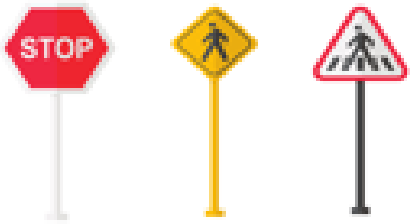
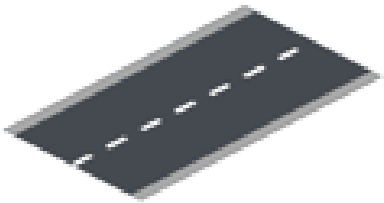
Computer Vision for Road Safety

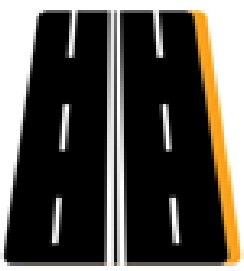
Using Deep Learning to Classify and Detect Road Signs and Pavement Defects



Karim Ashour
Selvia Nafaa
Doaa Emad
Rana Mohamed
Hafsa Essam



Supervisors
Dr. Mohamed Elhenawy
Dr. Abdallah Hassan



ACKNOWLEDGMENTS

First and foremost, we would like to thank Dr. Mohammed Elhenawy, and Dr. Abdullah Hassan, for their support, outstanding guidance and encouragement throughout our senior project.

Dr. Mohammed Elhenawy

Senior Research Fellow at Faculty of Health, Queensland University of Technology.

We extend our sincere thanks and gratitude to you for the effort exerted by your presence during the past period, we have learned more lessons and difficulties that we faced and were able to solve, we wish you success and thank you.

Dr. Abdullah Hassan

Assistant Professor, Computers and Systems Engineering Department, Minya University.

We thank you very much for all that you have given us Dr. Abdullah, you are an example of a wonderful doctor, we have benefited and learned many, much from you and we wish you good luck in the next Thank you.

We have submitted two extended abstracts in this topic to the Australasian Road Safety Conference 2023 for publication. The abstracts have therefore been accepted for publication.

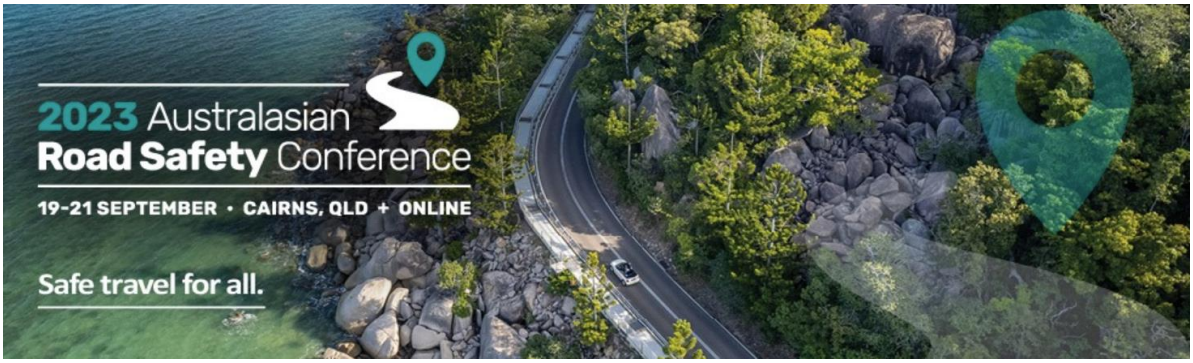


Fig 1.1: Australasian Road Safety Conference 2023

So, we covered in this book two use cases:

1- Part one (road signs classification and detection)
2- Part two (pavement defects detection)

ABSTRACT

Safety is dependent on the classification and detection of road signs, especially as automated road asset management and autonomous cars become more common. Road sign classification and detection are difficult due to several elements, such as lighting, weather, motion blur, and vehicle vibration. With the help of the German Traffic Sign Recognition Benchmark (GTSRB) Dataset, we were able to classify data with 99.17% accuracy using a group of carefully honed pre-trained CCN networks. In the detection task, we employed the Tiny LISA Traffic Sign Detection Dataset, and the mean average precision of the test data was 96.9%.

Road safety can be greatly impacted by pavement defects, thus finding, and fixing them is essential. The current state of practice in pavement problem detection requires a lot of labor and takes a long period. Many automobiles on the road today are equipped with cameras thanks to advancements in information and communication technology, creating vast amounts of crowdsourced data. This book highlights the use of deep learning to discover and categorize pavement defects due to the special characteristics of pavement defect detection, such as the significant foreground-background class imbalance.

To ensure accurate and dependable detection of pavement defects, we trained and tested various deep learning object detectors using the Road Damage Dataset 2022. The preliminary findings demonstrated that it is feasible to identify flaws quickly and effectively, lowering the probability of accidents and enhancing traffic safety. Additionally, by utilizing these techniques, maintenance and repair costs can be reduced, and the environmental impact of routine road inspections can be minimized.

By performing development approaches like semantic segmentation, merging, and cropping on RDD Japan dataset. And adjusting the loss function of the YOLO (you only look once) model we were able to detect 7 classes by 63.4 mAP and a precision of 66.9

Table of Contents

List of Figures

List of Tables

Chapter 1: Introduction and Models

Introduction

Road signs and pavements are two essential components of a safe and efficient transportation system. Road signs provide drivers with information about the road conditions and traffic regulations, while pavements ensure that vehicles can travel safely and smoothly. However, both road signs and pavements can deteriorate over time, which can lead to accidents and traffic congestion. [1]

Road signs detection is the process of identifying and locating road signs in images or videos. This information can be used to improve navigation systems, to ensure that road signs are visible to drivers, and to detect missing or damaged signs. Pavement defects detection is the process of identifying and locating pavement defects in images or videos. This information can be used to improve road maintenance, to prevent accidents, and to ensure that the road surface is safe for travel. [2]

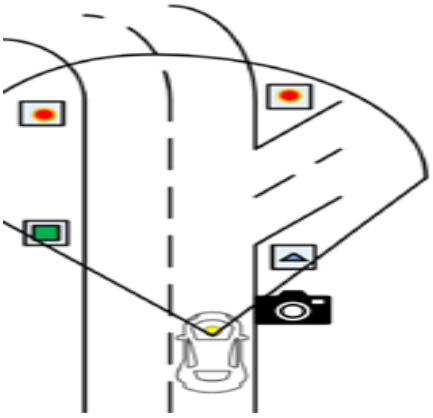


Fig 1.2: Vehicle with dash cams system detects all the viewed symbols [3].

The combination of road signs detection and pavement defects detection can help to improve road safety and efficiency. By identifying and locating road signs and pavement defects, we can make sure that drivers have the information they need to travel safely, and that the road surface is in good

condition. This can help to reduce accidents, traffic congestion, and environmental damage.

Overall, road signs detection and pavement defects detection are two important technologies that can help to improve road safety and efficiency. By combining these two technologies, we can make our roads safer and more efficient for everyone.[4]

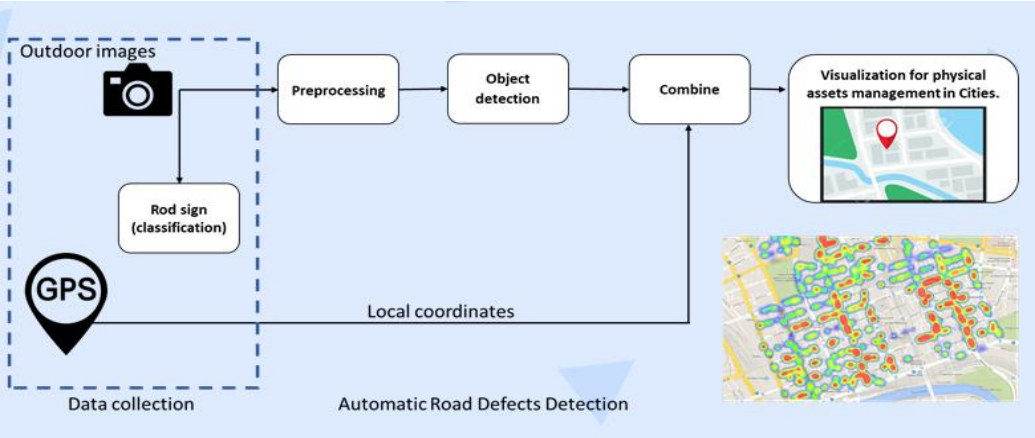


Fig 1.3: project application

Enhanced digital maps are used to store information about road signs, pavement defects, and other features of the road environment. This information can be used by autonomous vehicles (AVs) to navigate safely and efficiently.

For example, an AV can use the information in an enhanced digital map to identify road signs and traffic signals. This allows the AV to obey traffic laws and avoid collisions. The AV can also use the information in the map to detect pavement defects, such as potholes and cracks. This allows the AV to avoid these defects and prevent damage to its tires and suspension.

In addition to helping AVs navigate safely, enhanced digital maps can also be used to improve the efficiency of road maintenance. For example, the information in the map can be used to identify areas of the road that need to be resurfaced or patched. This information can then be used to prioritize road maintenance projects and ensure that roads are kept in good condition.

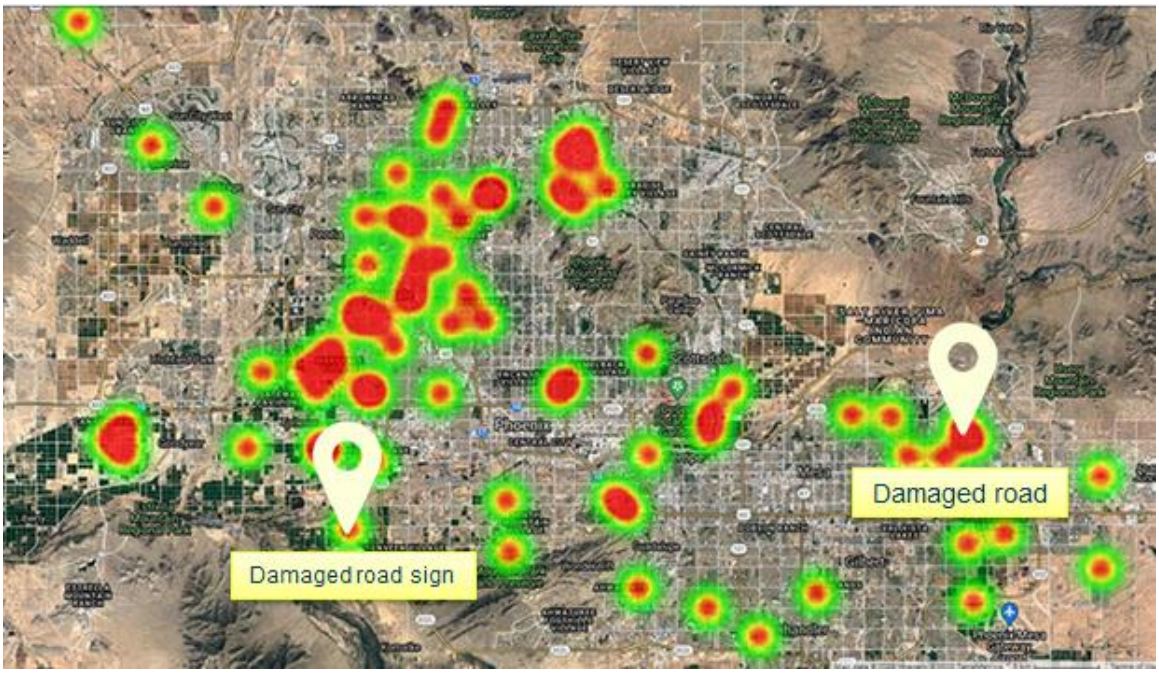


Fig 1.3: enhanced digital maps [5].

Overall, enhanced digital maps can play a valuable role in improving the safety, efficiency, and sustainability of road transportation. By storing information about road signs, pavement defects, and other features of the road environment, enhanced digital maps can help AVs navigate safely, improve the efficiency of road maintenance, and reduce traffic congestion.

Models

The You Only Look Once (YOLO) algorithm is a single-stage deep learning algorithm for target recognition. It complements traditional machine learning approaches for object detection by utilizing pattern recognition techniques. YOLO is based on the AlexNet architecture and stands out for its ability to provide real-time response, even on devices with limited computational power. By performing a single forward propagation through the neural network, YOLO can quickly determine predictions. Since its inception, YOLO has been widely utilized in object recognition tasks. Over time, it has undergone several versions of development, from v1 to v8. Integrating the most suitable YOLO model for road sign and pavement detection into road maintenance practices would greatly benefit the road maintenance community.

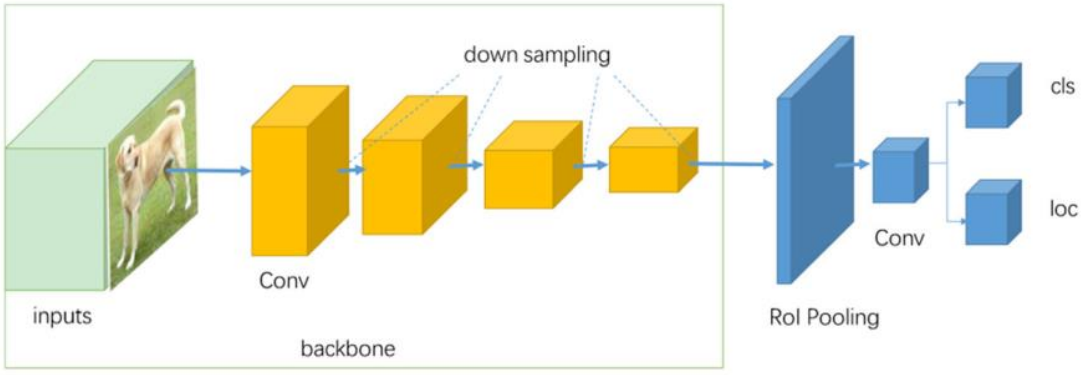


Fig 1.5: Generic architecture of single stage object detector

YOLOv5 introduction:

Glen Jocher, the founder, and CEO of Ultralytics, introduced YOLOv5 a few months after the release of YOLOv4 in 2020. YOLOv5 is an open-source algorithm known for its user-friendly nature, making it convenient to train, use, and deploy. Ultralytics offers a mobile version of YOLOv5 for both iOS and Android platforms, along with various integrations for labeling, training, and deployment processes. Data scientists often opt for YOLOv5 when working on projects involving classification tasks with freely available datasets from platforms like Kaggle. While YOLOv5 incorporates several improvements from YOLOv4, it differs in that it is developed using the PyTorch framework instead of Darknet. Notably, YOLOv5 uses a pre-training tool called AutoAnchor, developed by Ultralytics, to ensure that the anchor boxes used in the dataset and training settings are properly adjusted. AutoAnchor first applies a k-means function to the dataset labels to generate initial conditions for the Genetic Evolution (GE) algorithm. The GE algorithm then evolves the anchor boxes over 1000 generations by default, finding a set of anchors that are well-suited for the dataset and training settings. The algorithm uses CIoU loss to measure the similarity between predicted and ground-truth bounding boxes, and Best Possible Recall to measure the number of ground-truth bounding boxes that are correctly predicted.

Overview of the YOLOv5 Architecture

YOLO was a groundbreaking object detection model that was the first to predict bounding boxes and class labels in a single, end-to-end differentiable network. The YOLO network consists of three main parts:

- Backbone: The backbone of YOLOv5 is based on the CSPDarknet53 architecture. This architecture is a modified version of the Darknet53 architecture. The CSPDarknet53 architecture uses a number of techniques to improve the performance of the model, such as cross stage partial connections and bottleneck layers.
- Neck: The neck is a module that connects the backbone and the head of a YOLOv5 object detection model. It is responsible for fusing the features extracted by the backbone with the predictions made by the head. In YOLOv5, two different neck structures are used: SPPF and New CSP-PAN.
- Head: It uses a technique called Single-Shot MultiBox Detector (SSD) to predict the bounding boxes for objects. SSD does not use pre-defined anchor boxes, which can make it more accurate and efficient than anchor-based detection.

YOLOv5 uses the CSPDarknet53 architecture as its backbone, with an SPP layer for spatial pyramid pooling. The PANet neck then merges features from different layers of the backbone, and the YOLO detection head predicts bounding boxes and class labels. This architecture is further optimized by using techniques such as data augmentation and transfer learning.

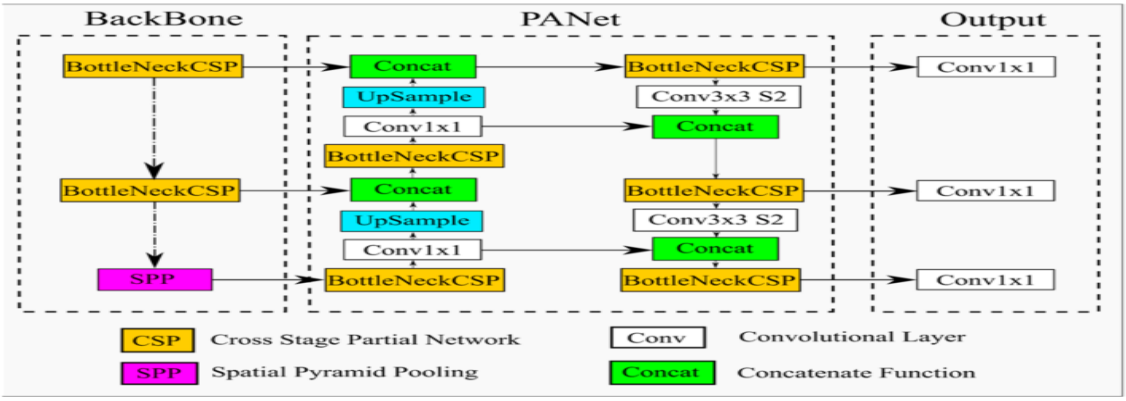


Fig 1.6: network architecture of yolov5

Some of YOLOv5's most important advantages over previous versions are smaller volume, higher speed, more precision, achieves most of its performance improvements through optimizations to its PyTorch implementation, while the model architecture itself remains largely similar to YOLOv4

Augmentation in YOLOv5:

During the training process, YOLOv5 employs a data loader that applies online data augmentation to each training batch. This augmentation technique helps enhance the model's performance. The data loader utilizes various augmentations techniques. These techniques include Mosaic, copy-paste, random affine, MixUp, HSV augmentation, random horizontal flip, and others from the albumentations package. The performance of object detection models on the COCO benchmark can be increased by using the powerful image augmentation technique known as mosaic augmentation. Addressing the "small object problem" is one of its main applications. Five scaled versions of YOLOv5 are provided, each with a variable width and depth of convolution modules to accommodate different applications and hardware limitations. These versions are trained using the COCOval 2017 dataset benchmark: YOLOv5n and YOLOv5s are lightweight variants intended for low-resource devices, whilst YOLOv5l and YOLOv5x are optimized for high performance. YOLOv5n stands for nano, YOLOv5s for small, YOLOv5m for medium, YOLOv5l for big, and YOLOv5x for extra-large.YOLOv5x is the largest and most complex model, but it also offers the highest accuracy.

Pretrained Models :

YOLOv5 provides five scaled versions that are trained on COCOval 2017 dataset benchmark ,each with varying width and depth of convolution modules to suit different applications and hardware requirements: YOLOv5n (nano), YOLOv5s (small), YOLOv5m (medium), YOLOv5l (large), and YOLOv5x (extra-large), YOLOv5n and YOLOv5s are lightweight models that are targeted for low-resource devices, while YOLOv5l and YOLOv5x are optimized for high performance. YOLOv5x is the largest and most complex model, but it also offers the highest accuracy.

Model	Size (pixels)	mAPval 50-95	mAPval 50	Speed CPU b1 (ms)	Speed v100 b1 (ms)	Speed V100 b32 (ms)	Params (M)	FLOPs @640(B)
YOLOv5n	640	28.0	45.7	45	6.3	0.6	1.9	4.5
YOLOv5s	640	37.4	56.8	98	6.4	0.9	7.2	16.5
YOLOv5m	640	45.4	64.1	224	8.2	1.7	21.2	49.0
YOLOv5l	640	49.0	67.3	430	10.1	2.7	46.5	109.1
YOLOv5x	640	50.7	68.9	766	12.1	4.8	86.7	205.7
YOLOv5n6	1280	36.0	54.4	153	8.1	2.1	3.2	4.6
YOLOv5s6	1280	44.8	63.7	385	8.2	3.6	12.6	16.8
YOLOv5m6	1280	51.3	69.3	887	11.1	6.8	35.7	50.0
YOLOv5l6	1280	53.7	71.3	1784	15.8	10.5	76.8	111.4
YOLOv5x6 +TTA	1280 1536	55.0 55.8	72.7 72.7	3136 -	26.2 -	19.4 -	140.7 -	209.8 -

Table1.1: Pretrained models' Checkpoints in yolov5

YOLOv5 Labeling Format:

Similar to YOLO Darknet TXT, the YOLOv5 PyTorch TXT annotation format also includes a YAML file with model configuration and class values. If your annotation tool does not support YOLOv5, you will need to convert annotations to work with YOLO

YOLOv8:

YOLOv8 is the latest version of the YOLO object detection model, and it offers a number of improvements over previous versions. One of the biggest advantages of YOLOv8 is its use of an anchor-free model. Anchor-free models do not use pre-defined anchor boxes, which can make them more accurate and efficient than anchor-based models.

Overview of the YOLOv8 Architecture

Its model architecture consists of Backbone, Neck, and Head. We introduce the design concepts of each part of the model architecture.

- Backbone :The backbone of our model is based on the CSPDarknet53 architecture to split the feature map into two parts.
  - The first part uses convolution operations.
  - The second part is concatenated with the output of the previous part.

  to improve the learning ability of the CNNs and reduce the computational cost of the model.

- Neck : YOLOv8 introduces C2f module by combining the C3f module (3 ConvModule and n DarknetBottleNeck) and the concept of ELAN from YOLOv7.The C2f module (2 ConvModule and n DarknetBottleNeck)connected through Split and Concat. Unlike YOLOv5, we use the C2f module instead of the C3f module.
  It uses both Feature Pyramid Network (FPN) and Path Aggregation Network (PAN) architectures to create a fusion of features obtained from the different layers of the network and pass that to the head.

- Head: Different from YOLOv5 model utilizing a coupled head, they split head, where the classification and detection heads are separated.
  It uses a technique called anchor-free detection to predict the bounding boxes for objects. Anchor-free detection does not use pre-defined anchor boxes, which can make it more accurate and efficient than anchor-based detection.

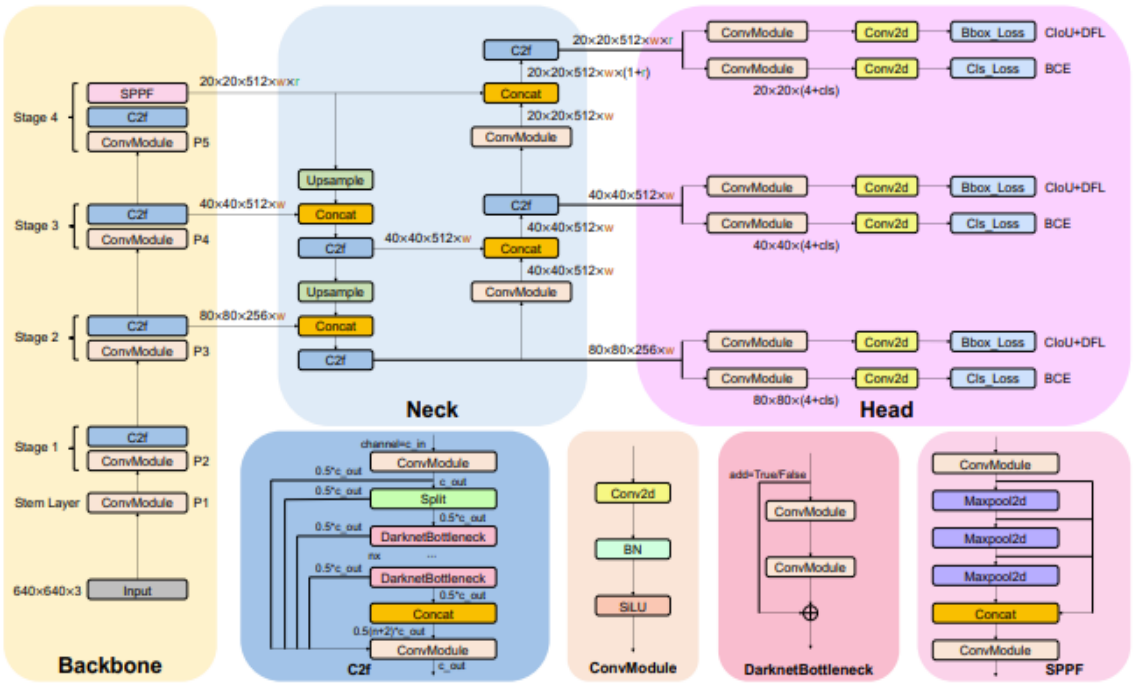


Fig 1.7 : detailed description of our model architecture (yolov8)

Pretrained Models:

YOLOv8 provides five scaled versions that are trained on COCOval 2017 dataset benchmark: YOLOv8n (nano), YOLOv8s (small), YOLOv8m (medium), YOLOv8l (large), and YOLOv8x (extra-large).

Model	Size (pixels)	mAPval 50-95	Speed CPU ONNX (ms)	Speed A100 TensorRT (ms)	Params (M)	FLOPs (B)
YOLOv8n	640	37.3	80.4	0.99	3.2	8.7
YOLOv8s	640	44.9	128.4	1.20	11.2	28.6
YOLOv8m	640	50.2	234.7	1.83	25.9	78.9
YOLOv8l	640	52.9	375.2	2.39	43.7	165.2
YOLOv8x	640	53.9	479.1	3.53	68.2	257.8

Table 1.2 : Pretrained models' Checkpoints in yolov8

The comparison between YOLOv5 and YOLOv8:

Table 1.3: Comparison between YOLOv5 & YOLOv8

Feature	YOLOv5	YOLOv8
Anchor-based or anchor-free	Anchor-based	Anchor-free
Head architecture	Single-stage head	Split Ultralytics Head
Training algorithm	Darknet	CSPDarknet
Data augmentation strategy	Image augmentation	Image and instance segmentation augmentation
Accuracy	High	Very high

Evaluation metrics:

Drawing a bounding box around each object that is detected in image makes object detection a particularly difficult problem. The metrics of accuracy, precision (P), recall (R), mean average precision (mAP), and F1-score were used to assess each YOLOv5 or YOLOv8 trained model's performance.
As shown in Equations (1)–(4):

$$precision = \frac{tp}{tp+fp} \quad (1)$$

$$recall = \frac{tp}{tp+fn} \quad (2)$$

$$MAP50 = \frac{1}{N}\sum_{N}^{i=1} API \quad (3)$$

$$f1-score = 2 \times \frac{precision \times recall}{precision + recall} \quad (4)$$

In object detection, true positive (TP), false positive (FP), false negative (FN), and intersection over union (IoU) are the four main metrics used to

assess an algorithm's performance.TP is the accurate identification of an object that is present in the image. FP refers to an inaccurate object detection. An object that the network can see in the image but is not detected is designated as FN.

IoU calculates the amount of overlap between the ground truth bounding box of the real object and the predicted bounding box. To ascertain whether a detection is accurate or inaccurate, a parameter known as the IoU threshold is used. the collection of suggested object pixels in A and the collection of actual object pixels in B, and calculates:

$$IOU(A,B) = \frac{A \cap B}{A \cup B}; IOU(A,B) \in [0,1] \qquad (5)$$

The IoU threshold is a value that determines whether a detection is considered correct or incorrect. A detection is considered correct if its IoU score is greater than the threshold. Otherwise, the detection is considered incorrect.

The AP metric is a measure of the performance of an object detection algorithm at a specific level of recall. The AP metric is calculated for different values of the IoU threshold. Therefore, it is necessary to specify the IoU threshold when calculating the AP metric.

Part 1

Road Signs Classification And Detection

Chapter 2: introduction and related work

Introduction

Roadside traffic signs are erected to notify drivers of impending hazardous driving conditions and to display pertinent information. Sometimes, excessive traffic, bad weather, or drivers who aren't paying attention put them at danger of missing a sign, which can result in accidents. Real-time traffic sign detection, which involves locating traffic signs in real-world photos, is a difficult computer vision challenge with significant practical value. Advanced driver assistance systems (ADAS) supporting the detection and recognition of traffic signs have entered the market, and a variety of methods have been developed. Most of the algorithms that are now in use employ color segmentation to look for both the location and the pixel of interest. There isn't a definite agreement on what the state-of-the-art is in this field despite the several competing theories. This can be explained by the dearth of thorough, objective comparisons of such techniques. By using the "LISA Dataset", we hope to fill this gap.

Related Work

The application for detecting traffic signs was and is still a difficult problem to tackle. It is suggested that several works attend a high performance. In the area of computer vision and the processing of visual data, deep learning approaches outperform traditional methods brilliantly. The R-CNN model was the first deep learning-based model for multi-class object recognition in the general contest, where several deep learning methods for object detection are offered.

Another well-known paradigm for object identification that prioritises speed is called YOLO (You Only Look Once). The YOLO model focuses on increasing inference speed and obtaining passable accuracy.[11]

A) Traffic sign recognition

The advent of artificial intelligence (AI) in recent years has updated the vehicle-aided driving system's pre-existing driving mode. The technology instantly reminds drivers to perform proper manoeuvres by gathering real-time information about the state of the road, therefore preventing automobile accidents caused by driver tiredness. The creation of autonomous cars

necessitates the quick and precise identification of traffic signs from digital imagery in addition to the auxiliary driving systems.

(TSR) is a difficult real-world computer vision challenge made to automatically detect traffic signs while a car is being driven. The signs offer useful real-time information on the state of the traffic, such as the speed limits and the existence of warnings. Traffic signs are made up of identical items whose size, colour, and shape adhere to a standard specification that makes them recognisable. traditional approaches based The detection stage and the classification stage are often engaged in the traffic sign identification pipeline, which traditionally bases itself mostly on colour and form patterns.[12]

CNN-based hierarchical learning, which is based on deep learning, has been used for years to analyse and recognise visual input. As a result, this deep network is gradually trained from low to high levels of abstraction, enabling the automatic identification and classification of traffic signals. Due to their depth, large-scale designs like ResNet and MobileNets have thus far produced state-of-the-art outcomes, which unquestionably affect the interest system's performance[13].

Semantic road detection

The advancement of artificial intelligence systems has led to the appearance of autonomous cars on public roads today. Perceiving and identifying roads such that the driver may subsequently be quickly alerted to any possible hazard or risk is the difficult challenge when designing autonomous driving systems. The most common method for identifying and detecting roads is camera-based vision.

1. Traditional techniques based.

Road identification systems often rely on the road's visual properties, such colours, forms, etc., which are not always simple to identify. These systems' resilience rely on their capacity to handle the issue of light sensitivity and fluctuating brightness in outdoor settings. Traditional road segmentation approaches have frequently employed colour- and shape-based methods for detecting and identifying road regions.

2. **Deep learning based.**
   One of the most challenging jobs in computer vision is CNN-based semantic segmentation, which is essential for comprehending traffic scenes. When compared to an image classification problem, this method entails intensively categorising every pixel in an image (pixel-wise prediction) using a segmented ROI's spatial mask prediction.[14][15]

Literature review

In order to maintain traffic flow efficiency and safety, proper inventory management of traffic signs is a crucial duty. This task is often carried out manually. Using a camera mounted on a moving vehicle, traffic signs are photographed, and manual localization and recognition are carried out offline by a human operator to ensure compatibility with the database. Applying such hard labor to thousands of km of roads, however, can take a very long time. By speeding up the identification of broken or missing traffic signs, automation would drastically reduce the amount of physical labor required for this operation and increase safety.

Replace human localization and traffic sign recognition with automated detection as a vital first step in automating this work. The issue of traffic sign recognition has previously attracted a lot of interest in the computer vision field, and effective detection and recognition methods have already been suggested. However, these solutions have only been developed for a few categories, mostly for traffic signs connected to autonomous cars and advanced driver assistance systems (ADAS).

Still up for debate that detection and recognition of a lot of traffic-sign categories. The challenge of recognizing and detecting traffic signs has been covered by a number of prior standards. The problem of traffic-sign detection (TSD), which is significantly more difficult to solve, was disregarded by some of them, who only concentrated on traffic-sign recognition (TSR). Other benchmarks that do address TSD often only cover a portion of the categories of traffic signs, typically those that are crucial for ADAS and autonomous vehicle applications. The majority of categories found in such benchmarks have a recognizable look and little inter-category volatility, making it possible to identify them using manually created detectors and classifiers. Round mandatory signs or triangular prohibitory signs are two

examples. However, due to their wide variance in appearance, many other traffic-sign classes that are not covered by the current criteria can be even more challenging to identify. Examples of these categories may differ greatly from one another in terms of real-world dimensions, aspect ratios, colors, and text and symbols (such as arrows). Due to the similarities in appearance of items from other categories, this frequently results in a high degree of intra-category (i.e. within-category) appearance variation and a low degree of inter-category (i.e. between-categories) variability.

One alternative would be to adapt current techniques using hand-crafted features and classifiers to handle these categories; but, doing so would take a lot of effort, especially given that many traffic-sign looks vary between nations. Use of feature learning based on actual instances is a far more sensible approach. This is simply adaptable and can capture a great degree of appearance heterogeneity across a variety of traffic signs. Recent developments in deep learning have shown hopeful outcomes for the detection and recognition of public objects. Some studies have already applied deep learning techniques for traffic sign detection and recognition, but they had only evaluated a small fraction of traffic sign categories. Lack of a comprehensive dataset with several hundred different categories and enough cases for each category is one of the key obstacles preventing deep learning from being applied to a wide collection of traffic-sign categories. Given that deep learning models comprise tens of millions of learnable parameters and that overfitting must be avoided, this issue is especially crucial in this context.

Chapter 3: Road signs datasets collection

Classification of road signs datasets

The 43 classes of traffic signs in the German Traffic Sign Recognition Benchmark (GTSRB) are divided into 12,630 test pictures and 39,209 training images. The pictures have varied lighting and beautiful settings

Fig 3.1: the German Traffic Sign Recognition Benchmark (GTSRB)

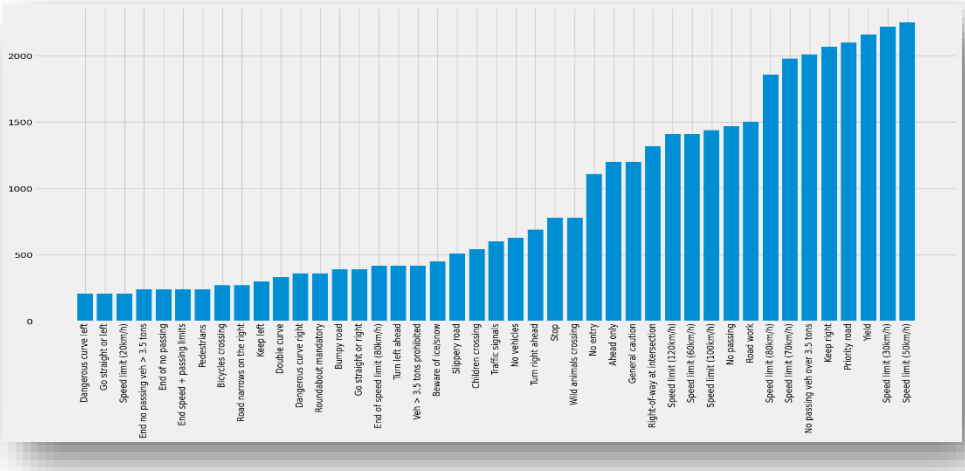


Fig 3.2: GTSRB dataset Classes

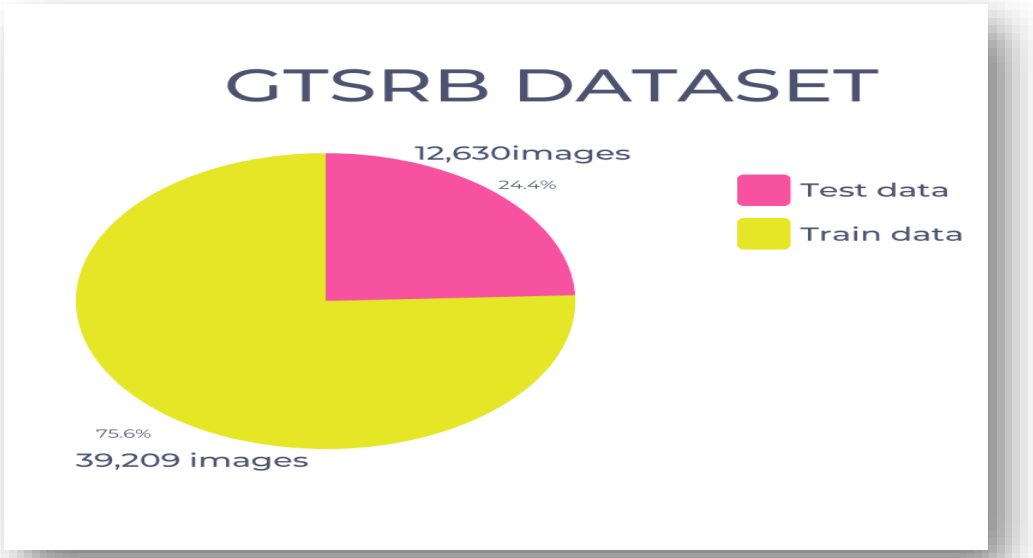


Fig 3.3: GTSRB dataset split.

Detection of road signs datasets

With the development of autonomous driving, there has been a focus on gathering data with all different kinds of road conditions, signs, and any element to notice while driving, leading to a multitude of datasets in the community specific for traffic sign detection.

Global traffic signs are frequently the topic of several datasets. For instance, The Laboratory for Intelligent and Safe Automobiles (LISA) Dataset and the German Traffic Sign Detection Benchmark (GTSDB). The ability to access traffic signs from all over the world is helpful to the computer vision community, but there is a critical disadvantage that applies to all public available traffic sign datasets: the absence of sun glare in the photos. We used tiny LISA Dataset because we cannot get access to LISA dataset, always get error **404**.

There are many datasets we used it but it doesn't effective i.e.: road sign detection dataset it contents very few number of classes (only 4 classes) , Traffic Sign Localization Detection YOLO Annotated dataset it contents 36 classes but it was unbalanced dataset and GTSDB dataset it not annotated and it's very difficult and expensive to annotate it by our self.

LISA Dataset:

Both standalone frames and videos are part of the LISA data set. Although not all frames have been extracted for annotation, every frame that has been marked may be linked back to the original video; as a result, the annotations can be used to validate systems using tracking.

LISA Dataset Statistics:

- Classification and detection.
- 47 classes.
- 6,610 total images.
- Including a video.
- From the United States.
- 6x6 to 167x168 pixel sign sizes.
- There are both color and grayscale images.
- The year is 2012.
- Pictures taken with various cameras.
- Images can be between 640x480 and 1024x522 pixels in size.

The differences between LISA and Tiny LISA dataset are:

- Classes of Tiny LISA: 9
- Total Image of Tiny LISA: 900

Class	Number Of Images
Stop	210
Merge	100
SpeedLimit25	80
SpeedLimit35	110
Yield	45
YieldAhead	45
SignalAhead	100
KeepRight	110
PedestrianCrossing	100

Table 3.1: Traffic sign classes in Tiny LISA dataset

Chapter 4: Classification of Road Signs

Introduction

The traffic signs that are now etched on the roads increase traffic safety by warning the driver of speed limitations and any additional potential hazards, such as deep exciting streets, unavoidable repair street operations, or any frequent crossroads. Vehicles are now a need for transportation in the daily movement of people due to the rapid advancement of the economy and innovation in the modern civilization [6].

A) **Description**

The primary goal is to categorise, identify, and recognise traffic signs using convolutional neural networks, which are composed of neurons with learnable weights and biases that aid in offering the high performance in recognising the traffic signs even in their challenging and susceptible settings. The dataset suggested by the Traffic Sign dataset will be loaded into this system first.[7]

One of the pretrained model architecture that we use it:
model_alex_net

```
* keras.layers.Conv2D(filters=128, kernel_size=(3,3), strides=(1,1),
  activation='relu', input_shape=(32,32,3)),
* keras.layers.BatchNormalization(),
* keras.layers.MaxPool2D(pool_size=(2,2)),
* keras.layers.Conv2D(filters=256, kernel_size=(3,3), strides=(1,1),
  activation='relu', padding="same"),
* keras.layers.BatchNormalization(),
* keras.layers.MaxPool2D(pool_size=(3,3)),
* keras.layers.Conv2D(filters=256, kernel_size=(3,3), strides=(1,1),
  activation='relu', padding="same"),
* keras.layers.BatchNormalization(),
* keras.layers.Conv2D(filters=256, kernel_size=(1,1), strides=(1,1),
  activation='relu', padding="same"),
* keras.layers.BatchNormalization(),
* keras.layers.Conv2D(filters=256, kernel_size=(1,1), strides=(1,1),
  activation='relu', padding="same"),
* keras.layers.BatchNormalization(),
* keras.layers.MaxPool2D(pool_size=(2,2)),
```

```
* keras.layers.Flatten(),
* keras.layers.Dense(512, activation='relu'),
* keras.layers.BatchNormalization(),
* keras.layers.Dropout(rate=0.5),
* keras.layers.Dense(43, activation='softmax')
```

Transfer Learning

There are instances where obtaining training data is costly or difficult. It is necessary to create high-performance learners who are taught using more easily accessible data from many disciplines. This technique is referred to as transfer learning.

A . Background

In several applications where it is feasible to identify patterns from previous data (training data) in order to predict future results, the study of data mining and machine learning has been widely and effectively used.

Finding training data that matches the feature space and anticipated data distribution parameters of the test data can be difficult and expensive in some circumstances.

As a result, a target domain has to be taught to a high-performance learner from a source domain that is related to the target domain. This is the concept underpinning transfer learning.

Because transfer learning is so widespread, it is unusual to train a model from scratch for tasks related to image or natural language processing.

Instead, data analysts and researchers prefer to begin with a model that has already been trained to identify common elements in pictures, such as edges and form, and to categorise things.

Inception, ImageNet, and AlexNet are common examples of models with transfer learning as their foundation.

Transfer learning is used to advance a student from one domain by transferring knowledge from a related area.[8].

B .Transfer Learning Steps

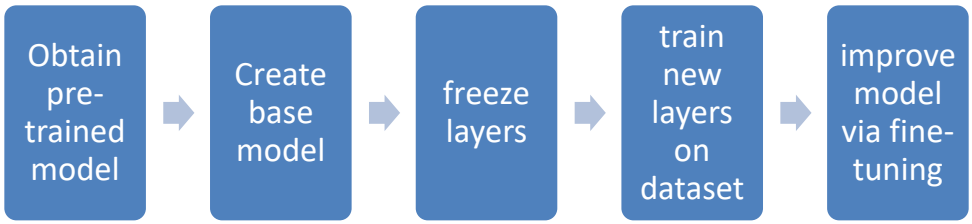


Fig 4.1: Transfer Learning Steps

1. Obtain pre-trained model:

Depending on the task, the initial step is to decide which pre-trained model we want to use as a basis of our training.
Here are the pre-trained models that we used it in our project:

- VGG-16
- XCeption
- Dense-Net
- Alex-Net

2. Create a base model:

The base model is one of the architectures that we choose in the first phase to be closely related to our objective, such as AlexNet or Xception.

3. Freeze layers:

To prevent having to make the model learn the fundamental features, the pre-trained model's initial layers must be frozen. We will lose all the learned knowledge if we do not freeze the early layers. This will be a waste of time, money, etc. and will be no different than training the model from start.

4. Add new trainable layers:

The feature extraction layers are the only data we are using from the basic model. To predict the specialized tasks of the model, we must build more layers on top of them. These are typically the last layers to be output.

5. Train the new layers.

The final output of the pre-trained model will almost certainly be different from the result we want for our model. Pre-trained models, for example that were trained on the ImageNet dataset, will produce 1000 classes. Our model must, however, function for two classes. In this situation, we need to train the model using a new output layer.

6. Fine-tune your model.

Unfreezing a portion of the base model and retraining the entire model on the full dataset at a very low learning rate are two steps in the fine-tuning process. The model's performance on the new dataset will improve thanks to the low learning rate, which also prevents overfitting.

C.Results

We have used many pre-trained models and made many attempts to reach the highest results in German Traffic Sign Recognition Benchmark (GTSRB) Dataset.

1. Vgg16 model

```
def create_model(input_shape, n_classes, optimizer='rmsprop', fine_tune=0):
    conv_base = VGG16(include_top=False,
                      weights='imagenet',
                      input_shape=input_shape)
    if fine_tune > 0:
        for layer in conv_base.layers[:-fine_tune]:
            layer.trainable = False
    else:
        for layer in conv_base.layers:
            layer.trainable = False
    top_model = conv_base.output
    top_model = Flatten(name="flatten")(top_model)
    top_model = Dropout(0.2)(top_model)
              output_layer = Dense(n_classes,
                    activation='softmax')(top_model)
```

```
model = Model(inputs=conv_base.input,
              outputs=output_layer)
    model.compile(optimizer=optimizer,
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
    return model
```

Vgg is consist of 16 layers:

- thirteen layers (covered both conv and pool)
- three (full connect layer)

We trained the output layer at epoch = 90 , batch-size = 256, shape-size =32*32. Then the results of model accuracy and loss are :

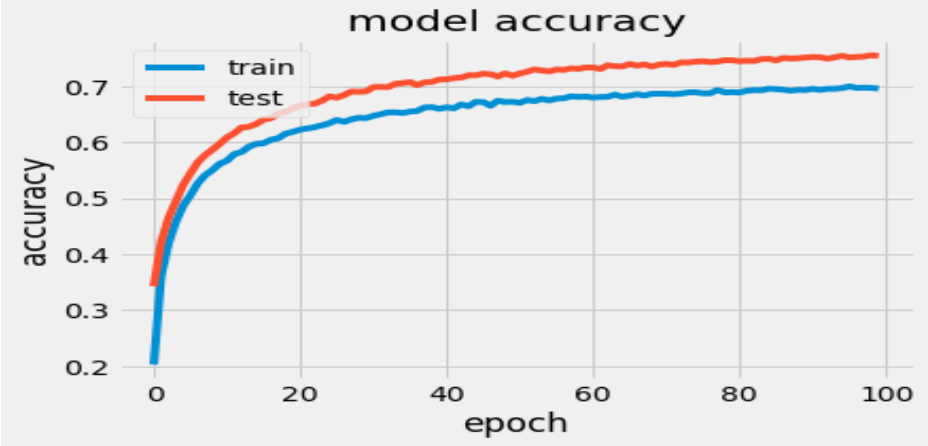


Fig 4.2: model accuracy of vgg16

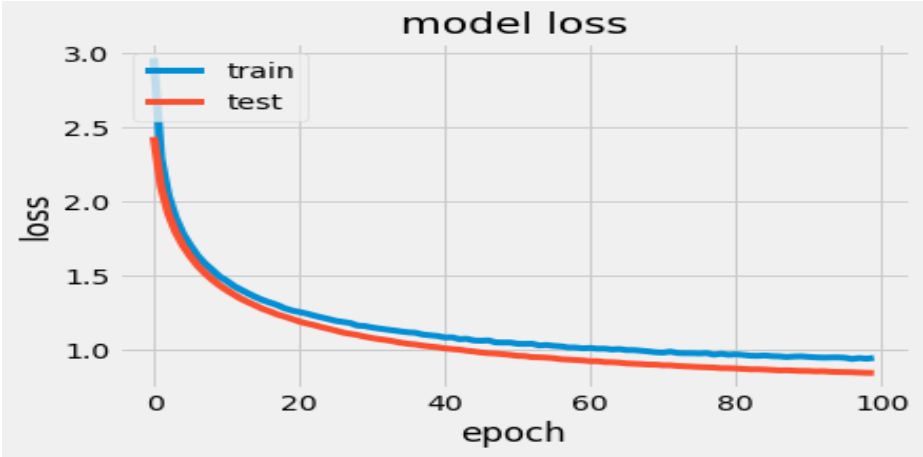


Fig 4.3: model loss of vgg16

The test data accuracy was poor 53%.
The confusion matrix was not good.

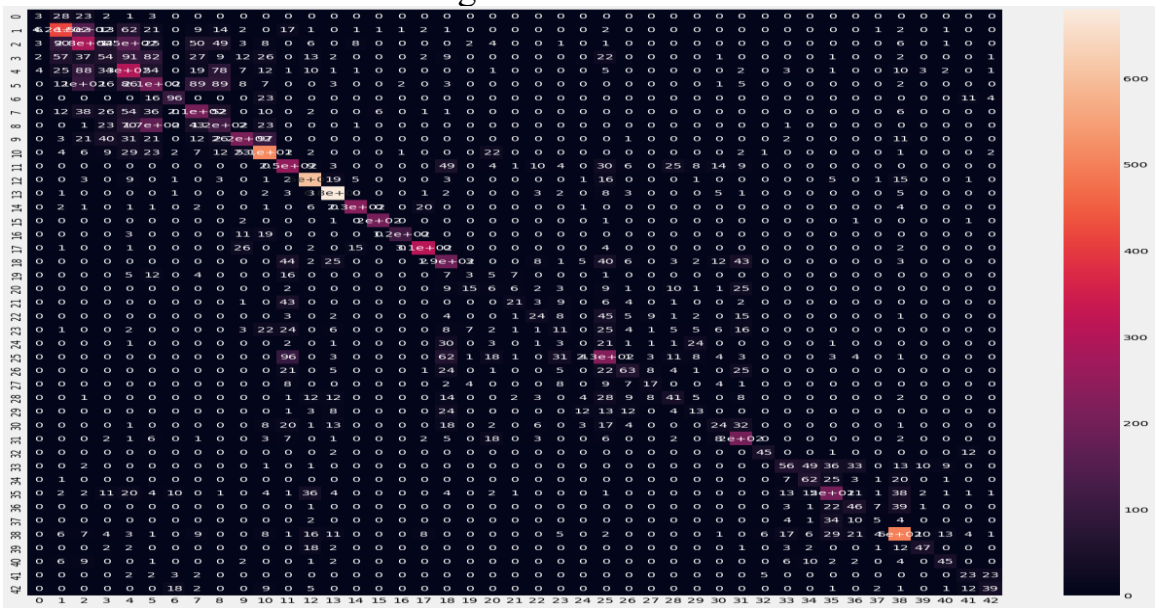


Fig 4.4: confusion matrix of vgg16

Fig 4.5: Predictions on Test Data of vgg16

Then we made epochs =100, batch-size =1024, and input-shape =64*64. Accuracy of test data was improved to 68%.

2. Alex-Net model

We adjust the architecture of alex-net in fully connected layers after maxPooling where we remove one from the three dense layers,and made the two dense layers only.

```
model=keras.models.Sequential([
    keras.layers.Conv2D(filters=128, kernel_size=(3,3), strides=(1,1),
activation='relu', input_shape=(32,32,3)),
    keras.layers.BatchNormalization(),
    keras.layers.MaxPool2D(pool_size=(2,2)),
    keras.layers.Conv2D(filters=256, kernel_size=(3,3), strides=(1,1),
activation='relu', padding="same"),
    keras.layers.BatchNormalization(),
    keras.layers.MaxPool2D(pool_size=(3,3)),
    keras.layers.Conv2D(filters=256, kernel_size=(3,3), strides=(1,1),
activation='relu', padding="same"),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2D(filters=256, kernel_size=(1,1), strides=(1,1),
activation='relu', padding="same"),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2D(filters=256, kernel_size=(1,1), strides=(1,1),
activation='relu', padding="same"),
    keras.layers.BatchNormalization(),
    keras.layers.MaxPool2D(pool_size=(2,2)),
    keras.layers.Flatten(),
    keras.layers.Dense(512, activation='relu'),
    keras.layers.BatchNormalization(),
    keras.layers.Dropout(rate=0.5),
    keras.layers.Dense(43, activation='softmax')
])
```

```
summery is:
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_70 (Conv2D)           (None, 30, 30, 128)       3584
_________________________________________________________________
batch_normalization_72 (Batch (None, 30, 30, 128)       512
_________________________________________________________________
max_pooling2d_42 (MaxPooling (None, 15, 15, 128)       0
_________________________________________________________________
conv2d_71 (Conv2D)           (None, 15, 15, 256)       295168
_________________________________________________________________
batch_normalization_73 (Batc (None, 15, 15, 256)       1024
_________________________________________________________________
max_pooling2d_43 (MaxPooling (None, 5, 5, 256)         0
_________________________________________________________________
conv2d_72 (Conv2D)           (None, 5, 5, 256)         590080
_________________________________________________________________
batch_normalization_74 (Batc (None, 5, 5, 256)         1024
_________________________________________________________________
conv2d_73 (Conv2D)           (None, 5, 5, 256)         65792
_________________________________________________________________
batch_normalization_75 (Batc (None, 5, 5, 256)         1024
_________________________________________________________________
conv2d_74 (Conv2D)           (None, 5, 5, 256)         65792
_________________________________________________________________
batch_normalization_76 (Batc (None, 5, 5, 256)         1024
_________________________________________________________________
```

```
max_pooling2d_44 (MaxPooling (None, 2, 2, 256)         0
_________________________________________________________________
flatten_14 (Flatten)         (None, 1024)              0
_________________________________________________________________
dense_40 (Dense)             (None, 512)               524800
_________________________________________________________________
batch_normalization_77 (Batc (None, 512)               2048
_________________________________________________________________
dropout_26 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_41 (Dense)             (None, 43)                22059
```

We trained the model at epochs=85, batch_size=512 then results of model accuracy and model loss are:

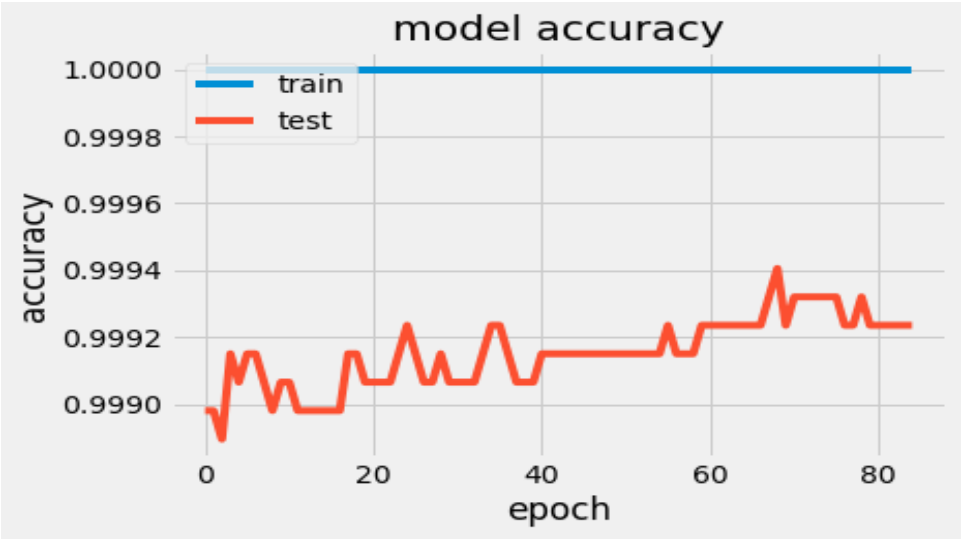


Fig 4.6: model accuracy of Alex-net

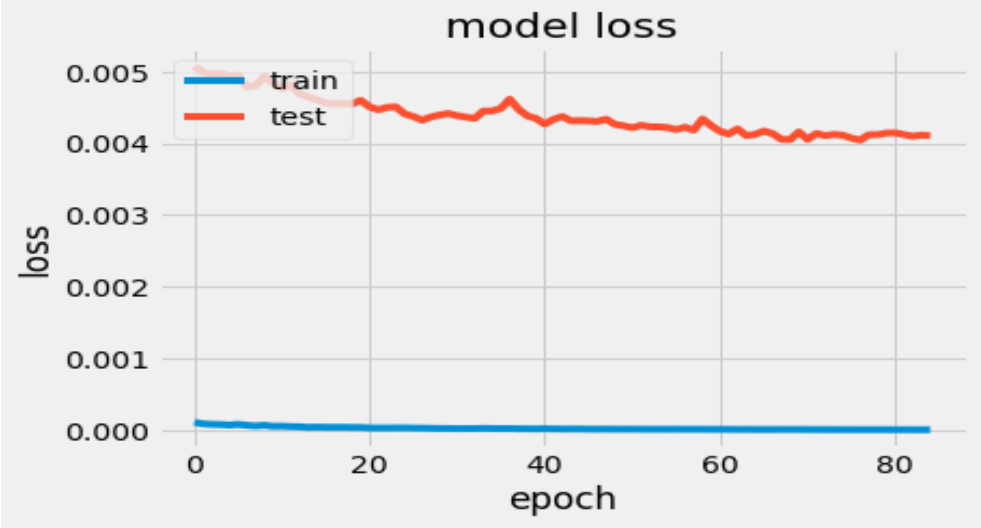


Fig 4.7: model loss of Alex-net

The test data accuracy was 96%.The confusion matrix was improved than vgg16.

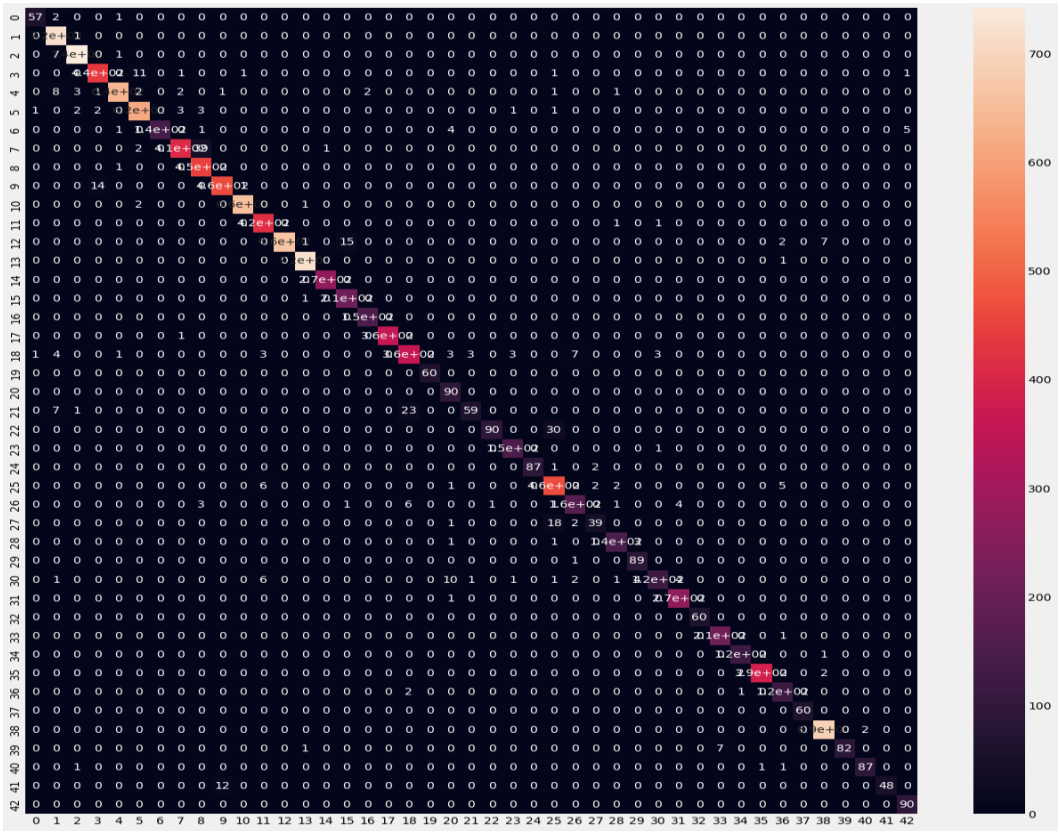


Fig 4.8: confusion matrix of Alex-net

Fig 4.9: Predictions on Test Data of Alex-net

Then we made the epochs =150 and batch-size=150. Accuracy of test data was improved to 98%. Update the model accuracy as:

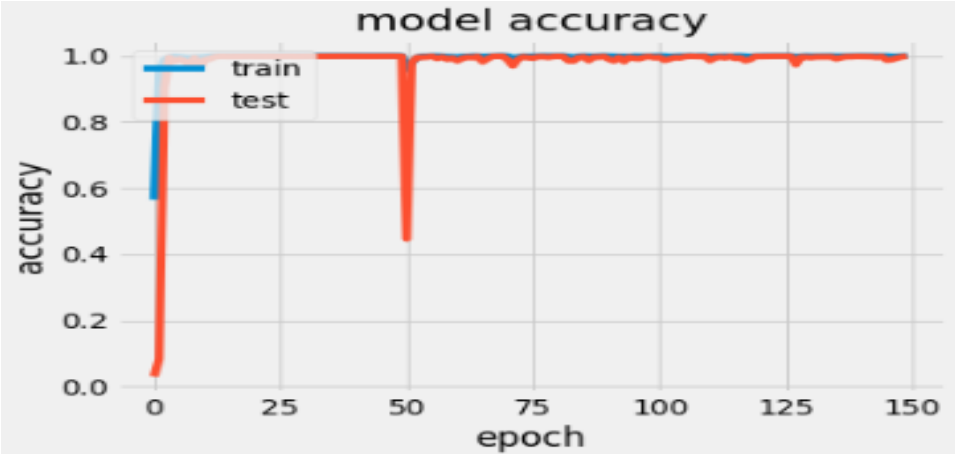


Fig 4.10: update model accuracy of Alex-net

3. Dense-Net

The structure of Dense -Net as:

```
import tensorflow as tf
from tensorflow.keras.layers import Input, Conv2D,
    BatchNormalization, Dense
from tensorflow.keras.layers import AvgPool2D,
    GlobalAveragePooling2D, MaxPool2D
from tensorflow.keras.models import Model
from tensorflow.keras.layers import ReLU, concatenate
import tensorflow.keras.backend as K
# Creating Densenet121
def densenet(input_shape, n_classes, filters = 32):

    #batch norm + relu + conv
    def bn_rl_conv(x,filters,kernel=1,strides=1):

        x = BatchNormalization()(x)
        x = ReLU()(x)
        x = Conv2D(filters, kernel, strides=strides,padding =
             'same')(x)
        return x

    def dense_block(x, repetition):

        for _ in range(repetition):
            y = bn_rl_conv(x, 4*filters)
            y = bn_rl_conv(y, filters, 3)
            x = concatenate([y,x])
        return x
```

```
def transition_layer(x):

    x = bn_rl_conv(x, K.int_shape(x)[-1] //2 )
    x = AvgPool2D(2, strides = 2, padding = 'same')(x)
    return x

input = Input (input_shape)
x = Conv2D(64, 7, strides = 2, padding = 'same')(input)
```

```
def transition_layer(x):

        x = bn_rl_conv(x, K.int_shape(x)[-1] //2 )
        x = AvgPool2D(2, strides = 2, padding = 'same')(x)
        return x

    input = Input (input_shape)
    x = Conv2D(64, 7, strides = 2, padding = 'same')(input)
    x = MaxPool2D(3, strides = 2, padding = 'same')(x)
    for repetition in [6,12,24,16]:
        d = dense_block(x, repetition)
        x = transition_layer(d)
    x = GlobalAveragePooling2D()(d)
    output = Dense(n_classes, activation = 'softmax')(x)
    model = Model(input, output)
    return model
input_shape = 32, 32, 3
n_classes = 43
model = densenet(input_shape,n_classes)
model.summary()
```

we did not make any modifications in architecture of denseNet unless made the number of classes and the input shape of images that fit our dataset.
We trained the model at epochs=50, batch_size=512 then results of model accuracy and model loss are:

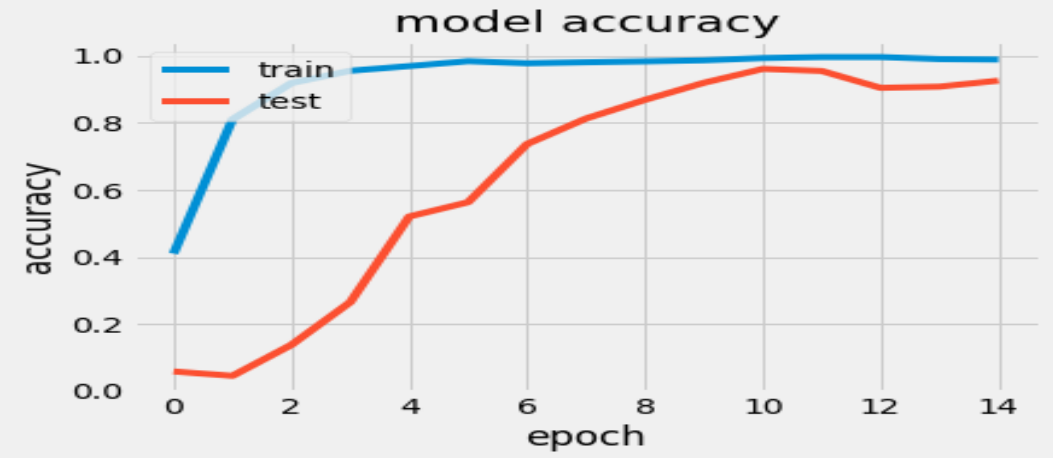


Fig 4.11: model accuracy of Dense-Net

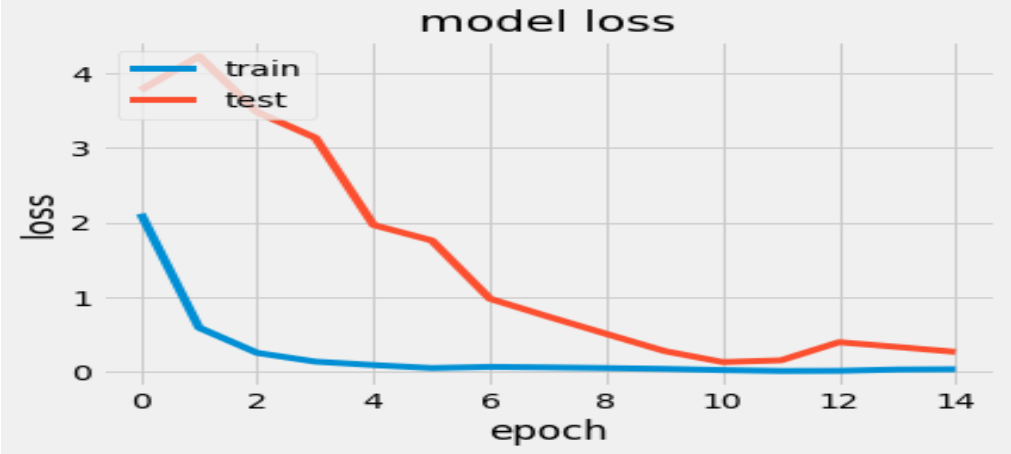


Fig 4.12: model loss of Dense-Net

The test data accuracy was 77%. The confusion matrix was as:

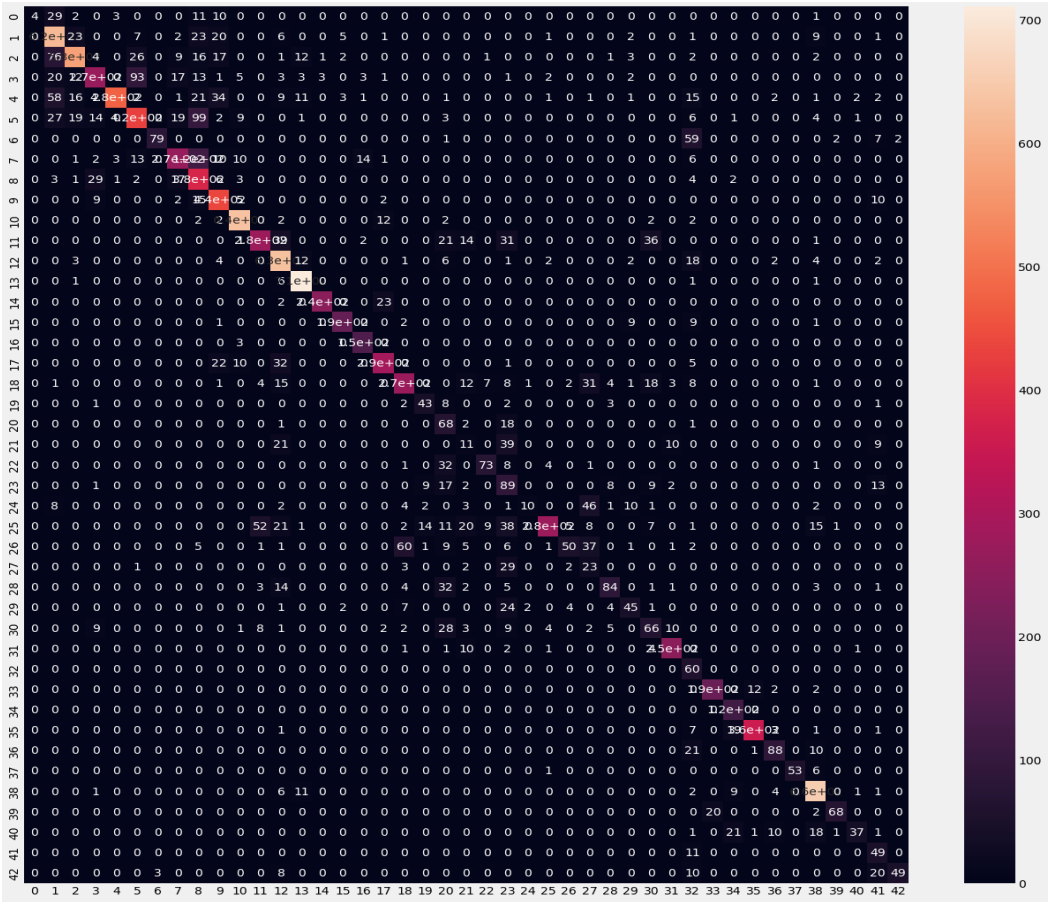


Fig 4.13: confusion matrix of Dense-net

Fig 4.14: Predictions on Test Data of Dense-net

then we made the epochs =100 and batch-size=128. Accuracy of test data improved to 95.4%.

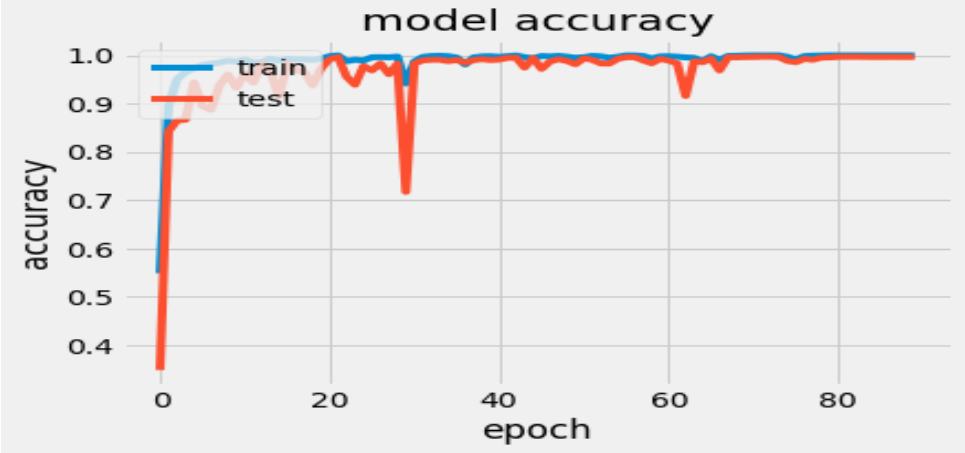


Fig 4.15: update model accuracy of Dense-net

4. XCeption-Net

The structure of XCeption -Net as:

```
#import necessary libraries
import tensorflow as tf
from tensorflow.keras.layers import Input,Dense,Conv2D,Add
from tensorflow.keras.layers import SeparableConv2D,ReLU
from tensorflow.keras.layers import
BatchNormalization,MaxPool2D
from tensorflow.keras.layers import GlobalAvgPool2D
from tensorflow.keras import Model
# creating the Conv-Batch Norm block
def conv_bn(x, filters, kernel_size, strides=1):

 x = Conv2D(filters=filters,
 kernel_size = kernel_size,
 strides=strides,
 padding = 'same',
 use_bias = False)(x)
 x = BatchNormalization()(x)
 return x
#creating separableConv-Batch Norm block
def sep_bn(x, filters, kernel_size, strides=1):

 x = SeparableConv2D(filters=filters,
 kernel_size = kernel_size,
 strides=strides,
 padding = 'same',
 use_bias = False)(x)
 x = BatchNormalization()(x)
 return x
# entry flow
def entry_flow(x):

 x = conv_bn(x, filters =32, kernel_size =3, strides=2)
```

```
x = ReLU()(x)
 x = conv_bn(x, filters =64, kernel_size =3, strides=1)
 tensor = ReLU()(x)

 x = sep_bn(tensor, filters = 128, kernel_size =3)
 x = ReLU()(x)
 x = sep_bn(x, filters = 128, kernel_size =3)
 x = MaxPool2D(pool_size=3, strides=2, padding = 'same')(x)

 tensor = conv_bn(tensor, filters=128, kernel_size = 1,strides=2)
 x = Add()([tensor,x])

 x = ReLU()(x)
 x = sep_bn(x, filters =256, kernel_size=3)
 x = ReLU()(x)
 x = sep_bn(x, filters =256, kernel_size=3)
 x = MaxPool2D(pool_size=3, strides=2, padding = 'same')(x)

 tensor = conv_bn(tensor, filters=256, kernel_size = 1,strides=2)
 x = Add()([tensor,x])

 x = ReLU()(x)
 x = sep_bn(x, filters =728, kernel_size=3)
 x = ReLU()(x)
 x = sep_bn(x, filters =728, kernel_size=3)
 x = MaxPool2D(pool_size=3, strides=2, padding = 'same')(x)

 tensor = conv_bn(tensor, filters=728, kernel_size = 1,strides=2)
 x = Add()([tensor,x])
 return x
# middle flow
def middle_flow(tensor):

 for _ in range(8):
 x = ReLU()(tensor)
 x = sep_bn(x, filters = 728, kernel_size = 3)
 x = ReLU()(x)
 x = sep_bn(x, filters = 728, kernel_size = 3)
 x = ReLU()(x)
```

```
x = sep_bn(x, filters = 728, kernel_size = 3)
        x = ReLU()(x)
        tensor = Add()([tensor,x])

    return tensor
# # exit flow

def exit_flow(tensor):

    x = ReLU()(tensor)
    x = sep_bn(x, filters = 728,  kernel_size=3)
    x = ReLU()(x)
    x = sep_bn(x, filters = 1024,  kernel_size=3)
    x = MaxPool2D(pool_size = 3, strides = 2, padding ='same')(x)

    tensor = conv_bn(tensor, filters =1024, kernel_size=1, strides =2)
    x = Add()([tensor,x])

    x = sep_bn(x, filters = 1536,  kernel_size=3)
    x = ReLU()(x)
    x = sep_bn(x, filters = 2048,  kernel_size=3)
    x = GlobalAvgPool2D()(x)

    x = Dense (units = 43, activation = 'softmax')(x)

    return x
# model code

input = Input(shape = (32,32,3))
x = entry_flow(input)
x = middle_flow(x)
output = exit_flow(x)

model = Model (inputs=input, outputs=output)
```

We trained the model at epochs=90, batch-size=1024 then results of model accuracy and model loss are:

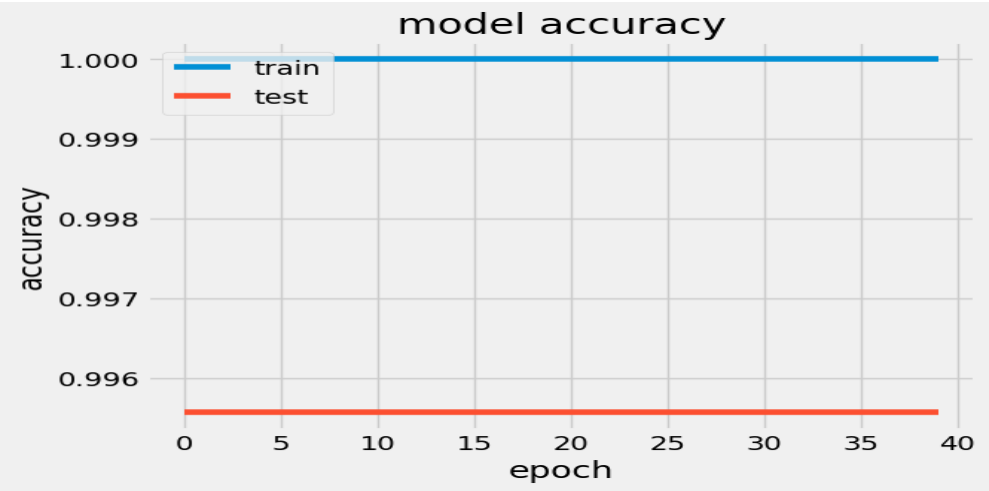


Fig 4.16: model accuracy of XCeption -net

Fig 4.17: model loss of XCeption -net

The test data accuracy was 93.6%.

Then we adjusted the structure of the Xeption-model, deleted middle_flow function and we made the epochs =150 and batch-size=1024, accuracy of test data improved to 97.89%.

The confusion matrix was as:

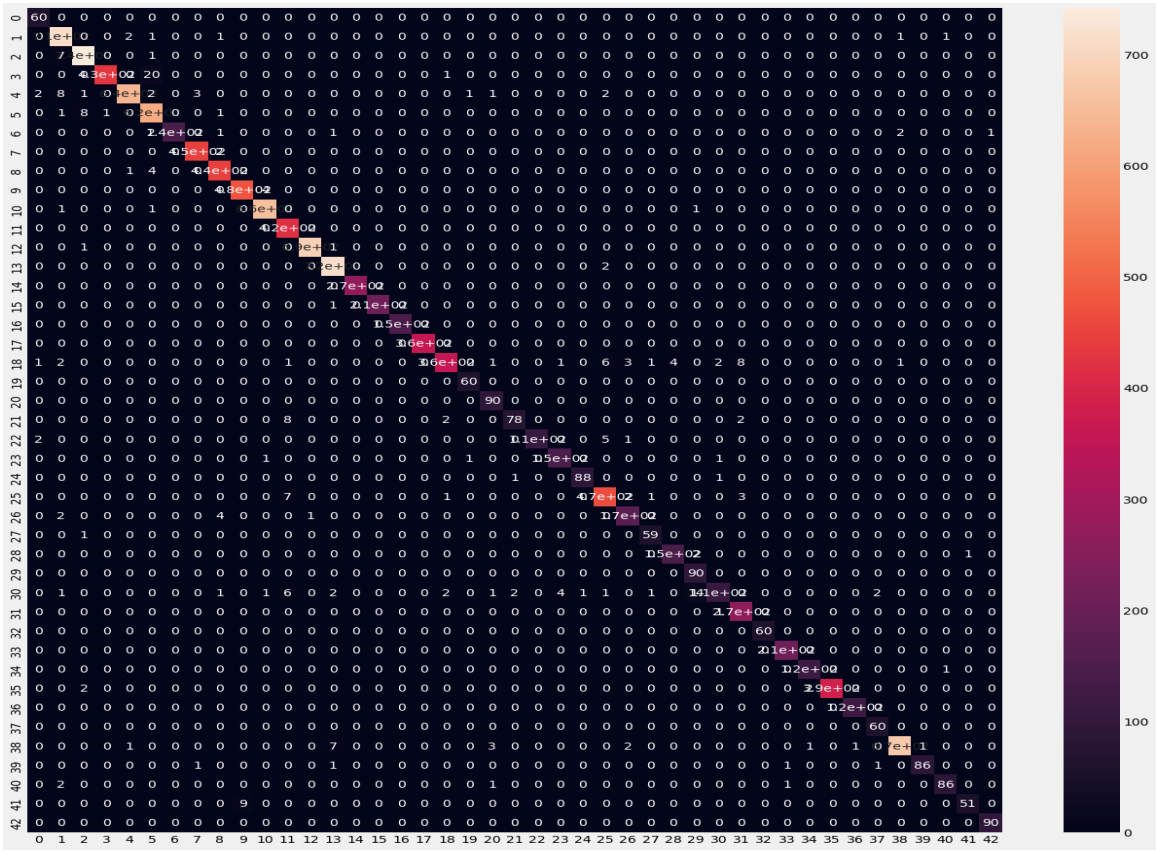


Fig 4.18: confusion matrix of Xception-net

Fig 4.19: Predictions on Test Data of Xception-net

5. Custom_model

We modify the architecture of vgg16 network ,get the very good result from that so we have model customization architecture as:

```
custom=keras.models.Sequential([
        keras.layers.Conv2D(filters=256, kernel_size=(3,3), strides=(1,1),
activation='relu', input_shape=(32,32,3)),
        keras.layers.Conv2D(filters=256, kernel_size=(3,3), strides=(1,1),
activation='relu', padding="same"),
        keras.layers.MaxPool2D(pool_size=(2,2)),
        keras.layers.BatchNormalization(),
        keras.layers.Conv2D(filters=256, kernel_size=(3,3), strides=(1,1),
activation='relu', input_shape=(32,32,3)),
        keras.layers.Conv2D(filters=256, kernel_size=(3,3), strides=(1,1),
activation='relu', padding="same"),
        keras.layers.MaxPool2D(pool_size=(2,2)),
        keras.layers.BatchNormalization(),
        keras.layers.Flatten(),
        keras.layers.Dense(512, activation='relu'),
        keras.layers.BatchNormalization(),
        keras.layers.Dropout(rate=0.5),
        keras.layers.Dense(43, activation='softmax')
])
```

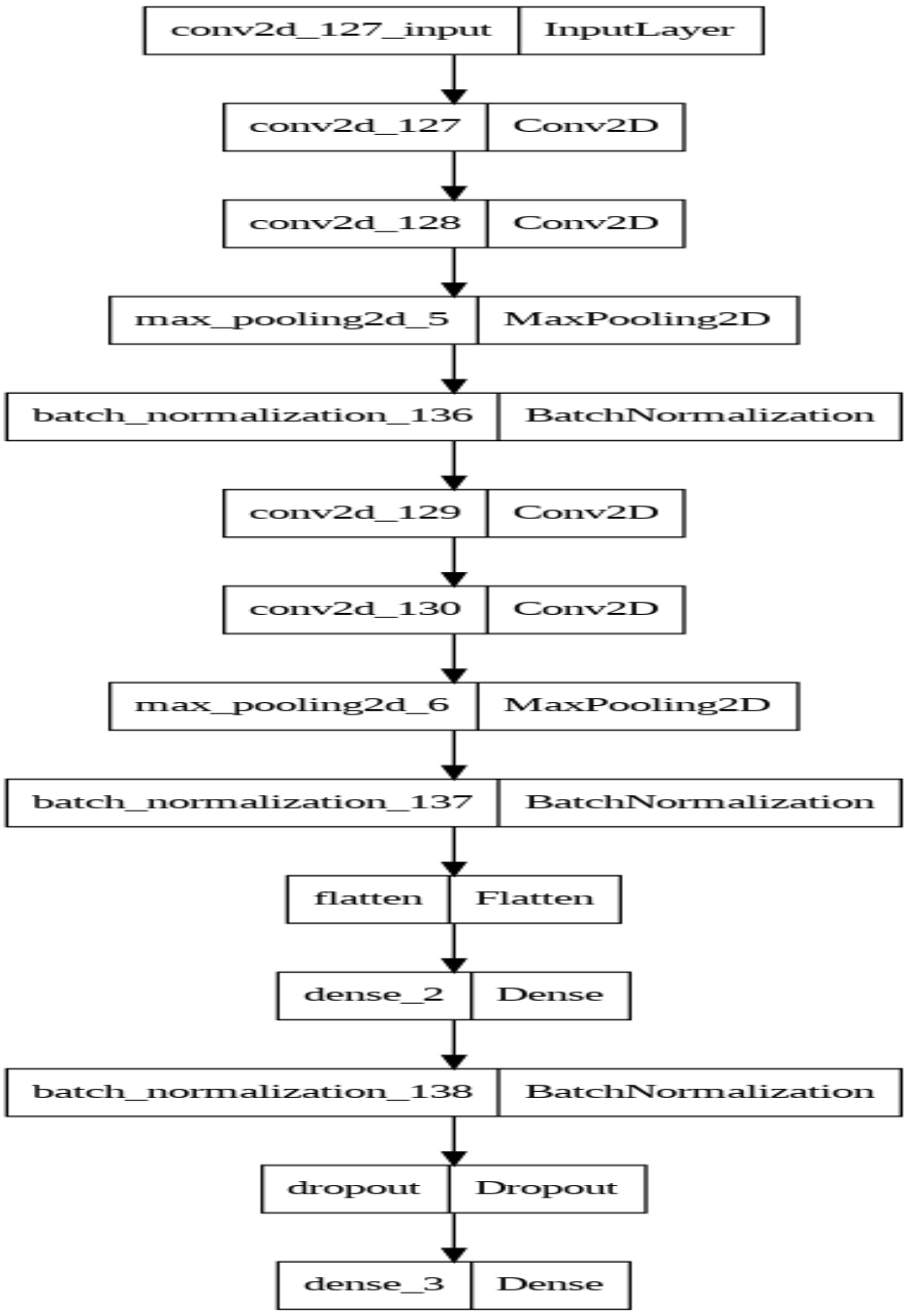


Fig 4.20: Custom model architecture (first visualization)

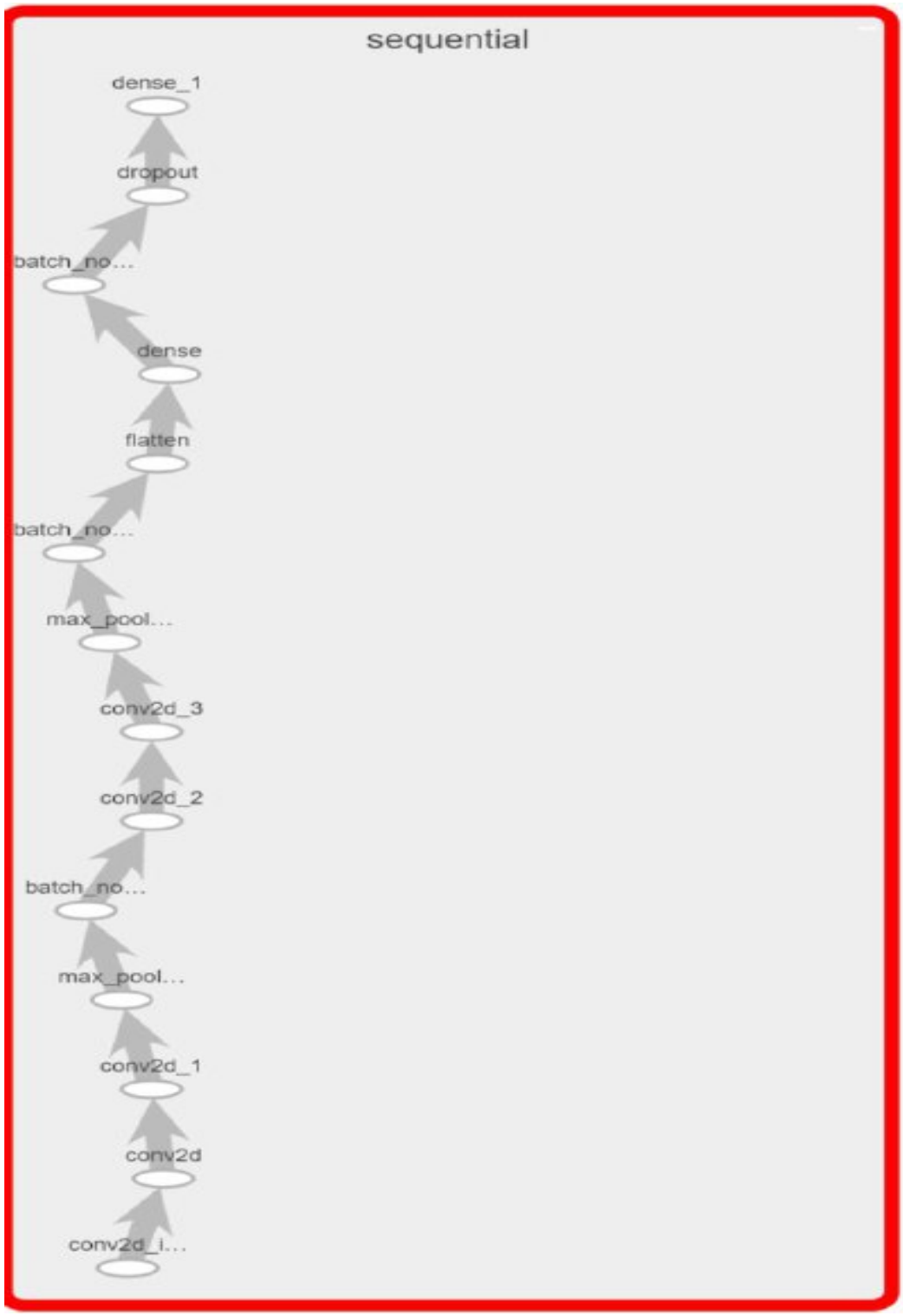


Fig 4.21: Custom model architecture (second visualization)

We trained the model at epochs=100, batch_size=1024 then results of model accuracy and model loss are:

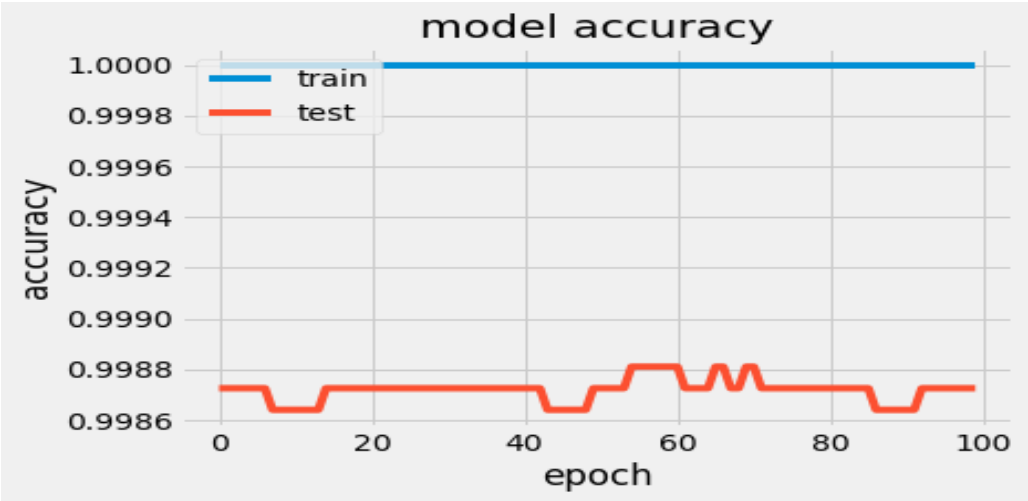


Fig 4.22: model accuracy of Custom-model

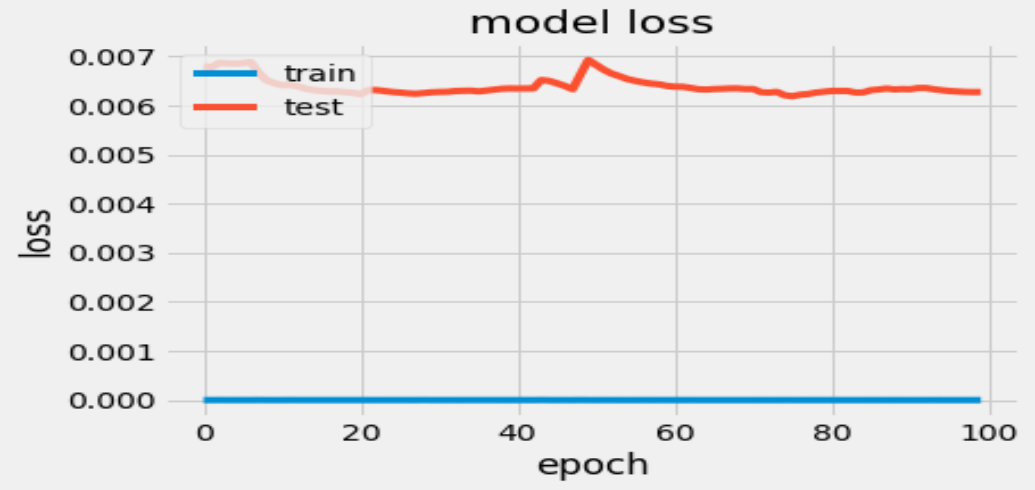


Fig 4.23: model loss of Custom-model

The test data accuracy was 98.65%. The results of the models are **summarized in this table**:

Model	Batch	Epoch	Accuracy	Input shape
Vgg16	1024	100	68	64*64
Vgg16	256	90	53	32*32
DenseNet	128	100	95.47	32*32
DenseNet	512	50	77	32*32
Alex	128	95	96	32*32
Alex	150	150	98	32*32
XCeption	1024	90	93.6	32*32
XCeption	1024	150	97.89	32*32
Custom_model	1024	100	98.65	32*32

Table 4.1: results of models

Chapter 5: Models ensembles

Overview:

An approach to machine learning called an ensemble model combines various other models in the prediction process. Base estimators are the name given to the models. It is a way to deal with low accuracy, which occurs when one model or algorithm may not be able to adequately match all the training data.

For certain data sets, a single method might not produce the ideal forecast. Machine learning algorithms have their limitations, and it might be difficult to create a model with high accuracy. The accuracy could be improved overall if we create and merge different models[9].

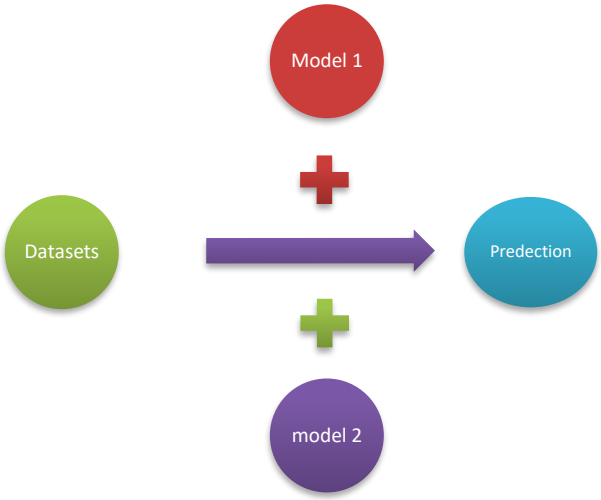


Fig 5.1: model predictions using multiple algorithms.

Soft and Hard Voting in Ensemble Methods

To get a result that is more accurate than any of the individual algorithms, ensemble methods aggregate the output of two or more various machine learning algorithms.

A. Soft voting

In soft voting, a result is generated by averaging the probability for each class. In this example, the ensemble would predict that an object was a 1 with (70 + 80 + 40) / 3= 63% probability if algorithm one indicated it was

a 1 with 70% probability , algorithm two projected it was a 1 with 80% probability and algorithm three projected it was a 1 with 40% probability.

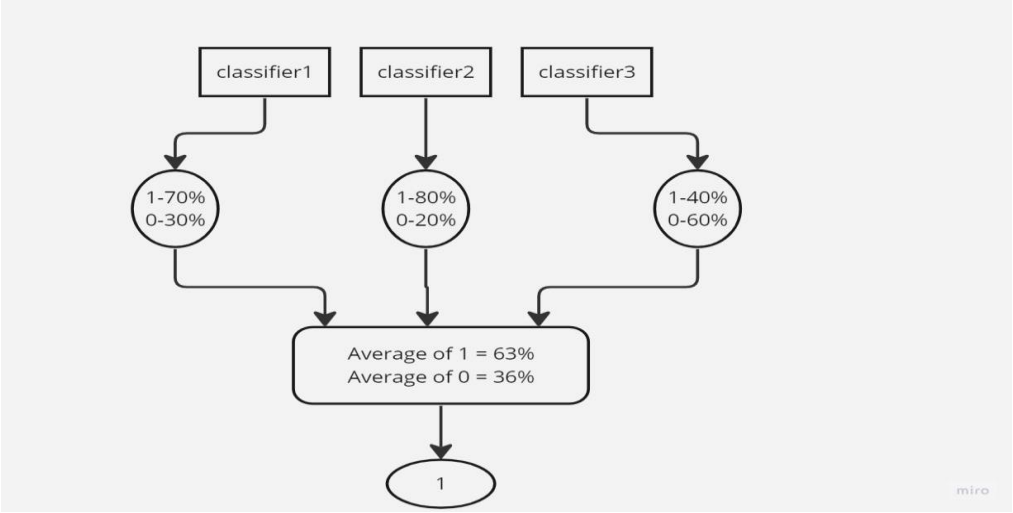


Fig 5.2: Soft Voting

B. Hard voting

In a hard vote, the ensemble chooses the class with the most votes after considering the predictions of each method. For instance, the ensemble will predict "red" if three algorithms predict "white," "red," and "red" for the color of a specific car, respectively[10].

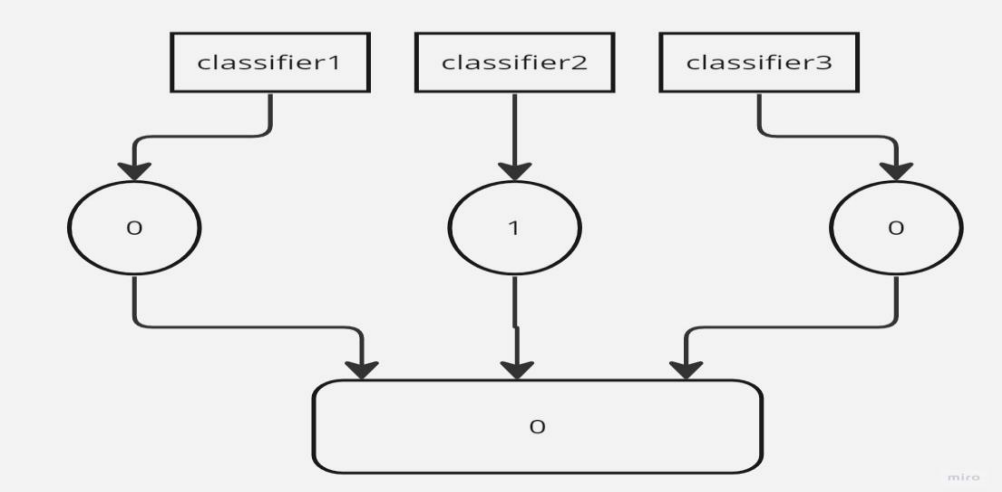


Fig 5.3: Hard Voting

Results

We did the ensemble method between the models using two trials to obtain the highest accuracy in the test data of germane traffic sign recognition benchmark (GTSRB) dataset.

A. The first trial

the first trial was between the three pre-trained models:

- Vgg16 with accuracy (68%)
- DenseNet with accuracy (95.4%)
- AlexNet with accuracy (96%)

We applied the majority (hard) voting on them then the accuracy of test data improved but was not good where the the ensemble (Hard Voting) gave us 96.1%.

And we applied the soft voting on them where it sees the probability of each classifier, then the accuracy of test data improved more than hard Voting where the ensemble (Soft Voting) gave us 96.8%.
The table below show these results and summarize it:

Model	**Accuracy**
VGG16	68.0%
DenseNet	95.4%
AlexNet	96.0%
Ensemble **(Soft Voting)**	96.8%
Ensemble **(Hard Voting)**	96.1%

Table 5.1: Results of ensemble between pre-trained networks(1st trial)

As shown in the above table, the highest accuracy of the ensemble is 96.8%; however, when we customized the VGG16 network, the ensemble accuracy reached 99.0%.

B. the Second trial

The second trial was between the four classifiers:

- Custom Model with accuracy (98.65%)
- DenseNet with accuracy (95.4%)
- AlexNet with accuracy (98%)
- XCeptionNet with accuracy (97.89%)

We applied the majority (hard) voting on four Networks then the ensemble model prediction accuracy improved and gave us 98.96%.

And we applied the soft voting on four models where it see the probability of each model, then the accuracy of test data improved more than hard Voting where the ensemble (Soft Voting) gave us 99.17%.

Model	**Accuracy**
Custom Model	98.65%
DenseNet	95.4%
AlexNet	98.0%
XCeptionNet	97.89%
Ensemble **(Soft Voting)**	99.17%
Ensemble **(Hard Voting)**	98.96%

Table 5.2: Results of ensemble between pre-trained networks (2nd trial)

Results of test ensemble (hard voting) on four images were Compatible as:

Fig 5.4: Hard Voting Results Sample

Chapter 6: Road signs Detection

Introduction

Genuine images from the target domain, pricey annotation, and balanced data sets are all requirements for detectors. The annotation process is expensive since each traffic sign needs to be annotated with a bound box, which is more difficult than just creating a class for a classification problem. Furthermore, deep learning-based detectors still which known for being data-hungry, requiring a large number of real-world photos to perform properly. Because it takes locating several traffic sign samples along the roadways, the collecting of such real-world photographs of traffic signs might be challenging. The traffic signs are not standardized globally since traffic laws vary from one nation to the next, necessitating the creation of a new data set for each one. Finally, a balanced number must be produced for each class through the picture capture. Some traffic signs are more prevalent than others under typical driving conditions, therefore gathering a minimal balance across the classes would necessitate gathering a lot more photos.

Methodology

A camera installed behind the inside rearview mirror is used to detect signs. It locates signs on either the left, right, or over the road and detect, then comparing them to an internal dataset. Once the sign is recognized, a display on the GPS or instruments alerts the driver to the issue.

To allow autonomous cars on our roads, robustness and reliability traffic sign detection is required. As a result, researchers have given a lot of attention to developing algorithms for detecting and recognizing traffic signs. At first, completely manual techniques predominated the literature on traffic signs. These approaches' primary mechanisms may be broken down into two distinct parts. Getting descriptive data, which might be displayed as a feature, a descriptor, or in any other form, is the main goal of the first block. The second block assesses how similar or unlike extracted representations are to specified representations, such as templates to determine the class of a traffic sign. In more recent datasets, a wide range of machine learning techniques predominated the state-of-the-art approaches, despite the fact that

these manual techniques could work well in tiny datasets like RUG for detection or recognition.

AdaBoost, neural networks, support vector machines, linear discriminant analysis, subspace analysis, ensemble classifiers, slow feature analysis, kd-trees, and random forests are some examples of machine learning algorithms that result in state-of-the-art performances in these researches. The studies listed above mostly use machine learning approaches to determine how handmade features/descriptors correspond to different kinds of traffic signs. Contrarily, convolutional neural networks (CNNs) enabled the learning of visual representations as well as their correspondences to sign classes directly from data, producing state-of-the-art recognition and detection outcomes in recent research. CNNs can be mixed with handmade features even though they are frequently used to remove designing characteristics.

Challenges in the detection and recognition

The challenges in recognizing and detecting traffic signs are:

- Variations in illumination with the time of day and because of the weather.
- Over time, the paint on the signage deteriorates.
- Characteristics of the image capture equipment.
- Other similar forms in the image.
- Geometric distortions brought caused by wind or storms.
- Occlusion by the trees, street lighting, buildings, people, etc.
- Detection may be hampered by vibration and motion blur.

Methods of road signs detection

The three categories of traffic sign detecting techniques are those that methods based on color, shape or machine learning. Most traffic signs are mostly red, blue, or yellow in color. This ability is widely used by authors. Utilized is related component segmentation with a color model. After that, a recognition algorithm or an appearance model verifies the regions of interest. These techniques are frequently quick and translation, rotation, and scaling invariant. The fundamental challenge of color-based approaches is how to be invariant to various lighting situations because color may be readily altered

by lighting circumstances. The image is converted into a color space and then thresholded in most of these techniques.

For shape-based approaches, a structural or global approach is used to analyze the image's contours. Because they can handle grayscale pictures and take into account the image's gradient, these techniques are typically more reliable than color-based ones. These techniques are particularly sensitive to occlusions and distortion, which greatly reduces their effectiveness. To solve this issue, some academics suggested utilizing the circle recognition method EDCircles to recognize circular road signs, and other researchers suggested using HOG and Linear SVM to detect circular and triangular signs. Systems that use shape-based techniques to reduce color fading brought on by lighting and environmental changes have trouble identifying occluded and damaged signs, which call for color-based techniques.

Finally, a classifier (cascade, SVM, or neural networks) is trained using examples in machine learning approaches. It is used on a sliding window to go across the image at different scales. These techniques, which integrate geometry and photometry, can be time-consuming to compute. They demand the creation of a learning base per sign type, a laborious process when there are many items to be detected. In order to reduce the incidence of false positives and enhance the rate of true positives, many researchers use a combination of color-based and shape-based approaches.

Results

Version_1:

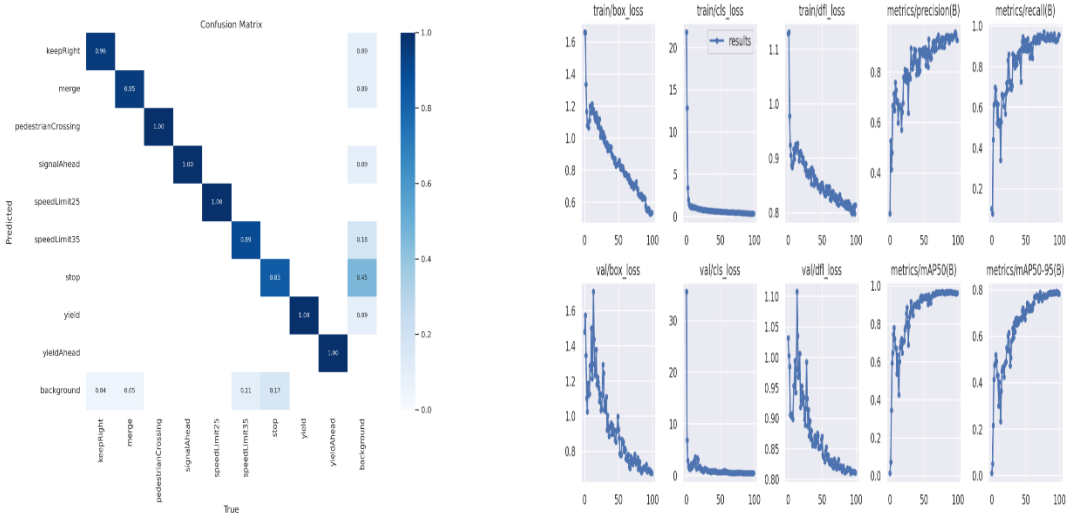
Train	Val	Test
70%	20%	10%

Table 6.1: Dataset splitting Version_1

	Epoch	Batch	MAP50		
			Train	Val	Test
Yolov8	100	64	96.3	96.3	94.8
Yolov5			95.7	93.7	96.9
Yolov8	100	16	96.3	96.3	95
Yolov5			93.6	92.6	96.1
Yolov8	90	16	97.1	97	94.6
Yolov5			94.4	92.4	96.3

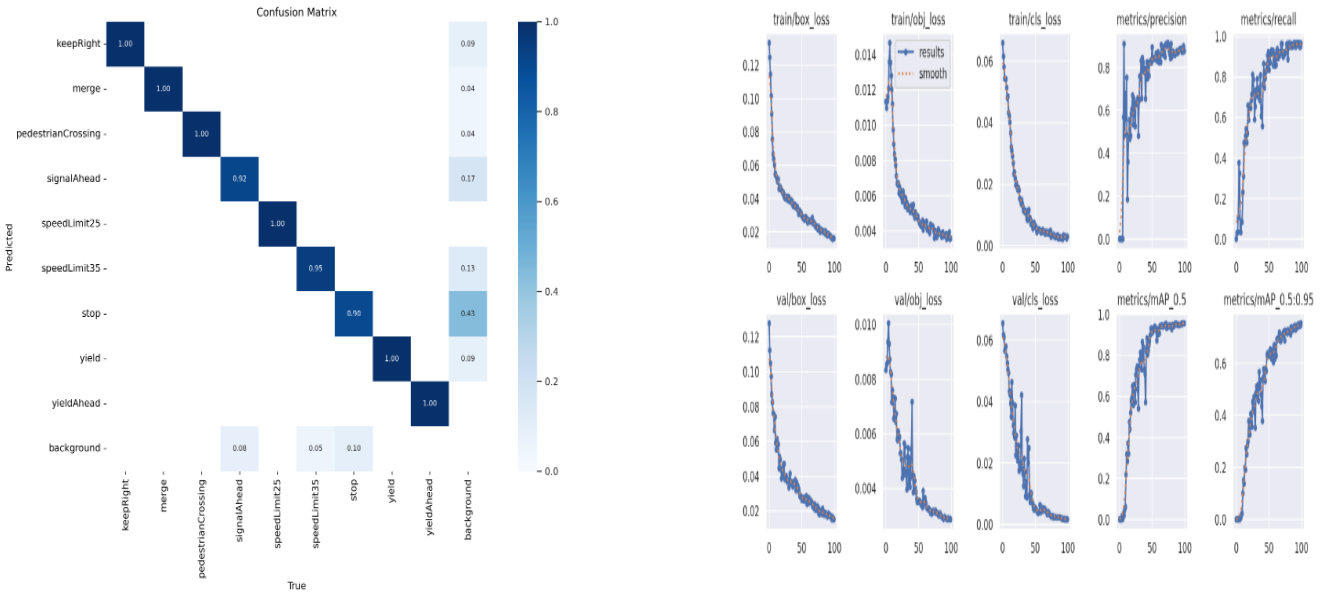
Table 6.2: Results of Version_1

The best MAP50 in version_1 was 96.9 with Yolov5, Epoch = 100 and Batch = 64.



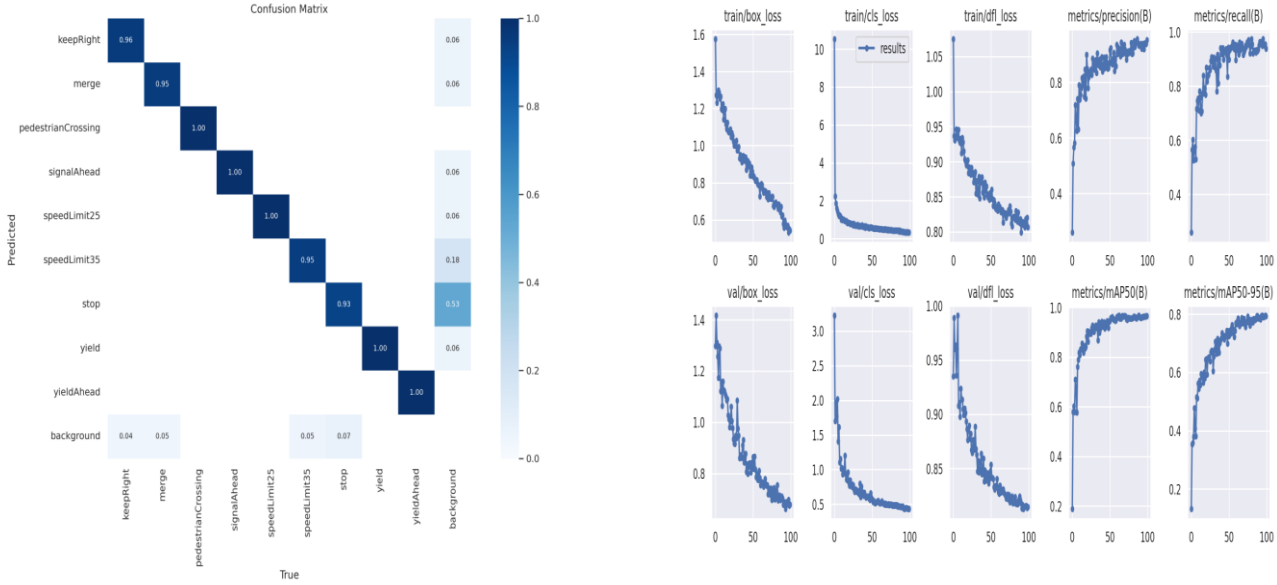
a. Confusion matrix b. curves

Fig 6.1: Results of version_1 Yolov8 with Epoch=100 && Batch=64



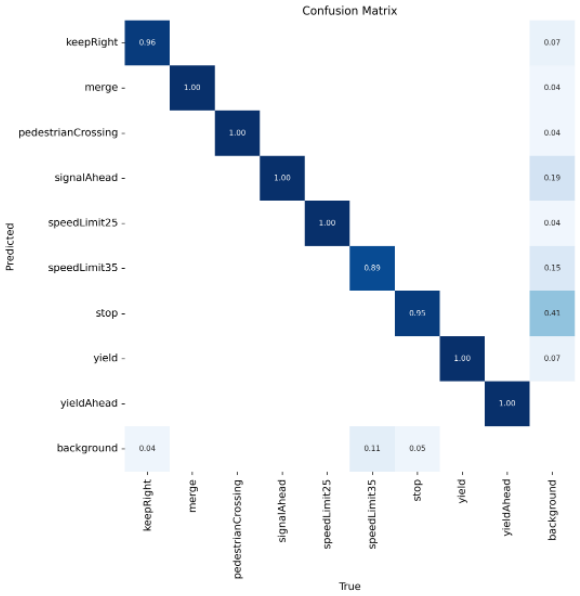
a. Confusion matrix b. Curves

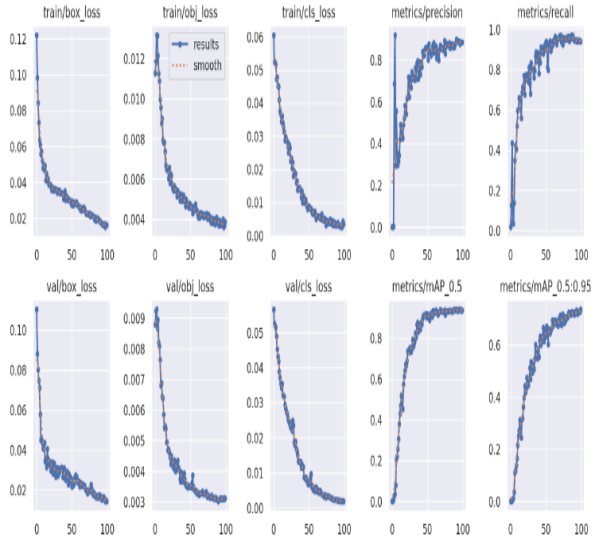
Fig 6.2: Results of version_1 Yolov5 with Epoch 100 && Batch =16



a. Confusion matrix b. Curves

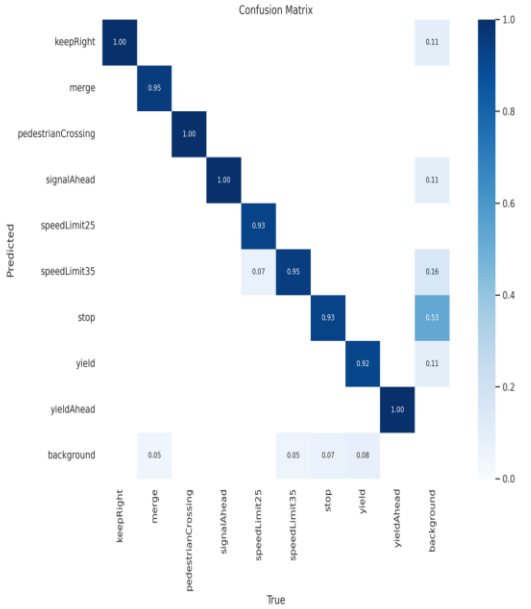
Fig 6.3: Results of version_1 Yolov8 with Epoch=100 && Batch=16

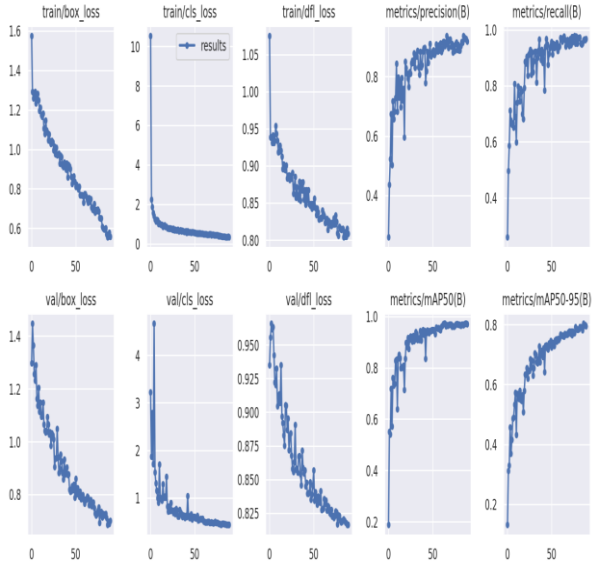




b. Confusion matrix b. Curves

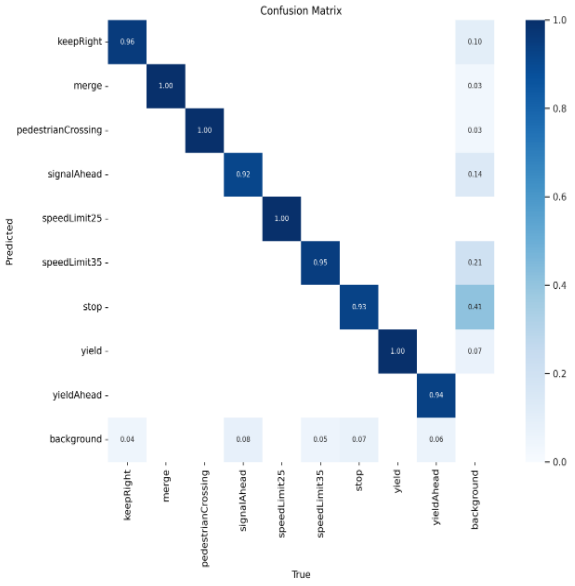
Fig 6.4: Results of version_1 Yolov5 with Epoch=100 && Batch=16

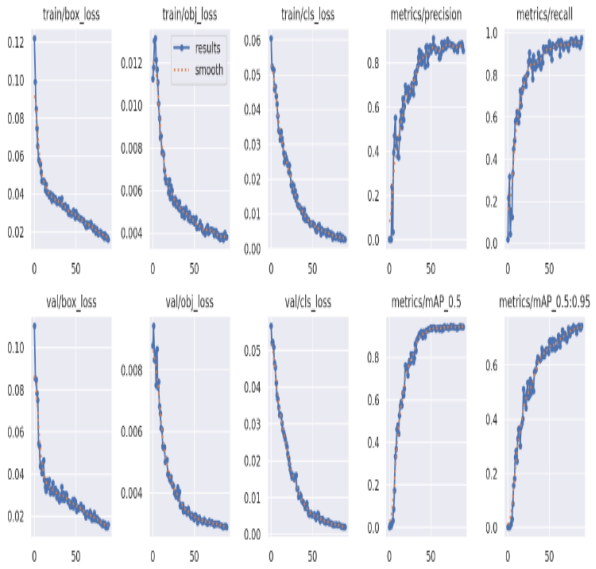




a. Confusion matrix b. Curves

Fig 6.5: Results of version_1 Yolov8 with Epoch=90 && Batch=16





b. Confusion matrix b. Curves

Fig 6.6: Results of version_1 Yolov5 with Epoch=90 && Batch=16

Version_2:

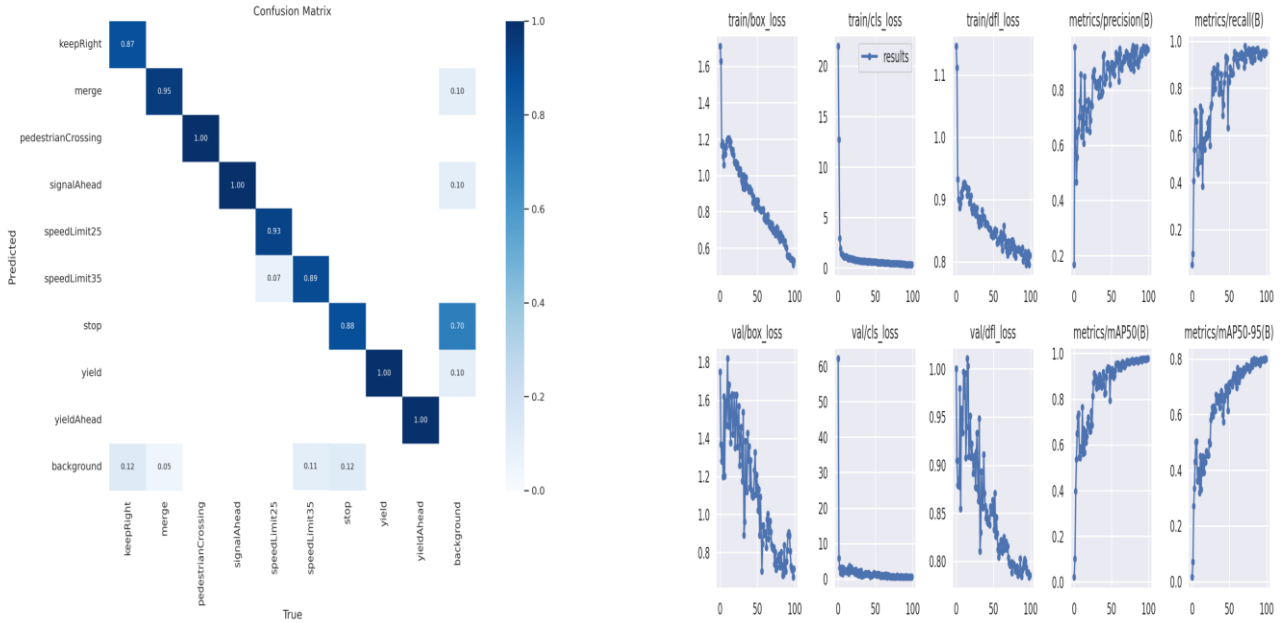
Train	Val	Test
70%	15%	15%

Table 6.3: Dataset splitting Version_2

	Epoch	Batch	MAP50		
			Train	Val	Test
Yolov8	100	64	97.7	97.7	93.6
Yolov5			95.2	93	92.8
Yolov8	100	16	96.8	96.8	94.6
Yolov5			94.1	92.4	93.6
Yolov8	90	16	96.2	96.2	94.1
Yolov5			92.9	89.8	92.9

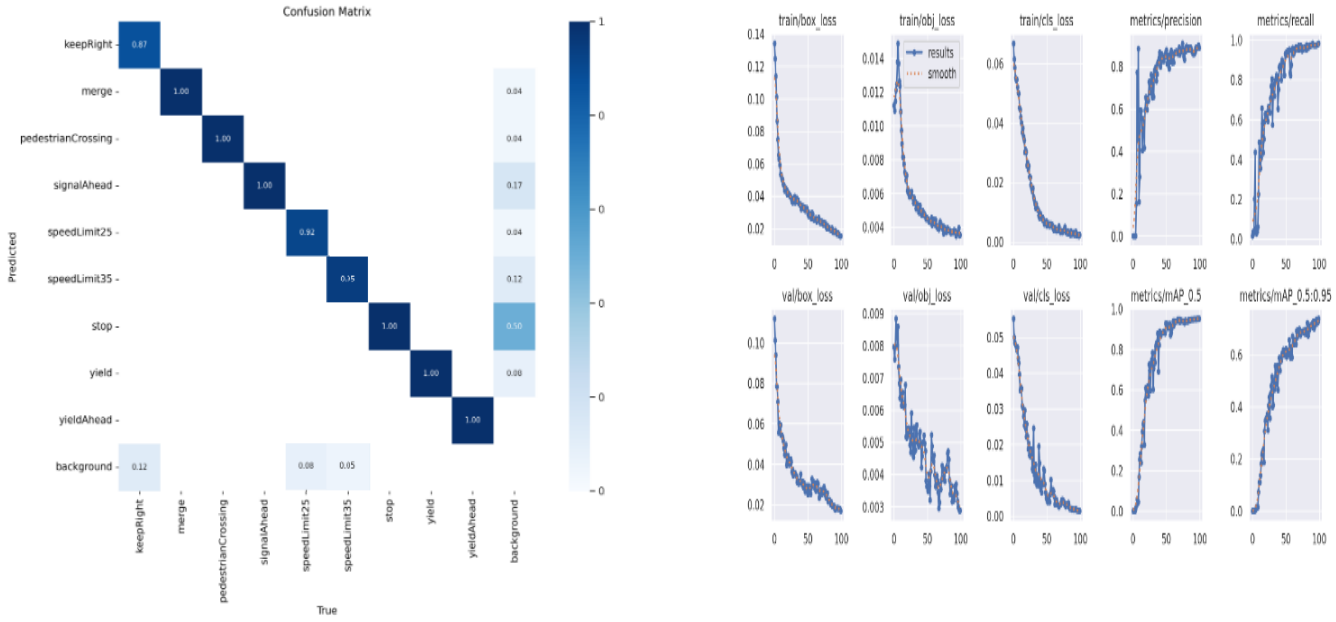
Table 6.4: Results of Version_2

The best MAP50 in version_2 was 94.6 with Yolov8, Epoch = 100 and Batch = 16.



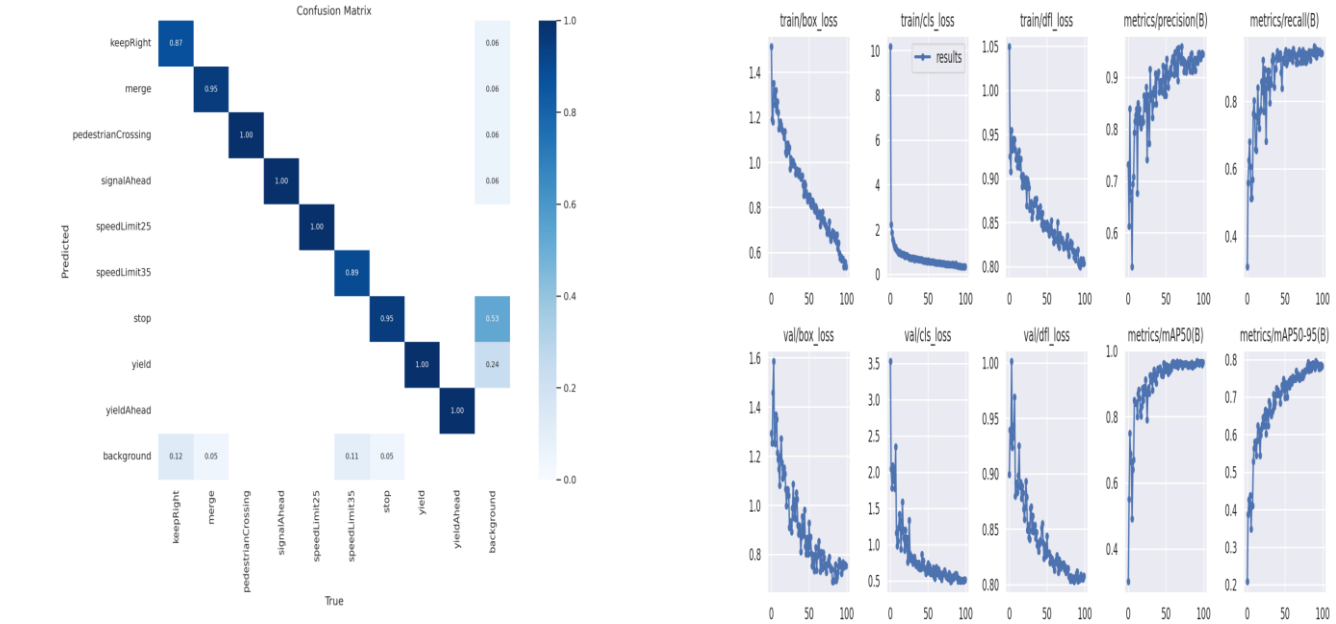
a. Confusion matrix b. Curves

Fig 6.7: Results of version_2 Yolov8 with Epoch=100 && Batch=64



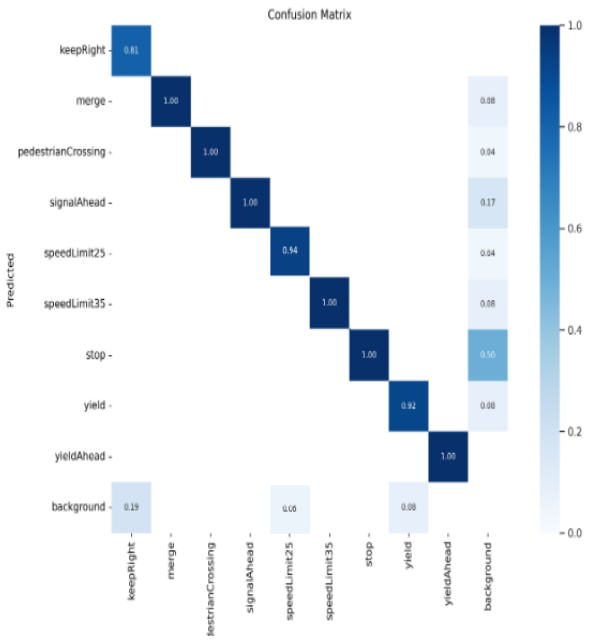
a. Confusion matrix b. Curves

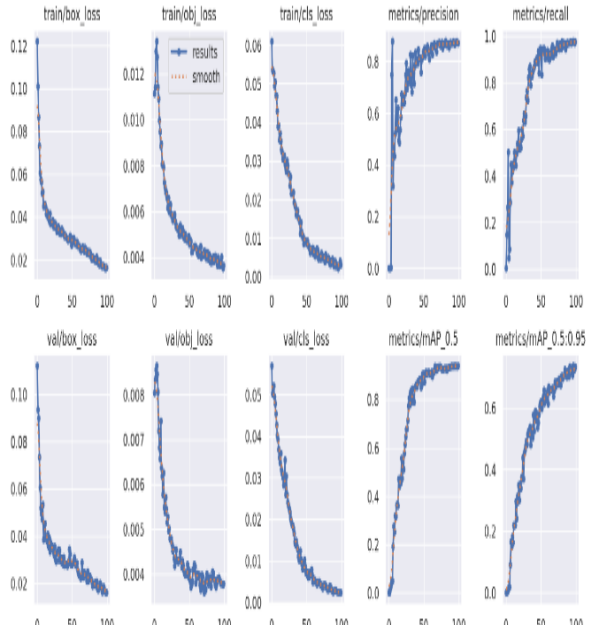
Fig 6.8: Results of version_2 Yolov5 with Epoch=100 && Batch=64



a. Confusion matrix b. Curves

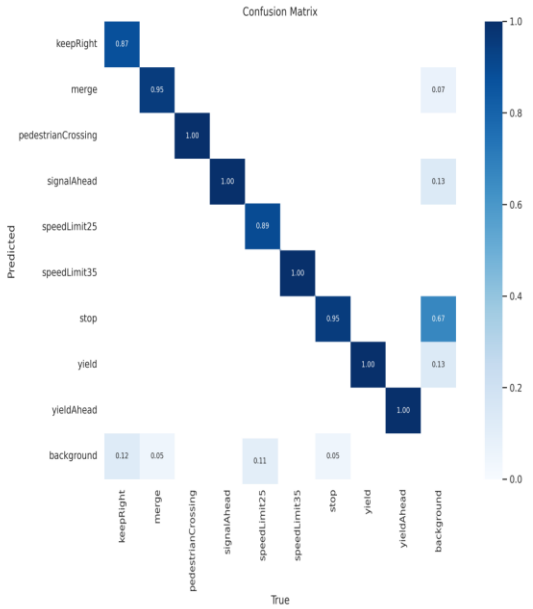
Fig 6.9: Results of version_2 Yolov8 with Epoch=100 && Batch=16

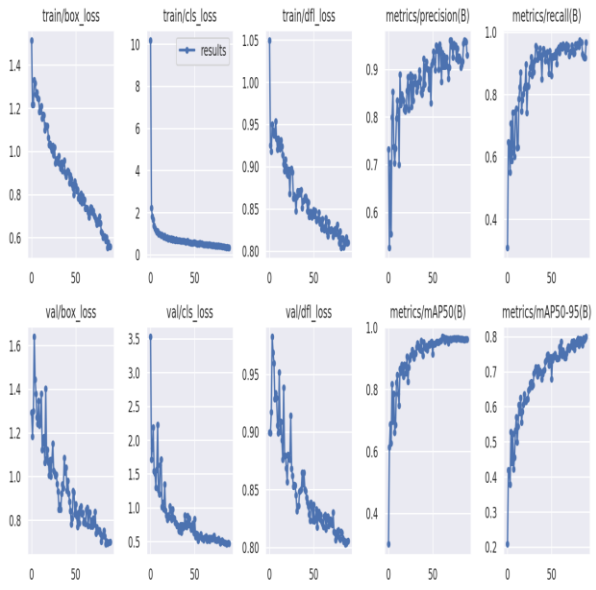




a. Confusion matrix b. Curves

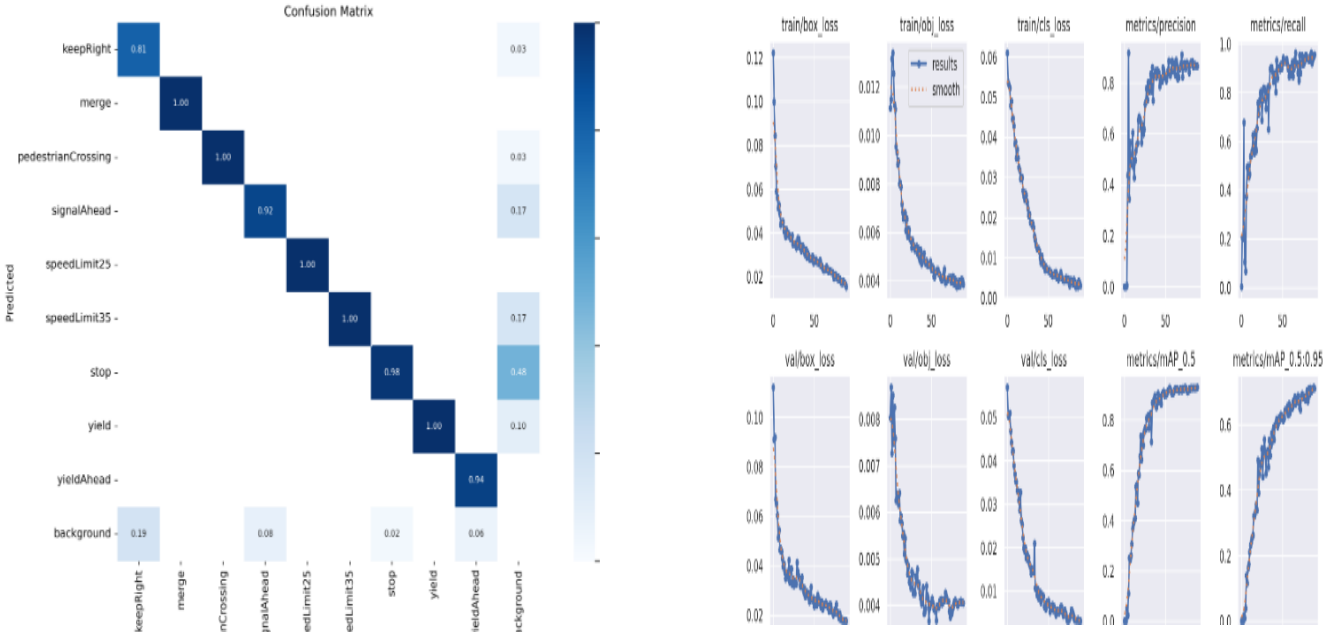
Fig 6.10: Results of version_2 Yolov5 with Epoch=100 && Batch=16





b. Confusion matrix b. Curves

Fig 6.11: Results of version_2 Yolov8 with Epoch=90 && Batch=16



a. Confusion matrix b. Curves

Fig 6.12: Results of version_2 Yolov5 with Epoch=90 && Batch=16

Part 2

Pavement Defect Detection

Chapter 7: introduction and related work

Introduction

Pavement defects are a major problem that can lead to road accidents, traffic congestion, and increased maintenance costs. Traditional methods for detecting pavement defects, such as visual inspection and manual measurements, are time-consuming and labor-intensive. To mitigate these problems, we propose a Deep learning-based method that can automate the roads inspection process in cities and allow governments to easily maintain city roads. These methods rely on object detection models to identify and localize the road defects which can then be mapped with their corresponding coordinates into digital maps. Therefore, helping authorities to find and repair roads before they cause accidents.

This project investigates the use of deep learning for pavement defect detection. The project will develop a deep learning model that can automatically detect and classify pavement defects in images. The model will be trained on a dataset of images that have been labeled with the type of pavement defect present. The project will assess the deep learning model's performance on a separate dataset of images. The findings will be compared to different methods.

The project's findings will have implications for the development of automated pavement defect detection systems.

Related work

The enhancement of digital maps and pavement defect detection is a non-trivial task due to the complexity of road pavements and the variety of defects that can occur. There has been a growing interest in this topic in recent years of research on this topic, using a variety of methods, including:

- **Image processing techniques:** These techniques use algorithms to extract features from images, such as edges, textures, and colors. These features can then be used to classify defects or to identify areas of interest for further inspection.

- **Machine learning techniques:** These techniques use algorithms to learn from labeled data. This data can be used to train models that can then be used to classify defects or to identify areas of interest for further inspection.
- **Deep learning techniques:** These techniques use artificial neural networks to learn from data. Deep learning models have been shown to be very effective for tasks such as image classification and object detection.

A comprehensive literature review was performed as part of this project.
a model based on (Mask R-CNN) architecture for detecting and segmenting defects of the road area in real-time mode. (Classification, segmentation, and object detection) [3]
However, the R-CNN approach can be time-consuming as it necessitates the use of multiple crops, resulting in substantial duplicate computation due to overlapping crops. This issue of redundant calculations was addressed by the introduction of Fast R-CNN [4], which processes the entire image once through a feature extractor, allowing the crops to share the computational burden of feature extraction. As mentioned earlier, the field of image processing has progressed rapidly over time. In our study, we primarily concentrate on four recent object detection systems: Faster R-CNN ([5]), You Only Look Once (YOLO) [6], Region-based Fully Convolutional Networks (R-FCN) system [7].

this paper proposes a new method for automated monitoring of road pavement aging conditions using recurrent neural networks (RNNs). RNNs are a type of deep learning model that are well-suited for tasks that involve sequential data, such as time series data. In this case, the sequential data is the spectral information of road pavement images. [8]

The proposed approach in [9] uses accelerometers to measure the vibration response of a vehicle as it travels over a road. The vibration response data is then used to identify areas of pavement distress. The proposed method was evaluated on a dataset of road sections in the Phoenix, Arizona, region. The results showed that the proposed method can accurately identify areas of pavement distress, such as cracks, potholes, and ruts.

For [10] identifying transverse cracks in asphalt pavement using vehicle vibration signal analysis. The proposed method uses a combination of fast Fourier transform (FFT) and discrete wavelet transform (DWT) to extract

features from the vehicle vibration signal. These features are then used to train a classifier to identify transverse cracks.

used a high-intensity near-infrared light-emitting diode mounted under the front bumper of a vehicle to measure whether the current road surface is dry, slippery, or icy to give the driver a hint about the current road conditions.
A highly intense near-infrared light-emitting diode (LED) positioned beneath the front bumper of a vehicle was employed to assess the current condition of the road surface, determining whether it is dry, slippery, or icy. This system provides drivers with valuable information about the prevailing road conditions. [11]

To analyze the existing state of the road, ground-penetrating radar (GPR) has frequently been utilized to detect potholes and cracks, offering a continuous profile of the road conditions. However, the cost of GPR is prohibitive, and it necessitates skilled operators. Additionally, the effectiveness of crack detection with GPR depends on the various layers of cracks. [12]

In an effort to detect potholes, a nonlinear support vector machine model with a Gaussian radial basis function was employed. However, this approach has limitations regarding the ambient light source and weather conditions, as well as the need for closer proximity for accurate detection. [13]
An attempt was made to identify road potholes with smooth, broken, and cracked pavements, utilizing a DL technique. However, the recognition rate is relatively low and is influenced by the ambient light source. Notably, recent advancements in DL techniques have yielded outstanding results in object detection for standard benchmark datasets and computer vision competitions [14] [15]. proposed a pothole classification model that employed edge detection and a YOLO-based feature extraction scheme for pothole detection. Nevertheless, these studies primarily focused on ideal conditions such as daytime on sunny days, posing challenges in the presence of weather variations, changes in illumination, poor weather conditions, object shadows, water, and low-illumination environments.
Unmanned aerial vehicles (UAVs) offer a flexible and reliable method for monitoring pavement marking conditions. Researchers have evaluated the use of UAVs in the transportation system and have presented examples of utilizing UAV images to identify pavement marking defects.

Although obtaining UAV images is a speedy and not-too-expensive process, there remains a deficiency in accurately identifying pavement marking defects. [16] [17] [18].

Chapter8: Dataset collection for pavement defects

Dataset collection:

Road pavement kinds are a crucial quality objective. By gathering adequate data for the training process on the non-conformity image dataset. The output variables associated with predictions made using the deep learning model are guaranteed to be accurate and reliable. The amount of the dataset improves the model's performance.

The RDD2022 open-source dataset, which was utilized as the experimental dataset in this study, is publicly accessible for research and teaching. The Crowdsensing-based Road Damage Detection Challenge (CRDDC), a Big Data Cup run in conjunction with the IEEE International Conference on Big Data'2022, is the name of the competition. the information consists of 47420 road photos that were gathered in China, Japan, the Czech Republic, India, and the United in order to suggest techniques for autonomous road damage detection in various nations. Smartphones were used to record the dataset, Google Street View images taken using high-resolution cameras from cars, motorcycles, and drones. However, because we have a limited amount of memory space and big data training model resources, we only selected a portion of the dataset (9290 Images of Road Pavement with Resolution 600 x 600 Pixels) that was collected from Japan. pictures with
annotations in PASCAL VOC format XML files. in this study, **the damage categories considered are**:

- Alligator Cracks
- Damaged crosswalk
- Damaged paint
- Longitudinal Cracks
- Manhole cover
- Potholes
- Transverse Cracks

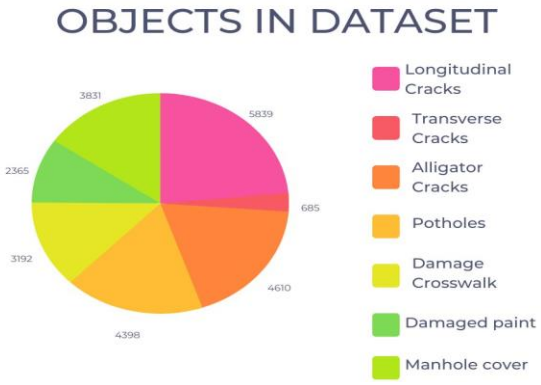


Fig 8.1: objects in dataset

Fig8.2: sample of dataset

Split the dataset:

The experimental dataset was randomly divided into 3 subsets for training, validation and testing. To prevent model overfitting, tune hyperparameters, and get unbiased. The relative sizes of the subsets were 70% (training), 20% (validation) and 10% (testing), respectively.

Yolo annotations:

Drawing a bounding box over the object in an annotation allows us to map an object to its corresponding label. As we previously said in the first chapter, Class labels and bounding box coordinates, defined by four values (xmin, ymin, xmax, ymax), are used for the dataset's annotations, which follow the PASCAL VOC XML format.

Fig 8.3: Annotations in PASCAL VOC

In Fig 7.3, We can see the bounding box file structure in xml format and related image example. To make XML format compatible with Yolo annotations format, we changed it to txt format. each line in the text file describes a bounding box (class x_center y_center width height)

```
6 0.6125 0.7283333333333334 0.765 0.07666666666666666
6 0.13916666666666666 0.7291666666666666 0.175 0.035
6 0.07833333333333334 0.7766666666666666 0.15666666666666668 0.02
2 0.18 0.7841666666666667 0.36 0.425
6 0.075 0.8083333333333333 0.15 0.03333333333333333
6 0.0225 0.8683333333333333 0.045 0.023333333333333334
```

Fig 8.4. annotation of the previous bounding boxes image in txt format

Challenges in dataset:

1. Imbalance dataset
2. Background size relative to defect size

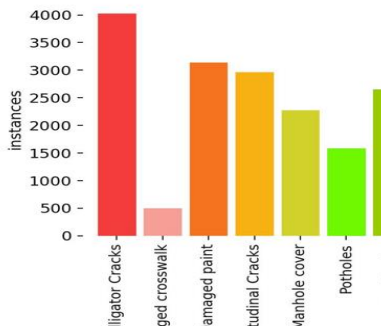


Fig 8.5: classes imbalance

fig 8.6: sample of dataset

Imbalance dataset:

In general imbalance problems have a large scope in machine learning, computer vision and pattern recognition, occur when the distribution regarding that input property affects the final object detection training performance.

There are 4 types of imbalances:

- Scale imbalance
- Spatial imbalance
- Objective imbalance
- Class imbalance

In our case we faced Class imbalance in the dataset . Class imbalance is observed when a class has more examples than others in the dataset. There is a clear and significant inequality among classes . in our case there are two classes : Damaged crosswalk and Damaged paint and are almost same.

Background size relative to defect size:

We faced another challenge in the dataset the images were taken as a wide view, However, the background space (many other objects) takes almost of images. So that it is difficult for a detector to detect pavement damage.

In the Methodology section we suggest and discuss solutions for how handling with these problems.

Chapter 9: Methodology for pavement defects

Methodology

The method makes use of the cutting-edge object detection model YOLOv5. The RDD2022 dataset, a sizable dataset of road damage, is used to assess the approach. The findings demonstrate the method's high degree of accuracy in detecting road defects.

YOLO works by dividing a picture into a SxS grid of cells, with each cell subsequently being in charge of producing bounding boxes. The actual feature extraction is subsequently handled by the convolutional layers. The described bounding boxes encircle the identified object, and YOLO then assigns a confidence score to the enclosing box. Each forecast consists of five elements, including the box's x, y, and (w, h) width and height coordinates. The (x,y) and (w,h) coordinates of the box's width and height make up the five components that make up each forecast. The normalized values of X, Y, W, and H are all between 0 and 1. The confidence score makes up the sixth element. The confidence score is calculated by YOLO using the IOU (Intersection over Union) evaluation metric. The IOU between the box that YOLO predicted and the box that contains the target object is the confidence score. Pr(Object) is the likelihood that the desired object will appear on the grid, and IOU is the likelihood that the ground truth and the bounding box will overlap. The unnecessary bounding boxes are eliminated using the Non-Maximum Suppression (NMS) technique. assurance as **Pr(Object)* IOU predtruth Pr(Class|Object)** represents the likelihood that the desired object will be found inside a bounding box. Pr(class)*IOU is employed to obtain class-specific confidence in each box. A total of 106 layers make up the basic architecture of YOLO, including 53 levels for the detection job and an additional 53 layers for DARKNET-53, which will be discussed in the next section. The neural network framework DARKNET [19], developed by Joseph Redmon and written in CUDA and C, serves as the foundation of YOLO. Its benefits include being quick, slim, and simple to use.

Most of the images in the training set are 600 X 600 pixels in size. On a desktop PC with access to the Google Colab and Kaggle virtual machines, the object detection models were trained. The procedure for performing tasks in order to find defects in the road surface is shown in Figure 3.1. The various YOLO models were initially reconfigured to be suited for the tasks of defect

detection using an RDD Japan benchmark dataset. The models were then trained and verified up until the loss function's steady-state line, at which point the changing average in the loss was no longer significant. The accuracy of object detection, which calls for creating a bounding box around each object in the image that is detected, was confirmed by evaluating the performance of an object detector.

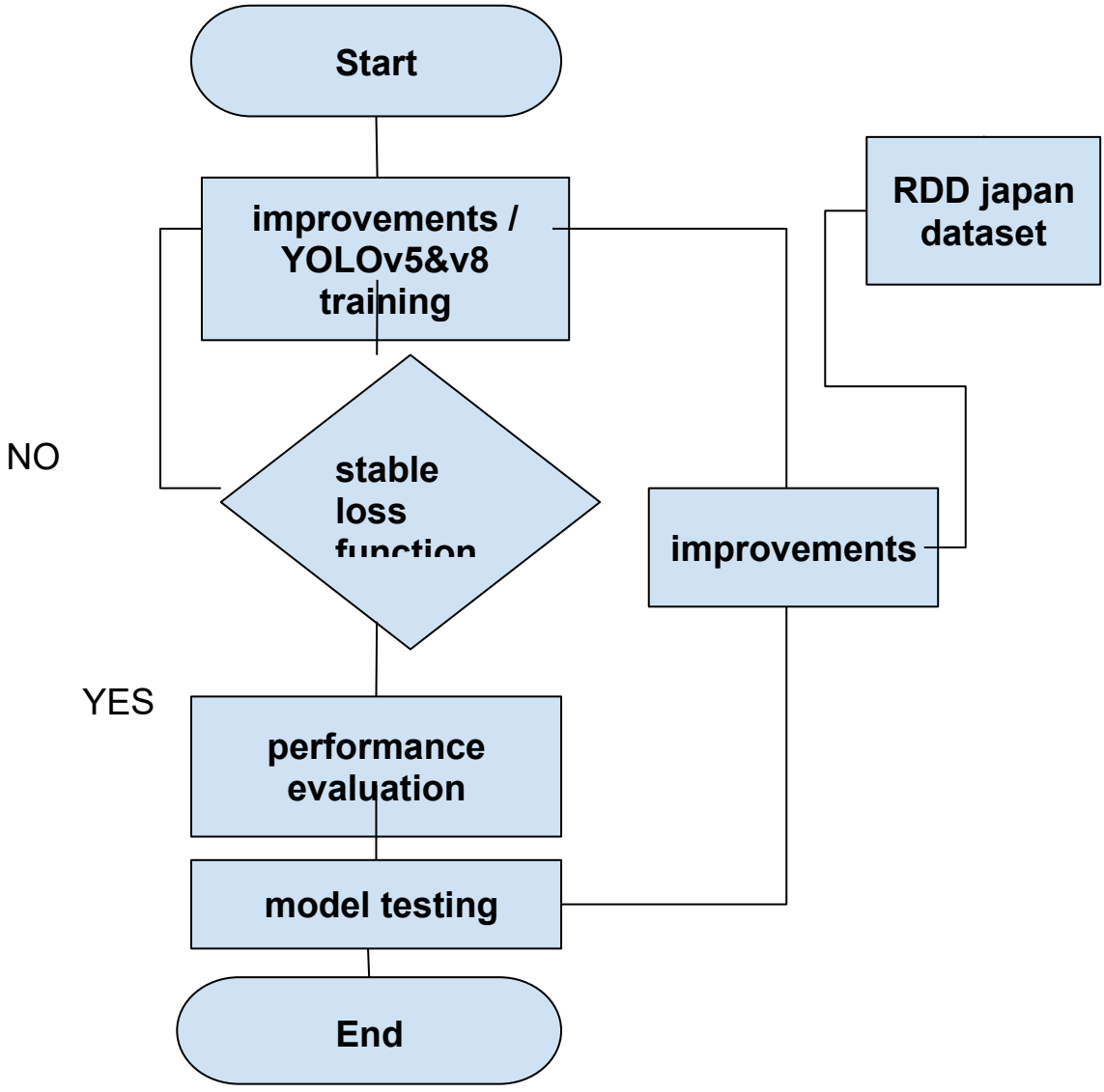


Fig 9.1:Detection of pavement defects methodology

Weights and loss function

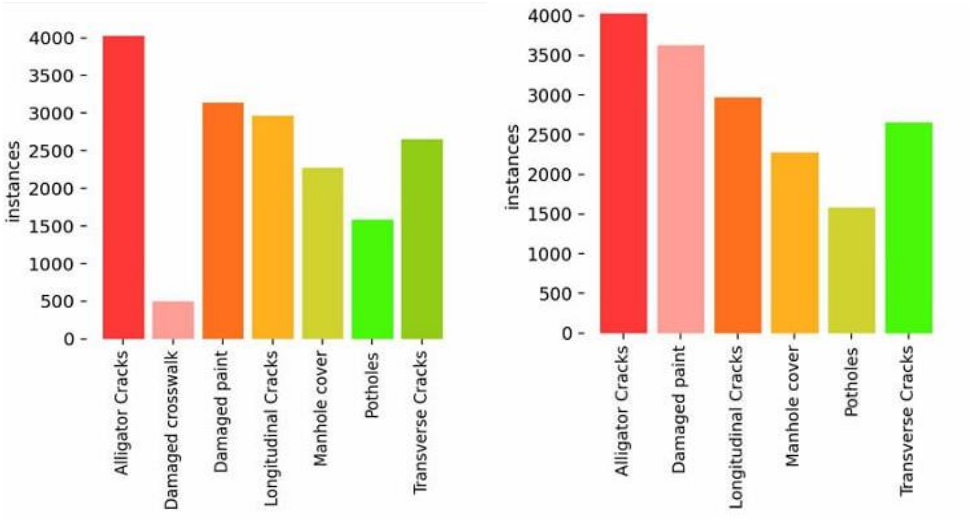
Weights in the Loss Function are a simple and good technique for handling Class Imbalance in our training dataset, obtaining higher accuracy, and improving your model performance. We experimented with Different weightings to achieve optimal performance. We created a tensor of weights based on the frequency of each class the final weights that have been used in the training dataset (0.35, 0.4, 0.5, 0.7, 0.8, 0.6) weights are different from one dataset to another. Modifying the hyperparameters must suit the nature of your dataset to get as best results you can from the used model. Modify only parameters that will make a difference in your dataset or in your model performance. Adjusting hyperparameters of your model must be done right otherwise it will lead to unwanted results. We have made a comparison before and after adding weighted loss see the results section.
Binary Cross-Entropy (BCE) Loss in equation (1) is shown below:

$$Lossn = -w[yn\ logxn + (1 - yn)log(1 - xn)] \quad (1)$$

where w is the weight, yn is the labelled value, and xn is the predicted value of the model.

Merging process

As mentioned in the dataset chapter, we faced an imbalance problem of RDD dataset classes so we merged two classes together **paint damage class and the crosswalk damage class** into one class and made them have one label (**paint damage**) as they similar, were the most minority classes and had the same characteristics, so we thought to consider crosswalk damages which had the smallest number of images as damage in paint too then we updated their annotations to suite the merging process. We changed all annotations containing crosswalk class Id to paint class Id. figure 3.2 shows classes before and after merging process



before merging after merging

fig 9.2: classes before and after merging process.

Semantic segmentation

The process of splitting a digital image into several image segments, also known as image regions or objects (pixel sets), is known as image segmentation in the context of digital image processing and computer vision. The goal of segmentation is to make an image's representation more understandable and straightforward to examine. To identify objects and borders (lines, curves, etc.) inside images, image segmentation is typically utilized. In summary, the procedure involves giving each pixel in an image a tag so that pixels with the same tag share a particular characteristic. Each pixel in a region is similar in terms of a feature or computed attribute, such as color, intensity, or texture.

blackout background elements

The second problem we faced was the percentage of background-size relative to the defect size as the RDD Japan dataset is a wide view dataset containing other elements that take up the largest space of the image so the model search for small objects in the whole image so we thought if we made the model focus and search about defects in the pavement only that may improve the performance. by applying semantic segmentations on dataset images to mask the pavement with a specific color. now abstraction of pavement becomes easier using its color ranges. we used for segmentation internImage pre-trained model trained on a cityscape dataset which has a particular mask for the pavement and road class with color id to identify pavement.Fig8.3shows some samples of model output.

Fig 9.3: output of the pre-trained model on the cityscape dataset

then we blacked out background elements using pavement color range(lower_range =[127, 36, 33] & upper_range = [179,255, 255]) and visualized images with bound boxes to assure that none of the defects blacked out with other image elements and the model still able to identify defects through the color. As shown in Fig 8.4

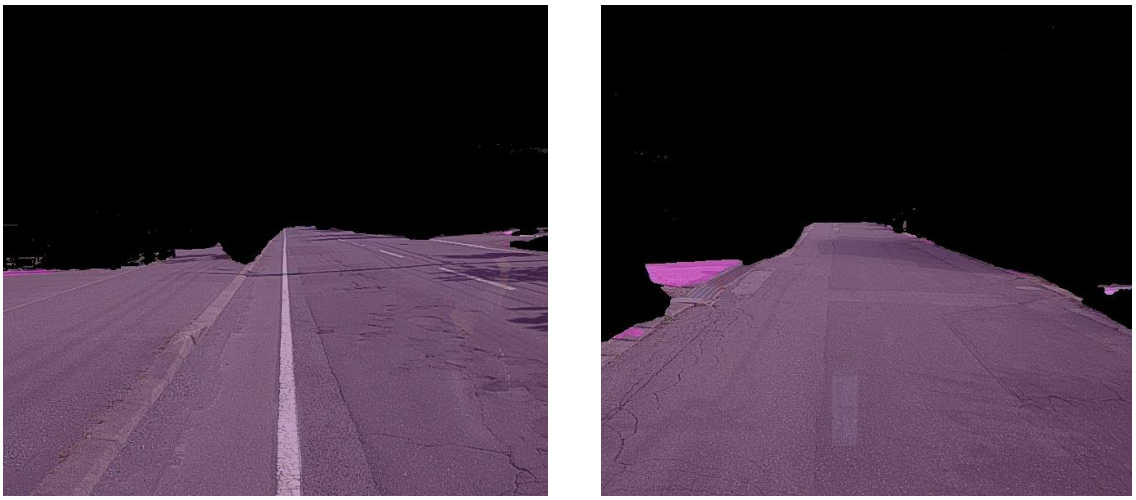


Fig 9.4: blackout of the background process

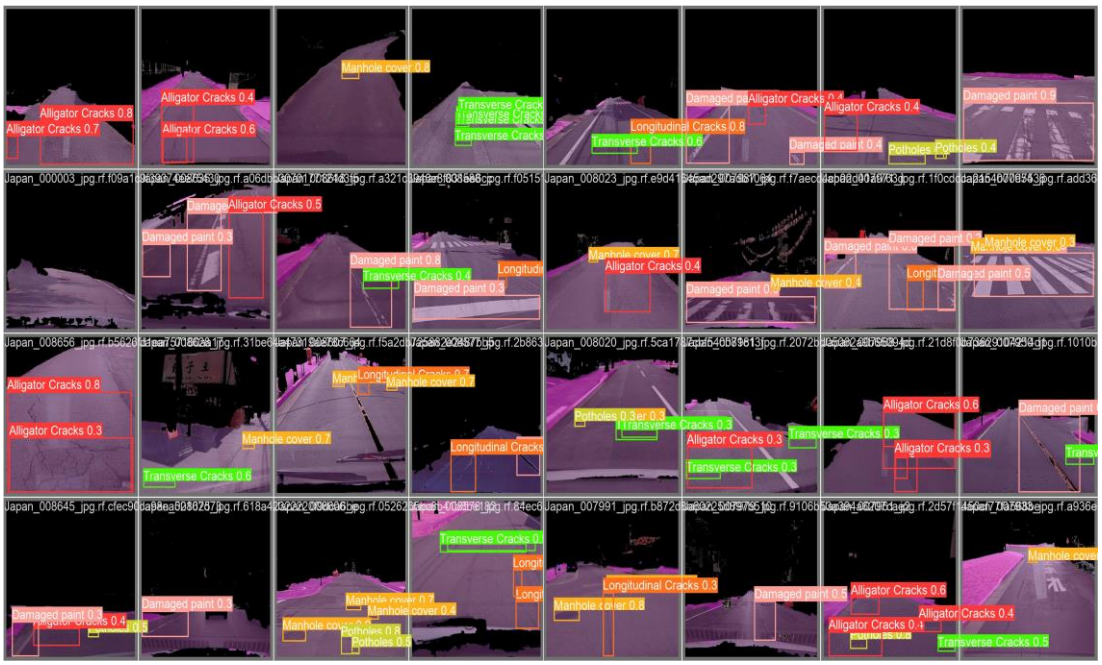


fig 9.5: visualization from a validation batch

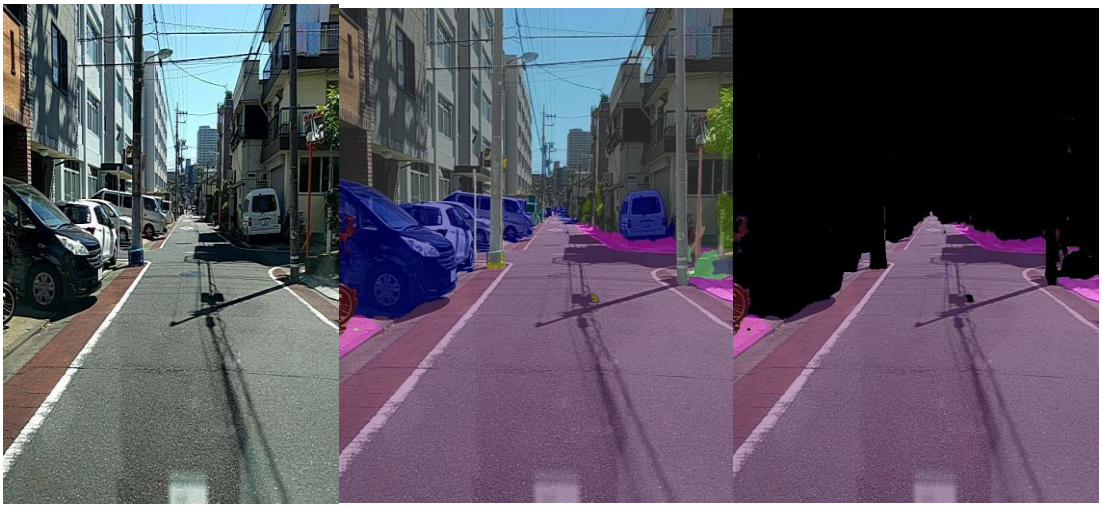


Fig 9.6 :Phases of pavement abstraction

Cityscape dataset

A large database that includes a variety of stereo video sequences shot in street settings across 50 different cities, together with a bigger set of 20 000 weakly annotated frames and 5 000 frames with high-quality pixel-level annotations. As a result, the dataset is significantly bigger than past attempts of a comparable nature. The Cityscapes Dataset is designed to evaluate the effectiveness of vision algorithms for the three main tasks of

semantic urban scene understanding: pixel-level, instance-level, and panoptic semantic labeling. assisting studies that seek to utilize massive amounts of (weakly) annotated data, such as for deep learning training.

merging the blacked-out dataset

We then merged the two mentioned classes **paint damage class and crosswalk damage class** in the segmented dataset too and run our model on the two states like the original dataset before and after the merging process.

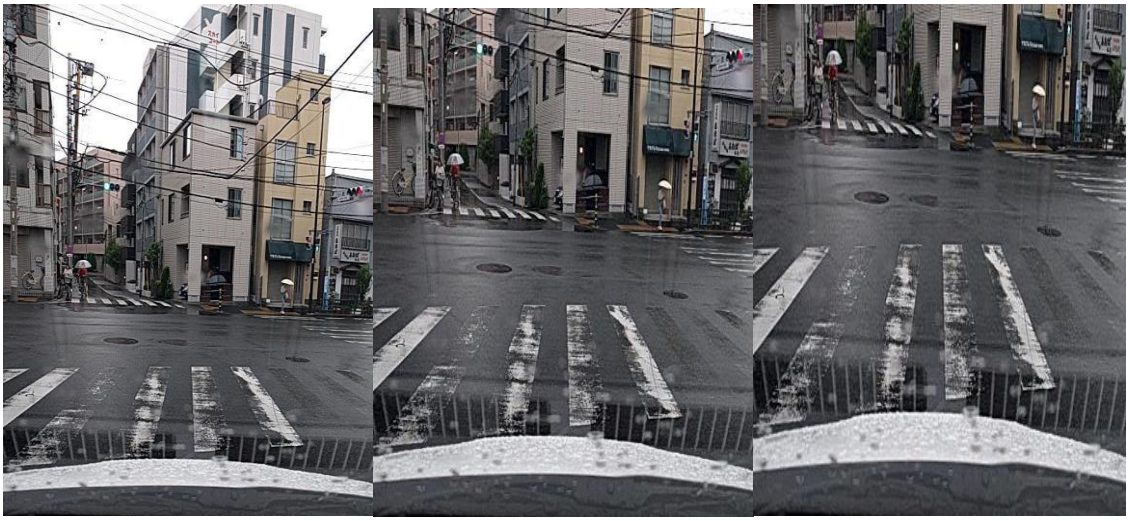
Blackout as a preprocess

we used test data of normal dataset (without segmentation) and weights of training model on the segmented dataset to see if it will be the same performance as the segmented test dataset and defects will be detected in not masked with background elements images as in segmented date. as expected results (see in results chapter) to normal test dataset was fine but much lower than segmented test images.

segmentation is a preprocess that should be done on any dataset that will be tested from a trained model with segmentation dataset weights, but if you do not have the ability to segment and blackout the background elements the performance still be good enough.

Cropping the top of the image

Another approach we applied to the RDD dataset has the best improvement and best results among all the work done we cut the top area from the images of the original not segmented dataset as the pavement that contains the defects located at the lower part of the image. most images in the dataset have a size of 600x600 pixels. we tested two versions of cutting. version its images cut to 600x420 pixels and the second version cut to 600x330 pixels. these ranges in most images contain the pavement. both have very close results and almost the same accuracy but the version with 600x420 pixel is a little bit better.



size 600*600 **size 600*420** **size 600*330**

Size 600*600 **size 600*420** **size 600*330**

Fig 9.7 : images for cutting process with two different sizes.

Chapter 10: Experiments and results

Results:

We showed various improvements on datasets like segmentation, merging, and cropping, and others on YOLO models like loss functions and training with different sizes (medium & large). The tables below show the results of various experiments we developed through pavement defect detection tasks. Cropped version one has the best results in all experiments with mAP 63.4 % on YOLO v8 witch better than YOLO v5 at the same conditions in the case of our dataset.

models	mAP	precision	recall	f1 score
YOLOv5	61.6%	64.6%	**60.1%**	62.27
YOLOv8	**62.2 %**	**65%**	61%	**62.94**

Table 10.1: Normal Japan dataset without any improvements on dataset

	mAP	precision	recall	f1-score
without weighted loss	60.9%	65.1 %	**59.6%**	**62.23**
with weighted loss	**61.4%**	67.9%	57.3	62.15

Table 10.2 :YOLOv5 results before and after weighted loss using Yolov5m.

models	mAP	precision	recall	f1 score
YOLOv5	58.3	63	55.3	58.9
YOLOv8	**60**	**64.4**	**56.7**	**60.34**

Table 10.3: normal dataset results after merging process.

models	mAP	precision	recall	f1 score
YOLOv5	60.5	63.7	**58.9**	61.21
YOLOv8	**62.4**	**66.1**	58.8	**62.24**

Table10.4:blacked out Japan dataset to solve background elements problem.

models	mAP	precision	recall	f1 score
YOLOv5	57.2	63	**54.2**	58.27
YOLOv8	**58.8**	**67.3**	54.1	**59.98**

Table 10.5 :blacked-out dataset results after merging process.

models	mAP	precision	recall	f1 score
YOLOv5	61.5	60.7	**63.5**	62.07
YOLOv8	**63.4**	**66.9**	58.7	**62.53**

Table 10.6: cropped dataset 1st version results.

models	mAP	precision	recall	f1 score
YOLOv5	60.9	60.9	**59.3**	60.1
YOLOv8	**63**	**67.3**	58.7	**62.71**

Table 10.7: cropped dataset 2nd version results.

Pretrained model	mAP	precision	recall	f1 score
YOLOv8m	**62.2**	**65.6**	58	61.56
YOLOv8l	**62.2**	65	**61**	**62.936**

Table 10.8: YOLOv8 deference in performance between large and medium model size.

models	mAP	precision	recall	f1 score
Normal dataset	54.5	61.9	51.5	56.22

Table 10.9:test set results of normal dataset using training weights of segmented dataset.

Chapter 11: Conclusion

We improved the accuracy of models in classification of road signs task by modifying the architectures of pre-trained models and using ensemble methods. This resulted in an accuracy of 99.17% in german traffic signs dataset benchmark and obtain high map accuracy in detection of road sign where the map50 arrived to 96.9% in the Tiny LISA Traffic Sign Detection Dataset.

Soft voting is more accurate than hard voting but more complex and Combine classifiers together leads to higher results and improve your model performance.

Data augmentation must be performed in a manner that is appropriate for the dataset, or it will result in undesirable outcomes such as overfitting, underfitting, bias, and noise.
We used the two models YOLOv5 and YOLOv8 for object detection.

Merging classes in RDD can be an effective way to balance our dataset and results increase model performance, YOLOv8 is more accurate than YOLOv5 for our datasets, under the same training conditions on the RDD dataset.

References

[1] Yamada K., Aryuanto and Limpraptono F. Y., ""“Intelligent machine vision system for traffic sign recognition”," November 2008..

[2] Alturki, A.S, "Traffic Sign Detection and Recognition Using Adaptive Threshold Segmentation with Fuzzy," *In Proceedings of the 2018 International Symposium on Networks,,* p. pp. 1–7, June 2018.

[3] Kulambayev, Bakhytzhan, G. Beissenova, N. Katayev, B. Abduraimova, L. Zhaidakbayeva, A. Sarbassova and O. Akhmetova, "Kulambayev, Bakhytzhan, et al. "A Deep Learning-Based Approach for Road Surface Damage Detection," Computers, Materials & Continua, 2022.

[4] Girshic and Ross., "Fast r-cnn.," in *IEEE international conference on computer vision.* , 2015.

[5] Ren and S. e. al, "Faster r-cnn: Towards real-time object detection with region proposal networks," *Advances in neural information processing systems ,* 2015.

[6] Redmon and Joseph, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016.

[7] Dai, Jifeng, K. He and a. J. Sun., "Instance-aware semantic segmentation via multi-task network cascades.," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016.

[8] Chen, Xiao, X. Zhang, J. Li, M. Ren and B. Zhou, "A new method for automated monitoring of road pavement aging conditions based on recurrent neural network," *IEEE Transactions on Intelligent Transportation Systems,* pp. 24510-24523., 2022.

[9] C.-H. Ho, M. Snyder and D. Zhang, "Application of vehicle-based sensing technology in monitoring vibration response of pavement conditions," *Journal of Transportation Engineering,* 2020.

[10] Q. Yang and S. Zhou, "Identification of asphalt pavement transverse cracking based on vehicle vibration signal analysis," *Road Materials and Pavement Design,* 2021.

[11] Lin, Yu-Chen, W.-H. Chen and C.-H. Kuo, "Implementation of pavement defect detection system on edge computing platform," *Applied Sciences ,* 2021.

[12] Fernandes, F. M and J. C. Pais, "Laboratory observation of cracks in road pavements with GPR," *Construction and Building Materials ,* 2017.

[13] Lin, Jin and Y. Liu, "Potholes detection based on SVM in the pavement distress image," *010 Ninth International Symposium on Distributed Computing and Applications to Business, Engineering and Science. IEEE,* 2010.

[14] Chung, T. Duc and M. A. Khan, "Watershed-based real-time image processing for multi-potholes detection on asphalt road," in *IEEE 9th International Conference on System Engineering and Technology*, 2049.

[15] Baek, Ji-Won and K. Chung, "Pothole classification model using edge detection in road image," *Applied Sciences,* 2020.

[16] Bu, Tianxiang, J. Zhu and T. Ma, "A UAV photography–based detection method for defective road marking.," *Journal of Performance of Constructed Facilities,* 2022.

[17] Khan and M. A. e. al., "UAV-based traffic analysis: A universal guiding framework based on literature survey," *Transportation research procedia,* 2017.

[18] Ruiza, A. Leal and H. Alzraieeb., "Automated pavement marking defects detection," in *ISARC. Proceedings of the International Symposium on Automation and Robotics in Construction*, 2020.

[19] Redmon J, Divvala S, Girshick R, Farhadi, "You only look once: unifed, real-time object detection," *IEEE,* pp. 779-788, 2016.

[20] Arcos-Garc´ıa A, Alvarez-Garc ´ ´ıa JA, Soria-Morillo LM, "Deep neural network for traffic sign recognition systems," *an analysis of spatial transformers and stochastic optimisation methods,* p. Neural Netw 99:158–165, (2018).

[21] Yu J, Zhu C, Zhang J, Huang Q, Tao D, "Spatial pyramid-enhanced NetVLAD with weighted," *IEEE Trans Neural Netw Learn Syst,* 2019.

[22] Bangquan X, Xiong WX, "Real-time embedded traffic sign recognition using efficient convolutional," *IEEE Access 7:53330–53346,* 2019.

[23] Girshick R, Donahue J, Darrell T, Malik J, "Rich feature hierarchies for accurate object detection and semantic segmentation," *Proceedings of the IEEE conference on computer vision and pattern recognition,* pp. 580-587, 2014.

[24] Xing J, Yan W, "Traffic sign recognition using guided image filtering. International Symposium on Geometry and Vision (ISGV),," *Springer, Berlin,* pp. 957-961, 2021.

[25] J. Stallkamp, M. Schlipsing, J. Salmen and C. Igel, "The German Traffic Sign Recognition Benchmark: A multi-class classification competition.," *IEEE International Joint Conference on Neural Networks. ,* pp. 1453-1460, 2011.

[26] P. Sermanet and Y. LeCun, , "Traffic sign recognition with multiscale convolutional networks," *International Joint Conference on,* p. 2809 – 2813., 2011.

[27] Nam J, Kim S, "Heterogeneous defect prediction," *Proceedings of the 2015 10th joint meeting on foundations of software engineering,* pp. 508-519, 2015.

[28] Yoav Freund and Robert E. Schapire., "A decision-theoretic generalization of on-line learning and an application to boosting.," *Journal of Computer and System Sciences,* p. 55(1):119–139, August 1997.

[29] Ke, Guolin, et al, "“Lightgbm: A highly efficient gradient boosting decision tree.” Advances in neural information processing systems," p. 3146–3154, (2017).

[30] Gray, Nicholas & Moraes, Megan & Bian, Jiang & Tian, Allen & Wang, Alex & Xiong, Haoyi & Guo, Zhishan. (2022). GLARE: A Dataset for Traffic Sign Detection in Sun Glare.

[31] A. Mogelmose, M. M. Trivedi and T. B. Moeslund, "Vision-Based Traffic Sign Detection and Analysis for Intelligent Driver Assistance Systems: Perspectives and Survey," in IEEE Transactions on Intelligent Transportation Systems, vol. 13, no. 4, pp. 1484-1497, Dec. 2012, doi: 10.1109/TITS.2012.2209421.

[32]M. Lopez-Montiel, U. Orozco-Rosas, M. Sánchez-Adame, K. Picos and O. H. M. Ross, "Evaluation Method of Deep Learning-Based Embedded Systems for Traffic Sign Detection," in IEEE Access, vol. 9, pp. 101217-101238, 2021, doi: 10.1109/ACCESS.2021.3097969.

[33] M. Swathi and K. V. Suresh, "Automatic traffic sign detection and recognition: A review," 2017 International Conference on Algorithms, Methodology, Models and Applications in

Emerging Technologies (ICAMMAET), Chennai, India, 2017, pp. 1-6, doi: 10.1109/ICAMMAET.2017.8186650.

[34] Houben, Sebastian & Stallkamp, Johannes & Salmen, Jan & Schlipsing, Marc & Igel, Christian. (2013). Detection of Traffic Signs in Real-World Images: The German Traffic Sign Detection Benchmark. Proceedings of the International Joint Conference on Neural Networks. 10.1109/IJCNN.2013.6706807.

[35] Tabelini, Lucas, Thiago Meireles Paixão, Rodrigo Berriel, Alberto Ferreira de Souza, Claudine Santos Badue, N. Sebe and Thiago Oliveira-Santos. “Effortless Deep Training for Traffic Sign Detection Using Templates and Arbitrary Natural Images.” *2019 International Joint Conference on Neural Networks (IJCNN)* (2019): 1-7.

[36] D. Tabernik and D. Skočaj, "Deep Learning for Large-Scale Traffic-Sign Detection and Recognition," in IEEE Transactions on Intelligent Transportation Systems, vol. 21, no. 4, pp. 1427-1440, April 2020, doi: 10.1109/TITS.2019.2913588.

[37] Bouti, Amal & Mahraz, M. & Riffi, Jamal & Tairi, H.. (2020). A robust system for road sign detection and classification using LeNet architecture based on convolutional neural network. Soft Computing. 24. 10.1007/s00500-019-04307-6.

[38] D. Temel, T. Alshawi, M-H. Chen, and G. AlRegib, ”Challenging Environments for Traffic Sign Detection: Reliability Assessment under Inclement Conditions,” in arXiv:1902.06857, 2019.
Park, Sung, Tran, Van, and Dong Lee. "Application of Various YOLO Models for Computer Vision-Based Real-Time Pothole Detection." Applied Sciences 11, no. 23 (2021): 11229. Accessed July 1, 2023. https://doi.org/10.3390/app112311229.

[39]A comparative study of YOLOv5 models performance for image localization and classification September 2022 Conference: 33rd Central European Conference on Information and Intelligent Systems (CECIIS 2022)At: Dubrovnik, Croatia

[40]https://docs.ultralytics.com/yolov5/tutorials/architecture_description/#1-model-structure

[41]YOLOv5_Improved_YOLOv5_Based_on_Transformer_Prediction_Head_for_Object_ICCVW_2021_paper.

[42]https://github.com/ultralytics/yolov5

[43]https://blog.roboflow.com/yolov5-improvements-and-evaluation/

[44]Ju, Rui, and Weiming Cai. "Fracture Detection in Pediatric Wrist Trauma X-ray Images Using YOLOv8 Algorithm." ArXiv, (2023). Accessed July 1, 2023. /abs/2304.05071.

[45]https://github.com/ultralytics/ultralytics

[46]Malta, Ana, Mateus Mendes, and Torres Farinha. 2021. "Augmented Reality Maintenance Assistant Using YOLOv5" Applied Sciences 11, no. 11: 4758.
https://doi.org/10.3390/app11114758