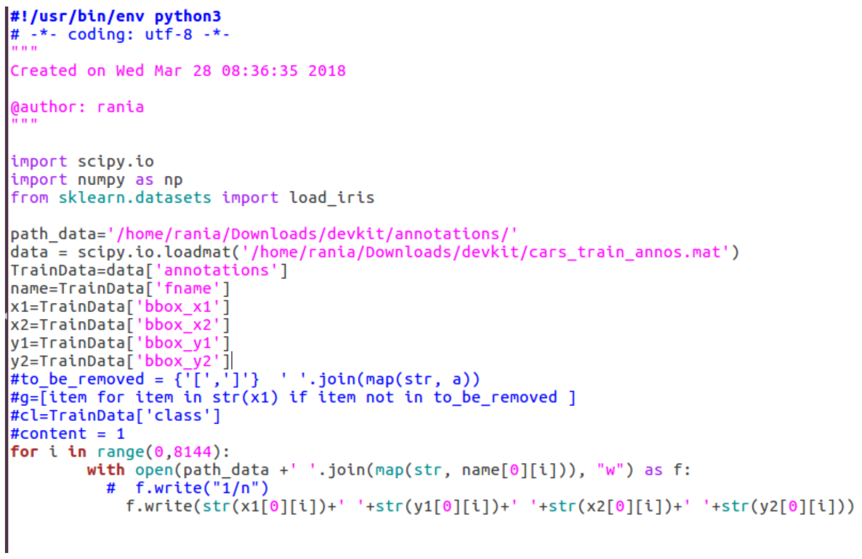
References

7.1 References

[1] J. Redmon et al., ”You Only Look Once: Real-Time Object Detection,” [EB/OL].https://pjreddie.com/darknet/yolov1/.
[2] J. Redmon, et al., ”You only look once: real-time object detection,” [C]//Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2016: 779-788.
[3] R. Girshick, et al., ”Rich feature hierarchies for accurate object detection and semantic segmentation,” [C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2014: 580-587.
[4] R. Girshick, ”Fast r-CNN,” [C]//Proceedings of the IEEE international conference on computer vision. 2015: 1440-1448.
[5] S. Ren et al., ”Faster R-CNN: Towards real-time object detection with region proposal networks,” [C]//Advances in neural information processing systems. 2015: 91-99.0
[6] S. Bell, C. L. Zitnick, K. Bala, and R. Girshick. Inside- outside net: Detecting objects in context with skip pooling and recurrent neural networks. arXiv preprint arXiv:1512.04143, 2015.
[7] J. Redmon et al., ”You Only Look Once: Real-Time Object DetectionV2,” [EB/OL].https://pjreddie.com/darknet/yolov2/.
[8] Fischer, Robert and Tadic-Galeb, Biljana. Optical System Design. New York: McGraw-Hill, 2000.
[9] Greivenkamp, John. Field Guide to Geometric Optics. Bellingham:SPIE, 2003.

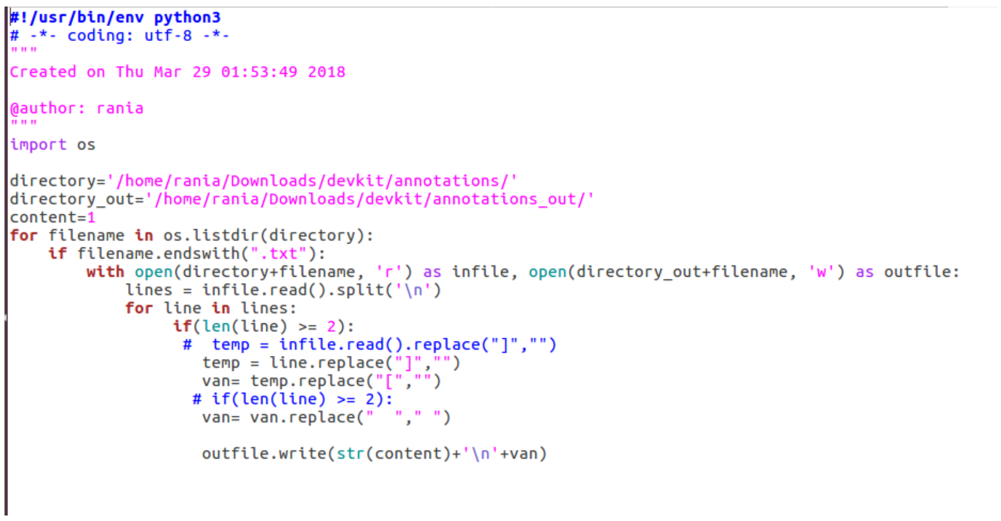
Appendex

annot.py



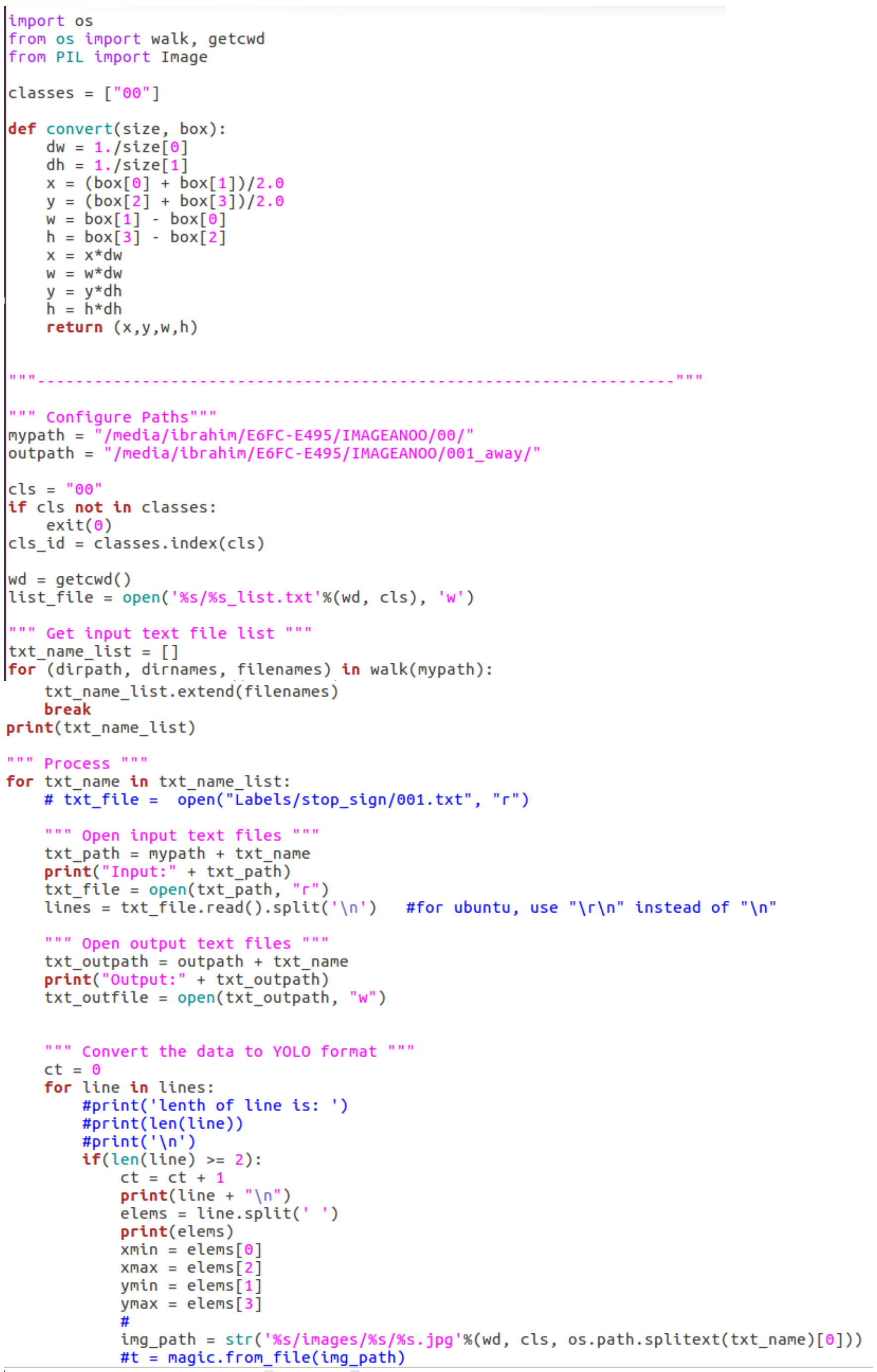
```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed Mar 28 08:36:35 2018

@author: rania
"""

import scipy.io
import numpy as np
from sklearn.datasets import load_iris

path_data='/home/rania/Downloads/devkit/annotations/'
data = scipy.io.loadmat('/home/rania/Downloads/devkit/cars_train_annos.mat')
TrainData=data['annotations']
name=TrainData['fname']
x1=TrainData['bbox_x1']
x2=TrainData['bbox_x2']
y1=TrainData['bbox_y1']
y2=TrainData['bbox_y2']
#to_be_removed = {'[',']'}  ' '.join(map(str, a))
#g=[item for item in str(x1) if item not in to_be_removed ]
#cl=TrainData['class']
#content = 1
for i in range(0,8144):
        with open(path_data +' '.join(map(str, name[0][i])), "w") as f:
          #  f.write("1/n")
            f.write(str(x1[0][i])+' '+str(y1[0][i])+' '+str(x2[0][i])+' '+str(y2[0][i]))
```

annot2.py

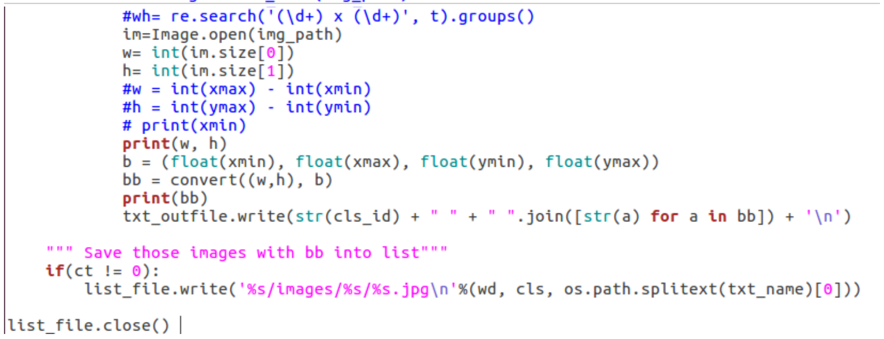


```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Thu Mar 29 01:53:49 2018

@author: rania
"""
import os

directory='/home/rania/Downloads/devkit/annotations/'
directory_out='/home/rania/Downloads/devkit/annotations_out/'
content=1
for filename in os.listdir(directory):
    if filename.endswith(".txt"):
        with open(directory+filename, 'r') as infile, open(directory_out+filename, 'w') as outfile:
            lines = infile.read().split('\n')
            for line in lines:
                 if(len(line) >= 2):
                  #  temp = infile.read().replace("]","")
                    temp = line.replace("]","")
                    van= temp.replace("[","")
                   # if(len(line) >= 2):
                    van= van.replace("  "," ")

                    outfile.write(str(content)+'\n'+van)
```

convert.py

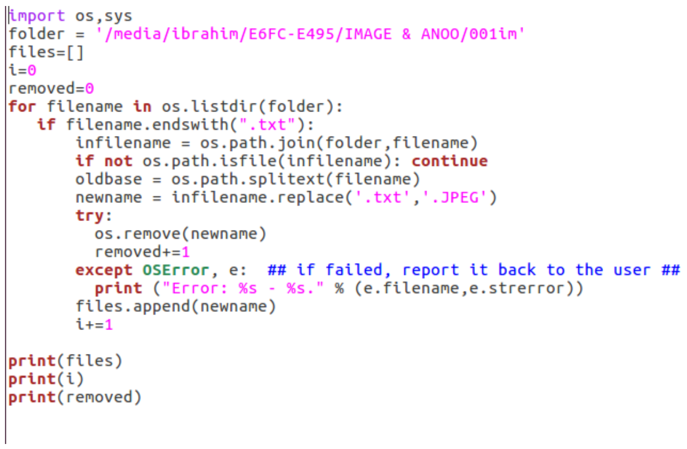


```python
import os
from os import walk, getcwd
from PIL import Image

classes = ["00"]

def convert(size, box):
    dw = 1./size[0]
    dh = 1./size[1]
    x = (box[0] + box[1])/2.0
    y = (box[2] + box[3])/2.0
    w = box[1] - box[0]
    h = box[3] - box[2]
    x = x*dw
    w = w*dw
    y = y*dh
    h = h*dh
    return (x,y,w,h)


"""-------------------------------------------------------------------"""

""" Configure Paths"""
mypath = "/media/ibrahim/E6FC-E495/IMAGEANOO/00/"
outpath = "/media/ibrahim/E6FC-E495/IMAGEANOO/001_away/"

cls = "00"
if cls not in classes:
    exit(0)
cls_id = classes.index(cls)

wd = getcwd()
list_file = open('%s/%s_list.txt'%(wd, cls), 'w')

""" Get input text file list """
txt_name_list = []
for (dirpath, dirnames, filenames) in walk(mypath):
    txt_name_list.extend(filenames)
    break
print(txt_name_list)

""" Process """
for txt_name in txt_name_list:
    # txt_file =  open("Labels/stop_sign/001.txt", "r")

    """ Open input text files """
    txt_path = mypath + txt_name
    print("Input:" + txt_path)
    txt_file = open(txt_path, "r")
    lines = txt_file.read().split('\n')   #for ubuntu, use "\r\n" instead of "\n"

    """ Open output text files """
    txt_outpath = outpath + txt_name
    print("Output:" + txt_outpath)
    txt_outfile = open(txt_outpath, "w")


    """ Convert the data to YOLO format """
    ct = 0
    for line in lines:
        #print('lenth of line is: ')
        #print(len(line))
        #print('\n')
        if(len(line) >= 2):
            ct = ct + 1
            print(line + "\n")
            elems = line.split(' ')
            print(elems)
            xmin = elems[0]
            xmax = elems[2]
            ymin = elems[1]
            ymax = elems[3]
            #
            img_path = str('%s/images/%s/%s.jpg'%(wd, cls, os.path.splitext(txt_name)[0]))
            #t = magic.from_file(img_path)
```



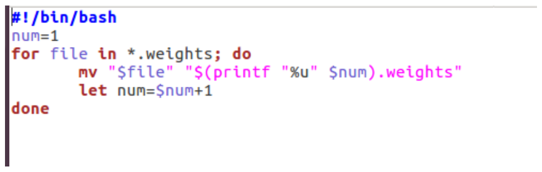
```python
            #wh= re.search('(\d+) x (\d+)', t).groups()
            im=Image.open(img_path)
            w= int(im.size[0])
            h= int(im.size[1])
            #w = int(xmax) - int(xmin)
            #h = int(ymax) - int(ymin)
            # print(xmin)
            print(w, h)
            b = (float(xmin), float(xmax), float(ymin), float(ymax))
            bb = convert((w,h), b)
            print(bb)
            txt_outfile.write(str(cls_id) + " " + " ".join([str(a) for a in bb]) + '\n')

    """ Save those images with bb into list"""
    if(ct != 0):
        list_file.write('%s/images/%s/%s.jpg\n'%(wd, cls, os.path.splitext(txt_name)[0]))

list_file.close()
```

deletefiles.py



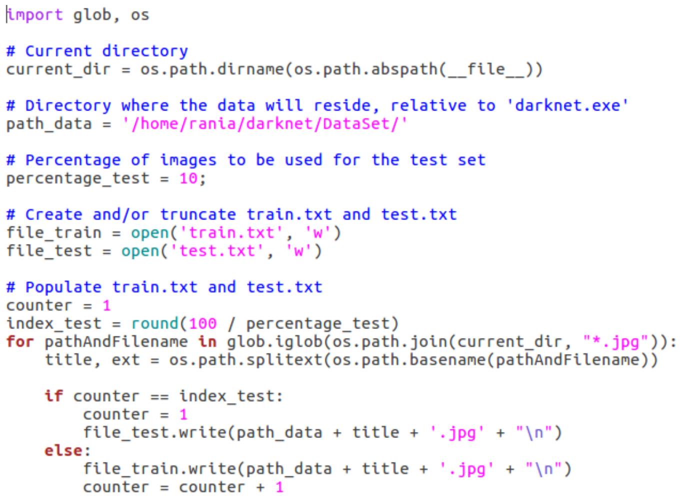
```python
import os,sys
folder = '/media/ibrahim/E6FC-E495/IMAGE & ANOO/001im'
files=[]
i=0
removed=0
for filename in os.listdir(folder):
   if filename.endswith(".txt"):
       infilename = os.path.join(folder,filename)
       if not os.path.isfile(infilename): continue
       oldbase = os.path.splitext(filename)
       newname = infilename.replace('.txt','.JPEG')
       try:
         os.remove(newname)
         removed+=1
       except OSError, e:  ## if failed, report it back to the user ##
         print ("Error: %s - %s." % (e.filename,e.strerror))
       files.append(newname)
       i+=1

print(files)
print(i)
print(removed)
```

rename.sh



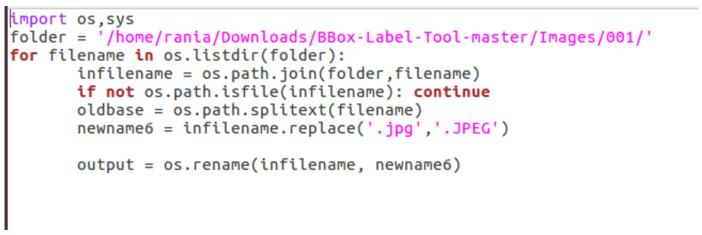
```bash
#!/bin/bash
num=1
for file in *.weights; do
       mv "$file" "$(printf "%u" $num).weights"
       let num=$num+1
done
```

process.py



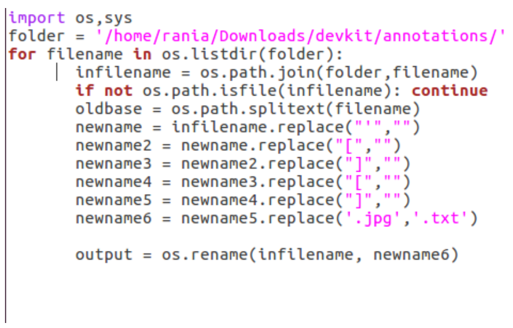
```python
import glob, os

# Current directory
current_dir = os.path.dirname(os.path.abspath(__file__))

# Directory where the data will reside, relative to 'darknet.exe'
path_data = '/home/rania/darknet/DataSet/'

# Percentage of images to be used for the test set
percentage_test = 10;

# Create and/or truncate train.txt and test.txt
file_train = open('train.txt', 'w')
file_test = open('test.txt', 'w')

# Populate train.txt and test.txt
counter = 1
index_test = round(100 / percentage_test)
for pathAndFilename in glob.iglob(os.path.join(current_dir, "*.jpg")):
    title, ext = os.path.splitext(os.path.basename(pathAndFilename))

    if counter == index_test:
        counter = 1
        file_test.write(path_data + title + '.jpg' + "\n")
    else:
        file_train.write(path_data + title + '.jpg' + "\n")
        counter = counter + 1
```

tojpg.py



```
import os,sys
folder = '/home/rania/Downloads/BBox-Label-Tool-master/Images/001/'
for filename in os.listdir(folder):
       infilename = os.path.join(folder,filename)
       if not os.path.isfile(infilename): continue
       oldbase = os.path.splitext(filename)
       newname6 = infilename.replace('.jpg','.JPEG')

       output = os.rename(infilename, newname6)
```

tojepg



```
import os,sys
folder = '/home/rania/Downloads/devkit/annotations/'
for filename in os.listdir(folder):
       infilename = os.path.join(folder,filename)
       if not os.path.isfile(infilename): continue
       oldbase = os.path.splitext(filename)
       newname = infilename.replace("'","")
       newname2 = newname.replace("[","")
       newname3 = newname2.replace("]","")
       newname4 = newname3.replace("[","")
       newname5 = newname4.replace("]","")
       newname6 = newname5.replace('.jpg','.txt')

       output = os.rename(infilename, newname6)
```

Distance

algorithm of regression to get the best model

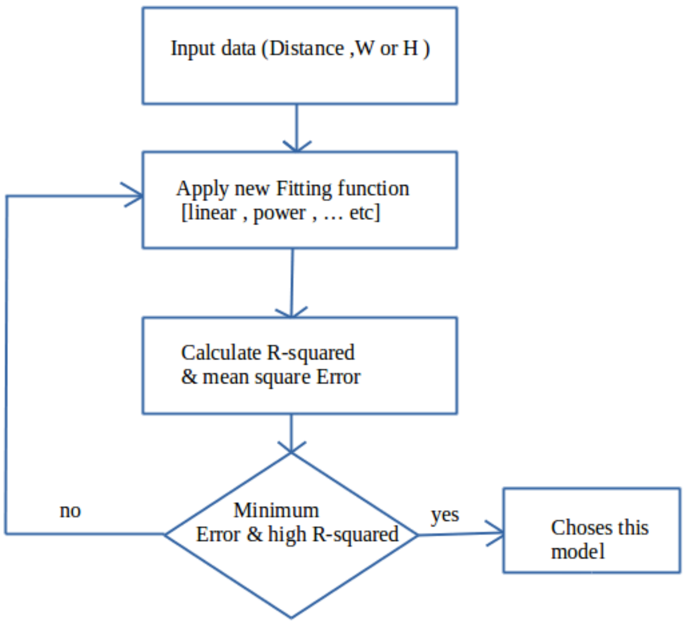


Figure 7.1: Model algorithm