VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY
UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

PROGRAMMING INTEGRATION PROJECT (CO3013)

Assignment (Semester: 241)

ROAD SURFACE SURVEILLANCE VEHICLES

Instructor: Assoc. prof. Thoai Nam, *HCMUT*
Doan Ngo Duc Phuong, *HCMUT*

Students: Nguyen Thien Loc - 2252460
Nguyen Thanh Minh - 2252481

HO CHI MINH CITY, JANUARY 2025

Contents

List of Figures

List of Tables

1 Introduction

1.1 Topic introduction

The issue of maintaining road surface damage plays a critical role in economic development and growth, as well-maintained roads facilitate efficient transportation, reduce travel costs, and boost trade and commerce. Moreover, it is also one of the major concerns globally due to its direct impact on road safety and the prevention of accidents. Poor road conditions can lead to vehicle damage, delays, and increased fuel consumption, further escalating economic losses. Therefore, road surface monitoring methods are continuously developed and improved to not only ensure the safety of road users but also to enhance the overall efficiency of transportation networks. Advanced technologies such as machine learning, automated detection systems are being integrated to provide more accurate, cost-effective, and timely solutions for road maintenance and management.

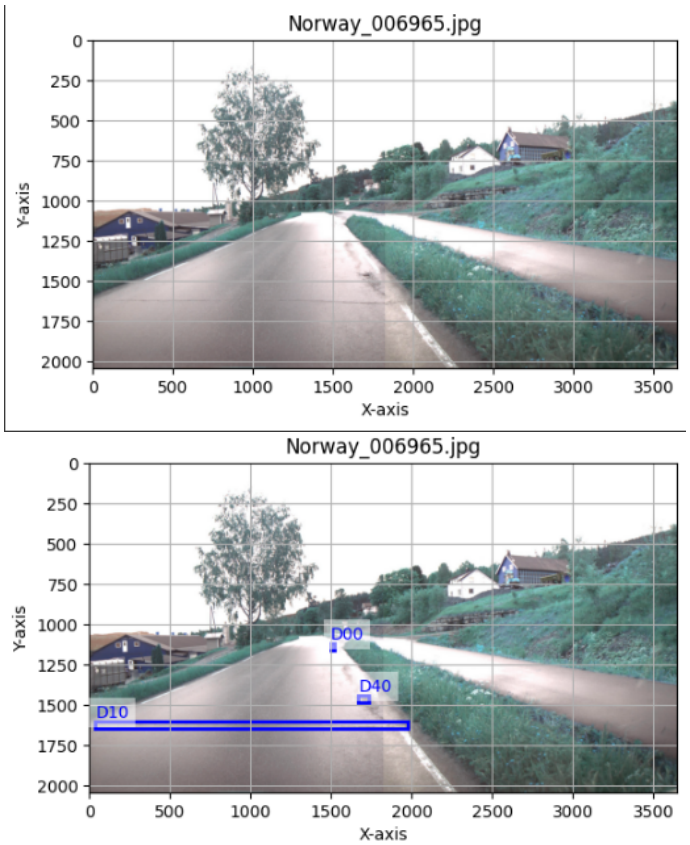


Figure 1: Examples of input and output of the project

1.2 Objectives of the topic

The causes of road accidents can generally be grouped into human error, road conditions, and vehicle conditions. However, the proportion of each factor can vary significantly between countries, as shown in Table 1 below, depending on factors such as the country's awareness of road safety, the enforcement of laws, and engineering aspects (including vehicle design and road infrastructure). Human error is the primary cause of road traffic accidents across all countries, followed by either road or vehicle conditions. Since human error is difficult to control, governments and transport authorities have placed greater emphasis on improving road conditions to enhance traffic safety. As a result, the following sections will focus on pavement surface conditions and their relationship to road traffic accidents.

The statistics of road accidents in the United States in 2006 show the number of accidents caused by road conditions out of the total number of accidents, as presented in Table 2.

Through the table, we can easily see the impact of road conditions on traffic accidents. Thus, recognizing and promptly repairing road surface damages will effectively prevent unnecessary accidents in both the present and the future.

Cause of Road Traffic Accident (%)				
Human error	**Road Condition**	**Vehicle Condition**	**Other factors**	**Country**
75.4	14.5	10.2	-	South Africa
85.5	2.9	5.1	6.4	Kenya
80	12.7	7.8	-	South Africa
90	4	8.4	-	Ghana
75.7	2.9	17.1	-	Congo
80.6	13.2	6.2	-	Malaysia
67	29	4	-	Saudi Arabia
29	64	7	-	Malaysia

Table 1: Cause of Road Traffic Accident

	No. of crashes	**%**	**No. of non-fatally injured people**	**%**	**No. of killed people**	**%**
All Crashes	16,954,351	100%	5,746,231	100%	42,642	100%
Road conditions	5,317,316	31.4%	2,194,829	38.2%	22,455	52.7%

Table 2: Cause of Road Traffic Accident

The objective of this project is to research, study, and apply deep learning models to address the problem of detecting road surface damages. In this task, we used YOLOv11 with the RDD2022 data set, which contains a total of 47,420 road images captured from various perspectives using devices such as drones, smartphone-mounted vehicles, or high-resolution cameras. The data set includes annotations for more than 55,000 instances of four categories of road damage: Longitudinal Cracks (D00), Transverse Cracks (D10), Alligator Cracks (D20), and Potholes (D40).

The data set originates from the Crowdsensing-Based Road Damage Detection Challenge (CRDDC), a competition focused on analyzing and evaluating data from six countries. India, Japan, the Czech Republic, Norway, the United States, and China. The challenge attracted the participation of more than 90 teams representing more than 20 countries, each offering different approaches and solutions.

The structure of the report

1. SECTION 1: INTRODUCTION This section represent the briefly introduction about the topic, the goal, significance as the structure of the report.
2. SECTION 2: DATASETS This section represent the basic knowledge about the RDD2022 Dataset use in this project
3. SECTION 3: THEORY This section represent detailed information about the theories applied in this project, including: Machine Learning, Neural Networks, Convolutional Networks, Object Classification and Segmentation, and Evaluation Metrics.
4. SECTION 4: MODELS This section represent detailed theory and architectur of Template Matching and R-CNN, YOLO models.
5. SECTION 5: EXPERIMENTS This section represent about the environment, data pre-process, data statistics, model setup.
6. SECTION 6: CONCLUSION This section represent the achieved results, the limitations, and the future directions of the topic.

2 Datasets

2.1 More information about RDD2022

The Road Damage Dataset (RDD) 2022 is an extended version of the existing RDD2020 dataset. The figure below illustrates the evolution of RDD2022 from its inception.

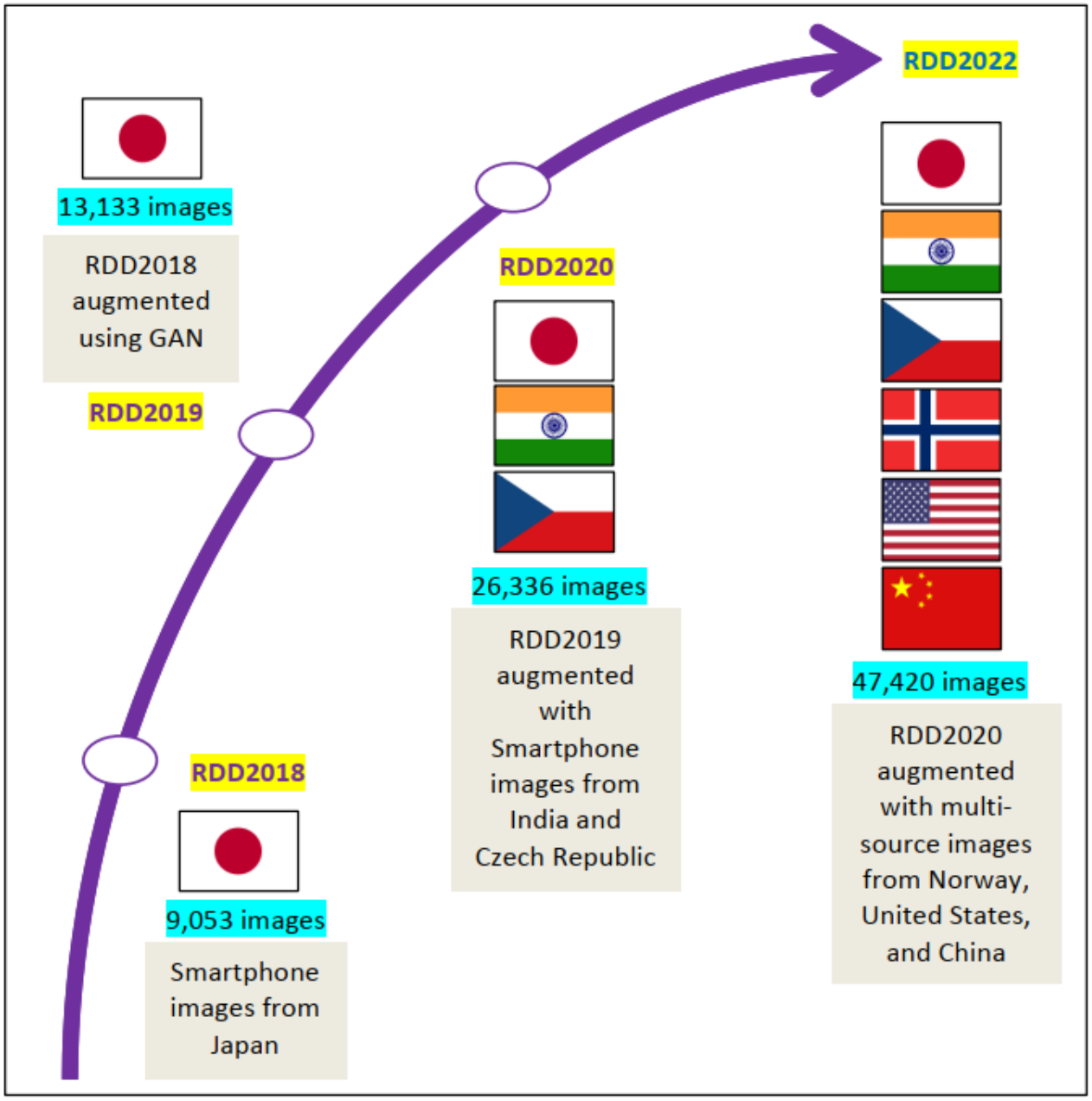


Figure 2: Evolution of RDD: From 2018 to 2022

Starting in 2018, the dataset initially contained 9,503 images, all captured using smartphones in Japan. By 2019, the dataset underwent significant improvements, with annotation corrections and data augmentation using Generative Adversarial Networks (GANs). In 2020, the dataset nearly doubled in size with additional image contributions from India and the Czech Republic. Finally, in 2022, the dataset reached near completion, comprising 47,420 images collected from six countries, and was officially released as part of the Crowdsensing-based Road Damage Detection Challenge (CRDDC 2022). Here are some example of the four types images and of damage consider in RDD2022 showin figure 3.

Figure 3: Examples of Road Damage Types in RDD2022:
(a) Longitudinal Cracks (D00), (b) Transverse Cracks (D10), (c) Alligator Cracks (D20), (d) Potholes (D40)

But in this project, we primarily focus on the RDD2022 dataset, which is mainly divided into two parts: the train set and the test set. The train set is provided with annotations, while the test set is used to evaluate the participants' models and, therefore, does not include annotations. The figure 4 below shows the distribution of images in the train and test sets and the figure 5 show the distribution of labels in the train sets.

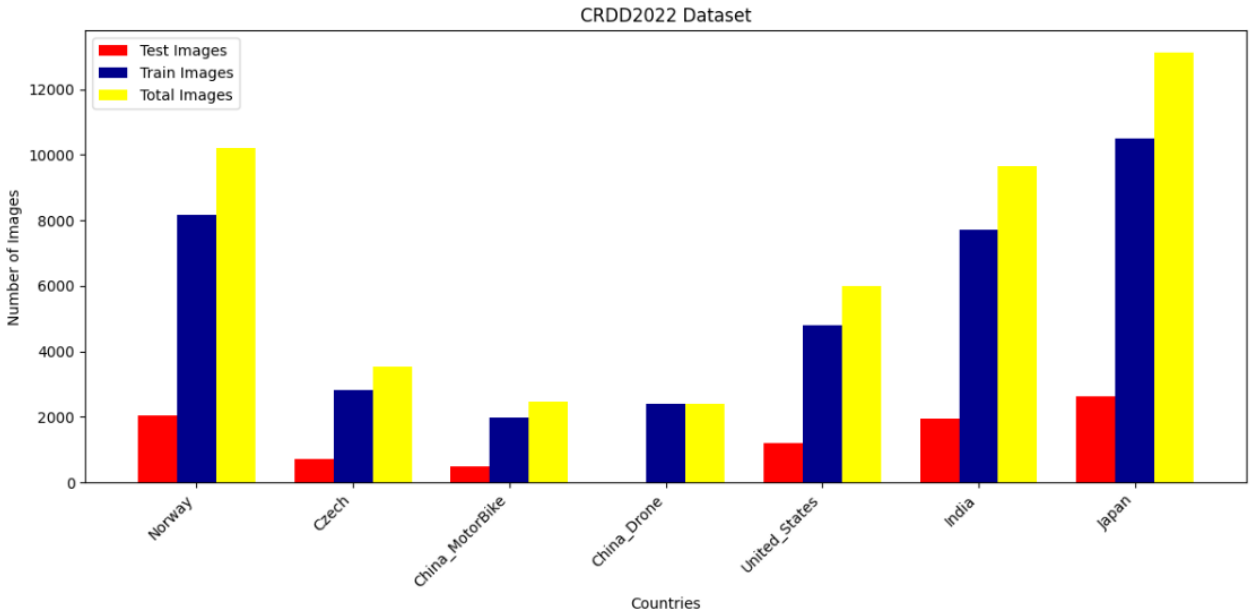


Figure 4: Distribution of train and test sets in RDD2022

1. Norway: The data from Norway consists of two classes of roads a) Expressways and b) County Roads (or Low Volume Roads). Both types of road classes are asphalt pavements. Data collection is done by the Norwegian Public Road Administration (Statens Vegvesen, SVV) and Inlandet Fylkeskommune (IFK). Images provided by SVV were collected on European Route E14, connecting the city of Trondheim in Norway to Sundsvall in Sweden. At the same time, the images from IFK belong to different county roads within Inllanndet County in Norway. The images were collected without any control over daytime/light, and all the images are natural without further processing. Further, the dataset captures diverse backgrounds, including clear grass fields, snow-covered areas, and conditions after rain. Furthermore, images with different illuminances, such as high sunlight and overcast weather resulting in daylight, are considered.

2. Czech: A substantial portion of road images was collected in Olomouc, Prague, and Bratislava municipalities and covered a mix of first-class, second-class, and third-class roads and local roads. A smaller portion of the road image dataset was collected along D1, D2, and D46 motorways to enhance the resilience of the targeted model.

3. China: RDD2022 considers two types of data from China: (a) images captured by Drones (represented as China_Drone or China_D), and (b) the images captured using Smartphone-mounted MotorBikes (represented as China_MotorBike or China_M). The drone images were obtained from Dongji Avenue in Nanjing, China.

The MotorBike images were collected on Jiu long hu campus, Southeast University. Images with normal light, under a shadow environment, and wet stains are covered.

4. United_States: The data from the United States consists of Google Street View images covering multiple locations, including California, Massachusetts, and New York.

5. India: The data from India includes images captured from local roads, state highways, and national highways, covering the metropolitan (Delhi, Gurugram) as well as non-metropolitan regions (mainly from the state of Haryana). All these images have been collected from plain areas. Road selection and time of data collection were decided based on road accessibility, atmospheric conditions, and traffic volume.

6. Japan: The data is collected from seven local governments in Japan (Ichihara city, Chiba city, Sumida ward, Nagakute city, Adachi city, Muroran city, and Numazu city). The municipalities have snowy and urban areas that vary widely from the perspective of regional characteristics like weather and budgetary constraints.

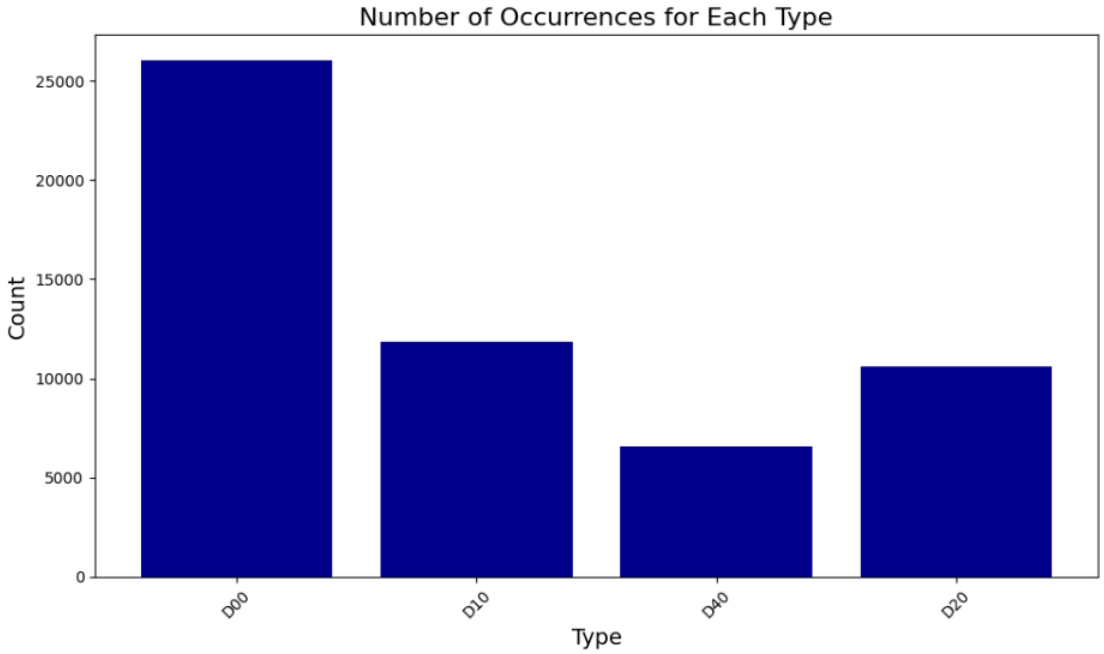


Figure 5: Distribution of labels types

2.2 The structure of RDD2022

RDD2022 can be accessed at the following GitHub repository: github.com/sekilab/RoadDamageDetector. RDD2022 is divided into 7 folders with the structure shown in Figure 6 and can be describe:

1. China_Drone: It contains data from China collected using drones. (Figure 8)
2. China_MotorBike: It covers data from China collected using motorbikes. (Figure 9)
3. Czech: It includes data from the Czech Republic collected using vehicle-mounted smartphones. (Figure 10)
4. India: It consists of data from India collected using vehicle-mounted smartphones. (Figure 11)
5. Japan: It includes data from Japan, collected using vehicle-mounted smartphones. (Figure 12)
6. Norway: It includes data from Norway collected using vehicle-mounted high-resolution cameras. (Figure 13)
7. United_States: It contains data from the United States collected using Google Street View. (Figure 14)

Figure 6: The structure of RDD2022

Each folder has a subfolder, 'train,' which includes images (.jpg) and their annotations (.xml). In addition, a 'test' folder is also included in each folder except for the China_Drone directory. The 'test' folder contains images (.jpg) for testing the model after training using the 'train' folder. The number and examples of the training images have been provided above, and a sample of the annonations (.xml) file will be shown below in Figure 7.

```
<annotation>
        <folder>images</folder>
        <filename>China_MotorBike_000002.jpg</filename>
        <size>
                <width>512</width>
                <height>512</height>
                <depth>3</depth>
        </size>
        <segmented>0</segmented>
        <object>
                <name>D00</name>
                <pose>Unspecified</pose>
                <truncated>1</truncated>
                <difficult>0</difficult>
                <bndbox>
                        <xmin>202</xmin>
                        <ymin>187</ymin>
                        <xmax>244</xmax>
                        <ymax>512</ymax>
                </bndbox>
        </object>
        <object>
                <name>D00</name>
                <pose>Unspecified</pose>
                <truncated>1</truncated>
                <difficult>0</difficult>
                <bndbox>
                        <xmin>198</xmin>
                        <ymin>1</ymin>
                        <xmax>254</xmax>
                        <ymax>141</ymax>
                </bndbox>
        </object>
</annotation>
```

Figure 7: The sampple of annonation (.xml): China_M_000002

Figure 8: Samples of China_Drone

Figure 9: Samples of China_Motorbike

Figure 10: Samples of Czech Republic

Figure 11: Samples of India

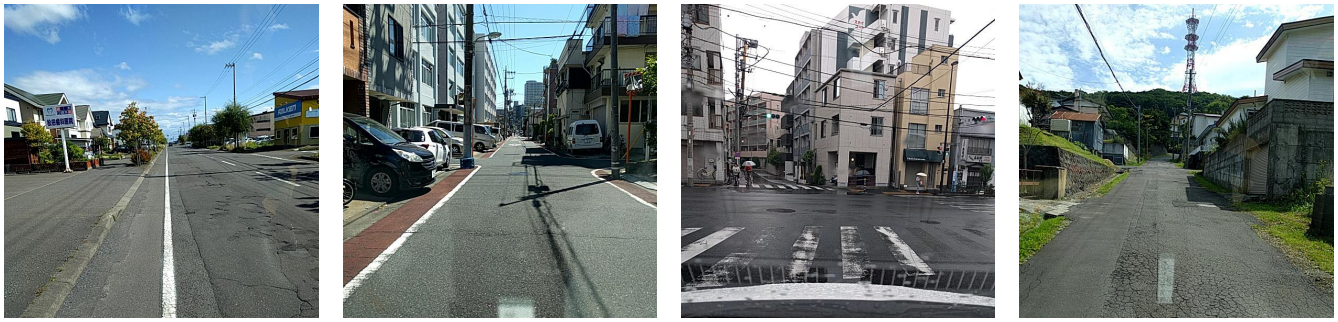


Figure 12: Samples of Japan

Figure 13: Samples of Norway

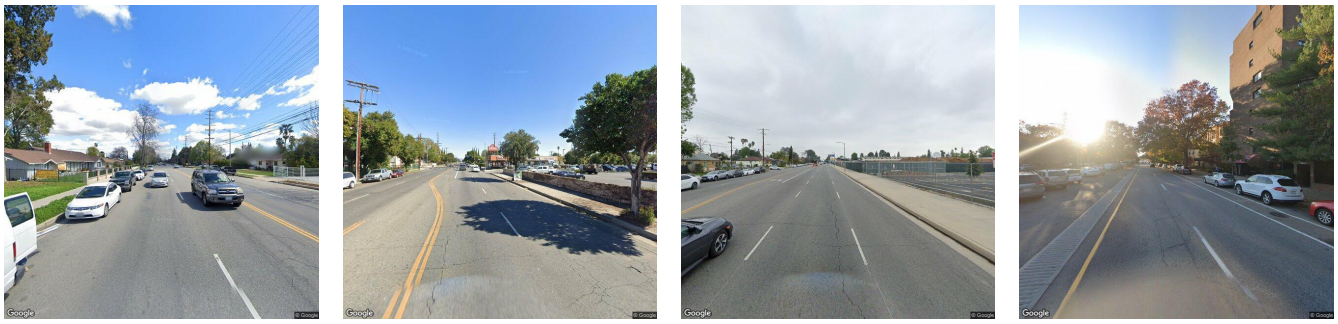


Figure 14: Samples of United_States

3 Theory

3.1 Supervised Learning

Supervised learning is a category of machine learning that uses labeled datasets to train algorithms to predict outcomes and recognize patterns. Unlike unsupervised learning, supervised learning algorithms are given labeled training to learn the relationship between the input and the outputs.

Supervised machine learning algorithms make it easier for organizations to create complex models that can make accurate predictions. As a result, they are widely used across various industries and fields, including healthcare, marketing, financial services, and more.

In supervised learning, the algorithm is trained on labeled data, where each input is associated with a corresponding output. The goal is to learn a mapping function f such that:

$$f(\mathbf{x}) = y,$$

where $\mathbf{x}$ is the input feature vector, and y is the output (label). The algorithm generalizes this mapping to predict outputs for unseen data.

3.1.1 Regression Model

A regression model provides a function that describes the relationship between one or more independent variables and a response, dependent, or target variable. To solve regression problem, it is nessesary to contruct formular in order to find out the best minimal or maximal points. There are 2 approaches in solutions for solving multi-dimentional mathematics using derivatives to solve for maximum or minimum points

The first one by examinate all of the critical points in one functions, in this ways, we can make sure that the solutions will be exits in one of the critical points, however, this will not be a good solutions for larger models with scability, as we have to have a good understands about the models in mathematics and understands about data relationship.

Another approach is to use **Gradient Decent**, in the **Gradient Decent**, we don't have to examinate all the relationships and contruct a well-known mathematics functions, insteads, we can use derivative in in order to minimize a function by iteratively moving in the direction of steepest descent, defined by the negative of the gradient. The goal is to minimize a cost function $J(\theta)$, where θ represents the parameters of the model. With this approaches, we do not ensure that the solutions is corrects, since we can stuck on a critical points that can be saddle points or a minimal critical points that is not the corrects ones. If the Learning Rate are too high, it can cause diverent behaviors, while small Learning Rate might costs time to trains.

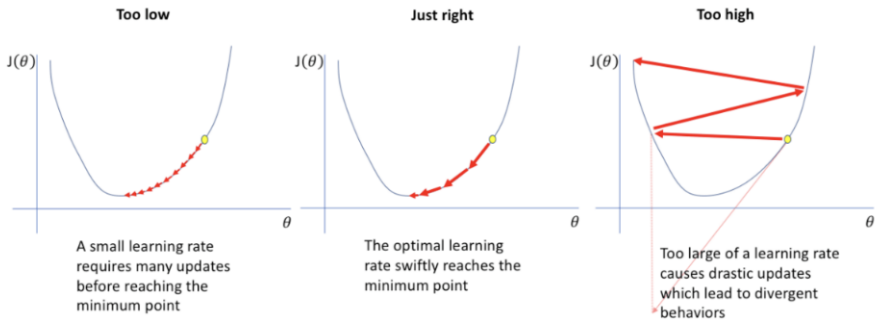


Figure 15: Learning Rate effects

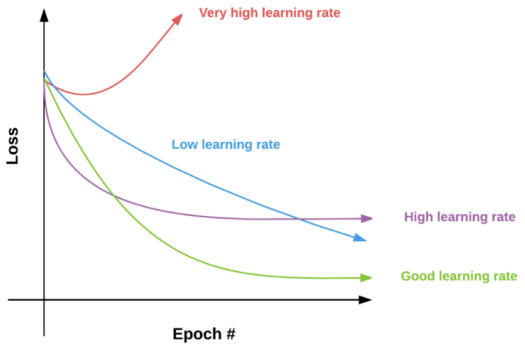


Figure 16: Learning Rate effects the Loss Functions

The solutions for Linear Regression using the **Gradient Decent** approches is inpirations for the solutions for Neutral Network approaches in the feutures.

Gradient Descent Update Rule

The parameters are updated iteratively as follows:

$$\theta_{t+1} = \theta_t - \eta \cdot \nabla J(\theta_t)$$

where:

- θ_t: The parameter vector at iteration t,
- η: The learning rate (step size),
- $\nabla J(\theta_t)$: The gradient of the cost function with respect to θ_t.

Gradient Descent for Linear Regression

Linear regression models the relationship between the input features X and the target variable y using the equation:

$$y = X\theta + \epsilon$$

where:

- X: The input feature matrix,
- θ: The parameter vector,
- ϵ: The error term.

Cost Function

The cost function for linear regression is the Mean Squared Error (MSE), defined as:

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

where:

- m: The number of training examples,
- $h_\theta(x^{(i)}) = x^{(i)} \cdot \theta$: The predicted value,
- $y^{(i)}$: The actual value.

Gradient Descent for Linear Regression

The gradient of the cost function with respect to θ is:

$$\nabla J(\theta) = \frac{1}{m} X^T (X\theta - y)$$

The parameter update rule becomes:

$$\theta_{t+1} = \theta_t - \eta \cdot \frac{1}{m} X^T (X\theta_t - y)$$

Algorithm for Gradient Descent

1. Initialize θ with random values or zeros.
2. Repeat until convergence:
   - Compute the gradient
   - Update the parameters
3. Return the optimized θ.

1. **Learning Rate Schedules:** Gradually adjust the learning rate over time using methods like:
   - **Exponential Decay:** $\eta_t = \eta_0 \cdot e^{-\lambda t}$, where η_0 is the initial learning rate, λ is the decay rate, and t is the epoch number.
   - **Step Decay:** Reduce the learning rate at specific intervals.
   - **Cyclic Learning Rates:** Oscillate the learning rate between a lower and upper bound.
2. **Adaptive Optimization Algorithms:** Use algorithms like Adam, AdamW, or RMSProp that automatically adjust learning rates based on gradients.

Comparison of Optimization Methods

Method	Advantages	Disadvantages	Formula and Explanation
SVD (Singular Value Decomposition)	Accurate for solving linear problems, dimensionality reduction.	Computationally expensive for large datasets.	Decomposes matrix A into $U\Sigma V^T$, where U and V are orthogonal matrices and Σ is a diagonal matrix. Used for pseudoinverse and low-rank approximations.
Adam (Adaptive Moment Estimation)	Fast convergence, combines benefits of RMSProp and momentum.	Sensitive to hyperparameter tuning, may converge to suboptimal solutions.	$\theta_{t+1} = \theta_t - \eta \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$ $\hat{m}_t$ and $\hat{v}_t$ are bias-corrected first and second moments of the gradients.
AdamW (Adam with Weight Decay)	Improves generalization by decoupling weight decay from the gradient.	Slightly more computationally expensive than Adam.	Same formula as Adam but applies weight decay independently: $\theta_{t+1} = \theta_t - \eta \cdot \left(\frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} + \lambda \cdot \theta_t \right)$

Method	Advantages	Disadvantages	Formula and Explanation
RMSProp	Handles non-stationary objectives well, reduces oscillations.	May get stuck in local minima for some problems.	$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} \cdot g_t$ Maintains an exponentially decaying average of past squared gradients.
SGD (Stochastic Gradient Descent)	Simple and scalable, works well with large datasets.	Slow convergence, sensitive to learning rate.	$\theta_{t+1} = \theta_t - \eta \cdot \nabla J(\theta_t)$
Nadam (Nesterov-accelerated Adaptive Moment Estimation)	Improves convergence by adding Nesterov momentum to Adam.	Can be computationally more intensive than Adam.	Similar to Adam with Nesterov momentum incorporated: $\theta_{t+1} = \theta_t - \eta \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$
Adagrad	Adapts learning rate for each parameter.	Aggressive decay of learning rates can make optimization stall.	$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t} + \epsilon} \cdot g_t$ G_t accumulates the squared gradients.

3.1.2 Clustering Algorithm

3.1.2.a Perceptron Learning Algorithm

The Perceptron Learning Algorithm (PLA) is a foundational method for solving binary classification problems. It operates on the principle of finding a hyperplane that separates two linearly separable classes. The algorithm iteratively updates the weights of the hyperplane to minimize misclassification.

However, there is drawbacks when analsying data with low Linearity relationships, by assuming that the data can be cut in half by a margin Linear Plane, if the

Given a dataset:

$$\{(\mathbf{x}_i, y_i)\}_{i=1}^{n}, \quad \mathbf{x}_i \in \mathbb{R}^d, \quad y_i \in \{-1, 1\},$$

where $\mathbf{x}_i$ represents the feature vectors and y_i the corresponding class labels.

The goal is to find a weight vector $\mathbf{w} \in \mathbb{R}^d$ and a bias $b \in \mathbb{R}$ such that:

$$y_i(\mathbf{w}^\top \mathbf{x}_i + b) > 0 \quad \forall i.$$

Initialize $\mathbf{w}_0 = \mathbf{0}$ and $b_0 = 0$. For each misclassified point $(\mathbf{x}_i, y_i)$, update the weights as:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \eta y_i \mathbf{x}_i,$$

$$b_{t+1} = b_t + \eta y_i,$$

where $\eta > 0$ is the learning rate.

The algorithm terminates when no misclassified points remain or after a predefined number of iterations.

—

Support Vector Machines (SVM)

Support Vector Machines (SVM) extend the idea of PLA by finding the hyperplane with the largest margin between classes. This margin maximization makes SVM more robust to noise and generalizable to unseen data.

SVM Optimization Problem For a linearly separable dataset, SVM solves the following optimization problem:

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2,$$

subject to:

$$y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1, \quad \forall i.$$

The dual formulation of the SVM problem is given by:

$$\max_{\boldsymbol{\alpha}} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i^\top \mathbf{x}_j),$$

subject to:

$$\sum_{i=1}^{n} \alpha_i y_i = 0, \quad 0 \leq \alpha_i \leq C \quad \forall i.$$

Here, $\boldsymbol{\alpha}$ are the Lagrange multipliers, and $C > 0$ is the regularization parameter.

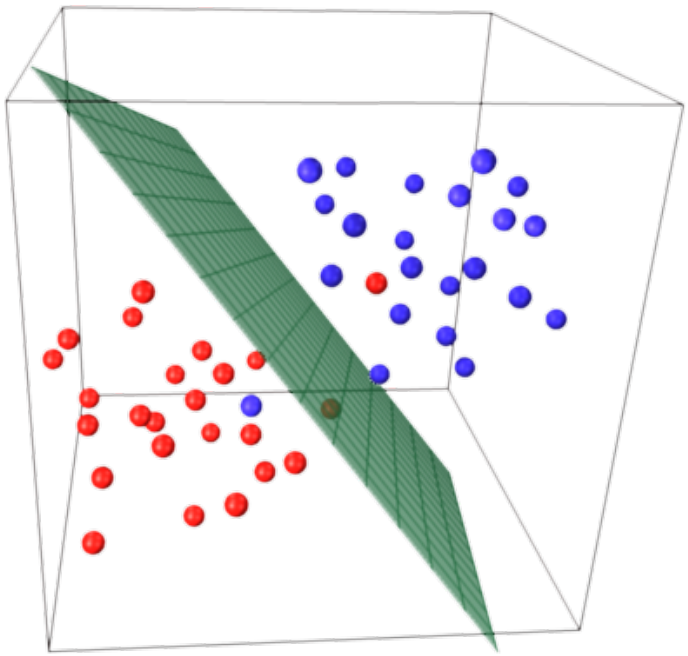


Figure 17: Hyper Plane for SVM

3.1.2.b Logistic regression

However, there are drawbacks for Perceptron Learning and SVM, as these models assume that the relationship between groups of objects can be divided by a linearity planes, however, in real world snerios and non-linearity relationship such as for Computer Vision, the data may have small Linearity relationship. To subpass this problem for Computer Visions, the combinations of PCA (Un-Supervised algorithm) and SVM have been conducted to Segment the objects.

Logistic regression is a supervised machine learning algorithm widely used for binary classification tasks, such as identifying whether an email is spam or not and diagnosing diseases by assessing the presence or absence of specific conditions based on patient test results. This approach utilizes the logistic (or sigmoid) function to transform a linear combination of input features into a probability value ranging between 0 and 1. This probability indicates the likelihood that a given input corresponds to one of two predefined categories. The essential mechanism of logistic regression is grounded in the logistic function's ability to model the probability of binary outcomes accurately. With its distinctive S-shaped curve, the logistic function effectively maps any real-valued number to a value within the 0 to 1 interval. This feature renders it particularly suitable for binary classification tasks, such as sorting emails into "spam" or "not spam". By calculating the probability that the dependent variable will be categorized into a specific group, logistic regression provides a probabilistic framework that supports informed decision making. This can be denoted as simple Neutral Network.

The Sigmoid Function was introduced to eliminate the effects of Linearity problem in Linear Regressions.

$$h_\theta(x) = \sigma(z) = \frac{1}{1 + e^{-z}}, \quad \text{where } z = \theta^T x$$

Based on threholds (mostly 0.50), if the output of the Sigmoid is outpass the Threholds, then the objects is label as True. In multiple labels problem, the input will be one-hot encoded into Binary forms, and in return, the results of probability will be outputed corresponding with each label, if all the labels below the threaholds, then it will be labeled as Default Labels

$$h_\theta(x) \geq 0.5 \implies \text{Class 1}, \quad h_\theta(x) < 0.5 \implies \text{Class 0}$$

$$\theta^T x \geq 0 \implies \text{Class 1}, \quad \theta^T x < 0 \implies \text{Class 0}$$

Using the Binary Cross-Entropy as Loss Functions, however, we can customize different Loss Functions based on the requirement, such as for Image Detections for YOLO.

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log(h_\theta(x^{(i)})) + (1-y^{(i)})\log(1-h_\theta(x^{(i)}))\right]$$

The Functions are then optimize as the follow:

$$\theta_j := \theta_j - \alpha\frac{\partial J(\theta)}{\partial \theta_j}, \quad \forall j = 0, 1, \ldots, n$$

Where:

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m}\left[h_\theta(x^{(i)}) - y^{(i)}\right]x_j^{(i)}$$

3.2 Un-Supervised Learning

Unsupervised learning is a framework in machine learning where, in contrast to supervised learning, algorithms learn patterns exclusively from unlabeled data. Other frameworks in the spectrum of supervisions include weak- or semi-supervision, where a small portion of the data is tagged, and self-supervision. Some researchers consider self-supervised learning a form of unsupervised learning.

There were algorithms designed specifically for unsupervised learning, such as clustering algorithms like k-means, dimensionality reduction techniques like principal component analysis (PCA), Boltzmann machine learning, and autoencoders. After the rise of deep learning, most large-scale unsupervised learning have been done by training general-purpose neural network architectures by gradient descent, adapted to performing unsupervised learning by designing an appropriate training procedure.

3.2.1 Principle Component Analysis (PCA)

PCA is a dimensionality reduction technique that finds a new set of orthogonal axes (principal components) capturing the maximum variance in the data. Below are the key mathematical formulations of PCA:

Input Data Representation

Let the input dataset be represented as a matrix X of dimensions $m \times n$, where:

$$X = \begin{bmatrix} x_1^T \\ x_2^T \\ \vdots \\ x_m^T \end{bmatrix}, \quad x_i \in \mathbb{R}^n$$

Here:

- m: Number of data samples.
- n: Number of features in each sample.

Mean-Centering the Data

The first step in PCA is to subtract the mean of each feature from the dataset:

$$\bar{x}_j = \frac{1}{m} \sum_{i=1}^{m} x_{ij}, \quad \text{for } j = 1, 2, \ldots, n$$

The mean-centered data matrix is:

$$X_{\text{centered}} = X - \mu, \quad \mu = \begin{bmatrix} \bar{x}_1 & \bar{x}_2 & \ldots & \bar{x}_n \end{bmatrix}$$

Covariance Matrix

The covariance matrix of the mean-centered data is computed as:

$$\Sigma = \frac{1}{m} X_{\text{centered}}^T X_{\text{centered}}$$

- $\Sigma \in \mathbb{R}^{n \times n}$ is a symmetric matrix where each entry Σ_{ij} represents the covariance between the i-th and j-th features.

Eigenvalue Decomposition

The covariance matrix Σ is decomposed into eigenvalues and eigenvectors:

$$\Sigma v = \lambda v, \quad v \in \mathbb{R}^n, \lambda \in \mathbb{R}$$

- v: Eigenvector, representing a principal direction in the data.
- λ: Corresponding eigenvalue, representing the variance along the principal direction.

The eigenvectors are orthogonal and form a basis for the new feature space.

Selecting Principal Components

Sort the eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_n$ in descending order and select the top k eigenvectors V_k corresponding to the largest k eigenvalues. These eigenvectors form the transformation matrix:

$$V_k = \begin{bmatrix} v_1 & v_2 & \ldots & v_k \end{bmatrix}, \quad V_k \in \mathbb{R}^{n \times k}$$

Dimensionality Reduction

Project the original data onto the k-dimensional subspace:

$$Z = X_{\text{centered}} V_k$$

Here:

- $Z \in \mathbb{R}^{m \times k}$: Reduced data representation.
- V_k: Principal components (transformation matrix).

Reconstruction of Data

To reconstruct the original data from the reduced data:

$$\hat{X} = Z V_k^T + \mu$$

3.3 Neural Network

A Neural Network (NN) is a computational model inspired by the structure and functioning of biological neural networks found in the human brain. It is a cornerstone of machine learning, particularly excelling in tasks that involve pattern recognition, data classification, and function approximation. Neural networks are composed of interconnected layers of nodes, often called neurons or perceptrons, designed to process and transform data through mathematical operations.

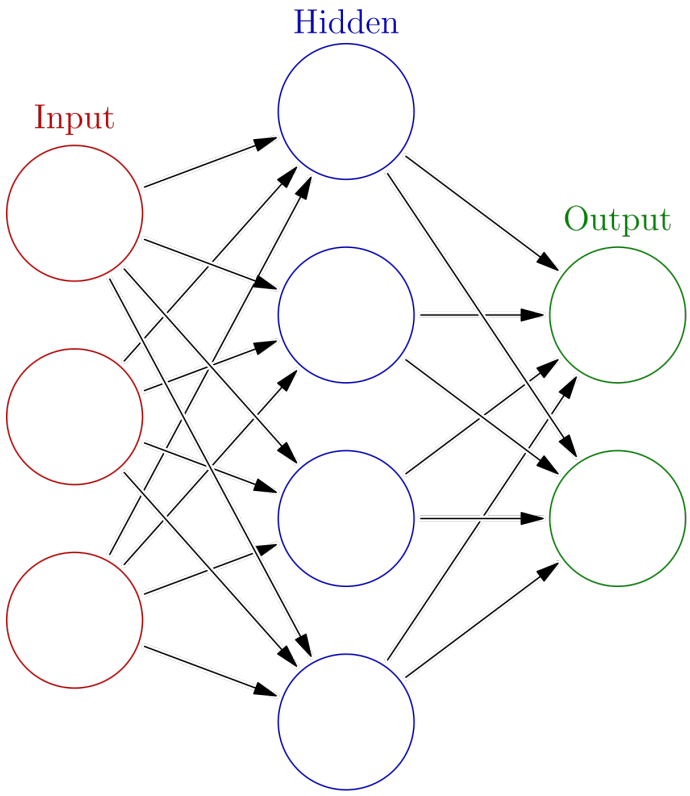


Figure 18: Neutral Network

The fundamental building block of a neural network is the artificial neuron, which performs a weighted sum of inputs followed by a nonlinear activation function. These neurons are organized into layers:

Input Layer: Receives raw data or features for processing.

Hidden Layers: Perform transformations to capture patterns and abstractions.

Output Layer: Produces the final result, such as a prediction or classification. A neural network learns to map inputs to outputs by adjusting the weights of its connections through an optimization process called training. Training involves minimizing a predefined loss function that quantifies the difference between the predicted output and the actual target. This is typically achieved using algorithms like gradient descent and its variants.

Compared with Linear Regression and Logistic Regression, Neutral Network is a Regression model that eliminated the effects of Linearity relationship, moreover, Neutral Network analyse deeper relationship than Logistic Regression, however, unlike Logistic Regression, the Neutral Network is a Black-box model, meaning we are un-likely to know how it is truely trains if the model is large, this may create a **Gradient Dissapear** and **Overfitting** of trainned nodes, which can be partly eliminate through **Dropout**

Similar to Logistic Regression, each Node of Neutral Networks is actually a Logistic Regression, with **Linear Layer** and corresponding **Activate Function** (Sigmoid for Logistic Regression) for Linearity removal and Binary transformations

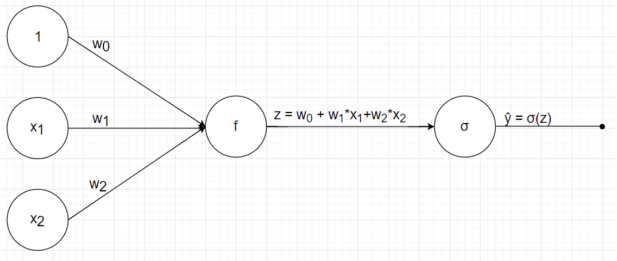


Figure 19: Simple Neutral Networks

Notation

- X: Input vector
- W_1, W_2: Weight matrices for the hidden layer and output layer
- b_1, b_2: Bias vectors for the hidden layer and output layer
- Z_1, Z_2: Linear transformations at each layer
- A_1, A_2: Activations at each layer
- y: Ground truth label
- $\hat{y}$: Predicted output
- L: Loss function
- σ: Activation function (e.g., ReLU or sigmoid)

Forward Propagation (Inference)

1. Compute the linear transformation for the hidden layer:
$$Z_1 = W_1 X + b_1$$
2. Apply the activation function:
$$A_1 = \sigma(Z_1)$$
3. Compute the linear transformation for the output layer:
$$Z_2 = W_2 A_1 + b_2$$
4. Apply the activation function to get the predicted output:
$$\hat{y} = \sigma(Z_2)$$

Backward Propagation

1. Compute the loss:
$$L = \text{Loss}(\hat{y}, y)$$
For example, using mean squared error:
$$L = \frac{1}{2}(\hat{y} - y)^2$$
2. Compute the gradient of the loss with respect to the output:
$$\frac{\partial L}{\partial Z_2} = (\hat{y} - y) \cdot \sigma'(Z_2)$$

3. Backpropagate through the output layer:

$$\frac{\partial L}{\partial W_2} = \frac{\partial L}{\partial Z_2} \cdot A_1^T$$
$$\frac{\partial L}{\partial b_2} = \frac{\partial L}{\partial Z_2}$$

4. Backpropagate through the hidden layer:

$$\frac{\partial L}{\partial Z_1} = (W_2^T \cdot \frac{\partial L}{\partial Z_2}) \cdot \sigma'(Z_1)$$
$$\frac{\partial L}{\partial W_1} = \frac{\partial L}{\partial Z_1} \cdot X^T$$
$$\frac{\partial L}{\partial b_1} = \frac{\partial L}{\partial Z_1}$$

5. Update the weights and biases using gradient descent:

$$W_1 \leftarrow W_1 - \eta \frac{\partial L}{\partial W_1}, \quad b_1 \leftarrow b_1 - \eta \frac{\partial L}{\partial b_1}$$
$$W_2 \leftarrow W_2 - \eta \frac{\partial L}{\partial W_2}, \quad b_2 \leftarrow b_2 - \eta \frac{\partial L}{\partial b_2}$$

where η is the learning rate.

3.4 Convolutional

In the early days of Computer Vision, most of the approaches focus on Feature Extraction using Un-Supervised Machine Learning combine with classification such as SVM or Neutral Networks to classify the pictures labels, however, it is not possible for Segmentational and Object Detection due to the computational problems and feature extractions, take example for a 64x64x3 pictures, if we use traditional feature extractions, let assume it is 32x32x6 using PCA, then we have to input the total of 12288 Input Nodes, for this we want to consider for more, control-able ways and faster computations.

Convolutional Neural Networks (CNNs) are a class of deep learning models specifically designed for processing data with a grid-like structure, such as images and time-series data. CNNs exploit the spatial and temporal hierarchies in data, making them highly effective for tasks like image recognition, object detection, and natural language processing.

The core building block of a CNN is the *convolution operation*, which applies a filter (or kernel) to an input to extract local patterns. Mathematically, for a 2D input $I(x, y)$ and a kernel $K(u, v)$, the convolution is defined as:

$$S(x, y) = \sum_u \sum_v I(x - u, y - v) \cdot K(u, v)$$

Here:

- $I(x, y)$: Input image or feature map.
- $K(u, v)$: Kernel (filter) of size $k \times k$.
- $S(x, y)$: Output feature map (result of the convolution).

In practice, the convolution operation is often implemented using a *cross-correlation*:

$$S(x, y) = \sum_u \sum_v I(x + u, y + v) \cdot K(u, v)$$

Kernels are designed to detect specific patterns in the input, such as edges, corners, or textures.

Operation	Kernel ω	Image result g(x,y)
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	

Figure 20: Kernels in Convolutional Operations

For the sizes, while **Stride** reduces the calculations, **Padding** was used to control the size after the Convolutional operations

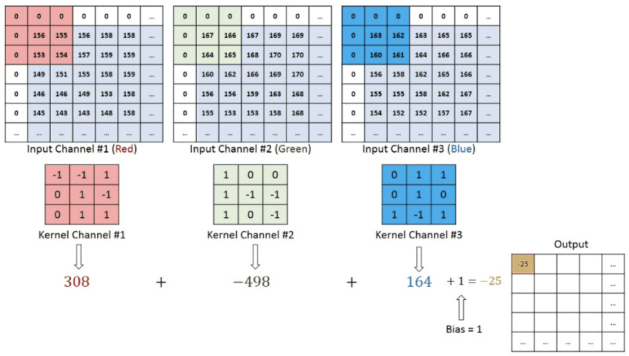


Figure 21: Convolutional Calculations

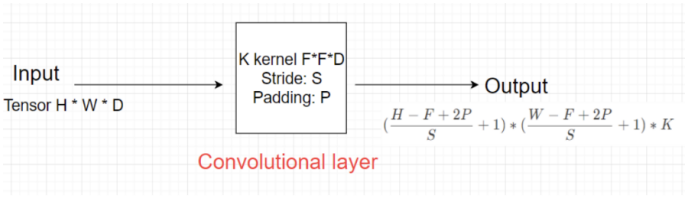


Figure 22: Convolutional Sizes

A typical CNN processes input data follow a Convolutional Process and Pooling to extracts the relationship and informations about the Image, then it will be transport to Regression Model such as SVM or Neutral Network to predicts the output.

1. Convolve the input I with kernels K to produce feature maps:

$$S^l = \sigma\left(W^l * S^{l-1} + b^l\right)$$

where:

- S^l: Feature map at layer l.
- W^l: Weight (filter) at layer l.
- b^l: Bias term at layer l.
- σ: Activation function (e.g., ReLU).

2. Apply pooling to reduce spatial dimensions.
3. Pass through fully connected layers to produce the final output.

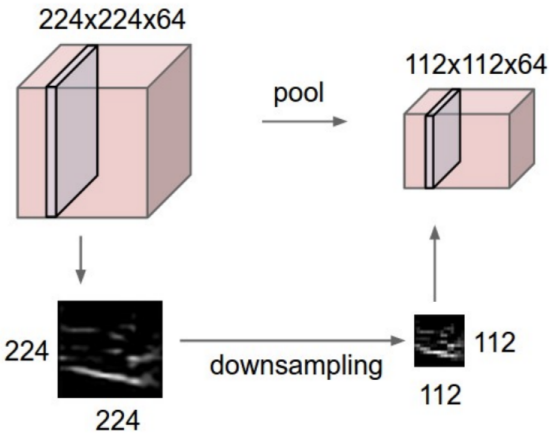


Figure 23: Pooling

For **Backbone Layer**, there are multiple Convolutional Layer, extracting Details in a single Pictures of different Spartial, take example for VGG16 feature extractions using 16 layers of Convolutional, after that, the results will be transfer to a Neutral Network for classification

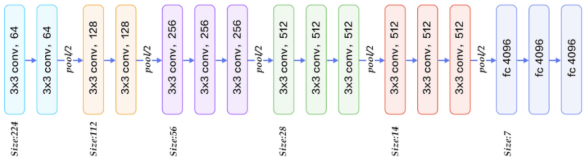


Figure 24: VGG16 structure

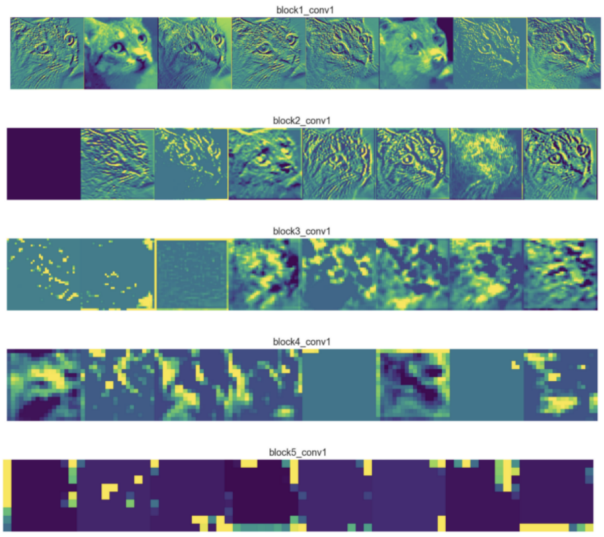


Figure 25: Feature Map of VGG16

3.5 Object Classification and Segmentation

Region Proposal Algorithm

Felzenszwalb's algorithm is a popular and efficient method for image segmentation, used to group pixels into regions based on local homogeneity. It models an image as a graph where each pixel is a node, and edges connect neighboring pixels based on their similarity in intensity and color. The algorithm merges regions progressively based on their similarity, starting from individual pixels and combining them into larger homogeneous regions.

The strength of Felzenszwalb's algorithm lies in its ability to detect boundaries of objects in a scalable manner. It utilizes a minimal spanning tree approach to find the optimal regions for segmentation while considering both color and spatial information. This algorithm is particularly well-suited for applications in object detection and scene understanding, where accurate delineation of objects from the background is crucial. The approach is computationally efficient, making it a strong candidate for real-time image processing in complex environments.

Algorithm 1 Felzenszwalb's Segmentation Algorithm

Input: A graph $G = (V, E)$ with n vertices and m edges. The input S is a segmentation of V into components $S = \{C_1, \ldots, C_r\}$.
Sort E into $\pi = (o_1, \ldots, o_m)$ by non-decreasing edge weight.
Start with a segmentation S^0, where each vertex v_i is in its own component.
for $q = 1$ **to** m **do**
  Construct S^q given S^{q-1} as follows:
  Let v_i and v_j denote the vertices connected by the q-th edge in the ordering, i.e., $o_q = (v_i, v_j)$.
  if v_i and v_j are in disjoint components of S^{q-1} and $w(o_q)$ is small compared to the internal difference of both those components **then**
    Merge the two components; otherwise, do nothing.
  end if
  Formally:
  if $C_i^{q-1} \neq C_j^{q-1}$ and $w(o_q) \leq \text{MInt}(C_i^{q-1}, C_j^{q-1})$ **then**
    S^q is obtained from S^{q-1} by merging C_i^{q-1} and C_j^{q-1}.
  else
    $S^q = S^{q-1}$.
  end if
end for
Output: The segmentation $S = S^m$.

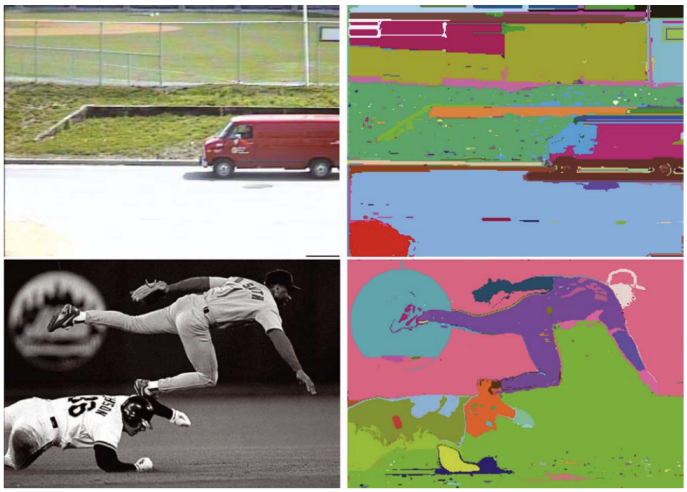


Figure 26: Result of Felzenszwalb's algorithm

Each segment is treated as a potential region of interest, and bounding boxes are computed by identifying the minimum and maximum x and y coordinates of the pixels belonging to each segment. These boxes are then refined by filtering out small or irregularly shaped boxes using criteria such as area or aspect ratio. To merge overlapping boxes and remove redundancy, Non-Maximum Suppression (NMS) is applied. NMS works by ranking boxes based on a

confidence score (e.g., derived from a classifier like an SVM). Starting with the highest-scoring box, it suppresses any overlapping boxes whose Intersection over Union (IoU) exceeds a set threshold, ensuring that only one representative box remains for each object.

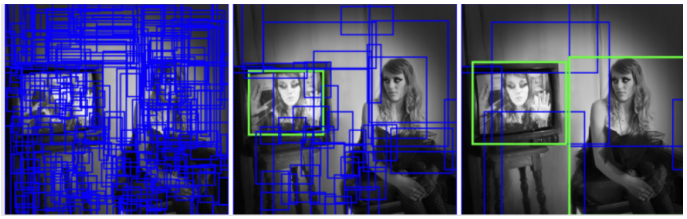


Figure 27: Region Proposal and NMS algorithm

3.6 One-Stage Detection and Two-Stage Detection

Aspect	**One-Stage Detection (YOLO)**	**Two-Stage Detection (Faster R-CNN)**
Pipeline Structure	combines object classification and localization into a single step, predicting bounding boxes and class labels directly from the input image.	Splits detection into two stages: region proposal generation and classification/localization, making it more sequential.
Speed	Designed for real-time performance, prioritizes speed over precision, making it ideal for applications requiring rapid inference.	it is slower compared to One-Stage detection due to its two-stage process.
Accuracy	achieves good accuracy, but it may struggle with small or overlapping objects in complex scenes.	provides higher accuracy, especially for detecting small objects and handling overlapping instances.
Complexity	has a simpler architecture, making it easier to train and deploy on resource-constrained devices.	is more complex, requiring more computational resources and careful tuning for optimal performance.
Use Case Suitability	Best suited for real-time tasks such as video surveillance, autonomous driving, and robotics.	Ideal for applications where accuracy is critical, such as medical imaging and high-precision object detection tasks.
Feature Pyramid	often uses a feature pyramid for multi-scale detection but may sacrifice precision at smaller scales.	incorporates multi-scale detection more effectively due to the region proposal mechanism.
Training Requirements	requires less training time and fewer resources due to its end-to-end design.	requires more extensive training and data preprocessing due to its two-stage design.

3.7 Data Augment

Data augmentation is an essential step in training object detection models such as YOLO (You Only Look Once). It helps improve the generalization capability of the model by artificially increasing the diversity of the training data. Key reasons include:

- **Improved Robustness:** Augmented data simulates variations in real-world conditions, such as lighting, orientation, and object occlusion.
- **Prevention of Overfitting:** By creating diverse training examples, augmentation reduces the model's tendency to memorize the training data.
- **Enhanced Performance:** YOLO models perform better on unseen data as augmentation introduces examples similar to edge cases.
- **Efficient Data Utilization:** For datasets with limited examples, augmentation effectively multiplies the dataset size without the need for additional data collection.

Moissac Augmentation

Moissac Augmentation involves randomly transforming the input images and their corresponding annotations to mimic different real-world scenarios. This technique includes:

- Random cropping of objects in the image.
- Adjusting object positions within the image while preserving annotations.
- Simulating partial occlusions or random scaling of objects.

This helps the YOLO model become robust to object localization under challenging scenarios.

Flip Augmentation

Flip augmentation involves flipping the image horizontally or vertically. For YOLO, horizontal flipping is commonly used. The benefits include:

- Simulates mirrored versions of objects.
- Maintains the diversity of object orientation within the dataset.

Care must be taken to adjust bounding box coordinates accordingly.

HSV Augmentation

Hue-Saturation-Value (HSV) augmentation is used to simulate changes in lighting conditions and color distributions by:

- Randomly altering the hue, saturation, and value of the image.
- Mimicking variations in environmental lighting.

HSV augmentation enhances the model's robustness to different illumination conditions.

Rotation Augmentation

Rotation augmentation rotates the image and the corresponding bounding boxes by a specified angle, either randomly or within a defined range. Benefits include:

- Increases robustness to tilted or rotated objects.
- Makes the model invariant to object orientations.

For YOLO, rotation must be applied with care to avoid loss of bounding box precision.

3.8 Evaluation Matrice

Computer vision is a field that enables machines to interpret and understand visual information from the world, similar to how humans perceive and understand visual stimuli. As machine learning and deep learning have become increasingly prevalent in computer vision tasks, evaluating the performance of algorithms has become crucial. This evaluation is typically done using **metrics**, which quantify the quality of a model's predictions against ground-truth data.

Metrics for computer vision can be broadly categorized based on the specific task or problem being addressed. Some common tasks include image classification, object detection, image segmentation, and image retrieval. Each of these tasks has different evaluation metrics that are designed to capture the accuracy of the predictions in different ways.

1. Accuracy

Accuracy is one of the most straightforward and widely used metrics for classification tasks. It is defined as the ratio of correct predictions to the total number of predictions:

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}}$$

However, accuracy may not be sufficient in imbalanced datasets where the number of samples across classes is uneven.

2. Precision, Recall, and F1-Score

For binary classification and multi-class classification problems, especially with imbalanced classes, precision, recall, and F1-score are more informative metrics. These metrics are derived from the confusion matrix.

Precision

Precision measures the ratio of true positive predictions to the total predicted positives:

$$\text{Precision} = \frac{TP}{TP + FP}$$

where TP is the number of true positives and FP is the number of false positives.

Recall

Recall (also known as sensitivity) measures the ratio of true positive predictions to the total actual positives, in Computer Vision, Recall measures is not strongly considered, while it is acceaptable to predicts wrong objects, meanwhile, in the concept of YOLO model, it minimize the Wrong-Objects proposal to minimize the process time, however, in taking serious consideration on performance in Computer Vision, Precision will be the stronger conclusion:

$$\text{Recall} = \frac{TP}{TP + FN}$$

where FN is the number of false negatives.

F1-Score

The F1-score is the harmonic mean of precision and recall, offering a balance between the two:

$$F1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

F1-score is particularly useful when the classes are imbalanced.

3. Intersection over Union (IoU)

IoU is a key metric used for evaluating object detection and segmentation tasks. It measures the overlap between the predicted bounding box and the ground-truth bounding box. The formula is given by:

$$\text{IoU} = \frac{\text{Area of Intersection}}{\text{Area of Union}}$$

A higher IoU indicates a better prediction.

4. Mean Average Precision (mAP)

Mean Average Precision (mAP) is a metric commonly used for object detection tasks. It evaluates the average precision across different recall levels. In YOLO and other object detection models, mAP is used to summarize the model's ability to detect objects correctly across multiple classes and IoU thresholds.

$$mAP = \frac{1}{C} \sum_{i=1}^{C} \text{AP}_i$$

where C is the number of classes, and AP_i is the average precision for class i.

4 Models

4.1 Template Matching - an early approach for Object Detection

Before the eras of deep learning models, machine learning approaches have been dominant for researchers worldwide to solve the Detection problem.

For the first ever approach for finding simple looking objects such as table, chair, ... Detection could be made by comparing what as known as "heatmap", where the focus areas with almost similar variance and distributions, but the limited have largely known if the objects having the same distribution but complex, such as human faces

cv2.TM_SQDIFF	$R(x,y) = \sum_{x',y'}(T(x',y') - I(x+x',y+y'))^2$
cv2.TM_SQDIFF_NORMED	$R(x,y) = \frac{\sum_{x',y'}(T(x',y') - I(x+x',y+y'))^2}{\sqrt{\sum_{x',y'} T(x',y')^2 \cdot \sum_{x',y'} I(x+x',y+y')^2}}$
cv2.TM_CCORR	$R(x,y) = \sum_{x',y'}(T(x',y') \cdot I(x+x',y+y'))$
cv2.TM_CCORR_NORMED	$R(x,y) = \frac{\sum_{x',y'}(T(x',y') \cdot I(x+x',y+y'))}{\sqrt{\sum_{x',y'} T(x',y')^2 \cdot \sum_{x',y'} I(x+x',y+y')^2}}$
cv2.TM_CCOEFF	$R(x,y) = \sum_{x',y'}(T'(x',y') \cdot I'(x+x',y+y'))$ where, $T'(x',y') = T(x',y') - 1/(w \cdot h) \cdot \sum_{x'',y''} T(x'',y'')$ $I'(x+x',y+y') = I(x+x',y+y') - 1/(w \cdot h) \cdot \sum_{x'',y''} I(x+x'',y+y'')$
cv2.TM_CCOEFF_NORMED	$R(x,y) = \frac{\sum_{x',y'}(T'(x',y') \cdot I'(x+x',y+y'))}{\sqrt{\sum_{x',y'} T'(x',y')^2 \cdot \sum_{x',y'} I'(x+x',y+y')^2}}$

Figure 28: Template Matching in cv2

In 1991, Turk and Pentland suggested an approach to face recognition that uses dimensionality reduction and linear algebra concepts of PCA to recognize faces. This has been an innovation in object detection at that time and has been widely used in the early Smartphone eras.

Algorithm 2 Alogrithm for Training Phrase

Input: Dataset of face images.
Output: Mean Eigen Faces and Lists of Eigen Face.
Step 1: Data Preparation
Collect face images and preprocess them (grayscale, normalization, resizing).
Step 2: Feature Extraction (PCA)
(a) Flatten each image into a vector.
(b) Compute the mean face vector.
(c) Center the data by subtracting the mean from each image vector.
(d) Compute the covariance matrix of the centered data.
Step 3: Template Construction
Create PCA templates from the projected face images.

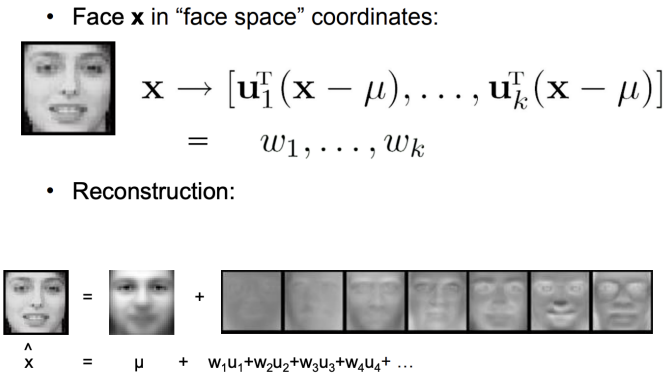


Figure 29: Inference Steps for reconizing

However, since Template PCA is an Un-Supervised Machine Learning approaches, we can not truly label an object as we want, as we only take the results and put our efforts on finding matching to find out which objects it is; another difficulty is the multiple detections, despite multiple approaches in the early 2000s by using Heat-map for analyzing the possible objects, the prediction rate was far behind than usable for multiple jobs, such as traffic car detections, while object resize is a huge problem with this algorithm.

Compare with the modern approaches, PCA algorithm performance on large datasets with multiple object detections is limited, but for datasets with small amount of data in one objects, PCA algorithm is a save-cost and suistable approaches. For example, take a Marathon contest for example, there are 1000 persons joining the run, each of them having around 8 pictures capture, Deep-learning approaches may product un-wanted product, since the training Neutral start as random, which do not close to Optimal Points, or even in opposite Extreme Point.

4.2 R-CNN Model

Unlike traditional Regression network, Traditional regression models require manual feature engineering. This means that domain experts need to identify and extract relevant features from raw data, such as pixel intensities, textures, or edges, before applying the model. Convolutional Neural Networks (CNNs) are a class of deep learning models specifically designed for processing grid-like data, such as images. They have become a cornerstone in computer vision tasks due to their ability to automatically learn hierarchical features directly from raw pixel data. A CNN consists of multiple layers, including convolutional layers that extract features, pooling layers that reduce spatial dimensions, and fully connected layers for classification or regression.

The core strength of CNNs lies in their ability to capture spatial hierarchies in images. Convolutional layers apply learned filters across the input image to detect features such as edges, textures, and shapes, which are further combined in deeper layers to recognize more complex patterns. Pooling layers, such as max pooling, down-sample the feature maps, making the model computationally efficient and robust to minor variations in input. The end-to-end trainable architecture of CNNs allows them to excel in tasks such as image classification, segmentation, and object detection. Convolutional layers with Neutral network focuses mostly on **sparse interaction, parameter sharing, and equivariant representation**

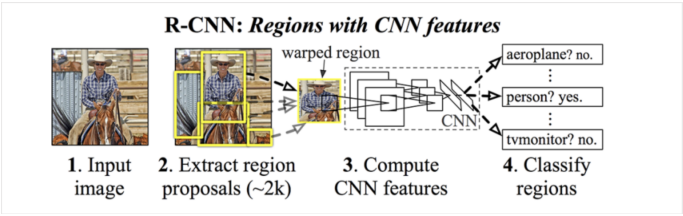


Figure 30: R-CNN models summary network

R-CNN (Regions with Convolutional Neural Networks) revolutionized object detection by integrating region proposal techniques with CNN-based feature extraction. R-CNN models work in three main stages: (1) generating region proposals using methods like Selective Search, (2) extracting fixed-size features from these regions using a CNN, and

(3) classifying each region and refining object localization using a linear SVM or a regressor.

The loss function in R-CNN is pivotal in training the model to both classify objects and localize them accurately. It typically consists of two main components:

Classification Loss

This component measures the error in assigning the correct object class to each region proposal. A common choice for classification loss is the **cross-entropy loss**, which is defined as:

$$\text{Cross-Entropy Loss} = -\sum_{i=1}^{C} y_i \log(p_i),$$

where y_i is the true label (one-hot encoded), p_i is the predicted probability, and C is the number of classes.

Bounding Box Regression Loss

This component measures how well the predicted bounding box aligns with the ground truth for an object. The regression loss is typically computed using the **Smooth L1 Loss**:

$$\text{Smooth L1 Loss} = \begin{cases} 0.5(t - t^*)^2 & \text{if } |t - t^*| < 1, \\ |t - t^*| - 0.5 & \text{otherwise,} \end{cases}$$

where t and t^* are the predicted and ground-truth bounding box coordinates, respectively.

Total Loss

The total loss for the R-CNN model is a weighted sum of the classification loss and the bounding box regression loss:

$$\text{Total Loss} = \lambda_{\text{cls}} \cdot \text{Classification Loss} + \lambda_{\text{reg}} \cdot \text{Bounding Box Regression Loss},$$

where λ_{cls} and λ_{reg} are hyperparameters that balance the contributions of the two losses.

R-CNN's introduction of region-based CNNs bridged the gap between object detection and deep learning, laying the foundation for more advanced models like Fast R-CNN and Faster R-CNN, which refine this process for better efficiency and accuracy.

4.3 Fast R-CNN Model

The integration of IoU pooling and region proposals with CNN feature extraction has revolutionized object detection by significantly enhancing both speed and accuracy. Region proposal techniques, such as Selective Search or Region Proposal Networks (RPNs), reduce computational overhead by identifying a small set of candidate regions likely to contain objects, avoiding the inefficiency of exhaustive sliding window methods. IoU pooling, particularly in Region-of-Interest (RoI) pooling, aligns these proposals with precomputed feature maps, standardizing feature dimensions and focusing on high-quality features. This synergy accelerates detection by leveraging shared CNN computations while maintaining precision, as IoU pooling ensures accurate feature extraction near object boundaries. By refining proposals and reducing redundant processing, these innovations enable modern object detection models, such as Fast R-CNN and Faster R-CNN, to achieve real-time performance without compromising on detection quality.

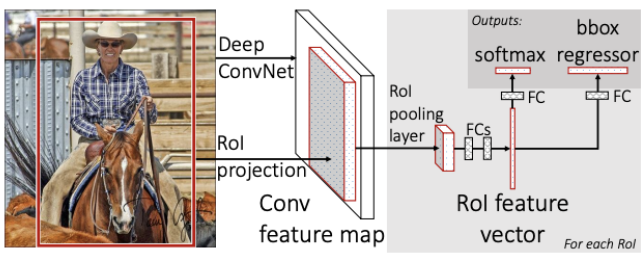


Figure 31: Fast-RCNN architecture

In modern object detection models like Fast R-CNN, the architecture typically includes two distinct layers after the feature extraction stage: the classification layer and the regression layer. These layers perform complementary tasks: the classification layer predicts the category of the object in a region, while the regression layer refines the bounding box coordinates to better align with the object's actual position. This separation enables specialized optimization for each task, improving both detection accuracy and localization precision.

Bbox regressor

Input:

- Features extracted from the *Region of Interest (RoI)* by the CNN.
- Initial region proposal coordinates: (x, y, w, h), where:
  - x, y: Center of the bounding box.
  - w, h: Width and height of the bounding box.

Output: Refined bounding box coordinates represented as offsets:

$$(\Delta x, \Delta y, \Delta w, \Delta h)$$

These offsets are used to adjust the initial proposal coordinates:

$$x' = x + w \cdot \Delta x, \quad y' = y + h \cdot \Delta y, \quad w' = w \cdot \exp(\Delta w), \quad h' = h \cdot \exp(\Delta h)$$

where (x', y', w', h') are the refined bounding box coordinates.

Final Bounding Box Regression (Post-NMS)

Input:

- Refined bounding box coordinates from the initial regression step.
- Corresponding features extracted by the CNN for these regions.

Output: Further refined bounding box coordinates:

$$(x'', y'', w'', h'')$$

4.4 Faster R-CNN model

Fast R-CNN improved object detection by introducing a unified framework for classification and localization, but it relied on external region proposal methods like Selective Search, which generated approximately 2,000 candidate boxes per image. This dependency on Selective Search posed two major disadvantages: computational inefficiency and limited correctness. The region proposal process was time-consuming and not trainable within the network, creating a bottleneck that hindered real-time performance. Moreover, the quality of proposals was suboptimal, as Selective Search heuristics often missed objects or generated redundant regions, limiting the model's ability to detect objects accurately, especially in complex scenes. Faster R-CNN addressed these issues by introducing the Region Proposal Network (RPN), which generates high-quality proposals directly within the network using shared convolutional features. The RPN, being end-to-end trainable, significantly improved both the speed and accuracy of the model. By reducing the number of candidate proposals to a manageable and relevant subset while leveraging shared computations, Faster R-CNN achieved real-time performance on GPUs and enhanced detection precision, making it more efficient and robust than Fast R-CNN.

The RPN is a fully convolutional network that slides over a shared feature map generated by the backbone CNN, such as VGG16 and REsNET,... For each spatial location on the feature map, the RPN predicts a fixed number of anchor boxes at different scales and aspect ratios. Each anchor box is evaluated for objectness (whether it contains an object or not) and refined to better fit potential objects. The algorithm involves generating these anchors, classifying them as object or background based on their Intersection over Union (IoU) with ground truth boxes, and adjusting the anchor coordinates through bounding box regression. The loss function of the RPN combines two components: classification loss and regression loss. The classification loss uses binary cross-entropy to distinguish between object and background anchors, while the regression loss employs Smooth L1 loss to refine anchor coordinates. The total RPN loss is a weighted sum of these two terms, ensuring balanced optimization of object classification and localization. This approach allows the RPN to efficiently generate high-quality region proposals that are tightly integrated with the Faster R-CNN architecture, dramatically improving detection speed and accuracy.

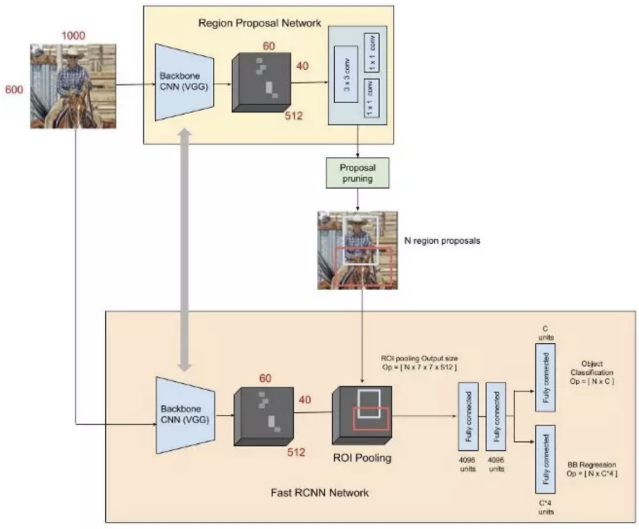


Figure 32: Faster R-CNN Structure

Calculation of Initial Proposal Boxes

Anchor boxes are generated relative to the center of the sliding window's position on the feature map. Mathematically, if the sliding window is centered at (x, y) in the feature map, and the scale and aspect ratio of the anchor box are given by (s, r), the anchor box coordinates $(x_{\min}, y_{\min}, x_{\max}, y_{\max})$ can be computed as:

$$x_{\min} = x - \frac{s \cdot \sqrt{r}}{2}, \quad y_{\min} = y - \frac{s}{2 \cdot \sqrt{r}},$$

$$x_{\max} = x + \frac{s \cdot \sqrt{r}}{2}, \quad y_{\max} = y + \frac{s}{2 \cdot \sqrt{r}}.$$

Here, s represents the anchor box's size, and r is the aspect ratio ($r = \frac{\text{height}}{\text{width}}$).

Expanding Over Multiple Anchors

For each sliding window position, k anchors are generated, where k is the total number of scale and aspect ratio combinations, for example (128x128) with ratios (1:1,2:1), **k will be 4**, this can be known as **Intial Boxes**. If the feature map has dimensions $W \times H$, then $W \times H \times k$ anchors are generated, where k is the total number of scale-aspect ratio combinations. The total number of proposals is proportional to the size of the feature map and the number of anchors, as proposals are generated for every position on the map.

At each sliding-window location, Faster R-CNN simultaneously predict multiple region proposals, where the number of maximum possible proposals for each location is denoted as k. So the reg layer has 4k outputs encoding the coordinates of k boxes (x,y,w,h), and the cls layer outputs 2k scores that estimate probability of object or not object for each proposal.

Loss Function for Faster R-CNN models

The loss function for training the RPN combines classification and regression tasks. The total loss L is defined as:

$$L(\{p_i\}, \{t_i\}) = \frac{1}{N_{\text{cls}}} \sum_i L_{\text{cls}}(p_i, p_i^*) + \lambda \frac{1}{N_{\text{reg}}} \sum_i p_i^* L_{\text{reg}}(t_i, t_i^*).$$

Components:

1. **Classification Loss $L_{\text{cls}}(p_i, p_i^*)$:** A binary cross-entropy loss that evaluates how well the RPN classifies anchor i as an object ($p_i^* = 1$) or background ($p_i^* = 0$). p_i is the predicted probability of the anchor being an object.

$$\mathcal{L}_{\text{cls}} = -\left(P_i^* \cdot \log(P_i) + (1 - P_i^*) \cdot \log(1 - P_i)\right)$$

2. **Regression Loss $L_{\text{reg}}(t_i, t_i^*)$:** A Smooth $L1$ loss used to measure the difference between the predicted bounding box adjustments $t_i = (t_x, t_y, t_w, t_h)$ and the ground truth adjustments t_i^*. The adjustments are calculated as:

$$t_x = \frac{x - x_a}{w_a}, \quad t_y = \frac{y - y_a}{h_a}, \quad t_w = \log\left(\frac{w}{w_a}\right), \quad t_h = \log\left(\frac{h}{h_a}\right),$$

   where (x_a, y_a, w_a, h_a) are the anchor box coordinates and dimensions, and (x, y, w, h) are the predicted bounding box parameters.

3. **Weights p_i^* and λ:** p_i^* ensures that regression loss is applied only to positive anchors (those matched with ground truth objects). λ balances the contributions of classification and regression losses.

4.5 YOLOv1

Fast R-CNN introduced significant improvements in object detection by integrating the region proposal process with feature extraction, classification, and bounding box regression into a single network. However, it still relied on a two-stage pipeline where the generation of region proposals (e.g., using Selective Search) and the classification tasks were sequential. This dependency on external proposal algorithms and the need for individual region processing limited the overall speed of Fast R-CNN, making it challenging for real-time applications.

YOLOv1 (**You Only Look Once**) revolutionized object detection by introducing a single-stage, unified framework. Unlike Fast R-CNN, which treats detection as a pipeline of multiple tasks, YOLOv1 formulates object detection as a regression problem. A single neural network predicts bounding boxes and class probabilities directly from the input image in a single forward pass.

The key advancements in YOLOv1 include:

- **Grid-Based Prediction:** The input image is divided into an $S \times S$ grid, where each grid cell is responsible for predicting bounding boxes and class probabilities for objects whose center falls within that cell. Such that the **output shape** is S*S*(4+c) where S is the number of Grids, C is the number of labels
- **Unified Architecture:** YOLOv1 uses a single convolutional neural network (CNN) to simultaneously predict multiple bounding boxes and their corresponding class probabilities, avoiding the need for external region proposal mechanisms.
- **Real-Time Performance:** By treating detection as a single regression task, YOLOv1 achieves real-time detection speeds while maintaining competitive accuracy.

While YOLOv1 trades off some accuracy for speed compared to Fast R-CNN, its simplified design and real-time capabilities mark a paradigm shift in object detection, making it highly suitable for applications requiring rapid and efficient detection.

To fine-tunning the structure of the **YOLOv1** structure and idea, a given loss-function has to support for the possible of the single training. Loss functions in YOLO are of two types: classification loss and regression loss. The only exception is the **YOLOv1**, where the problem of object detection was formulated as a regression problem. Till **YOLOv3**, the losses were Squared loss for bounding box regression and Cross Entropy Loss for object classification. But, from **YOLOv4**, researchers started focusing more on the IoU-based losses, as it was a better estimate of bounding box localization accuracy.

The Formations for YOLOv1 total Loss Functions:

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

where:

- S: Number of grid cells.
- B: Number of bounding boxes per grid cell.
- λ_{coord}: Weight for the coordinate loss.
- λ_{noobj}: Weight for the no-object confidence loss.
- $\mathbb{1}_{ij}^{\text{obj}}$: Indicator function, 1 if object is present in grid cell i and bounding box j, 0 otherwise.
- $\mathbb{1}_{ij}^{\text{noobj}}$: Indicator function, 1 if no object is present in grid cell i and bounding box j, 0 otherwise.
- (x_i, y_i, w_i, h_i): Predicted bounding box coordinates (center and dimensions).
- $(\hat{x}_i, \hat{y}_i, \hat{w}_i, \hat{h}_i)$: Ground truth bounding box coordinates.
- C_i: Predicted object confidence score.
- $\hat{C}_i$: Ground truth confidence score.
- $p_i(c)$: Predicted class probability for class c.
- $\hat{p}_i(c)$: Ground truth class probability for class c.

The YOLOv1 loss function can be broken down into the following components:

$$\mathcal{L} = \mathcal{L}_{\text{coord}} + \mathcal{L}_{\text{confidence}} + \mathcal{L}_{\text{class}}$$

The Coord loss penalizes the predicted bounding box coordinates (x, y, w, h) if they deviate from the ground truth. It ensures that the predicted box aligns closely with the actual object. It is defined as:

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

λ_{coord} is a hyperparameter (default $\lambda_{\text{coord}} = 5$) that assigns higher weight to localization loss.

For **the Confidence Loss function** of the predictions, it penalties the error in which the objects segment a non-releastic objects, this loss approaches aim to push the Recall rate.

$$\mathcal{L}_{\text{confidence}} = \sum_{i=1}^{S^2} \sum_{j=1}^{B} \mathbb{1}_{ij}^{\text{obj}} (C_i - \hat{C}_i)^2 + \lambda_{\text{noobj}} \sum_{i=1}^{S^2} \sum_{j=1}^{B} \mathbb{1}_{ij}^{\text{noobj}} (C_i - \hat{C}_i)^2$$

where - λ_{noobj} (default $\lambda_{\text{noobj}} = 0.5$) reduces the weight of background confidence loss, avoiding overwhelming the model with non-object predictions.

Segmentation trainning will be trainned by the Class Loss Function, for a given pi of size SxSxc, where S is the numbers of grids, c is the percentage of the single Classes

$$\mathcal{L}_{\text{class}} = \sum_{i=1}^{S^2} \mathbb{1}_i^{\text{obj}} \sum_{c=1}^{C} (p_{i,c} - \hat{p}_{i,c})^2$$

4.6 YOLOv2 (YOLO 9000)

YOLOv1 have proposed weakness model instability, especially during early iterations, while Faster R-CNN proposed archors boxes with multi-shape and boxes, YOLOv1 only used a fixed number of grids with centre objects, while it's removal of the proposed boxes, but the diffucult in finding the objects centre have been arrounds.

YOLOv2 introduces the concept of **anchor boxes** similar to Faster R-CNN model to improve bounding box prediction. Each predicted bounding box is parameterized relative to a predefined set of anchor boxes. This section explains how the bounding boxes are represented and calculated. It also proposed K-Means clustering, by this approaches, it can defied grids center that can contains the center of objects and simplified fast-finding shape of the objects, after this steps, YOLOv2 propsed Covoluntional Neutral Network to transform the proposed of the K-Means in the final product. A 4 tuples of **tx, ty, tw, th** is used to transform the given bounding boxes

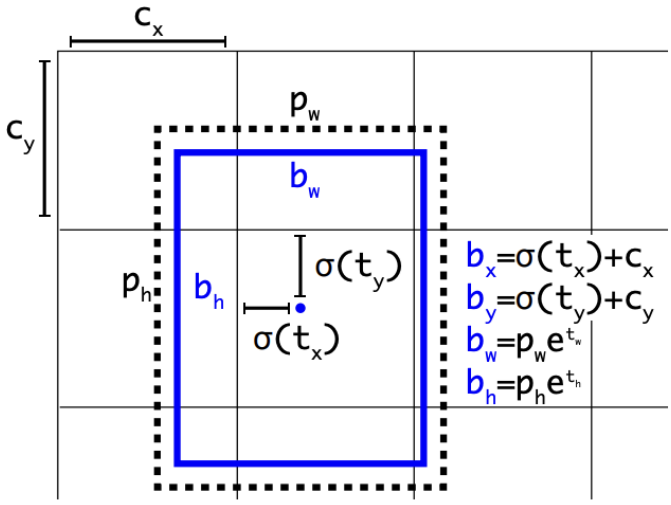


Figure 33: YOLOv2 Bounding boxes with dimension priors and location prediction

For each grid cell, YOLOv2 predicts B bounding boxes. A bounding box is parameterized as:

$$\mathbf{b} = (t_x, t_y, t_w, t_h)$$

$$\textbf{shape}(\textbf{output}) = S * S * B * (4 + 1 + C)$$

where:

- t_x, t_y: Predicted offsets for the center of the bounding box relative to the top-left corner of the grid cell.
- t_w, t_h: Predicted scaling factors for the width and height of the bounding box, relative to the dimensions of the anchor box.
- S: Number of Grids
- B: Number of archor Boxes proposed.

The final bounding box is calculated as:

$$b_x = \sigma(t_x) + c_x, \quad b_y = \sigma(t_y) + c_y$$
$$b_w = p_w e^{t_w}, \quad b_h = p_h e^{t_h}$$

where:

- (b_x, b_y): Final center coordinates of the predicted bounding box.
- (b_w, b_h): Final width and height of the predicted bounding box.
- (c_x, c_y): Top-left corner of the grid cell.
- (p_w, p_h): Width and height of the anchor box.
- $\sigma(t_x)$ and $\sigma(t_y)$: Apply the sigmoid function to ensure the center coordinates are within the grid cell bounds.

4.7 YOLOv3: FPN introductions

4.7.1 Bouding Box Prediction

Following YOLOv2 (YOLO 9000), YOLOv3 predicts bounding boxes using dimensional clusters as anchor boxes. The network predicts four coordinates for each bounding box: t_x, t_y, t_w, t_h. YOLOv3 also predicts an **objectness score** for each bounding box using **logistic regression**.

- The score will be 1 if a bounding box prior overlaps an object more than any other prior.
- If a bounding box overlaps a ground truth object, but is not the best overlap and exceeds a
- certain threshold, the prediction is ignored (with a base threshold of 0.5).

YOLOv3 only assigns one bounding box prior for each ground truth object. If a bounding box prior not assigned to a ground truth object, it occurs no loss for coordinate or class predictions, only objectness.

4.7.2 Class Prediction

Each box predicts the class the bounding box may contain using **multilabel classification**. **Independent logistic classifiers** are used instead of softmax, as softmax assumes only one class is present, which is unsuitable for cases with overlapping labels. During training, **binary cross-entropy loss** is applied for class predictions.

4.7.2.a Prediction Across Scales

YOLOv3 predicts bounding boxes at three different scales, following a method similar to Feature Pyramid Networks (FPN).

- Starting from the base feature extractor, additional convolutional layers are added. The final convolutional layer predicts a 3D tensor that encodes bounding box coordinates, objectness scores, and class predictions.
- To enhance the feature maps, the feature map from two layers earlier is upsampled by a factor of 2. Additionally, a feature map from an earlier layer in the network is merged with the upsampled map to capture more meaningful semantic information and finer-grained information from the earlier feature map.

  Several more convolutional layers are then applied to process this combined feature map, and the system eventually predicts another tensor, which is twice the size of the original.
- Then perform the process one more time to predict the final scale. This allows the predictions at the third scale to leverage all previous computations as well as the fine-grained features from the earlier layers of the network.

4.7.3 Feature Extractor

YOLOv3 utilizes a new network for feature extraction, combining elements from the network used in YOLOv2 (Darknet-19) and modern residual networks. It employs a mix of 3x3 and 1x1 convolutional layers, with the addition of **shortcut connections**. The architecture is significantly larger, consisting of 53 convolutional layers, known as Darknet-53.

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

Figure 34: DarkNet-53

Darknet-53 surpasses Darknet-19 from YOLOv2 in strength while being more efficient than ResNet-101 and ResNet-152. The comparison between these networks is illustrated in the figure below:

Backbone	Top-1	Top-5	Bn Ops	BFLOP/s	FPS
Darknet-19 [15]	74.1	91.8	7.29	1246	**171**
ResNet-101[5]	77.1	93.7	19.7	1039	53
ResNet-152 [5]	**77.6**	**93.8**	29.4	1090	37
Darknet-53	77.2	**93.8**	18.7	**1457**	78

Figure 35: Compare Darknet-53 with the others

4.8 YOLOv4: Introduction to Baby-Bies

4.8.1 Object detector Architecture

The common object detector composed of:

- Input:
  - Image, Patches, Image Pyramid
- Backbone: The Feature Extractor
  - VGG16, ResNet-50, SpineNet, EfficientNet-B0/B7, CSPResNeXt50, CSPDarknet53
- Neck: Compose different layers of the backbone to the detectors
  - **Additional blocks:** SPP, ASPP, RFB, SAM

- **Path-aggregation blocks:** FPN, PAN, NAS-FPN, Fully-connected FPN, BiFPN, ASFF, SFAM

- Head:
  - **Dense Prediction (one-stage):**
    + RPN, SSD, YOLO, RetinaNet (anchor based)
    + CornerNet, CenterNet, MatrixNet, FCOS(anchor free)
  - **Sparse Prediction (two-stage):**
    + Faster R-CNN, R-FCN, Mask RCNN (anchor based)
    + RepPoints(anchor free)

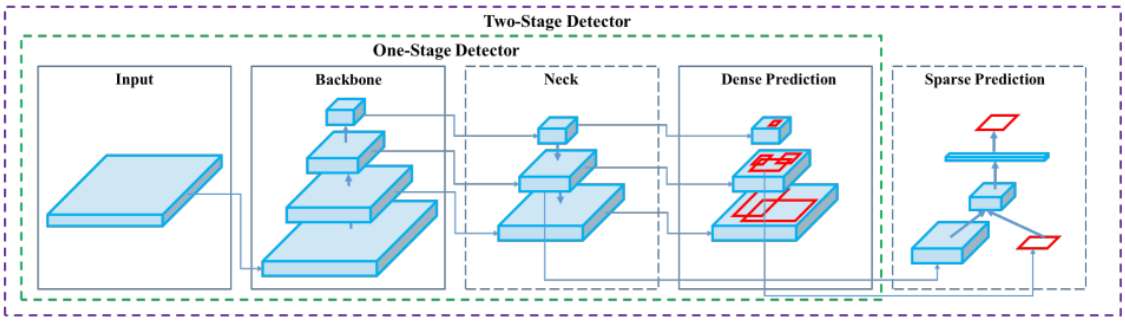


Figure 36: Object detector architecture

4.8.2 Bag of freebies (BoF)

Methods that only change the training strategy or only increase the training cost as "bag of freebies." Data augmentation is commonly adopted by object detection methods and fits the definition.

- Data Augmentation
  The goal of data augmentation is to enhance the diversity of input images, ensuring the object detection model becomes more robust when handling images from various environments. Commonly used data augmentation techniques include photometric distortions: brightness, contrast, hue, saturation,noise, ... and geometric distortions: random scaling, cropping, flipping, rotation,... . And some advice method such as DropOut, DropConnect, DropBlock, Random Erase, CutOut.... In addition, some methods apply to multiple images: MixUp, CutMix,...

- Semantic Distribution Bias in Datasets
  - Data imbalance between different classes:
    + hard negative example mining, online hard example mining (two-stage object detector)
    + focal loss (one-stage object detector)
  - The relationship of the degree of association between different categories with the one-hot hard representation.
    + The label smoothing proposed is to convert hard label into soft label for training, which can make model more robust.

- Objective Function of BBox Regression
  The traditional object detector usually uses Mean Square Error (MSE) to di rectly perform regression on the center point coordinates and height and width of the BBox. As for anchor-based method, it is to estimate the correspond ing offset. To address this, IoU loss was introduced, considering the overlap between predicted and ground truth BBoxes. Further enhancements, including GIoU, DIoU, and CIoU losses, have incorporated aspects like object shape, orientation, center distance, and aspect ratio. These improvements achieve better accuracy and faster convergence for BBox regression tasks.

4.8.3 Bag of specials (BoS)

Plugin modules and post-processing methods that only slightly increase the inference cost but significantly improve the accuracy of object detection are referred to as "bag of specials."
In general, these plugin modules are designed to improve specific aspects of a model, such as expanding the receptive field, incorporating attention mechanisms, or enhancing the ability to integrate features. Additionally, post-processing serves as a technique for refining and filtering the model's prediction results.

- Receptive Field: Common modules that can be used to enhance recep tive field are SPP, ASPP, and RFB.
- Attention module: divided into channel-wise attention and point wise attention, and the representatives of these two attention models are Squeeze-and-Excitation (SE)
- Feature integration: skip connection or hyper-column
- Post-processing: NMS is used to eliminate poorly predicted bounding boxes for the same object, retaining only the candidate boxes with the highest confidence scores.

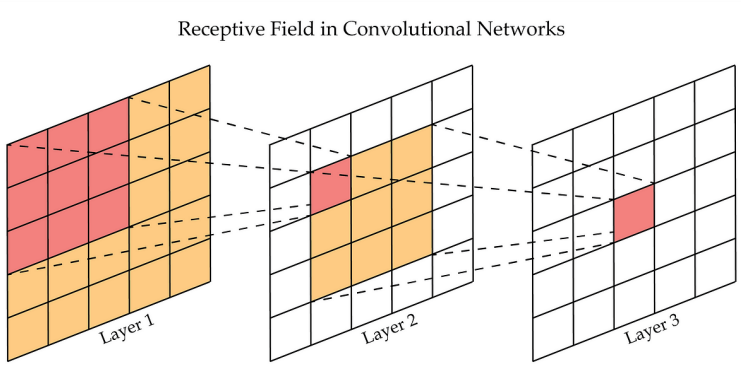


Figure 37: Receptive Field

4.8.4 YOLOv4 architecture

YOLOv4 consists of:

- Backbone: CSPDarknet53
- Neck: SPP, PAN
- Head: YOLOv3

YOLOv4 uses:

- Bag of Freebies (BoF) for backbone: CutMix and Mosaic data augmentation, DropBlock regularization, Class label smoothing
- Bag of Specials (BoS) for backbone: Mish activa tion, Cross-stage partial connections (CSP), Multi input weighted residual connections (MiWRC)
- Bag of Freebies (BoF) for detector: CIoU-loss, CmBN, DropBlock regularization, Mosaic data aug mentation, Self-Adversarial Training, Eliminate grid sensitivity, Using multiple anchors for a single ground truth, Cosine annealing scheduler, Optimal hyper parameters, Random training shapes
- Bag of Specials (BoS) for detector: Mish activation, SPP-block, SAM-block, PAN path-aggregation block, DIoU-NMS

4.9 YOLOv8: Real-Time Flying Object Detection

YOLOv8's backbone is composed of four sections, each containing a single convolution layer followed by a c2f module. The c2f module is a new introduction to the CSPDarknet53 architecture. The module consists of splits, with one end passing through a bottleneck module (which consists of two 3x3 convolutions with residual connections). The output from the bottleneck module is then divided into N parts, where N represents the size of the YOLOv8 model. These parts are eventually concatenated and passed through a final convolutional layer. This last layer is where the activations are obtained.

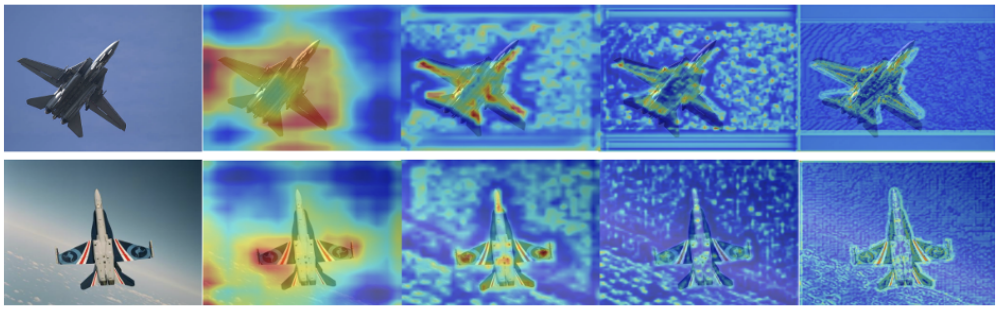


Figure 38: Four stages of c2f module

- In the first activation, this module identifies the aircraft's two wings and recognizes the object as a plane.
- The second activation map corresponds to the second c2f module in our backbone. It highlights strong activations across various parts of the aircraft, such as the wings, body, cockpit, and payload.
- The third activation map begins to focus on the fine details of the aircraft's components, likely examining subtle differences in the jet's structure.
- Finally, the model's final c2f module activates extremely fine-grained details and outlines in the respective images

YOLOv8 is an anchor-free model. This means it predicts directly the center of an object instead of the offset from a known anchor box. Anchor free detection reduces the number of box predictions, which speeds up Non-Maximum Suppression (NMS), a complicated post processing step that sifts through candidate detections after inference.

YOLOv8 apply mosaic augmentation to the training dataset. Mosaic augmentation is a technique where four random images from the training set are merged into a single mosaic image. Each quadrant of this image contains a random crop from one of the four input images, and this composite image is then used as input for the model

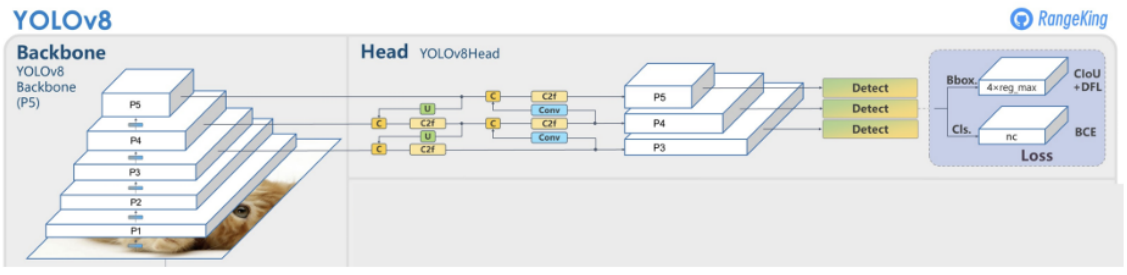


Figure 39: YOLOv8 architecture

4.10 YOLOv10: One-To-One and One-To-Many model

YOLOv10, the latest iteration in the YOLO (You Only Look Once) family of object detection models, represents a significant advancement in real-time object detection. This model builds on the strengths of its predecessors by introducing innovative features designed to enhance accuracy, speed, and robustness across a wide range of applications. Among its standout features is the emphasis on a consistent matching metric, a key innovation that improves the overall performance of the model.

The consistent matching metric in YOLOv10 addresses a fundamental challenge in object detection: ensuring that the predicted bounding boxes are evaluated against the ground truth in a reliable and meaningful way. This metric unifies the matching process across different scales and detection stages, reducing ambiguities and improving the model's ability to identify objects with higher precision. By aligning the evaluation criteria across all levels of the detection pipeline, YOLOv10 ensures that the training process focuses on minimizing inconsistencies, leading to a more stable and effective learning experience.

$$M_{\text{cons}} = \lambda_{\text{scale}} \cdot \text{IoU} + \lambda_{\text{conf}} \cdot P_{\text{conf}}, \tag{1}$$

where:

- IoU is the Intersection over Union, measuring the overlap between the predicted bounding box and the ground truth.
- P_{conf} is the predicted confidence score of the bounding box.
- λ_{scale} and λ_{conf} are adaptive weights, dynamically adjusted based on object scale and detection confidence.

This metric ensures that both localization quality (via IoU) and confidence in predictions are taken into account, balancing their contributions to the model's training and evaluation.

- **Improved Localization**: By incorporating IoU directly into the metric, YOLOv10 emphasizes accurate localization of objects, particularly at smaller scales where precision is critical.
- **Confidence Calibration**: The inclusion of P_{conf} ensures that predictions with higher confidence are prioritized, reducing the impact of low-quality detections.
- **Multi-Scale Consistency**: The adaptive weights λ_{scale} allow the metric to scale seamlessly across large, medium, and small objects, enhancing detection performance across diverse object sizes.
- **Stability in Training**: By aligning the evaluation criteria at all stages, the metric minimizes inconsistencies, leading to a more stable and efficient training process.

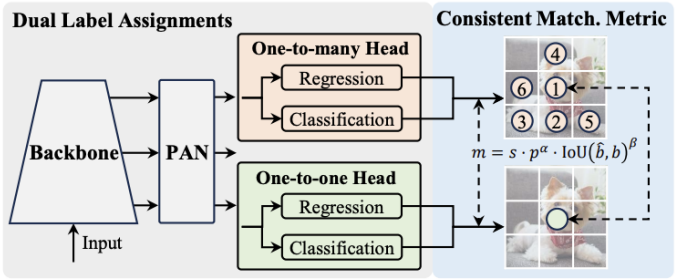


Figure 40: YOLOv10 Dual Labels Assignment

This focus on consistency not only enhances the detection performance but also contributes to YOLOv10's ability to generalize across diverse datasets and challenging scenarios. From detecting small objects in cluttered environments to identifying complex shapes in dynamic scenes, the consistent matching metric empowers YOLOv10 to achieve state-of-the-art results in both real-time and high-precision object detection tasks.

4.11 YOLOv11: Key Architectural Enhancements from YOLOv8

YOLOv11 introduces advanced architectural components to enhance feature extraction and attention mechanisms. These include the **C3K2 Block** and the **C2PSA Module**, which improve both computational efficiency and object detection performance.

C3K2 Block

The **C3K2 Block** builds upon the standard C3 module by introducing kernel diversity and depthwise separable convolutions. It can be expressed mathematically as:

$$\mathbf{F}_{\text{out}} = \mathbf{F}_{\text{merge}}\big(\text{Conv}_{k_1}(\mathbf{F}_{\text{in}}), \text{Conv}_{k_2}(\mathbf{F}_{\text{in}}), \ldots, \text{Conv}_{k_N}(\mathbf{F}_{\text{in}})\big), \tag{2}$$

where:

- $\mathbf{F}_{\text{in}}$ is the input feature map.
- $\text{Conv}_{k_i}(\cdot)$ denotes a convolution operation with kernel size k_i.
- $\mathbf{F}_{\text{merge}}$ represents a merging operation, typically concatenation followed by a pointwise 1×1 convolution to fuse features.

By combining multiple kernel sizes $k_1, k_2, \ldots, k_N$, the C3K2 Block captures both local and global contexts, improving multi-scale feature representation. The depthwise separable convolution reduces the computational cost by factorizing the convolution into depthwise and pointwise operations, reducing complexity from $\mathcal{O}(C_{\text{in}} \cdot C_{\text{out}} \cdot K^2)$ to $\mathcal{O}(C_{\text{in}} \cdot K^2 + C_{\text{in}} \cdot C_{\text{out}})$.

C2PSA Module (Attention Mechanism)

The **C2PSA Module** introduces a cross-stage partial self-attention mechanism, which can be formulated as:

$$\mathbf{F}_{\text{attn}} = \sigma\big(\mathbf{Q} \cdot \mathbf{K}^\top / \sqrt{d}\big) \cdot \mathbf{V}, \tag{3}$$

where:

- $\mathbf{Q} = \mathbf{W}_q \cdot \mathbf{F}_{\text{in}}$, $\mathbf{K} = \mathbf{W}_k \cdot \mathbf{F}_{\text{in}}$, and $\mathbf{V} = \mathbf{W}_v \cdot \mathbf{F}_{\text{in}}$ are the query, key, and value matrices derived from the input feature map $\mathbf{F}_{\text{in}}$.
- $\sigma(\cdot)$ represents the softmax function for normalizing attention scores.
- d is the dimensionality of the feature space, used for scaling.

The attention mechanism selectively enhances relevant spatial and channel-wise features, enabling better focus on significant regions in the image.

In addition, the C2PSA Module applies attention selectively across cross-stage partial connections, reducing redundant computations by focusing only on critical paths.

Performance Improvements over YOLOv8

The enhancements introduced by the C3K2 Block and C2PSA Module contribute to improved performance over YOLOv8 in several aspects:

- **Efficiency**: The depthwise separable convolutions and partial connections reduce the number of FLOPs (floating-point operations), expressed as:
$$\text{FLOPs} \propto C_{\text{in}} \cdot C_{\text{out}} \cdot K^2 \cdot H \cdot W. \tag{4}$$
By minimizing C_{in} and K^2, YOLOv11 achieves lower computational overhead than YOLOv8.
- **Robustness**: The C2PSA Module ensures better detection in challenging scenarios, such as occlusion and low-light conditions, by dynamically enhancing feature importance.

5 Experiments

5.1 Enviroment

The model was trained using the following environment configuration:

- **Python version:** 3.9.6
- **PyTorch version:** 2.5.1+cu124
- **CUDA device:** NVIDIA GeForce RTX 2080 SUPER
- **CUDA memory:** 7974 MiB
- **RAM:** 32 Gb

5.2 Data Pre-process

To use YOLOv11, preprocessing the input data is required to convert it from the PASCAL VOC format to the correct YOLO format. The model can then utilize this preprocessed dataset for training.

The YOLO labeling format includes a `.txt` file with the same name as the corresponding image file. Each `.txt` file contains annotations for the corresponding image, specifying the class name, coordinates, height, and width of the object. A single `.txt` file can contain annotations for multiple objects. The format for each object is represented on a single line as follows:

$$\langle \text{class} \rangle \ \langle x_{\text{center}} \rangle \ \langle y_{\text{center}} \rangle \ \langle \text{width} \rangle \ \langle \text{height} \rangle$$

Below is an example of YOLO annotations for an image containing two different objects:

$$[0, 0.4325, 0.7516, 0.0950, 0.1933]$$

For PASCAL VOC annotation files, each bounding box is presented by x_{min}, x_{max}, y_{min}, y_{max} and classify types, for the pictures informations, there are width, height of the pictures.

The following equations calculate the center coordinates $(x_{\text{center}}, y_{\text{center}})$ and the width and height of a bounding box (w_s, h_s) normalized by the dimensions of the image:

$$x_{\text{center}} = \frac{(x_{\text{min}} + x_{\text{max}})}{2w}$$

$$y_{\text{center}} = \frac{(y_{\text{min}} + y_{\text{max}})}{2h}$$

$$w_s = \frac{(x_{\text{max}} - x_{\text{min}})}{w}$$

$$h_s = \frac{(y_{\text{max}} - y_{\text{min}})}{h}$$

Here:

- x_{min}, x_{max}: Minimum and maximum x-coordinates of the bounding box.
- y_{min}, y_{max}: Minimum and maximum y-coordinates of the bounding box.
- w, h: Width and height of the image.
- $x_{\text{center}}, y_{\text{center}}$: Center coordinates of the bounding box, normalized by image dimensions.
- w_s, h_s: Width and height of the bounding box, normalized by image dimensions.

Figure 41: YOLO box format

5.2.1 File formats

Before the model trainning using YOLO, we have to set-up the file formats for the following structure:

```
Collect_Data/
+-- images/
|   +-- train/
|   +-- val/
+-- labels/
    +-- train/
    +-- val/
```

For each .jpg files, we have a .txt files in the labels following the file is distributed into train files or validation files, the left-over files will be used as Test files.

In model.yaml, we set up the configuration for YOLO to define the locations and defied labels types, we only conducts experiment on D00, D10, D20 and D40

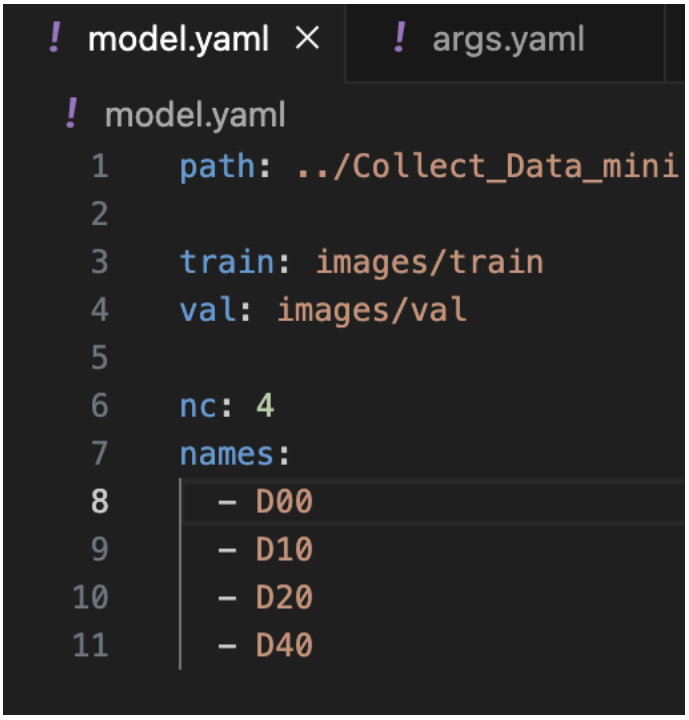


Figure 42: model.yaml structure

5.2.2 Train-test Split

At first, all the files from different nations is collected and split into Train, Validations and Train files, for 38385 pictures, we used 27636 (72%) of files for trainning, 3071 of files for Validations (8%) for validations and the rest of (20 %) is used for Test files

5.3 Data Statistics

YOLO model have weakness when predicts objects locate in a less distributed areas, by making statistic, we can use Data Augment to overcome this problem

For normalize x and y, we can capture that the boxes center are mostly distributed in one area, if the object are in (0.2, 0.2) (followed by (x,y)), the confidence for this predicts will be very small compare to the object at (0.5,0.8), to solve this, we can use Rotations, Flip and other newly distributed way. However, when looking at the Box shape, all of them are parrallel to the XoY coordinates, meaning that, for the ω angle detection in YOLOv11 will mostly arround 0 or 90

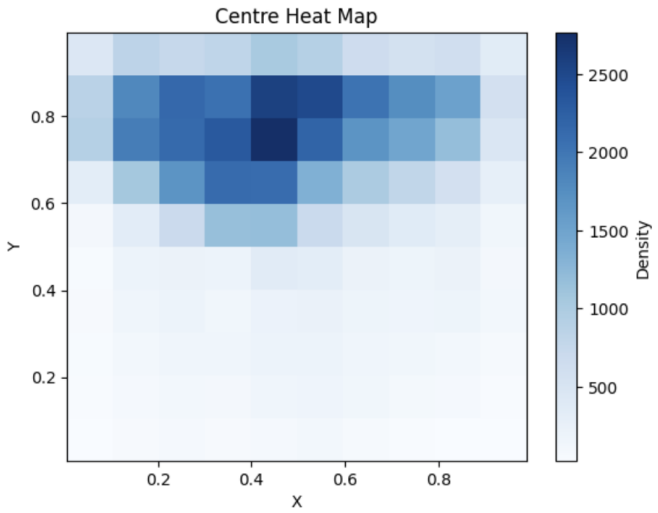


Figure 43: Centre Object heat map

For the Box Shape distributions, we can see some small similarity in D00 and D40 shape distributions, for the other distributions, we see the different between any groups, this suggest that the Wrong-classification between Labels may not be occurs.

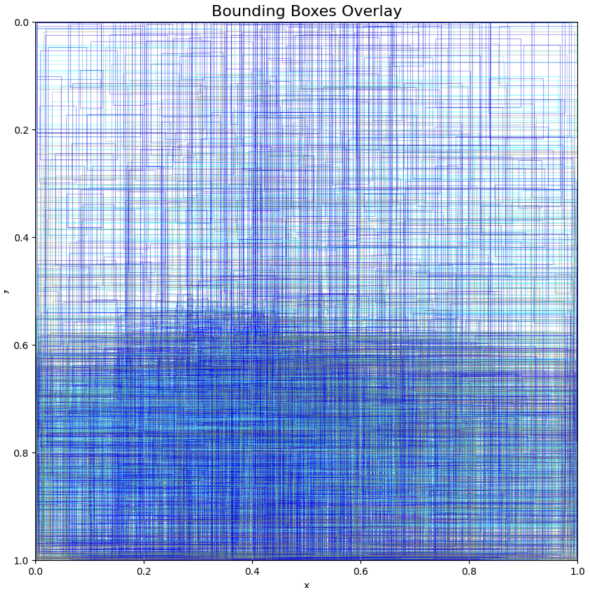


Figure 44: Box location

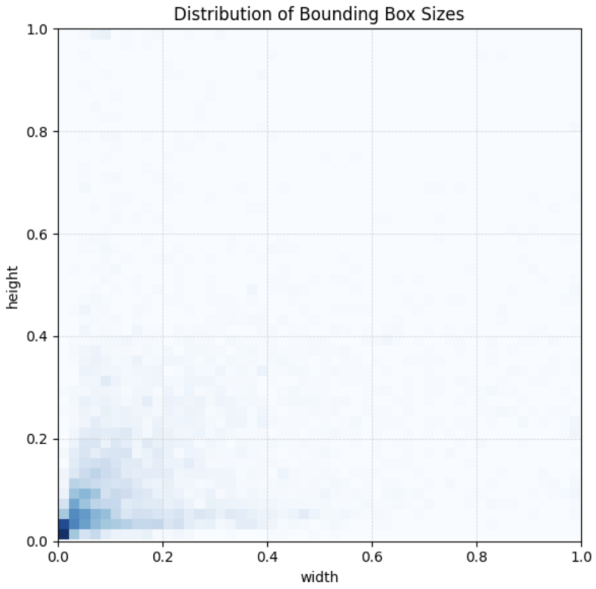


Figure 45: Distribution of Bounding Box Size

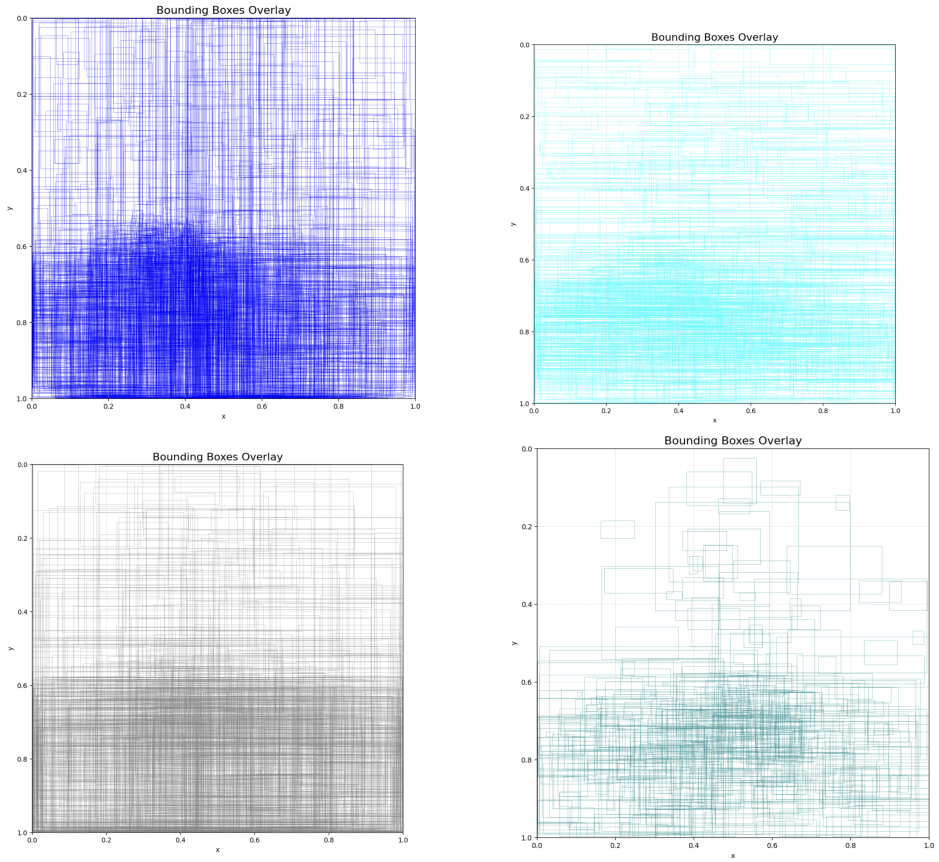


Figure 46: Bouding Boxes Overlay:
(a) D00 overlays, (b) D10 overlays, (c) D20 overlays, (d) D40 overlays

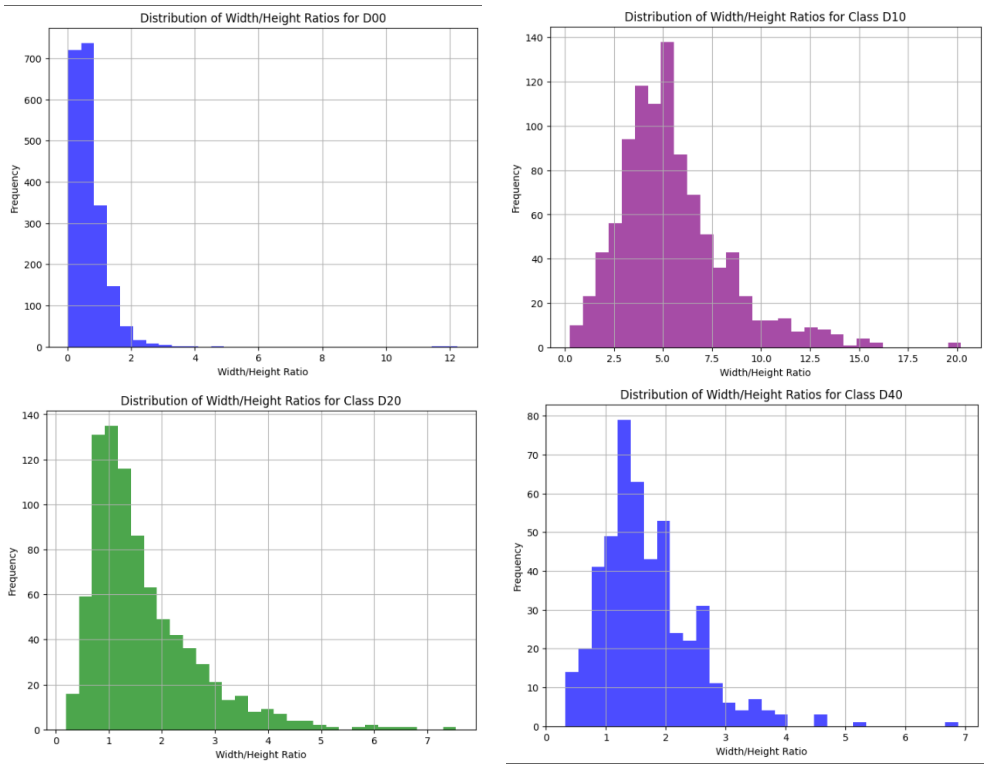


Figure 47: Distribution of Width/Height Ratios:
(a) D00, (b) D10, (c) D20, (d) D40

5.4 Model

YOLOv11n having the structure of YOLOv11, how-ever, it have limited access with the C3 Layer in C3K2 Feature Abtractions, the input and output for the Model

- Input: Input Image Size with 3 colors 640x640x3
- Output: S×S×B(5+C) tuple, where S is 3 grids size of 80x80, 40x40 and 20x20, B is number of Boxes, C is the number of Classes

The results then move to NMS to remove any similarity boxes.

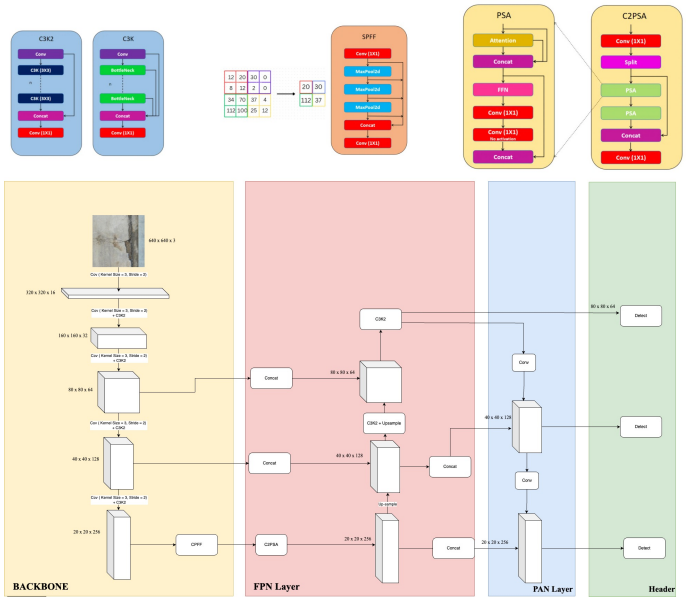


Figure 48: YOLOv11n model

5.4.1 Data Augment

1. **Mosaic (mosaic: 1.0)**: Mosaic augmentation is highly impactful in object detection tasks. It enables the model to generalize well by exposing it to objects in diverse contexts. Authors of YOLO estimate that using Mosaic can improve F1 score from 0.2-0.8% (using COCO datasets)
2. **HSV Adjustments (hsv-h, hsv-s, hsv-v):** These are essential for robustness to environmental lighting conditions. Scale (scale: 0.5):

   Scaling ensures the model detects objects at varying sizes. Flip (fliplr: 0.5):
3. **Horizontal flipping**

5.4.2 Model configuration

File size

Determine the input image size. Original image sizes are highly diverse, so resizing images to dimensions suitable for the model architecture is essential to ensure accuracy, training speed, and compatibility with available hardware. YOLOv11n suggests two input sizes: 1280x1280 and 640x640.

After experimentation, a trade-off must be made between training time and accuracy. It was observed that the accuracy difference between these two sizes is not significant, but the training speed with an input size of 640x640 is approximately 1.3 times faster than with 1280x1280. Therefore, all training processes for different YOLOv11n models will use 640x640 as the input size before feeding the data into the model.

Batch

Use the largest batch size that the hardware allows to train the model faster. Small batch sizes make the batch normalization process ineffective and should be avoided. The results of each batch are stored in RAM and aggregated after completing the training process for all batches. The larger the batch size, the more memory is consumed. A batch size of 32 is used during the training process.

Espoch

With examinate Espoch of 150, we found that there the overfitting occurs at espoch from 95-100, using Drop-out of 0.3, such that we used Espoch of 100

Dropout

Dropout is a regularization technique used in neural networks to prevent overfitting. It works by randomly "dropping out" (setting to zero) a fraction of neurons during training at each forward pass. For recommendations, the Dropout rate for YOLO is arround 0.3-0.5, the larger the model, the smaller the Dropout rate, such that we choose 0.3

Activate Function

AdamW have shown more potential in preventing update values to sub-optimal solutions than Adam, however, it will be longer in computing time than Adam. AdamW with lr0 = 0.001 and momentum=0.937

Cache

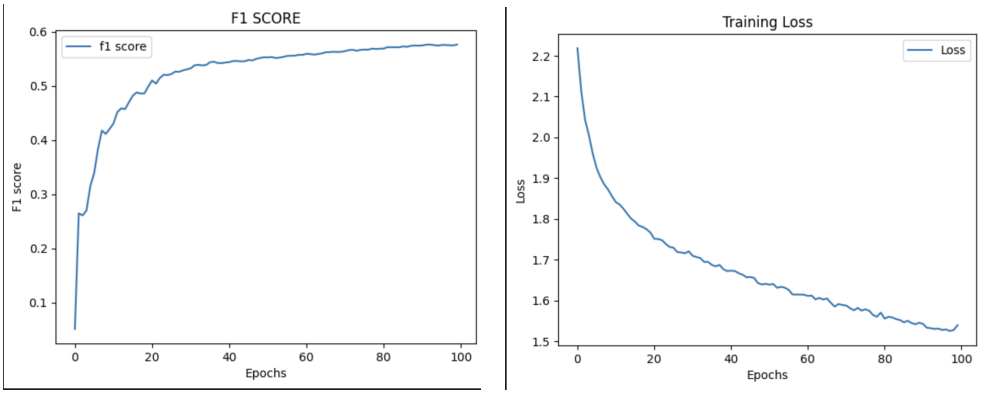
Allow cache for faster access to data

```
task: detect
mode: train
model: yolo11n.pt
data: model.yaml
epochs: 100
time: null
patience: 100
batch: 32
imgsz: 640
save: true
save_period: -1
cache: disk
device: cuda
workers: 8
project: null
name: train8
exist_ok: false
pretrained: true
optimizer: AdamW
verbose: true
seed: 0
deterministic: true
single_cls: false
rect: false
cos_lr: false
close_mosaic: 1
resume: false
amp: true
fraction: 1.0
profile: false
freeze: null
multi_scale: false
overlap_mask: true
mask_ratio: 4
```

Figure 49: Configation for Model

6 Evaluations

After the trainning process, we focus on F1 score alterations and Trainning Loss, from the espoch 95-100, the trainning may become over-fitting, as Loss become higher after the trainning



(a) F1 scores over espochs (b) Loss Funcion over trainning espochs

Despite the structure change to improve the overall Recall rate, the error in predict objects that is Background is high, espicially in D00.

From figure 46, observation can be make through distribution effection and bounding box distribution effects the overall prediction, take example for D00 labels, which having less Normal Distribution in Shape distribution and mostly gather at one location, the results get 0.48 predict on backgrounds, wherever, D20 having more random in both Shape and Bounding Box distribution, which performs better than other labels.

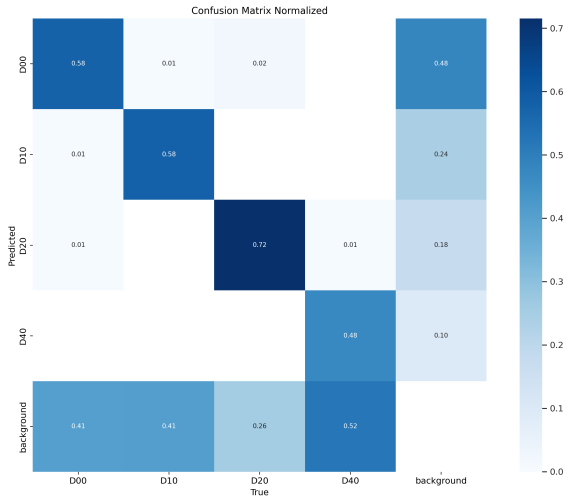


Figure 51: Normalize Confusion Matrices for D00, D10, D20, D40

Class	Images	Instances	Box(P)	R	mAP@50
all	3071	4434	0.593	0.535	0.564
D00	1087	2112	0.596	0.528	0.555
D10	633	969	0.581	0.531	0.554
D20	670	812	0.628	0.663	0.691
D40	301	541	0.568	0.416	0.455

Table 4: Performance metrics for the dataset, showing precision (Box(P)), recall (R), and mean average precision (mAP) across different classes.

To compare the performance between the 2 models, we will compare with relevant models from Phan Van Hung Master thesis results, we contructs in the same enviroment, for data Augement, Phan Van Hung Master thesis used HSV colors and MOISAC data augment

Class	Images	Instances	mAP@50	F1	Inference FPS
YOLOv11n (our)	7678	10232	0.592	0.644	294
YOLOv5s	7677	10856	0.574	0.632	119
YOLOv5l	7677	10856	0.609	0.645	83
YOLOv5x	7677	10856	0.604	0.658	70
YOLOv5x6	7677	10856	0.613	0.66	35

Table 5: Performance metrics for the test dataset, showing precision (Box(P)), recall (R), and mean average precision (mAP) across different classes.

7 Conclusion

7.1 Completed Tasks

- Theoretical research on Machine Learning, Neural Network as well as the object detection problem.
- Research some model such as Template Matching, R-CNN, Fast-CNN, Faster-CNN and YOLOv1, YOLOv2, YOLOv3, YOLOV4, YOLOv8, YOLOv10, YOLOv11.
- Study and utilize frameworks like PyTorch, OpenCV, and TensorFlow.
- Evaluate and analyze the performance metrics when using the YOLOv11 model on the RDD2022 dataset.

7.2 Limitations

- Data augmentation has not been optimized yet.
- Optimal configuration may not met, changing configuration costs a lot of time
- There are still many background predictions

7.3 Future plan

- Further explore data augmentation methods
- Apply more than one dataset.
- Develop additional models, especially two-stage detection models, to compare and evaluate performance metrics.
- Development of Time-Series model, application with fast-transportment in Highway asphalt detection to prevent un-wanted accident
- Develop Structure bone for Instance Detection and Rebuilds 3D objects detections for cracks in roads structure

Inference

1. Mkwata, R., & Chong, E. E. M. (2022). Effect of pavement surface conditions on road traffic accident-A Review. In E3S web of conferences (Vol. 347, p. 01017). EDP Sciences.
2. Richard Szeliski, Computer Vision: Algorithms and Applications 2nd Edition Priyanka, Yashwant Prasad Singh, Face Recognition Using PCA Based Eigenfaces
3. Priyanka, Yashwant Prasad Singh, Face Recognition Using PCA Based Eigenfaces
4. Phan Van Hung (2022). LUẬN VĂN THẠC SĨ, PHÁT HIỆN HƯ HỎNG TRÊN BỀ MẶT ĐƯỜNG NHỰA DỰA VÀO MẠNG NƠ-RON HỌC SÂU
5. Zaloshnja, E., & Miller, T. R. (2009, October). Cost of crashes related to road conditions, United States, 2006. In Annals of Advances in Automotive Medicine/Annual Scientific Conference (Vol. 53, p. 141). Association for the Advancement of Automotive Medicine.
6. Shlens, J. (2014). A tutorial on principal component analysis. arXiv preprint arXiv:1404.1100.
7. Girshick, R. (2015). Fast r-cnn. arXiv preprint arXiv:1504.08083.
8. Ren, S., He, K., Girshick, R., & Sun, J. (2016). Faster R-CNN: Towards real-time object detection with region proposal networks. IEEE transactions on pattern analysis and machine intelligence, 39(6), 1137-1149.
9. Hussain, M. (2023). YOLO-v1 to YOLO-v8, the rise of YOLO and its complementary nature toward digital manufacturing and industrial defect detection. Machines, 11(7), 677.
10. Redmon, J. (2016). You only look once: Unified, real-time object detection. In Proceedings of the IEEE conference on computer vision and pattern recognition.
11. Redmon, J., & Farhadi, A. (2017). YOLO9000: better, faster, stronger. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 7263-7271).
12. Farhadi, A., & Redmon, J. (2018, June). Yolov3: An incremental improvement. In Computer vision and pattern recognition (Vol. 1804, pp. 1-6). Berlin/Heidelberg, Germany: Springer.
13. Bochkovskiy, A., Wang, C. Y., & Liao, H. Y. M. (2020). Yolov4: Optimal speed and accuracy of object detection. arXiv preprint arXiv:2004.10934.
14. Reis, D., Kupec, J., Hong, J., & Daoudi, A. (2023). Real-time flying object detection with YOLOv8. arXiv preprint arXiv:2305.09972.
15. Wang, A., Chen, H., Liu, L., Chen, K., Lin, Z., Han, J., & Ding, G. (2024). Yolov10: Real-time end-to-end object detection. arXiv preprint arXiv:2405.14458.
16. Khanam, R., & Hussain, M. (2024). Yolov11: An overview of the key architectural enhancements. arXiv preprint arXiv:2410.17725.
17. Vũ Hữu Tiệp, Machine Learning cơ bản