UNDERGRADUATE THESIS

AUTOMATIC TRAFFIC ACCIDENT DETECTION USING DEEP LEARNING METHODS

DETEKSI OTOMATIS KECELAKAAN LALU LINTAS MENGGUNAKAN METODE DEEP LEARNING

IBNU BORISMAN FARREL
20/457751/PA/19789

ELECTRONICS AND INSTRUMENTATION INTERNATIONAL UNDERGRADUATE PROGRAM STUDY PROGRAM
DEPARTMENT OF COMPUTER SCIENCE AND ELECTRONICS
FACULTY OF MATHEMATICS AND NATURAL SCIENCES
GADJAH MADA UNIVERSITY
2024

HALAMAN PENGESAHAN

SKRIPSI

AUTOMATIC TRAFFIC ACCIDENT DETECTION USING DEEP LEARNING METHODS

Telah dipersiapkan dan disusun oleh

Ibnu Borisman Farrel (IUP)

20/457751/PA/19789

Telah dipertahankan di depan Tim Penguji

pada tanggal 30 Agustus 2024

Susunan Tim Penguji

[illegible]

M. Idham Ananta Timur, S.T., M.Kom.
Ketua Penguji

[illegible]

Prof. Dr.techn. Ahmad Ashari, M.I.Kom.
Anggota Penguji

[illegible]

Dr. Raden Sumiharto, S.Si., M.Kom
Pembimbing

Mengetahui,
a.n. Dekan FMIPA UGM
Wakil Dekan Bidang Pendidikan, Pengajaran
dan Kemahasiswaan

UNIVERSITAS GADJAH MADA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM

[illegible]

Prof. Drs. Roto, M.Eng., Ph.D.
NIP. 196711171993031020

STATEMENT OF ANTI-PLAGIARISM

I, the undersigned:

Name	: Ibnu Borisman Farrel
NIM	: 20/457751/PA/19789
Enrolment year	: 2020
Study Program	: Electronics and Instrumentation (IUP)
Faculty	: Mathematics and Natural Sciences

Hereby declare that this research has never been for the purpose of obtaining a degree submitted to any other institution and, to the extent of my knowledge, does not contain contributions written and published by any other author prior to the submission of this Research unless expressly referred to otherwise within its contents and included in the Bibliography. Therefore, I declare that this Research is ant plagiarism in any content, and inthe future, if it is proven to be plagiarised from someone else's work and/or intended to be submitted work or opinion that is from someone else's work, the author will accept an academic sanction and/or other applicable sanction.

Yogyakarta, 17 September 2024

10000
METERAI TEMPEL
F1E75AMX001533091

[illegible]

Ibnu Borisman Farrel

20/457751/PA/19789

FOREWORD

Praise and gratitude to the presence of Allah SWT for His abundance of gifts and guidance so that the author can complete the undergraduate thesis with the title "Automatic Traffic Accident Detection Using Deep Learning Methods".

This undergraduate thesis aims as one of the fundamental requirements to fulfil the academic requirements of the Electronics and Instrumentation Undergraduate Study Program. The preparation of this undergraduate thesis relies on the various parties that came to aid in the completion process. Therefore, the author would like to express his deepest gratitude and thanks to:

1. Allah Subhanahu Wa Ta'ala, for his affluence of strength, gifts, convenience, health, and faith that aids through the whole process of completing various series of undergraduate thesis smoothly and successfully.
2. Prayers from Jaenal Fanani, Rinta Duppi, family, siblings, and friends that provide physical, spiritual, and mental support.
3. Drs. Yohanes Suyanto, M.Ikom. as Head of the Electronics and Instrumentation Study Program, Gadjah Mada University.
4. Dr. Raden Sumiharto, S.Si., M.Kom Who has taken his time and patience to guide and provide thoughts, ideas, and blessings so that the author can complete this undergraduate thesis as excellently as possible.
5. Lecturers of the Electronics and Instrumentation study program who have guided and imparted their knowledge to the author to further his studies within this undergraduate thesis and complete it.
6. My cherished one who always gives spirit and endless support to the author
7. Rafi Satriawan and Muhammad Awfa Farhan and all friends in Depok who are always present when the writer needs advice and are always there to receive it complaints from the author during the research process
8. Friends that are family and places that will never be forgotten, namely; Poetry, Voltshot, Dio, Debarix, Zerror, Angkel Firman, Lil Nas, Jengat, Brofidz, Abdoel, Aweme, Akainu, The Gardens, 911 SCH, The Q and Habit

for their constant support and guidance for the author that have helped to complete this research process.

9. The Gamaforce team for the support and togetherness in completing this undergraduate thesis.
10. The entirety of the ELINS family that are unable to be mentioned one by one. The author expresses his appreciation and gratitude for the support, assistance, togetherness in completing this undergraduate thesis.
11. As well as all parties who have helped and supported in report writing process that are unable to be mentioned one by one.

The author realizes that there are still many mistakes and errors in this undergraduate thesis due to the limitations of knowledge and abilities of the author and would like to express and expect constructive comments, ideas and suggestions from all parties. As a result, the author expects that this undergraduate thesis will be useful for all readers in the present and future to continue this study.

Yogyakarta, 3th of August, 2024

Author,

[illegible]

Ibnu Borisman Farrel

ABSTRACT

Automatic Traffic Accident Detection Using Deep Learning Methods

By
Ibnu Borisman Farrel
20/457751/PA/19789

Traffic accidents in urban areas pose significant concerns, often resulting in severe consequences. Traditional methods of accident detection face challenges like delays and inefficiencies, particularly in remote locations. This research proposes an innovative solution leveraging deep learning and computer vision techniques to automate traffic accident detection using CCTV footage with limited dataset.

The research utilizes advanced models such as YOLOv8 for object detection and various Convolutional Neural Network (CNN) architectures, including MobileNetV2, ResNet-50, and Inception V4, for anomaly classification. The proposed framework efficiently detects and classifies accidents in traffic road while ensuring minimal computational load. These results highlight the potential of the proposed method to significantly improve road safety and traffic management, contributing to the development of smarter and safer urban environments.

The system processes video feeds from CCTV cameras to detect potential accidents, using YOLOv8 to identify objects and track their movements. Detected anomalies indicative of traffic accidents are then classified using the CNN architectures such as MobileNet V2, ResNet50 and Inception V4 to confirm their presence. MobileNetV2, designed for resource-constrained environments, ensures low computational load while providing high accuracy, making it particularly suitable for large-scale deployment. Experimental results demonstrate the effectiveness of the proposed approach. YOLOv8 achieves a speed report of 27.85 milliseconds per frame, enabling detection and tracking process fast. MobileNetV2 achieves an accuracy score of 94%, showcasing its ability to classify traffic accidents accurately while maintaining low computational requirements. ResNet-50 and Inception V4 models achieve accuracy scores of 96% and 80%, respectively, demonstrating strong performance in anomaly classification tasks.

Keyword: CNN, YOLOv8, Traffic Accidents, Computer Vision

INITISARI

DETEKSI OTOMATIS KECELAKAAN LALU LINTAS MENGGUNAKAN METODE DEEP LEARNING

By
Ibnu Borisman Farrel
20/457751/PA/19789

Kecelakaan lalu lintas di daerah perkotaan menimbulkan kekhawatiran yang signifikan. Metode tradisional untuk mendeteksi kecelakaan menghadapi tantangan seperti keterlambatan dan ketidakefisienan. Penelitian ini mengusulkan solusi inovatif dengan memanfaatkan teknik pembelajaran mendalam dan *Computer Vision* untuk mendeteksi secara otomatis kecelakaan lalu lintas menggunakan rekaman CCTV.

Penelitian ini menggunakan model canggih untuk klasifikasi anomali, seperti YOLOv8 untuk deteksi objek dan berbagai arsitektur Convolutional Neural Network (CNN), termasuk MobileNetV2, ResNet-50, dan Inception V4. Kerangka kerja yang diusulkan secara efisien mendeteksi dan mengklasifikasikan kecelakaan di jalan raya dengan memastikan beban komputasi minimal dan data yang terbatas. Beban komputasi rendah dan akurasi tinggi dari sistem yang diusulkan menjadi solusi untuk meningkatkan sistem pemantauan lalu lintas, memungkinkan waktu respons yang lebih cepat, dan berpotensi mengurangi dampak kecelakaan lalu lintas.

Sistem ini memproses video yang bersumber dari kamera CCTV untuk mendeteksi potensi kecelakaan, menggunakan YOLOv8 untuk mengidentifikasi objek dan melacak pergerakannya. Anomali yang terdeteksi yang mengindikasikan kecelakaan lalu lintas kemudian diklasifikasikan menggunakan arsitektur CNN seperti MobileNet V2, ResNet50, dan Inception V4 untuk memastikan keberadaannya. MobileNetV2 yang dirancang untuk memastikan beban komputasi rendah sambil memberikan akurasi tinggi, sehingga cocok untuk penerapan skala besar. Hasil eksperimen menunjukkan efektivitas pendekatan yang diusulkan. YOLOv8 mencapai laporan kecepatan 27,85 milidetik per frame, memungkinkan proses deteksi dan pelacakan berjalan cepat. MobileNetV2 mencapai skor akurasi 94%, menunjukkan kemampuannya untuk mengklasifikasikan kecelakaan lalu lintas dengan akurat sambil mempertahankan kebutuhan komputasi yang rendah. Model ResNet-50 dan Inception V4 mencapai skor akurasi masing-masing 96% dan 80%, menunjukkan kinerja yang kuat dalam tugas klasifikasi anomali.

Keyword: CNN, YOLOv8, Kecelakaan Lalu Lintas, *Computer Vision*

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I: INTRODUCTION

1.1 Research Background

The Traffic accidents in Indonesia pose a significant concern, often leading to both loss of life and damage to property. Prompt and efficient initial care is crucial in such situations, given that every moment is critical for the well-being of the victims Prahmawati (2021). Traditional approaches to addressing accidents, such as dialing emergency numbers, may encounter challenges like delays and are challenging to execute when the incident occurs in a remote location. The number of accidents on traffic may vary from time to time, giving its dataset relatively limited. Due to the limited dataset on accident, imbalanced dataset may occur during this research. Despite this, surveillance cameras (CCTV) can be employed for accident monitoring. Nevertheless, overseeing traffic incidents through widespread CCTV usage in a country like Indonesia demands substantial manpower and presents difficulties in sustaining the implementation process. In the year 2022, there were approximately 204,450 road traffic accidents reported in Indonesia, signifying a notable rise when compared to the preceding year. During that same year, Indonesia witnessed a substantial number of traffic-related fatalities, with around 27,530 individuals losing their lives on the country's streets and highways.

Improvements in consumer-grade technologies like video cameras have significantly enhanced traffic monitoring systems. Nowadays, many State Traffic Management Centers use live CCTV feeds to manage and respond to various highway incidents. However, these modern systems are expensive to maintain due to their manual operation, resulting in slower incident detection and response times. Additionally, the high volume and quality of data from these systems can be challenging for operators to handle. Thus, there's a need for scalable solutions that can swiftly process camera data to identify and address traffic incidents or irregularities. This study introduces a method that efficiently detects anomalies or incidents in CCTV videos using advanced deep learning and computer vision models.

As urban traffic systems continue to grow, the need for comprehensive traffic management and accident alert systems has become integral to urban infrastructure development. Detecting unusual traffic patterns is a key aspect of video analysis, capturing attention from both academic and industry circles. The intricate nature of traffic scenes and the varying camera perspectives present significant challenges in this area. Anomalies in traffic are infrequent compared to regular activities, making it essential to devise intelligent computer vision techniques for automatic anomaly detection in videos. A practical anomaly detection system aims to promptly identify deviations from standard patterns and pinpoint when these anomalies occur. Thus, anomaly detection can be viewed as a broad video analysis task that distinguishes unusual events from regular activities.

A solution to this problem is to use Artificial Intelligence deep learning techniques. such as the development of deep learning architecture, which is capable of automating accident detection in CCTV images Tamagusko (2018). With this technology, an automatic rapid response system capable of detecting traffic accidents and providing initial warnings to first responder institutions can be developed. Deep learning techniques involve the application of advanced neural networks and machine learning algorithms to identify and respond to traffic accidents more effectively. By analyzing various data sources, such as video feeds, AI systems can swiftly identify patterns and anomalies associated with accidents, leading to quicker response times and potentially reducing the severity of incidents. Advanced deep learning techniques come with multiple benefits, such as improved precision in detecting accidents, the capacity to handle a wide range of data sources, and the flexibility to adapt to varying environmental conditions and traffic situations. They can be seamlessly incorporated into a range of systems, like those used for traffic management and surveillance, enabling the real-time identification of accidents and prompt notifications. This, in turn, results in quicker emergency responses and an overall enhancement of road safety.

Several studies in this area have demonstrated encouraging findings. For instance, Tian & Kim (2023) introduced a technique to enhance Faster R-CNN's performance in identifying vehicles by pre-classifying the weather conditions depicted in traffic images. They utilized dark channel prior (DCP) along with block-matching and 3D filtering (BM3D) algorithms to enhance image quality and reduce noise based on the weather conditions. Their Faster R-CNN model incorporated VGG16 and ResNet101 for improved feature extraction and accuracy. Meanwhile, Luo et al. (2021) employed Neural Architecture Search (NAS) to optimize feature extraction for vehicle detection within Faster R-CNN. They also implemented feature enrichment to ensure proper detection of vehicles that appear small or are obscured by other objects. Additionally, they enhanced image quality using the Retinex-based image adaptive correction (RIAC) algorithm. Another research have been conducted using the development of deep learning-based models by several researchers. Robles-Serrano (2021) have developed an accident detection model using the Inception V4 architecture and obtained accuracy results of 97%. However, deep learning architectures tend to have a fairly high computational load due to the complexity of the architecture. This presents a challenge to detect accidents in real-time. In this research, the computational load will be more considered to develop an architecture that has the ability to detect real-time accidents. The development of an architecture that has a light computing load will make the implementation of accident detection using CCTV cameras capable of being implemented on a large scale.

1.2 Problem Statement

Traffic accident that are recorded can be quite rare and limited to find its data, therefore we are dealing with an imbalanced dataset. There are more data on non-accident dataset than accident dataset. Not to mention, the complexity of deep learning architectures often results in a significant computational load which can be challenging to detect traffic accidents and can slow down the processing of data from traffic cameras. The research is to obtain a high accuracy model and minimal computational load with limited dataset.

1.3 Research Objective

Due to the limited dataset, the aim is to develop a deep learning architecture that deals with imbalanced dataset. To also create an automated system that can efficiently detect traffic accidents within the vast pool of video recordings and images obtained from CCTV cameras while ensuring a minimal computational load. Lastly is to compare the performance of MobileNet V2, ResNet50 and Inception V4 model by using multiple deep learning architecture.

1.4 Research Scope

a. The research focuses on detecting and classifying accidents in CCTV videos from sources like Twitter, YouTube, and public broadcasts using the YOLOv8 architecture for object detection and CNN with the Simple Moving Average (SMA) method for anomaly detection.
b. Anomalous frames will be classified using MobileNetv2, Inception v4, and ResNet-50 models, trained on a limited dataset.
c. The study is limited to accident detection in specific datasets and video sources, utilizing the YOLOv8 and CNN models for the task.

1.5 Research Benefits

The development of deep learning techniques with a high accuracy model using an imbalanced dataset that is going to make accident traffic detection able to be implemented in a bigger scale. By implementing this technology, it can achieve a much faster and efficient process of the first response for traffic accidents and is expected to reduce the number of fatalities in traffic accident.

CHAPTER II: LITERATURE REVIEW

A method that has been proposed by Robles-Serrano (2021) in Table 2.1 No. 1 using deep learning architecture to automatically detect traffic accident from videos. The first part uses a modified Inception V4 architecture to extract a vector of visual characteristics, while the second part processes these characteristics using a recurrent component ConvLSTM layer-based neural network architecture to extract temporal information from the input data. to extract the temporal component associated with the occurrence of the event. Finally, a binary classifier based on support vector machines is applied to distinguish between the occurrence of an anomaly and a common event. The proposed method assumes that traffic accident events are described by visual features occurring through a temporal way. The visual and temporal features are learned in the training phase through convolution and recurrent layers using built-from-scratch and public datasets. An accuracy of 98% is achieved in the detection of accidents in public traffic accident datasets, showing a high capacity in detection independent of the road structure.

A research that was done by Divakar Yadav (2023) in Table 2.1 No. 2, which he utilize the use of The Joint Detection and Embedding model (JDE) for Pedestrian detection and tracking and ResNet 101 (R101) is also used for the appearance feature. The datasets are taken from the UCSD anomaly detection dataset which is available online. The Dataset is then divided into two clips, which both clips have different amount of training and testing video samples. To detect the behavior pattern, Convolutional Auto Encoder (CAE) is used in the training model. The CAE includes a denoising filter which makes it a denoising filter encoder. With this, noise has to be added into the dataset for the CAE input. The Convolutional Auto encoder (CAE) has a straightforward architecture: input undergoes encoding in the encoder layer, followed by decoding in the decoder layer. The decoder reconstructs the data to closely resemble the original input. The acquired data is then compared to the original data; if there is less reconstruction error, the sample is deemed normal; if there is a significant reconstruction error, the sample is deemed abnormal. This results in an average precision of 91.63% accuracy which falls under the category of a good model performance.

A research done by Sardar Waqar Khan (2022) in Table 2.1 No. 3, proposed the use of CNN to detect anomaly in traffic surveillance video. This research is centered on using a convolutional neural network (CNN) to identify anomaly in traffic accident videos. To gather the necessary training data, the research employs a video-based traffic surveillance system (VTSS) with added features, known as VCSS. Over time, VTSS systems have seen improvements in data collection, storage, and cleaning, leading to a wealth of available online content. The aim of this research is to detect traffic accidents by analyzing frame images from VTSS videos, providing assistance to accident victims. The CNN model utilizes a convolutional autoencoder stack to learn spatial patterns from images. Training was conducted on the VAID dataset, which includes 1360 images, and validation was performed using a set of 30 videos. A significant challenge faced during the research was label flickering between images, which had a detrimental effect on accuracy. To overcome this, author paper 3 used a rolling prediction average algorithm was implemented alongside the CNN model to improve video classification. The results of the research were promising, achieving an accuracy rate of 82% when tested on videos captured by high-resolution cameras in VTSSs.

By using an ensemble of deep learning and decision tree, Armstrong Aboah (2021) in Table 2.1 No. 4 suggested a method for Traffic anomaly detection. The vehicle detection model is carried out using YOLOv5 with CSPDarknet53 backbone. First the Videos go through an automated sorting process. Then, a concurrent process measures background features and detects items in the foreground. After that, possible abnormalities are found in the background photos by processing them using a vehicle object detector. Lasty, A decision tree is then utilized to identify, and categories false anomalies based on pre-established standards. The start and end times of anomalies are calculated by superimposing the findings from the foreground and anomaly detections. The researcher is able to achieve a f1 score of 0.8571 which indicates that this model has been successful for detecting anomalies in the video. An RMSE of 101.0071 indicates that this model is not able to perform detection from the video that are too small or far away from the camera.

Research done by Yuxiang Zhao (2021) in Table 2.1 No. 5 uses two two-stage as a detector for traffic anomaly detection. The first stage uses Faster R-CNN with SENet-152 for the main detector. The second stage is for an auxiliary to minimize any fail detection by using Cascade R-CNN with CBResnet-200. Further applying Feature Pyramid Network (FPN) to have a high feature map. To filter out any suspicious event, the researcher uses pixel tracking by observing and comparing the objects position using IOU algorithm. If the results of two region in an intersection is above 0.5, the start timestamp will be updated of an anomaly. The Experiment results in a F1 score of 0.9524 with a time error of 5.3080 seconds which demonstrated a great model.

By comparing three different methods to find the most optimal and accurate result for vehicle counting. In Table 2.1 No. 6, Azizi Abdullah (2020) uses 3 models which are YOLOv3 with DarkNetv19, Faster RCNN with Resnet101 and SSD Inception. Using bounding boxes to detect vehicles for every n-frames with the use of transfer learning. The trajectory of the vehicle corner point is then extracted through n frames. Lastly the three algorithm is applied to count the number of vehicles on the street. Out of the three algorithms, YOLOv3 with Darknet19 provide with the best result with an average accuracy from 66.29% to 80.90% before and after retraining using a data annotation for the counting system. However, The YOLOv3 DarkNet19 model shows decreased performance during low-light conditions in both morning and night scenarios when applied to the custom dataset. To address this issue, it is suggested to enhance the model's capabilities by retraining it using a custom dataset specifically tailored to poor illumination environments. This involves annotating data using a dedicated tool and implementing transfer learning with weight training initialization. The outcome is a notably improved accuracy in object counting.

YOLO algorithm is proposed in Table 2.1 No. 7 by Amir Mahmud Husein (2024) for multi-vehicle tracking and detection on traffic surveillance. The main focus is to detect vehicle classes such as motorcycles, buses, trucks, and cars or special vehicles such as ambulances and others. The dataset is collected from CCTV recordings of the traffic at Juanda Katamso intersection, Gatot Subroto Intersection, and Uniland Intersection all located in Indonesia. Image extraction is performed from video with a maximum image of 250 for each video. Using 13 vehicle classes for labeling vehicle objects in the extracted image and will be used for training, validation, and test results with a percentage of 70%, 20% and 10% respectively. Challenges occur during the process of detecting vehicle due to the inconsistent data labelling process and the proximity of the camera to the detected object affects the classification of vehicles, despite the high accuracy of detection. CPU 1 (Intel Core i3 8th Gen with a Hard Disk Drive) and CPU 2 (Intel Core i3 7th Gen with a Solid State Drive) were utilized for vehicle detection employing the YOLOv4 model. Upon experimentation, it was determined that the inference time on GPU amounted to 9 minutes and 23 seconds. Conversely, CPU 1 required 2 hours, 4 minutes, and 30 seconds for inference, while CPU 2 demanded 2 hours, 31 minutes, and 1 second. The YOLOv4 model exhibited a mean Average Precision (mAP) value of 77.88%.

Table 2. 1 Literature Review

No	**Researcher**	**Title**	**Description**
1	Sergio Robles-Serrano, German Sanchez-Torres, John Branch-Bedoya	Automatic Detection of Traffic Accidents from Video Using Deep Learning Techniques	Proposes a method combining spatial dynamic attention and recurrent neural networks to predict accidents in first-person view videos.
2	Divakar Yadav, Arti Jain Saumya Asati, and Arun Kumar Yadav	Video Anomaly Detection for Pedestrian Surveillance	The paper presents a deep learning model abnormal pedestrian videos, vital for security setups.
3	Sardar Waqar Khan, Roobaea Alroobaea, Jawaid Iqbal	Anomaly Detection in Traffic Surveillance Videos Using Deep Learning	The research paper explores deep learning methods for spotting anomalies in traffic videos, including the GKIM model for unusual objects

			and pedestrian movements, automated incident detection.
4	Armstrong Aboah, Maged Shoman, Vishal Mandal, Yaw Adu-Gyamfi	A Vision-based System for Traffic Anomaly Detection using Deep Learning and Decision Trees	The paper introduces a vision-based method, combining deep learning and decision trees, to swiftly detect irregularities collisions using YOLO V5.
5	Yuxiang Zhao, Wenhao Wu, Yue He, Yingying Li, Xiao Tan, Shifeng Chen	Good Practices and A Strong Baseline for Traffic Anomaly Detection	The paper introduces a framework for identifying traffic anomalies in videos.
6	Azizi Abdullah	Vehicle Counting using Deep Learning Models: A Comparative Study	The research paper examines deep learning models for vehicle counting, comparing Faster R-CNN, SSD, YOLO, and a basic tracking technique.
7	Amir Mahmud Husein1, Kevi Noflianhar Lubis, Daniel Salim Sidabutar, Yansan Yuanda, Kevry, Ashwini Waren	Computer Vision-Based Intelligent Traffic Surveillance: Multi-Vehicle Tracking and Detection	The paper introduces a real-time traffic monitoring system using YOLOv4 to identify vehicle categories in highway videos.

CHAPTER III: THEORETICAL BASIS

3.1 Traffic Accidents in Indonesia

According to Law Number 22 of 2009 regarding Traffic and Road Transport, traffic accident is an unexpected and unintentional event on the road involving a vehicle with or without other road users which results in human casualties or property loss. According to Indonesian Transportation Society (MTI) that is published by The Voice Of Indonesia (VOI), the number of traffic accidents in Indonesia in 2023 reached 116,000 cases, marking a 6.8 percent increase from the previous year. The significant number of traffic accidents in East Java is attributed to the high volume of vehicles, particularly motorcycles.

3.2 Traffic CCTV Indonesia

The traffic CCTV systems in Indonesia generally store images in commonly used formats such as JPEG and have a resolution of 1920 × 1080 pixels or higher. The captured images have a color depth of 24 bits per pixel (bpp) and a wide standard aspect ratio of 16:9, offering a wide view. Additional metadata, including timestamps, geolocation information, and camera IDs, is also present in Indonesian traffic CCTV footage.

3.3 Artificial Intelligence

Artificial Intelligence (AI) refers to the capabilities of a system computers to execute tasks that usually require human intelligence that includes pattern recognition and decision-making Russell (2016). AI generally involves components such as input data, pre-processing, models representing system intelligence, learning algorithms, inference/decision making, output generation, and feedback cycles for continuous improvement. Input data is processed and used to train a model, which can then make predictions or decisions based on the new data.

3.4 Deep Learning

Deep Learning is a sub-field of machine learning focused on neural network training to learn and make predictions from volume 3 big data Sarker (2021). At its core, deep learning uses complex algorithms inspired by the structure and function of the human brain. The algorithm consists of several layers of artificial neurons that are interconnected, forming what is generally referred to as a neural network. The foundation of deep learning lies in its ability to automatically learn hierarchical representations of data over layered networks. Each layer processes input data and extracts abstract and meaningful features. By iteratively adjusting neuron weights and biases based on training data, deep learning is able to deal with unstructured, high-dimensional data such as images, audio, and text.

3.5 Convolutional Neural Network (CNN)

The Convolutional Neural Network (CNN) is a sophisticated deep learning framework designed specifically for analyzing spatial data, inspired by the visual processing mechanisms found in living organisms Lindsay (2020). Comprising distinct layers such as the output layer, fully connected layer, and convolution and pooling layers, CNN begins its operation with the convolution layer, which directly processes input images. Within this layer, a process known as convolution occurs, involving the multiplication of elements within a filter or kernel with corresponding elements in the input image matrix, followed by summation. As the filter traverses the image, a scalar product is computed at each position, generating a matrix termed a feature map. This feature map can be computed using the following Equation 3.1.

$$n_{Out} = \left(\frac{n_{in} - k + 2p}{s}\right) + I \qquad (3.1)$$

Where n_{Out} is the size of the feature map, n_{in} is the size of the input matrix, k is the size of the filter matrix, p is the size of the padding, and s is the stride. The equation for the convolution operation is shown in 3.2.

$$S(i,j) = (I \times K) = \sum m \sum n\, I\ (i-m, j-n)\, K(m,n) \tag{3.2}$$

Where $S(i,j)$ is the convolution result function, I is the input, K is the kernel, (i,j) is the output pixel index, and (m,n) is the kernel pixel index. Each feature map unit has a connection to the previous layer. Figure 3.1, architecture that was developed by visualizes the connections between layers in a CNN.

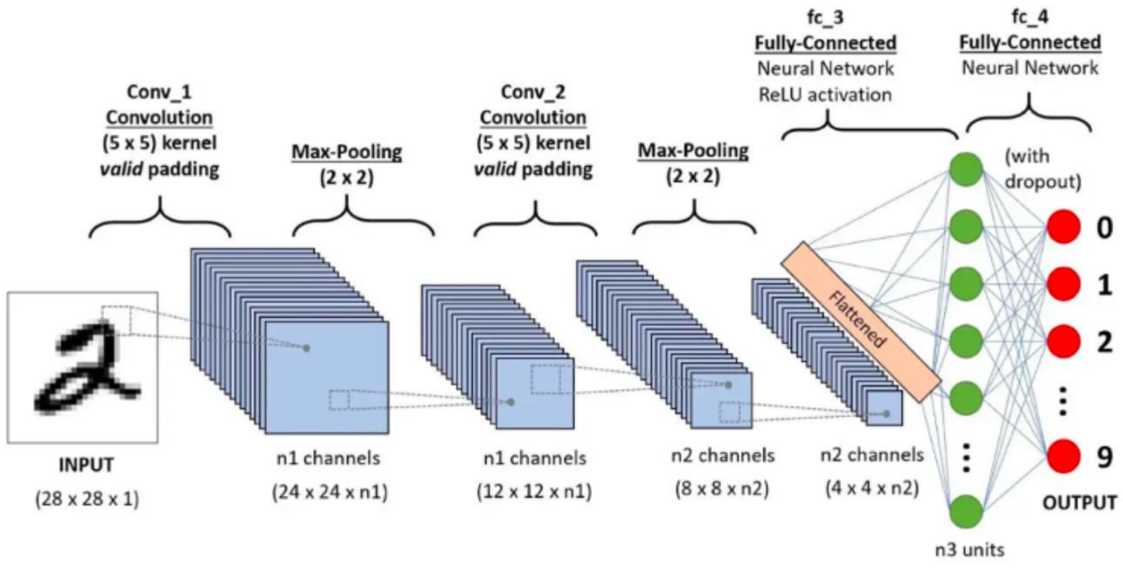


Figure 3. 1 Visualization of connections between layers in CNN (Lindsay, 2020)

The input and kernel will be combined to generate a new feature map, which will then undergo processing using a non-linear activation function. The convolution layer's parameter sharing feature helps reduce model complexity. The results from this layer will feed into the pooling layer, simplifying them into a single output. Finally, the fully connected layer will summarize the model's predictions based on these results.

3.6 Object Detection

Object detection is a vital technology in digital imaging, aiming to accurately identify and locate the boundaries of objects within an image Dong *et al*, (2013). This process goes beyond mere identification, also involving the classification of objects into various categories based on their visual attributes. By enabling machines to interpret visual data, object detection facilitates informed decision-making.

3.7 You Only Look Once (YOLO)

YOLO (You Only Look Once) stands out for its ability to detect objects in real-time. Its architecture comprises a CNN that conducts both object localization and classification concurrently. Unlike conventional object detection methods, which rely on classifiers, YOLO approaches detection through a regression task applied to distinct bounding boxes and associated class probabilities. Figure 3.2 illustrates the architecture of YOLO.

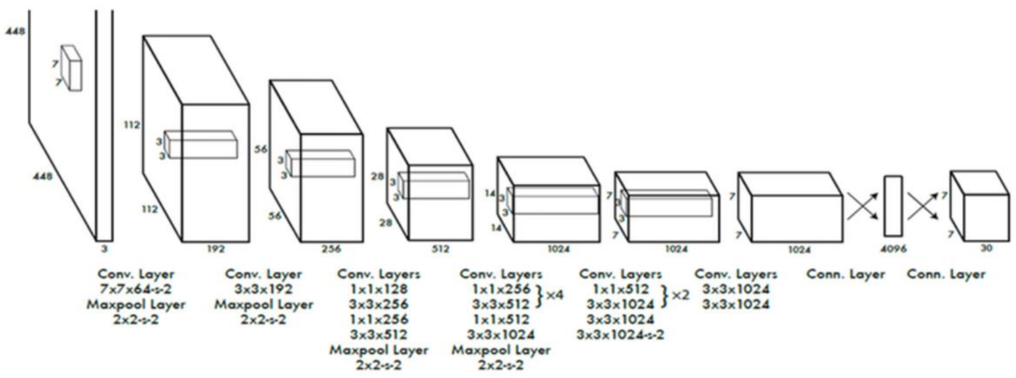


Figure 3. 2 YOLO architecture (Joseph Redmon *et al*., 2016)

The YOLO algorithm utilizes a convolutional neural network (CNN) structure to execute object detection by partitioning the image into a grid composed of cells typically sized at 7 × 7 or 13 × 13. Each cell is responsible for predicting the quantity of bounding boxes; for instance, in a 7 × 7 grid, there are typically two bounding boxes predicted per cell, thereby requiring each cell to predict two bounding boxes. Each bounding box is described using five components: firstly, the coordinates of its center point (x, y), followed by its width and height as the second and third element. The fourth element denotes the confidence score, indicating the presence of an object within the bounding box. Lastly, the fifth element represents the probability of each class's existence within the bounding box.

The YOLO model incorporates a CNN structure comprising multiple convolutional layers succeeded by a fully connected layer. These convolutional layers generate a detailed feature map with spatial details from the input image. Subsequently, the fully connected layer utilizes these features to forecast bounding box coordinates and class probabilities for every grid cell. To ensure coherence, softmax activation is applied to predict class probabilities, ensuring they sum up to one for each bounding box within the cell.

3.8 YOLO V8

YOLOv8 is an advanced pre-trained object detection model known for its high accuracy and efficiency which was done by Y. Swathi and Manoj Challa (2024). It features a streamlined architecture with a Backbone for extracting features, a Neck for processing features at multiple scales, and a Head for detecting objects. YOLOv8 incorporates advanced techniques like spatial pyramid pooling and path aggregation networks, resulting in improved performance.

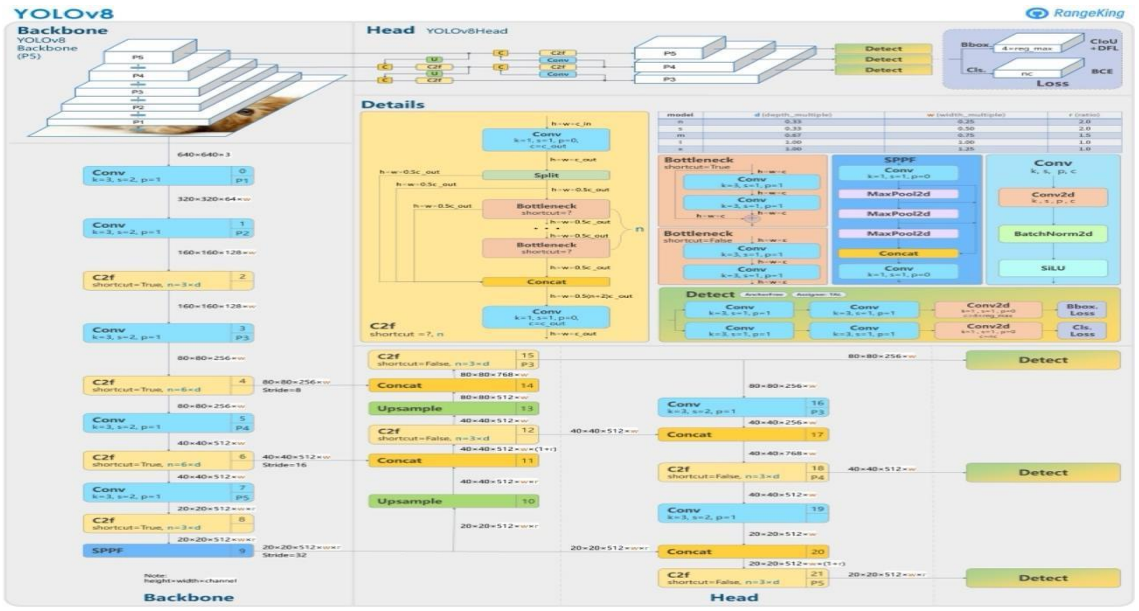


Figure 3. 3 YOLO V8 architecture (Y. Swathi and Manoj Challa, 2024)

The YOLOv8 architecture, illustrated in figure 3.3, incorporates several essential components for object detection tasks. The Backbone consists of a series of convolutional layers that extract important features from the input image. The SPPF layer and subsequent convolution layers process features at different scales, while the Upsample layers enhance the resolution of the feature maps. The C2f module merges high-level features with contextual information to improve detection accuracy. Finally, the Detection module utilizes a combination of convolutional and linear layers to map the high-dimensional features to output bounding boxes and object classes.

3.9 MobileNet V2

MobileNet is a convolutional neural network architecture specifically designed for mobile and embedded devices with limited computational resources Andrew G *et al*, (2017). MobileNet consists of two layers, in which its model is based on depth-wise separable convolutions and consist of two main components that is made up of depth-wise convolution and 1 X 1 pointwise convolutions shown in Figure 3.4.

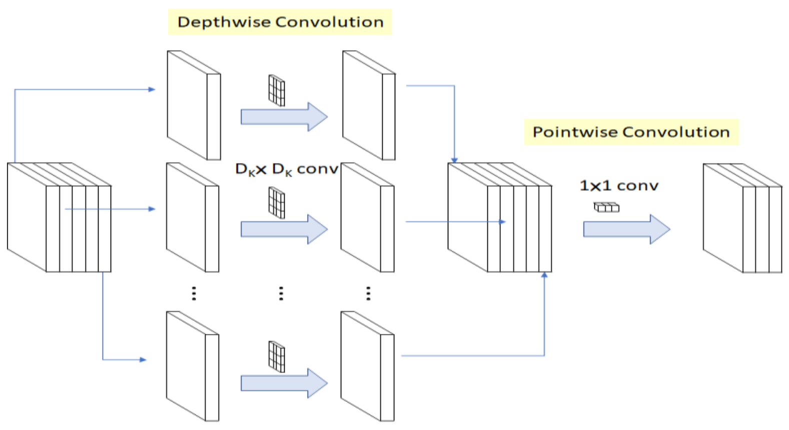


Figure 3. 4 MobileNet architecture (François Chollet, 2017)

The Mobilenetv2 was introduced by Kaiming He *et al* (2015) as an improvement over the original MobileNet model, aiming to achieve better performance in terms of accuracy and efficiency which the bottleneck convolutions had been utilized. The expansion ratio in each bottleneck block shows how the input size compares to the inner size. Inside these blocks, there's an initial input followed by several smaller blocks. They're connected directly because they contain all the important information, while an expansion layer is just an extra detail. Instead of using the usual blocks that connect layers with lots of channels, MobileNetV2 uses inverted residuals, which connect the smaller blocks. This helps save memory and works a bit better. The pre-trained MobileNetV2 model has 16 of these blocks shown in Figure 3.5.

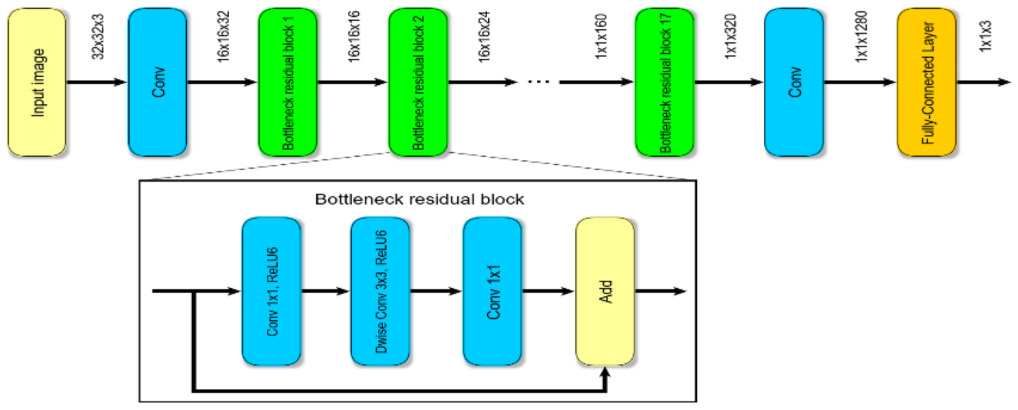


Figure 3. 5 MobileNet V2 architecture (Kaiming He *et al*., 2015)

3.10 ResNet-50

The ResNet-50 architecture is a type of Residual Network (ResNet) that is proposed by Kaiming He *et al*, (2015). Its model architecture is characterized by its deep residual ring framework, which introduces skipping connections to allow the training process to be much faster and improves its performance network by mitigating the vanishing gradient problem. The skip This enables the network to learn residual function instead of direct mapping. The ResNet50 model is designed to process input images sized 224×224 pixels. It consists of a total of 50 layers including 3 convolutional layers and 4 residual blocks, incorporating various components such as convolutional layers, batch normalization layers, activation layers, and identity blocks shown in Figure 3.6.

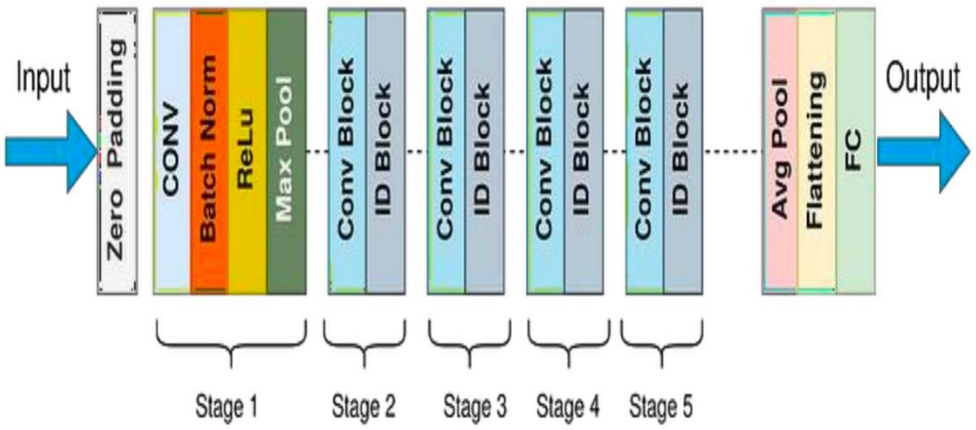


Figure 3. 6 ResNet50 architecture (Kaiming He *et al.*, 2015)

ResNet-50 has two distinct types of skip connections: the identity shortcut and the projection shortcut. The identity shortcut simply preserves the input of the block, ensuring that the information from earlier layers is directly passed through. Conversely, the projection shortcut employs a 1x1 convolutional layer to adjust the dimensions of the input to match those of the output, facilitating smoother information flow through the network. The choice between these two shortcut types is determined by the disparity in dimensions between the input and output feature maps. This versatile approach enhances the network's ability to effectively learn complex representations while maintaining computational efficiency.

3.11 Inception V4

The Inception V4 architecture builds upon its predecessors, including Inception V1, V2, and V3. Like its predecessors, Inception V4 takes input images sized at 299×299 pixels. It follows a similar structure, consisting of a stem module, multiple inception modules, and a final classification layer shown in Figure 3.7. Layers are incrementally added to the model, beginning with the input layer, and appropriate activation functions and padding are applied. The stem module is responsible for extracting features from the input image, while the inception modules contain multiple parallel convolutional and pooling layers.

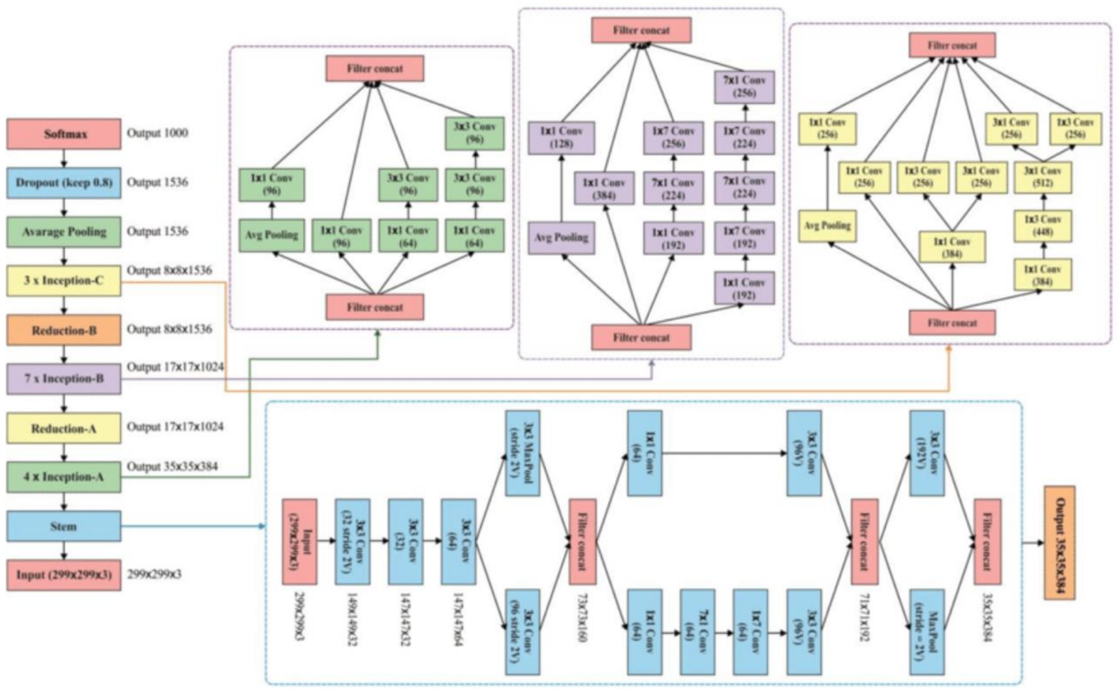


Figure 3. 7 Inception version 4 architecture (Christian Szegedy *et al*., 2015)

This model can undergo training without needing to split duplicates, a departure from earlier versions of Inception that required different copies to fit into memory Szegedy *et al*, (2017). This approach employs memory optimization during backpropagation to reduce the memory requirements. Thanks to Inception layers, internal components can determine the most suitable filter size for gathering necessary data. Additionally, Reduction modules serve as pooling layers situated between the three Inception modules. Inception V4 is set up to handle different types of input changes, like size, rotation, and lighting. It does this by using several different filters and special connections to keep information flowing smoothly. It also creates maps that are easy to understand, so we can better see how the network makes decisions. This might help us understand deep learning models better overall.

3.12 Anomaly Detection

The result of anomaly in the given input will be detected with the use of Simple Moving Average (SMA). This method is able to calculate the average of several data points over a predetermined time range. The Simple Moving Average has a mathematic equation shown in Equation 3.3.

$$SMA(t) = \frac{(d_{(t-n+1)} + d_{(t-n+2)} + \ldots + d_t)}{n} \tag{3.3}$$

The Simple Moving Average (SMA) will be computed at every frame t, where t is greater than or equal to n for n data points. To detect anomalies, the deviation of $d_t$from the SMA at frame t will be assessed. If the absolute variance surpasses a predetermined threshold, an anomaly will be identified.

3.13 Confusion Matrix

The confusion matrix offers a comparative overview of the classification outcomes generated by the model against the actual classification results. It is presented as a matrix table, detailing how well the classification model performs on a set of test data with known true values. An illustration of such a matrix table can be seen in Figure 3.8.

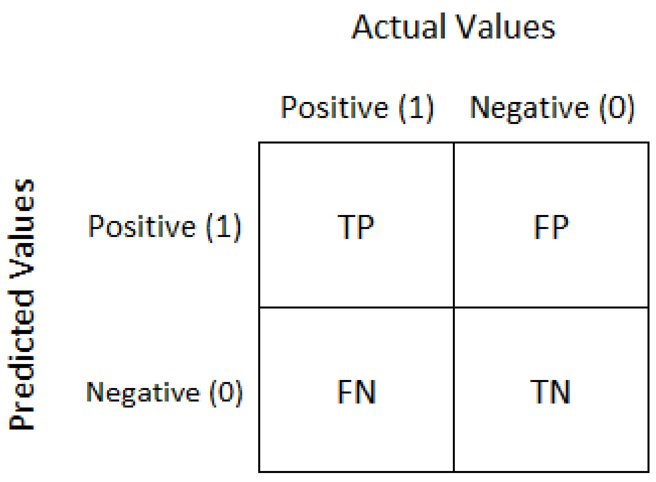


Figure 3. 8 Confusion matrix (Analytics Vidhya, 2021)

The confusion matrix shows four things: True Positive (TP) when positive data is predicted correctly, True Negative (TN) when negative data is predicted correctly, False Positive (FP) when positive data is predicted incorrectly, and False Negative (FN) when negative data is predicted incorrectly. It helps measure how well a classification model works for both binary and multiclass problems. In binary classification, where there are only two output classes, one is usually treated as positive and the other as negative.

3.14 Classification of Anomalies Metrics

Classification metrics are numerical indicators that provide information about how well a model performs a given task. Evaluating the model's performance involves assessing various classification metrics, each offering a distinct perspective on the model's effectiveness. Thus, selecting appropriate metrics is vital for accurately evaluating the model.

3.14.1 Accuracy Score

Accuracy Score is a type of matric that measures how often a model accurately predicts the given data. It is determined by dividing the number of correct predictions by the total number of predictions. The general mathematical formula for calculating accuracy is shown in 3.4 below.

$$\boldsymbol{Accuracy = \frac{Number\ of\ correct\ prediction}{Total\ number\ of\ correct\ prediction}} \qquad (3.4)$$

When it comes to binary classification, the process for determining the accuracy score is shown Formula 3.5 below.

$$\boldsymbol{Accuracy = \frac{TP + TN}{TP + TN + FP + FN}} \qquad (3.5)$$

Where in the equation TP is the true positive, TN is the true negative, FP is the false positive, and FN is the false negative.

3.14.2 Precision

The accuracy score represents the ratio of correctly classified positive samples to all samples classified as positive, regardless of whether they were classified correctly or incorrectly. Precision, on the other hand, assesses how accurately the model identifies a sample as positive. The procedure for calculating the precision score is shown in 3.6 below.

$$Precision = \frac{TP}{TP + FP} \qquad (3.6)$$

When the model makes many incorrect positive classifications or few correct positive classifications, it raises the denominator, leading to a smaller precision. Conversely, precision increases when the model correctly identifies a large number of true positives (maximizing true positives) and minimizes the number of false positives it.

3.14.3 Recall

Recall is a type of metrics that is calculated by dividing the number of positive samples accurately classified as positive by the total number of positive samples. It assesses the model's effectiveness in identifying positive samples, with greater recall indicating a higher number of correctly identified positive samples. The method for determining the recall value is in 3.7 below.

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

The accuracy of classification of positive samples is essential for recall, whereas the classification of negative samples does not impact it. Even if all negative samples were incorrectly labelled as positive, as long as the model correctly identifies all positive samples, the recall will remain at 1.

3.14.4 F1-score

F1- score measures the model’s accuracy in binary classification task that aims to predict one of two possible outcome. It combines accuracy and recall, which measure the model's ability to correctly identify positive examples and avoid misclassifying negative ones. The F1-score is calculated using a specific formula shown in 3.8 below.

$$F1 = \frac{2}{\frac{1}{Recall} + \frac{1}{Precision}} \tag{3.8}$$

A high recall score means the model can identify most of the positive instances in the data, high precision score suggests it makes few wrong positive predictions.

CHAPTER IV: RESEARCH METHODOLOGY

To get an accurate model, we need to understand the problem that will most probably occur during this research. False prediction can cause a huge problem if we are implementing this research into a bigger scale therefore getting a good quality dataset is important because we are training a model to predict the correct classification. To prevent false prediction pre-processing methods are also used to eliminate unnecessary information that is in the dataset by filtration or augmentation before it is analysed. To also have a low computational load, multiple deep learning architecture is used to know which gives an accurate result with a low computational load. This chapter discusses the steps taken in this research. The following will discuss the research stages, materials and equipment, and tests that will be conducted in this research.

4.1 Tools and Materials

This research study requires training and testing large amount of dataset which can take a long time for the model to accurately give its results. The use of Google Colaboratory is use as the programming environment. The Research equipment used for this research are shown in Table 4.1.

Table 4. 1 Research Equipment

No	Name	Specification
1	Laptop	MacBook Pro M2 2023, RAM 16GB
2	Operating System	Sonoma 14.1.2
3	Environment Programming	Google Colaboratory
4	Programming Language	Python

4.2 Research Stages

The first of all stages is to study each architecture and deep learning methods before collecting the data. After collecting the data, Pre-Processing is needed to ensure that the data is appropriately formatted for model training to improve model effectiveness and dependability. Next stage is to split the data into training and testing data to ensure an accurate result. For the training data, YOLOv8 model is used to detect objects with Simple Moving Average (SMA). Then, it tracks them to find anomalies. The role of YOLO V8 is to filter the dataset into feeding it into the CNN process. The system picks the frame with the highest anomaly and the frame that detects accident will then be checked with the MobileNetv2, ResNet50 and Inception v4 CNN Classification model to see if it's an accident or not. Finally, it gives a simple "Accident Detected" or "No Accident Detected" result with an accuracy for all the architecture for comparison. The process can be seen in Figure 4.1. More details on the YOLO V8 and each CNN stages in the model architecture.

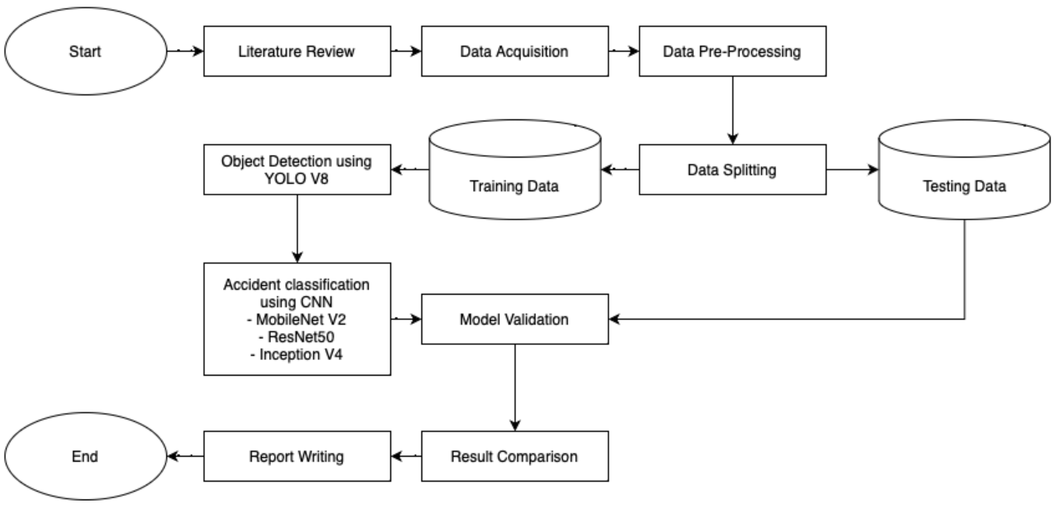


Figure 4. 1 Flowchart of the research

4.3 Data Acquisition

Data acquisition for this research are collected in various sources and gathered in one folder on Kaggle, The data that are collected are public and available for use as research data. There will be an additional dataset added for this research to train the model for an accurate result. Table 4.2 provides detailed information regarding the dataset used in the research process.

Table 4. 2 The dataset

Data	Accident	Non Accident
Training	369 images	422 images
Testing	27 images	54 images
Validation	46 images	52 images

4.4 Data Pre-processing

Preprocessing serves a crucial role in preparing input data for deep learning models, guaranteeing that the data is both clean and well-organized, and appropriately modified to facilitate effective learning and generalization. It addresses various challenges associated with real-world data, ultimately resulting in enhanced model performance and more precise predictions. This research will conduct a variety of data preprocessing techniques such as:

a. Image Resizing

As fixed-size input is often required for the models, resizing is required for preprocessing to guarantee that input photos are of a consistent size. Before supplying photos to the network, they are resized to a predefined dimension, which guarantees consistency in the input data and enables the model to learn from consistent spatial information across images. By streamlining the computational process and enabling effective batch processing, this standardisation makes it possible to recognise objects in a video sequence more accurately and smoothly across many frames or images.

b. Normalization

Since normalisation uniformizes the pixel values of input images, it ensures consistency and enhances the neural network's convergence during training, making it a crucial component for preprocessing. Normalisation normalises the distribution of the input data by scaling pixel values to a common range, usually [0, 1] or [-1, 1]. This makes the input data more suited to optimisation algorithms such as gradient descent. This method lessens the effect of changes in contrast and brightness between frames, allowing the model to acquire critical traits for object detection instead of being swayed by unimportant variables.

c. Data Augmentation

In order to improve the generalization of the model and dealing with imbalanced dataset, data augmentation is crucial for preprocessing since it generates variants of the input photos, enriching the training dataset. Data augmentation effectively augments the training data without requiring extra labelled samples by applying changes to the input frames, such as random cropping, flipping, rotation, and colour jittering. This allows the model to be exposed to a wider range of visual events. In situations where there is a lack of annotated data, this procedure helps to avoid overfitting, improves the model's adaptability to various real-world circumstances, and eventually improves object detection performance.

d. Batching

To improve computational efficiency during inference, batching is required. Through batch processing of numerous frames at once, the model is able to take advantage of the parallelism present in contemporary hardware accelerators such as GPUs. By using batching, the model can more effectively use its resources, resulting in a reduction of the total inference time per frame and the ability to recognise objects in video streams in real-time or almost real-time.

e. Feature Extraction

In order to extract significant characteristics pertinent to object detection, feature extraction entails running the input frames through the convolutional layers of the network. This is a crucial step in the preprocessing process. By examining the spatial data included in the frames, these convolutional layers are able to recognize patterns and characteristics that point to items of interest. Through the extraction of these features, the model is able to perform accurate and efficient object recognition in films by successfully localizing and classifying objects in the input frames based on their distinguishing properties.

4.5 Data Splitting

When a model perform well on the training data but does poorly on testing data, this is called overfitting. To prevent this, we can split the data into three: training data, validation dataset and testing data. The ratio of the training data will be 80% 10% for validation and 10 % for the testing data. By separating the data, the overfitting problem can be detected and can be readjusted so that the model can output a more accurate result.

4.6 Model Architecture Evaluation

In 2021, Faisal Dharma Adhinata conducted research on using MobileNetV2 for face mask recognition, achieving a high accuracy of 95.42%. Through transfer learning, MobileNetV2 can be utilized without retraining to classify the dataset by loading the feature extractor layers and their pre-trained weights from ImageNet. This allows the softmax layer to receive the necessary input to distinguish different data types. This effectiveness is due to the last extractor layer of MobileNetV2 functioning as a global average pooling layer, enabling the preceding convolutional feature maps to serve as class mappers. Given its promising accuracy, this model will be used in the current research for traffic accident detection.

ResNet-50 is a well-known CNN architecture for its effectiveness in image recognition tasks. In 2022, Endang Suherman applied ResNet-50 to End-to-End

Object Detection (DETR) with a 90% accuracy rate. The research demonstrated that the combination of ResNet-50 and DETR achieves higher accuracy compared to DETR models without ResNet-50. Furthermore, ResNet-50 + DETR can detect objects more quickly than traditional CNN models. The study shows that incorporating ResNet-50 into the DETR system significantly enhances object detection performance and speed, which is beneficial for real-time applications. This model architecture will be utilized in the current research to detect traffic accidents due to its promising results.

In 2022, Ali Alqahtani and his colleague proposed a transfer learning-based approach for detecting COVID-19 using the Inception V4 model architecture, achieving an impressive accuracy of 99.63%. This deep neural network has evolved from Inception-v1 and GoogLeNet, featuring a more streamlined and consistent design due to several inception modules. The Inception-v4 was chosen for this research because of its outstanding performance in the ILSVRC 2012 competition, where it had a 3.08% error rate across three residual networks. Additionally, it achieved a top-1 accuracy of 80%, surpassing Inception v3 and GoogLeNet. The development of Inception-v4 included advancements from Inception-v1, Batch Normalization (BN) Inception, and factorization techniques. Due to its history of providing an accurate result, this model architecture will be used for this research to detect traffic accidents.

4.7 Model Architecture

The YOLO V8 model architecture depicted in Figure 4.2 is utilized for object detection from CCTV cameras. Initially. Detected objects undergo tracking, with the tracking results serving as parameters for the anomaly detection algorithm. The system filters frames based on detected anomalies and selects the frame with the highest anomaly value. This frame is then classified using three CNN architecture which are MobileNetV2, ResNet50 and Inception V4 shown in Figure 4.2, Figure 4.3 and Figure 4.4 respectively. as a classification model, facilitating accident incident classification. The classification process yields an image with a description indicating whether an accident has been detected or not. The result of the three architecture will then be compared from the accuracy score, F1-score, recall and precision.

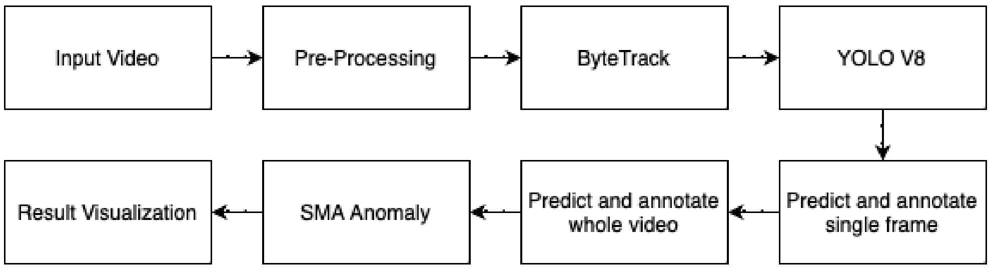


Figure 4. 2 The framework of the YOLO V8 model

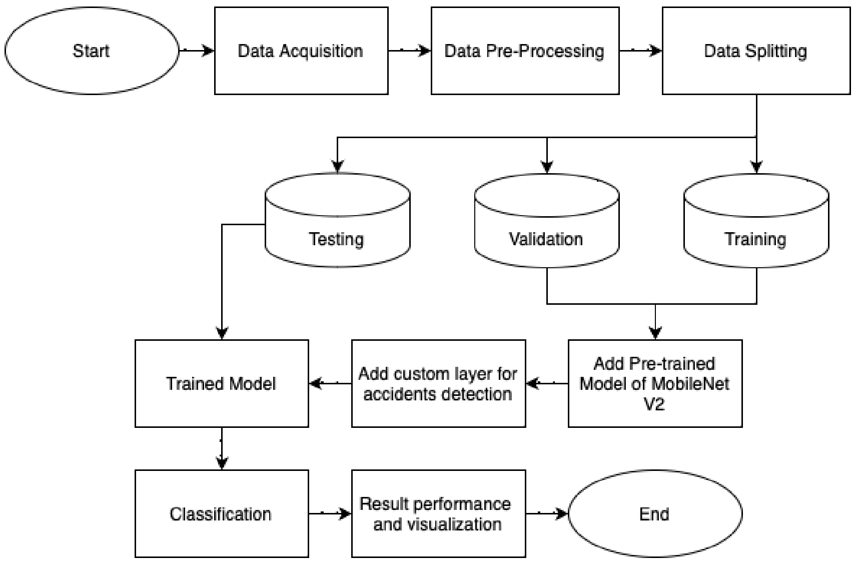

Start
Data Acquisition
Data Pre-Processing
Data Splitting
Testing
Validation
Training
Add Pre-trained Model of MobileNet V2
Add custom layer for accidents detection
Trained Model
Classification
Result performance and visualization
End

Figure 4. 3 The framework of MobileNet V2 Pre-trained model

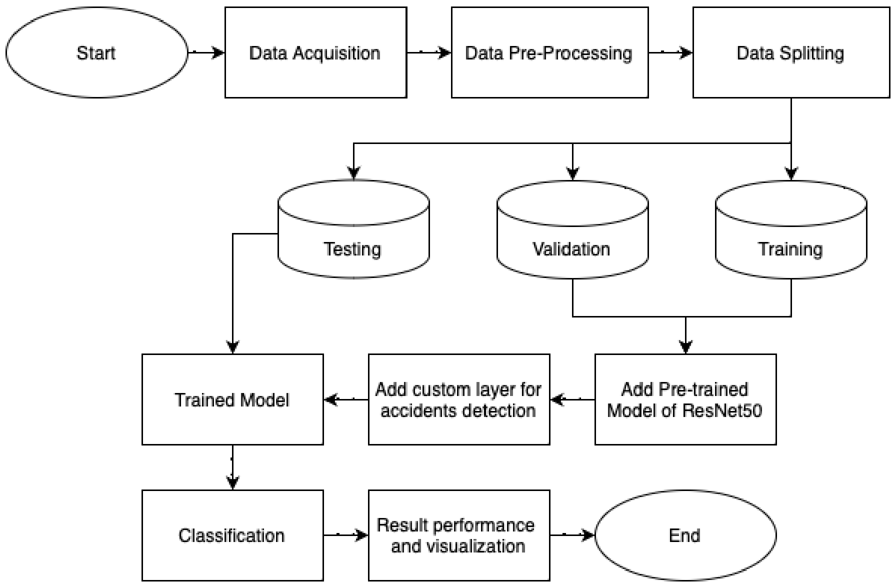

Start
Data Acquisition
Data Pre-Processing
Data Splitting
Testing
Validation
Training
Trained Model
Add custom layer for accidents detection
Add Pre-trained Model of ResNet50
Classification
Result performance and visualization
End

Figure 4. 4 The framework of ResNet 50 Pre-trained model

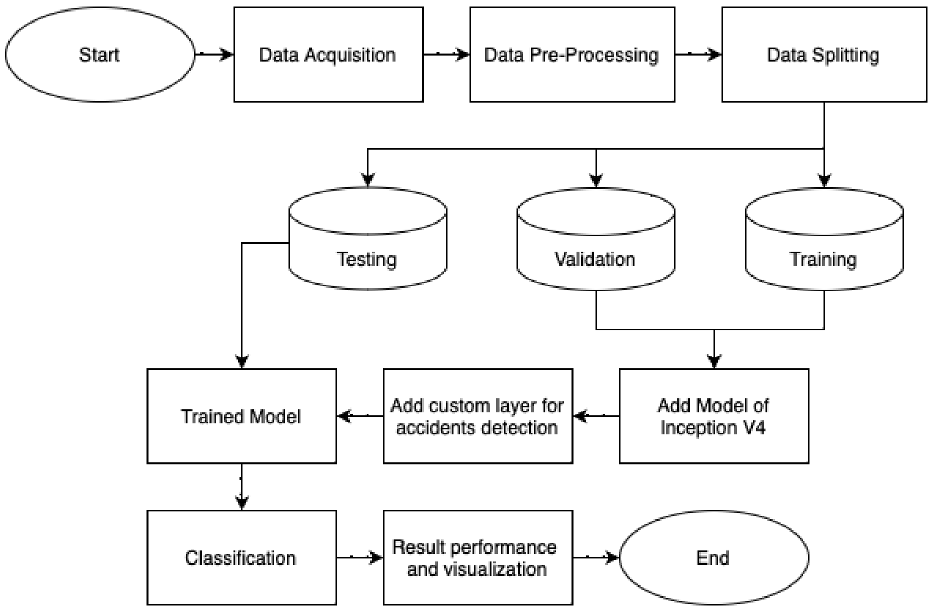


Figure 4. 5 The framework of Inception V4 Pre-trained model

In Figure 4.2, video is collected and inputted to the model for preprocessing. the YOLO V8 model architecture provides high accuracy and speed, ideal for real-time this research, while ByteTrack excels in maintaining robust, consistent tracks even in challenging scenarios. This combination is well-suited for applications. The YOLO V8 is utilized to predict and annotate single frame for analysis which provides precise, detailed insights for specific keyframes, ideal for spot-checking or debugging. Then it predict and annotate the whole video for analysis which offers a continuous stream of predictions across all frames, enabling comprehensive tracking and trend analysis throughout the entire video. With the result of the this, we can create a trend graph which detects any anomaly using Simple Moving Average in the while video.

In the model process starts with the input dataset images which has been imported from Kaggle which is then used for pre-processing. The data is then divided into training, validation, and testing data of 80, 10 and 10 respectively. The training and validation dataset is used to train the model using the pre-trained model and additional custom model for specific task. After the model is trained, the model is then tested using the test dataset and it classifies the images. the result is shown using metrices and visualization. This procedure will be implemented into the three model for this research which can be seen in Figure 4.2, Figure 4.3, and Figure 4.4.

4.7.1 Model Architecture of MobileNetV2

The MobileNetV2 architecture can handle input images of 224×224 pixels, with an input_shape variable as (224, 224, 3), It consist of Depthwise Separable Convolutions. The first layer, known as depthwise convolution, performs lightweight filtering by applying a single convolutional filter to each input channel. The subsequent layer, a 1×1 convolution referred to as pointwise convolution, constructs new features by computing linear combinations of the input channels. The total layer for MobileNet V2 is 53 layers.

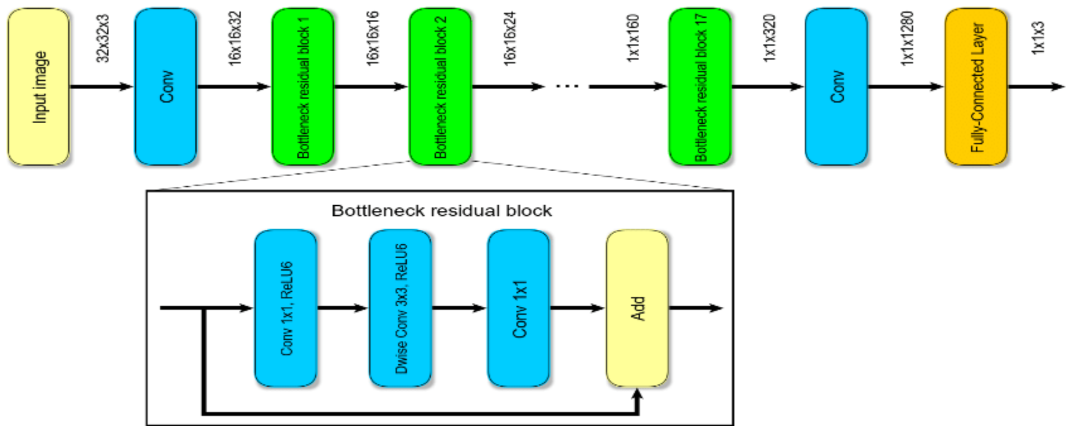


Figure 4. 6 Visualization of MobileNetV2 (Kaiming He *et al*., 2015)

The structure of MobileNetV2 shown on Figure 4.6 begins with a primary fully convolutional layer featuring 32 filters, succeeded by 19 residual bottleneck layers. ReLU6 serves as the non-linear activation function due to its resilience in low-precision computation scenarios. The model consistently employs a kernel size of 3 × 3, adhering to contemporary network standards, while integrating dropout and batch normalization techniques throughout the training process. A custom layer for the model will be added to be able to perform specific task like accidents detection.

4.7.2 Model Architecture of ResNet50

The ResNet50 architecture handles input images of 224×224 pixels, with an input_shape variable as (224, 224, 3), where 3 corresponds to the color channels (RGB). It comprises 50 layers, including convolutional layers, batch normalization layers, activation layers, and identity blocks. A significant innovation in ResNet50 is the use of identity blocks, which help the network learn residual connections and address the vanishing gradient issue. The model is built layer by layer, starting with the input layer, and incorporates appropriate activation functions and padding throughout.

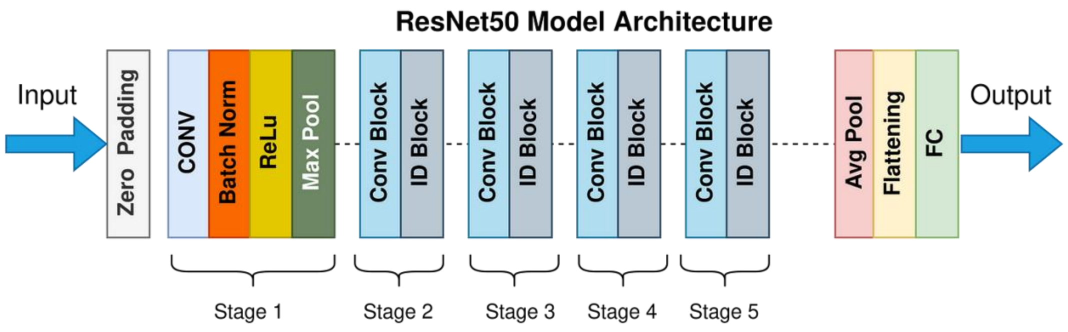


Figure 4. 7 Visualization architecture of ResNet50 (Kaiming He *et al*., 2015)

Figure 4.7 illustrates the structure of ResNet50. Utilizing pre-trained weights sourced from the ImageNet dataset is implemented to initialize the model's weights. This practice not only accelerates the training phase but also enhances the network's overall performance. A custom layer will also be added for the model to be able to perform specific task like accidents detection.

4.7.3 Model Architecture of Inception V4

The Inception V4 architecture handles input images of 299×299 pixels, with the input_shape variable set to (299, 299, 3), where 3 signifies the RGB color channels. This architecture is composed of a stem module, several inception modules, and a final classification layer. Layers are added sequentially, starting with the input layer, utilizing suitable activation functions and padding. The stem module extracts features from the input image, and the inception modules contain multiple parallel convolutional and pooling layers. The structure of the Inception V4 architecture is shown in Figure 4.8.

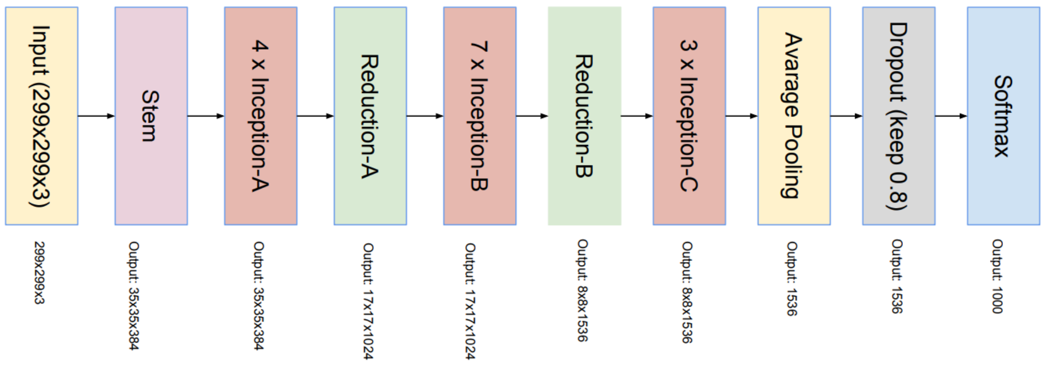


Figure 4. 8 Visualization of Inception V4 (Christian Szegedy *et al*., 2015)

In Inception V4, there are several different types of inception modules, including Inception-A, Inception-B, Inception-C, and the Reduction-A and Reduction-B modules. These modules are incorporated into the model to enhance its depth and width. Inception V4 is still a new model and is still improving to this day, the performance might not be as good since it is still in developing process. A custom layer will also be added for the model to be able to perform specific task like accidents detection.

4.8 Evaluation Procedure

The model will undergo validation to generate various metrics aimed at evaluating its overall performance. Once all metrics from the model are obtained, individual model performances will be scrutinized across each dataset employed for training. The metrics collected will encompass accuracy, precision, recall, confusion matrix, and F-score.

4.9 Model Validation

The model will be tested with new data to see how well it works and to spot any problems like overfitting or underfitting. The quality of the features chosen can affect how well the model performs on this test. If the features don't give good information or miss important details, the model might not do well. Meaning it's important to check how well the chosen features work.

CHAPTER V: RESEARCH IMPLEMENTATION

5.1. Research Environment

For the research environment, it will be conducted by the personal computer of this research. The specification of the computer is listed in Table 5.1 with details. For the environment itself, Google Collaboratory notebook with Python as the programming language for all the model that will be created for a fair comparison and computations for the span of the research experimentation.

Table 5. 1 Research computer specification

Architecture	Specification
Operating System	MacOS Sonoma 14.1.2
RAM	8GB RAM
CPU	10-core CPU
GPU	16-core GPU

5.2. Import YOLOV8 Libraries

In this research, we will import multiple libraries for the YOLOv8 function shown in Figure 5.1. It sets up the environment and imports necessary libraries for using YOLOv8 for object detection and tracking tasks. It begins by checking the GPU status and setting the working directory to home. It installs the ultralytics package, which includes the YOLO models, and imports various modules for handling object detection and tracking. The yolox library and BYTETracker are used for advanced tracking. Utility functions for calculating Intersection Over Union (IoU) and data classes for organizing detection and video processing tasks are imported from onemetric and supervision. Visualization tools from matplotlib, supervision, and OpenCV (cv2) help in displaying detection results and annotating video frames. This comprehensive setup allows for building, training, and visualizing object detection and tracking models using YOLOv8.

```
!nvidia-smi
import os
HOME = os.getcwd()
print(HOME)

import ultralytics
import sys
import yolox

from yolox.tracker.byte_tracker import BYTETracker, STrack
from onemetric.cv.utils.iou import box_iou_batch
from dataclasses import dataclass

import supervision
from supervision.draw.color import ColorPalette
from supervision.geometry.dataclasses import Point
from supervision.video.dataclasses import VideoInfo
from            supervision.video.source            import
get_video_frames_generator
from supervision.video.sink import VideoSink
from           supervision.notebook.utils           import
show_frame_in_notebook
from   supervision.tools.detections   import   Detections,
BoxAnnotator
from  supervision.tools.line_counter  import  LineCounter,
LineCounterAnnotator

from typing import List
import numpy as np
from ultralytics import YOLO
import cv2

import matplotlib.pyplot as plt
```

Figure 5. 1 Importing YOLO V8 Libraries

5.3. Import CNN Model Libraries

This research will involve importing multiple libraries, each providing specific functionalities to aid the experimentation process and implementation. Figure 5.2 shows the source code for the necessary libraries of MobileNet V2, ResNet50, and

Inception V4. All model implementations will be executed in the same Jupyter notebook file, allowing all required libraries to be imported in a single command cell. The datasets used in this research are sourced from Kaggle.

The provided program imports various libraries and modules for image processing and machine learning tasks, including popular deep learning frameworks like TensorFlow and Keras, as well as computer vision libraries like OpenCV and PIL. It also imports modules for data visualization, preprocessing, and machine learning techniques. These imports establish the dependencies needed for developing and training machine learning models, particularly for image classification tasks.

Additionally, the code includes evaluation and metric modules such as accuracy score, logistic regression, and support vector machines. Data visualization modules from matplotlib and seaborn are also imported, enabling the creation of charts and visualizations to assess the model's performance.

```
!kaggle datasets download -d ckay16/accident-detection-from-
cctv-footage
import cv2
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.keras import backend as K
from tensorflow.keras import datasets, models, losses
from tensorflow.keras.layers import Input, Conv2D,
MaxPooling2D, AveragePooling2D
from tensorflow.keras.layers import Dense, Activation,
BatchNormalization
from tensorflow.keras.layers import Flatten, Dropout,
concatenate
from tensorflow.keras.utils import get_file
from tensorflow.keras.layers import Input, Convolution2D,
BatchNormalization, Activation, AveragePooling2D,
MaxPooling2D, concatenate, Flatten, Dense, Dropout
from tensorflow.keras.models import Model
```

Figure 5. 2 Importing CNN Model Environment

5.4. CNN Model Dataset Initialisation

To initialise the dataset which we have imported, we will define the training, testing and validation data to the correct path directory shown in Figure 5.3. Each data set will leverage the keras function to load and pre-process the images for training testing and validation the neural network.

```
# Paths to your datasets
training_ds = '/content/data/train'
testing_ds = '/content/data/test'
validation_ds = '/content/data/val'
# Load the training dataset
training_ds = tf.keras.utils.image_dataset_from_directory(
    training_ds,
    seed=101,
    labels='inferred',
    label_mode='categorical',
    image_size=(img_height, img_width),
    batch_size=batch_size
)

# Load the testing dataset
testing_ds = tf.keras.utils.image_dataset_from_directory(
    testing_ds,
    seed=101,
    labels='inferred',
    label_mode='categorical',
    image_size=(img_height, img_width),
    batch_size=batch_size
)

# Load the validation dataset
validation_ds = tf.keras.utils.image_dataset_from_directory(
    validation_ds,
    seed=101,
    labels='inferred',
    label_mode='categorical',
    image_size=(img_height, img_width),
    batch_size=batch_size
)
```

Figure 5. 3 Dataset Initialisation

In each dataset, we set the seed value to 101 which sets the random seed for shuffling and transformations, ensuring reproducibility. Setting up labels and image size to 250 x 250. Lastly, we also set batch size parameter to 100 which sets 100 images per batch during the process. We can see that there are 791 files with 2 classes which belongs to the training dataset, 100 files with 2 classes which belongs to the validation dataset and 98 files with 2 classes which belongs to the testing dataset.

5.5. YOLOV8 Model Pre-Processing

In the YOLOv8 Pre-processing, implicit pre-processing steps is performed by the YOLOv8 model internally when loading and processing the video frames. The implicit pre-processing involves automatically handling the input video frames to prepare them for object detection. When the video frames are fed into the YOLOv8 model, the model internally performs several pre-processing steps, including resizing the frames to the required input dimensions, normalizing the pixel values (usually scaling them to a range of [0, 1]), and converting the data to the appropriate format. These steps ensure that the video frames match the input specifications of the YOLOv8 model, allowing for accurate and efficient object detection without the need for explicit pre-processing code from the user. This internal handling simplifies the process and reduces the need for manual data preparation.

5.6. CNN Model Pre-Processing

The Pre-processing phase of this research is to resize the image to 250x250 since the dataset image is on 250x250 and we don’t want to crop the image. Another step is to normalize the pixel values, and optimizing the data pipeline for efficient model training. The normalize function scales the pixel values to a range of [0, 1], which helps improve the model's training performance and stability by standardizing the input data. This pre-processing is essential to ensure that the model receives consistently scaled input data and can train more efficiently by reducing the time spent on data loading and processing. The Pre-processing phase can be seen in Figure 5.4.

```
img_shape = (250, 250, 3)

base_model                                                     =
tf.keras.applications.MobileNetV2(input_shape=(250,250,3),
weights='imagenet', include_top=False)

base_model.trainable = False

 def normalize(image, label):
    image = tf.cast(image, tf.float32) / 255.0  # Scale pixel
values to [0, 1]
    return image, label
```

Figure 5. 4 Pre-processing Phase

5.7. YOLOv8 and ByteTrack Model Building

ByteTrack is utilized in this code seen in Figure 5.5 for its superior object tracking abilities, particularly in situations where objects are temporarily occluded or moving quickly—conditions that often challenge traditional trackers. This reliability is essential for tasks like vehicle accidents, where maintaining accurate detection in difficult conditions is crucial. Although YOLOv8 and ByteTrack are both used for object detection and tracking, they operate independently: YOLOv8 detects objects in individual frames, while ByteTrack assigns IDs to these objects across frames. Due to the lack of inherent linkage between their outputs, manual matching of bounding boxes is necessary to correlate YOLOv8 detections with ByteTrack IDs.

```
from typing import List
import numpy as np

def detections2boxes(detections: Detections) -> np.ndarray:
    return np.hstack((
        detections.xyxy,
        detections.confidence[:, np.newaxis]
    ))

def tracks2boxes(tracks: List[STrack]) -> np.ndarray:
    return np.array([
        track.tlbr
        for track
        in tracks
    ], dtype=float)

def match_detections_with_tracks(
    detections: Detections,
    tracks: List[STrack]
) -> Detections:
    if not np.any(detections.xyxy) or len(tracks) == 0:
        return np.empty((0,))

    tracks_boxes = tracks2boxes(tracks=tracks)
    iou = box_iou_batch(tracks_boxes, detections.xyxy)
    track2detection = np.argmax(iou, axis=1)

    tracker_ids = [None] * len(detections)

    for        tracker_index,        detection_index        in
enumerate(track2detection):
        if iou[tracker_index, detection_index] != 0:
            tracker_ids[detection_index]                       =
tracks[tracker_index].track_id

    return tracker_ids
```

Figure 5. 5 Importing ByteTrack for YOLO V8

5.8. CNN Model Building

Figure 5.6 illustrates the construction of a neural network model based on the ResNet50 architecture. The process begins by creating an instance of the ResNet50 model using Keras' ResNet50 function. The input parameter specifies the input image dimensions as (250, 250, 3), representing a 250×250 pixel image with three RGB color channels. Setting the weights parameter to ImageNet initializes the model with pre-trained weights from the ImageNet dataset. The top parameter is set to False to exclude the final fully connected layers of the ResNet50 model.

```
img_shape = (250, 250, 3)
base_model_ResNet50                                              =
tf.keras.applications.resnet50.ResNet50(input_shape=(250,250,
3), weights='imagenet', include_top=False)
base_model_ResNet50.trainable = False

#Add final few layers to combine with ResNet50
ResNet50_model = tf.keras.Sequential([
    base_model_ResNet50,
    layers.Conv2D(32, 3, activation='relu'),
    layers.Conv2D(64, 3, activation='relu'),
    layers.Conv2D(128, 3, activation='relu'),
    layers.Flatten(),
    layers.Dense(len(class_names), activation= 'softmax')
])
ResNet50_model.summary()
```

Figure 5. 6 ResNet50 Model Architecture Build

In this model, the ResNet50 model is added as the initial layer and used as a pre-trained feature extractor. Subsequently, three 2D convolutional layers are added with 32, 64, and 128 filters respectively, each using a 3x3 kernel and ReLU activation which helps enhance feature extraction, increase model capacity, progressively refine features and adapt the model to the dataset. a flatten layer to convert the 2D matrix data to a 1D vector and a dense (fully connected) layer with a softmax activation to produce the final class probabilities. The activation function is set to softmax to generate probability forecasts for each class. Additionally, a loop iterates through each layer in the model, setting the trainable property to False to freeze the weights of the ResNet50 model, ensuring they remain unchanged during training.

```
img_shape = (250, 250, 3)
base_model                                                   =
tf.keras.applications.MobileNetV2(input_shape=(250,250,3),
weights='imagenet', include_top=False)
base_model.trainable = False

Mobile_model = tf.keras.Sequential([
    base_model,
    layers.Conv2D(32, 3, activation='relu'),
    layers.Conv2D(64, 3, activation='relu'),
    layers.Conv2D(128, 3, activation='relu'),
    layers.GlobalAveragePooling2D(),
    layers.Flatten(),
    layers.Dense(len(class_names), activation= 'softmax')
])
Mobile_model.summary()
```

Figure 5. 7 MobileNet V2 Architecture Build

Figure 5.7 illustrates the construction of a neural network model based on the MobileNetV2 architecture. The input parameter specifies the input image dimensions as (250, 250, 3), representing a 250×250 pixel image with three RGB color channels. Setting the weights parameter to ImageNet initializes the model with pre-trained weights from the ImageNet dataset. The top parameter is set to False to exclude the final fully connected layers of the MobileNetV2 model. The overall structure and other layers in this model remain the same, including batch normalisation, fully-connected layers, and activation functions, the MobileNetV2 model incorporates the utilisation of residual connections.

```
conv2d_bn(x, nb_filter, num_row, num_col,
              padding='same',         strides=(1,         1),
use_bias=False):

    if K.image_data_format() == 'channels_first':
        channel_axis = 1
    else:
        channel_axis = -1
    x = Convolution2D(nb_filter, (num_row, num_col),
                      strides=strides,
                      padding=padding,
                      use_bias=use_bias,

kernel_regularizer=regularizers.l2(0.00004),

kernel_initializer=initializers.VarianceScaling(scale=2.0,
mode='fan_in', distribution='normal', seed=None))(x)
    x          =          BatchNormalization(axis=channel_axis,
momentum=0.9997, scale=False)(x)
    x = Activation('relu')(x)
    return x


def block_inception_a(input):
    if K.image_data_format() == 'channels_first':
        channel_axis = 1
    else:
        channel_axis = -1

    branch_0 = conv2d_bn(input, 96, 1, 1)

    branch_1 = conv2d_bn(input, 64, 1, 1)
    branch_1 = conv2d_bn(branch_1, 96, 3, 3)

    branch_2 = conv2d_bn(input, 64, 1, 1)
    branch_2 = conv2d_bn(branch_2, 96, 3, 3)
    branch_2 = conv2d_bn(branch_2, 96, 3, 3)

    branch_3    =    AveragePooling2D((3,3),    strides=(1,1),
padding='same')(input)
    branch_3 = conv2d_bn(branch_3, 96, 1, 1)

    x = concatenate([branch_0, branch_1, branch_2, branch_3],
axis=channel_axis)
    return x
```

Figure 5. 8 Inception V4 Architecture Build

Figure 5.8 illustrates the construction of the Inception-v4 architecture, a deep convolutional neural network used for image classification. The code defines several functions that form the foundation of the Inception-v4 model. After batch normalization and activation, a convolution process begins, utilizing parameters such as strides, padding, and regularization. Activations are normalized with batch normalization, followed by the application of the ReLU activation function for non-linearity. The base function constructs the Inception-v4 architecture by layering multiple inception and reduction modules, starting with convolutional layers and alternating between inception and reduction modules. This structure helps the network capture hierarchical properties of varying complexity. It depicts the complete Inception-v4 model creation by connecting the input tensor to the inception base and optionally incorporating pre-trained weights. The function loads weights trained on the ImageNet dataset. After adding the Inception V4, we will also add our model so that it will perform the desired task detection.

5.9. YOLOv8 Model Evaluation

Yolo V8 is evaluated through each frame and predicts the result shown in Figure 5.9. The predictions are then compared with ground truth annotations by calculating Intersection over Union (IoU) to determine matches. The result is then evaluated further using the Simple Moving Average (SMA) method to detect and predict if there is any displacement which is cause by the crash movement of the vehicle.

```
results = model(frame)
detections = Detections(
    xyxy=results[0].boxes.xyxy.cpu().numpy(),
    confidence=results[0].boxes.conf.cpu().numpy(),
    class_id=results[0].boxes.cls.cpu().numpy().astype(int)
)
# format custom labels
labels = [
    f"{CLASS_NAMES_DICT[class_id]} {confidence:0.2f}"
    for _, confidence, class_id, tracker_id
    in detections
```

Figure 5. 9 Model evaluation for YOLO V8

This code processes a single frame from a video using a YOLOv8 model for object detection and annotates the detections. It begins by creating a frame generator from the source video. An instance of box annotator is created to handle the drawing of detection boxes with specified color, thickness, and text properties. The first video frame is acquired from the generator. The YOLOv8 model predicts objects in this frame, and the results are converted into a detection object, which includes bounding box coordinates, confidence scores, and class IDs. Custom labels are formatted for each detection, combining class names and confidence scores. The box annotator then annotates the frame with these detection boxes and labels. Finally, the annotated frame is displayed inline within the notebook using Matplotlib.

5.10. CNN Model Evaluation

Figure 5.10 illustrates a programme that trains, evaluates, and generates a performance report with several metrics to measure the model's classification performance on the test dataset.

```
from sklearn.metrics import classification_report
import numpy as np

y_true = np.concatenate([y for x, y in testing_ds])

y_pred = ResNet50_model.predict(testing_ds)
y_pred = np.argmax(y_pred, axis=1)

report      =      classification_report(y_true,      y_pred,
target_names=class_names)

print(report)
```

Figure 5. 10 Evaluation for CNN model

It concatenates true labels from the testing dataset, then uses the model to predict labels, converting predicted probabilities to class indices. Similarly, it converts true labels from one-hot encoded format to class indices. A classification report is then generated, which maps class indices to their corresponding names with target_names=class_names. Finally, the classification report, detailing precision, recall, F1-score, and support for each class, is printed to the console, providing a detailed evaluation of the model's performance.

```
sns.set(style='whitegrid', context='notebook')

plt.figure(figsize=(14, 6))

plt.subplot(1, 2, 1)
plt.plot(Model.history['accuracy'])
plt.plot(Model.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')

plt.subplot(1, 2, 2)
plt.plot(Model.history['loss'])
plt.plot(Model.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')

plt.tight_layout()

plt.show()
```

Figure 5. 11 Model evaluation plot for CNN model

Another evaluation is to visualize the accuracy of the training and validation data through all the epoch seen in Figure 5.11. This sets the plotting style using Seaborn and creates a 14x6 inch figure with two subplots. The first subplot plots the training and validation accuracy over epochs, adding a title, axis labels, and a legend. The second subplot plots the training and validation loss over epochs, also adding a title, axis labels, and a legend. This helps to compare the model's performance on training and validation data over time.

CHAPTER VI: RESULT AND DISCUSSION

6.1. Result of Object Detection Using YOLO V8

The YOLO V8 is a pre-trained model that is trained by the COCO dataset which successfully detects object shown in Figure 6.2 after detection from the CCTV Video from an image shown in Figure 6.1 before detection. With its confidence scores that is relatively low due to the amount of vehicles that are closely packed or overlapping, the model struggles to distinguish between individual objects, leading to a low confidence scores.

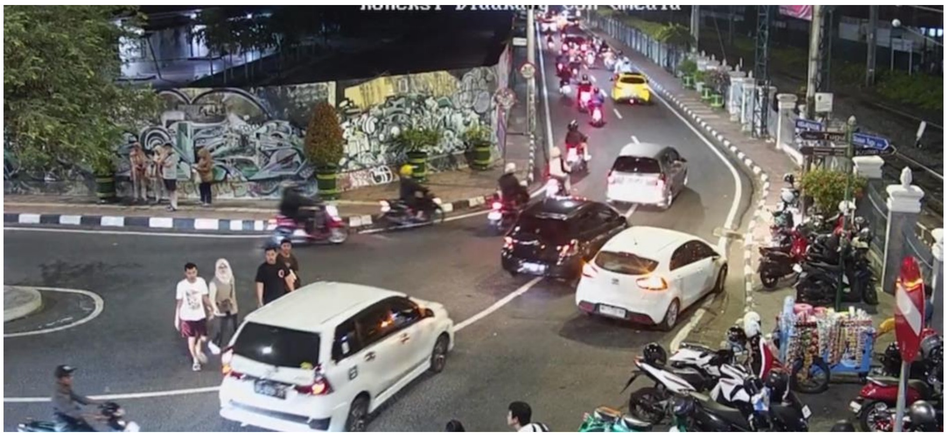


Figure 6. 1 CCTV Image before detection

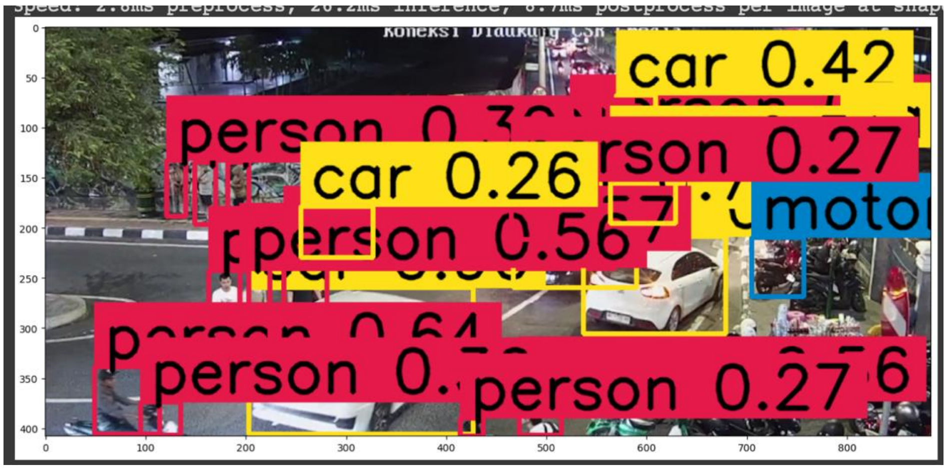
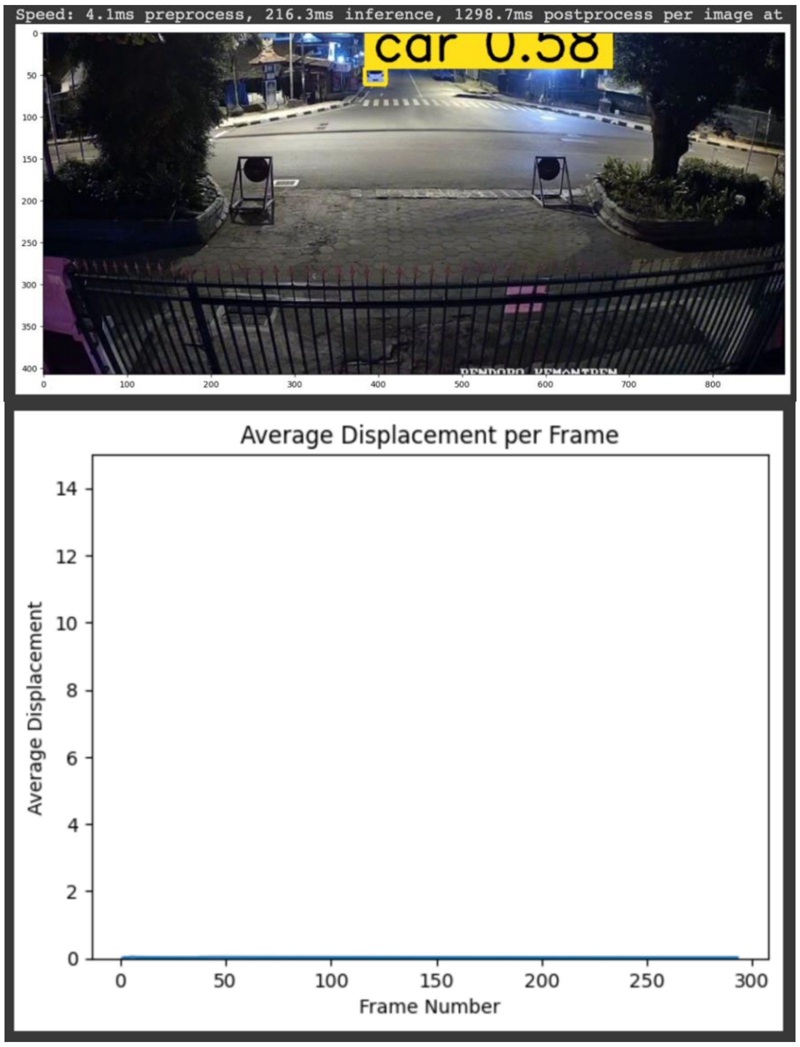


Figure 6. 2 CCTV Image after detection

6.2. Result of Anomaly

The YOLO V8 successfully detects object from the input videos without any crashes that has been acquired from the CCTV camera of Jogja in Pendopo Kemantren. To detect whether there is an anomaly, a simple moving average graph (SMA) is used to see if there is any spike or drastic increase which shows that there is a sudden change of movement in the video. Figure 6.3 shows that there are no sudden change of movements in the video to detect if there is any anomaly which therefore the graph shows a constant low value.



```
Speed: 4.1ms preprocess, 216.3ms inference, 1298.7ms postprocess per image at
```

Figure 6. 3 Anomaly detection without any accident

Same follows for the accident data. The YOLO V8 successfully detects object from the input video with accident that has been acquired from the CCTV camera from an Indian news in Urdu. To detect whether there is an anomaly, a simple moving average graph (SMA) is used to see if there is any spike or drastic increase which shows that there is a sudden change of movement in the video. Figure 6.4 shows that there is a sudden change of movements in the video frame. Around frame number 3000, there is a sharp increase in average displacement. This sudden peak suggests a significant change or anomaly, which could be an unusual event such as an accident.

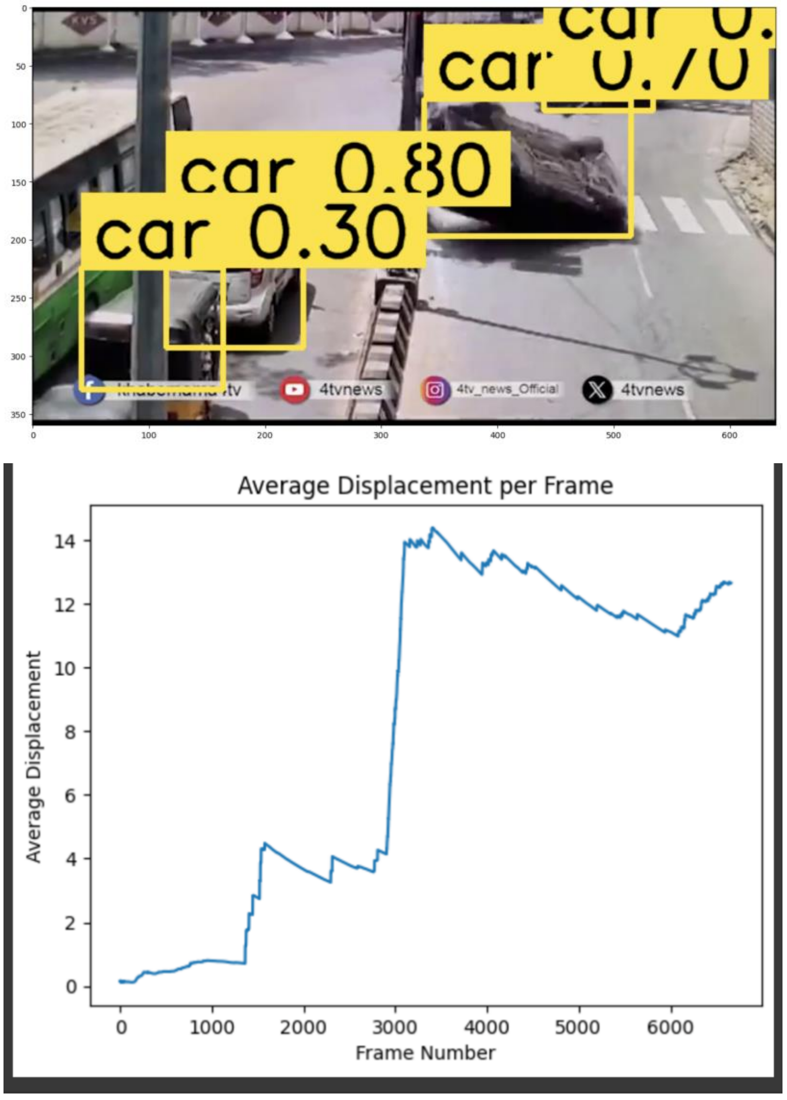


Figure 6. 4 Anomaly detection with accident

6.3. Hyper-parameter Tuning For CNN Model

The initial step in training a CNN model involves carefully adjusting hyper-parameters to achieve the best possible performance. This process includes tweaking various hyper-parameters, each playing a significant role in the model's learning process. Among these are the learning rate, which determines the magnitude of the model's learning steps; the optimizer, which updates the model parameters based on the learning rate; and the number of epochs, which specifies how many times the entire training dataset is used to adjust the model's weights. This strategic tuning of hyper-parameters is crucial for maximizing the model's accuracy and enhancing its ability to perform its intended function effectively. Below are the Hyper-parameter that are set for this training and testing.

6.3.1. Learning Rate

To estimate the optimal learning rate, tests are required to find the optimal value for the model. The learning rate is tested in 100 iterations from a high value from 0.1 to 0.01.

Table 6. 1 Learning Rate value comparison

Learning Rate	Accuracy
0.001	0.932
0.01	0.920
0.1	0.919

By examining the experimental results in Table 6.1 reveals that a learning rate of 0.001 tend to have the highest accuracy at 0.932. When the learning rate is increased to 0.01, the accuracy drops to 0.920. Further increasing the learning rate to 0.1 results in an accuracy of 0.929. The best performance is achieved with a smaller learning rate of 0.001, suggesting that smaller parameter update steps enhance performance. However, as the learning rate rises, the model's ability to converge to the optimal solution worsens, leading to reduced accuracy.

6.3.2. Optimiser

To know what optimiser suits best for the model, these optimisers are compared and tested for the best accuracy to use for the model.

Table 6. 2 Optimiser Comparison

Optimiser	Accuracy
Adam	0.930
SGD	0.909
RMSprop	0.898

From the Table 6.2, it can be seen that the Adam optimizer achieves the highest accuracy at 0.930. In comparison, the SGD optimizer attains an accuracy of 0.909, and the RMSprop optimizer achieves 0.898. These findings indicate that the choice of optimizer influences the model's accuracy on the dataset. The Adam optimizer performs the best, surpassing the other optimizers in accuracy. Both RMSprop and SGD achieve similar accuracies, though still lower than that of the Adam optimizer.

6.3.3. Epoch

Table 6.3 presents the value for accuracy that is trained by epochs. The epochs starts at 5, 10 and an increment of 10 up to a 100. The best accuracy value for the epoch will be chosen and use for the model training.

Table 6. 3 Epoch Comparison

Epoch	Accuracy	Epoch	Accuracy	Epoch	Accuracy
5	0.819	40	0.930	80	0.920
10	0.870	50	0.939	90	0.930
20	0.910	60	0.910	100	0.920
30	0.921	70	0.931		

Upon examining the table, it is evident that the model achieved an accuracy of 0.819 after 5 epochs. With an increase to 10 epochs, there was a significant improvement in accuracy, reaching 0.870. The accuracy continued to improve, peaking at 0.939 after 50 epochs. However, beyond 50 epochs, the model's accuracy declined slightly, settling between 0.910 and 0.920 up to 100 epochs. Thus, 50 epochs provided the optimal accuracy for the model.

The decline in accuracy after increasing the number of epochs could be due to the dataset images lacking sufficient new information for the model to learn. As a result, the model starts to focus on specific patterns and noise within the training set, which may not generalize well to new data. This can cause the model to overshoot the optimal solution or become trapped in oscillations, ultimately reducing its accuracy.

6.3.4. Optimal Parameter

The optimal parameter is shown in Table 6.1 below. Adam (Adaptive Moment Estimation) Optimiser is regarded as an excellent optimizer compare to the others. It individually adjusts the learning rate for each parameter based on the first and second moments of the gradients, making it versatile and effective for various problems. Adam generally performs well with minimal hyperparameter tuning, often achieving faster convergence and superior results compared to traditional methods. The optimal learning rate for this optimiser is 0.001 which works well on many problem. 50 epochs will be executed for an optimal result.

Table 6. 4 Optimal Parameter for CNN model

Hyper-parameter Tuning	Value
Optimiser	Adam
Learning rate	0.001
Epoch	50

6.4. Model Training

In this model training, each CNN models that are developed for this research will be trained using all of the available dataset. MobileNet V2, ResNet50, and Inception V4 will be validated separately to retrieve information on the overall strength of the model.

The MobileNet V2 model was trained with the hyperparameters determined from the previous experiments. During the training process, a notable and consistent increase in the accuracy score was observed with each epoch. This consistent improvement indicates that the model is effectively learning and capturing the underlying patterns in the data as seen in Figure 6.5.

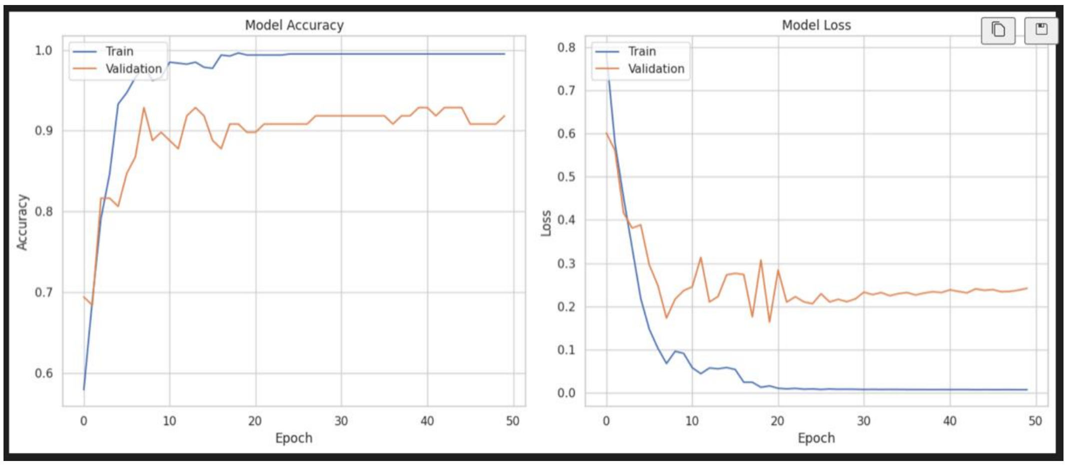


Figure 6. 5 Training result of MobileNet V2

The observed increase in accuracy with each epoch signifies that the model's learning process is efficient and that it is successfully converging towards an optimal solution. The graphs show that the model's training accuracy rapidly increases and stabilizes close to 1.0, with training loss decreasing steadily to a very low level, indicating effective learning from the training data. However, there is a noticeable gap between training and validation accuracy after around 10 epochs, with training accuracy nearing 1.0 while validation accuracy stabilizes around 0.9 which might occur because of overfitting.

The ResNet50 model demonstrated a good training performance, with a clear upward trend in accuracy across epochs, as illustrated in Figure 6.6. This consistent growth in accuracy underscores the ResNet50 architecture's in learning and capturing detailed patterns within the dataset.

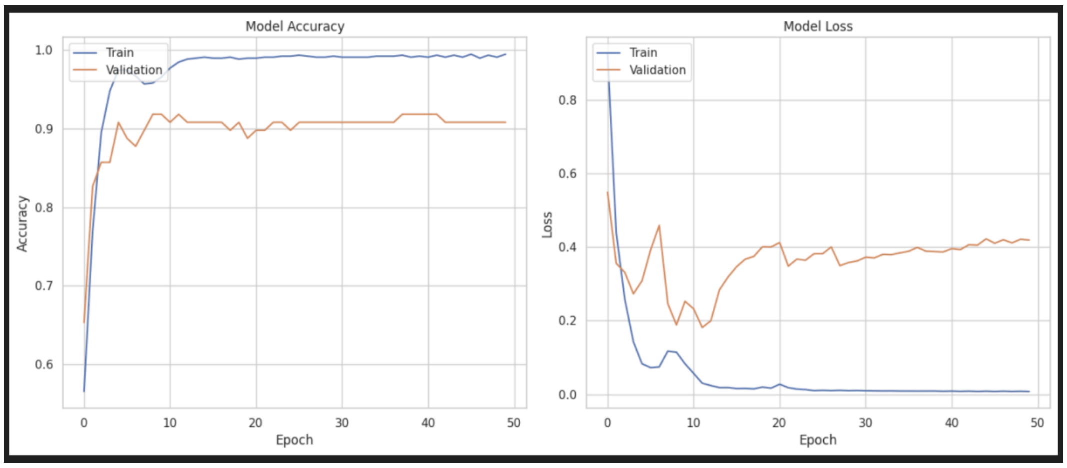


Figure 6. 6 Training result of ResNet50

The graph shows that the training accuracy rapidly climbs and stabilizes near 1.0, indicating that the model is effectively learning from the training data. The training loss decreases sharply and remains low, reflecting minimal error. On the other hand, the validation accuracy increases initially but then levels off around 0.9, indicating a steady but slightly lower performance on unseen data compared to the training set. The validation loss decreases at first but then fluctuates and eventually settles at a higher level than the training loss, which again might be because of overfitting.

The Inception V4 model also demonstrated a good training performance, with a clear upward trend in accuracy across epochs, as illustrated in Figure 6.7. This consistent growth in accuracy underscores the Inception V4 architecture's in learning and capturing detailed patterns within the dataset.

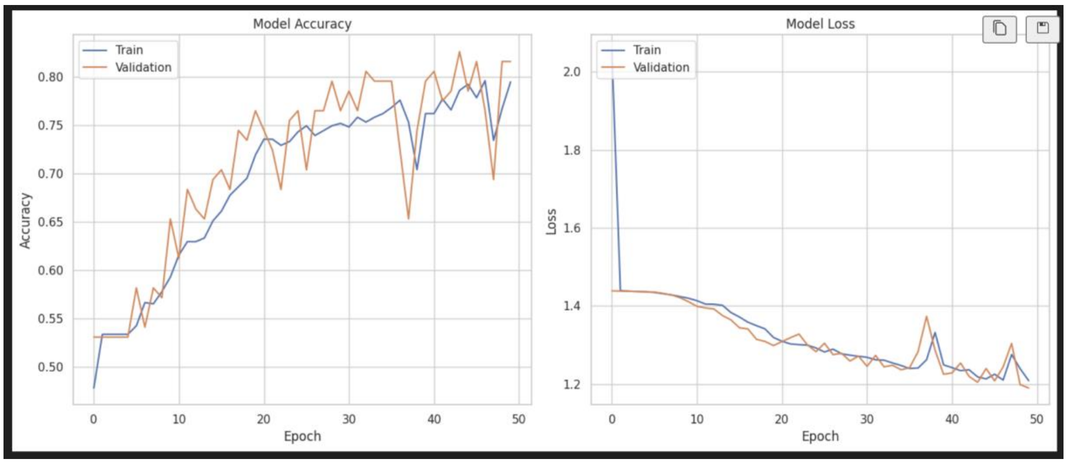


Figure 6. 7 Training result of Inception V4

The graph shows that both training and validation accuracies steadily increase, demonstrating effective learning, though there are noticeable fluctuations in the validation accuracy, indicating variable performance on unseen data. Eventually, the validation accuracy converges with the training accuracy around 0.8. Both training and validation losses decrease sharply at first and then continue to decline gradually, with the validation loss being slightly higher and more variable than the training loss. Looking at the trend of the model loss, we can see that it decreases more if we are training with more epochs, meaning that there are room for improvements such as adding more epochs to the training process which can result in a better performance model. This overall alignment between training and validation performance, despite some inconsistencies, suggests that the model learns effectively.

In conclusion, the three models, MobileNet V2, ResNet50, and Inception V4, have demonstrated different training performances and characteristics with and imbalanced dataset. MobileNet V2 shows a stable and consistent increases in accuracy throughout the training process, indicating its effectiveness in learning and capturing patterns within the dataset. However, the graph shows a potential overfitting which makes the hyperparameter and the model room for more improvements.

ResNet50 also displayed a good training performance with a steady increase in accuracy with each epoch and reach a stable accuracy. This indicates that the deep architecture of ResNet50 effectively learned complex features and improved performance over time. However, the graph also shows a potential overfitting which makes the hyperparameter and the model room for more improvements.

Inception V4 also showed a steady increase in accuracy per epoch. However, the graph indicates that there are spike or a sudden change in value due to factors such as noise in the data, small validation dataset, batch size and many more. Despite the problem faced, the model shows an excellent training performance with the chosen hyperparameters and architecture of Inception V4 that is well-suited for this task.

With all the model result. The need for techniques such as regularization, dropout, or more down sampling methods to improve the model, along with fine-tuning more hyperparameters for better overall performance.

6.5. CNN Model Testing

The Model testing procedure, the results of each test will be observed and evaluated. To facilitate a comprehensive understanding, obtained results will be presented in a visually informative manner. The visual representation may include Image Results, charts, plots, or other visual aids that effectively convey the collected data and highlight important patterns or trends.

Based on the performance report shown in Table 6.5, the provided results for the MobileNet V2 model demonstrate a great overall performance in terms of precision, recall, F1-score, and accuracy. The accuracy scores for both accident and non-accident datasets are both around 0.94, which indicates that the model correctly classifies approximately 94% of the instances in the test data. The recall for both classes scores show a high value, with scores of 0.94. This indicates that the model effectively captures a large portion of the positive instances for each class, minimizing false negatives. The model achieved an F1-score of 0.94 for the accident class and non-accident class, indicating a balanced performance between precision and recall. The precision of accident scores a high value of 0.94 for accident and 0.94 for non-accident which indicates that the model performs well in avoiding misclassification of negative instances as positive.

Table 6. 5 MobileNetV2 Performance

Class	Precision	Recall	F1-Score	Accuracy
Accident	0.92	0.94	0.93	0.94
Non Accident	0.94	0.93	0.93	

Figure 6.8 shows the result of testing an image of the model MobileNet V2 given the correct label and the prediction of the model for comparison. By analysing the image, it successfully detects if there is an accident or no accident in the road.

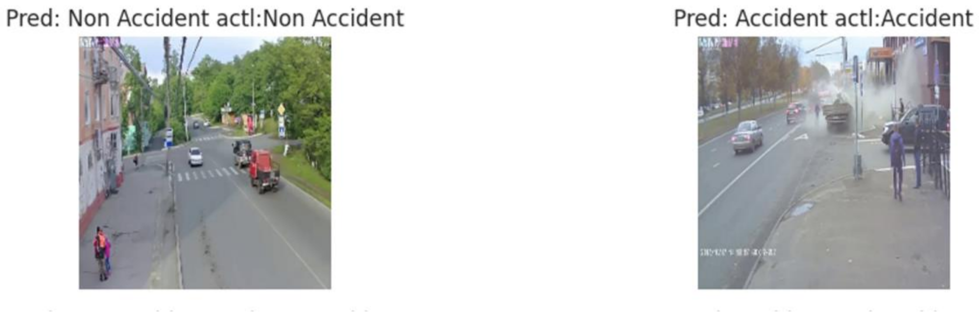

Pred: Non Accident actl:Non Accident
Pred: Accident actl:Accident

Figure 6. 8 MobileNet V2 test image

The confusion matrix for MobileNet V2 in Figure 6.9 shows the classification model's performance by comparing actual labels with predicted ones. The model accurately identified 44 instances as "Accident" and 50 instances as "Non-Accident," demonstrating high accuracy. However, there are some errors as 3 instances were falsely predicted as "Accident" on false positives, and another 3 instances were mistakenly predicted as "Non-Accident" on false negatives. the model shows a strong performance with a few classification mistakes.

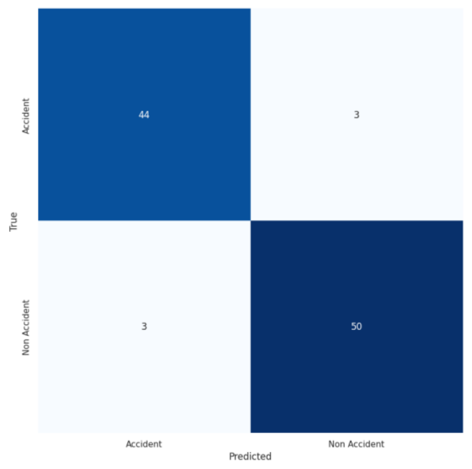


Figure 6. 9 Confusion matrix result of MobileNet V2

For the ResNet50 result in Table 6.6, the model demonstrates an excellent overall performance in terms of precision, recall, F1-score, and accuracy. The accuracy scores for both accident and non-accident datasets are both around 0.96, which indicates that the model correctly classifies approximately 96% of the instances in the test data. The recall for the class accident scores a value of 0.94 and 0.98 for non-accident. This indicates that the model effectively captures a large portion of the positive instances for each class, minimizing false negatives. The model achieved a high value for F1-score of 0.96 for both accident class and non-accident class, indicating a balanced performance between precision and recall. The precision of accident scores a high value of 0.98 for accident and 0.95 for non-accident which indicates that the model also performs well in avoiding misclassification of negative instances as positive.

Table 6. 6 ResNet50 Performance

Class	Precision	Recall	F1-Score	Accuracy
Accident	0.98	0.94	0.96	0.96
Non Accident	0.95	0.98	0.96	

Figure 6.10 shows the result of testing an image of the model ResNet50 given the correct label and the prediction of the model for comparison. By analysing the image, it successfully detects if there is an accident or no accident in the road.

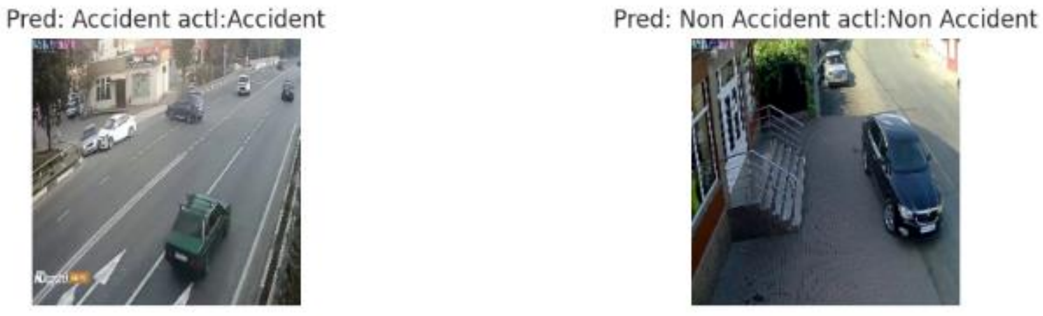

Pred: Accident actl:Accident
Pred: Non Accident actl:Non Accident

Figure 6. 10 ResNet50 test image

The confusion matrix for ResNet50 in Figure 6.11 shows the classification model's performance by comparing actual labels with predicted ones. The model accurately identified 44 instances as "Accident" and 52 instances as "Non-Accident," demonstrating high accuracy. However, there are some errors as 3 instances were falsely predicted as "Accident" on false positives, and 1 instances were mistakenly predicted as "Non-Accident" on false negatives. the model shows a strong performance with a few classification mistakes.

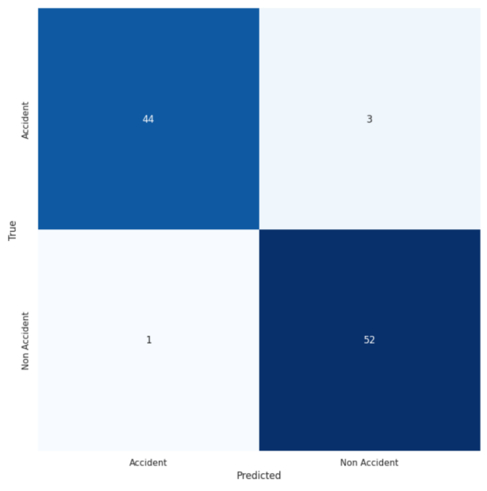


Figure 6. 11 Confusion matrix result of MobileNet V2

Lastly, for Inception V4 result in Table 6.7, the model demonstrates a good overall performance in terms of precision, recall, F1-score, and accuracy. The accuracy scores for both accident and non-accident datasets are both around 0.80, which indicates that the model correctly classifies approximately 80% of the instances in the test data. The recall for the class accident scores a value of 0.64 which is relatively low compared to non-accident which score a value of 0.96. This indicates that the model misses many actual accident cases, leading to a high number of false negatives. The model achieved a value for F1-score of 0.74 for accident class and 0.84 for non-accident class, indicating that the model misses many actual accident cases. In contrast, the non-accident class has a balanced precision of 0.74 and high recall of 0.96, resulting in fewer false negatives and positives.

The precision of accident scores a high value of 0.94 for accident and a low value score of 0.74 for non-accident which indicates that the model correctly identifies most predicted accidents but has a higher rate of false positives for non-accidents.

Table 6. 7 Inception V4 Performance

Class	Precision	Recall	F1-Score	Accuracy
Accident	0.94	0.62	0.74	0.80
Non Accident	0.74	0.96	0.84	

Figure 6.12 shows the result of testing an image of the model Inception V4 given the correct label and the prediction of the model for comparison. By analysing the image, it successfully detects if there is an accident or no accident in the road. However, due to the accuracy of the model, there are many miss prediction on the testing data.

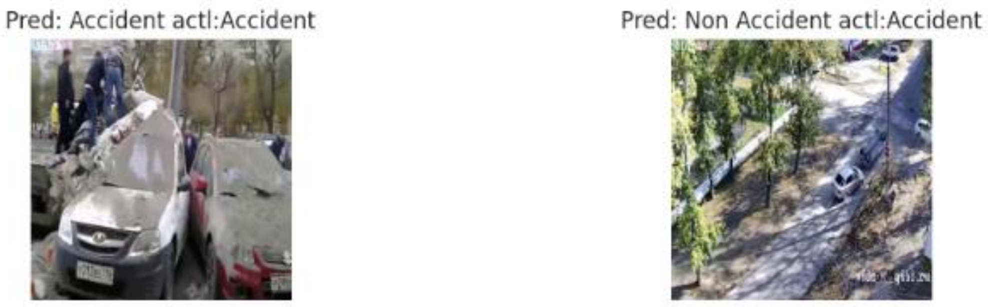

Pred: Accident actl:Accident
Pred: Non Accident actl:Accident

Figure 6. 12 Inception V4 test image

The confusion matrix for Inception V4 in Figure 6.13 shows the classification model's performance by comparing actual labels with predicted ones. The model accurately identified 29 instances as "Accident" and 51 instances as "Non-Accident," demonstrating low accuracy for identifying accidents. There are many errors as 18 instances were falsely predicted as "Accident" on false positives, and 2 instances were mistakenly predicted as "Non-Accident" on false negatives. the model shows a poor performance compare to the other model with many classification mistakes.

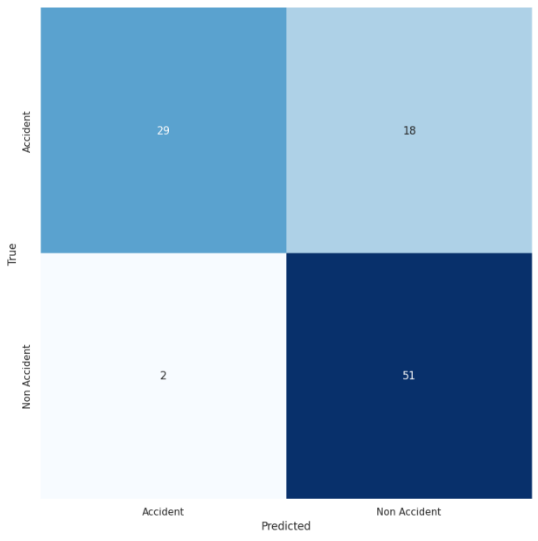


Figure 6. 13 Confusion matrix result of Inception V4

With all the model tested, we can see that each model have a different value of precision, recall, F1-score and accuracy of both classes. The Table 6.8 shows the overall of all the models.

Table 6. 8 Overall performance of all model

Model Accuracy				
CNN Model	Precision	Recall	F1	Accuracy
MobileNet V2	0.93	0.94	0.93	0.94
ResNet50	0.97	0.96	0.96	0.96
Inception V4	0.84	0.79	0.79	0.80

The performance of each model is affected by various factors, making it challenging to determine why one model might outperform another. For the models in question, there are several possible reasons. Each CNN model in the research has distinct design features tailored for specific tasks. MobileNet V2, for instance, is recognized for its efficient and lightweight structure, making it perfect for mobile and embedded vision applications, with inverted residuals and linear bottlenecks that boost performance and maintain high accuracy despite its small size. ResNet50 employs residual learning to train deep networks effectively, addressing the vanishing gradient issue. Inception V4 is a more intricate model that utilizes multiple-sized kernels and different modules to extract a wide array of features, making it well-suited for complex tasks.

An accuracy between 70% and 90% is often considered good. Models achieving below 70% accuracy might need further optimization or fine-tuning to the model (Sebastian Raschka, 2018). However, accuracy alone can be misleading, especially with imbalanced datasets. Other metrics like precision, recall, and F1 score which provide a more comprehensive evaluation of a model's performance. For highly imbalanced datasets, the F1 score is often more informative than accuracy (Ashokkumar Palanivinayagam et al., 2023).

6.6. Model Computational Load

The result of object detection using YOLO V8 is as shown in Table 6.9, with its speed of detection and the average speed for both accident and non-accident videos.

Table 6. 9 Speed report of YOLO V8

Class video	Detection Speed	Average speed
Accident	25.23ms/frame	27.85ms/frame
Non Accident	30.47ms/frame	

The model successfully detects objects like cars, bus, motorcycle, and other vehicles with the average speed of 25.23ms/frame for accident and 20.47ms/frame speed for non-accident. For the overall average speed of both classes is 27.85ms/frame. This shows a good performance with a low computational load given the time taken for the model to process the video.

For the CNN model, by examining the time and number of parameters, we can estimate the computational load of each model presented in Table 6.10. The total parameters in each model represent the cumulative weights and biases that the model learns during training. The parameter size helps approximate the memory usage (RAM or GPU memory) required to store these parameters during training and inference. Additionally, the total training time indicates the duration of the training process, reflecting whether the model requires a high or low computational load.

Table 6. 10 Computational load of all model

Model	MobileNetV2	ResNet50	Inception V4
Total Parameters	2719266	24270178	41709410
Parameter size (MB)	10.37 MB	92.58 MB	159.11 MB
Total time for training	45 min 50 sec	3 hrs 2 min 12 sec	4 hrs 2 min 54 sec

MobileNetV2 is the most efficient option for environments with limited resources due to its low computational load. It features the fewest parameters at 2,719,266, a parameter size of 10.37 MB, and the shortest training time of 45 minutes and 50 seconds. In comparison, ResNet50, which provides high accuracy, includes 24,270,178 parameters and a parameter size of 92.58 MB, resulting in a training time of 3 hours, 2 minutes, and 12 seconds. Inception V4, while having the lowest accuracy among the three, requires the most computational resources, with 41,709,410 parameters, a parameter size of 159.11 MB, and a training duration of 4 hours, 2 minutes, and 54 seconds. Therefore, MobileNetV2 is best suited for applications that need rapid deployment and minimal resource use, whereas ResNet50 and Inception V4 are more fitting for cases with ample computational resources. This are some of many factors that we can roughly calculate if the model needs a high or low computational load.

CHAPTER VII: CONCLUSION AND SUGGESTION

7.1. Conclusion

This research set out to explore the potential of deep learning models in detecting accidents on traffic roads with limited dataset, specifically focusing on Object Detection using YOLO V8 and Convolutional Neural Networks (CNN) for image classification tasks.

The YOLO V8 model is able to filter the dataset to feed into the CNN model for it to detect if there are any accidents or not in the image frame. The YOLO V8 exhibited an average processing time of 27.85ms per frame. The CNN models demonstrated significant prediction accuracy with an imbalanced dataset, achieving above 80 % for all models. The training times varied according to the complexity of each model, with MobileNet V2 which takes only 45 minutes and 50 seconds, indicating a lower computational load for MobileNet V2. With that being said, YOLO V8 and MobileNetV2 is suitable for this research since it gives the shorter time to detect and shorter time to train giving the model overall a low computational load. Despite the limited dataset and accuracy not being the highest amongst the three model, MobileNet V2 a accuracy of 94% with is a reasonable value for an accuracy.

7.2. Suggestion

a. Researchers can develop models that balance optimal accuracy and computational load using limited datasets.
b. To reduce computational load, it's recommended to improve data pre-processing techniques, like video frame processing.
c. Lowering the frame rate can speed up object detection and reduce computational demands.
d. Model accuracy can be improved by using pre-trained CNN models with faster processing times and lighter computational loads.
e. Implementing data augmentation techniques can further enhance model accuracy.

REFERENCES

Abdullah, A. and Oothariasamy, J. (2020). Vehicle counting using deep learning models: A comparative study. *International Journal of Advanced Computer Science and Applications*, 11. doi:https://doi.org/10.14569/IJACSA.2020.0110784.

Aboah, A., Shoman, M., Mandal, V., Davami, S., Adu-Gyamfi, Y. and Sharma, A. (2021). A vision-based system for traffic anomaly detection using deep learning and decision trees. pp.4202–4207. doi:https://doi.org/10.1109/CVPRW53098.2021.00475.

Dong, L., Yali, L., Fei, H. and Shengjin, W. (2013). Object detection in image with complex background. doi:https://doi.org/10.2991/icmt-13.2013.58.

He, K., Zhang, X., Ren, S. and Sun, J. (2016). Identity mappings in deep residual networks. pp.630–645. doi:https://doi.org/10.1007/978-3-319-46493-0_38.

Husein, A.M., Lubis, N., Salim Sidabutar, Daniel, Yuanda, Y., Kevry and Waren, A. (2024). Computer VisionBased Intelligent Traffic Surveillance: MultiVehicle Tracking and Detection. *SinkrOn*, [online] 8(1), p.384391. doi:https://doi.org/10.33395/sinkron.v9i1.13204.

Khan, S.W., Hafeez, Q., Khalid, M.I., Alroobaea, R., Hussain, S., Iqbal, J., Almotiri, J. and Ullah, S.S. (2022). Anomaly detection in traffic surveillance videos using deep learning. *Sensors*, 22, p.6563. doi:https://doi.org/10.3390/s22176563.

Lindsay, G.W. (2020). Convolutional neural networks as a model of the visual system: Past, present, and future. *Journal of Cognitive Neuroscience*, [online] 33, pp.2017–2031. Available at: https://api.semanticscholar.org/CorpusID:210839618.

Luo, J., Fang, H., Shao, F., Zhong, Y. and Hua, X. (2020). Multi-Scale traffic vehicle detection based on faster R-CNN with NAS optimization and feature enrichment. *Defence Technology*, 17. doi:https://doi.org/10.1016/j.dt.2020.10.006.

Palanivinayagam, A., El-Bayeh, C.Z. and Damaševičius, R. (2023). Twenty years of machine-learning-based text classification: A systematic review. *Algorithms*, [online] 16. doi:https://doi.org/10.3390/a16050236.

Raschka, S. (2018). *Model evaluation, model selection, and algorithm selection in machine learning*.

Robles-Serrano, S., Sanchez-Torres, G. and Branch-Bedoya, J. (2021). Automatic Detection of Traffic Accidents from Video Using Deep Learning Techniques. *Computers*, [online] 10(11), p.148. doi:https://doi.org/10.3390/computers10110148.

Sabharwal, A. and Selman, B. (2011). S. Russell, p. Norvig, artificial intelligence: A modern approach, third edition. *Artif. Intell.*, 175, pp.935–937. doi:https://doi.org/10.1016/j.artint.2011.01.005.

Sarker, I. (2021). Deep learning: A comprehensive overview on techniques, taxonomy, applications and research directions. *SN Computer Science*, 2. doi:https://doi.org/10.1007/s42979-021-00815-1.

Suparmaji, A. and Wahyono, dan (2023). Combination of convolutional neural network and AdaBoost for breast cancer diagnosis. *E3S Web of Conferences*, 465. doi:https://doi.org/10.1051/e3sconf/202346502053.

Szegedy, C., Ioffe, S., Vanhoucke, V. and Alemi, A. (2016). Inception-v4, inception-resnet and the impact of residual connections on learning. *AAAI Conference on Artificial Intelligence*, 31. doi:https://doi.org/10.1609/aaai.v31i1.11231.

Tamagusko, T. and Ferreira, A. (2023). Machine learning for prediction of the international roughness index on flexible pavements: A review, challenges, and future directions. *Infrastructures*, 8, p.170. doi:https://doi.org/10.3390/infrastructures8120170.

Tian, E. and Kim, J. (2023). Improved vehicle detection using weather classification and faster r-cnn with dark channel prior. *Electronics*, 12, p.3022. doi:https://doi.org/10.3390/electronics12143022.

Yadav, D., Jain, A., Asati, S. and Yadav, A.K. (2023). Video anomaly detection for pedestrian surveillance. pp.489–500. doi:https://doi.org/10.1007/978-981-19-7867-8_39.

Zhao, Y., Wu, W., He, Y., Li, Y., Tan, X. and Chen, S. (2021). *Practices and a strong baseline for traffic anomaly detection*. [online] 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW). Available at: https://api.semanticscholar.org/CorpusID:234336284.