

People's Democratic Republic of Algeria

The Ministry of Higher Education and Scientific Research

University of Mohamed El-Bachir El-Ibrahimi - Bordj Bou Arreridj

Faculty of Sciences and Technology

Department Electronic' MCIL'

Thesis

Presented to get

The master's diploma

Faculty: Science and technology

Specialty: Industrial Electronic

By

- Belaalia Asma
- Namoune Khaoula

Entitled

Object Detection Using YOLO

Supported on:

Before a jury composed of:

Last name & First name	Grade	Quality	Establishment
Dr. BEKKOUCHE Tewfik	Dr.	President	Univ-BBA
Dr. SID AHMED Soumia	Dr.	supervisor	Univ-BBA
Dr. AZOUG Seif Eddine	Dr.	examiner	Univ-BBA
Dr. BOUDCICHE Djamel	Dr.	Co-supervisor	Univ-BBA

College year 2022/2023

Acknowledgements

First, we would like to praise Allah the Almighty, the Most Gracious, and the Most Merciful for His blessing given to me during my study and in completing this thesis.

We would like to express my gratitude and sincere thanks to my principal supervisor **Dr.SID AHMED Soumia**, for his guidance, advice, feedback and continuous support throughout the whole thing. I would also like to thank my associate supervisor **Dr. BOUDECHICHE DjamelEddine**, for his guidance, and feedback on improving our thesis.

We are very much grateful to all teachers of the University of Mohamed Elbachir El Ibrahimi, who provided us with the tools necessary for the success of our university studies during my course.

Finally yet importantly, we would like to thank our families for supporting us throughout our life. Without their love and encouragement, we would not have finished this thesis.

Table of contents

Introduction	5
Chapter 1: Deep learning.....	6
1.1 Introduction.....	1
1.2 Artificial intelligence	1
1.2.1 Choosing between Machine Learning and Deep Learning	2
1.3 Deep learning.....	2
1.4 Neural networks.....	3
1.4.1 Neuron from Biology to AI	3
1.4.2 Neural network parameters and hyper-parameters	4
1.4.3 Improving neural networks.....	5
1.5 Deep learning Architecture	7
1.5.1 Supervised Deep Learning.....	7
1.5.2 Unsupervised learning architecture	12
1.6 Conclusion	13
Chapter 2: Object detection	14
2.1 Introduction.....	15
2.2 Object Detection	15
2.3 Object detection Algorithms	16
2.3.1 One stage object detection Algorithm	18
2.3.2 Two Stage Object Detection Algorithm	18
2.4 YOLO (You Only Look Once).....	20
2.4.1 Overview of the YOLOv5 Architecture	20
2.5 Main evaluation metrics.....	22
2.6 Conclusion	23
Chapter 3: Models implementation and results of experiments	33
3.1 Introduction.....	26
3.2 Software and libraries used in the implementation.....	26
3.2.1 Software.....	26
3.2.2 Programming language.....	27
3.2.3 Frameworks	27
3.2.4 Libraries.....	28
3.3 Data.....	28
3.4 Convolution Neural Network Implementation	30
3.4.1 Training CNN Model.....	32
3.5 Transfer Learning Implementation	39
3.5.1 Result interpretation ResNet model.....	40
3.6 Comparative Analysis between the Results of CNN and ResNet Architecture.....	47
3.7 YOLO implementation	47
3.7.1 The difference between YOLOv5 and YOLOv8	48
3.7.2 YOLOv5 results.....	48
3.7.3 YOLOv8 results.....	53

3.7.4	YOLOv5 and YOLOv8 Results Interpretation.....	57
3.8	Conclusion	58

List of Figures

Figure 1.1:	Relationship between AI, ML and DL.....	2
Figure 1.2:	comparing a Machine Learning approach to categorizing vehicles (left) with Deep Learning (right).....	2
Figure 1.3:	The biological neurons VS the Artificial Neural Network.	4
Figure 1.4:	dropout technique.....	6
Figure 1.5:	CNN Architecture.	8
Figure 1.6:	Example of Convolution Operation.	9
Figure 1.7:	Graph of Sigmoid activation.....	10
Figure 1.8:	Rectified Linear Unit (Relu) activation function.	11
Figure 1.9:	Max-pooling.....	11
Figure 1.10:	fully connected layer.....	12
Figure 2.1:	localizing object.	15
Figure 2.2:	classifying objects.	16
Figure 2.3:	classification of object detection categories and algorithms.....	17
Figure 2.4:	SSD single shot detector model.	18
Figure 2.5:	Fast R-CNN.	19
Figure 2.6:	Faster R-CNN.	19
Figure 2.7:	YOLO Architecture.	21
Figure 2.8:	Intersection over union.	22
Figure 3.1:	Dataset.....	28
Figure 3.2:	Datasets Annotated.	30
Figure 3.3:	CSV file.	30
Figure 3.4:	CNN Architecture.	31
Figure 3.5:	Malaria performance metrics	33
Figure 3.6:	Malaria training results.	33
Figure 3.7:	Drone performance metrics.....	36
Figure 3.8:	Drone training results.....	36
Figure 3.9:	Lung Cancer performance metrics.....	38
Figure 3.10:	Lung Cancer training results.	38
Figure 3.11:	ResNet50 Architecture.....	40
Figure 3.12:	Malaria performance metrics using ResNet50.....	42
Figure 3.13:	Drone performance metrics.....	44
Figure 3.14:	Lung Cancer performance metrics.....	46
Figure 3.15:	Malaria detection using YOLOv5.....	49
Figure 3.16:	MAP and loss plots after training the YOLOv5 Nano on the Malaria detection Dataset.	49
Figure 3.17:	Precision and l Recall plots after training the YOLOv5 Nano on the Malaria detection Dataset.....	50
Figure 3.18:	Drone detection using YOLOv5.	50

Figure 3.19: MAP and loss plots after training the YOLOv5 Nano on the Drone detection Dataset.	51
Figure 3.20: Precision and l Recall plots after training the YOLOv5 Nano on the Drone detection Dataset.	51
Figure 3.21: Lung Cancer detection using YOLOv5.	51
Figure 3.22: Precision and l Recall plots after training the YOLOv5 Nano on the Lung Cancer detection Dataset.	53
Figure 3.23: MAP and loss plots after training the YOLOv5 Nano on the Lung Cancer detection Dataset.	53
Figure 3.24: Malaria detection using YOLOv8.	53
Figure 3.25: MAP and loss plots after training the YOLOv8 Nano on the Malaria detection Dataset.	54
Figure 3.26: Drone detection using YOLOv8.	55
Figure 3.27: Precision and l Recall plots after training the YOLOv8 Nano on the Drone detection Dataset.	56
Figure 3.28: MAP and loss plots after training the YOLOv8 Nano on the Drone detection Dataset.	56
Figure 3.29: Lung Cancer detection using YOLOv8.	56
Figure 3.30: MAP and loss plots after training the YOLOv8 Nano on the Lung Cancer detection Dataset.	57
Figure 3.31: Precision and Recall plots after training the YOLOv8 Nano on the Lung Cancer detection Dataset.	57

List of tables

Table 3-1: The number of the dataset used.	29
Table 3-2: IOU results for Malaria datasets, 6 samples.	32
Table 3-3: IOU results for Drone datasets, 6 samples.	45
Table 3-4: IOU results for Lung Cancer datasets, 6 samples.	38
Table 3-5: IOU results for Malaria datasets using ResNet50, 6 samples.	42
Table 3-6: IOU results for Drone datasets using ResNet50, 6 samples.	44
Table 3-7: IOU results for Lung Cancer datasets using ResNet50, 6 samples.	46
Table 3-8: Total Parameters of ResNet and CNN models.	48
Table 3-9: Total Parameters of YOLOv5n and YOLOv8n models.	49

List of Equation

Equation 1: $\partial k = \partial k - \gamma (d(\text{cost}))/d(\partial k)$	7
Equation 2: $\text{sigmoid}(x) = 1/(1 + e^{(-x)})$	10
Equation 3: $\text{ReLU}(x) = \max(x, 0)$	10
Equation 4: $J(Bp, Bgt) = \text{IOU} = (\text{area}(Bp \cap Bgt)/\text{area}(Bp \cup Bgt))$	22
Equation 5: $\text{MAP} = 1/N \sum_{i=1}^N \text{AP}_i$	23

List of Acronyms

AI : Artificial Intelligence.....	1
CNN : Convolution Neural Network	8
ConvNets : Convolutional Networks.....	7
CPU : Central Processing Units.....	21
CSV : Comma-Separated values.....	29
DL : Deep Learning.....	1
FC : Fully Connected.....	12
Fast R-CNN : Fast Region-based Convolutional Neural Network.....	16
Faster R-CNN : Faster Region-based Convolutional Neural Network.....	16
GPU : Graphic Processor Unit.....	2
IOU : Intersection over Union.....	22
ML : Machine Learning.....	1
MAP : Mean Average Precession.....	9
MSE : Mean Squared Error.....	43
ReLu : Rectified Linear Unit.....	8
RPN : Region Proposal Network.....	19
R-CNN : Regions-Based Convolutional Neural Network.....	19
SSD : Single Shot Detector.....	15
TPU : Tensor Processing Unit.....	26
VGG : short for Visual Geometry Group.....	10
XML : Extensible Markup Langage.....	29
YOLO : You Only Look Once.....	14

Introduction

"Artificial intelligence is just the beginning. It's the opening act of a far more profound transformation driven by technology"

This is what Elon Musk said, as AI continues to advance, it holds the key to unlocking unprecedented levels of efficiency, automation, and problem-solving capabilities across industries. From healthcare to transportation, finance to education, the advancements in machine learning algorithms and neural networks have fueled breakthroughs in computer vision, speech recognition, and decision-making systems, enabling AI to comprehend, analyze, and derive insights from vast amounts of data with unprecedented speed and accuracy.

Within the realm of AI, object detection stands as a compelling example of the impact and potential of this technology. By leveraging advanced algorithms and Deep Learning techniques, object detection has transformed how machines perceive and interact with visual data. Through object detection, AI systems are capable of analyzing images or videos to identify and localize objects of interest. This ability has far-reaching implications across industries and applications. For instance, in healthcare, object detection contributes to the accurate diagnosis of diseases by analyzing medical images, leading to earlier detection and improved patient outcomes.

Our research thesis aims to design and implement various object detection algorithms in order to conduct a comparative analysis between older algorithms and the newest ones, starting with CNN, transfer learning, and then proceed to YOLOv5. Additionally, we will explore the latest algorithm, YOLOv8, which was launched on January 10th, 2023.

To achieve this goal, we have structured our thesis into three chapters:

Chapter 1: in this chapter we will discuss the Deep Learning field and focusing on the CNN architecture.

Chapter 2: we will go deep into object detection techniques

Chapter 3: this chapter contains the comparative result's implementation of our CNN architecture, transfer learning and YOLO.

Chapter 1: Deep learning

Abstract

This chapter provides an overview of Deep Learning and neural network which represent the key components of modern Artificial Intelligence (AI), also we explored Deep Learning where we focused on CNN architecture.

1.1 Introduction

1.2 Artificial intelligence

1.3 Deep learning

1.4 Neural networks

1.5 Deep Learning Architecture

1.6 Conclusion

1.1 Introduction

Today, AI is being used in a wide range of applications, such as natural language processing, computer vision, robotics, and autonomous vehicles. AI development has been fueled by advances in hardware, such as faster processors and more powerful GPUs, as well as breakthroughs in software algorithms and data management. In this chapter we will discuss Deep Learning which is one of the AI subsystems that offer additional features such as the use of Big Data. In the beginning, we provide a brief overview of neural networks that simulate the mechanism of learning in a biological organism. Then we go over different types of Deep Learning then the most popular Deep Learning type called supervised learning, specifically CNN'S.

1.2 Artificial intelligence

Artificial Intelligence (AI) is known as the ability of robots or computer programs to carry out operations that would typically require human intelligence to complete, such as learning, problem solving, and decision-making.

Artificial Intelligence is frequently used as a catchall term for Machine Learning and Deep Learning, but they are not interchangeable terms, where Machine Learning (ML) is a subset of AI that employs statistical techniques to allow machines to learn from data without being explicitly programmed. In other words, it is a method of teaching machines to recognize patterns and predict outcomes based on data on the other side Deep Learning (DL) is a subset of ML that involves the use of multiple-layer artificial neural networks to learn from data. Deep learning algorithms can analyze massive amounts of data and automatically learn to recognize patterns and features that humans are incapable of identifying.

To put it simply, Deep Learning is a subset of the larger field of AI that is a more specialized form of machine learning. Machine Learning models are all Deep Learning models, but not all Deep Learning models are Machine Learning models. Similarly, while all Machine Learning models are a subset of AI, not all AI systems employ Machine Learning[8].

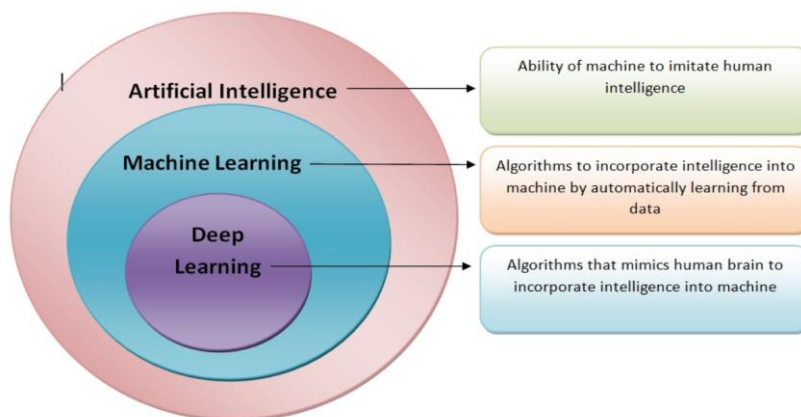


Figure 1.1: Relationship between AI, ML and DL.

1.2.1 Choosing between Machine Learning and Deep Learning

Depending on your application, the volume of data you are processing, and the kind of issue you are trying to solve, Machine learning and Deep Learning offer a variety of methods and models to choose from. For Deep Learning to be effective, the model must be trained on thousands of images, which requires high performance GPUs to process data quickly. When deciding between Machine Learning and Deep Learning, take into account the performance of the GPU and the amount of annotated data available. Machine Learning may be more advantageous than Deep Learning if you lack either of those resources. In order to get accurate findings with Deep Learning, you'll need at least a few thousand images. The model will process all those images faster if it has a high-performance GPU [1].

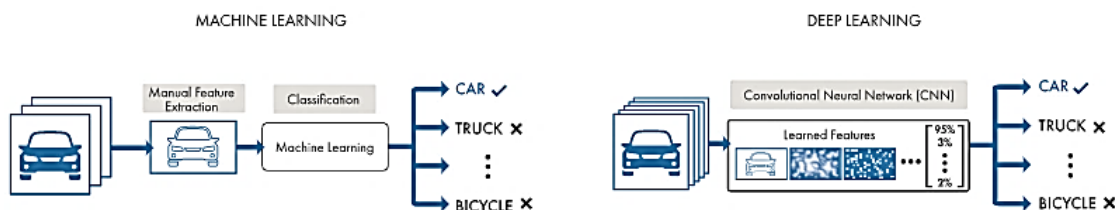


Figure 1.2: comparing a Machine Learning approach to categorizing vehicles (left) with Deep Learning (right).

1.3 Deep learning

Deep learning is a subfield of Machine Learning that involves training artificial neural networks with multiple layers to learn and recognize patterns in data. The term "deep" refers to the fact that these neural networks typically have many layers, allowing them to learn complex representations of data. Nowadays, Deep Learning is used to develop many revolutionary technologies that have a significant impact on our daily lives.

Autonomous driving:

Because of Deep Learning, autonomous driving for vehicles has become a reality. Some algorithms identify road signs while others locate pedestrians. This technology greatly improves road safety and the driving experience of motorists.

In the field of medicine:

Deep Learning can also be used to distinguish between cancerous and non-cancerous tumors. It scans X-ray photos with greater accuracy than the human eye and therefore allows for earlier treatment to increase the patient's chances of recovery.

In the agricultural sector:

Organic agriculture relies on intelligent drones capable of identifying weeds by scanning several hectares of plantations during a flyover. This allows farmers to focus their energy only on the areas that need weeding. Some use cases of Deep Learning algorithm [1].

1.4 Neural networks**1.4.1 Neuron from Biology to AI**

The foundational unit of the human brain is the neuron. The brain is composed of billions of these cells, each of those forms an average of 6,000 connections with other neurons, at its core, the neuron is optimized to receive information from other neurons, process this information in a unique way, and send its result to other cells [3].

This process is summarized in **Figure 1.3**. The neuron receives its inputs along antennae-like structures called dendrites. Each of these incoming connections is dynamically strengthened or weakened based on how often it is used (this is how we learn new concepts!), and it's the strength of each connection that determines the contribution of the input to the neuron's output. After being weighted by the strength of their respective connections, the inputs are summed together in the cell body. This sum is then transformed into a new signal that's propagated along the cell's axon and sent off to other neurons[3].

In a traditional neural network, neurons are fully connected between different layers. Layers that sit between the input layer and output layer are called hidden layers. Each hidden layer contains several neurons that are fully connected to all neurons in the previous layer. The issue with this densely-connected network architecture is that it is not suitable for processing large images. To handle large images, the most preferred approach is to use a convolutional neural network[3].

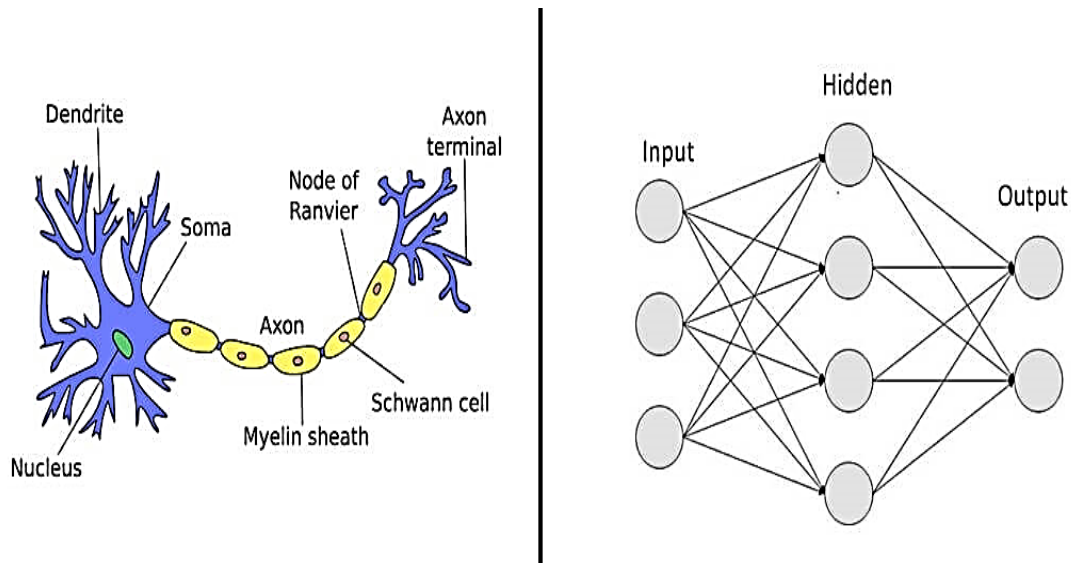


Figure 1.3: The biological neurons VS the Artificial Neural Network.

1.4.2 Neural network parameters and hyper-parameters

The cost function: is a mathematical function that compares the model's predictions to the actual values of the target variable in the training data. The goal of training the model is to minimize this cost function, by adjusting the weights and biases of the neurons in the network [16].

Weights: are neural network variables that decide the strength of connections between neurons. Weights are learned and adjusted during the training process to minimize the cost function. They are typically seeded at random and updated iteratively during training [16].

Bias: A bias parameter is added to each neuron in the network to help it learn more complex relationships between the input and output variables. During training, the bias is also learned and updated along with the weights [16].

Learning rate: is a hyper-parameter defined as the amount of minimization in the cost function in each iteration . It regulates the gradient descent algorithm's step size, which is used to train the neural network's weights. A low learning rate can lead to sluggish convergence, whereas a high learning rate can cause the algorithm to overshoot the minimum. It is typically set to a minor value, like 0.1 or 0.01, however the exact value may change depending on the issue [16].

Batch size: is a hyper-parameter that controls how many training examples are used in each gradient descent algorithm iteration. In other words, it specifies how many training instances are analyzed prior to an update to the weights. Small batch sizes can lead to quicker convergence and less memory use, but they can also make training noisier. Large batch sizes help lessen training process noise, but they can also use more memory and take longer to reach convergence [16].

Epoch: an epoch is defined as a single training iteration of all batches in both forward and backpropagation. The number of epochs required to train a neural network depends on factors such as the complexity of the task and the size of the dataset [16].

1.4.3 Improving neural networks

Improving neural networks is critical because it can lead to improved performance and more accurate predictions on a given task. There are several reasons why neural networks should be improved:

- ✓ **Better Accuracy:** Predicting an output with accuracy from an input is the main objective of neural networks. We can increase the predictions accuracy by enhancing the neural network's architecture and training techniques.
- ✓ **Faster Training:** Training neural networks can take a long period as they grow larger and more complicated. The network can be trained faster and more effectively by altering the training procedures.
- ✓ **Improved generalization:** A neural network with good design should perform well on data that it has not been trained on, or generalize well to unknown data.
- ✓ **Enhanced interpretability:** Due to their complexity, neural networks can be challenging to interpret, but enhancing interpretability can aid both academics and professionals in understanding how the model generates its predictions.

There are several techniques used to improve neural network such as:

1.4.3.1 Regularization technique

In order to avoid overfitting problem that we faced in the experimental part of this dissertation-regularization is a method used in neural networks. When a neural network learns the training data too thoroughly, it over fits and becomes unable to apply to new, untrained data. Regularization discourages the model from learning overly complex patterns in the data that might not be generalizable by adding a penalty word to the loss function. This helps to minimize overfitting.

In neural networks, regularization can be achieved in a number of ways, including:

Dropout: chooses a collection of nodes to be dropped out at random with a fixed probability p during each training iteration. Typically, the probability p is chosen between 0.2 and 0.5. Different nodes are dropped out for various training examples because the dropout process is carried out separately on each example. All nodes are used during testing, but their outputs are multiplied by probability p to guarantee that the layer's total output is the same as it was during training[2].

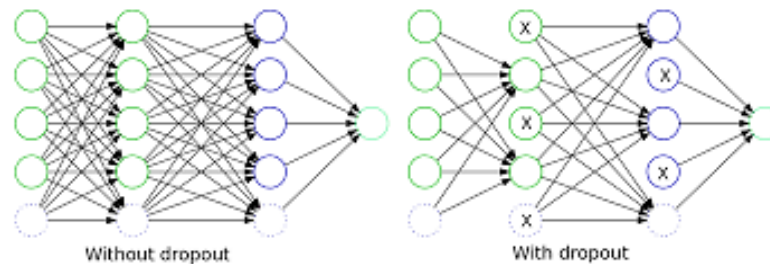


Figure 1.4: dropout technique.

Data augmentation: By generating new versions of the existing data through various transformations, data augmentation is a Deep Learning method used to artificially expand a training dataset. The model can learn more reliable and generalizable features this way, which will enhance its performance on untrained data. When applied to neural networks, a set of predefined transformations are usually applied to the input data before it is fed to the model. Random cropping, flipping, rotation, scaling, and color jittering are a few typical changes[2].

1.4.3.2 Optimization techniques

For neural networks to minimize the loss function and perform better, optimization methods are crucial. Here are a few of the popular Deep Learning optimization methods:

Gradient Descent: The most widely used optimization method in Deep Learning is gradient descent. In order to minimize the loss, it includes computing the gradient of the loss function with respect to the parameters of the model and updating them in the opposite direction of the gradient.

$$\theta^k = \theta^k - \gamma (d(\text{cost}))/d(\theta^k) \quad (1)$$

Where θ^k are the actual weights and γ is the learning rate.

Adam: (Adaptive Moment Estimation) is a popular optimization method for neural network training. The algorithm keeps track of both the square and the exponentially declining average of previous gradients. To take into consideration the initialization of the moving averages, bias correction terms are also included [5].

The Adam optimization algorithm utilizes a hybrid of two gradient descent techniques by combining the benefits of both momentum and RMSprop algorithms. This adjustable learning rate approach results in improved performance.

1.5 Deep learning Architecture

Supervised and unsupervised Deep Learning architectures are two types of architectures used in Deep Learning that differ in the way they learn from data.

1.5.1 Supervised Deep Learning

In supervised learning, the Deep Learning architecture is trained on labeled data, which means that the input data is already associated with the correct output or target. The architecture uses the labeled data to learn to predict the correct output for new, unseen input data. Convolutional neural network (CNN or Conv-Nets) is among the most commonly used architectures in supervised Deep Learning.

1.5.1.1 Convolutional Neural Network (ConvNet/CNN)

Convolutional Neural Network (ConvNet or CNN) is one of the most popular deep learning architectures that consists of multiple number of layers. The main concept of Conv-Nets is to obtain local features from input (usually an image) at higher layers and combine them into more complex features at the lower layers. However, because of its complex architecture, training such networks on large datasets can take several days, and it is computationally expensive. As a result, deep networks are frequently trained on GPUs [4].

ConvNet consists of a sequence of different types of layers to achieve different tasks. A typical Convolutional neural network consists of the following layers:

- Convolutional layer
- Activation Function Layer (ReLU)
- Pooling layer
- Fully-connected layer.

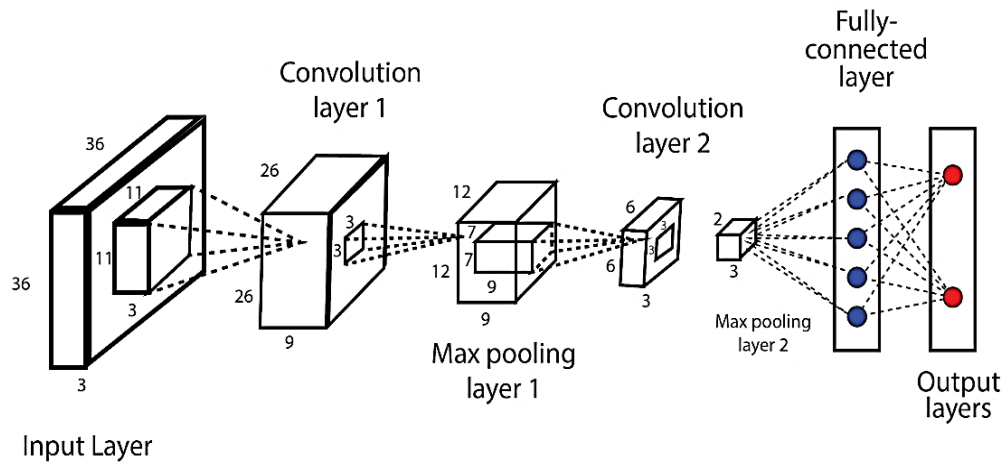


Figure 1.5: CNN Architecture.

These layers are stacked up to make a full ConvNet architecture. Convolutional and activation function layers are usually stacked together followed by an optional pooling layer. Fully connected layer makes up the last layer of the network and the output of the last fully connected layer produces the class scores of the input image. In addition to these main layers mentioned above, ConvNet may include other types of layers, such as batch normalization and dropout layers to improve the training time and address the overfitting issue respectively [4].

Convolution Layer: Convolution layer is the core building block of a Convolutional Neural Network which uses convolution operation (represented by $*$) in place of general matrix multiplication. Its parameters consist of a set of learnable filters also known as kernels. Figure 1.6 below shows how a filter is convolved with the input to get the feature MAP. Feature MAP is obtained after adding a bias term and then applying a non-linear function to the output of the convolution operation. The purpose of non-linearity function is to introduce non-linearity in the ConvNet model[4].

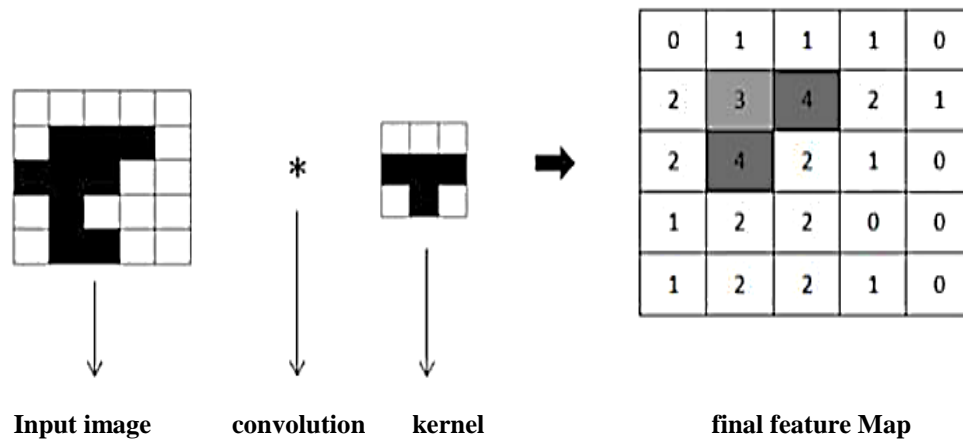


Figure 1.6: Example of Convolution Operation.

Hyper-parameters: Convolutional Neural Network architecture has many hyper-parameters that are used to control the behavior of the model. Some of these hyper-parameters control the size of the output while some are used to tune the running time and memory cost of the model. The four important hyper-parameters in the convolution layer of the ConvNet are given below:

- Filter Size:** Filters can be of any size greater than 2×2 and less than the size of the input but the conventional size varies from 11×11 to 3×3 . The size of a filter is independent of the size of input.
- Number of Filters:** This can be any reasonable number of filters. Alex Net used 96 filters of size 11×11 in first convolution layer. VGG Net used 96 filters of size 7×7 and another variant of VGG Net used 64 filters of size 11×11 in first convolution layer.
- Stride:** It is the number of pixels to move at a time to define the local receptive field for a filter. Stride of one means to move across and down a single pixel. The value of stride

should not be too small or too large. Too small stride will lead to heavily overlapping receptive fields and too large value will overlap less and the resulting output volume will have smaller dimensions spatially.

- d) **Zero Padding:** This hyper-parameter describes the number of pixels to pad the input image with zeros. Zero padding is used to control the spatial size of the output volume [4].

Activation Function Layer: Activation functions decide whether a neuron should be activated or not by calculating the weighted sum and further adding bias with it. They are differentiable operators to transform input signals to outputs, while most of them add non-linearity. Because activation functions are fundamental to Deep Learning, let's briefly survey some common activation functions. There are many of activation functions in use with artificial neural networks (ANN) and some of the commonly used activation functions are:

✓ **Logistic/Sigmoid Activation Function:**

The logistic activation function, also known as the sigmoid activation function, is a commonly used activation function in neural networks. It is represented by the mathematical equation:

$$\text{sigmoid}(x) = \frac{1}{1+e^{-x}} \quad (2)$$

The figure bellow show the sigmoid activation function graph:

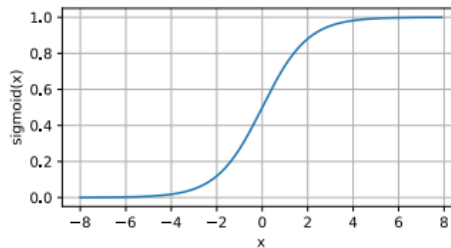


Figure 1.7: Graph of Sigmoid activation.

✓ **ReLU Activation Function:**

ReLU Function The most popular choice, due to both simplicity of implementation and its good performance on a variety of predictive tasks, is the rectified linear unit (ReLU) (Nair and Hinton, 2010). ReLU provides a very simple nonlinear transformation. Given an element x , the function is defined as the maximum of that element [5].

$$\text{ReLU}(x) = \max(x, 0) \quad (3)$$

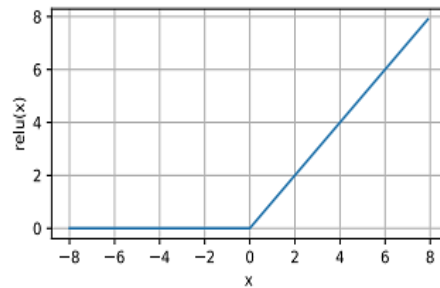


Figure 1.8: Rectified Linear Unit (Relu) activation function.

Pooling Layer: In ConvNet, the sequence of convolution layer and activation function layer is followed by an optional pooling or down sampling layer to reduce the spatial size of the input and thus reducing the number of parameters in the network. A pooling layer takes each feature MAP output from the convolutional layer and down samples it i.e. pooling layer summarizes a region of neurons in the convolution layer. There are few pooling techniques available and the most common pooling technique is Max-Pooling. Max-Pooling simply outputs the maximum value in the input region. The input region is a subset of input (usually 2×2). For example, if input region is of size 2×2 the Max-Pooling unit will output the maximum of the four values as shown in Figure 11. Other options for pooling layers are average pooling and L2-norm pooling. Pooling layer operation discards less significant data but preserves the detected features in a smaller representation. The intuitive reasoning behind pooling operation is that feature detection is more important than feature's exact location. This strategy works well for simple and basic problems but it has its own limitations and does not work well for some problem [4].

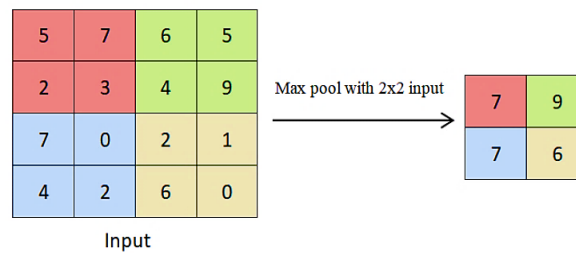


Figure 1.9: Max-pooling.

Fully Connected Layer: In ConvNets, The fully connected layers can be stacked to learn even more sophisticated combinations of features. The input of the FC layer comes from the last pooling or convolutional layer. This input is in the form of a vector, which is created from the feature MAPs after flattening. The output of the FC layer represents the final CNN output, as illustrated in Figure 1.12.

So, the fully connected layer collects the resulting data from the convolutional and pooling layer, analyzes the data from the layers independently and makes the final classification or localization decision. [4].

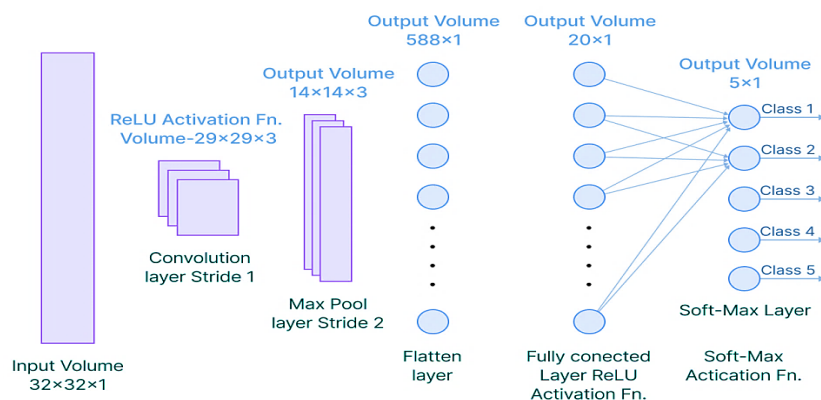


Figure 1.10: fully connected layer.

1.5.2 Unsupervised learning architecture

In unsupervised learning, the Deep Learning architecture is trained on unlabeled data, which means that the input data does not have any associated output or target. The architecture uses the structure and patterns in the input data to learn to represent the data in a more compact and informative way.

Examples of unsupervised learning architectures include auto-encoders and generative adversarial networks (GANs). These architectures are commonly used for tasks such as data compression, feature extraction, and data generation. The following are some of the most important reasons for the importance of Unsupervised Learning:

Unsupervised learning is beneficial for extracting useful insights from data. It is very similar to how humans learn to think through their own experiences, making it closer to true AI, it operates on unlabeled and uncategorized data, making it more important. In the real world, we do

not always have input data with corresponding output, so unsupervised learning is required to solve such cases.

Overall, the choice of a supervised or unsupervised Deep Learning architecture depends on the nature of the task, the type of data available, and the desired outcome [3].

1.6 Conclusion

As shown above in this chapter we presented a brief approach about Deep Learning, which will serve as a foundation for our exploration of object detection. Then, we dealt with the fundamental studies of Deep Learning including neural network concepts, how to improve a neural network. Also, we covered one of the most commonly used deep neural networks, CNN, and provided an overview of each CNN component and layer, from the convolutional layer to the final layer known as the fully connected layer (FC).

All these notions let us proceed towards the next chapters, where we will put them into practice by implementing these theoretical concepts, we will gain a better understanding of how deep learning architectures work and will be able to build our own deep neural network.

Chapter 2: Object detection

Abstract

This chapter is an overview of the background and main focus of our thesis. The categories algorithms of object detection are introduced, the YOLO algorithm and the evaluation metrics are discussed in detail.

2.1 Introduction

2.2 Object Detection

2.3 Object detection Algorithms

2.4 YOLO (You Only Look Once)

2.5 Main Evaluation metrics

2.6 Conclusion

2.1 Introduction

Object detection is a crucial but difficult vision task. It is an essential component of many applications, including object tracking, scene understanding, image auto-annotation, and image search. Numerous computer vision disciplines had already used it, including intelligent video surveillance (Arun Hampapur 2005), artificial intelligence, military guidance, safety detection and robot navigation, as well as medical and biological applications.

In this chapter of our project, we are interested on object detection. We will explore the primary algorithms of object detection, then we go deeply into YOLO object detection algorithm, and finally, talk about the evaluation metrics of an object detection model.

2.2 Object Detection

Object detection is the field of computer vision that deals with the localization and classification of objects contained in an image or video. Object localization, involves detecting one or more objects in an image and drawing a bounding box around them, Bounding boxes are used to indicate the location of objects within an image it represented by four integers: (x_1, y_1) for the upper-left corner and (x_2, y_2) for the lower-right corner[11] An example of a bounding box can be seen in **figure 2.1**.

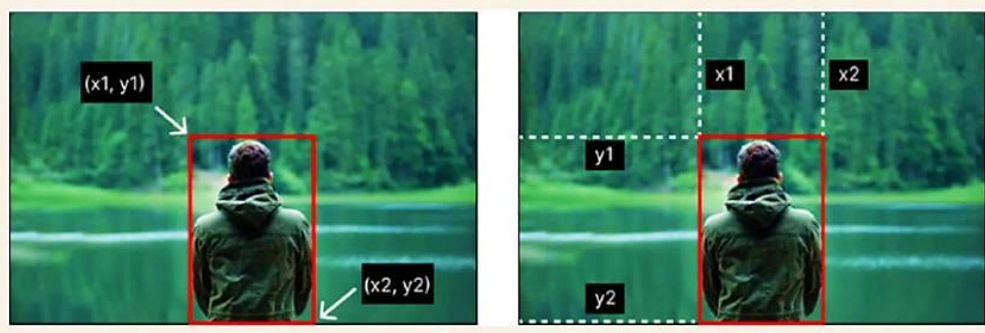


Figure 2.1: localizing object.

Localizing an object in a picture means predicting a bounding box around the object and can be expressed as a regression problem. On the other hand, Image classification includes identifying an object's class from an image.

Once an object is localized, the object detector model predicts the class or category that the object belongs to. This is done by considering the pixels within the region of the localized object. The model outputs probabilities for each class it is trained to identify. For example, if the model is

trained to detect men and women, it will provide probabilities for both categories [11]. An example of this is shown in **figure 2.2**



Figure 2.2: classifying objects.

These two tasks are combined in object detection, which involves locating and categorizing one or more objects in an image.

2.3 Object detection Algorithms

In general, there are two types of object detection algorithms: single stage and two stage [7]. Object detection in images and videos is performed using single-stage object detection algorithms, also referred to as one-stage detectors. In a single forward pass, these algorithms use a single neural network to forecast the bounding boxes and class probabilities for every object in the input image.

The method used by single-stage object detection algorithms is to divide the incoming image into a grid of cells and forecast the bounding boxes and class probabilities for each cell. The network generates a collection of bounding boxes, each of which includes a confidence value and a probability distribution over object classes. Then duplicate bounding boxes are removed, leaving only the most probable ones, using non-maximum suppression.

You Only Look Once (YOLO) and SSD (Single Shot Detector) are a couple of well-known single-stage object detection methods [7]. Even though these algorithms are actually faster than two-stage detectors, they may give up some accuracy in order to operate in real-time on low-power devices. Numerous applications, including robotics, surveillance, and automated driving, where real-time performance is essential, make extensive use of single-stage object detection algorithms.

Two-stage object detection algorithms, on the other hand, perform object detection in two stages. In the first stage, the algorithm generates region proposals, which are potential object

locations generated by a separate neural network called a region proposal network (RPN). The RPN receives an image as input and returns a set of candidate regions, each with a score indicating the likelihood of the region containing an object. In the second stage, the region proposals are refined and classified into different object categories. This is accomplished through the use of a separate network that accepts each region proposal as input and outputs the final bounding box coordinates and class probabilities for each object.

Fast R-CNN (Region-based Convolutional Neural Network) and Faster R-CNN (Faster Region-based Convolutional Neural Network) are the most widely used two-stage object detection algorithms [7]. These algorithms are usually more accurate than single-stage detectors, but they are slower and require more computation. They are used in high-accuracy applications such as medical imaging, satellite imagery, and object recognition in natural scenes.

On the whole, two-stage object detection algorithms are an effective strategy for object detection in images and videos, especially when high accuracy is required but real-time performance is not a priority.

The algorithm to use is determined by the specific application requirements and constraints. In **Figure 2.3** presented below, we can observe examples of Object detection Algorithms categorized into the two primary types: single-stage and two-stage algorithms for object detection.

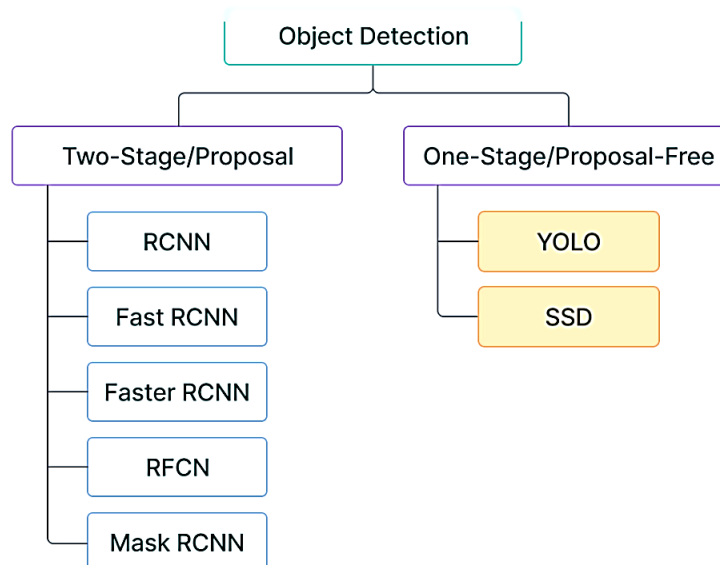


Figure 2.3: classification of object detection categories and algorithms.

2.3.1 One stage object detection Algorithm

2.3.1.1 Single Shot Detector (SSD)

A single deep neural network is used in the Single Shot Detector (SSD) method to find objects in images. In the SSD method, the output space of the bounding boxes is discretized into a collection of default boxes with various aspect ratios. The technique scales per feature map location after discretization [7]. To naturally handle objects of varying sizes, the Single Shot Detector network combines predictions from multiple feature maps with varied resolutions.

Advantages of SSD:

- SSD completely eliminates proposal generation and subsequent pixel or feature resampling stages and encapsulates all computation in a single network.
- Easy to train and straightforward to integrate into systems that require a detection component.

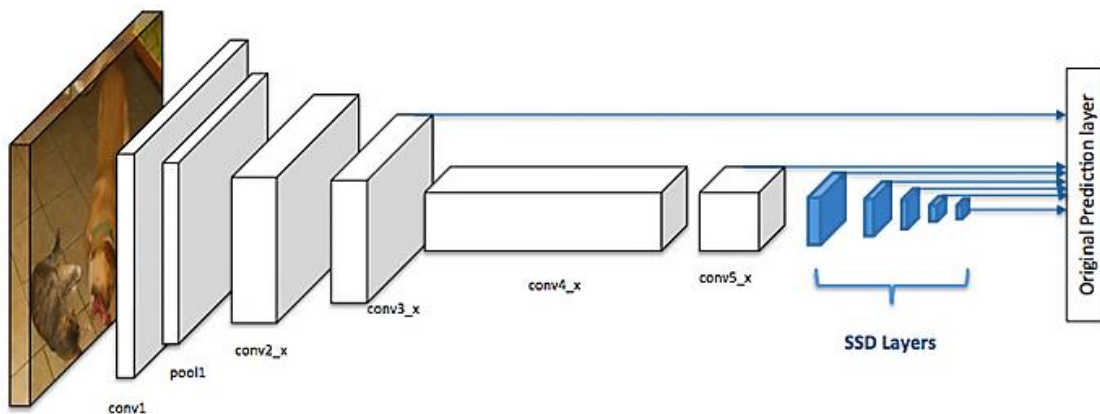


Figure 2.4: SSD single shot detector model.

SSD has competitive accuracy to methods that utilize an additional object proposal step, and it is much faster while providing a unified framework for both training and inference

2.3.2 Two Stage Object Detection Algorithm

2.3.2.1 Fast R-CNN

Fast Region-Based Convolutional Network Method, also known as Fast R-CNN, is an object detection training technique written in Python. This algorithm primarily improves the speed and precision of R-CNN and SPPNet while addressing their flaws [7].

The benefits of fast R-CNN:

- Higher detection quality (mAP) than R-CNN, SPPNet
- Training is single-stage, using a multi-task loss
- Training can update all network layers
- No disk storage is required for feature caching

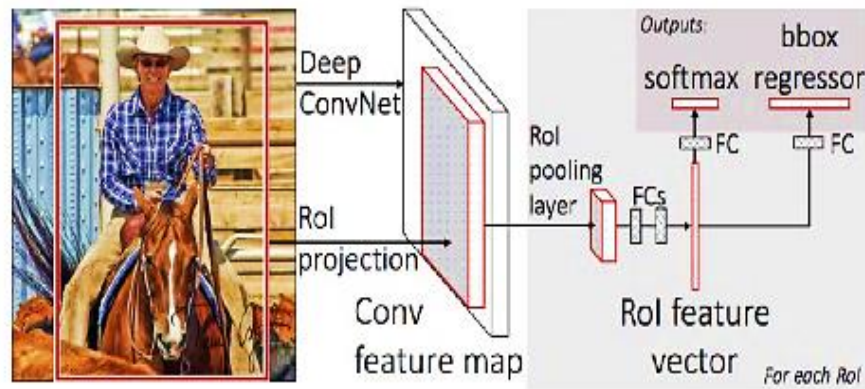


Figure 2.5: Fast R-CNN.

2.3.2.2 Faster R-CNN

R-CNN and Faster R-CNN are both object recognition algorithms. This method makes use of the Region Proposal Network (RPN), which is more efficient than R-CNN and Fast R-CNN and shares full-image convolutional features with the detection network. Fast R-CNN detects objects using high-quality region proposals produced by a Region Proposal Network, which is essentially

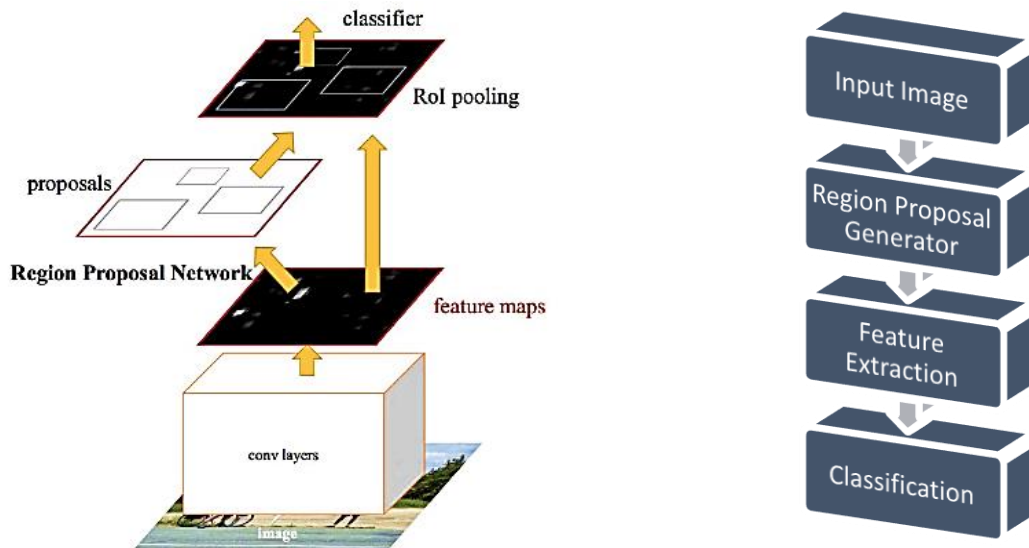


Figure 2.6: Faster R-CNN.

a fully convolutional network that concurrently predicts the object bounds and objectness scores at each location of the object [7].

We will now move on to the most useful algorithm, YOLO (You Only Look Once), which will be the main focus of our implementation. Our implementation will cover two variations of the algorithm, specifically YOLOv5 and YOLOv8.

2.4 YOLO (You Only Look Once)

You Only Look Once (YOLO) is an algorithm that uses convolutional neural networks for object detection, it has gained significant popularity among researchers worldwide as one of the most prevalent techniques for object detection. It is a very good choice when we need real-time detection, without loss of too much accuracy.

In our thesis, we will be implementing YOLOv5 and YOLOv8. YOLOv5, introduced by Glenn Jocher on June 9, 2020, followed the recent release of YOLOv4 on April 23, 2020. Both YOLOv5 and YOLOv8 are efficient object detection models that demonstrate exceptional performance in real-time image processing. However, YOLOv8 surpasses YOLOv5 in terms of speed, making it a preferable option for applications where real-time object detection is crucial. YOLOv8 was launched on January 10th, 2023.

2.4.1 Overview of the YOLOv5 Architecture

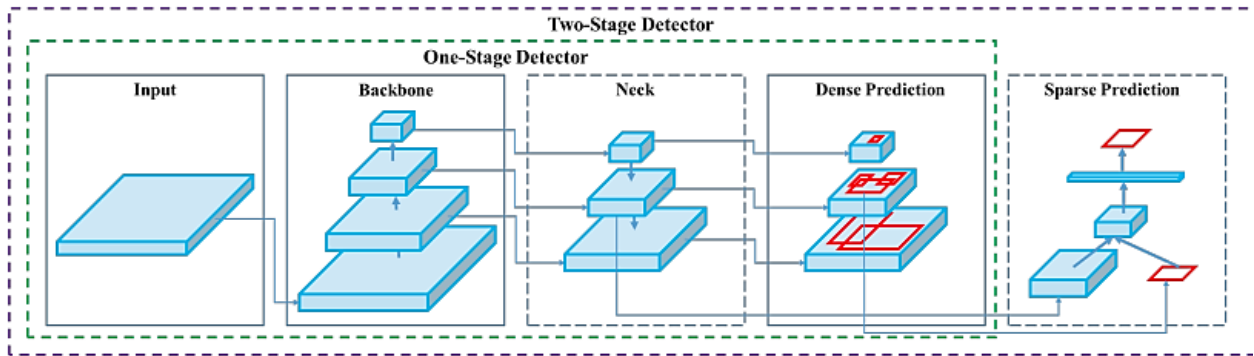
A detection model contains a Backbone, Neck and Head module. The backbone module exploits the essential features of different resolutions, and the neck module fuses the features of different resolutions. Finally, multiple head modules perform the detection of objects in different resolutions [14].

Backbone: A convolutional neural network that extracts features from the given image. Tiny YOLO has only 9 convolutional layers, so it's less accurate but faster and better suited for mobile and embedded projects. On the other hand, Darknet53, the backbone utilized in YOLOv3, consists of 53 convolutional layers, making it more accurate but slower in terms of computation. In YOLOv4, the backbones available include VGG, ResNet, Spine-Net, Efficient-Net, Res-NeXt, or Darknet53, providing flexibility in choosing the desired trade-off between accuracy and speed [10].

Neck: The feature network receives inputs multiple levels of features from the backbone and outputs a list of fused features that represent salient characteristics of the image [10].

Head: The final class/box network employs the fused features to make predictions regarding the object's class and location. In simpler terms, locate bounding boxes and classify what's inside each box [10].

The figure below represents the architecture of YOLO:



Input: { Image, Patches, Image Pyramid }

Backbones: { VGG16 [68], ResNet-50 [26], Darknet53 [81] }

Neck: { FPN[44], PANET[26], [47], BI-FPN [77] }

Heads: Dense Prediction { (one-stage): RPN [64], SSD [50], YOLO [61,62,63], RetinaNet[45], FCOS [78] }

Sparse Prediction { (two-stage): Faster R-CNN [64], R-FCN [9] }

Figure 2.7: YOLO Architecture.

The YOLOv5 contains 5 models in total. Starting from YOLOv5 nano (smallest and fastest) to YOLOv5 extra-large (the largest model)[15]. The following is a short description of each of these:

YOLOv5n: It is a newly introduced Nano model, which is the smallest in the family and meant for the edge, IoT devices, and with OpenCV DNN support as well. It is less than 2.5 MB in INT8 format and around 4 MB in FP32 format. It is ideal for mobile solutions [15].

YOLOv5s: It is the small model in the family with around 7.2 million parameters and is ideal for running inference on the CPU [15].

YOLOv5m: This is a medium-sized model with 21.2 million parameters. It is perhaps the best-suited model for many datasets and training as it provides a good balance between speed and accuracy [15].

YOLOv5l: It is the large model of the YOLOv5 family with 46.5 million parameters. It is ideal for datasets where we need to detect smaller objects [15].

YOLOv5x: It is the largest among the five models and has the highest MAP among the 5 as well. Although it is slower compared to the others and has 86.7 million parameters [15].

2.5 Main evaluation metrics

Among different annotated datasets used by object detection challenges and the scientific community, the most common metric used to measure the accuracy of the detections is the AP. Before examining the variations of the AP, we should review some concepts that are shared among them. The most basic are the ones defined below:

- True positive (TP): A correct detection of a ground-truth bounding box.
- False positive (FP): An incorrect detection of a nonexistent object or a misplaced detection of an existing object.
- False negative (FN): An undetected ground-truth bounding box; It is important to note that, in the object detection context, a true negative (TN) result does not apply, as there are infinite number of bounding boxes that should not be detected within any given image. The above definitions require the establishment of what a “correct detection” and an “incorrect detection” are. A common way to do so is using the intersection over union (IOU). It is a measurement based on the Jaccard Index, a coefficient of similarity for two sets of data [12]. In the object detection scope, the IOU measures the overlapping area between the predicted bounding box B_p and the ground-truth bounding box B_{gt} divided by the area of union between them, that is

$$J(B_p, B_{gt}) = IOU = \frac{\text{area}(B_p \cap B_{gt})}{\text{area}(B_p \cup B_{gt})} \quad (4)$$

as illustrated in **Figure 2.5**

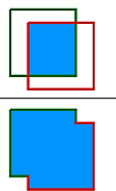
$$IOU = \frac{\text{area of overlap}}{\text{area of union}} = \frac{\text{area of overlap}}{\text{area of union}}$$


Figure 2.8: Intersection over union.

Mean Average Precision (MAP) is a performance indicator that is frequently used to rate object detection systems. It is an indicator of how effectively the algorithm can locate and identify things in an image. Precision and recall values are obtained for each class of object in the dataset in order to determine the MAP for object detection. Recall is the percentage of true positive detections out of all ground truth objects, whereas precision is the percentage of true positive detections out of all

detections. Following that, the MAP is determined by averaging the AP (Average Precision) scores obtained from all classes. The area under the precision-recall curve is used to calculate each class's AP score. In conclusion, MAP is a measure of an object identification algorithm's performance across all classes, accounting for both recall and precision. Better performance is indicated by a greater MAP [12].

$$\text{MAP} = \frac{1}{N} \sum_{i=1}^N \text{AP}_i \quad (5)$$

Where AP_i being the AP in the i_{th} class and N is the total number of classes being evaluated.

2.6 Conclusion

This chapter provided an overview of object detection, a fundamental task in computer vision. We explored two main approaches: one-stage and two-stage object detection algorithms. The one-stage algorithms, such as SSD and YOLO with particular emphasis on the widely adapted YOLO algorithm and the two-stage algorithms, exemplified by Fast R-CNN and Faster R-CNN. Additionally, we highlight the key evaluation metrics employed to assess the performance of object detection models.

In the subsequent chapter, we will shift our focus towards the implementation of the highly regarded algorithm known as YOLO. Specifically, we will explore the latest versions, namely YOLOv5 and YOLOv8.

Chapter 3: Models implementation and results of experiments

Abstract

This chapter focuses on the implementation of various object detection models, namely the CNN model, transfer learning, YOLOv5, and YOLOv8. It covers the architecture and training process of these models, highlighting their strengths and limitations throughout a comparative analysis of the results.

3.1 Introduction

3.2 Software and libraries used in the implementation

3.3 Data

3.4 Convolutional neural network implementation

3.5 Transfer learning implementation

3.6 Comparative analysis between the result of CNN and ResNet architecture

3.7 YOLO Implementation

3.8 Conclusion

3.1 Introduction

After gathering all the relevant information and understanding how object detection works, this chapter will focus on constructing our Convolutional Neural Network (CNN) to detect objects across diverse datasets. We will evaluate its performance using various metrics. Additionally, we will explore transfer learning to improve the obtained results. Also, by retraining YOLOv5n and YOLOv8n on our custom datasets, we aim to determine whether employing algorithms with less parameters and lower accuracy is truly necessary for our specific datasets, or if it would be more advantageous to utilize our proposed architecture. This comparison holds significant importance when considering implementation in embedded systems.

3.2 Software and libraries used in the implementation

3.2.1 Software

In our project we use Google Colab as software tool as it offers the ability of using free GPU, here is a brief overview of Google Colab.

Google Colab (short for Collaboratory) is a Google cloud-based service that allows users to execute Python scripts and conduct machine learning tasks on CPUs, GPUs, and TPUs. It employs the following techniques:

- **Virtual machines:** They are used by Google Colab to execute scripts and carry out calculations on the cloud. These virtual machines come preconfigured with a number of machine learning libraries and frameworks, including TensorFlow and PyTorch.
- **Jupyter Notebooks:** Google Colab is built on Jupyter notebooks, which offer an interactive environment for executing code, viewing data, and creating documentation. Users have the ability to make, modify, and share notebooks.
- **GPU and TPU Acceleration:** Tensor Processing Units (TPUs) and GPU support are available in Google Colab to enhance the speed of machine learning operations. As a result, it becomes possible to efficiently handle larger datasets and train models more rapidly.
- **Integration with Google Drive:** Google Colab is connected with Google Drive, allowing users to save and retrieve data and notes from any location. Users may also access their data by mounting their Google Drive in Colab.

- **Collaboration Features:** Google Colab's collaboration features allow users to collaborate with others by sharing their notebooks and code. Users may also provide comments on code and notebooks and collaborate in real time.
- **Code Snippets:** Google Colab has a large library of code snippets and examples for easily implementing basic machine learning tasks and approaches.

Overall, Google Colab is a robust and adaptable platform for machine learning and data science, with a variety of features and tools to support a wide range of workflows and projects.

3.2.2 Programming language

Python was the programming language of choice for our project, particularly in Google Colab. Python is widely used in Google Colab and is recognized as a popular high-level programming language. It is extensively employed in various domains, including web development, data analysis, machine learning, artificial intelligence, and more.

3.2.3 Frameworks

- **TensorFlow:** TensorFlow is an open-source Machine Learning framework created by the Google Brain team and was made public in 2015. It enables developers to create, train, and deploy Machine Learning models on a variety of platforms. TensorFlow provides a number of tools, libraries, and resources for developing and testing Machine Learning models, and it covers a wide range of use cases, including image and audio recognition, natural language processing, and time series analysis.
- **Keras:** is a Python-based open-source high-level neural network API that can operate on top of TensorFlow, Microsoft Cognitive Toolkit (CNTK), Theano, or PlaidML. It was created by François Chollet, who now works at Google, and was distributed under the permissive MIT license.
- **PyTorch:** is a Python machine learning package that is open source. It was created largely by Facebook's AI Research (FAIR) team to provide a flexible and fast framework for creating Deep Learning models. PyTorch supports dynamic computational graphs and automated differentiation, allowing developers to quickly create and train neural networks.

3.2.4 Libraries

- **OpenCV:** OpenCV (Open Source Computer Vision Library) is a free and open source software library for computer vision and Machine Learning.
- **NumPy:** NumPy (Numerical Python) is a Python library that supports massive multi-Dimensional arrays and matrices, as well as a vast number of high-level mathematical functions for working with these arrays.
- **Matplotlib:** Matplotlib is a Python data visualization package that may be used to generate static, animated, and interactive displays.
- **Pandas:** is an Open-Source data manipulation and analysis package written in Python. It includes tools for data cleansing, exploration, and transformation, as well as data structures for effectively storing and processing massive datasets.

3.3 Data

In order to train and confirm the performance of our CNN mode, we have used a variety of datasets. For the purpose of comparison, we retrained YOLOv5 and YOLOv8 using the exact same datasets. We have the flexibility to choose from six distinct categories including: malaria, drone, lung cancer, aircraft, breast cancer, brain tumor.

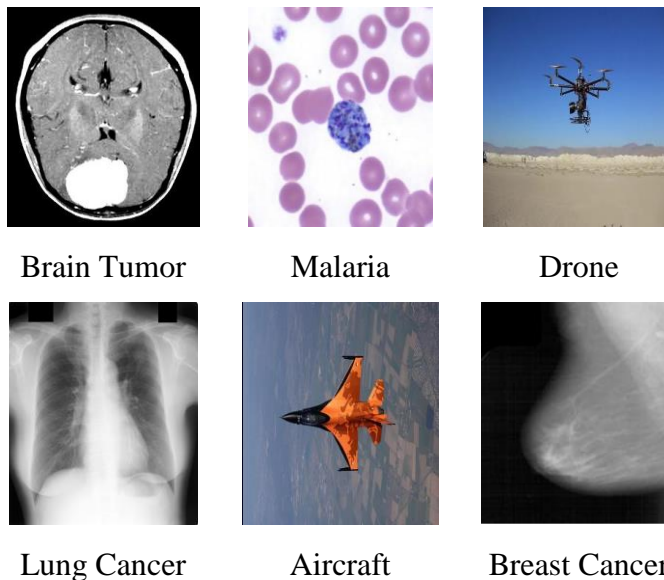


Figure 3.1: Dataset.

The dataset does not have bounding boxes around the objects, we need to add them ourselves. This is often one of the hardest and most costly parts of a Deep Learning project. It is a good idea to spend time looking for the right tools.

Our supervisor provided us with a Brain Tumor dataset, and we utilized the LabelImg tool for the annotation of the images. However, the tool generated the annotations in XML format, which necessitated the integration of a code snippet to convert them into CSV format.

Both Malaria and Aircraft Dataset, were obtained from Roboflow. The images of Malaria had black frames, to address this issue, we decided to crop out the black frames from the images. After collecting the dataset, the next step was the annotation, which entailed labeling the objects of interest in each image by drawing bounding boxes around them. We opted to utilize Roboflow as an annotation tool for efficient and accurate annotation tasks.

The third dataset, was "Drone" which was obtained from Roboflow along with their accompanying CSV file. The dataset was utilized in its original form without any modifications.

The Lung Cancer dataset was sourced from Kaggle, and subsequently, Roboflow was employed for the annotation process, it is important to note that the dataset is not extensive, comprising only 100 images.

In the table below we represent all the data used in this thesis, it shows the number of samples in each dataset that were used for both training and testing.

Dataset	Number of images		Total
	Train	Test	
Lung Cancer	80	20	100
Breast Cancer	100	66	166
Brain tumor	120	92	212
Malaria	150	62	213
Aircraft	200	80	280
Drone	400	100	500

Table3-1: The number of the dataset used.



Figure 3.2: Datasets Annotated.

After the labeling process, we got the information about the object desired which is typically represented by four values: the coordinates of the top-left corner ($Xmin, Ymin$) and the coordinates of the bottom-right corner ($Xmax, Ymax$) and save them in a CSV (comma-separated values) file.

This is a sample of the CSV file structure:

	A	B	C	D	E	F	G	H
1	filename	width	height	class	xmin	ymin	xmax	ymax
2	JPCLN003_pr	300	300	cancer	117	55	220	184
3	JPCLN082_pr	300	300	cancer	128	59	232	245
4	JPCLN050_pr	300	300	cancer	133	64	227	208
5	JPCLN017_pr	300	300	cancer	128	64	241	217
6	JPCLN105_pr	300	300	cancer	124	78	218	223
7	JPCLN021_pr	300	300	cancer	137	68	219	229
8	JPCLN063_pr	300	300	cancer	109	81	209	254
9	JPCLN130_pr	300	300	cancer	129	83	239	237

Figure 3.3: CSV file.

3.4 Convolution Neural Network Implementation

To achieve our goal “object detection”: first, we tried to build a convolution neural network model, we decided to use Google Colab that allows us to use its free GPU.

We experimented with multiple CNN architectures and extensively modified the number of layers and filters within each architecture. However, our training process yielded unsatisfactory results. After numerous attempts, we eventually settled on the following architecture as our final choice: The architecture consists of four Convolutional layers, with Max-pooling applied between

each layer. This is followed by a Fully Connected layer comprising four dense layers. The specific size and number of filters can be seen in Figure 3.4.

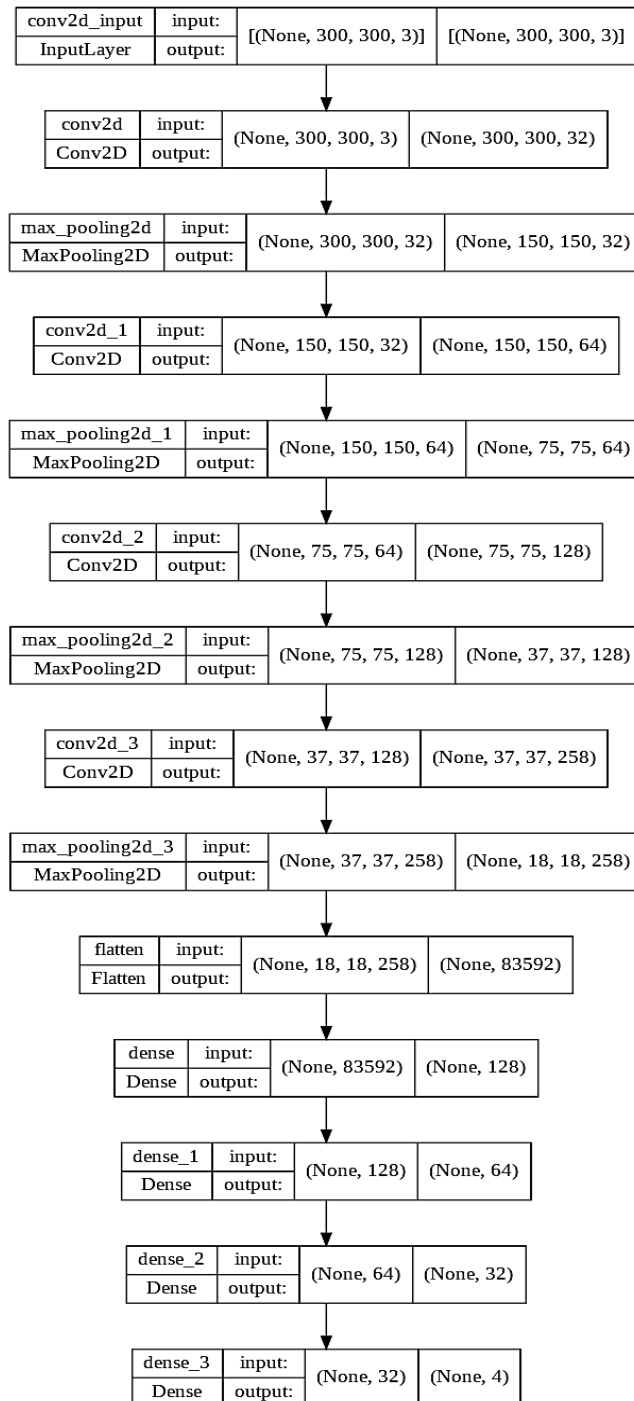


Figure 3.4: CNN Architecture.

3.4.1 Training CNN Model

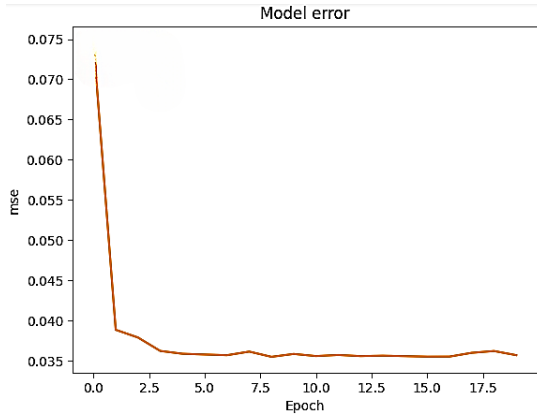
Once the CNN architecture was selected, we proceeded to train our model by fine-tuning the hyper-parameters, loss function and optimizer. These hyper-parameters include the learning rate, number of epochs, batch size, beta1, beta2. By carefully adjusting these parameters, we aimed to optimize the performance and convergence of our model during the training process. During the training process, we discovered that the size of objects in images can significantly affect their detection. For instance, in the Breast Cancer dataset, the cancerous regions were often in their initial stages, making them quite small. As a result, our model encountered difficulties in accurately localizing these tiny objects. Conversely, in the Aircraft dataset, the objects were significantly larger, occupying the majority of the frame. To address this issue, we carefully selected parameters that yielded satisfactory results on three different datasets. The summary of these datasets and the corresponding parameter choices are presented in the following tables

3.4.1.1 Result using Malaria Dataset

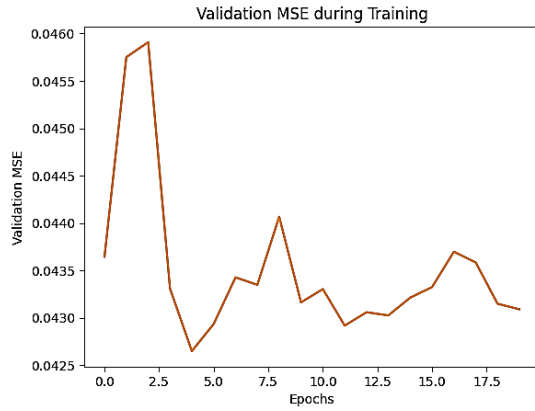
Table 3-2 presents the IOU outcomes acquired from 6 test images Dataset following the training of the network for 20, 60 and 100 epochs, respectively. The training employed the Adam optimizer with the subsequent parameter settings: learning rate=0.001, beta_1=0.9, beta_2=0.999.

Images	Number of epochs	IOU
1	20	75
	60	89
	100	58
2	20	74
	60	84
	100	37
3	20	62
	60	79
	100	22
4	20	57
	60	75
	100	16
5	20	55
	60	69
	100	08
6	20	50
	60	66
	100	03

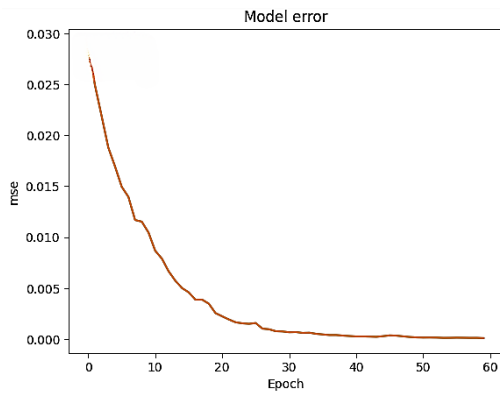
Table3-2: IOU results for Malaria datasets, 6 samples.



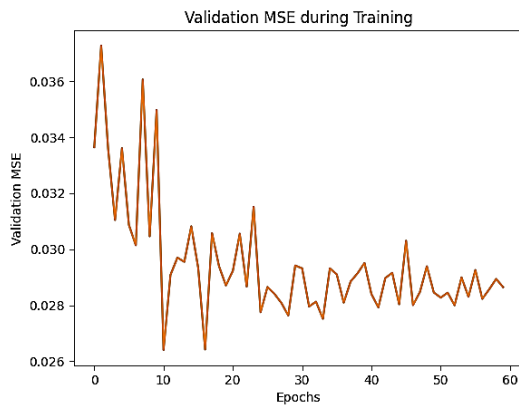
Training curve for 20 epochs



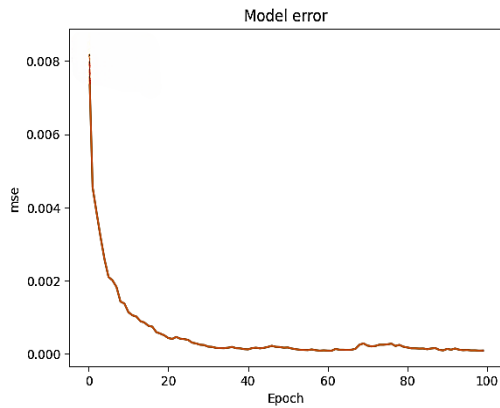
Validation curve for 20 epochs



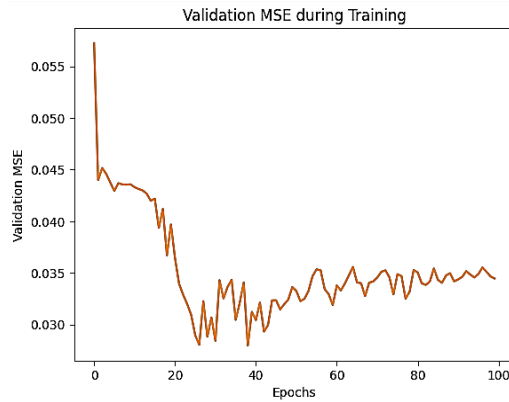
Training curve for 60 epochs



Validation curve for 60 epochs



Training curve for 100 epochs



Validation curve for 100 epochs

Figure 3.5: Malaria performance metrics .

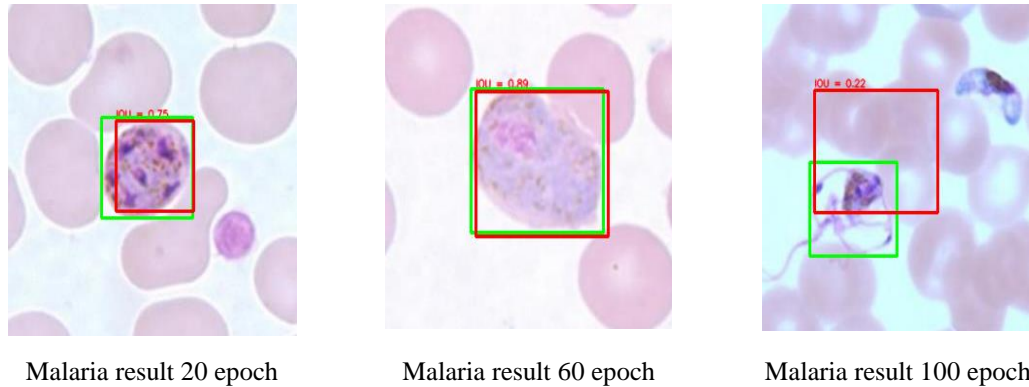


Figure 3.6: Malaria training Results.

When working with the Malaria Dataset, we observed that the choice of the number of epochs significantly impacted the performance of our CNN model. After experimentation, we determined that training the model for 60 epochs was most suitable for this dataset.

Considering the size of the dataset, we found that training for 100 epochs led to poor results. To validate this finding, we analyzed the performance curves, specifically the loss and validation curves. These curves provide valuable insights into the model's learning progress. With 60 epochs, we observed a gradual decrease in both the loss and validation curves, indicating that the model was effectively learning and generalizing from the data. On the other hand, when training for 100 epochs, the curves exhibited undesirable behaviors. The loss still decreased, while the validation curve showed signs of overfitting or a lack of generalization.

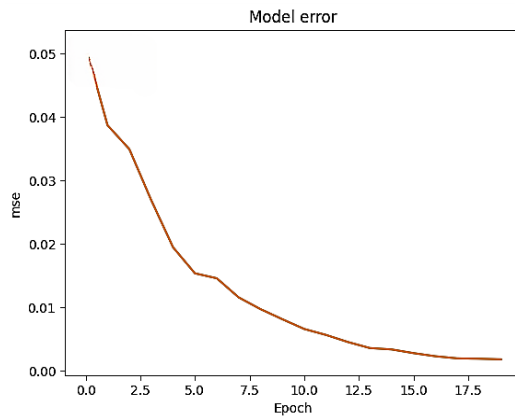
Based on these observations, we concluded that training for 60 epochs strikes a balance between model performance and dataset size for the Malaria Dataset. It ensures adequate learning while avoiding overfitting or poor generalization.

3.4.1.2 Result using Drone Dataset

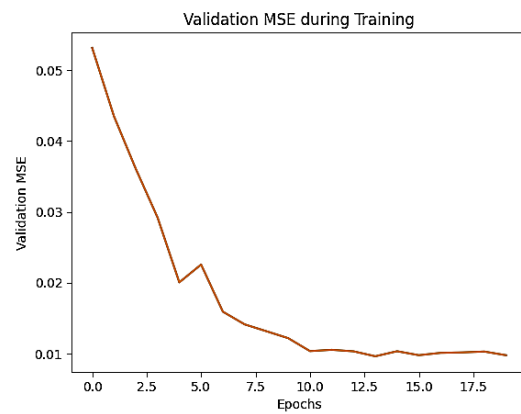
Table 3-3 and **figure 3.7** below presents the results of our IOU and performance metrics during the training and validation of our CNN model, respectively using the same hyper-parameters as the previous dataset.

Images	Number of epochs	IOU
1	20	27
	60	76
	100	94
2	20	21
	60	62
	100	83
3	20	18
	60	52
	100	80
4	20	15
	60	43
	100	79
5	20	10
	60	37
	100	68
6	20	09
	60	25
	100	67

Table 3-3: IOU results for Drone datasets, 6 samples.



Training curve for 20 epochs



Validation curve for 20 epochs

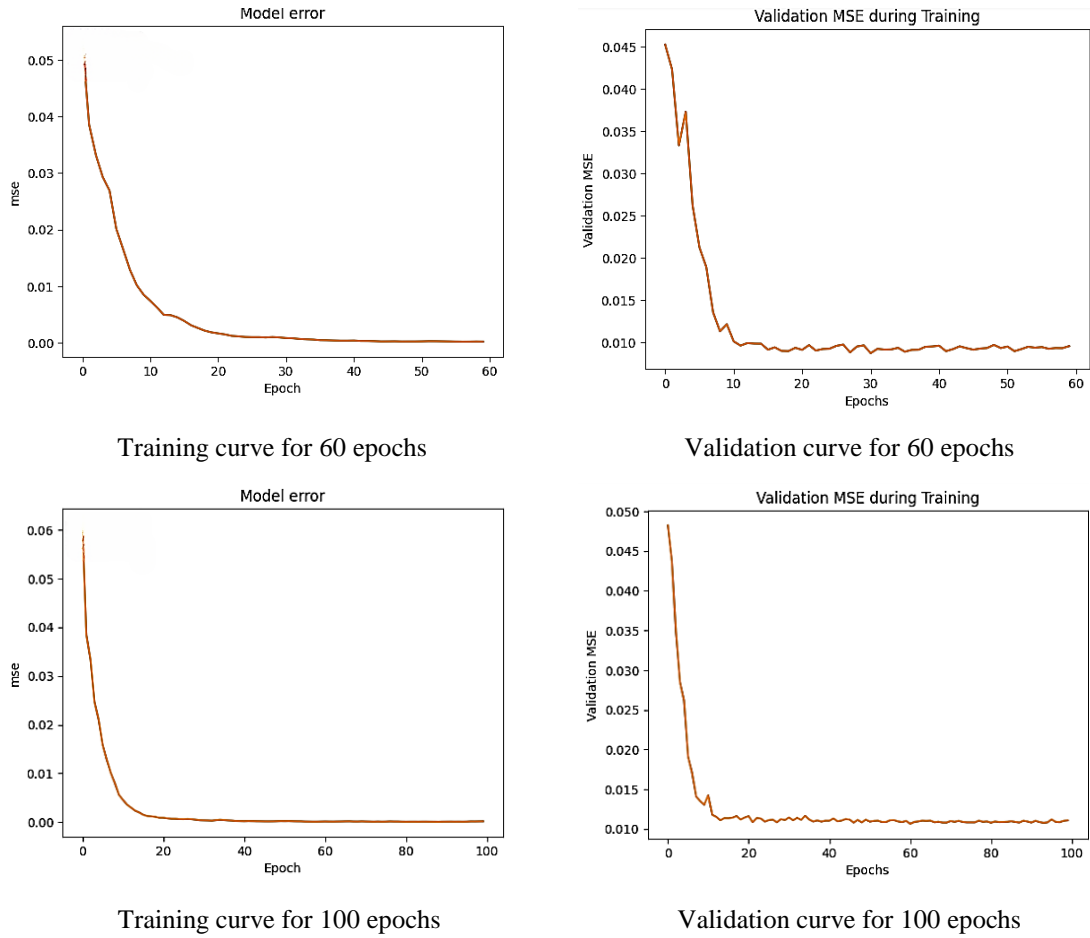
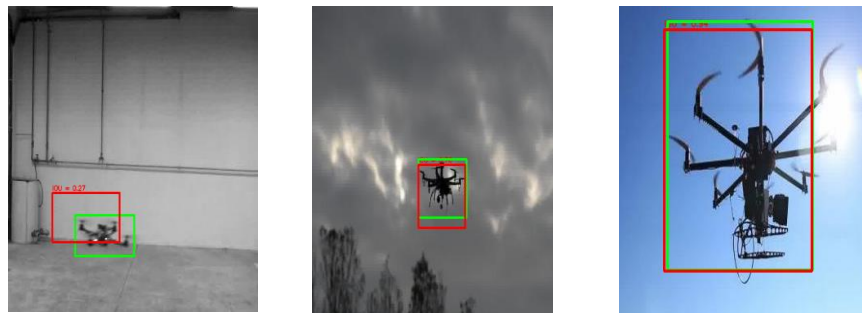


Figure 3.7: Drone performance metrics.

The **figure 3.8** below shows the 3 samples of the outcome using 20, 60 and 100 epochs



Drone result 20 epochs Drone result 60 epochs Drone result 100 epochs

Figure 3.8: Drone training Results.

In this particular case, the dataset consists of 500 images. Through experimentation, we found that training the model using 100 epochs yielded superior outcomes compared to using 20

or 60 epochs. Our CNN model exhibited promising results during both the training and validation stages, as evidenced by the performance curves. Increasing the number of epochs beyond 100 led to a deterioration in the model's performance, indicating overfitting. Therefore, we decided to limit the training to 100 epochs to ensure optimal results.

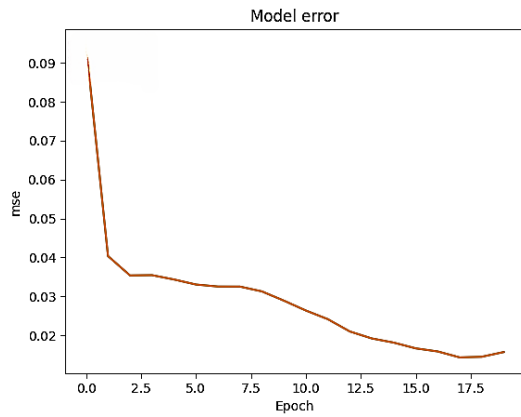
It is worth noting that by using more epochs, the model's performance continued to improve initially. However, beyond a certain point, the model started to exhibit signs of divergence, indicating a loss of generalization capability. This divergence can be attributed to overfitting, where the model becomes too specific to the training data and fails to generalize well on unseen data.

3.4.1.3 Result using Lung Cancer Dataset

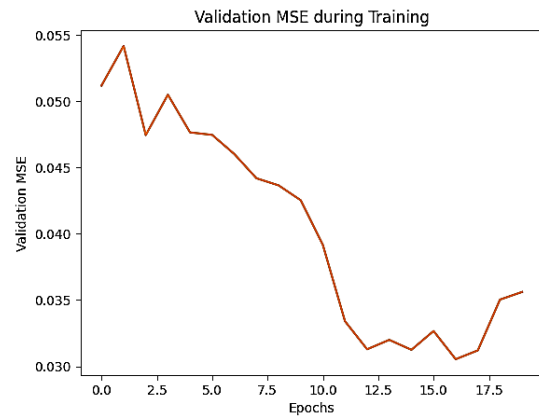
In a similar manner as we did with the previous dataset, we applied the same process to analyze the Lung Cancer dataset. The final results are summarized in **Table 3-4**, while Figure 3.9 provides a visual representation of these results.

Images	Number of epochs	IOU
1	20	92
	60	94
	100	91
2	20	85
	60	89
	100	90
3	20	80
	60	84
	100	81
4	20	79
	60	79
	100	84
5	20	73
	60	75
	100	82
6	20	69
	60	74
	100	80

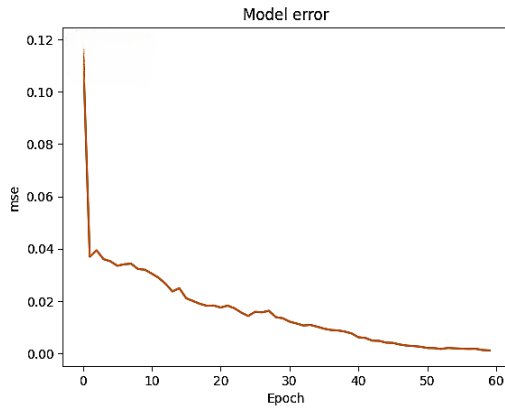
Table3-4: IOU results for Lung Cancer datasets, 6 samples.



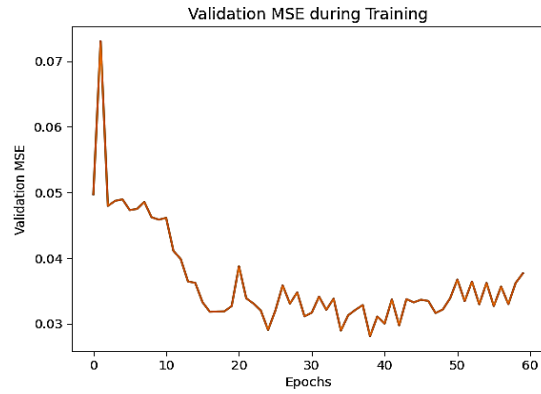
Training curve for 20 epochs



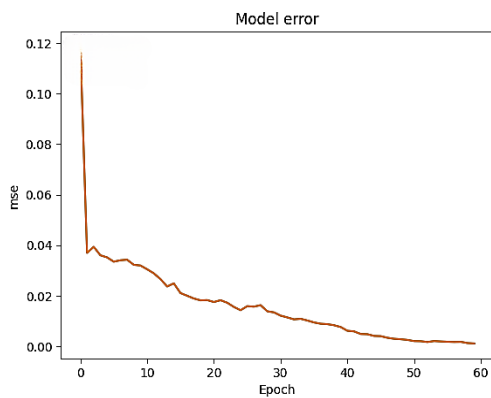
Validation curve for 20 epochs



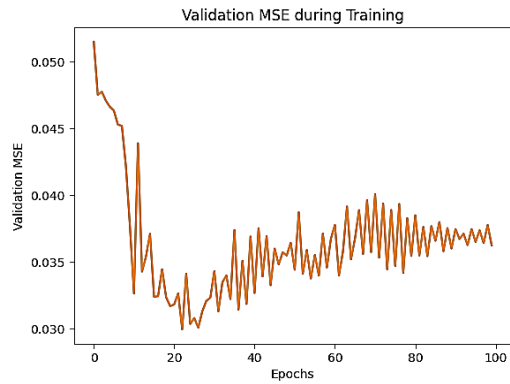
Training curve for 60 epochs



Validation curve for 60 epochs



Training curve for 100 epochs



Validation curve for 100 epochs

Figure 3.9: Lung Cancer performance metrics.

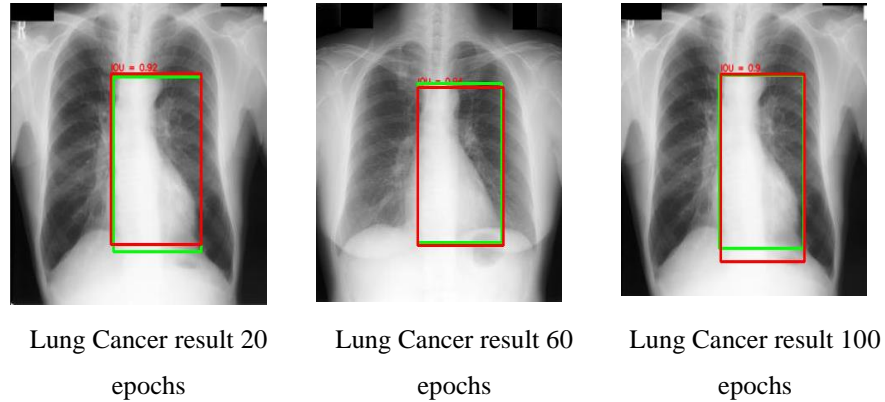


Figure 3.10: Lung Cancer training results.

During our experimentation, we noticed that our model achieved exceptional performance and accuracy compared to previous datasets, despite the relatively small size of the dataset. One of the primary reasons for this success lies in the inherent similarity of the objects depicted within the images. The high degree of resemblance among the objects facilitated the learning process for our model, enabling it to effectively recognize and detect these objects with greater ease and precision. Additionally, we conducted various adjustments to the number of epochs during training, further optimizing the model's performance. These findings highlight the advantages of employing a dataset where the objects exhibit a strong visual resemblance, leading to improved learning outcomes and enhanced detection capabilities.

3.5 Transfer Learning Implementation

To improve the performance of our model on the object detection target we used the transfer learning that is a pre-trained model on large scale dataset and we chose the ResNet.

First, we load the ResNet 50 model with pre-trained weights from the “ImageNet” dataset and frozen them, then we build our new model where:

- The ResNet 50 model was added as the first layer
- Global Average Pooling2D layer added as second layer, which reduces the spatial dimensions of the features and calculates the average value for each feature MAP
- Two dences layers are added with 64 and 32 units respectively, both using ReLu activation function.
- Finally, a dense layer with 4 units is added, representing the output layer.

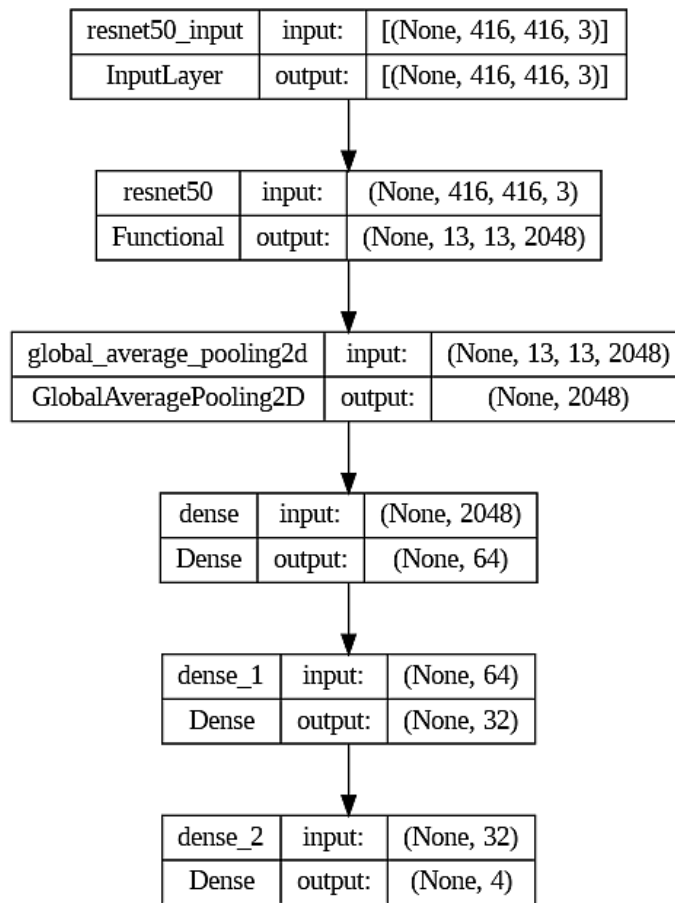


Figure 3.11: ResNet50 Architecture.

3.5.1 Result interpretation ResNet model

After implementing ResNet, we noticed significant improvements in the performance metrics of our model. The loss and validation Mean Squared Error (MSE) decreased across all types of data, indicating enhanced performance. Furthermore, we were able to effectively reduce overfitting, resulting in increased values of Intersection over Union (IOU). The figure provided below illustrates the performance of our model following training with ResNet.

3.5.1.1 Result using Malaria Dataset

The following **Table 3-5** represents the IOU results for Malaria datasets using ResNet50 we took 6 test samples, The training employed the Adam optimizer with the subsequent parameter settings: learning rate=0.001, beta_1=0.9, beta_2=0.999. We trained our model with deferent number of epochs 20, 60 and 100.

Images	Number of epochs	IOU
1	20	74
	60	70
	100	66
2	20	69
	60	69
	100	65
3	20	66
	60	64
	100	64
4	20	55
	60	61
	100	62
5	20	49
	60	58
	100	60
6	20	33
	60	53
	100	58

Table 3-5 : IOU results for Malaria datasets using ResNet50, 6 samples.

The **Figure 3.12** below shows the performance metrics during the training and validation of our ResNet Model.

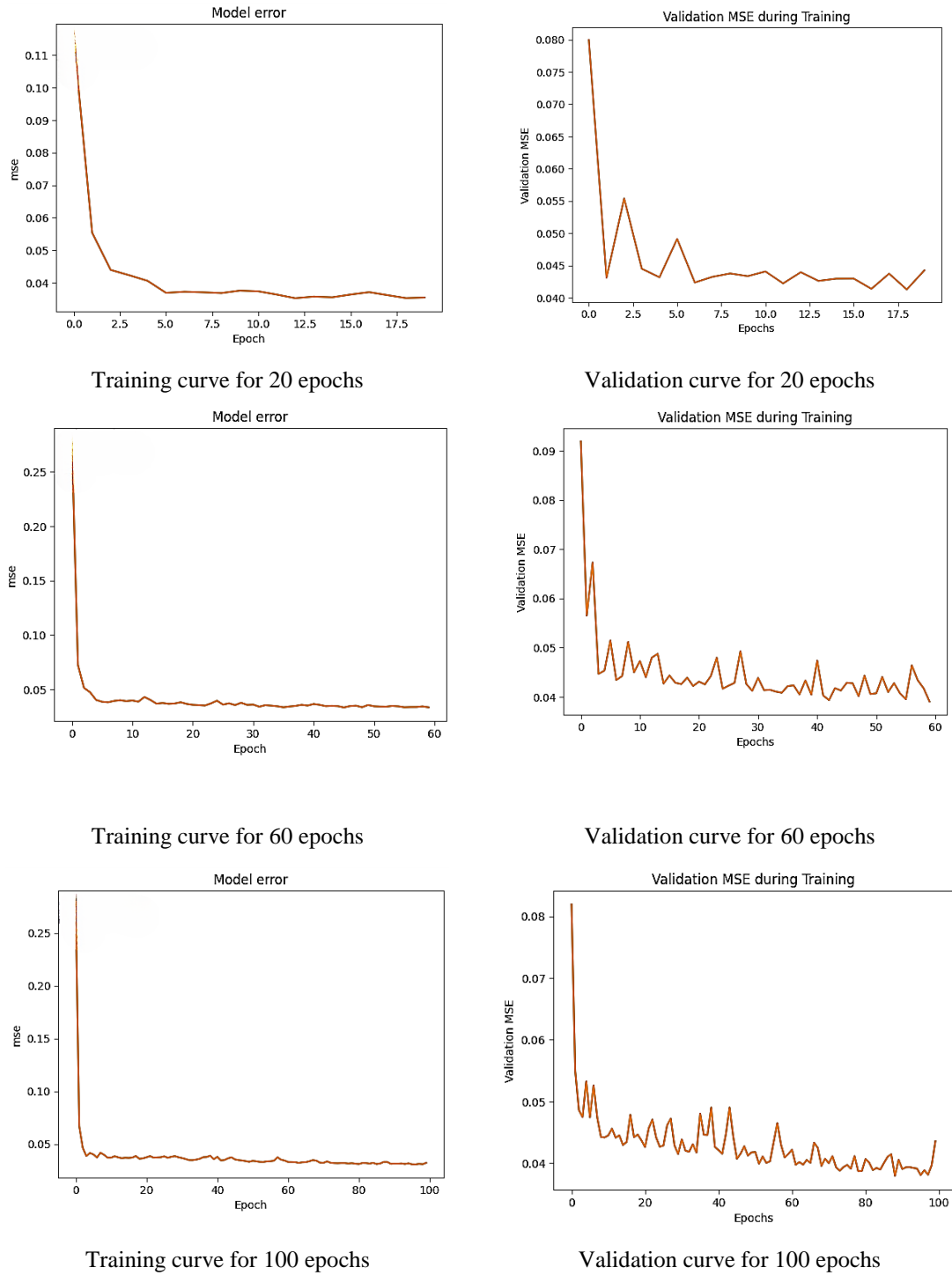


Figure 3.12: Malaria performance metrics using ResNet50.

When implementing the ResNet architecture, we found that training for 60 epochs yielded superior detection results and produced smoother loss and validation curves. Despite conducting experiments with both 20 and 100 epochs, the outcomes remained acceptable in terms of performance and accuracy.

3.5.1.2 Result using Drone Dataset

The **Table 3-6** and the **figure 3.13** shows IOU and the performance metrics curve respectively using our ResNet Model.

Images	Number of epochs	IOU
1	20	53
	60	68
	100	71
2	20	49
	60	66
	100	70
3	20	47
	60	65
	100	67
4	20	34
	60	64
	100	63
5	20	30
	60	63
	100	61
6	20	24
	60	60
	100	62

Table 3-6: IOU results for Drone datasets using ResNet50, 6 samples.

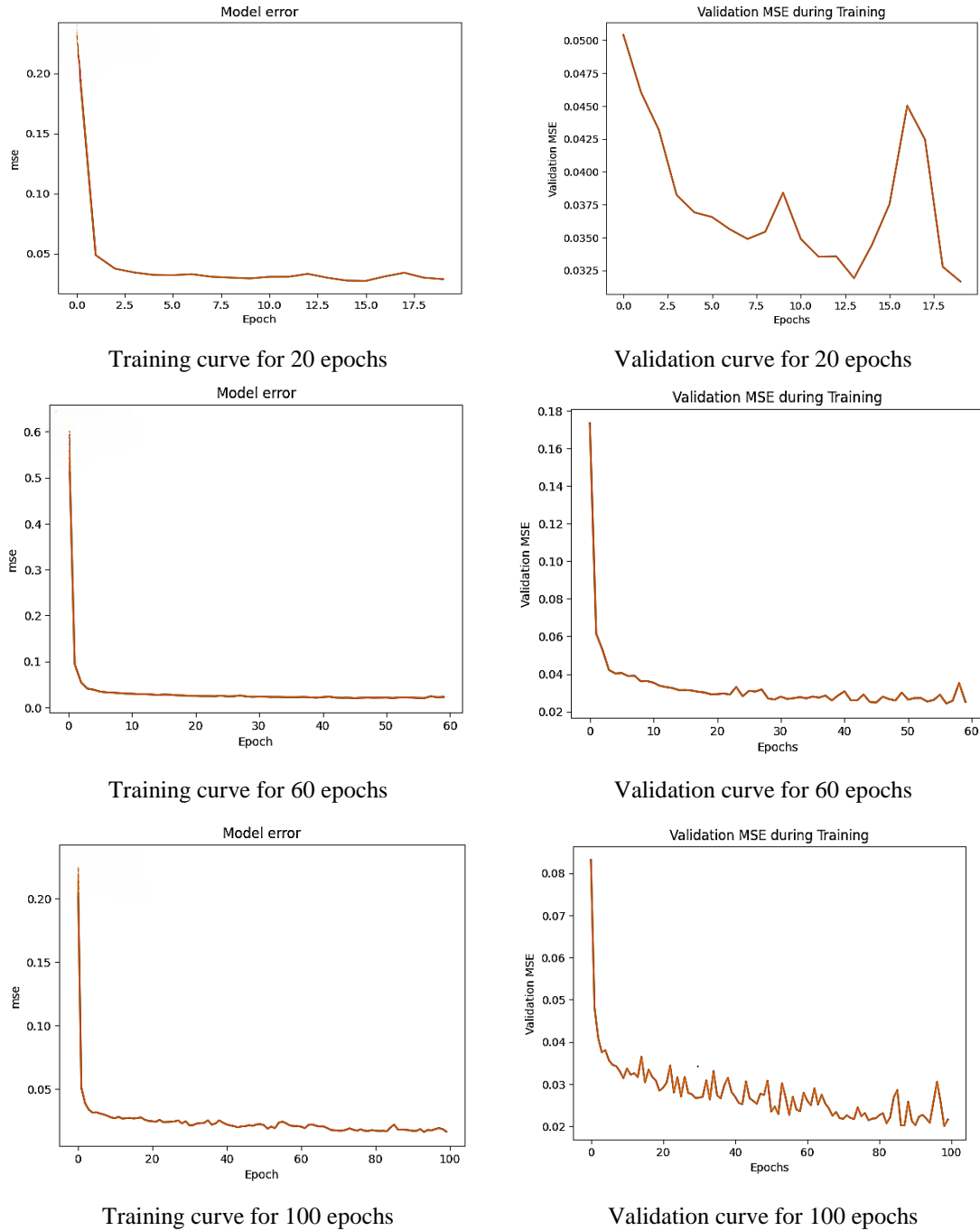


Figure 3.13: Drone performance metrics.

Upon applying the ResNet technique to our model, we noticed that the worst outcome was obtained with only 20 epochs. However, as we increased the number of epochs, we observed a significant improvement in the results.

3.5.1.3 Result using Lung Cancer Dataset

Table 3-7 and the **figure 3.14** bellow provides the results using the Lung Cancer Dataset

Images	Number of epochs	IOU
1	20	96
	60	68
	100	87
2	20	89
	60	89
	100	86
3	20	84
	60	84
	100	83
4	20	82
	60	83
	100	81
5	20	80
	60	82
	100	75
6	20	79
	60	78
	100	70

Table3-7: IOU results for Lung Cancer datasets using ResNet50, 6 samples.

As we saw when we implemented the CNN Model, we observed that the desired object in the Lung Cancer Dataset appeared consistently similar across all the images. As a result, we achieved excellent performance with 20, 60, and 100 epochs using ResNet.

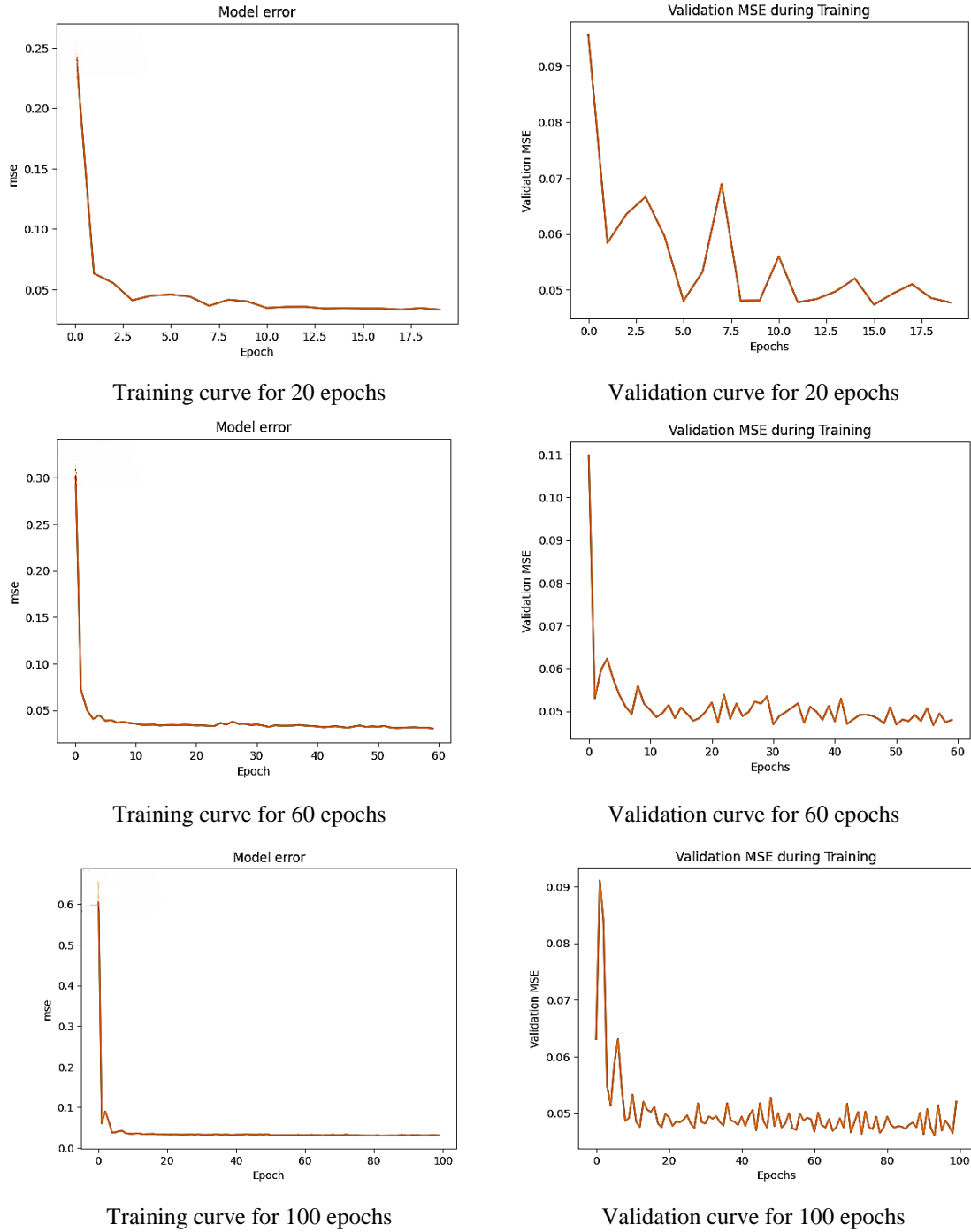


Figure 3.14: Lung Cancer performance metrics.

3.6 Comparative Analysis between the Results of CNN and ResNet Architecture

After implementing ResNet, we noticed significant improvements compared to our CNN model in terms of performance metrics. The loss and validation Mean Squared Error (MSE) decreased for all types of data, indicating enhanced performance. Furthermore, we were able to effectively reduce overfitting. The Table 3-8 below provide the total number of parameters using CNN and ResNet models.

Models	Total Params
CNN	22,759,846
ResNet	23,721,060

Table3-8: Total Parameters of ResNet and CNN models.

These two architectures produced IOU values ranging from 60 to 95, which is generally considered acceptable in the majority of cases. However, the number of parameters is big if our goal is to implement it in an embedded system. developing a model that has a lower number of parameters and at the same time gives high IOU is a real challenge, such models require powerful hardware, ample data, and a development team. The YOLO developers have successfully created multiple versions, and model names including YOLO-N(Nano), YOLO-S(Small), YOLO-M(Medium), YOLO-L(Large) and YOLO-X(X-large).

Next, we will implement the nano YoloV5 and nano YoloV8, the term “nano” stands for the low number of parameters.

3.7 YOLO implementation

After evaluating CNN and ResNet architectures, we decided to switch to YOLO due to its superior performance and extensive training on a large dataset. We implemented both YOLOv5 and the latest version, YOLOv8 (launched on January 10th, 2023), and conducted a thorough comparison between them.

3.7.1 The difference between YOLOv5 and YOLOv8

When it comes to object detection, there are numerous models to choose from. Among them, YOLOv8 and YOLOv5 stand out as two highly popular and state-of-the-art models developed by Ultralytics. In our thesis, we will concentrate on the nano variants of these two versions, which can be effectively deployed on embedded systems. YOLOv8, the latest addition to the YOLO family, builds upon the success of previous versions and introduces new features and enhancements to improve performance and flexibility. On the other hand, YOLOv5 is celebrated for its speed, simplicity, and accuracy.

When selecting the optimal object detection model, several factors come into play. These factors encompass speed, accuracy and ease of use. In terms of speed, both YOLOv8 and YOLOv5 exhibit fast processing capabilities, with YOLOv8 being notably faster (as shown in **Table 3-9**), making it suitable for real-time object detection applications. In terms of accuracy, YOLOv8 outperforms YOLOv5 due to architectural enhancements. As for ease of use, both models are easy to use, but YOLOv5, built on the PyTorch framework, offers the easiest integration and deployment for developers [13].

Models	Params (Million)	Accuracy (mAP)	CPU Time(ms)	GPU Time(ms)
YOLOv5n	1.9	45.7	45	6.3
YOLOv8n	3.2	37.3	8.4	0.99

Table3-9: Total Parameters of YOLOv5n and YOLOv8n models.

3.7.2 YOLOv5 results

We used TensorBoard for plotting graphs of training loss, Recall, precision and MAP metrics, where TensorBoard is web-based visualization tool provided by TensorFlow, a popular Deep Learning framework. It is used for visualizing and analyzing various aspects of Machine Learning Models performances.

- **Malaria Dataset**

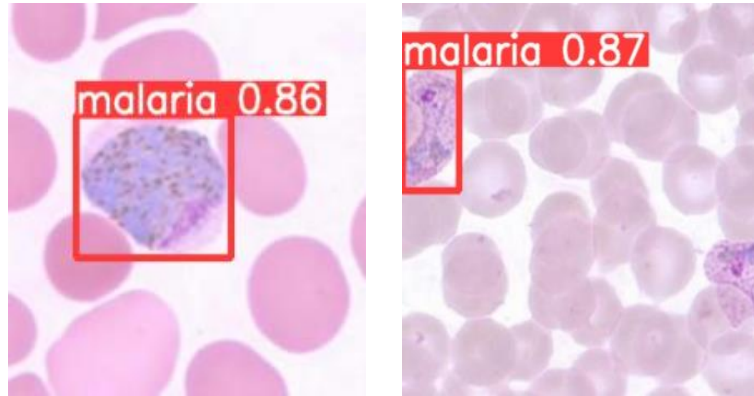


Figure 3.15: Malaria detection using YOLOv5.

Figure 3.16 represents the plots of the loss and MAP metrics during training of YOLOv5 on Malaria Dataset.

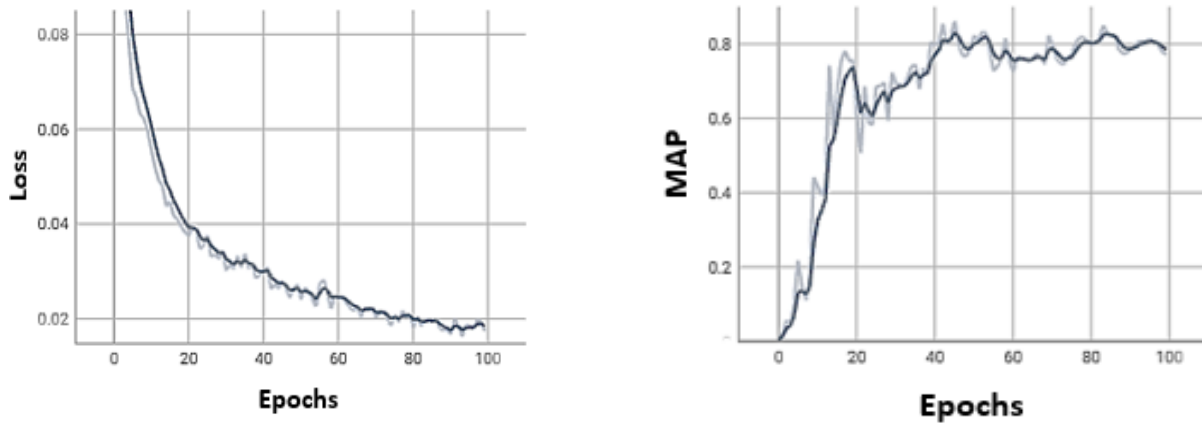


Figure 3.16: MAP and loss plots after training the YOLOv5 Nano on the Malaria detection Dataset.

Figure 3.17 below shows the precision and Recall graphs during the training of Malaria Dataset using YOLOv5.

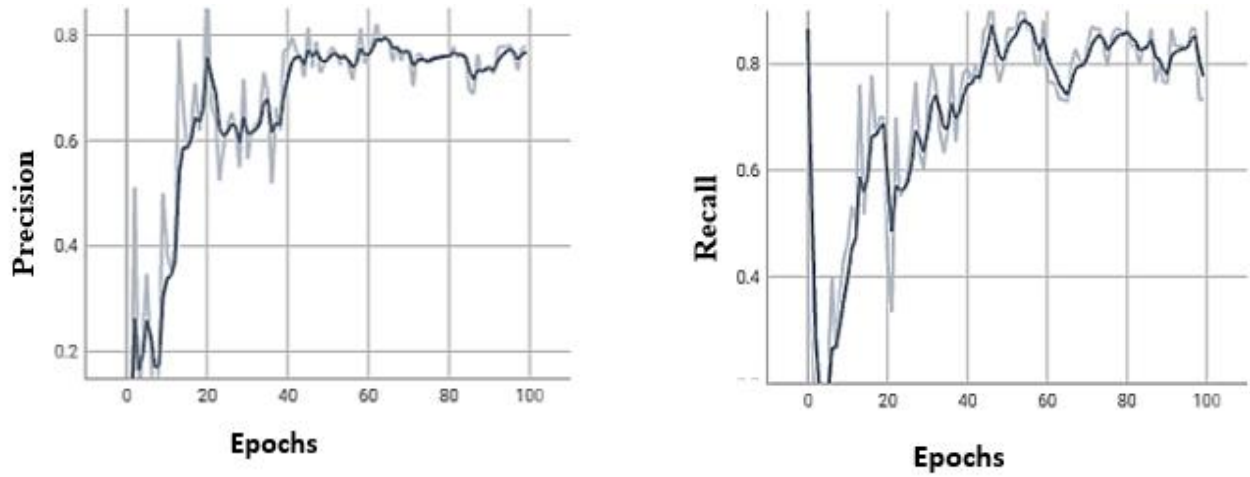


Figure 3.17: Precision and Recall plots after training the YOLOv5 Nano on the Malaria detection Dataset.

- **Drone Dataset:**

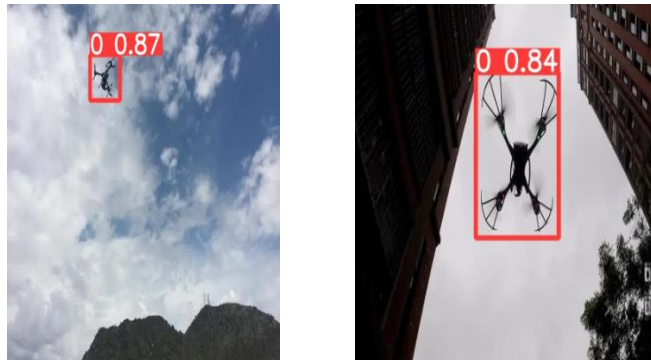


Figure 3.18: Drone detection using YOLOv5.

The Results of the loss, MAP, Recall and precision of YOLOv5 training on Drone Dataset are shown on the **figure 3.19** and **figure 3.20** below.

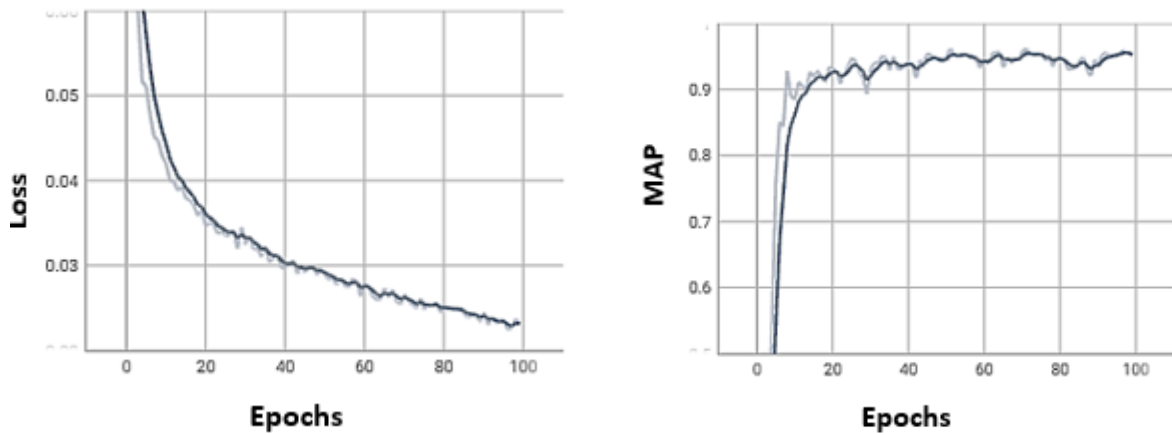


Figure 3.19: MAP and loss plots after training the YOLOv5 Nano on the Drone detection Dataset.

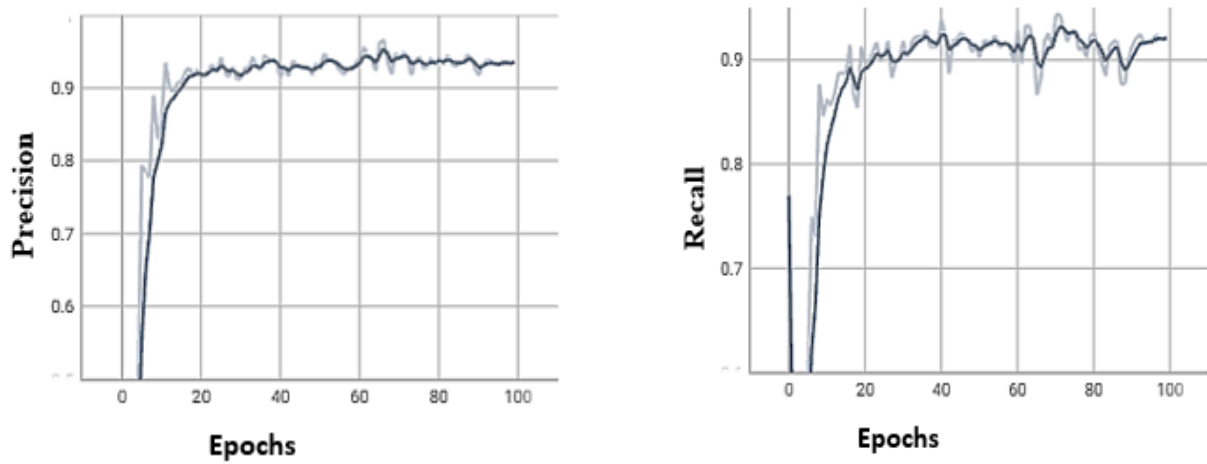


Figure 3.41: Precision and 1 Recall plots after training the YOLOv5 Nano on the Drone detection Dataset.

- **Lung Cancer Dataset**

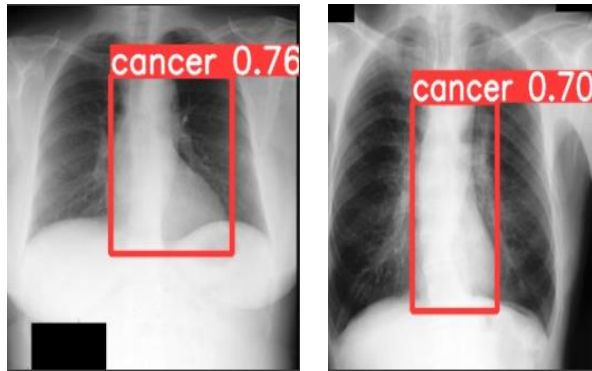


Figure 3.20: Lung Cancer detection using YOLOv5n.

For the Lung Cancer Dataset the training results are provided in the following figures .

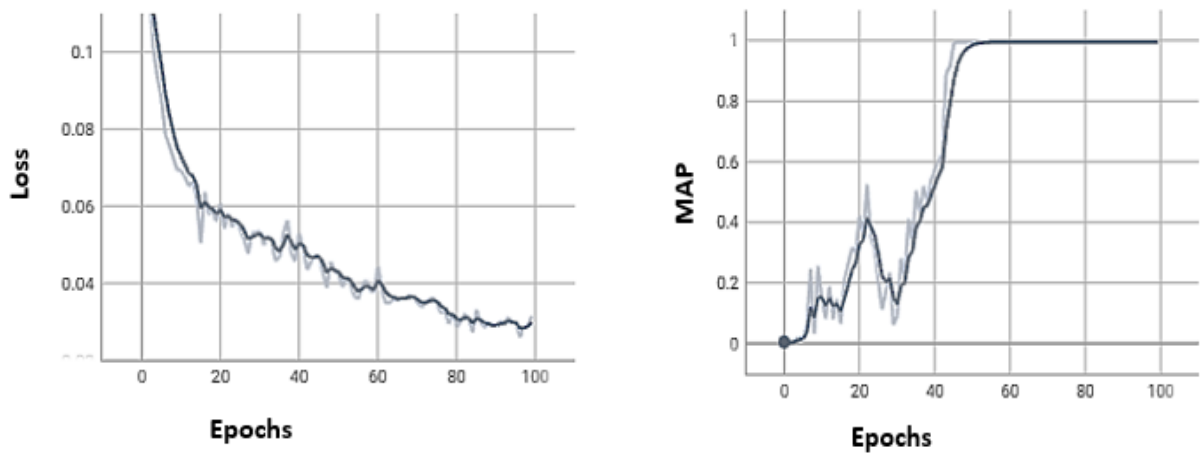


Figure 3.21: MAP and loss plots after training the YOLOv5 Nano on the Lung Cancer detection Dataset.

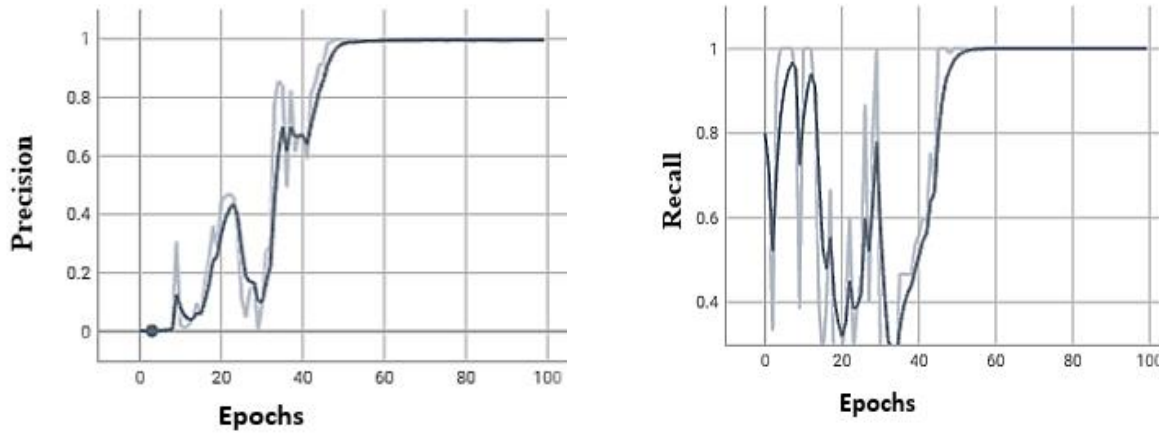


Figure 3.22: Precision and l Recall plots after training the YOLOv5 Nano on the Lung Cancer detection Dataset.

3.7.3 YOLOv8 results

Now we move to the results of the YOLOv8 implementation on our customized Datasets.

- **Malaria Dataset**

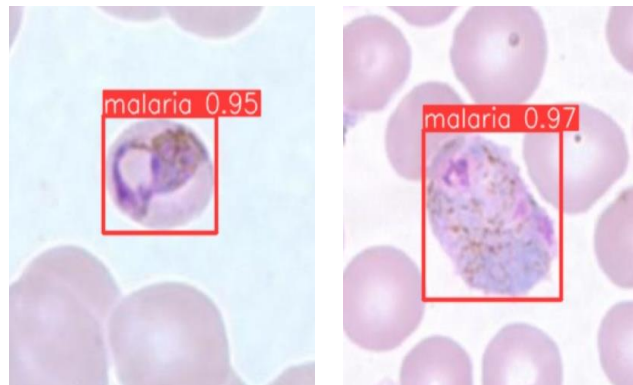


Figure 3.23: Malaria detection using YOLOv8.

The following figures illustrate the outcomes of the YOLOv8 implementation on Malaria Dataset.

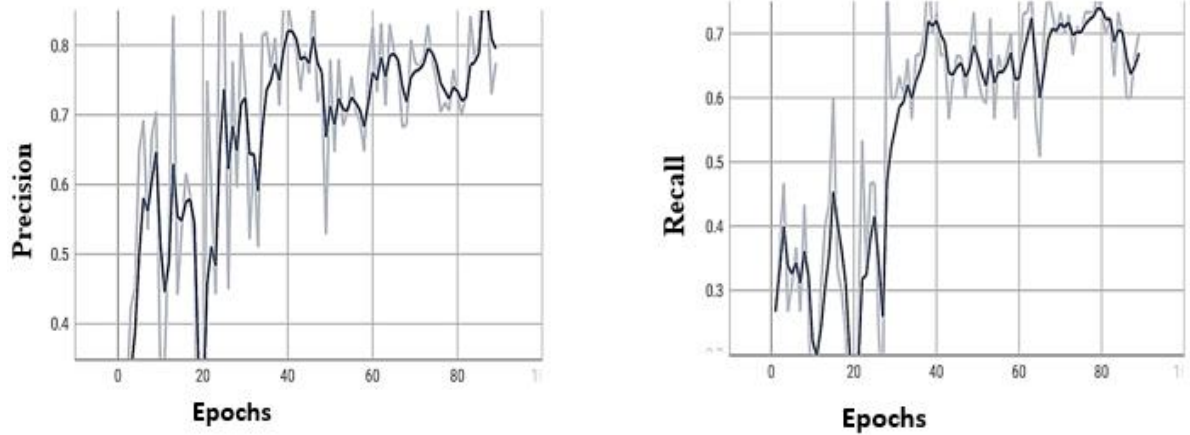


Figure 3.24: Precision and Recall plots after training the YOLOv8 Nano on the Malaria detection Dataset.

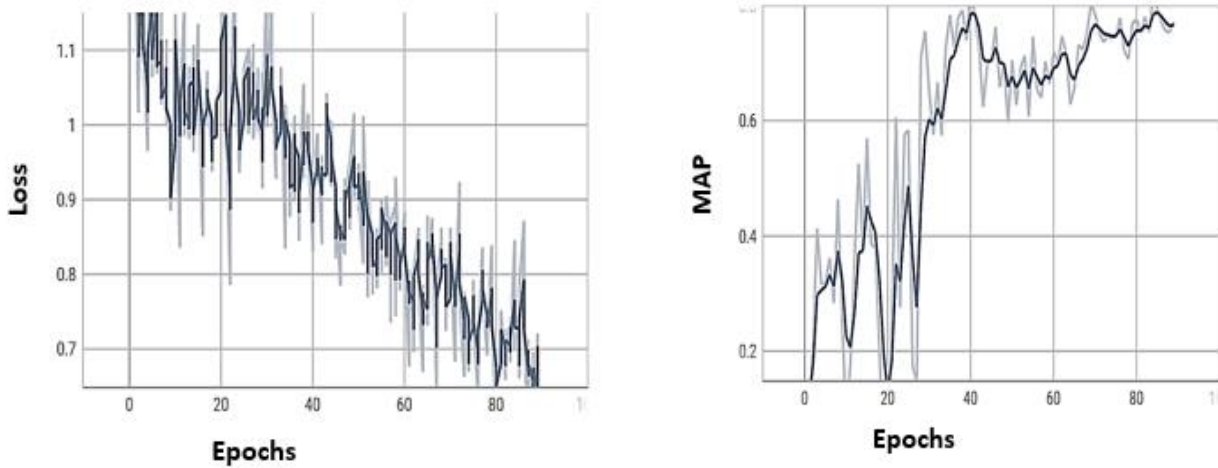


Figure 3.25: MAP and loss plots after training the YOLOv8 Nano on the Malaria detection Dataset.

- **Drone Dataset**



Figure 3.26: Drone detection using YOLOv8.

The provided figures showcase the results obtained from the implementation of YOLOv8 on the Drone Dataset.

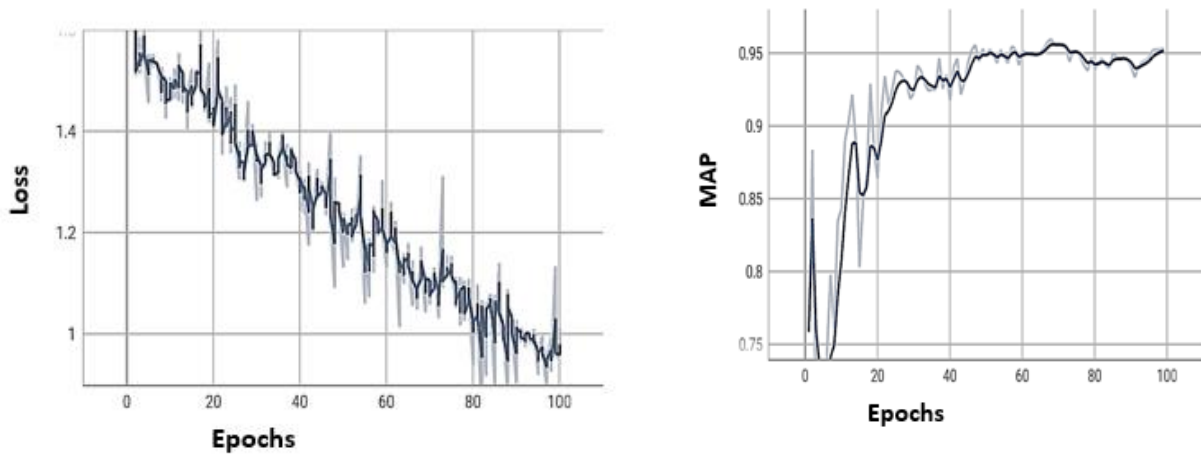


Figure 3.27: MAP and loss plots after training the YOLOv8 Nano on the Drone detection Dataset.

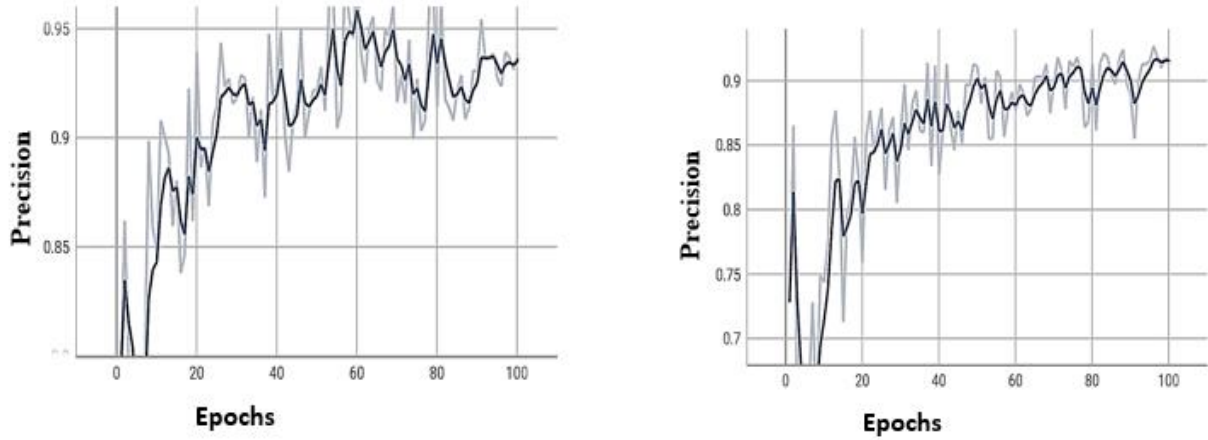


Figure 3.28: Precision and Recall plots after training the YOLOv8 Nano on the Drone detection Dataset.

- **Lung Cancer Dataset**

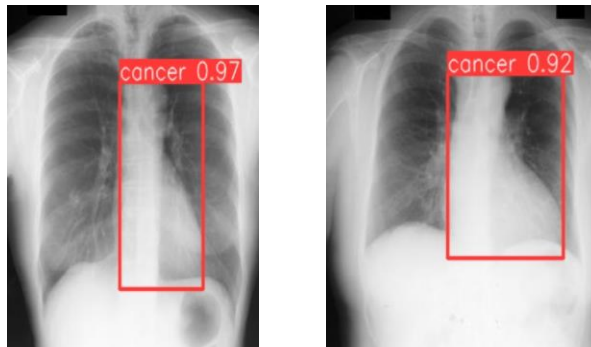


Figure 3.29: Lung Cancer detection using YOLOv8.

The performance metrics for loss, Mean Average Precision (MAP), recall, and precision of YOLOv8 training on the Lung Cancer Dataset are depicted in **figures 3.30** and **3.31**.

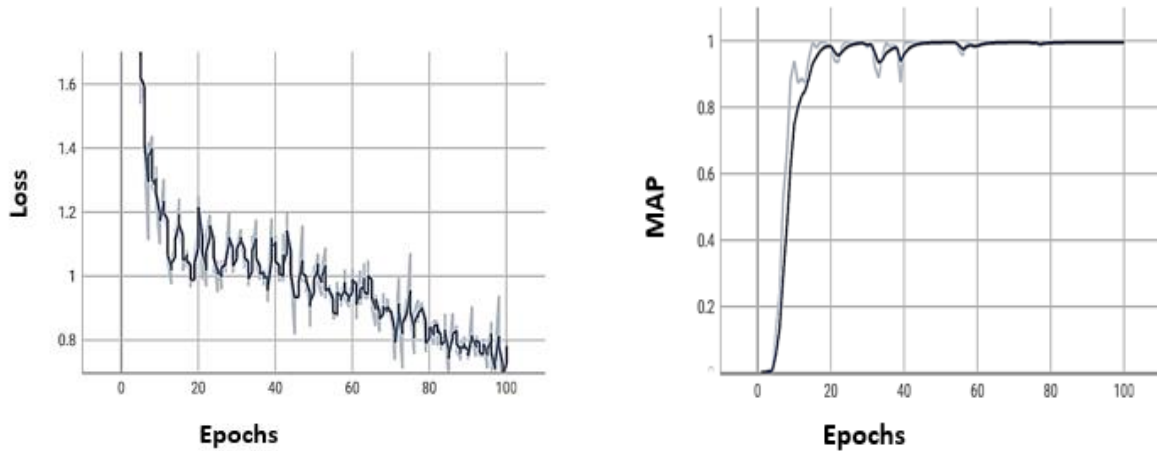


Figure 3.30: MAP and loss plots after training the YOLOv8 Nano on the Lung Cancer detection Dataset.

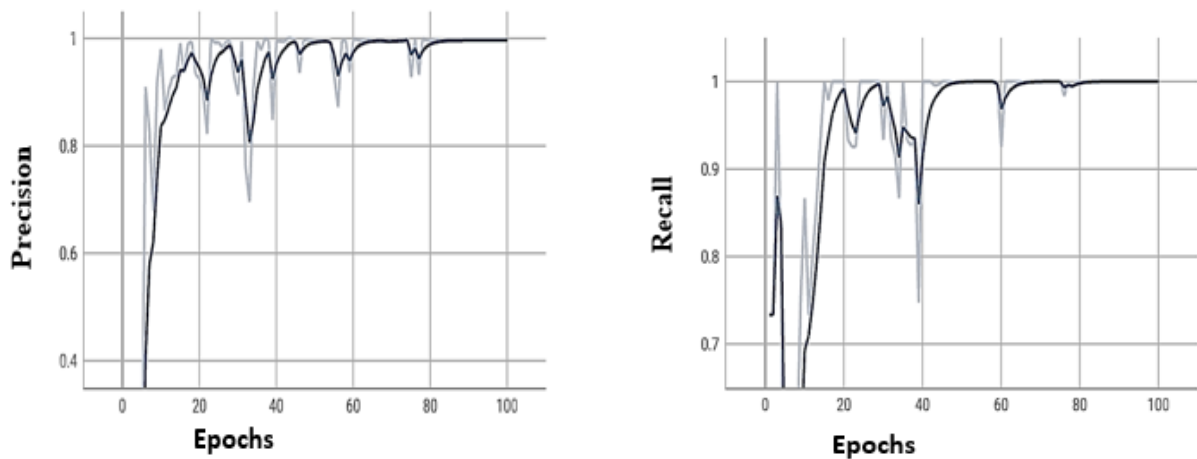


Figure 3.31: Precision and Recall plots after training the YOLOv8 Nano on the Lung Cancer detection Dataset.

3.7.4 YOLOv5 and YOLOv8 Results Interpretation

With only 1.9 million and 3.2 million parameters, the YOLOv5n and YOLOv8n models proved to be effective compared to the CNN model, which had 23 million parameters. They showed low loss values, high MAP scores, and accurate object detection performance with impressive recall and AP values.

We have done our comparison with the same Datasets on both, YOLOv5n and YOLOv8n and same hyper-parameters, we train them with 100 epochs. Images size is 300×300 which was previously rescaled.

The loss metric serves as a measure of the model's ability to effectively reduce the disparity between the predicted bounding boxes and the ground truth annotations throughout the training process. As we monitored the model's progress, in both YOLOv5n and YOLOv8n we observed a gradual decline in the loss values. This decline indicates that the model became increasingly adept at accurately predicting the bounding boxes, aligning more closely with the ground truth annotations over time. The diminishing loss values signify the model's improved capability to minimize the deviation between predicted and actual object locations, ultimately leading to enhanced object detection performance.

MAP (Mean Average Precision) serves as a widely adopted metric in object detection, providing an assessment of the precision and recall trade-off across various object categories as we mentioned in the chapter 2. Throughout the training of YOLOv5n and YOLOv8n, we closely monitored the progress and noticed a consistent increase in MAP scores. This upward trend in MAP values signifies notable improvements in detection performance across the complete.

Spectrum of object classes. Higher MAP scores indicate that the models achieved a more balanced combination of precision and recall, resulting in enhanced accuracy and reliability in identifying objects of diverse categories. This overall advancement in MAP highlights the effectiveness of YOLOv5n and YOLOv8n in capturing a wide range of objects with increased precision and recall compared to earlier stages of training.

3.8 Conclusion

In this chapter we present the steps that we passed through to choose the appropriate dataset which is represented by the Lung cancer, Malaria and Drone datasets. Once we identified the suitable dataset, we proceed to construct our CNN model, to further enhance the capabilities of our model we explore the concept of transfer learning by leveraging a pre-trained ResNet Network. In addition to the CNN model, we also investigate the performance of two prominent object detection algorithms YOLOv5n and YOLOv8n and we compared the results.

In this implementation we achieved our goal to find the suitable object detection model with high speed and accuracy. For us the YOLOv8n was the best and accurate for object detection tasks, on the other hand the CNN model was also effective but it's hard to build an appropriate architecture with customized dataset.

Conclusion

We have made significant advancements since the first chapter, successfully achieving our objective of object detection.

We implement our model on multiple datasets in various fields such as healthcare, the military security by detection military aircraft in the national airspace and Drone detection for save the National Internal Security from Espionage. We look forward to utilization of these technologies by the government in order to keep pace with advancement taking place in the world.

To recap, in our theoretical part, we tried to understand the problem that is inserted in our theme and finding appropriate ways to solve. In the other hand, our practical part contains the methods that are used for object detection implementation, among the techniques used we found that the efficient in aspect of Accuracy, speed, real time detection was the YOLO but according to the used version, for instance we implement YOLOv5n and YOLOv8n we conclude that eightieth version was the best for us.

This project has introduced us to a vast world, which is artificial intelligence, we have learned new things such as python as programing language, mastered Google Colab, and the fundamental principle of Deep Learning.

Bibliography

- [1] Janiesch, Christian & Zschech, Patrick & Heinrich, Kai. (2021). Machine learning and deep learning.
- [2] Lydia, Agnes & Chandrasekar, Sheela. (2022). A Comparative Study on Regularization Techniques in Convolutional Neural Networks. 784-793.
- [3] Buduma, N., Buduma, N. and Papa, J., 2022. *Fundamentals of deep learning*. " O'Reilly Media, Inc."
- [4] Wani, M. & Bhat, Farooq & Afzal, Saduf & Khan, Asif. (2020). Basics of Supervised Deep Learning. 10.1007/978-981-13-6794-6_2.
- [5] Zhang, A., Lipton, Z.C., Li, M. and Smola, A.J., 2021. Dive into deep learning. *arXiv preprint arXiv:2106.11342*.
- [6] Samreen, S.S., 2022. OBJECT DETECTION USING ARTIFICIAL INTELLIGENCE. *Journal of Engineering Sciences*, 13(12).
- [7] Redmon, Joseph & Divvala, Santosh & Girshick, Ross & Farhadi, Ali. (2016). You Only Look Once: Unified, Real-Time Object Detection. 779-788. 10.1109/CVPR.2016.91.
- [8] n: Gharaibeh, M.; Alzu'bi, D.; Abdullah, M.; Hmeidi, I.; Al Nasar, M.R.; Abualigah, L.; Gandomi, A.H. Radiology Imaging Scans for Early Diagnosis of Kidney Tumors: A Review of Data Analytics-Based Machine Learning and Deep Learning Approaches. *Big Data Cogn. Comput.* 2022, 6, 29
- [9] Sunil, Gagandeep May 2019. Study of Object Detection Methods and Applications on Digital Images.
- [10] Bochkovskiy, A., Wang, C.Y. and Liao, H.Y.M., 2020. Yolov4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934*.
- [11] Zhao, Zhong-Qiu & Zheng, Peng & Xu, Shou-Tao & Wu, Xindong. (2019). Object Detection With Deep Learning: A Review. *IEEE Transactions on Neural Networks and Learning Systems*. PP. 1-21. 10.1109/TNNLS.2018.2876865.
- [12] Padilla, R., Netto, S.L. and Da Silva, E.A., 2020, July. A survey on performance metrics for object-detection algorithms. In *2020 international conference on systems, signals and image processing (IWSSIP)* (pp. 237-242). IEEE.

- [13] Terven, Juan & Cordova-Esparza, Diana-Margarita. (2023). A Comprehensive Review of YOLO: From YOLOv1 to YOLOv8 and Beyond.
- [14] Kateb, Faris & Monowar, Muhammad Mostafa & Hamid, Md. Abdul & Ohi, Abu & Ph. D., M.. (2021). FruitDet: Attentive Feature Aggregation for Real-Time Fruit Detection in Orchards. *Agronomy*. 11. 2440. 10.3390/agronomy11122440.
- [15] Horvat, Marko & Jelečević, Ljudevit & Gledec, Gordan. (2022). A comparative study of YOLOv5 models performance for image localization and classification.
- [16] Gaspar, Angel & Oliva, Diego & Cuevas, Erik & Zaldivar, Daniel & Cisneros, Marco & Pajares, Gonzalo. (2021). Hyperparameter Optimization in a Convolutional Neural Network Using Metaheuristic Algorithms. 10.1007/978-3-030-70542-8_2.

Abstract

The thesis focused on implementing and enhancing object detection techniques using computer vision. Two main strategies were explored: Convolutional Neural Networks (CNN) and the You Only Look Once (YOLO) approach. The study began by examining the fundamentals of neural networks to gain a better understanding of object detection mechanisms. A custom CNN architecture was then developed and implemented to suit the specific datasets. Additionally, the performance of the proposed model was compared to YOLO through the implementation of YOLOv5 and YOLOv8. This allowed for the evaluation of the effectiveness of the custom approach and analysis of the results obtained from the different models.

Keywords: YOLOv5, YOLOv8, Convolutional Neural Network, computer vision, neural network, artificial intelligence (AI)

Résumé

La thèse se concentrait sur la mise en œuvre et l'amélioration des techniques de détection d'objets utilisant la vision par ordinateur. Deux stratégies principales ont été explorées : les réseaux neuronaux convolutifs (CNN) et l'approche You Only Look Once (YOLO). L'étude a commencé par examiner les fondamentaux des réseaux neuronaux pour mieux comprendre les mécanismes de détection d'objets. Une architecture CNN personnalisée a ensuite été développée et mise en œuvre pour s'adapter aux ensembles de données spécifiques. De plus, les performances du modèle proposé ont été comparées à celles de YOLO grâce à la mise en œuvre de YOLOv5 et YOLOv8. Cela a permis d'évaluer l'efficacité de l'approche personnalisée et d'analyser les résultats obtenus à partir des différents modèles.

Mots-clés : YOLOv5, YOLOv8, Réseau de neurones convolutionnels, vision par ordinateur, l'intelligence artificielle (IA)

ملخص

واحدة من التطبيقات الرئيسية لرؤية الكمبيوتر هو الكشف عن الأشياء. هذه الأطروحة تناولت تقنيتين متمثلتين في YOLO ، و CNN ، حيث قمنا بتطوير شبكتنا العصبية على مجموعته البيانات الخاصة بنا. بعد ذلك قمنا بمقارنه هذا النموذج المقترح لدينا من خلال تنفيذ كل من YOLOv5 و YOLOv8 ، سمح لنا هذا بتقييم فعالية نموذجنا وتحليل النتائج المتحصل عليها. **الكلمات المفتاحية:** الشبكة العصبية العميقة، الرؤية الحاسوبية، الذكاء الاصطناعي، YOLOv5، YOLOv8.