Exploring the Object Detection Algorithms in Computer Vision

Thesis Submitted in partial fulfilment of the requirements for the award of the degree of

Master of Science

in

Mathematics and Computing

Submitted by

Pooja Verma

302103005

Under the Supervision of

Dr. Harish Garg
Associate Professor, SOM

School of Mathematics

Thapar Institute of Engineering and Technology

Patiala Punjab, India-147004
July 2023

Certificate

This is to crefy the work presented in this thesid entiteled "Exploring the Object Detection Algorithms in Computer Vision" in partial filfillment of the requirement for the award of Degree of Master of Science in Mathematics and Computing, submitted in the School of Mathematics (SOM), Thapar Institute of Engineering and Technology, Patiala is an authentic record of my work carried out under the supervision of Dr. Harish Garg, Associate Professor at School of Mathematics Department, Thapar Institute of Engineering and Technology, Patiala.
The report has not been submitted to any other institute for the award of any other degree.

Pooja Verma
Roll No: 302103005

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

Verified by:

Dr. Harish Garg
(Associate Professor)

Acknowledgement

I would like to express my gratitude to professor **Dr. Mahesh Kumar Sharma**, Head of Department, School of Mathematics, Thapar Institute of Engineering & Technology, Patiala, who has given me the opportunity to work in this field and for providing us with best facilities in the Department and his timely suggestions.

I would like to make my deepest appreciation and gratitude to **Dr. Harish Garg** for his valuable guidance, and encouragement during the course of this project. Their advices and discussion were valuable and their attitude towards research always inspired me.

I want to thank **Dr. Rohit Garg** for providing a good background for my studies and enlightening my path of carrier.

I would also like to extend my thanks to all my classmates, lab mates for their time, invaluable suggestions and help. I feel very utmost thankful to **Miss Anjali**(Research Scholar) for helping me at each step in my research work. I thank my parents and family members for the support, encouragement and good wishes, without which I would not have been able to complete my Dissertation.

Pooja Verma
Roll No: 302103005

Abstract

This dissertation explores the groundbreaking field of computer vision and its application to object detection, focusing on the YOLO algorithms. The research investigates the use of YOLOv8 for cow stall number detection in dairy farming, aiming to automate and streamline management practices. Through extensive evaluation, YOLOv8 demonstrates superior generalization and accuracy in identifying cow stall numbers. Also, extensive analysis of performance of yolov7 and yolov8 on subsets of roboflow-100 dataset.

Chapter 2 lays the technical foundation, introducing machine learning and deep learning models for classification and object detection. In Chapter 3, the CowStallNumber dataset is examined, and the performance of YOLOv8 is evaluated with various model variants. The findings show that YOLOv8 achieves exceptional accuracy, making it a robust solution for cow stall number identification. Chapter 4 presents a comprehensive comparative analysis of YOLOv7 and YOLOv8 on diverse datasets. The insights gained help researchers and practitioners select the most suitable model for different object detection tasks.

In conclusion, this dissertation contributes significantly to the field of computer vision by showcasing the power of YOLOv8 for object detection. By automating cow stall number identification, dairy farmers can streamline management, improve productivity, and comply with industry regulations. The SGD optimizer-trained YOLOv8m model with a 1024-pixel input size is recommended for the task. The remarkable performance of YOLOv8 holds the potential to transform dairy farming practices and enhance production. Future studies may explore further optimizations, transfer learning, and broader applications of YOLOv8 in dairy farming and related domains.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Problem Statement

In recent years, computer vision has emerged as a groundbreaking field, transforming how robots perceive and interpret visual information in the world around us. One crucial aspect of computer vision is object detection, which aims to accurately recognize and locate various objects in images and videos. This capability has profound applications in robotics, surveillance, autonomous vehicles, medical imaging, and beyond. In this dissertation, we delve into the fascinating realm of object detection within the context of computer vision. The primary focus of this study is to investigate the state-of-the-art approaches and methodologies in object detection algorithms. With the power of deep learning and neural networks, modern object detection models have achieved impressive accuracy and real-time performance, allowing machines to understand complex visual environments like never before. Furthermore, object detection holds immense significance across various real-world applications. It becomes indispensable for autonomous driving, surveillance, robotics, and augmented reality by enabling automated systems to recognize and locate objects accurately. Deep learning algorithms like YOLO have revolutionized object detection by providing unified models that simultaneously predict multiple bounding boxes and class probabilities, ensuring exceptional speed and accuracy.

In dairy farming, efficient management is vital for optimal productivity. However, the individual management of cows can be time-consuming and error-prone, especially in large herds. By assigning cow stall numbers farmers can streamline administration, monitor cow health and productivity effectively, and maintain efficient record-keeping. This approach also aids in reproductive management and ensures compliance with industry regulations. Traditionally, establishing cow stall numbers required manual procedures, but our research leverages object detection technology to automate the process. By training the YOLOv8 algorithm on the CowStallNumber dataset, we evaluate its performance for different input sizes and model variants. Chapter 3 of this dissertation presents in-depth findings and their implications for dairy farm management. Chapter 4 of this dissertation compares the performance of the two latest YOLO versions, YOLOv7 and YOLOv8 on different size of datasets. Through this comparative analysis, we aim to provide valuable insights to researchers and practitioners, helping them select the most suitable YOLO model for their specific object detection tasks.

1.2 Contribution to Research

This dissertation makes significant contributions to object detection in computer vision. In Chapter 3, we explore the application of the YOLOv8 algorithm for cow stall number detection using the CowStallNumber dataset. Through extensive evaluation of the prediction results, we demonstrate that YOLOv8 exhibits superior generalization, making it a powerful and robust solution for

accurately identifying cow stall numbers. In Chapter 4, our research delves into a comprehensive comparative analysis of YOLOv7 and YOLOv8. By carefully evaluating their strengths and limitations, we provide valuable insights to researchers and practitioners, aiding them in selecting the most optimal model for various object detection tasks.

1.3 Thesis Structure

- **Chapter 2:** This forthcoming chapter is intended to provide readers with the technical foundation of Machine Learning and Deep Learning models created for the purposes of classification and object detection problems.
- **Chapter 3:** This chapter examines the cowstall dataset and gives information on the performance of the Yolov8 model with various variants.
- **Chapter 4:** This chapter presents a comparison between the performance of Yolov7 and Yolov8 on a variety of datasets that are accessible through Roboflow 100.

Chapter 2

Background

This chapter introduces the reader to some basic introduction to the terms Artificial Intelligence (AI), Machine Learning (ML), and Deep Learning (DL) Also, the technical background about Artificial Neural Networks, Convolutional Neural Networks and You Only Look Once (YOLO).

2.1 Artificial Intelligence

The term Artificial Intelligence (AI) was formally founded in 1956. the dream of building intelligent machines that can predict, classify and optimize drove the research in the field. Today, AI plays an immense role in many invisible tasks in our day-to-day life.

Artificial intelligence in the field of study essentially makes intelligent machines that can perform tasks like humans. It includes Machine learning (ML), Representation, Deep learning (DL), and Reinforcement learning. It has wide application in many sections due to its high-performance accuracy and wide applicability. However, many of these models need to be more capable of explaining their behaviour. It is difficult to penetrate the internal working of these models, earning them the title of "black-box" and "opaque" models.

This has given rise to a new and upcoming field of Explainable AI (XAI). It produces explainable AI models that can explain its results at each network level without losing performance accuracy. They are more reliable, trustworthy, and "transparent" models.

Another way to build trustworthy models is to develop an interpretation of these models. There have been many discussions about the "interpretability" and "interpretation" of black-box models of AI. The first means developing models explaining how they reached a particular result. At the same time, the latter explains how AI models make decisions using a particular approach. Interpretations can be further used for the interpretability of the model. Deep learning is the field of study in Artificial Intelligence that largely depends on how the human brain "works" to learn from a large amount of data. Deep learning solves many applications in today's growing world. It allows machines to learn from experience with no human intervention. Deep learning is gaining importance every day due to its high applicability in complex tasks that human can perform intuitively but is hard to explain to a machine.

2.2 Artificial Neural Networks

Artificial Neural Networks (ANNs) are extremely sophisticated computational models of layers of neurons based on the human brain. McCulloch and Pitts [6] created a simple logical neural model inspired by brain neural activity in 1943, which was the first thought of employing a computing

implementation of biological neurons. Frank Rosenblatt released the Perceptron Model [34] in 1958, which is still considered the conceptual foundation of modern multi-layer artificial neural networks.

2.2.1 Perceptron

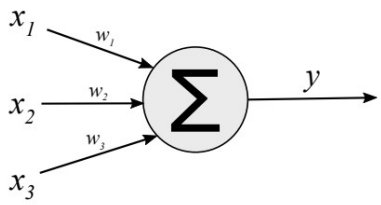


Figure 2.1: A Simple Perceptron

This section introduces Perceptron's fundamental notion. This neuron, shown in Figure 2.2.1, contains inputs $x_1, x_2, \cdots, x_n$ and variable weights $w_1, w_2, \cdots, w_n$ that output a binary value y. The output can be expressed as a function:

$$y = \begin{cases} 1 & ; \quad \text{if } \sum_j x_j w_J > \text{threshold} \\ 0 & ; \quad \text{if } \sum_j x_j w_J \leq \text{threshold} \end{cases} \tag{2.2.1}$$

We simplify Equation (2.2.1) by expressing the sum term as a dot product and inserting a bias b $= -$threshold, indicating a new weight w_0 for a new pseudo-input $x_0 = 1$, so that the decision rule may be rewritten as follows:

$$y = \begin{cases} 1, \text{if } w.x > 0 \\ 0, \text{otherwise} \end{cases} \tag{2.2.2}$$

In other words, the Perceptron takes inputs x, multiplies them by weights w that represent the significance of the input, and outputs $y = 1$ if the result exceeds a predefined threshold. This model can demonstrate artificial intelligence by learning the appropriate weights for a task. Assume we have several instances of inputs x with true labels (or targets) y. By altering the consequences w, we want to develop a Perceptron to learn to output the correct output y for each instance x. The Perceptron Learning Algorithm explained this technique. In essence, the learning algorithm modifies the randomly initialized weights based on what the correct output should be compared to its true label iteratively until all inputs match their labels correctly. If we provide a positive data example as input, we want the model to output $y = 1$, or in other words, $x.w > 0$. If this is not the case, the learning method will increase the weights. It would decrease the weights for negative cases such that $x.w < 0$

Rosenblatt demonstrated that the Perceptron could solve the logical operations AND, OR, and NOT with two inputs and one output value. However, Minsky and Papert [28] highlighted some of Perceptron's limitations, most notably the required linear separability of the data, which famously means the model's inability to solve the XOR problem. The XOR problem and many more complex problems necessitate a more appropriate layout of stacked neurons in multi-layer systems.

2.2.2 Modern Neurons and Activations

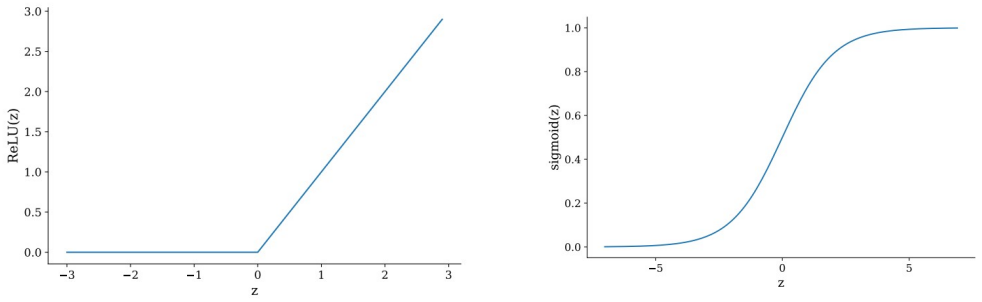
Before coming to the concept of Multi-Layer Perceptrons, it is necessary to first relax the defination of the perceptron and substitute it with a contemporary representation of a neuron (or unit). The output function described in 2.2.1 employs a severe step function based on the dot product of the input and weights. Modern neurons apply varying functions on $x.w$ to output a result y, named

activation functions. We provide two widespread activations that are also employed in this thesis. Let x represent a neuron's input vector, $z = x.w$, and y represent its single output, which relies on the activation function applied:

$$y = \text{ReLU}(x) = \max(0, x) \tag{2.2.3}$$

$$y = \sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.2.4}$$

The first activation function described in Equation 2.5.3 is the ReLU ("rectified linear unit") activation function [29]. In contemporary neural networks, it is the de facto standard activation function. The sigmoid (σ) activation function is represented by Equation 2.2.4. Because one of its properties



(a) ReLU Activation

(b) Sigmoid Activation

Figure 2.2: Plots of the ReLU activation function and Sigmoid activation function.

is that it limits the output range to [0, 1]; it is frequently used as an activation function in a single output layer to represent an output probability for binary classification tasks. Figure 2.2 illustrates these two functions visually.

Non-linearity is arguably the most essential attribute of current activation functions. If neurons performed a sequence of linear computations, their utility would be limited to linear separation tasks. By adding non-linearity through these activation functions, we meet a necessary criterion for approximating complicated non-linear functions using multi-layer neural networks. The differentiability and gradient features of the activations are also relevant aspects, as we shall see in the following sections.

2.2.3 Multi-Layer Perceptrons

A Multi-Layer Perceptron (MLP) is a network of layers filled with neurons. Figure 2.3 shows an MLP with four layers, two inputs (x_1, x_2), and a single output. The first layer displayed is the input layer containing the input data. Despite having the exact representation as the other neurons, they do not perform any calculations and output the input values. The inputs are subsequently routed to several hidden levels. The term merely denotes that these layers are neither input nor output. Finally, the network produces a single value y. Adding more hidden layers deepens the network while increasing the number of units in a layer widens the layer. Although there is no universal definition, networks with more than three layers are commonly referred to as "deep networks" (hence "Deep Learning"). We may also change the number of output neurons to meet our goal, for example, ten neurons for a classification task with ten potential classes.

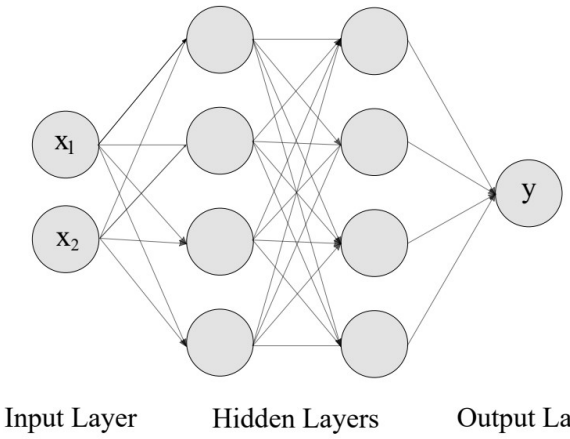


Figure 2.3: A multi-layer perceptron with an input layer, two hidden layers, and an output layer.

One noticeable feature of this MLP is that information travels exclusively in one way, from the input layer via the hidden levels to the output layer. This is why this network is also known as a feedforward neural network. The connections between neurons are another topological aspect. Figure 2.3 depicts a typical setup in which a neuron consumes all the outputs from all preceding neurons. This layer is known as a fully-connected or dense layer. MLP designs are highly versatile, spanning from sparse connections to recurrent information flow, and are beyond the scope of this background chapter.

A multi-layer network's complexity can make it extremely powerful. Unfortunately, the MLP cannot train with the Perceptron Learning Algorithm. Deep learning research was in a state of decline until the breakthrough backpropagation method [36] became widely known. Essentially, this approach efficiently propagates the mistake backward through the network to get cost gradients in relation to the weights. Because we recognize the importance of gradients, we replace the binary step function from the Perceptron with activation functions that have better differentiability and gradient properties.

2.2.4 The Components of a Neural Network

Input Data

The neural network requires inputs x, representing time series data, picture data, audio data, and much more. If the input contains true labels (or targets) y that the model should reproduce, the model is undertaking a supervised learning job. There are other options for instructing the network to build a supervisory objective independently (self-supervised learning) or to learn information from unlabeled input (unsupervised learning).

Network Architecture

The network architecture (also known as configuration) refers to the decisions made about the network's layers and neurons. The architecture may be made deeper by adding additional hidden layers, broader by adding more neurons to each layer, tweaked by modifying the activation functions, or more sophisticated layer combinations specified here. No optimum design can be used for every neural network; it is a hands-on, trial-and-error technique in which judgments are made in each case based on the job, input data and labels, computational power availability, and other factors. A network selection experiment is a frequent practice in which numerous distinct configurations are assessed and compared to each other to discover a high-performing architecture.

Cost Function

Finally, we want to build and optimize our network to obtain the highest performance attainable. To do so, we must develop a metric that accurately reflects our work performance and strive to improve it. In more technical terms, we want to use an optimizer to enhance a performance metric called the loss function L or the cost function $J(\theta)$ of the network given parameters θ. The binary cross-entropy loss function is used as an example. The specification of this loss function, which is mostly utilized for binary classification tasks, is shown in Equation 2.2.5.

$$Loss(y, \hat{p}) = -(y \log(\hat{p}) + (1 - y) \log(1 - \hat{p})) \tag{2.2.5}$$

The binary class labels $y = 0, 1$ relate to the data example's true label. In this case, $\hat{p}$ is the network's output (with parameters θ), expressing the chance that the model assigns to a data example belonging to class y = 1. If the network incorrectly predicts a very high probability $\hat{p}$ for a negative example with $y = 0$, the loss will be pretty large. On the other hand, an extremely high value of $\hat{p}$ for a positive example with $y = 1$ will drive the loss functions toward 0.

In this case, a smaller loss function indicates the network's classification performance is higher. The loss function is often written in this manner so that the optimizer is tasked with minimizing network loss and, ideally, improving network performance. The loss function often refers to the performance metric for individual data points, but the cost function $J(\theta)$ produces a measure for the entire dataset with regard to the network parameters θ. The cost function in our binary classification instance would be the total of the losses for each data sample. It may, however, be flexibly modified to include more detailed evaluations of single data point loss values. In a larger sense, the cost function defines what should be optimized, so it is frequently referred to as the objective (function).

Optimizer

We have established that we aim to optimize (typically minimize) a cost function $J(\theta)$. The optimizer in a neural network structure is entrusted with accomplishing this aim. Many optimization algorithms adhere to the basic principle of gradient descent. This iterative algorithm updates the parameters θ_i based on the cost gradient with respect to the network parameters (via backpropagation) until convergence or stopping criteria is met. It can be expressed as follows:

$$\theta_{i+1} = \theta_i - \alpha \Delta \mathrm{J}(\theta) \tag{2.2.6}$$

where α is the learning rate, which determines how steep the changes are, a high learning rate might result in irregular training without convergence to any form of optimum, whereas a low learning rate can cause training to be overly sluggish. Before performing an optimization step, this batch gradient descent approach examines each training example's cost. This can be sluggish and inefficient, especially for massive datasets required in deep learning. One method is to divide the data into smaller training samples mini-batches and optimize each separately. This fundamental method is extended by modern optimization techniques such as Adam [21].

2.3 Convolutional Neural Networks

Convolutional Neural Networks (ConvNets, CNNs) are primarily based on the findings of visual cortex in cats and monkeys conducted in the 1950s and 1960s [17] [18] [19]: Neurons served as receptive fields, firing when particular restricted portions of a visual field were revealed: some fired when raw edges were exhibited, while others fired when more complicated patterns were shown. CNNs apply this concept to pictures by functioning as feature detectors for restricted, particular patterns in images. Although a popularised publication dates back to 1989 with LeCun et al. [23] using a CNN for handwritten code recognition, it was with AlexNet [22] in 2012 that CNNs gained immense popularity for computer vision tasks. Krizhevsky et al. demonstrated a deep convolutional

neural network capable of breaking new ground in picture classification on the ImageNet challenge and an

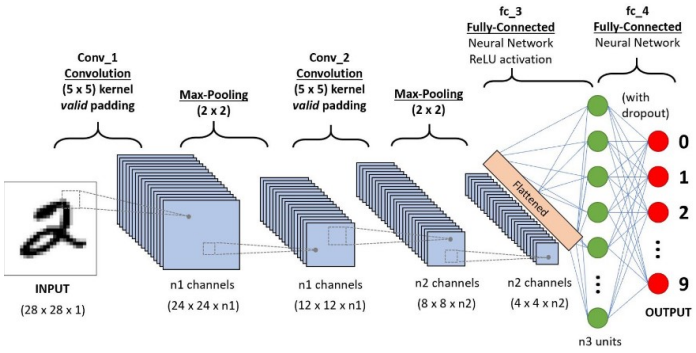


Figure 2.4: Architecture of CNN.

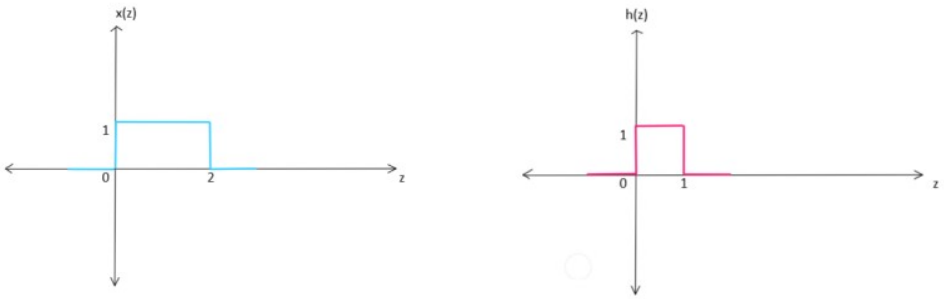
efficient GPU implementation to enable training despite its millions of parameters. As they state in their paper, it was just a matter of time and GPU advances until the performance could be boosted, especially with newer and more advanced models like ResNet [14].

2.3.1 Convolutional

Convolution of mathematical functions

Convolution is a mathematical procedure on two functions that yields a third function that expresses how the form of one is affected by the other in functional analysis. Convolution describes both the outcome function and the technique of computing it [5]. It is defined as the integral of the product of two functions when one is inverted and shifted. The integral is computed for all shift values, yielding the convolution function [12].

$$y(t) = h(t) * x(t) = \int_{-\infty}^{\infty} x(z)h(t-z)dz \tag{2.3.1}$$



(a) (b)

Figure 2.5: Graph of x(t) and h(t)

Here, $x(z)$ and $h(z)$ are depicted in Figures 2.5a and 2.5b, demonstrating that $x(z)$ is 1 if $0 < z < 2$ and 0 otherwise, and $h(z)$ is 1 if $0 < z < 1$ and 0 otherwise. Now we change $h(z)$ as $h(-z)$ so that its graph is shifted to the left of the origin, and then we modify it to $h(t-z)$ according to

the value of t. The graph of $x(z).h(t-z)$ is depicted in Figure 2.6a.

Now we remain to integrate it.
Case 1: when $t < 0$ then $y(t) = 0$ because $x(z) = 0$ for $z < 0$. So, $x(z).h(t-z) = 0$. Hence $y(t) = 0$ for $t < 0$.
Case 2: when $0 < t < 1$, then both $x(z)$ and $h(t-z)$ is 1 as shown in Figure 2.6a. So,

$$y(t) = \int_0^t x(z)h(t-z)dz = \int_0^t dz = 1 \tag{2.3.2}$$

Case 3: when $1 < t < 2$ then both $x(z)$ and $h(t-z)$ are 1 as shown in Figure 2.6a. So,

$$y(t) = \int_{t-1}^t x(z)h(t-z)dz = \int_{t-1}^t dz = 1 \tag{2.3.3}$$

Case 4: when $2 < t < 3$ then,

$$y(t) = \int_{t-1}^2 x(z)h(t-z)dz = \int_{t-1}^2 dz = 3 - t \tag{2.3.4}$$

Case 5: when $t > 3$ then $y(t) = 0$ because $x(z) = 0$ for $z > 2$ as shown in Figure 2.6a. So, $x(z).h(t-z) = 0$. Hence $y(t) = 0$ for $t > 3$.

(a) (b)

Figure 2.6: Graph of x(z).h(t-z) and y(t)

so,

$$y = \begin{cases} 0 & \text{if } t \leq 0 \\ t & \text{if } 0 \leq t \leq 1 \\ 1 & \text{if } 1 \leq t \leq 2 \\ 3-t & \text{if } 2 \leq t \leq 3 \\ 0 & \text{if } 3 \leq t \end{cases} \tag{2.3.5}$$

Convolution kernels

The convolutional layer is the heart of a convolutional network. This layer comprises multiple kernels, sometimes known as filters, that slide over the input (such as an image) to identify specific characteristics. These trainable weight filters conduct a convolution operation on the input and output a feature map. The convolution action of a kernel K over an input I is defined as follows:

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n) \tag{2.3.6}$$

Applying kernels on a black and white image set. We have a black-and-white image set with only one channel. So, we only have one kernel corresponding to that channel Figure 2.7 shows an input

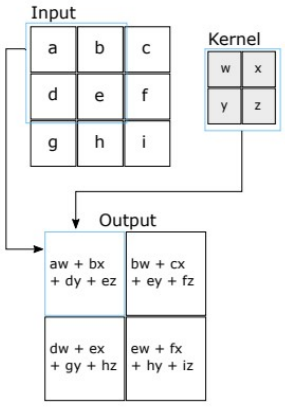


Figure 2.7: A (3×3) Input is convolved with a (2×2) kernel with trainable weights w, x, y, z. The output is of dimension (2×2).

of a 3×3 size picture set and a kernel of 2×2 size matching this image. The size of the output matrix is determined by the size of the image and kernel sets [12]. If we have an input of $M \times N$ and a kernel of $A \times B$ giving $A < M$ and $B < N$, then the output is of order $P \times Q$ where

$$P = \left\lceil \frac{M - A}{s} + 1 \right\rceil \tag{2.3.7}$$

$$Q = \left\lceil \frac{N - B}{s} + 1 \right\rceil \tag{2.3.8}$$

Here 's' is strider(Stride is the number of pixels shifts over the input matrix. s=1 means a one-pixel shift)

Applying convolution kernels on RGB image. Since we have three channels in an RGB image set, and for each channel, we have three different kernels. In Figure 2.8, we have a $6 \times 6 \times 3$ input set with three different kernels of 3×3 size for each channel. As illustrated in Figure 2.8, each value in the output matrix is computed. The issue with applying filters to input is that the picture becomes shrieked if we use a neural net with several layers, resulting in a small image after filtering. So, the pixel in the corner will only be covered once, but the pixel in the middle will be covered multiple times. This means we have more information on the middle pixel and some loss on the image's corners. Padding is used to address this issue which will be covered in the coming section.

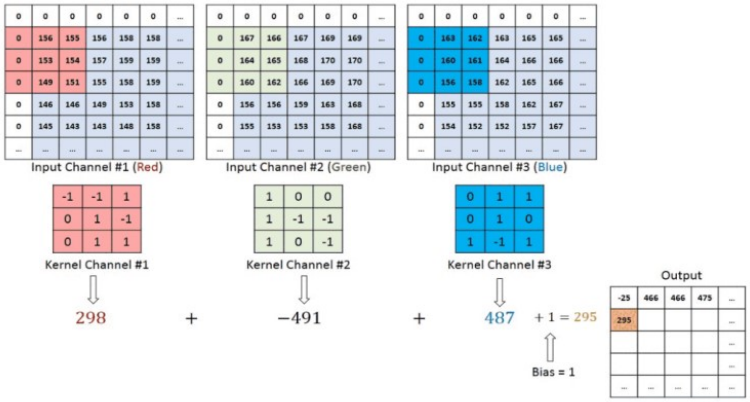


Figure 2.8: A $6 \times 6 \times 3$ Input is convolved with a $(3 \times 3 \times 3)$ kernel.

2.3.2 Concept of Padding and Pooling

Padding: Padding is added to our input to retain information on corners. Padding is merely the addition of layers of zeros to our input pictures. We add two rows (one at the top and one at the bottom) and two columns (one on the extreme left and one on the right) to our initial $M \times N$ size picture collection. As a result, our initial picture set of size $M \times N$ becomes $(M+2) \times (N+2)$. We can also add rows and columns as we see fit.

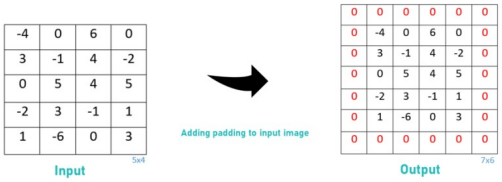


Figure 2.9: Padding in CNN

In Figure 2.9, the input size is 5×4, but after padding two rows and two columns, it becomes 7×6. Valid convolution occurs when no padding is added to the input picture. In this case Input matrix is of $N \times N$ size, and kernel is of $F \times F$ size, then the output is of $(N-F+1) \times (N-F+1)$ size. i.e. $(N \times N) * (F \times F) = (N-F+1) \times (N-F+1)$
Another circumstance in which padding is used is when we need the output to be the same size as the input, known as the same convolution. The size of the filter used determines this padding. We require such value,

$$N + 2P - F + 1 = N \tag{2.3.9}$$

$$\Rightarrow \quad P = \frac{F-1}{2} \tag{2.3.10}$$

Where input size is $N \times N$ and filter is of size $F \times F$ and P is the size of padding.
Case 1: when F is ODD, we usually take an odd filter as it focuses on the central pixel.
Case 2: when F is EVEN, then we have no such case of central pixel, and also the selection of number of padding becomes complex.

Pooling: A pooling procedure is commonly used on the output of a convolutional layer. Pooling summarises the information from the discovered features in the pooling input. There are several pooling operations, the most common of which are max-pooling and average pooling. The procedure

of max-pooling keeps the most significant value of a region, whereas average pooling retains the average value. The operation applies a pooling filter of a specific size and stride, with no trainable weights, over the input to generate a pooling output of the selected type. A max-pooling operation using a 2×2 filter and stride 2 is shown in Figure 2.9. Pooling is occasionally substituted with convolutional layers with higher strides, which simulate a subsampling effect [37].

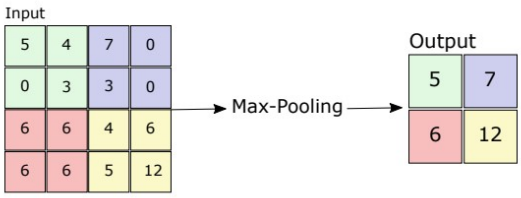


Figure 2.10: Pooling in CNN

2.4 CIFAR-10 dataset

The CIFAR-10 dataset is widely recognized as a benchmark dataset in computer vision and machine learning. It has played a pivotal role in the development and assessment of numerous image classification algorithms. Comprising ten distinct classes representing different objects and scenes, the dataset includes categories like aircraft, automobiles, animals (cats, deer, canines, amphibians), horses, ships, and vehicles. Published by the Canadian Institute for Advanced Research (CIFAR) in 2009, the CIFAR-10 dataset contains 60,000 32x32 color images, with each class comprising 6,000 images. The dataset is thoughtfully divided into two subsets: a training set containing 50,000 labeled images and a test set with 10,000 unlabeled images. This division allows researchers to effectively evaluate the performance of classification models by training them on a substantial amount of labeled data and then assessing their performance on unseen test samples. In conclusion, the CIFAR-10 dataset's significance lies in its role as a standardized benchmark for evaluating image classification models in the fields of computer vision and machine learning

Figure 2.11: Sample images of CIFAR-10 Dataset

2.5 Experiment Setup

In this study, we assessed the effectiveness of Multi-Layer Perceptron (MLP) and Convolutional Neural Network (CNN) models using the CIFAR-10 dataset. The MLP models were trained with

a learning rate of 0.001 and sparse categorical cross-entropy loss, employing the Adam optimizer. Meanwhile, the CNN models used the Adam optimizer with its default settings. To avoid overfitting, we incorporated early stopping during the training process. The experiments were conducted on Google Colab's T4 GPU, which significantly improved the efficiency of the training procedure.

2.6 Performance Measure of MLP and CNN

The provided table 4.2 contains the performance of MLP and CNN models on the CIFAR-10 dataset. Here are the insights drawn from the data:

Table 2.1: Performance of MLP and CNN Networks on CIFAR-10

Architecture	Accu (train)	Accu (valid)	epoches
MLP Models			
3 Dense layers (128-64-32 neurons)	0.6585	0.4682	356
4 Dense layers (128-64-32-20 neurons)	0.6174	0.4795	155
5 Dense layers (256-128-64-32-20 neurons)	**0.6835**	**0.4771**	**125**
CNN Models			
Conv(16)-MaxPool-Conv(32)-MaxPool-Conv(64)-MaxPool-Flatten-FC(128)	0.9304	0.6225	212
Conv(32)-MaxPool-Conv(64)-MaxPool-Conv(128)-MaxPool-Flatten-FC(256)-FC(128)	**0.9776**	**0.6646**	**70**
Conv(32)-MaxPool-Conv(64)-MaxPool-Conv(128)-MaxPool-Conv(256)-MaxPool-Flatten-FC(128)	0.9881	0.5744	53
CNN Models with Padding			
Conv(16)-MaxPool-Conv(32)-MaxPool-Conv(64)-MaxPool-Flatten-FC(128) (Padding)	0.9775	0.6675	79
Conv(32)-MaxPool-Conv(64)-MaxPool-Conv(128)-MaxPool-Flatten-FC(256)-FC(128) (Padding)	**0.9854**	**0.7117**	**63**
Conv(32)-MaxPool-Conv(64)-MaxPool-Conv(128)-MaxPool-Conv(256)-MaxPool-Flatten-FC(128)(Padding)	0.9796	0.6911	52

Performance of MLP Models: The table displays the results of three MLP architectures with varying numbers of dense layers and neurons. The accuracy of the training and validation sets is relatively lower for the MLP models than for the CNN models. Even the best-performing MLP model, "5 Dense layers (256-128-64-32-20 neurons)""5 Dense layers (256-128-64-32-20 neurons)," achieves only around 68.35% accuracy on the training set and 47.71% on the validation set.

Performance of CNN Models: The CNN models demonstrate significantly better accuracy on the CIFAR-10 dataset than the MLP models. The first CNN model, "Conv(16)-MaxPool-Conv(32)-MaxPool-Conv(64)-MaxPool-Flatten-FC(128)," achieves an accuracy of 93.04% on the training set and 62.25% on the validation set showcasing the benefits of using convolutional layers for image-related tasks.

Effect of Model Depth: Comparing the CNN models, it can be observed that deeper networks do not always guarantee better performance. The model "Conv(32)-MaxPool-Conv(64)-MaxPool-Conv(128)-MaxPool-Flatten-FC(256)-FC(128)" performs better on both the training and validation sets compared to the model with an additional convolutional layer ("Conv(32)-MaxPool-Conv(64)-MaxPool-Conv(128)-MaxPool-Conv(256)-MaxPool-Flatten-FC(128)"). Adding more layers does not always lead to improved performance and might increase the risk of overfitting.

Effect of Padding: The table also includes results for CNN models with padding. Padding helps retain spatial information during convolutional operations. The models with padding generally show slightly improved accuracy compared to their counterparts without padding, especially evident in the model "Conv(32)-MaxPool-Conv(64)-MaxPool-Conv(128)-MaxPool-Flatten-FC(256)-FC(128)".

Overfitting: While some CNN models achieve high training accuracy, there is a noticeable gap between training and validation accuracy, indicating possible overfitting—regularization techniques such as dropout, weight decay, or early stopping address this issue and improve generalization.

Training Efficiency: The CNN models tend to have fewer epochs than the MLP models, which indicates that they require fewer iterations to converge. This highlights the training efficiency and effectiveness of CNN architectures on image-based datasets like CIFAR-10.

In conclusion, the table illustrates the superiority of CNN models over MLP models for image classification tasks like CIFAR-10. CNN architectures are better at capturing spatial features and hierarchical patterns present in images, leading to improved accuracy. Deeper CNN models may not always provide additional benefits, and padding can enhance performance by preserving spatial information. Overall, CNN models outperform MLP models on the CIFAR-10 dataset and are more suitable for image classification tasks.

2.7 YOLO

YOLO (You Only Look Once) is a real-time object detection algorithm that uses neural networks. Joseph Redmon proposed it in his 2015 research article "You Only Look Once: Unified, Real-Time Object Detection" [30]. Because of its speed and precision, YOLO has been employed in various applications, such as detecting traffic lights, Autonomous vehicles, and Medical imaging. YOLO predicts object's height, width, center, and class using a single bounding box regression. The algorithm has changed by introducing new versions, such as YOLOv7 and YOLOv8.

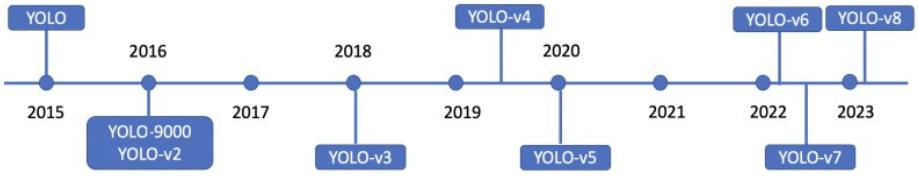


Figure 2.12: Evolution of YOLO

Idea of YOLO: Many object detection systems require multiple iterations of the image to detect every object or two stages to detect the objects. YOLO doesn't have to go through these tedious procedures. It just needs to look at the image once to identify every object, which is why the name "YOLO" (You Only Look Once) was chosen. It also explains why the model is so quick. The basic idea of YOLO is to have a single network for object detection, making box prediction and class prediction in a single shot. So, YOLO reframes the object detection problem as a single-stage regression problem; when we say regression problem, we mean all the losses we are calculating here are regression losses. Figure2.13 shows this in pictorial representation.

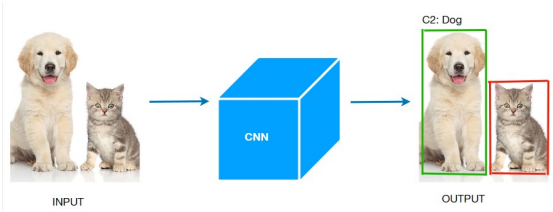


Figure 2.13: Idea of YOLO

2.7.1 Working of YOLO

In the YOLO (You Only Look Once) step-by-step process, the first step involves taking an input image of any resolution. Next, the input image is resized to a fixed size of 448×448, which remains constant in YOLOv1. The resized image is then divided into a grid of $S \times S$ cells, where S is set to 7 in YOLOv1. As a result, the image is split into a 7×7 grid, with each grid cell covering an area of 64×64 pixels due to the division $448/7 = 64$, as visually depicted in Figure 2.14. During the object detection process, the algorithm checks if the center of an object falls within a particular grid cell. If the object's center lies within a specific grid cell, that grid cell takes on the responsibility of detecting and localizing the object. In other words, the object is assigned to the grid cell where its center point occurs, allowing the algorithm to accurately associate objects with the corresponding grid cells for effective object detection.

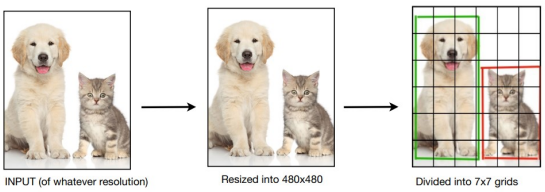


Figure 2.14: Input division

The YOLO (You Only Look Once) algorithm performs both classification and localization tasks simultaneously on each of the $7 \times 7 = 49$ grid cells. However, due to the constraints of the classification and localization network, each grid cell can identify only one object.
Despite its advantages, YOLO encounters certain challenges due to its grid-based approach. With a 7×7 grid, YOLO can potentially detect up to 49 objects. However, when multiple objects fall within a single grid cell, the model might struggle to identify all of them, leading to a close object detection difficulty in YOLO. This issue arises when an object spans multiple grid cells, and the model might mistakenly detect the same object multiple times.
To address this problem, YOLO employs a technique called non-max suppression. Non-max suppression helps alleviate redundancy by filtering out less confident bounding boxes and retaining only the most confident prediction for each object. This process ensures that each object is detected only once, even if it spans across multiple grid cells, thereby enhancing the accuracy and efficiency of object detection in YOLO.
In YOLO, each of the 7×7 grid cells predicts B bounding boxes (often set to $B = 2$), and the model assigns a confidence score to each box. The confidence score reflects the model's level of certainty that the corresponding bounding box contains an object.

$$Confidence\ Score = Pr(object) * IOU_{pred}^{truth} \tag{2.7.1}$$

Using the confidence score in YOLO, we can prevent the model from falsely detecting backgrounds by assigning a confidence score of zero when no object is present in the cell. Conversely, if an object is detected, the confidence score should reflect the Intersection over Union (IOU) between the predicted bounding box and the ground truth.

Although YOLO generates a total of $7 \times 7 \times 2 = 98$ boxes (two boxes per grid cell), most of these boxes will have very low confidence scores, making them less reliable. Hence, we can discard these boxes to focus on more confident predictions.
For each bounding box, the model produces four values (x, y, w, h) representing the box's position and dimensions. The (x, y) coordinates denote the box's center relative to the limits of the grid cell, while the width w and height h are calculated relative to the entire image.
The confidence prediction corresponds to the IOU between the predicted box and any ground truth box. Additionally, each grid cell forecasts conditional class probabilities $Pr(Class_i|Object)$, which are contingent upon the grid cell containing an object. Despite predicting multiple boxes ($B = 2$) per cell, the model only generates one set of class probabilities for the grid cell, multiplying these probabilities by the individual box confidence predictions during testing. This ensures the model's classification predictions are influenced by the confidence of the detected boxes.

$$Pr(Class_i|Object) \ * \ Pr(Object) \ * \ IOU_{pred}^{truth} \ = \ Pr(Class_i) \ * \ IOU_{pred}^{truth} \tag{2.7.2}$$

This provides us with class-specific confidence scores for each box. These scores encode both the probability of that class and the likelihood of that class appearing in the box and how well the predicted box fits the object.

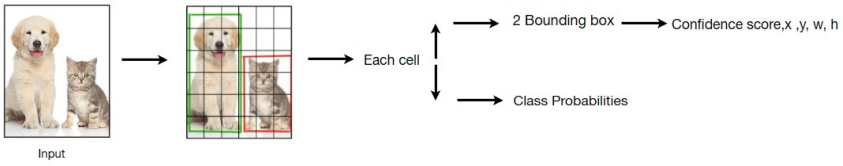


Figure 2.15: Output Shape

The YOLO predictions are organized and stored as a tensor of dimensions $S \times S \times (B \times 5 + \text{Classes})$. In this tensor, $S \times S$ represents the grid cells, B is the number of predicted bounding boxes per grid cell (e.g., $B = 2$), and Classes denotes the number of classes that the model can classify.
Each element of the tensor contains the necessary information for each grid cell. Specifically, for each grid cell, there are B bounding box predictions, each consisting of five values (x, y, w, h, and confidence score), along with the class probabilities for the different classes. The tensor effectively stores all the localization, confidence, and class information required for object detection in YOLO.

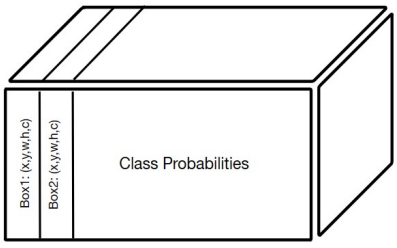


Figure 2.16: Output Tensor

Network Architecture

YOLO adopts a unified neural network that simultaneously predicts multiple bounding boxes and associated class probabilities. Inspired by the GoogleNet model for image classification, YOLO uses 1×1 reduction layers followed by 3×3 convolutional layers. The architecture comprises 24 convolutional layers, followed by two fully connected layers. As previously mentioned, the network's final output is the tensor $S \times S \times (B \times 5 + \text{Classes})$, containing predictions for bounding boxes and class probabilities.

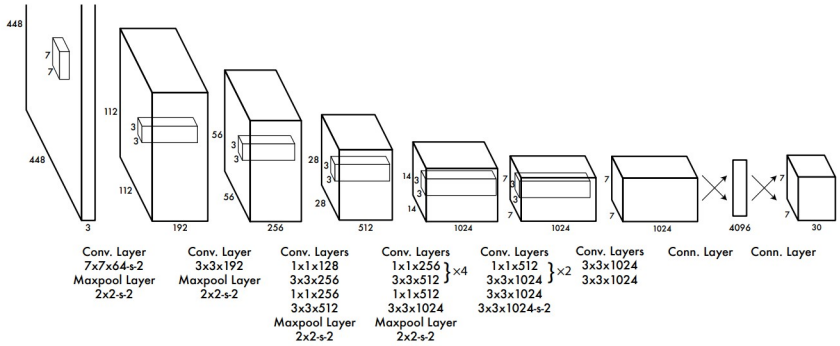


Figure 2.17: Yolov1 Architecture

In Figure 2.17, the original architecture of YOLOv1 trained on the PASCAL VOC detection dataset is shown. The PASCAL VOC detection dataset includes 20 classes, resulting in an output tensor of $7 \times 7 \times 30$. In the depicted architecture, the last convolutional layer's output ($7 \times 7 \times 1024$) is flattened into a 50176-dimensional feature vector. It is then passed through two fully connected layers to an output layer with 1470 feature vectors. This output is reshaped into $7 \times 7 \times 30$ feature maps, producing the required output tensor. The above architecture allows YOLO to efficiently handle object detection tasks, providing accurate predictions for bounding boxes and class probabilities in real-time.

Chapter 3

Cow Stall Numbers Detection Using YOLOv8

3.1 Introduction

In dairy farming, management involves several problems that might hinder effective operations. One challenge is the time-consuming nature of individual cow management. Closely monitoring each cow's health, behavior, and production in a big herd becomes time-consuming, and record-keeping may be complex and error-prone, especially in large-scale activities. Implementing a practical solution is critical, and cow stall numbers provide one. Farmers may simplify administration responsibilities by issuing unique stall numbers to each cow. Cow stall numbers make individual identification and tracking more manageable, allowing for more effective monitoring of health, behavior, and production performance. Furthermore, by giving a unique identification for each cow, they promote efficient record-keeping, boosting data accuracy and facilitating data retrieval and analysis. Cow stall numbers also help reproductive management by optimizing breeding programs, detecting heat cycles, and managing calving intervals. Finally, cow stall numbers aid industry compliance by assuring traceability and animal welfare regulations. Dairy farmers may overcome managerial issues, increase productivity, and drive the success of their dairy operations by using cow stall numbers.

Traditionally, establishing cow stall numbers has depended on manual procedures such as eye examination or manual data input. However, these systems are time-consuming, prone to mistakes, and difficult to adopt in large-scale dairy operations. Object detection technology eliminates these constraints by automating the process of finding stall numbers. Object detection algorithms may be trained to recognize and find cow stall numbers within photos or video footage collected in the barn or milking parlor. These systems can identify and retrieve stall numbers from visual input using techniques like deep learning and computer vision. The advantages of utilizing object detection to determine cow stall numbers are numerous. It drastically minimizes the amount of labor and time necessary for manual identification. Instead of manually checking each cow's stall number, the system can quickly analyze the visual data and automatically extract the essential information. This efficiency saves dairy farmers and their teams time, allowing them to focus on other crucial areas of herd management. Object detection also improves data accuracy and record-keeping. Automating the identification procedure reduces the odds of human mistakes in transcribing or recording stall numbers. The equipment can dependably gather and preserve accurate data, assuring the herd's records' integrity. This accuracy is required for various applications, including breeding programs, health management, milk production analysis, and industry compliance.

Deep learning-based object detection techniques are classified into two types: one-stage algorithms and two-stage algorithms. Unlike the well-known two-stage technique R-CNN [9] and its upgraded algorithms [8] [33], one-stage algorithms often compute the position and class probabili-

ties of objects immediately, significantly improving the model's inference speed. In addition to the conventional SSD model [27], YOLO series algorithm models provide only trade-offs of inference speed and accuracy for real-time applications among one-stage algorithms.

We train the YOLOv8 [11] algorithm for cow stall number detection using the CowStallNumber dataset [42] and compare the performance of YOLOv8 algorithm models for different input picture sizes and model sizes.

This chapter's contributions are summarised as follows:

- This work first applies YOLOv8 to the cow stall number detection task and demonstrates that YOLOv8 algorithm possesses better generalization by evaluating the prediction results of the model.

3.2 Related Work

Numerous scientific research on topics comparable to the one given in this chapter has been conducted. For example, Lichao Huang et al. [16] uses the Faster Region-based Convolutional Neural Network (Faster R-CNN) method for cow tail identification, which achieves higher accuracy and faster detection times than the usual R-CNN approach. Another famous example is Anagnostopoulos et al. [1], which uses infrared technology to determine an object's heat signature, allowing for more precise identification in low-light or low-contrast environments. The study of Zhang et al. [42] has the same goal as ours. They use ResNet34 as a backbone, with two extra output layers dedicated to stall number categorization and regression for bounding box coordinates. The experimental result obtains a 92% accuracy in stall number identification and a 40.1% IoU score in stall number position prediction. Lastly, Radek Jan Holik [15] implemented transfer learning using the pre-trained ResNet101 model for cow stall number detection using the CowStallNumber dataset, and their experimental results yielded a 91.91% accuracy in stall number recognition and an 18.50% Intersection over Union (IoU) score for predicting stall number positions.

Despite previous research efforts, there remains a gap in the literature regarding using the YOLOv8 model for stall number detection. Motivated by this gap and the possible benefits of the YOLOv8 model in terms of speed and accuracy, this work intends to investigate the YOLOv8 model's performance in cow stall number detection. We anticipate improved efficiency and accuracy in stall number recognition and localization by using the characteristics of the YOLOv8 architecture, contributing to improving dairy farming management practices.

3.3 Problem Approch

This section will discuss the model training method, validation and testing on the dataset, our model's architecture, and the data augmentation technique used during training. Figure 3.1 depicts a flowchart of the model training and performance evaluation procedure. We randomly split the CowStallNumbers dataset's 1303 images into the training, validation, and test sets, with the training set increased via data augmentation from the original images.

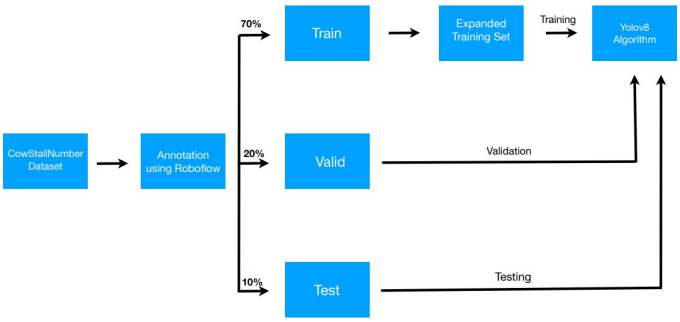


Figure 3.1: Flowchart of problem approch

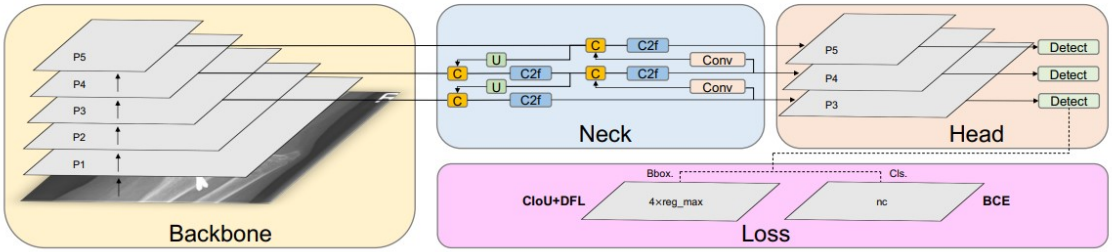


Figure 3.2: Model architecture of the YOLOv8 algorithm. .

3.3.1 Data Augmentation

Effectively using data augmentation procedures during model training is critical for dataset expansion and improved model performance. This study used numerous data augmentation approaches to diversify the dataset and improve its generalization skills. We used the horizontal flip technique to enhance the dataset, a 90-degree rotation in clockwise and anti-clockwise orientations, and bounding box adjustments such as crop (0% minimum zoom, 20% maximum zoom) and rotation (-15° to +15°).
We used the Roboflow platform's features to simplify the deployment of various data augmentation strategies. This platform featured an easy-to-use interface and tools for performing these modifications to the dataset. We were able to effectively increase the dataset by utilizing Roboflow's resources, which improved the model's capacity to generalize and excel in object identification tasks.

3.3.2 Model Architecture

Figure 3.3 depicts YOLOv8 model architecture, comprising the Backbone, Neck, and Head. The design principles of each aspect of the model architecture, as well as the modules of distinct portions, are introduced in the following subsections.

Backbone

The YOLOv8 model's backbone is based on the Cross Stage Partial (CSP) architecture proposed by Wang et al. in their work [40]. The CSP architecture divides the feature map into two halves. The first part applies convolution procedures, while the second part concatenates with the output of the preceding section, resulting in increased CNN learning ability while reducing the model's computational cost.
YOLOv8, as introduced by Glenn et al. in their paper [11], incorporates the C2f module, which combines the C3 module with the ELAN idea from YOLOv7 [39]. This combination allows the

model to gather deeper gradient flow information. The C3 module in YOLOv8 comprises three ConvModules and n DarknetBottleNeck, while the C2f module consists of two ConvModules and n DarknetBottleNeck connected by Split and Concat. The ConvModule is composed of Conv-BN-SiLU layers, and n represents the number of bottlenecks.
In contrast to YOLOv5 [10], YOLOv8 uses the C2f module instead of the C3 module, which contributes to its improved performance.Moreover, YOLOv8 further reduces computational cost by lowering the number of blocks in each step. Specifically, in Stages 1 to 4, the number of blocks is decreased to 3, 6, 6, and 3, respectively. To enhance the model's inference performance, YOLOv8 incorporates the Spatial Pyramid Pooling - Fast (SPPF) module in Stage 4, an upgrade on Spatial Pyramid Pooling (SPP) [13]. These architectural changes result in a YOLOv8 model with greater learning ability and reduced inference time.
In summary, YOLOv8 builds upon the CSP architecture, introduces the C2f module, reduces the number of blocks in each stage, and incorporates SPPF to enhance its performance.

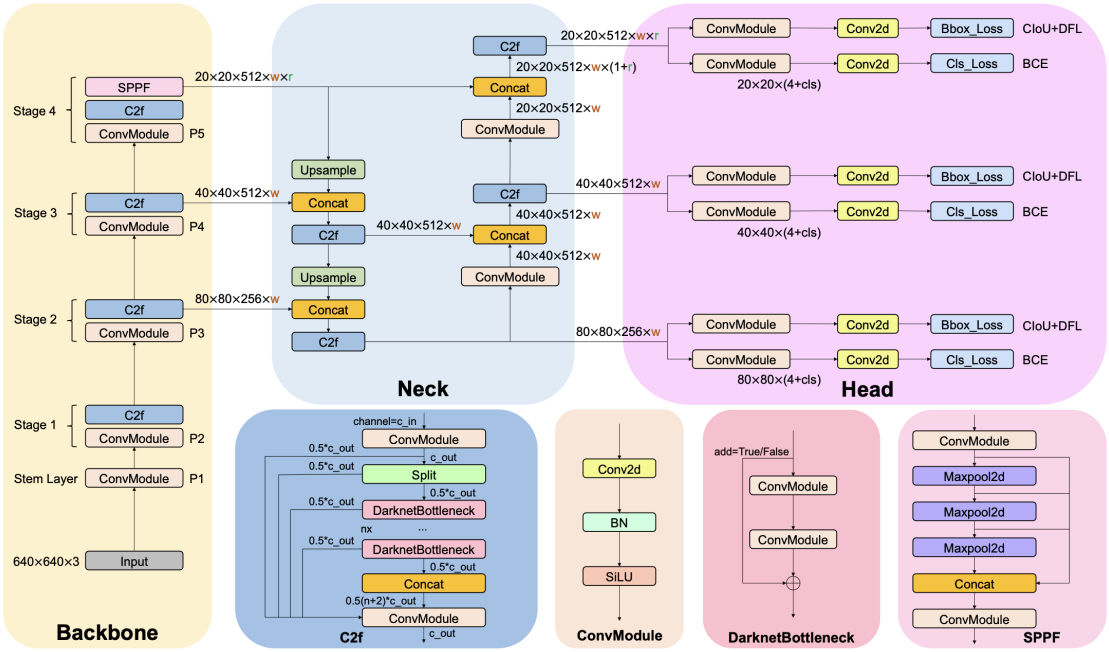


Figure 3.3: Detailed illustration of YOLOv8 architecture

Neck

Deeper neural networks generally capture more comprehensive feature information, resulting in denser predictions. However, excessively deep networks may lose object position information, and extensive convolution processes can lead to information loss, especially for small objects. To address this, the YOLOv8 model architecture incorporates multi-scale feature fusion using the Feature Pyramid Network (FPN) [25] and Path Aggregation Network (PAN) [26] concepts. Deeper neural networks can capture more comprehensive feature information, leading to more detailed predictions. However, overly deep networks may lose object position details, and extensive convolution can cause information loss, especially for small objects. To address this, the YOLOv8 architecture uses multi-scale feature fusion through the Feature Pyramid Network (FPN) and Path Aggregation Network (PAN) to combine information from different network levels. The higher levels have more information from additional layers, while the lower levels retain location details with fewer convolutions. Inspired by YOLOv5, FPN upsamples top to bottom to enrich lower level features. Conversely, PAN downsamples bottom to top to add information to higher levels. This FPN and PAN combination ensures accurate predictions for varying image sizes. YOLOv8 uses FP-PAN to reduce computation

costs while maintaining multi-scale fusion benefits. By incorporating these advanced techniques, YOLOv8 achieves better performance and efficiency for object detection.

Head

In contrast to the YOLOv5 model, which utilizes a coupled head, our YOLOv8 model adopts a decoupled head [7], where the classification and detection heads are separated. As depicted in Figure 3.3, the YOLOv8 model eliminates the objectness branch while retaining the classification and regression branches. While the Anchor-Based approach employs multiple predefined anchors in the image to compute four offsets for regressing the object's location from the anchors, our YOLOv8 model uses an Anchor-Free [38] technique. In the Anchor-Free approach, the model locates the object's center and calculates the distance between the center and the bounding box. This Anchor-Free mechanism allows for more flexible and precise object localization by directly predicting the bounding box based on the center point. By incorporating a decoupled head and adopting the Anchor-Free approach, YOLOv8 achieves improved object detection performance and more efficient training compared to its predecessors.

3.4 Experiments

3.4.1 Dataset

Zhang et al. [42] introduced the CowStallNumbers dataset, which aimed to enhance the identification of cow stall numbers. This dataset was derived from cow teat recordings and consisted of a total of 1,303 images, with 1,042 images allocated for training and 261 images for testing. The stall numbers within the dataset ranged from 0 to 60. To conduct the experiments, the dataset was randomly divided into three subsets: a training set, a validation set, and a test set. The training set encompassed approximately 70% of the original dataset, comprising 913 images, while the validation set included 260 images, accounting for approximately 19.9% of the dataset. The remaining 10% of the images, amounting to 130 images, were allocated to the test set. In order to augment the dataset and introduce further variations, various techniques were applied, including horizontal flipping, 90-degree rotations in clockwise and anti-clockwise orientations, and bounding box adjustments involving crop operations (with 0% minimum zoom and 20% maximum zoom) as well as rotation within the range of -15° to +15°.

3.4.2 Experiment setup

We utilized pre-trained YOLOv8 models on the COCO val2017 dataset for our training. Based on experimental data from Ultralytics [10] [11], YOLOv5 was trained for 300 epochs, while we extended the training duration to 500 epochs for YOLOv8. However, since we used pre-trained models, we initially set the total number of epochs to 250 and the patience to 50. The patience value determines that the training process will terminate if there is no noticeable progress after 50 epochs. During the training of YOLOv8m, we noticed that the model performance peaked at epoch 131, and the training process terminated at epoch 161. To save computational resources, we reduced the number of epochs for all model training to 170.

According to Glenn [11], the Adam [21] optimizer is more suitable for training small custom datasets and model hyperparameters, while the SGD [35] optimizer performs better on large datasets. Therefore, we trained the YOLOv8 models individually using both the Adam and SGD optimizers to compare their effects on model training. For training with the SGD optimizer, we used a starting learning rate of 1×10^{-2}, and for training with the Adam optimizer, the starting learning rate was set to 1×10^{-3}. The outcomes of this comparison are presented in Table 3.1.

Table 3.1: Comparison of model performance of YOLOv8s and YOLOv8m trained on the CowStall-Number dataset using SGD and Adam optimizers.

Model	**size (pixels)**	**Optimizer**	**Best Epoch**	**mAP^{val} 50**	**mAP^{val} 50-95**	**speed GPU A4000 (ms)**
YOLOv8s	640	SGD	104	0.972	0.766	8.5
YOLOv8s	640	Adam	117	0.949	0.769	10.2
YOLOv8s	1024	SGD	88	0.976	0.772	12.4
YOLOv8s	1024	Adam	147	0.952	0.766	11.1
YOLOv8m	640	SGD	40	0.975	0.787	10.4
YOLOv8m	640	Adam	131	0.966	0.783	10.6
YOLOv8m	1024	SGD	37	0.979	0.782	17.2
YOLOv8m	1024	Adam	114	0.975	0.778	17.2

In Section 3.4.3, for each experiment trial, we employed SGD [35] as the optimizer with a weight decay of 5×10^{-4} and a starting learning rate of 1×10^{-2}. We conducted training using input image sizes of 640 and 1024 and trained several models on an NVIDIA RTX A4000 16GB GPU. For image sizes of 640, we selected a batch size of 16, and for image sizes of 1024, we chose a batch size of 8. The models were trained using Python 3.11 and PyTorch 2.0.1. We recommend that readers conduct their training using Python 3.10 or higher and PyTorch 1.7 or higher.

3.4.3 Experiment Results

In this section, we aim to assess the practicality of YOLOv8 models for cow stall number identification in real-world dairy farming scenarios by comparing their mean average accuracy (mAP 50) and inference time.

To eliminate the potential impact of optimizer choice on model performance during training, we compare models trained using two different optimizers: SGD [35] as proposed by Glenn [11], and Adam [21], which is another popular optimizer. As shown in Table 3.1, training the model with the SGD optimizer leads to lower weight changes. The YOLOv8m model trained with the SGD optimizer achieves its best performance at the 37th epoch with an input picture size of 1024, while the model trained with the Adam optimizer reaches its highest performance at the 114th epoch. For the YOLOv8m model with an input picture size of 1024, the mAP 50 obtained with the SGD optimizer is 0.979. Additionally, the variation in inference time for models trained with different optimizers is less than 0.9ms.

Table 3.2: Comparison of model performance of different sizes of YOLOv8.

Model	**size (pixels)**	**mAP^{val} 50**	**mAP^{val} 50-95**	**speed CPU ONNX (ms)**	**speed GPU A4000 (ms)**	**params (M)**
YOLOv8n	640	0.944	0.757	82.1	8.5	3.0
YOLOv8s	640	0.972	0.766	201.3	10.2	11.1
YOLOv8m	640	0.975	0.787	390.4	12.4	25.8
YOLOv8l	640	0.976	0.783	654.5	11.1	43.6
YOLOv8n	1024	0.974	0.776	208.4	10.4	3.0
YOLOv8s	1024	0.976	0.772	483.3	10.6	11.1
YOLOv8m	1024	0.979	0.782	993.9	17.2	25.8
YOLOv8l	1024	0.976	0.774	1828.2	17.2	43.6

3.5 Conclusion

This chapter used the deep learning-based object identification algorithm YOLOv8 to recognize cow stall numbers. Cow stall numbers help manage, monitor, and record dairy cows. Object recognition technology for cow stall number identification minimizes human work, increases data accuracy, and allows real-time applications. We presented the Backbone, Neck, and Head components of the YOLOv8 algorithm for cow stall number identification. To improve generalization, we extended the CowStallNumbers dataset. The YOLOv8 model detected cow stall numbers in our experiments. The YOLOv8m model with an input size of 1024 pixels has the most excellent mAP (mean average precision) of 0.979 for IoU 0.5 and 0.782 for IoU 0.5 to 0.95, delivering exceptional accuracy and localization. We also tested the model using SGD and Adam optimizers. SGD outperformed Adam for the YOLOv8m model with a 1024-pixel input. In conclusion, YOLOv8 successfully detected cow stall numbers, giving dairy producers an effective and precise option for individual cow management. The SGD optimizer-trained YOLOv8m model with a 1024-pixel input size was best for this assignment. The YOLOv8 model's remarkable performance promises to alter dairy farming management practices, improve production, and ensure industry compliance. Future studies might optimize model hyperparameters, explore transfer learning, and apply YOLOv8 to additional dairy farming elements.

Chapter 4

YOLOv7 vs. YOLOv8: Comparative Evaluation of Object Detection Algorithms

4.1 Introduction

Object detection is critical in real-world contexts due to its vast range of applications and advantages. It allows various automated systems to see and comprehend their environment by accurately recognizing and localizing things within images or video streams. This capacity is required for autonomous driving, surveillance, robotics, and augmented reality. Traditional approaches, such as Single Shot Detector (SSD) [27], Region-based Convolutional Neural Networks (R-CNN) [27], and Fast Region-based Convolutional Neural Networks (Fast R-CNN) [8] , have made significant progress in this field by merging handmade characteristics with machine learning algorithms. However, the development of deep learning, particularly with game-changing algorithms such as You Only Look Once (YOLO) [30], has transformed object detection. YOLO, announced in 2015, reframed object recognition as a single regression issue, allowing it to directly estimate bounding box coordinates and class probabilities from picture pixels. This all-encompassing strategy has significantly increased the object-detecting system's accuracy, speed, and efficiency. Deep learning algorithms, such as YOLO and its following iterations, have performed extraordinarily in complicated real-world circumstances, making them essential tools for various applications.

The YOLO approach uses a unified model to forecast multiple bounding boxes and their associated class probabilities for all objects in an image simultaneously. This novel technique provides exceptional speed and accuracy, distinguishing YOLO from other algorithms at the time of its debut. YOLO displayed higher performance in terms of detecting and identifying object coordinates by speeding the object detection process [30]. This object detection algorithm has found applications in various fields, including wildlife monitoring, drones, military, autonomous driving, healthcare, and other computer vision tasks. Also, it has witnessed major developments throughout the years, resulting in several incarnations such as YOLOv1, YOLOv2, YOLOv3, YOLOv4, YOLOv5, YOLOv6, YOLOv7 and YOLOv8. Each iteration brought advancements in accuracy, speed, and feature additions.

Given these developments and the necessity for comparison research, this chapter focuses on comparing the performance of the two latest YOLO versions: YOLOv7 and YOLOv8. By comparing these two versions, we want to obtain insight into their respective strengths and limitations, supporting researchers and practitioners in picking the best YOLO model for their unique detection of object tasks.

This chapter's contributions:

- This study aims to perform a comparative analysis of YOLOv7 and YOLOv8, providing valuable insights into their strengths and limitations. The ultimate objective is to assist in selecting the optimal model for object detection tasks.

4.2 Related Work

The YOLO series of algorithms have been developed explicitly for practical applications, focusing on enhancing model inference speed while preserving model accuracy in real-world scenarios. The YOLOv1 model [30] utilized an input image size of $448 \times 448 \times 3$ and employed GoogLeNet for feature map extraction. The extracted feature maps were subsequently fed into two fully connected layers to generate the final output. The YOLOv1 model employed direct regression, resulting in a significant reduction in computational requirements and an enhancement in model inference speed. The YOLOv2 model, as reported in the literature [31], employed DarkNet-19 as its underlying network architecture. A passthrough layer was also incorporated to facilitate the fusion of high and low-level semantic information. This integration enhanced the model's efficacy in detecting objects of smaller sizes. The YOLOv3 model, as proposed in the literature [32], introduced a revised backbone network known as DarkNet-53. Additionally, the model incorporated the notion of Feature Pyramid Networks (FPN) [25] to facilitate object detection by utilizing multi-scale feature maps. The CSPDarkNet-53 model was proposed in YOLOv4 [2], which incorporated the Cross Stage Partial (CSP) [40] and DarkNet-53 to enhance the gradient fusion information of the model. Furthermore, YOLOv4 incorporated Spatial Pyramid Pooling (SPP) [13] to increase the receptive field and integrated Path Aggregation Network (PAN) [26] into the Neck of the architecture, resulting in a reduction in computational complexity. The YOLOv5 model, as described in reference [10], incorporated the Spatial Pyramid Pooling - Fast (SPPF) technique [20] as a replacement for the SPP architecture. Additionally, the model was enhanced by including the Focus module, resulting in improved performance. The YOLOv6 model, as described in the literature [24], incorporates the EfficientRep network [41] as its backbone and introduces the Rep-PAN architecture, which is based on the PAN [26] and Rep [4] models. This architecture is integrated into the Neck component of the YOLOv6 model. The YOLOv6 algorithm architecture developed by Wang et al. [24] was influenced by YOLOX [7], leading to decoupling operations for the Head. The YOLOv7 model, as proposed in reference [39], introduced a novel planned re-parameterized convolution module and the notion of model scaling and expansion. These innovations were designed to enhance the model's parameter utilization efficiency. YOLOv8 [11] is based on the CSPDarknet-53 backbone network, which is a modified version of the Darknet-53 network. The CSPDarknet-53 network includes several improvements that make it faster and more accurate than the Darknet-53 network.

4.3 Problem Approch

A systematic methodology is used to compare the performance of YOLOv7 and YOLOv8 in object detection, which comprises a random selection of datasets from the Roboflow100 [3] dataset. Ten small datasets, three medium datasets, and two large datasets were selected. The datasets chosen include various situations and object types, allowing us to evaluate the model's performance in various circumstances. Each dataset is used to train YOLOv7 and YOLOv8 independently. Comparison evaluation metrics included accuracy, precision, recall, and mAP@50. This section will also go into the architectural components of YOLOv7 and YOLOv8. For a visual representation of the methodology, refer to Figure 4.1, which presents a flow chart outlining the step-by-step approach.

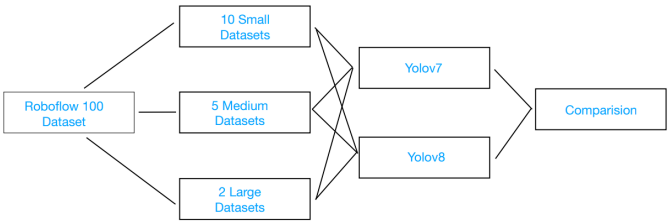


Figure 4.1: Flowchart of problem approch

4.3.1 Model Architecture

This section will explore the architectural details of the two models being compared:

YOLOv7

The YOLOv7 architecture can be divided into three main sections: input, backbone, and head. In contrast to YOLOv5, the neck and head layers are referred to as the head layers in YOLOv7. Each module serves the same purpose as in YOLOv5. The backbone is responsible for feature extraction, whereas the head is used for prediction. The first step in preparing input images is to align them into a fixed size of 640×640 in RGB format. These aligned images are then sent into the backbone network, which generates three distinct sizes of features that are fed into the head layer for feature map output. Following the RepVGG block and convolution processes, the three image detection tasks, namely classification, background classification, and border prediction, are carried out. Finally, the model produces the final output, which includes the predicted bounding boxes and their associated class probabilities.

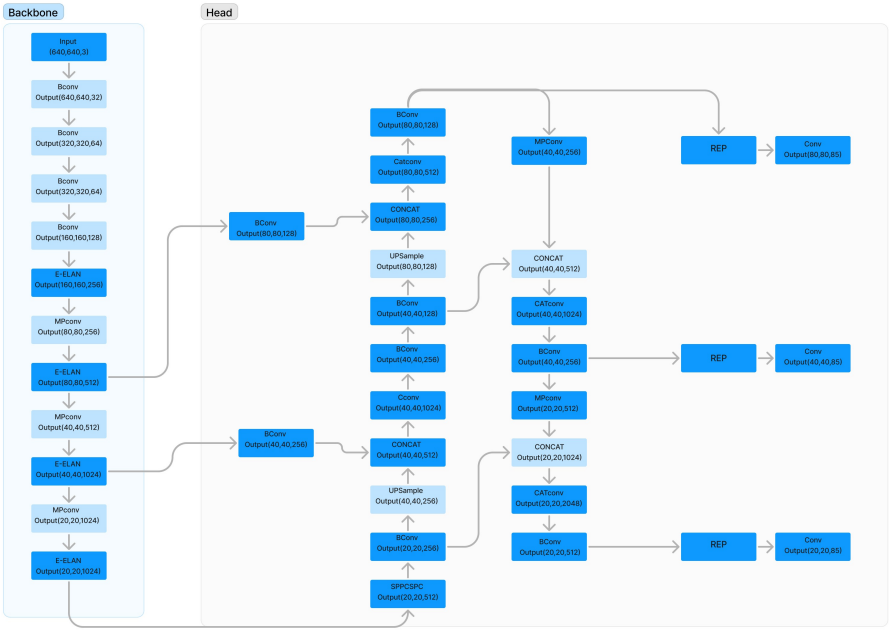


Figure 4.2: YOLOv7 Architecture

Overall, YOLOv7 is similar to YOLOv5 in terms of its general architecture and approach. However, YOLOv7 introduces specific improvements and modifications, such as the replacement of internal network components, the incorporation of an auxiliary training head, and advancements in

label allocation concepts, among others. YOLOv7 implements the YOLOv5 pre-processing concept, focusing on training and testing on moderately large images, such as 640×640 and 1280×1280. It incorporates various data augmentation techniques, including mosaic data enhancement, adaptive anchor frame computation, and adaptive image scaling. YOLOv7's backbone layer is composed of multiple BConv layers, E-ELAN layers, and MPConv layers. The BConv layer is made up of a convolutional layer, a Batch Normalization (BN) layer, and an activation function called LeakyReLU. These components work together to enhance the feature extraction process. Overall, YOLOv7 represents an evolution and improvement over YOLOv5, incorporating various enhancements to achieve better performance in object detection tasks.

E-ELAN Structure

In YOLOv7, the E-ELAN layer is based on the ELAN structure, which leverages expand, shuffle, and merge cardinality to enhance the network's ability to learn without disrupting the current gradient flow. The ELAN structure is depicted in Figure 4.3, where different colors of Bconv represent distinct convolution kernels, with k representing kernel length and breadth size, s representing stride, o representing outchannel, and i representing inchannel. There are three types of convolutions: the point convolution kernel convolution ($k = 1, s = 1$), which keeps the length and breadth of the input and output constant; the convolution kernel's convolution ($k = 3, s = 1$), which maintains constant output length and breadth; and if $s = 2$, the output's length and breadth are halved compared to the input's.

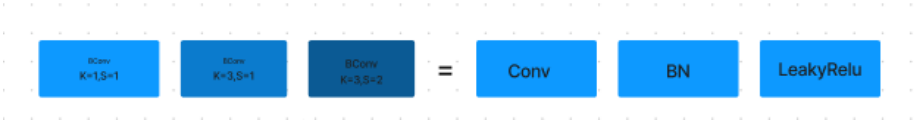


Figure 4.3: ELAN Layer

Similarly, the E-ELAN layer, as illustrated in Figure 4.4, is constructed from various convolutions while maintaining the same length and breadth for both its input and output. When $o = i$, it signifies that the number of output channels is equal to the number of input channels, ensuring no change in channel dimensions. On the other hand, $o \neq i$ indicates no direct correlation between the number of output channels and the number of input channels, although their values may not be unequal. The widths and lengths of both imout and output remain the same, ensuring consistency in spatial dimensions.

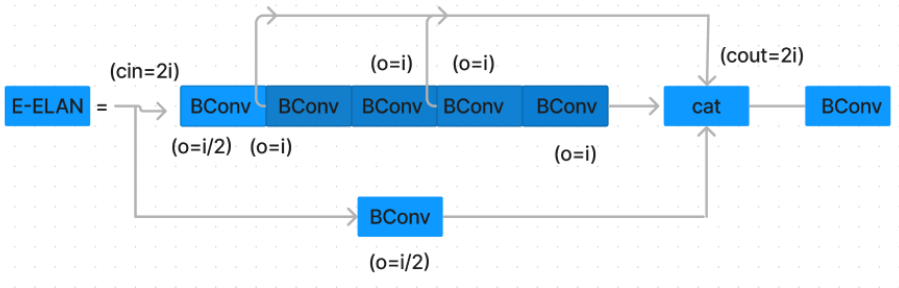


Figure 4.4: E-ELAN Layer

Furthermore, regarding the channel aspect, when $o = 2i$, it means that the number of output channels (o) is twice the number of input channels (i). This results in concatenating and splicing the output with the output channels of four convolution layers, each having a channel size of $i/2$.

The E-ELAN layer's unique design enables it to effectively handle deep network learning and convergence, while preserving spatial dimensions and effectively adjusting channel dimensions. This makes it a valuable component in YOLOv7's architecture.

MPConv structure

In earlier versions of YOLO, downsampling was achieved using max pooling followed by a 3×3 convolution with a stride of 2. However, in YOLOv7, both approaches are utilized: max pooling and 3×3 convolution with stride 2. Importantly, the number of channels in the backbone before and after the max pooling operation remains constant. The MPConv structure is depicted in Figure 4.5, and it is a key component of YOLOv7's backbone.

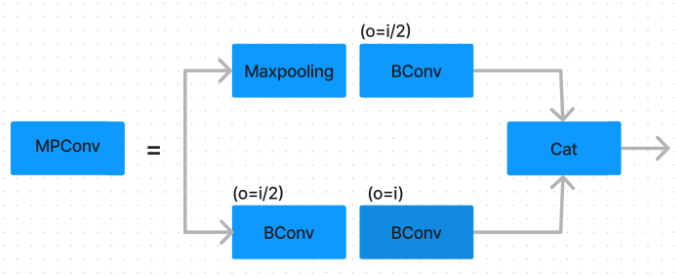


Figure 4.5: The MPConv structure

The MPConv layer has the same number of input and output channels, and its output length and width are half of the input length and width. The top branch achieves this by halving the length and breadth through max pooling, while the BConv layer cuts the number of channels in half. On the other hand, the lower branch, through the first BConv layer, halves the number of channels, while the second BConv layer with $(k = 3, s = 2)$ operation reduces the width and length by half. The outputs from the upper and lower branches are then concatenated to produce the final output with length and width halved.

In the YOLOv7 backbone, multiple BConv, E-ELAN, and MPConv layers are combined to effectively extract features from the input data.

Head

The detection head in YOLOv7 follows a similar structure to that of YOLOv5, using an anchor-based approach instead of the decoupling head concept seen in YOLOX and YOLOv6. The Head layer in YOLOv7 consists of various components, including the SPPCPC layer, multiple BConv layers, several MPConv layers, multiple Catconv layers, and finally, the output of three RepVGG blocks, forming the three Head layers. The overall structure of the Head layer is illustrated in Figure 4.2.

SPPCSPC Layer

The output channel of the SPPCSPC layer is determined by the "out-channel" parameter. Additionally, a hidden-channel value is computed as $int(2 \times e \times$ out-channel), which is used to extend the hidden-channel (referred to as "hc" hereafter). Typically, a value of $e = 0.5$ and hc = out-channel are used. Figure 4.6 provides a visual representation of the specific flowchart of the SPPCPC layer.

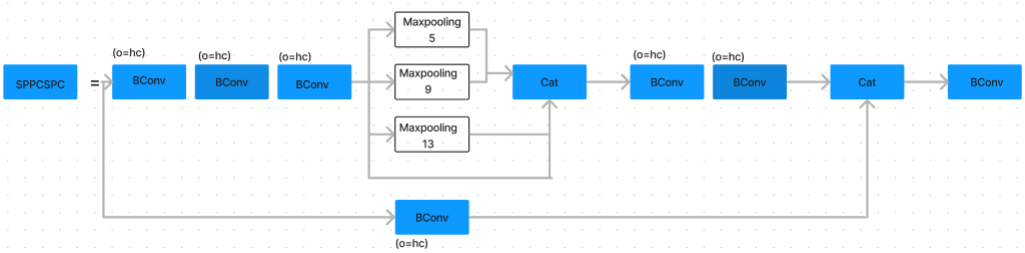


Figure 4.6: SPPCSPC Layer

Catconv Layer

The Catconv layer operates similarly to the E-ELAN layer. Figure 4.7 illustrates the specific flowchart of the Catconv layer. It should be noted that the lengths and widths of the Catconv layer's input and output remain unchanged. However, on the channel dimension, the output is obtained by concatenating six Conv-layer output channels as $i/2$, where i represents the input channel size.

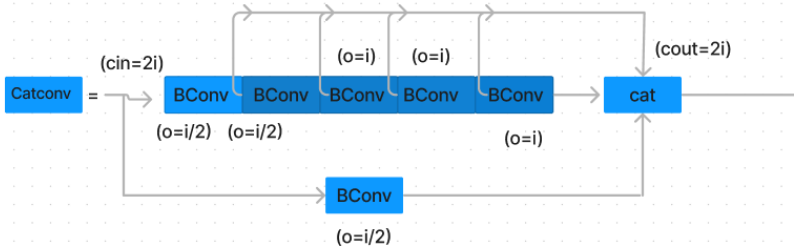


Figure 4.7: Catconv Layer

REP Layer

The REP layer, specifically the repvgg_block layer, is a network layer designed for easy deployment. Yolov6 also utilizes this layer. The structure of the REP layer is illustrated in Fig 4.8. During both the training and deployment phases, the REP layer has different structures. In the training phase, it consists of a 3×3 convolution and a 1×1 branch. Additionally, a BN (Batch Normalization) branch is added if the input and output channels, as well as the height and width (h and w) sizes, remain constant. The three branches are then combined to produce the final output. During the deployment phase, the branch parameters are re-parameterized to the main branch, and the 3×3 main branch convolution output assists in efficient deployment.

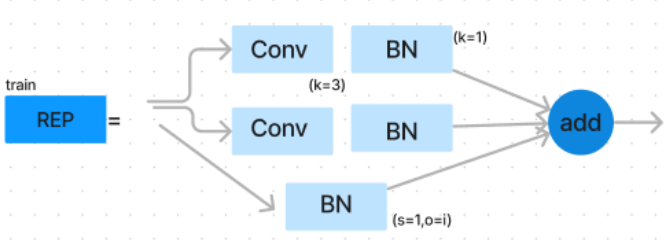


Figure 4.8: REP Layer

DownC Layer And ReOrg Layer

In Yolov7, there are other network designs that include DownC and ReOrg layers. The DownC layer is similar to the MPConv layer, and the ReOrg layer follows the same slicing principle as in Yolov5. The DownC layer structure is shown in Fig 4.9.

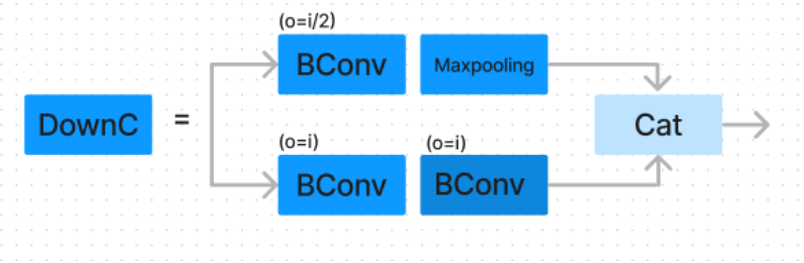


Figure 4.9: DownC Layer And ReOrg Layer

The architecture of YOLOv8 is as similar, as mentioned in the section 3.3.2

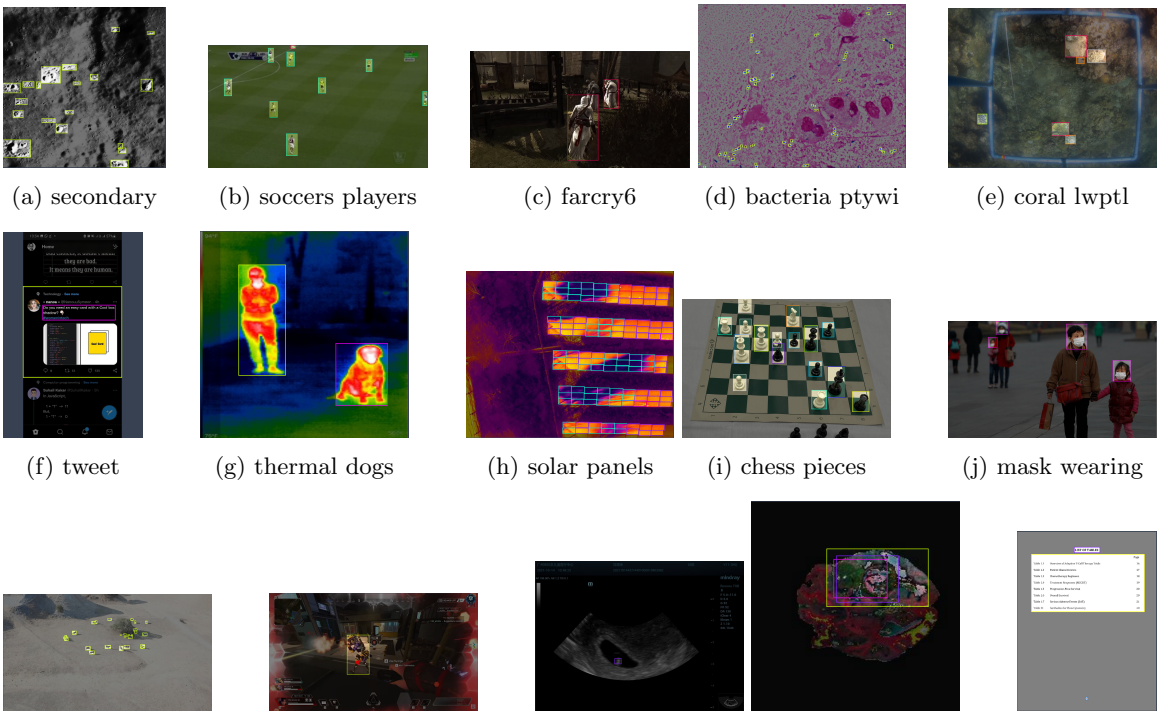
4.4 Experiments

4.4.1 Dataset

Ciaglia et al., have recently introduced the Roboflow-100(RF100) [3], a comprehensive dataset comprising 100 datasets, spanning across seven distinct imagery domains. RF100 encompasses a staggering collection of 224,714 images and covers a wide range of 805 class labels. The construction of this dataset involved over 11,170 hours of meticulous labeling by computer vision practitioners within the Roboflow Universe platform. RF100 was carefully curated from over 90,000 publicly available datasets and 60 million publicly accessible images. The objective behind the release of RF100 is to provide researchers with a semantically diverse and multidomain benchmark that enables them to assess the generalizability of their models using real-life data. We randomly selected a subset of ten small datasets, three medium datasets, and two large datasets for our experiments. Detailed information about these datasets can be found in the accompanying table 4.1.

Table 4.1: Dataset Details

Small Datasets					
Dataset	**Category**	**Train**	**Test**	**Valid**	**Classes**
secondary chains	aerial	103	16	43	1
soccers players	aerial	114	16	33	3
farcry6 videogame	videogames	82	14	24	11
bacteria ptywi	microscopic	30	10	30	1
coral lwptl	underwater	427	74	93	14
tweet posts	documents	87	9	21	2
thermal dogs	electromagnetic	142	20	41	2
solar panels	electromagnetic	112	19	30	5
chess pieces	real world	202	29	58	13
mask wearing	real world	105	15	29	2
Medium Datasets					
aerial cows	aerial	1084	299	340	1
apex videogame	videogames	2583	415	691	2
gynecology mri	electromagnetic	2122	253	526	3
Large Datasets					
brain tumor	electromagnetic	6930	990	1980	3
paper parts	documents	8472	1209	2359	46



(a) secondary (b) soccers players (c) farcry6 (d) bacteria ptywi (e) coral lwptl

(f) tweet (g) thermal dogs (h) solar panels (i) chess pieces (j) mask wearing

(k) aerial cows (l) apex videogame (m) gynecology mri (n) brain tumor (o) paper

Figure 4.10: Grid of Images

4.4.2 Experiment Setup

Our study compared the performance of the YOLOv7 and YOLOv8 models, which were initially trained on the COCO val2017 dataset [26]. To train these models on our selected datasets, we referred to the experimental data published by Ultralytics [10] [11]. YOLOv7 was trained for 200 epochs, while YOLOv8 was trained for 500 epochs in their initial training. Since we used pre-trained models, we conducted our training for 50 epochs for each model. We adjusted the input picture size to 640 to ensure compatibility with the pre-trained models. The training process was carried out using NVIDIA RTX A4000 16GB GPUs. For picture sizes of 6470, we chose a batch size of 16. The training was implemented using Python 3.11 and PyTorch 2.0.1. We recommend that readers train with Python 3.10 or higher and PyTorch 1.7 or higher versions for optimal results.

4.4.3 Experiment Results

The provided table 4.2 contains the performance comparison of YOLOv7 and YOLOv8 on different datasets. Here are the insights drawn from the data:

Table 4.2: Performance Comparison

Dataset	Precision		Recall		mAP@50		mAP@50-90		Speed (ms)	
	v8	v7	v8	v7	v8	v7	v8	v7	v8	v7
Small Datasets										
secondary chains	**0.467**	0.245	**0.438**	0.411	**0.383**	0.199	**0.206**	0.0689	**33.3**	78.5
soccers players	**0.952**	0.42	**0.937**	0.735	**0.951**	0.477	**0.555**	0.205	**26.3**	52.6
farcry6 videogame	0.631	**0.645**	0.434	**0.507**	**0.393**	0.263	**0.276**	0.156	**37.9**	95.5
bacteria ptywi	**0.569**	0.181	**0.638**	0.0232	**0.539**	0.00562	**0.253**	0.00078	**41.2**	111.4
coral lwptl	0.285	**0.331**	0.26	**0.307**	0.198	**0.224**	0.139	**0.153**	**40.1**	45
tweet posts	**0.916**	0.846	0.997	**1**	**0.988**	0.949	**0.988**	0.556	314	**116.1**
thermal dogs	**0.985**	0.795	**0.953**	0.753	**0.96**	0.829	**0.806**	0.658	**38.4**	83.9
solar panels	**0.804**	0.505	**0.755**	0.382	**0.815**	0.306	**0.63**	0.165	**121.5**	84.3
chess pieces	**0.97**	0.748	**0.988**	0.648	**0.976**	0.778	**0.832**	0.586	114.2	**81.9**
mask wearing	**0.995**	0.942	**0.97**	0.647	**0.994**	0.715	**0.865**	0.403	**24.8**	76
Medium Datasets										
aerial cows	**0.831**	0.288	**0.704**	0.174	**0.781**	0.105	**0.43**	0.0275	**6.8**	21.8
apex videogame	**0.84**	0.762	**0.847**	0.738	**0.872**	0.755	**0.591**	0.460	**11.4**	20.8
gynecology mri	**0.287**	0.0676	**0.153**	0.136	**0.240**	0.0748	**0.112**	0.0182	**10.1**	21.2
Large Datasets										
brain tumor	**0.874**	0.828	**0.739**	0.673	**0.799**	0.702	**0.532**	0.386	**11.4**	18.3
paper parts	0.822	**0.833**	**0.845**	0.833	**0.852**	0.839	**0.637**	0.613	**10.8**	18.9

Precision and Recall: Across most datasets, YOLOv8 consistently outperforms YOLOv7 regarding precision and recall. Higher precision values indicate that YOLOv8 is better at accurately identifying relevant objects in the images. Similarly, better recall values suggest that YOLOv8 can detect a higher percentage of actual positive objects, making it more suitable for object detection tasks.

mAP@50 and mAP@50-90: YOLOv8 also shows superior mean average precision (mAP) values at IoU thresholds of 50% and 50-90%. This indicates that YOLOv8 achieves better accuracy in localizing and classifying objects at different intersection levels over union (IoU), further highlighting its improved performance compared to YOLOv7.

Speed: YOLOv8 demonstrates better speed performance (lower millisecond values) than YOLOv7 across most datasets. The lower inference times suggest that YOLOv8 is more efficient, making it more practical for real-time applications or scenarios with strict latency constraints.

Dataset Impact: The performance of both YOLOv7 and YOLOv8 varies across different datasets. While some datasets show better performance with YOLOv8, there are cases where YOLOv7 performs comparably or slightly better. This indicates that the choice between YOLOv7 and YOLOv8 should consider the dataset's specific characteristics and the application's requirements.

Dataset Size: Interestingly, the performance difference between YOLOv7 and YOLOv8 seems more pronounced on small and medium-sized datasets. The performance gap on larger datasets becomes relatively minor, suggesting that YOLOv8's advantages may be more apparent when data is limited.

Consideration for Application: When speed is a crucial factor, YOLOv8 is a better choice due to its improved speed and comparable or better accuracy. However, YOLOv7 might still be a viable option for specific datasets or use cases, mainly when it exhibits competitive performance and offers more straightforward implementation.

In conclusion, the provided performance comparison reveals that YOLOv8 generally outperforms

YOLOv7 in precision, recall, mAP, and speed, making it a preferable choice for most object detection tasks. Nevertheless, the choice of the YOLO version should be based on specific application requirements and dataset characteristics to achieve the best possible performance and efficiency.

4.5 Conclusion

This chapter compares YOLOv7 with YOLOv8. We wanted to understand their strengths and weaknesses to enable academics and practitioners to choose the best object detection model. We trained YOLOv7 and YOLOv8 models using a Roboflow-100 (RF100) dataset subset, including various pictures and object types. YOLOv8 outperformed YOLOv7 in most measures. At different IoU thresholds, YOLOv8 had greater accuracy, recall, and mAP. YOLOv8 is also detected faster for real-time applications. The CSPDarknet-53 backbone network and other enhancements made YOLOv8 more accurate and efficient than YOLOv7. This research helps practitioners choose a YOLO model for their object detection tasks by comparing YOLOv7 with YOLOv8. YOLOv7 or YOLOv8 would rely on the application, hardware resources, and accuracy-speed trade-off. As object detection advances, monitoring future YOLO iterations and assessing their effectiveness in different contexts would be helpful.

REFERENCES

[1] A Anagnostopoulos, M Barden, J Tulloch, K Williams, B Griffiths, C Bedford, M Rudd, Androniki Psifidi, G Banos, and Georgios Oikonomou. A study on the use of thermal imaging as a diagnostic tool for the detection of digital dermatitis in dairy cattle. *Journal of dairy science*, 104(9):10194–10202, 2021.

[2] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934*, 2020.

[3] Floriana Ciaglia, Francesco Saverio Zuppichini, Paul Guerrie, Mark McQuade, and Jacob Solawetz. Roboflow 100: A rich, multi-domain object detection benchmark. *arXiv preprint arXiv:2211.13523*, 2022.

[4] Xiaohan Ding, Xiangyu Zhang, Ningning Ma, Jungong Han, Guiguang Ding, and Jian Sun. Repvgg: Making vgg-style convnets great again. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 13733–13742, 2021.

[5] Sarah H Farghaly and Samar M Ismail. Floating-point fir-based convolution suitable for discrete wavelet transform implementation on fpga. In *2019 Novel Intelligent and Leading Emerging Sciences Conference (NILES)*, volume 1, pages 158–161. IEEE, 2019.

[6] Frederic B Fitch. Warren s. mcculloch and walter pitts. a logical calculus of the ideas immanent in nervous activity. bulletin of mathematical biophysics, vol. 5 (1943), pp. 115–133. *The Journal of Symbolic Logic*, 9(2):49–50, 1944.

[7] Zheng Ge, Songtao Liu, Feng Wang, Zeming Li, and Jian Sun. Yolox: Exceeding yolo series in 2021. *arXiv preprint arXiv:2107.08430*, 2021.

[8] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[9] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[10] J. Glenn. Ultralytics yolov5. https://github.com/ultralytics/yolov5, 2022.

[11] Joseph Glenn. Ultralytics yolov8. https://github.com/ultralytics/ultralytics, 2023.

[12] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE transactions on pattern analysis and machine intelligence*, 37(9):1904–1916, 2015.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[15] Radek Jan Holik. Detection and localization of stall numbers using deep cnn.

[16] Lichao Huang, Yi Yang, Yafeng Deng, and Yinan Yu. Densebox: Unifying landmark localization with end to end object detection. *arXiv preprint arXiv:1509.04874*, 2015.

[17] David H Hubel and Torsten N Wiesel. Receptive fields of single neurones in the cat's striate cortex. *The Journal of physiology*, 148(3):574, 1959.

[18] David H Hubel and Torsten N Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology*, 160(1):106, 1962.

[19] David H Hubel and Torsten N Wiesel. Receptive fields and functional architecture of monkey striate cortex. *The Journal of physiology*, 195(1):215–243, 1968.

[20] DH Kim and T MacKinnon. Artificial intelligence in fracture detection: transfer learning from deep convolutional neural networks. *Clinical radiology*, 73(5):439–445, 2018.

[21] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[22] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

[23] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[24] Chuyi Li, Lulu Li, Hongliang Jiang, Kaiheng Weng, Yifei Geng, Liang Li, Zaidan Ke, Qingyuan Li, Meng Cheng, Weiqiang Nie, et al. Yolov6: A single-stage object detection framework for industrial applications. *arXiv preprint arXiv:2209.02976*, 2022.

[25] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125, 2017.

[26] Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, and Jiaya Jia. Path aggregation network for instance segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8759–8768, 2018.

[27] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14*, pages 21–37. Springer, 2016.

[28] Jan Mycielski. Marvin minsky and seymour papert, perceptrons, an introduction to computational geometry. 1972.

[29] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[30] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[31] Joseph Redmon and Ali Farhadi. Yolo9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7263–7271, 2017.

[32] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[33] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28, 2015.

[34] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[35] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[36] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[37] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.

[38] Zhi Tian, Chunhua Shen, Hao Chen, and Tong He. Fcos: A simple and strong anchor-free object detector. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(4):1922–1933, 2020.

[39] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7464–7475, 2023.

[40] Chien-Yao Wang, Hong-Yuan Mark Liao, Yueh-Hua Wu, Ping-Yang Chen, Jun-Wei Hsieh, and I-Hau Yeh. Cspnet: A new backbone that can enhance learning capability of cnn. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*, pages 390–391, 2020.

[41] Kaiheng Weng, Xiangxiang Chu, Xiaoming Xu, Junshi Huang, and Xiaoming Wei. Efficientrep: An efficient repvgg-style convnets with hardware-aware neural network design. *arXiv preprint arXiv:2302.00386*, 2023.

[42] Youshan Zhang. Stall number detection of cow teats key frames. *arXiv preprint arXiv:2303.10444*, 2023.