E-SCOOTER RIDER DETECTION SYSTEM IN DRIVING ENVIRONMENTS

by

Kumar Apurv

A Thesis

Submitted to the Faculty of Purdue University

In Partial Fulfillment of the Requirements for the degree of

Master of Science

Department of Computer and Information Science

Indianapolis, Indiana

August 2021

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL

Dr. Renran Tian

Department of Computer Information & Graphics Technology

Dr. Jiang Yu Zheng

Department of Computer & Information Science

Dr. Gavriil Tsechpenakis

Department of Computer & Information Science

Approved by:

Dr. Shiaofen Fang

To the loving memories of
my grandfather and my grandmother

ACKNOWLEDGMENTS

This thesis would not have been possible without the unconditional support of many, and I would like to take this opportunity to give my thanks to the ones who made it possible.

First, I would like to express my sincere thanks to Dr. Renran Tian for his valuable support and guidance throughout. His invaluable expertise in formulating this research and methodologies helped me push myself into a highly creative zone. His support in providing me with all the tools I needed helped me move in the right direction and complete my work.

I would like to thank Dr. Jiang Yu Zheng and Dr. Gavriil Tsechpenakis for serving on my thesis committee and their invaluable suggestions on improving my work. Their brainstorming helped me figure out the constraints in my work and improve it thereafter.

Furthermore, this work would not have been possible without the love and understanding of my friends and family. Their continuous support made me want to be a better version of myself and create something valuable every day. They are always there for beautiful conversations and moral support. This journey would not have been possible without them.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

CNN	Convolutional Neural Networks
ROI	Region of Interest
SGD	Stochastic Gradient Descent
MS COCO	Microsoft Common Objects in Context
ReLU	Rectified Linear Unit
YOLO	You Only Look Once
TP	True Positive
TN	True Negative
FP	False Positive
FN	False Negative
TPR	True Positive Rate
FPR	False Positive Rate
ROC	Receiver Operating Characteristic Curve
AUC	Area Under Curve

ABSTRACT

E-scooters have become omnipresent vehicles in major cities around the world. The numbers of e-scooters keep escalating, increasing their interactions with other cars on the road. An e-scooter rider's normal behavior varies enormously from other vulnerable road users. This situation creates new challenges for vehicle active safety systems and automated driving functionalities, which require the detection of e-scooter riders as the first step. There is no open-sourced existing image dataset or computer vision model to detect these e-scooter riders. This work presents a novel vision-based system to differentiate between e-scooter riders and regular pedestrians and a benchmark dataset for e-scooter riders in a natural environment. An efficient pipeline built using two existing state-of-the-art convolutional neural networks (CNN), You Only Look Once (YOLOv3) and MobileNetV2, performs detection of these vulnerable vehicle riders. By fine-tuning and training MobileNetV2 over the dataset, the model can precisely distinguish between e-scooter riders and pedestrians in an input image. The whole pipeline can generate output at a precision of 0.95 and a recall of around 0.75 on the test sample. Moreover, the classification accuracy of trained MobileNetV2 on top of YOLOv3 is over 91% for e-scooter riders with precision and recall over 0.9.

1. INTRODUCTION

1.1 Background

Today, we can see the rapid growth of e-scooters especially across the United States and UK. E-scooters are a great way to commute for shorter distances due to its convenience and cost-effectiveness. They are seen as the vehicles of the future as they are also eco friendly, and many cities have welcomed this idea wholeheartedly. However, at the same time, e-scooters can be seen as dangerous means of transport [1], not just to the riders but also to other pedestrians and vehicles on the road.

E-scooter's movements are swift and risky. Accidents can easily occur and there has been a spike in injuries over the last few years. A report by Journal of the American Medical Association (JAMA) [2] shows a 365% increase in annual hospital admissions in North America after the e-scooter sharing companies came into the picture in 2018. Many of these accidents are related to the e-scooter's interaction with cars and other vehicles on the road. Although these numbers are scary, e-scooters can still be seen as the way forward [3] if efforts are shown to improve the safety in the environment around e-scooters.

1.2 Motivation

E-scooter riders can be detected using a typical computer vision model. This detection mechanism can help these vehicles understand the E-scooters' behavior and predict and attenuate possible accidents. Existing deep learning and computer vision models can detect people in natural environments, but that alone is not enough to identify the e-scooter riders. The natural behavior of e-scooter riders is different compared to a pedestrian on the street, and therefore computer vision models should be able to distinguish between the two.

While deep learning models are very good at interpreting patterns of the object, they need a massive amount of data to train. Without a lot of data, these models cannot learn the complexities of the object very well. There is no open-sourced dataset for the e-scooter riders, and therefore to build a model, the data needs to be collected from natural environments. Deep learning models are also computationally expensive to train. Training such models

on millions of data samples can sometimes take days to finish. Due to this difficulty, many large organizations make their trained models and datasets available to the public to help the research community.

1.3 Contributions

In this work, a system to detect e-scooter riders is built. This system is able to detect people and distinguish e-scooter riders among them. For an image input, the system outputs bounding box coordinates for people and e-scooter riders.

Today, many open-source object detection models can detect people in images with high accuracy. An object detection model is used to detect people in the image, and also to generate a benchmark dataset for e-scooter riders. Using the benchmark dataset, an image classifier is trained to distinguish e-scooter riders amongst people.

The system generates output at an accuracy of 93%. This detection model can help the autonomous vehicles learn and predict the behavior of the e-scooter riders and thus mitigate potential accidents.

2. LITERATURE SURVEY

2.1 Object Detection and Localization

Object detection is a task in computer vision involving both finding one or more objects within an image and classifying each object in the image. Localization is identifying the location of these objects and drawing a bounding box around these objects. It is a challenging task in computer vision to successfully detect and localize multiple objects in the image. However, over the last decade, there have been some significant achievements in computer vision.

Before deep learning, object detection techniques used feature descriptors to obtain embeddings from various regions in the image. The early object detection system typically consisted of three different stages:

(1) Proposal generation: The objective was to find the region of interest (ROI) in the image using sliding windows.

(2) Feature vector extraction: From the sliding windows, the semantic information was captured. Feature vectors encoded by low-level descriptors like Haar, HOG (Histogram of Gradients), or SIFT (Scale Invariant Feature Transform) represent this semantic information.

(3) Region classification: In the final step, region classifiers learned, generally using SVM (Support Vector Machines), to assign categorical labels to each ROI.

Progress in these traditional methods was limited and had small developments in building complicated systems.

Today, deep learning techniques [4] outperform all the traditional methods for classification, detection, and segmentation tasks. The developments in computing resources and availability of large-scale image datasets, such as ImageNet [5], made it possible to train several convolutional neural networks [6]. A typical object detection model based on deep learning can be categorized into two-stage detectors (R-CNN, Fast R-CNN, and its variants) and one-stage detectors (YOLO and its variants).

2.1.1 R-CNN

R-CNN (Region-based Convolutional Neural Networks) [7] is a two-stage object detection method. Firstly, it identifies bounding boxes or regions of interest for objects using selective search. In the second step, it extracts independent features through CNN from each region of interest for classification. Since features are extracted from each region proposal separately, R-CNN is prone to heavy duplicated computations and is time-consuming.

2.1.2 Fast R-CNN

Fast R-CNN, introduced by Girshick [8], makes R-CNN run faster. The training procedure was improved by creating a unified framework obtained from three independent models. Instead of extracting the CNN feature vectors for each region proposal, Fast R-CNN aggregates them into a single forward pass over the entire image. The region proposals share this feature matrix and are used to learn the object classifier and the bounding box regressor, which speeds up the computation, compared with R-CNN, during training and testing. However, the improvement is not significant, and using various models makes the process expensive.

2.1.3 Faster R-CNN

Faster R-CNN [9] was an attempt at speeding up the solution. Unlike Fast R-CNN, it integrated the region proposal into the CNN model and proposed a unified RPN model (Region Proposal Network). RPN generates object proposals on each position of the feature map using a sliding window and then a feature vector for each position. These feature vectors are then fed into two branches called object detection layer and bounding box regression layer. Later, these results were then fed to the final layer for classification and localization.

2.1.4 YOLO

YOLO (You Only Look Once) [10], an end-to-end deep learning model, was developed by Redmon et al. for real-time object detection. YOLO uses a single neural network and

regression techniques to predict the bounding boxes and classification of objects. YOLO divides the whole image into 7×7 grid cells and considers each cell to detect objects. A prediction is made for the location and bounding box coordinates and the class of the object. With this framework, the proposal generation step was omitted altogether, which reduced the computational expenses. Following YOLO, several other variants of the method were released.

2.1.5 YOLOv3

There have been many versions of YOLO over the years, with YOLOv3 [11] providing a significant improvement over the technique. Although the accuracy of the YOLO models is similar but not as good as Region-Based Convolutional Neural Networks (R-CNNs), they are common for object detection because of their detection speed, often demonstrated in real-time on video or with camera feed input.

Architecture:

YOLOv3 model takes an image of shape $(416, 416, 3)$ as an input and passes the input tensor into its Convolutional Neural Network (CNN). The CNN architecture uses Darknet-53 for feature extraction. The Darknet-53 architecture, as shown in table 2.1, mainly uses 3×3 and 1×1 filters with residual networks after every few convolutional layers. These convolutional layers are also followed by batch normalization layers and use Leaky ReLU activation. Pooling is avoided due to its contribution towards the loss of low-level features.

Output of the model:

The output of YOLOv3 is a list of detections represented through their bounding box using six numbers (pc, bx, by, bh, bw, c). Here, (bx, by) is the coordinate of the top left corner of the bounding box, bw is the width and bh is the height of the bounding box respectively, pc is the probability of the prediction, and c is an n-dimensional vector where n is the number of classes the model is trained on. These predictions are then processed again to generate the final predictions. In the post-processing step, the detections with probabilities less than

Table 2.1. Darknet-53 Architecture

	Type	Filters	Size	Output
	conv	32	3x3	256x256
	conv	64	3x3 / 2	128x128
	conv	32	1x1	
1 x	conv	64	3x3	
	residual			128x128
	conv	128	3x3 / 2	64x64
	conv	64	1x1	
2 x	conv	128	3x3	
	residual			64x64
	conv	256	3x3 / 2	32x32
	conv	128	1x1	
8 x	conv	256	3x3	
	conv			32x32
	conv	512	3x3 / 2	16x16
	conv	256	1x1	
8 x	conv	512	3x3	
	residual			16x16
	conv	1024	3x3 / 2	8x8
	conv	512	1x1	
4 x	conv	1024	3x3	
	residual			8x8
	avgpool		Global	
	connected		1000	
	softmax			

a certain threshold are ignored. There can be multiple detections for a single object, and non-max suppression can remove these duplicate detections. Finally, the model outputs the bounding box coordinates and the confidence value for each detection.

COCO Dataset:

The first step for this implementation is to download the pre-trained model weights from the official YOLO website. These weights were trained using DarkNet on the MSCOCO dataset [12], a large-scale object detection, segmentation, and captioning dataset. This dataset defines 80 objects, including pedestrians, cars, bicycles, trucks, buses, and motorcycles. For the scope of this work, we need to detect people who are present in this dataset.

2.2 Visual Feature Extraction

Training deep CNN models is a computationally expensive process. Sometimes these models can take days or even weeks to train on a large dataset fully. Transfer learning can help mitigate this problem [13]. It is a process by which a model trained to solve a problem can be used to solve another similar problem.

Several layers in a CNN learn different features about the object. The initial layers of a neural network typically learn general features of an object, like edges. The intuition behind transfer learning is that these layers already trained to learn general features of objects can be adapted to extract similar available features from new objects. So, the model weights obtained from pre-trained models developed for standard image recognition tasks can be used to give a head start to learn a new image classifier. The models trained on benchmark image datasets, such as ImageNet, can perform new image classification tasks.

Transfer learning reduces the time of training a neural network by a considerable margin. It is essentially helpful if the new datasets are not large enough to train a classifier from scratch. Transfer learning also helps in reducing generalization errors.

The ImageNet challenge gave rise to many state-of-the-art models and several other innovations in the architecture and training mechanisms of CNNs. Many winning models of this competition have released their model weights easily accessible, inspiring the wave in deep learning. These models serve as the ground for transfer learning today. Some of such models are discussed below:

2.2.1 VGG

Proposed by Simonyan and Zisserman, VGG-16 [14] is CNN which was trained for image classification over the ImageNet dataset. The model makes an improvement over AlexNet and achieves a 92.7% accuracy in ImageNet. The architecture of VGG network is shown in 2.1. VGG takes an image of dimensions $(224, 224, 3)$ as input. There are only convolutional and pooling layers in the network. A 3×3 sized kernel is used for convolution, and a 2×2 sized max pooling is used. The VGG network is trained on a total of 138 million parameters

and takes exceptionally long to train. VGG outputs a feature vector representation of size $(1, 4096)$.

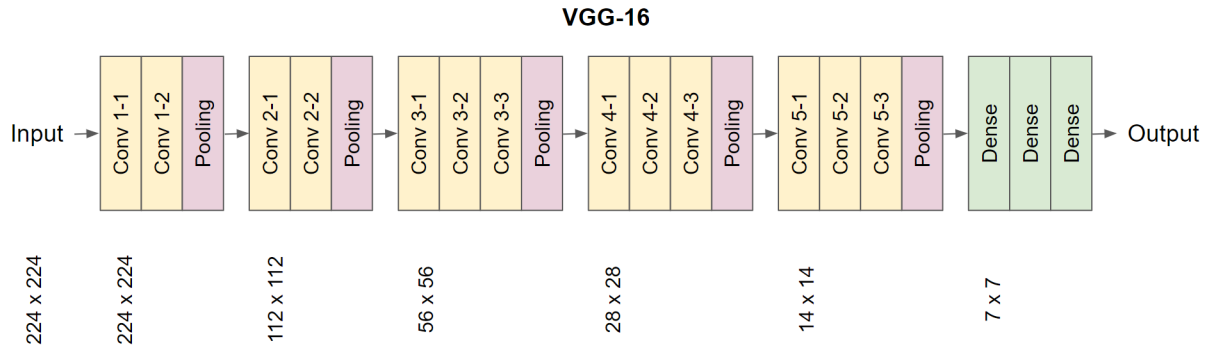

Figure 2.1. VGG-16 architecture

2.2.2 GoogLeNet

Google proposed GoogLeNet (Inception variants) in 2014 [15] and was the winner of the ILSVRC 2014 image classification challenge. It provided significant improvement over AlexNet and VGG. The overall architecture is 22 layers deep and was designed for devices with low computational capacity.

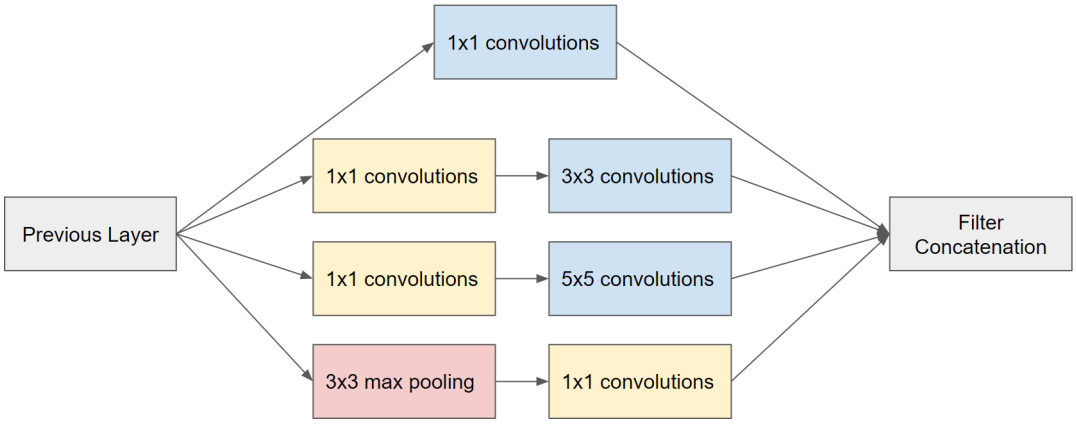

Figure 2.2. Inception module

GoogLeNet uses 1×1 convolution to increase the depth of the architecture and reduces the number of parameters significantly. GoogLeNet uses a global average pooling layer instead

of fully connected layers, which is used in AlexNet [6]. Fully connected layers increase the computational cost as they contain a majority of parameters of the architecture. With the global average pooling layer, GoogLeNet takes a feature map of 7×7 and averages it to 1×1, which significantly reduces the number of parameters to train.

The inception module in figure 2.2 performs a 1×1, 3×3, 5×5 convolution, and 3×3 max pooling on the input. The different sized filters handle objects at multiple scales, and all these outputs are then stacked together to generate a final result.

Some intermediate classifier branches, called an auxiliary classifier, in the middle of the architecture are used during training, eliminating the vanishing gradient problem and also helps in regularization. These branches have a 5×5 average pooling layer with a stride of 3, a 1×1 conv layer with 128 filters, two fully connected layers of 1024 outputs and 1000 linear outputs, and softmax classification layers.

2.2.3 ResNet50

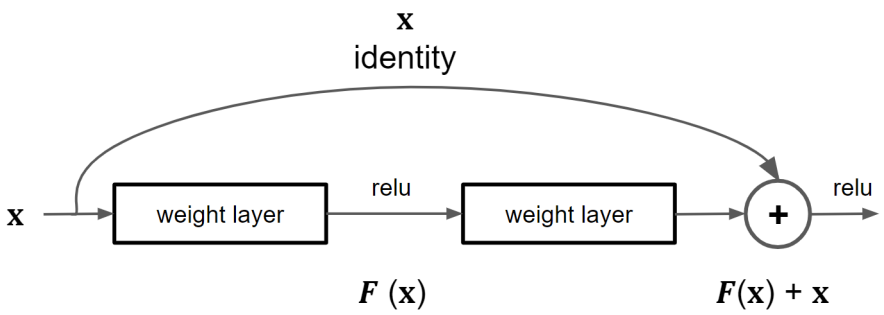

Figure 2.3. Residual block

ResNet50 is a variant of the ResNet model [16]. During backpropagation, a deep neural network can face vanishing gradients problems which make learning difficult. ResNet introduces 'skip connections,' which can solve this problem. As shown in figure below, Skip connections allow the network to learn the identity function by passing the input through the block, as shown in figure 2.3, without having to pass through other weight layers. This helps in building a very deep neural network by stacking up additional layers. During training,

the network can skip through layers that have less relevance. ResNet50 architecture consists of 48 convolutional layers, one max pool, and one average pooling layer. ResNets reduces the computational cost by a considerable margin and provides the benefit of depth at the same time.

2.2.4 MobileNetV2

MobileNetV2 [17] is a neural network architecture developed at Google, especially for machines with limited computing power. It is a state-of-the-art network used for image classification, object detection, and semantic segmentation. MobileNetV2 reduces the learnable parameters by a considerable margin, which makes them faster and easily trainable.

The state-of-the-art method shows a high performance due to the introduction of inverted residual and linear bottleneck layers. They use depth-wise separable convolutions introduced in the first version of MobileNet.

Standard Convolutions:

Consider the input tensor and the output tensor after some transformation of dimensions as shown below:

$$input : (D_u, D_u, M), output : (D_v, D_v, N)$$

A convolution operation transforms the input volume into the size of the output. As described in figure 2.4, a standard convolution [18] would use N filters for this transformation, each with a dimension of:

$$(D_r, D_r, M)$$

One way to convolve is to take a dot product, at every step, of the overlapping regions between the filter and the input volume. This results in a single value at each stage, as shown in the figure 2.5. However, spatial filtering gets obscured in this approach.

In another approach, using spatial filtering, the inner product is calculated concerning every channel. A filter slides along the spatial dimensions of the input volume rather than spanning across the depth. As seen in the figure 2.6, at every step, channel m along the

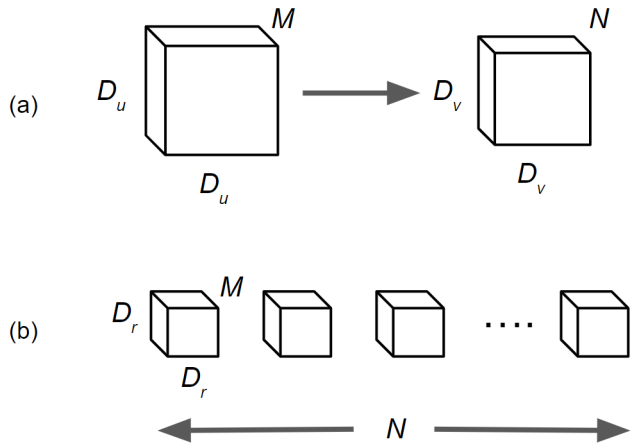

Figure 2.4. Transformations in a standard 2D convolution

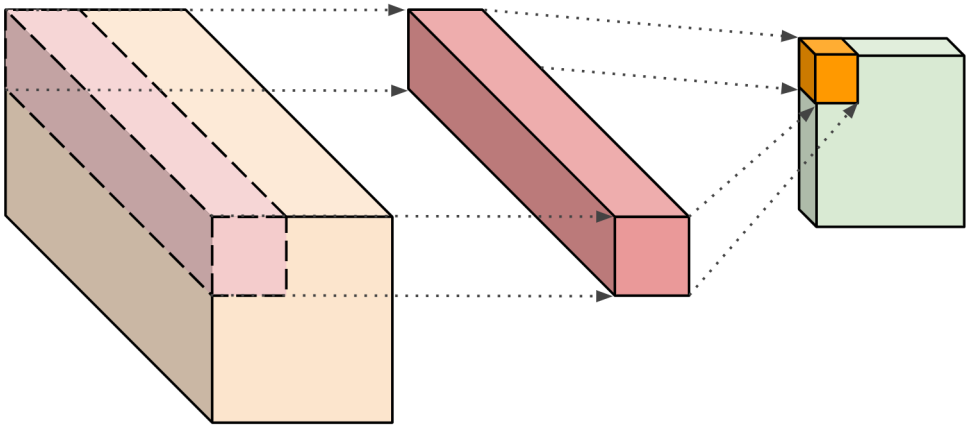

Figure 2.5. Interpretation of a 2D convolution

input volume is convolved with channel m in the filter. This approach takes care of the spatial filtering but misses out on the linear combinations obtained from the depth.

These reduce the computational cost by a considerable margin in comparison with standard convolutions. While a traditional convolutional method can contain spatial filtering and linear combinations, it is not possible to factorize the two stages. The depth-wise separable convolutions structures around these factorizations. The computational cost is in the order of:

$$D_v^2 D_r^2 MN$$

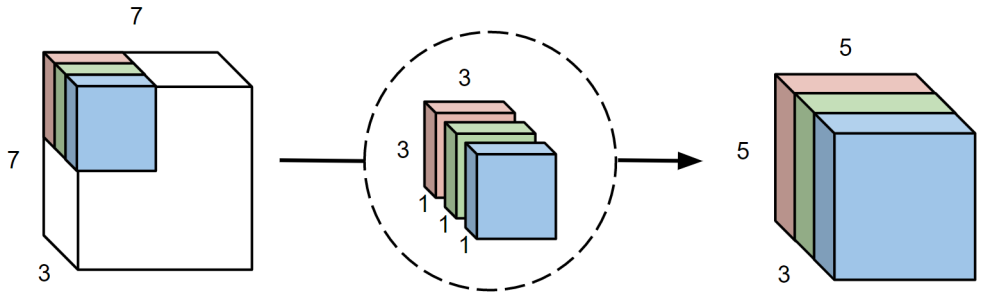

Figure 2.6. Convolution based on each channel

Depth-wise Separable Convolutions:

Depth-wise separable convolutions [19] reduce the computational cost in comparison to standard convolutions. They use factorized convolution to obtain fewer computations. In this case, the transformation undergoes in the manner shown in figure 2.7. Firstly, using depth-wise convolutions, M single-channel filters map the input volume on a per-channel basis to generate an intermediate tensor of volume

$$(D_v, D_v, M)$$

This step takes care of the spatial filtering component. Linear combinations could get captured through point-wise convolutions, using 1×1 filters along with the depth of the intermediate tensor. N 1×1 filters are required to obtain the output tensor. An example of these stages are shown in the figure 2.8.

This approach has the computational cost of

$$D_v^2 D_r^2 M + D_v^2 M N$$

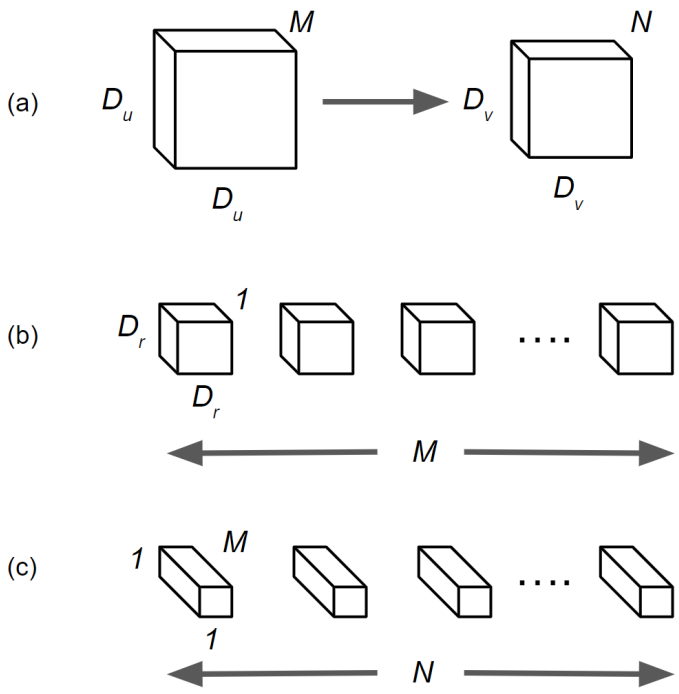

Figure 2.7. Transformations in a Depth-wise Separable Convolutions

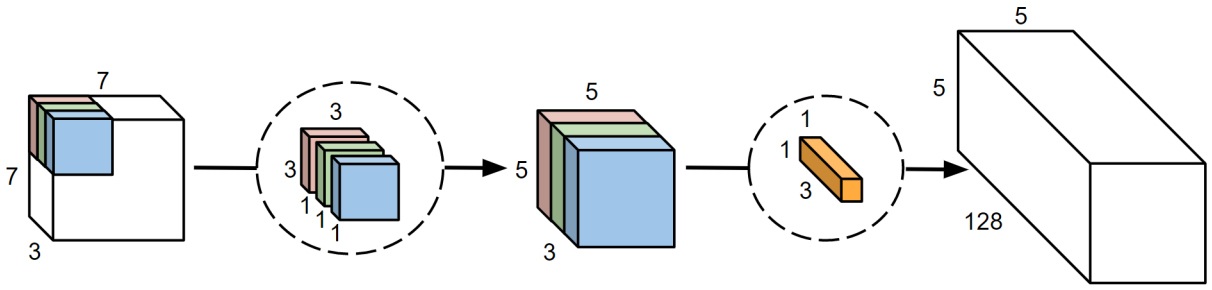

Figure 2.8. Example of a Depth-wise Separable Convolutions

The ratio between the computational cost of depth-wise separable convolutions and standard convolution is

$$\frac{1}{N} + \frac{1}{D_r^2}$$

Linear Bottleneck Layer:

MobileNetV2 has bottleneck layers with a linear activation. The idea is that data of interest could be represented in a tiny subspace. The dimensionality of a layer can be reduced to a low-dimensional subspace until we get the manifold of interest. However, if the features are already in a low dimension, using ReLU in the networks compresses a lot of information. The following two properties indicate this property:

(1) After the ReLU transformation, if the manifold of interest has a non-zero volume, it indicates a linear transformation.

(2) If the input manifold lies in a low-dimensional subspace, ReLU can preserve complete information.

Therefore, we can capture a low-dimensional manifold of interest by inserting linear bottleneck layers into the convolutional layers.

Inverted Residuals:

Standard residual block constitutes an input followed by bottleneck layers, which are then followed by expansion. The thick layers contain shortcuts between them. The bottleneck layers have all the relevant information, and the expansion layer can be seen as a non-linear transformation. MobileNetV2 takes advantage of this and uses shortcuts directly between the bottleneck layers. Here, ReLU is used as a non-linear activation for its robustness.

Architecture:

MobileNetV2 is a fully convolutional network, primarily built using the inverted residual networks called 'bottleneck.' In table 2.2, each line describes a sequence repeating for n times. The expansion factor t is applied to the input. The layers in each sequence have c output channels. The first layer in each sequence uses a stride s, and the rest use a stride of 1. All the spatial convolutional kernels are of size 3×3.

Table 2.2. MobileNetV2 Network Architecture

Input	Operator	Expansion Factor	Output Channels	Repetition	Stride
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1×1	-	1280	1	1
$7^2 \times 1280$	avgpool 7×7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1×1	-	k	-	

2.3 Related Works

With the achievements in object detection, we have seen astonishing results in pedestrian detection as well. A CNN model [20] developed by Sermanet, LeCun et al. in 2013 achieved state-of-the-art results in pedestrian detection across primary pedestrian benchmark datasets. In 2017, Du et al. proposed Fused DNN [21], a deep neural network architecture for fast detection. It uses a fusion of several architectures and processes them in a parallel manner for fast and robust results. Several state-of-the-art algorithms, like YOLO, SSD [22], and variants of R-CNN have been trained on benchmark datasets to detect pedestrians. In 2018, Ouyang, Zhou, et al. proposed a deep learning architecture to improve pedestrians' classification considering deformation and occlusion handling factors. Zhang et al. [23] focused on a similar problem by adding an attention network in a Faster R-CNN network.

The state-of-the-art methods have achieved high accuracy in predicting pedestrians which has solved a real world problem, like understanding the behavioral patterns of a person to plan the trajectory of autonomous vehicles [24] [25] and protect these vulnerable road users [26]. However, in a real scenario, people riding vehicles like bicycles, e-scooters, skateboards are also classified by these trained models as pedestrians. The typical behavior of these riders differs significantly from normal pedestrians, and it is important to detect whether the pedestrian is walking or riding a vehicle. To detect cyclists, Li et al. proposed a Fast R-CNN based detection framework [27] and a benchmark dataset [28] in 2016. In

2017, Saleh et al. proposed a Faster R-CNN based architecture for detecting and localizing the cyclists in depth images [29]. Foroozandeh, in 2017, proposed a detection and tracking system for analysis of cyclists in urban traffic [30]. While there has been some research to detect bicyclists, detection of e-scooter riders is yet to be explored.

3. RESEARCH PLAN

3.1 Research Gap

Today, no open-source detection models for real traffic scenarios can detect e-scooter riders. The existing open-source datasets also do not have e-scooter riders as a class. This gap makes it challenging to build a computer vision model to predict such riders. State-of-the-art datasets like COCO, Kitti [31], Cityscapes [32], BDD100k [33], and Google Open Images Dataset V4 [34] have a person as a class but not e-scooter or e-scooter riders.

To improve the safety of e-scooters, the autonomous driving community should study the behavior and movement of such vulnerable vehicles. Today, they can rely on existing models to detect every person in the image, including the e-scooter rider. But following that, they still have to manually identify the vulnerable riders among all persons, which is an expensive process.

3.2 Research Objectives

This work will eliminate the gap and develop a system that will help the autonomous driving community identify e-scooter riders automatically. Once the e-scooter rider is detected across frames, its behavior can be studied, and models can be built to predict future movements of these vulnerable vehicles. Furthermore, once the autonomous vehicle predicts the movement of an e-scooter rider, it can plan its route accordingly to eliminate the chances of accidents, thereby making the e-scooter riders safer around heavy vehicles.

3.3 Research Plan

Firstly, a dataset is built by collecting data around the city of Indianapolis. With six cameras mounted on top, an ego vehicle is driven around on multiple days to collect image data from different angles and distances. The data is collected from four different days for about an hour each day. Six cameras are operating at ten frames per second, which accumulates a mammoth amount of images.

Manually labeling bounding boxes around persons in all these images is an expensive and laborious task. Today, many existing computer vision models can predict people with high accuracy. These prediction models output both classification and localization details through bounding boxes. E-scooter riders are also predicted as belonging to class 'person' by these existing models. Since the existing models can also detect these riders, they can extract sections from images containing an e-scooter rider. Therefore, if all regions predicted as belonging to a 'person' are extracted, an image dataset can be generated for an image classification task. To create a final dataset, the images obtained through cropping out the image sections are manually separated into two categories: one belonging to an e-scooter rider and the other belonging to a normal person. The pipeline for this process is defined in figure 3.1.

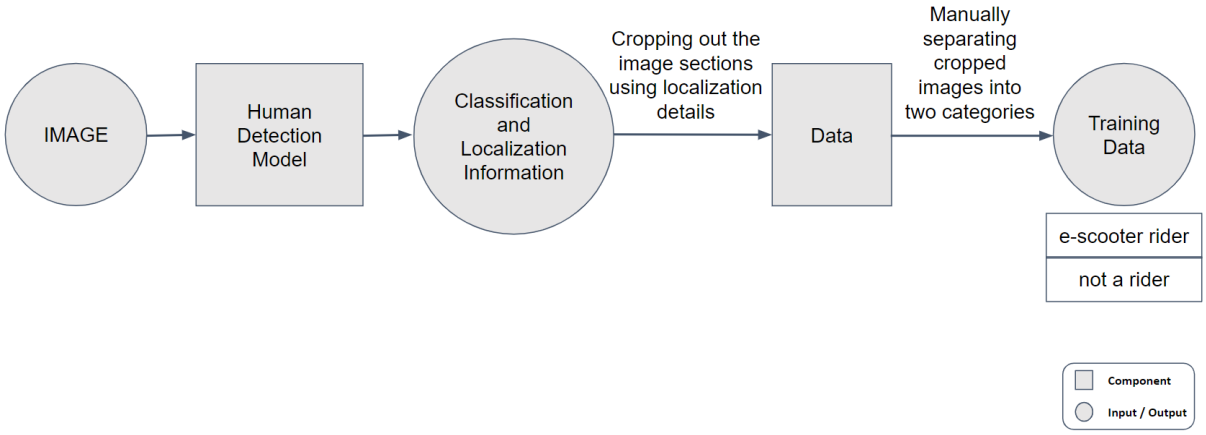

Figure 3.1. Plan for training data generation

Once the image dataset is generated, an image classification model will predict whether the extracted image belongs to an e-scooter rider or a random pedestrian. To train a model for such image classification tasks, CNNs will be used. A novel CNN architecture will be developed and trained using the dataset generated in the previous step. Multiple variations and hyperparameter optimization will be tried to obtain a CNN architecture that can perform most efficiently.

Due to a lack of computational resources, the training process can be expensive. If the model fails to output with higher accuracy, a transfer learning technique will be used for this

classification. Transfer learning is helpful in such cases because of the flexibility of neural networks (already trained on large open-source datasets like ImageNet) to adapt to similar datasets for an image classification task. Initially, the inner layers of these models will not be trained. However, if a scope of improvement with fine-tuning these inner layers is seen, then an extensive training process will be used to train a few hidden layers to achieve the desired performance. A simple pipeline of this process is described in figure 3.2.

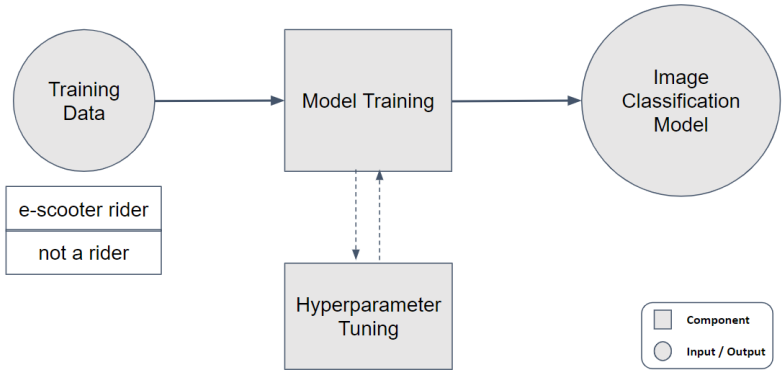

Figure 3.2. Plan for model training

After the prediction model is ready, a pipeline will be created by collating the base and trained models to find all e-scooter riders and other persons in an image. This pipeline, described in figure 3.3, will run across all images in the video input and generate video output with colored bounding boxes around people and e-scooter riders for visual understanding.

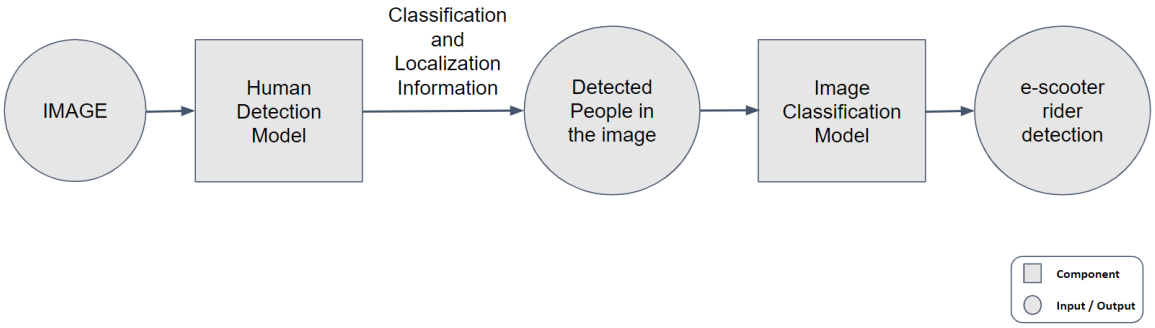

Figure 3.3. Final pipeline for e-scooter rider detection

4. SYSTEM PIPELINE

Some object detection models can classify and localize a person in an image with high accuracy. These models also detect e-scooter riders as 'person,' with the same accuracy. This is because an e-scooter hides only a tiny portion of the human body. In most cases, the human body is obvious and can be predicted by the existing detection algorithms as 'person.'

Since the existing object detection model can classify and localize a person riding an e-scooter as belonging to class 'person', these models can be used first to detect the e-scooter riders as a person. Using this detection information, an image classification model can predict which detections belong to an e-scooter rider. Using the prediction information from the image classification model for every localized object detected by a base object detection model, a pipeline is developed to detect e-scooter riders. The pipeline of the detection system can be seen in figure 4.1.

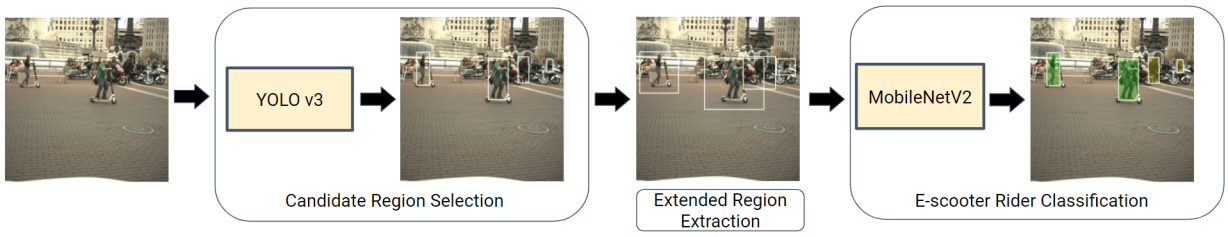

Figure 4.1. Model Pipeline

4.1 Candidate Region Selection

In this step, an existing object detection model is used to detect every person in the image. As discussed, the e-scooter riders will also be detected as 'person' by the model. YOLOv3 is used as the object detection model to output the required bounding boxes. YOLOv3 is a fast algorithm and uses a single neural network to perform classification and localization. With a pre-trained YOLOv3 model, the bounding boxes for people in the image is extracted as (x, y, w, h), where (x, y) is the image coordinate of the top left corner of the box, and w and h are the width and height of the box.

(a) Input image (b) Output of YOLOv3 model

Figure 4.2. Detecting people for Candidate Region Selection

YOLOv3 is trained on images of size $(416, 416, 3)$. So before passing the image into the model, the $(2048, 2048, 3)$ images are resized into an array of size $(416, 416, 3)$. YOLOv3 outputs the bounding box coordinates for this image in the format discussed above. If there are n people in an image, YOLOv3 is used to output an $(n, 4)$ dimensional vector representing the bounding box coordinates of each detection. The output of this stage can be seen in figure 4.2.

Let us consider the man in the front riding an e-scooter in this scene. His bounding box coordinates detected by YOLOv3 are $(1103, 513, 248, 514)$. Further steps will show how this information is used to predict that he is an e-scooter rider.

4.2 Extended Region Extraction

The bounding box coordinates obtained as output from YOLOv3 are processed again in this step. The bounding boxes obtained from a pre-trained classifier are strictly bound to a person. If the person is riding an e-scooter, the e-scooter pixels are not encaptured by the bounding boxes.

To include the e-scooter pixels as well, the bounding boxes are enlarged on three sides, right, left and bottom. These enlargements are done with respect to the size of each bounding box. The person can be driving in any direction, and therefore the e-scooter pixels can spread in any direction on the horizontal axis of the image. To account for this, the bounding box is increased by its width on both left and right sides, leading to three times the original width of the box. To include the pixels of the e-scooter's standing platform, the bounding box is enlarged on the bottom by 1.25 times of its original height. The enlarged bounding boxes can be seen in the figure 4.3. From observation, these dimensions ensure that the e-scooter pixels are captured along with the rider in every scenario.

Consider (x', y') as the coordinate of the top left corner, w' as the width, and h' as the height of the modified bounding box. The modified bounding box dimensions can be written as:

$$(x', y', w', h') = (x - w, y, 3w, h + h/4) \tag{4.1}$$

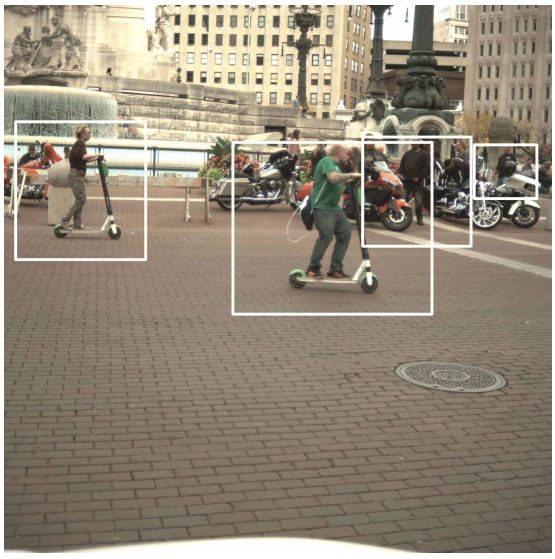

Figure 4.3. Enlarged bounding boxes to obtain extended region around each detection

So, after modifying n bounding boxes for an image, an $(n, 4)$ dimensional vector is obtained. This vector can then be used to extract corresponding regions from the image, which are called as *image segments*. The image segments are of various sizes depending on how far the person is in the image. For uniformity, each image segment is resized into a fixed height

and width, represented by (H_s). The size of each image segment after resizing becomes $(H_s, H_s, 3)$. Finally, a vector of size $(n, H_s, H_s, 3)$ is generated to represent n image sections in an image which is then passed into an image classification model for predictions.

After enlargement, the bounding box coordinates for the man riding the e-scooter becomes $(855, 513, 744, 642)$ based on equation 4.1. The calculations for this modification is shown below. The enlarged segment for this extracted region is shown in figure 4.4.

$$(1103 - 248, 513, 3 \times 248, 514 + 514/4) = (855, 513, 744, 642)$$

Figure 4.4. Extracted image segment for the considered example

4.3 Binary Image Classification

An image classifier is trained to predict whether the image segment has an e-scooter rider or not. In the step, the image segments extracted from the previous step, as a vector of size $(n, H_s, H_s, 3)$, is the input into the image classification model. A prediction vector of length n containing prediction scores is obtained as an output from the trained image classifier.

The image classification model predicts the example segment of the e-scooter rider, as shown in the figure 4.5. This information is further used with the localization information obtained from YOLOv3, to make the final detection of an e-scooter rider.

Several networks are trained, using Tensorflow [35], to build this classification model and are listed below. Each classifier learns to distinguish between e-scooter riders and other persons. Multiple iterations of hyperparameter tuning are used to figure out the best parameters for each classifier.

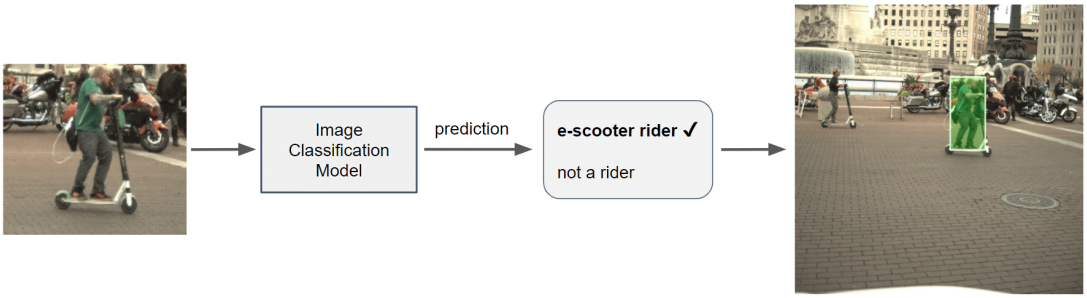

Figure 4.5. Role of the image classification model

4.3.1 Approach 1: A Novel CNN

To start with, a novel CNN architecture is designed and trained to differentiate between the e-scooter riders and other persons on the road. The architectural diagram of the proposed CNN is shown in the figure 4.6.

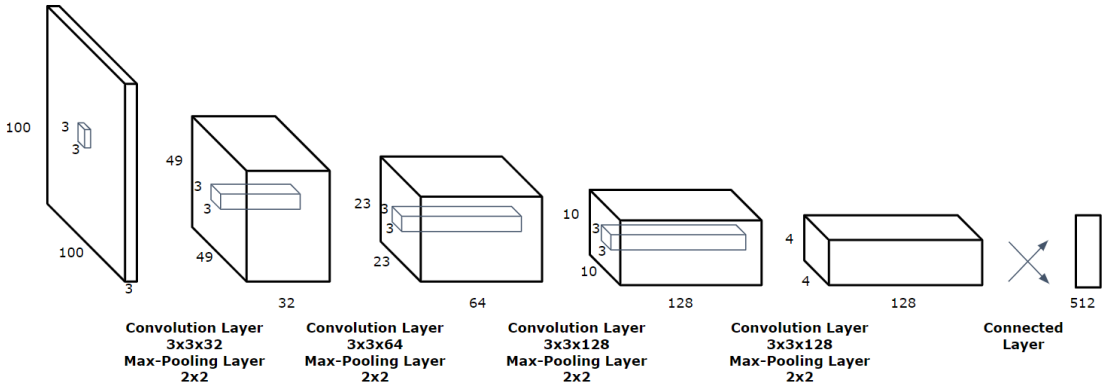

Figure 4.6. Proposed novel convolutional neural network

First, an image is reshaped into a $(100, 100, 3)$ vector which acts as an input layer to the neural network. Next, the input data is scaled to a range of $[0, 1]$. The input layer is followed by four consecutive convolutional and max-pooling layers. In all these layers, the kernel size is 3×3, and 'valid' padding is used. The pool size in max-pooling layers is $(2, 2)$. The stride is of dimensions $(1, 1)$, and 'relu' is used as an activation function.

The first convolutional layer has 32 filters and is followed by a max-pooling layer. This is followed by a second convolutional layer with 64 filters and then a max-pooling layer. Then, two consecutive convolutional layers with 128 filters and max-pooling layers follow. Following this, the layers are flattened and is passed into a dense layer with 512 units. 'Relu' is used as an activation function here as well. Finally, the network ends with a dense layer with a single unit, using 'sigmoid' as the activation function. The final layers outputs a value between $[0, 1]$, denoting non-riders and riders.

4.3.2 Approach 2: Frozen Weights from MobileNetV2

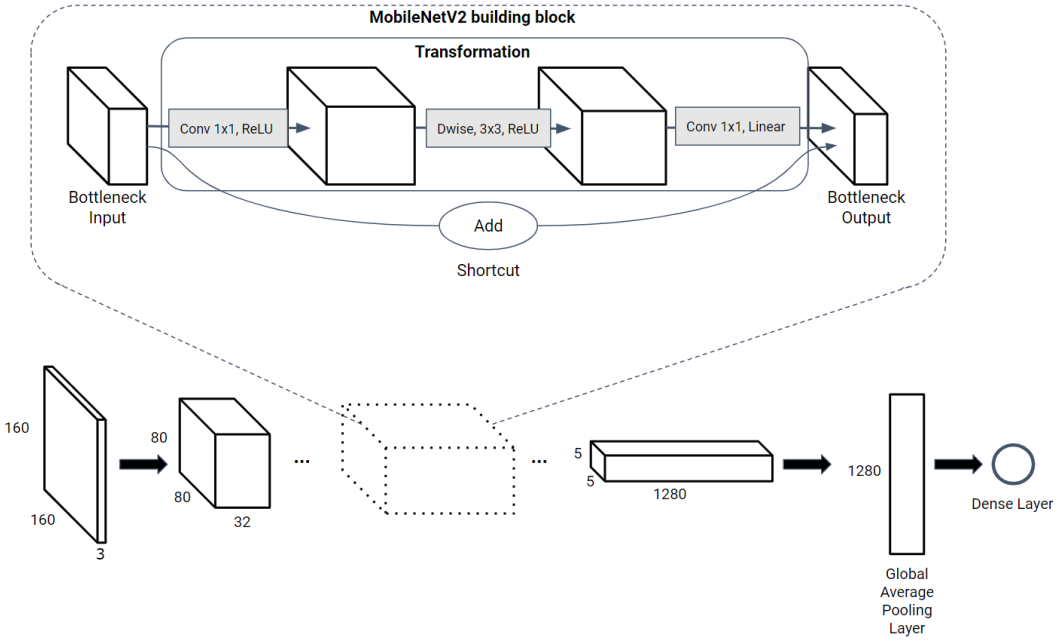

Figure 4.7. MobileNetV2 architecture for binary classification task

MobileNets are a family of neural networks designed by Google for mobile devices. They focus on using limited computational power and memory and provides state-of-the-art accuracy. In addition, MobileNets reduce the learnable parameters dramatically, making them more accessible and faster to train than other traditional networks. As a result, MobileNets are very efficient in image classification, segmentation, and object detection tasks. The

Google team has also made pre-trained weights available to the public. These pre-trained weights were obtained through training the network on the ImageNet dataset.

For this step, trained MobileNetV2 is used for the binary classification task. Google trained MobileNetV2 on input data of size $(160, 160, 3)$ scaled to $[-1, 1]$, and therefore the input data is preprocessed to match these requirements. Using the bottleneck layer, i.e., the last layer of MobileNetV2, the general features of the image segment are extracted as a $(5, 5, 1280)$ tensor. On top of this output vector, an average pooling operation is performed over the 5×5 spatial locations, and a flattened vector of size 1280 is obtained. These features are then passed into a dense layer trained to output the prediction value in the range of $[0, 1]$. Here, values closer to 0 predict non-riders, and values closer to 1 predict e-scooter riders. The final architecture is shown in the figure 4.7

4.3.3 Approach 3: Fine-Tuning MobileNetV2

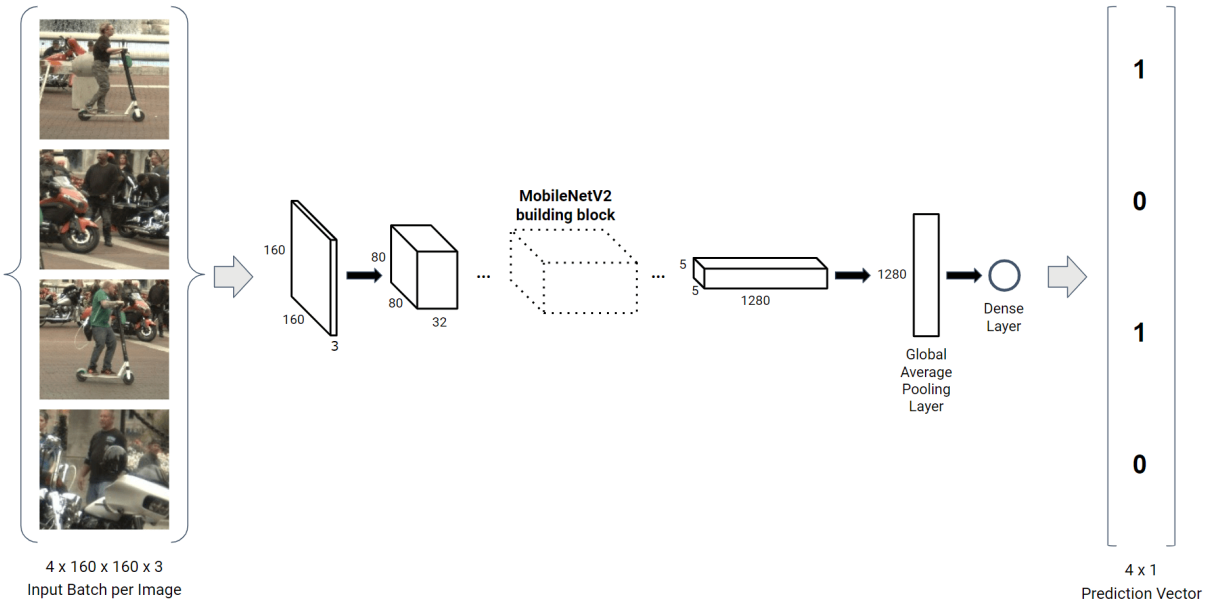

Figure 4.8. Prediction vector generation for the image segments

To improve the model performance further, fine-tuning over some of the top layers of MobileNetV2 is considered. Fine-tuning the layers will allow the updation of the weights during training. Bottom layers are not updated as they learn simple features like edges. The

final few layers of a model learns the complex functions and therefore they are updated to learn the features more specific to the e-scooter rider dataset.

The architecture of the classification model is shown in figure 4.8. The $(n, 160, 160, 3)$ tensor obtained from the previous step is passed as input to the the trained MobileNetV2 model. Again, using a single forward pass, MobileNetV2 outputs a $(n, 1)$ prediction vector containing values in the range $[0, 1]$.

4.4 Final Output of the Pipeline

For n persons predicted by YOLOv3, the image classification model outputs a $(n, 1)$ vector with values in range of $[0, 1]$. Based on a predefined threshold value, the prediction score outputted by the network can be used to make the final classification. For example, if the threshold value is considered at 0.5, all values above 0.5 is identified as an e-scooter rider, and vice versa. As shown in figure 4.9, the bounding boxes and classification for all persons in the scene are obtained. The bounding boxes for e-scooter riders are colored in green, and yellow is used for boxes belonging to other persons.

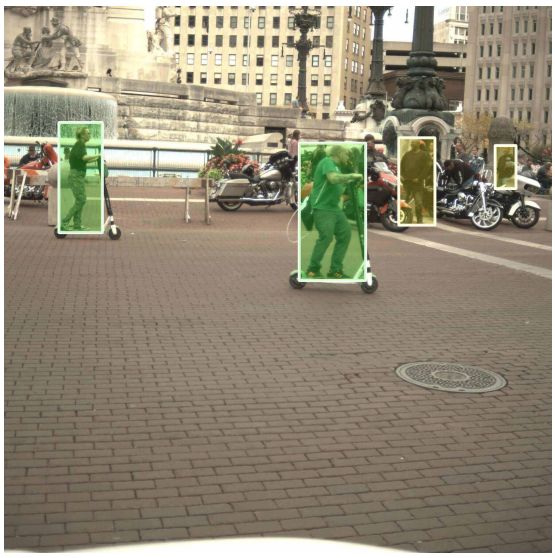

Figure 4.9. Final output of the model pipeline

5. DATASET CREATION AND MODEL TRAINING

5.1 Data Collection

An ego-vehicle shown in figure 5.1, with six cameras mounted on top, was used for the data collection. The data was collected around the city of Indianapolis, especially in the downtown area where people generally use e-scooter for short commute. The six cameras were placed in a way, shown in figure 5.2, that they collectively collect the entire scenario around the vehicle, and also had some overlap in their field of view. Three cameras faced the front side of the car to cover the front, front-left and front-right views of the vehicle. The remaining three cameras faced the rear side of the vehicle, with once facing the back, and the other two facing the back-left and back-right sections of the vehicle.

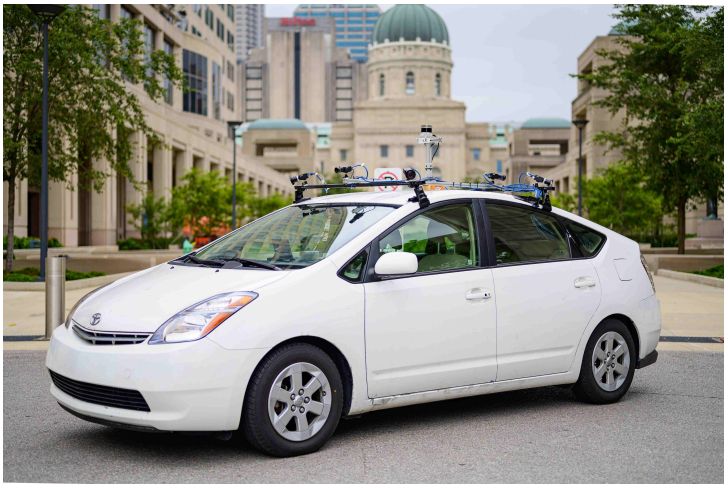

Figure 5.1. Ego vehicle used for data collection

Several scenarios were collected where an e-scooter interacted with the ego vehicle. Since the data collected are from 360 degrees, the ego vehicle could capture the entire interaction with an e-scooter rider. To build the dataset, 83 unique interactions were used which came from riders at varied angles and distances. Some of these interactions had multiple riders in the scene. These establish the data as a representative of the realistic scenarios.

Moreover, FLIR Grasshopper-3 cameras were used for the data collection. These cameras were operated at 10 fps images and collected high resolution images (2048×2048). The main advantage of using Grasshopper-3 is the reduced blur in images during fast motion of the

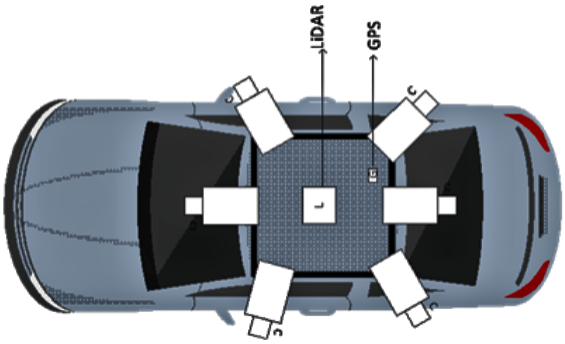

Figure 5.2. Top view of ego vehicle

vehicle, and its high exposure range which allows it to operate well in well illuminated and low light conditions.

5.2 Dataset Generation

From the data collected, the sections where the ego vehicle had an interaction with an e-scooter rider is separated out manually. The starting and ending timestamps of the interaction are used for this extraction. Using images from all six cameras, an e-scooter rider image dataset is generated. A total of 83 different interactions were considered for the dataset generation.

One way to generate the dataset would be to manually label the images with bounding boxes around the e-scooter riders and other people. But labeling is a long and expensive process. Today, many state-of-the-art algorithms have obtained good accuracy in classifying and localizing a person in an image. YOLOv3 is fast and can be used to extract bounding boxes for people in an image. Therefore, the labels for this dataset are obtained through a semi-automatic process discussed below.

Firstly, an image is inputted into the pre-trained YOLOv3 model to obtain the bounding boxes for people in the image. These bounding boxes are further modified to obtain a larger region around the person, as discussed in section 4.2. These modified bounding boxes are then used to extract the corresponding sections of the image, which are then stored as separate images in their original size in a folder. With the cameras running at 10 frames per second,

the image segments belonging to a unique person look very similar across consecutive frames which will bring redundancy in the dataset. To reduce this redundancy, two consecutive images are skipped every time. With this process, the image segments from both 'e-scooter rider' and 'not a rider' classes are obtained in the folder. Following this, the images from two classes are manually separated into two separate folders as shown in figures 5.3 and 5.4.

Figure 5.3. Training examples of e-scooter riders

Figure 5.4. Training examples of non riders

A total of 9180 image segments for 'e-scooter rider' and over 40,000 image segments from 'not a rider' classes are obtained. Due to this huge imbalance in the dataset, extra images for e-scooter riders are obtained through web mining. There does not exist a dedicated dataset for e-scooter riders yet, so only 228 unique images, shown in figure 5.5, could be collected. From these 228 images, 295 image segments of e-scooter riders could be extracted. With this, a total of 9475 images for e-scooter riders are collected. To generate a balanced dataset,

the ‘not a rider’ class is randomly downsampled to 9473 image segments. With this, a total of 18, 948 image sections from both classes are collected.

Figure 5.5. Data collection through web mining

5.3 Data Augmentation

Training deep learning models requires a large amount of training data. It is often difficult to collect such data to obtain high performance from the deep learning model. For training purposes, data augmentation [36] can take care of this problem up to a certain extent. The amount of training data can multiply by making slight modifications to the already existing data or creating synthetic data from the existing data.

Data augmentation can also act as a regularizer and be used to reduce overfitting during the training process. These transformations are generated using random horizontal and vertical flipping, height and width cropping, color modification, rotation, zooming in and out, noise injection, etc. Moreover, complicated mechanisms like Generative Adversarial Networks (GANs) [37] can create new and synthetic images for the training process.

Figure 5.6 shows examples of random data augmentation results obtained by flipping the images horizontally, rotating randomly, and randomly zooming.

Figure 5.6. Data augmentation obtained through random flip, rotation and zoom

5.4 Model Training

5.4.1 Splitting the dataset

To train a classifier to distinguish between e-scooter riders and other pedestrians, the dataset is further split [38] into train and test datasets. Out of the 83 e-scooter interactions coming from different days of data collection, the data collected on a single day is separated as test data. The test data contains 12 different e-scooter interactions, giving a total of 2497 image segments from both classes.

5.4.2 Activation Functions

The *sigmoid* activation function is used as the activation function in the output layer of the neural networks for the image classification model. Sigmoid outputs the value in range $(0, 1)$ and is helpful in cases where the probability is predicted as an output. The sigmoid

function is monotonic, but its derivative is not. Sigmoid also can cause a vanishing gradient problem which prevents deep models from training. The sigmoid is calculated as:

$$Sigmoid(x) = \frac{1}{1 + e^{-x}}$$

ReLu, or rectified linear activation function, is one of the most used activation functions for the hidden layers. It is less prone to the vanishing gradient problem. The ReLu function returns 0 for a negative input value and the value itself for a positive input value. It is calculated as:

$$ReLu(x) = max(0, x)$$

The plot of the activation functions [39] [40] used are shown in figures 5.7 and 5.8.

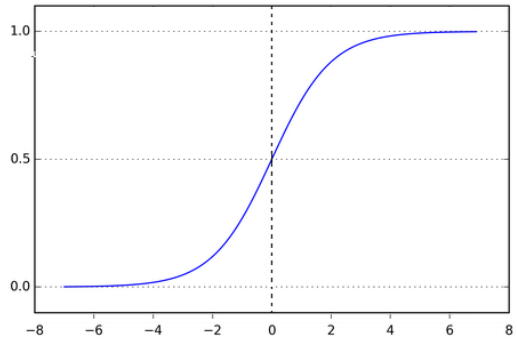

Figure 5.7. Sigmoid activation function

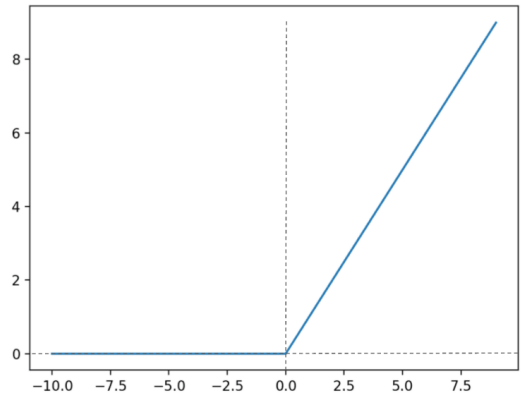

Figure 5.8. ReLu activation function

5.4.3 Loss Function

For binary classification problems, *Binary Cross Entropy* is used as the loss function. It compares the predicted values to actual labels, in this case, 0 or 1, and calculates the score. The score is then used to penalize the probabilities based on how close or how far the prediction and actual values are.

Binary cross-entropy [41] can be modeled as the equation below. Here, y is used to denote the class label for the observation, and p is the predicted probability for that observation.

$$Loss = -y \log(p) + (1 - y) \log(1 - p) \tag{5.1}$$

5.4.4 Optimization Algorithm

Adam [42] is an optimization method and an extension to stochastic gradient descent (SGD) [43]. It is used to optimize an objective function and is adopted in several machine learning applications in computer vision. Adam maintains a learning rate for each weight in the network, and each weight is separately adapted during the training process. This is different from stochastic gradient descent, which maintains a unique learning rate for all weights which does not change during the training process.

Adam is built on the benefits of two other extensions of SGD: Adaptive Gradient Algorithm (AdaGrad) [44] and Root Mean Square Propagation (RMSProp). AdaGrad uses a learning rate for every weight in the network, improving the performance of problems with sparse gradients. RMSProp also uses a learning rate for every weight, which adapts on the basis of how quickly these weights are changing. To evaluate the change in weights, the average of recent magnitudes of the gradients for the weight is calculated.

Adam calculates an exponential moving average of the gradient and the square gradient. It uses beta1 and beta2 parameters to control the decay rates of these moving averages.

5.4.5 Training Process

The three networks described in 4.3 are trained individually and evaluated to identify the best performing model. The learning curve from training these models is evaluated. Also, the performance of the model is tested on the test dataset of 2497 image segments. The results are described in chapter 6.

Training the Novel CNN Architecture:

The CNN architecture described in 4.6 is trained for this approach. The input data is rescaled to the range of $[0, 1]$ and passed into the network in a batch of 32. Using adam optimizer and binary cross entropy as the loss function, the network is trained for fifty epochs. Dropout layers at the rate of 0.2 are added after every convolutional and max-pooling layer and after the first dense layer of 512 units. Different data augmentation techniques, discussed in 5.3, are used to increase the training samples.

Training MobileNetV2 with Frozen Weights:

The input data is rescaled and resized into $(160, 160, 3)$ vectors. They are passed into the network during training in a batch of 32. Data augmentation techniques are used here to increase the volume and variance in the training data.

All the internal MobileNetV2 layers are kept frozen during training. Freezing the hidden layers prevents the weights in the layers from getting updated during the training process. After the global pooling layer, 1280 parameters are obtained. A dense layer with one node follows the global pooling layer, collectively amounting to 1281 total trainable parameters.

MobileNetV2 trains to output a value in the range of $[0, 1]$, as sigmoid is used as the activation function in the final dense layer. Adam [42] optimizer at the learning rate of 0.0001 is used, and dropout layers are added before the final dense layer at the rate of 0.3. This model trains for ten epochs, and the performance is noted.

Fine-Tuning MobileNetV2:

In this stage, some hidden layers of MobileNetV2 are also trained to improve performance. There are a total of 154 trainable layers in MobileNetV2. Different iterations for a number of frozen layers in the range of 125 to 150 are tried for fine-tuning. The model is trained for another 15 epochs after training with frozen weights for ten epochs in the previous step. Finally, their performances are compared with each other, and the best-performing model is selected. In this stage, adam optimizer is used at one-tenth of the initial learning rate (0.00001) as the model can easily overfit during fine-tuning.

6. RESULTS

6.1 Metrics

6.1.1 Confusion Matrix

Let "E-scooter rider" be a positive class and "Not an E-scooter rider" be a negative class. The prediction of e-scooter riders can be summarized using a 2×2 confusion matrix, shown in table 6.1, which uses four outcomes to demonstrate the performance.

True Positive (TP):

When the model correctly predicts that the input data belongs to the positive class. For example, the outcome is true positive when the model predicts an "e-scooter rider" as "e-scooter rider."

True Negative (TN):

When the model correctly predicts that the input data belongs to the negative class. For example, the outcome is true negative when the model predicts a "not an e-scooter rider" as "not an e-scooter rider."

False Positive (FP):

When the model incorrectly predicts that the input data belongs to the positive class. For example, the outcome is false negative when the model predicts "not an e-scooter rider" as an "e-scooter rider."

False Negative (FN):

When the model incorrectly predicts that the input data belongs to the negative class. For example, the outcome is false positive when the model predicts an "e-scooter rider" as "not an e-scooter rider."

Table 6.1. Confusion Matrix Representation

	E-scooter Rider (predicted)	Not an E-scooter Rider (predicted)
E-scooter rider (actual)	True Positive	False Negative
Not an E-scooter rider (actual)	False Positive	True Negative

6.1.2 Classification Report

Accuracy:

Accuracy of a model can be seen as the fraction of total predictions it predicted right. In terms of positives and negatives, it can be calculated as:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

However, accuracy does not tell the whole story, especially if a class-imbalance exists in the test dataset.

Precision:

The precision value aims at answering the question, *"What percentage of positive predictions were actually correct?"*

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

Recall:

The recall value aims at answering the question, *"What percentage of actual positives were identified correctly?"*

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

F1 score:

F-measure or F1 score is the harmonic mean of precision and recall. A good model has high values for both precision and recall. With the increase in precision, the model loses its advantage on recall and vice versa. Therefore, to balance precision and recall, a high F1 score is aimed for a prediction model.

$$F\text{-}measure = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

Area Under the ROC Curve (AUC):

ROC (receiver operating characteristic curve) is a graph to show the performance of a classification model. Two parameters mentioned below are used to plot this curve:

True Positive Rate (TPR): TPR is the same as recall. It is also called *Sensitivity*, defined as:

$$TPR = \frac{TP}{TP + FN}$$

False Positive Rate (FPR): FPR is also called as *Specificity*, defined as:

$$FPR = \frac{FP}{FP + TN}$$

The ROC curve plots TPR on the Y-axis versus FPR on the X-axis at different classification thresholds plotted from $(0, 0)$ to $(1, 1)$. With a lower classification threshold, the model will classify more items as positive, thereby increasing the false positives and true positives.

AUC measures the two-dimensional area underneath the ROC Curve. Intuitively, AUC can be seen as the probability of the model ranking a random positive example more highly than a negative one. AUC score ranges in the range $[0, 1]$. A model making more accurate predictions has an AUC close to 1.

6.2 Analysis of trained neural networks

6.2.1 Performance of the Novel CNN Architecture

The model is trained for fifty epochs. As seen in figure 6.1, the accuracy curve starts flattening after around ten epochs. After several hyperparameter optimizations and data augmentation to improve the model, accuracy of around 85% on the test dataset is achieved. The classification report of this model's performance is shown in table 6.2.

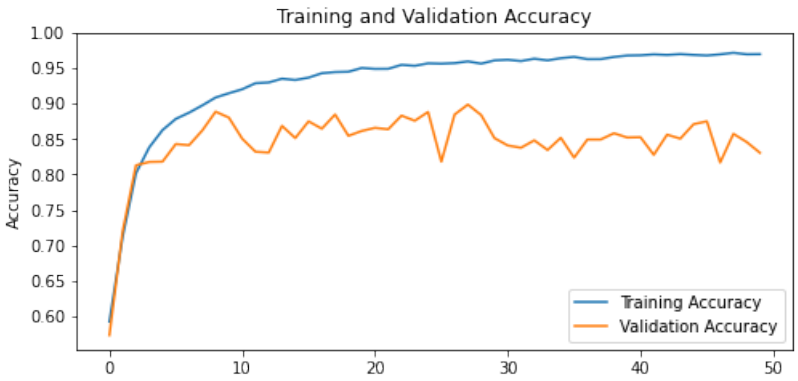

Figure 6.1. Learning curve of proposed CNN on test dataset

Table 6.2. Classification Report of Novel CNN

Class Label	Precision	Recall	F1-score	Support
E-scooter rider	0.90	0.79	0.84	1269
Non rider	0.81	0.91	0.85	1228
		Accuracy	0.85	

The confusion matrix of this model's predictions is shown in table 6.3.

Table 6.3. Confusion Matrix of Novel CNN

	E-scooter Rider (predicted)	Non Rider (predicted)
E-scooter rider (actual)	1002	267
Non Rider (actual)	115	1113

An ROC-AUC score, or the area under the ROC curve, obtained is 0.89, as shown in figure 6.2.

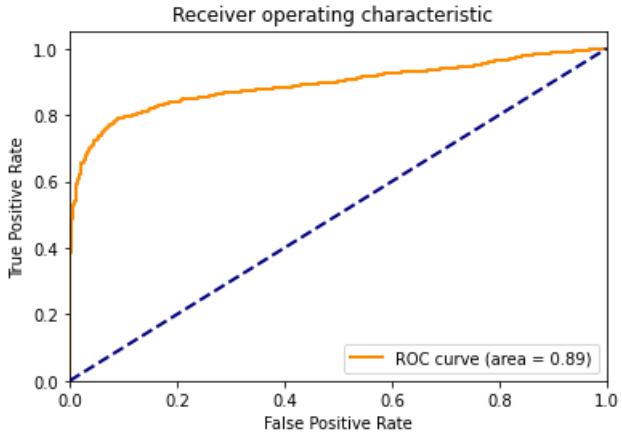

Figure 6.2. ROC-AUC curve: Proposed CNN

6.2.2 Performance of MobileNetV2 with frozen weights

Initially, the convolutional base of MobileNetV2 is kept frozen while training. Freezing the hidden layers prevents the weights in the layers from getting updated during the training process. Adam [42] optimizer at the learning rate of 0.0001 is used and dropout layers are added before the final dense layer at the rate of 0.3. As seen in figure 6.3, training this model for ten epochs already gave us an accuracy of 84% on the test data.

The classification report and confusion matrix of this model's performance is shown in table 6.4 and table 6.5 respectively.

Table 6.4. Classification Report: Trained MobileNetV2 with Frozen Weights

Class Label	Precision	Recall	F1-score	Support
E-scooter rider	0.81	0.89	0.85	1269
Non rider	0.88	0.78	0.83	1228
		Accuracy	0.84	

The area under the ROC curve is 0.93. The graph is shown in figure 6.4.

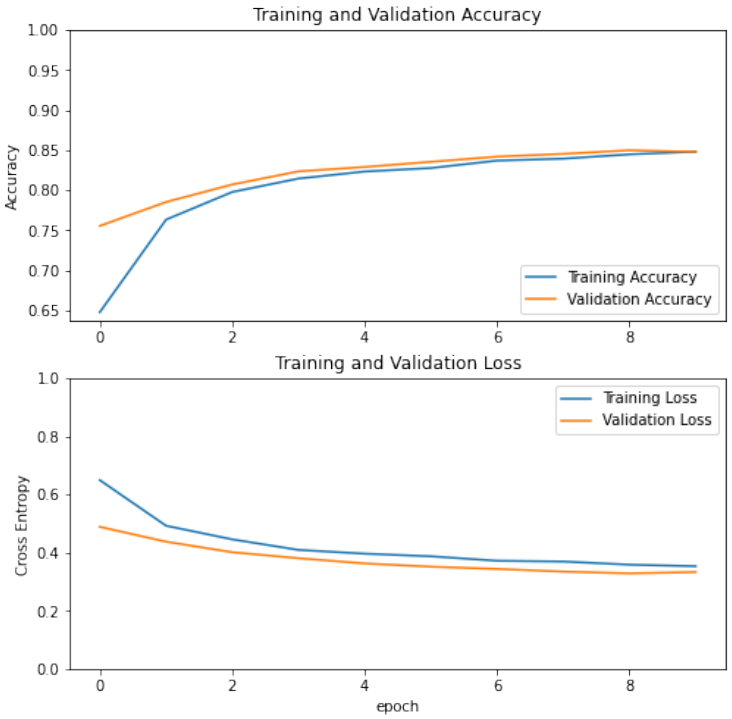

Figure 6.3. Learning curve of MobileNetV2 with frozen weights

Table 6.5. Confusion Matrix: Trained MobileNetV2 with Frozen Weights

	E-scooter Rider (prediction)	Non Rider (prediction)
E-scooter rider (actual)	1135	134
Non Rider (actual)	271	957

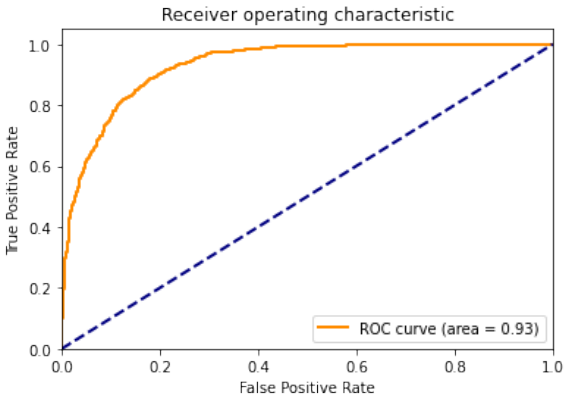

Figure 6.4. ROC-AUC curve: MobileNetV2 with frozen weights

6.2.3 Performance after Fine-Tuning MobileNetV2

There are a total of 154 trainable layers in MobileNetV2. Initially, first 150 layers are kept frozen, and later the number of frozen layers are decreased over several iterations. The model is trained for another 15 epochs using Adam optimizer, at one-tenth of the initial learning rate (0.00001). This is done because the network is at the risk of easily overfitting during fine tuning. Several iterations of hyperparameter tuning are tried to find the best hyperparameter tuple.

Hyperparameter Optimization:

The hyperparameter values can be adjusted to control the training process. With hyperparameter tuning, the optimal tuple of hyperparameters are obtained. This tuple can minimize the loss function on the dataset.

In Table 6.6, some random results obtained from particular values for several hyperparameters are shown. The hyperparameters considered for fine-tuning are:

- Number of frozen layers
- Dropout rate
- Epochs

For each iteration, the model is trained for ten epochs using an Adam optimizer at the initial learning rate of 0.0001. After fine-tuning is enabled, i.e., after ten epochs, the learning rate is reduced to one-tenth of the original value.

Table 6.6. Model Performance on Different Hyperparameter Values

	Total frozen layers	Dropout rate	Epochs	Training Accuracy	Validation Accuracy
1	125	0.2	50	99.03	90.23
2	151	0.2	50	98.70	90.63
3	145	0.3	40	97.90	91.13
4	140	0.3	25	93.50	90.83
5	**130**	**0.3**	**25**	**98.70**	**92.70**

In table 6.6, the best test accuracy was obtained with hyperparameter entries in the fifth row. The learning curve of the model with these hyperparameters is shown in figure 6.5. It is seen that the model performance on validation set starts saturating after around fifteen total epochs. In a scenario like this, while training a large network, the network stops generalizing and starts to learn several statistical noise in the training dataset.

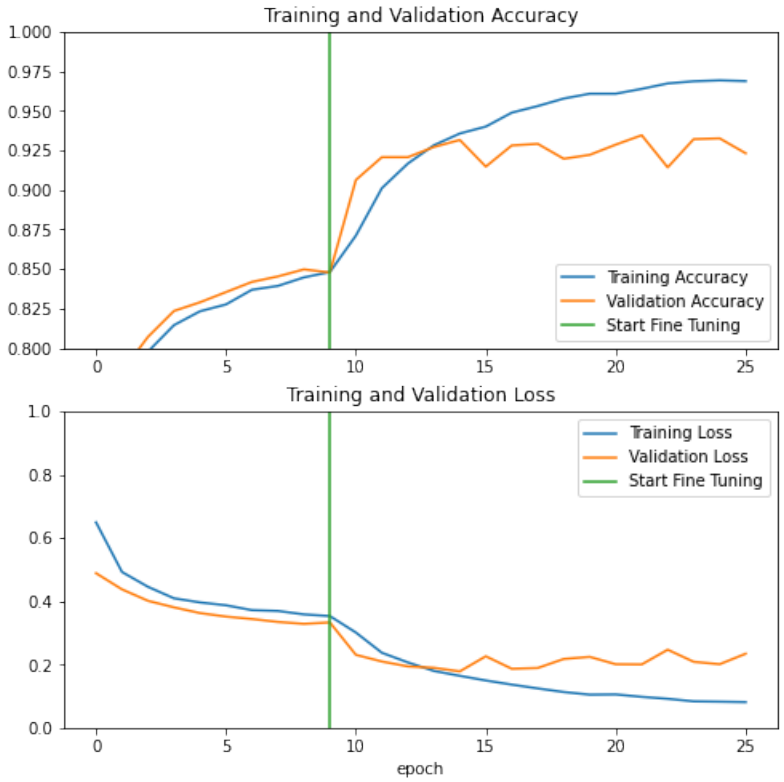

Figure 6.5. Learning curve of MobileNetV2 after fine-tuning

The classification report of the performance of fine-tuned MobileNetV2 is shown in table 6.7. The confusion matrix of the predictions of this model is in table 6.8.

Table 6.7. Classification Report: Fine-Tuning MobileNetV2

Class Label	Precision	Recall	F1-score	Support
E-scooter rider	0.96	0.83	0.89	1269
Non rider	0.84	0.97	0.90	1228
		Accuracy	0.90	

Table 6.8. Confusion Matrix: Fine-Tuning MobileNetV2

	E-scooter Rider (prediction)	Non Rider (prediction)
E-scooter rider (actual)	1049	220
Non Rider (actual)	39	1189

An ROC-AUC score after fine tuning MobileNetV2 is 0.97, and the curve can be seen in figure 6.6.

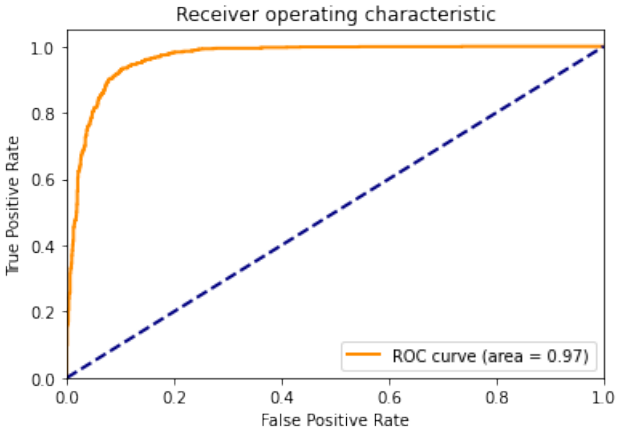

Figure 6.6. ROC-AUC Curve: Fine-Tuning MobileNetV2

6.2.4 Picking the Best Performing Classification Model

Table 6.9. Comparison of Models in Predicting E-Scooter Riders

	Novel CNN	MobileNetV2 (Frozen Weights)	MobileNetV2 (Fine-Tuning)
Precision	0.90	0.81	0.96
Recall	0.79	0.89	0.83
F-1 Score	0.84	0.85	0.89
AUC-ROC Score	0.89	0.93	0.97

From the table 6.9, it can be said that MobileNetV2 after fine-tuning gives the best classification performance. It has the best precision and f-1 score, and the best AUC-ROC score.

For binary classification problems, the ROC curve is an excellent evaluation metric. This probability curve plots TPR on the Y-axis and FPR on the X-axis. The area under the ROC curve (AUC) measures the classifier's prediction ability, and its values range from $[0, 1]$. As the AUC increases, the performance of the model at distinguishing between the two classes becomes better. If AUC is equal to 1, the classifier can perfectly distinguish between two types. So the more significant the AUC value, the better is the overall performance of the model. With a greater AUC, the classifier can detect more true positives and true negatives than false positives and false negatives.

The fifth entry in the table 6.6 gives the best performance amongst all. As evident from several learning graphs, model's performance on the validation data starts saturating around fifteen total epochs (the first ten epochs are with frozen weights, and the following epochs are by tuning weights of some of the final layers). Beyond this point, the network might start learning the noise in the training dataset which can cause overfitting.

To reduce overfitting and generalization error, *early stopping* mechanism is used. Early stopping is a regularization technique that allows a model to train long enough so that it learns to map the input to output, at the same time stopping it from overfitting to the training data. Using the parameters mentioned in fifth entry in table 6.6, the model was trained again but with an early stopping mechanism. The model in this run, stopped after sixteen epochs and obtained a test accuracy of 93.21%, as shown in table 6.10. The learning graph of training process is shown in 6.7.

Table 6.10. Model Performance on Best Hyperparameter Tuple with Early Stopping

Total frozen layers	Dropout rate	Epochs	Training Accuracy	Validation Accuracy
130	0.3	16	94.03	93.21

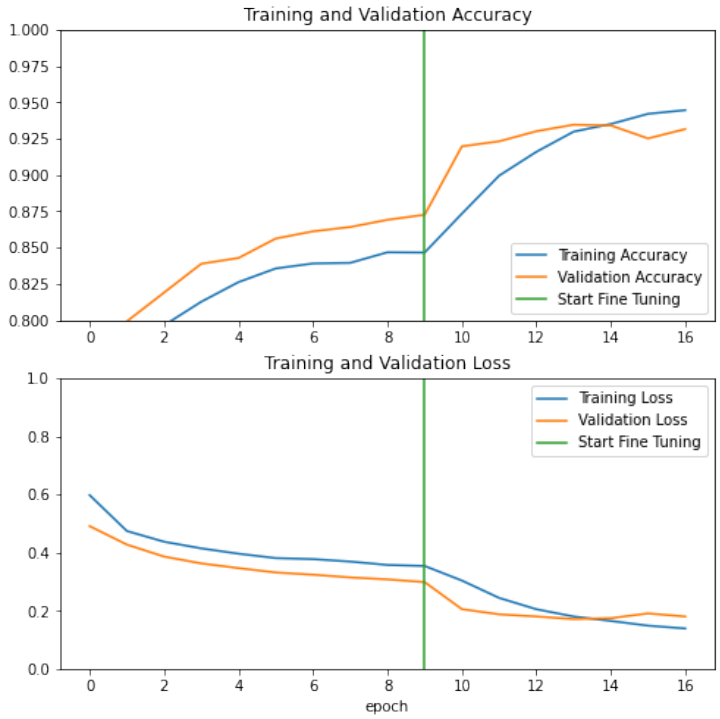

Figure 6.7. Learning graph after early stopping

6.3 Binary Classification Performance on Test Dataset

As mentioned in section 5.4, the test dataset consists of 2497 image segments (cropped images). 1269 of those images belong to the e-scooter rider class, and 1228 are from the non-rider class. The best classifier obtained from the previous step is tested with these image segments, and the results are recorded.

The MobileNetV2 classifier outputs a value between 0 and 1 for input. These values are interpreted as confidence scores for the model's output. If the model outputs a value of 0.9 and classifies the input as an e-scooter rider, then it can be said that the model is 90% confident that the input image belongs to the e-scooter rider class. A confidence threshold can be used as a measure to classify the inputs into respective categories.

The performance of the model, with a confidence score of 0.5, is shown in Table 6.11. Out of 1269 image segments of e-scooter riders, the model predicts 1083 of them correctly and predicts 54 non-riders as e-scooter riders. The f1-scores of the model in predicting e-scooter riders and non-riders are around 0.9. The ROC-AUC score for the same is 0.90.

Table 6.11. Binary Classification Performance; Confidence threshold = 50%

Class Label	Precision	Recall	F-measure	Support
E-scooter rider	0.95	0.85	0.90	1269
Non rider	0.86	0.96	0.91	1228
		Accuracy	0.90	

Different threshold values are tried, and the one with the highest ROC-AUC score is considered the optimal threshold value to classify e-scooter riders. The optimal performance is achieved at a threshold of 60%, i.e., if the model outputs a value greater than 0.6, the input is considered to have an e-scooter rider. The classification report for this performance is shown in Table 6.12. The model classifies e-scooter riders as positive for 1107 cases and wrongly predicts 71 non-riders as e-scooter riders. The f-1 score of the model at this confidence threshold is also around 0.9, and the ROC-AUC score is 0.91.

Table 6.12. Binary Classification Performance; Confidence threshold = 60%

Class Label	Precision	Recall	F-measure	Support
E-scooter rider	0.94	0.87	0.90	1269
Non rider	0.88	0.94	0.91	1228
		Accuracy	0.91	

6.4 Performance of the Entire Pipeline on the Test Sample

As mentioned in 5.4, twelve scenarios are initially separated to generate the test dataset. These scenarios are used again to create a *test sample* of 300 original images. These images contain feed from all cameras, and 276 of these images have at least one person present. Out of these 276 images, 121 of them have an e-scooter rider present. Following this, the entire detection pipeline is tested on the test sample, and the results are recorded.

The number of e-scooter riders and other people is manually counted, and ground truth is determined. 1156 different people are recorded from 276 images, giving an average of 4 people per image. In candidate region selection, YOLOv3 overall detects 762 out of 1156

instances as 'person' at a confidence threshold of 60%. YOLOv3 misses 394 cases, and so the pipeline ignored these too. In these 394 missed instances, 31 belonged to e-scooter riders. In total, 140 e-scooter riders were detected by YOLOv3.

Table 6.13. Performance of Pipeline against Ground Truth

Outcome	True Positive	False Positive	False Negative	Precision	Recall
E-scooter rider	127	6	44	0.95	0.74

As mentioned in Table 6.13, the pipeline detected 127 e-scooter riders correctly from 171 total riders. There are 13 instances wrongly classified as non-riders, and 31 are undetected, summing up to 44 cases of false negatives. Thus, the pipeline has a recall of 0.74 in detecting e-scooter riders. The default performance of YOLOv3 contributes to a lower recall value for the entire system.

6.5 Analysis of Trained Image Classifier on the Test Sample

Table 6.14. Performance Analysis of Trained MobileNetV2 only

Class Label	Precision	Recall	F-measure
E-scooter rider	0.95	0.91	0.93
Non rider	0.98	0.99	0.98
		Accuracy	0.975

As mentioned above, YOLOv3 could detect 762 out of 1156 people across 300 test samples. The performance of the trained MobileNetV2 classifier is analyzed further on these 762 instances. Out of 762 people, there are 140 e-scooter riders, and 622 are non-riders. The classifier correctly predicts 127 e-scooter riders out of 140. From 622 non-riders, the model correctly predicts 616 of them. The classification report is described in Table 6.14. The results show that the model has precision, recall, and f1-score of over 0.9 for both classes. The overall accuracy measure is 0.975 for the model. The results show that the trained classifier can distinguish between the two classes very well.

6.6 Wrong Predictions

The examples shown in the figures below are from the test dataset on which the trained model is evaluated. Some false negatives and false positives examples are shown.

6.6.1 False Negatives

Figure 6.8 shows some cases in which the model wrongly predicted an e-scooter rider as a non-rider. In some cases, the e-scooter is obstructed by other objects, making it difficult for the model to identify the correct patterns. The model finds it especially difficult in issues where the e-scooter rider is facing directly away from the camera's point of view, as only a portion of the rear wheel of the e-scooter is visible. In some cases, the e-scooter riders are very far in the scene, making the image segments blurry, which might further contribute to failure in identifying patterns.

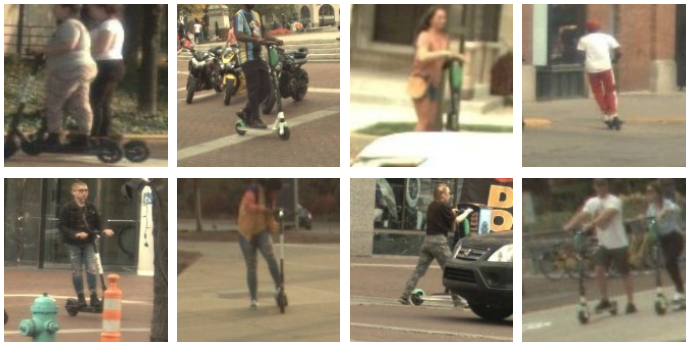

Figure 6.8. Examples of false negative predictions

6.6.2 False Positives

Figure 6.9 shows cases in which a non-e-scooter rider was classified as an e-scooter rider. Some cyclists and, in one instance, an electric segway rider is classified as an e-scooter rider. In addition, there are some special cases in which a person is not riding, but an e-scooter is present in the image section, which occurs due to enlargement of the bounding box region. The model classifies these cases into e-scooter riders.

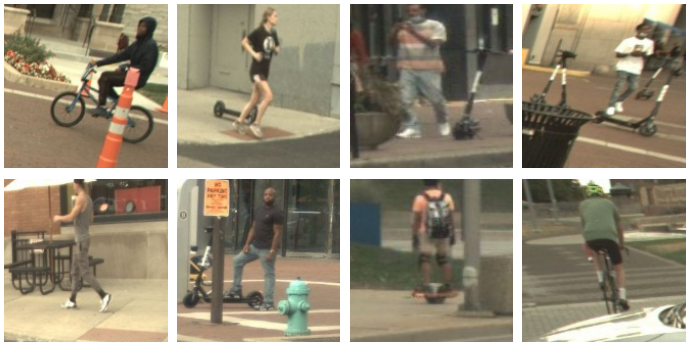

Figure 6.9. Examples of false positive predictions

6.7 Visual Output of the Model Pipeline

The detected bounding boxes and classification results are displayed on the input image to generate a final output for visual understanding. The bounding boxes obtained from YOLOv3 model are drawn on the input image. The prediction values obtained for each of these bounding boxes from MobileNetV2, are used to color these bounding boxes with a specific color to denote their classification. The bounding box around e-scooter riders are painted in green, and the bounding boxes around other people are painted in yellow. Some of the outputs from the model pipeline are shown in Fig. 6.10.

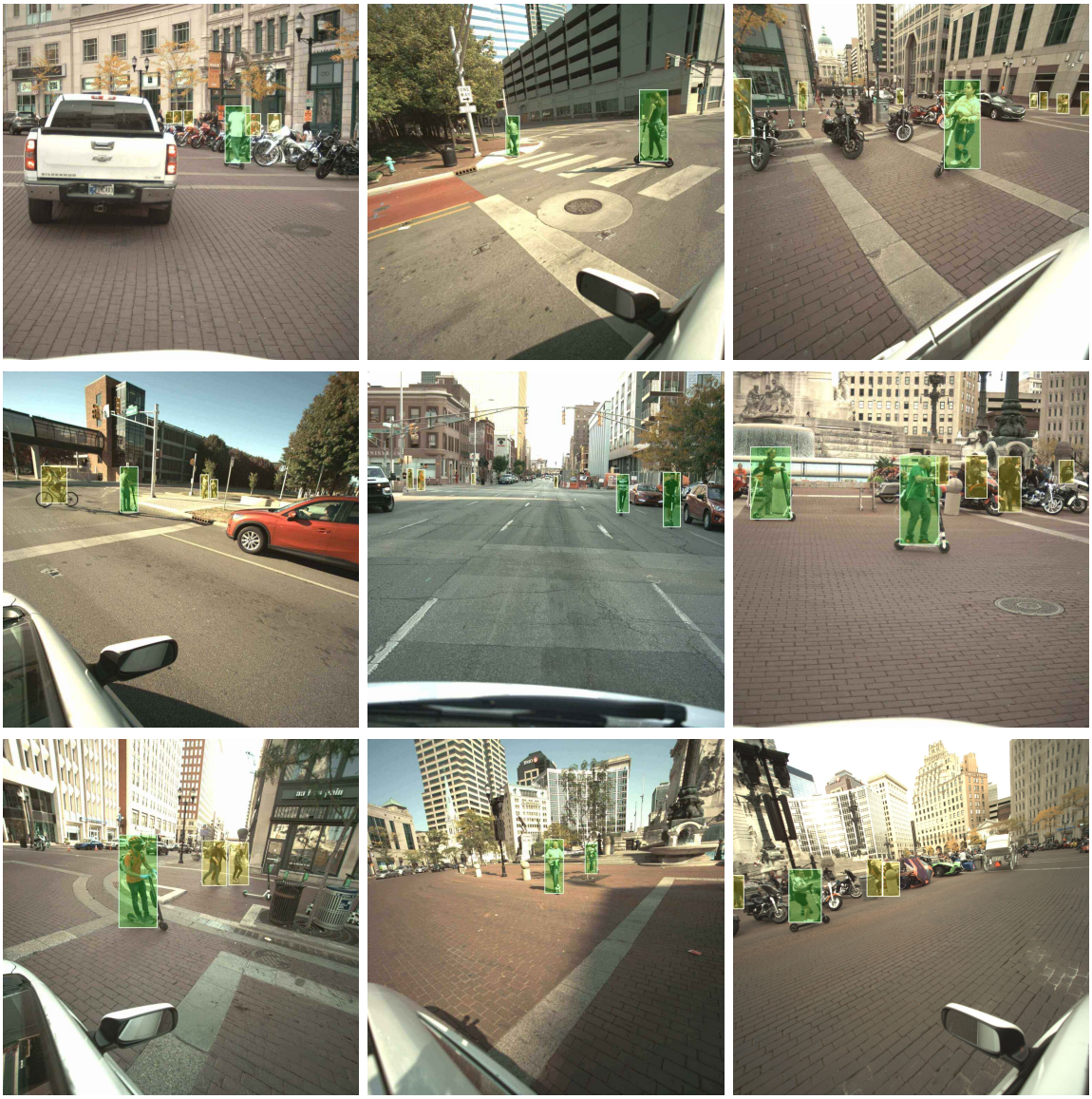

Figure 6.10. Model performance on raw data

7. CONCLUSION

This thesis proposes a classification approach and introduces a novel benchmark dataset for e-scooter riders in natural traffic scenes. An existing classifier, YOLOv3, is used to detect people, and MobileNetV2 is trained to distinguish e-scooter riders. The proposed system can differentiate between an e-scooter rider and any other pedestrians on the road with this vision approach. The system produces highly accurate results with precision and recall of the classifier over 0.9. The autonomous driving community can study and understand e-scooter riders' behavior with this detection system.

To improve the performance further, training using more data from the cases involving overlapping of e-scooter riders and pedestrians can be done. When the e-scooter rider is far and facing away from the camera, the body of the e-scooter hides, making it difficult for the model to classify correctly across frames. Particular cases like these can be studied to improve performance. Moreover, a single neural network can be built for a multiclass classification system to identify various vulnerable road users who use other vehicles in future works.

REFERENCES

[1] M. M. Baqer, *On the reliability of vanet safety applications for diverse traffic participants*, 2020.

[2] N. K. Namiri, H. Lui, T. Tangney, I. E. Allen, A. J. Cohen, and B. N. Breyer, "Electric scooter injuries and hospital admissions in the united states, 2014-2018," *JAMA surgery*, vol. 155, no. 4, pp. 357–359, 2020.

[3] K. Shankari, *e-mission: an open source, extensible platform for human mobility systems*. eScholarship, University of California, 2019.

[4] X. Wu, D. Sahoo, and S. C. Hoi, "Recent advances in deep learning for object detection," *Neurocomputing*, vol. 396, pp. 39–64, 2020.

[5] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*, Ieee, 2009, pp. 248–255.

[6] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, pp. 1097–1105, 2012.

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

[8] R. Girshick, "Fast r-cnn," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Dec. 2015.

[9] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *arXiv preprint arXiv:1506.01497*, 2015.

[10] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[11] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.

[12] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *European conference on computer vision*, Springer, 2014, pp. 740–755.

[13] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on knowledge and data engineering*, vol. 22, no. 10, pp. 1345–1359, 2009.

[14] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[15] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[16] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[17] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4510–4520.

[18] V. Dumoulin and F. Visin, "A guide to convolution arithmetic for deep learning," *arXiv preprint arXiv:1603.07285*, 2016.

[19] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[20] P. Sermanet, K. Kavukcuoglu, S. Chintala, and Y. LeCun, "Pedestrian detection with unsupervised multi-stage feature learning," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2013, pp. 3626–3633.

[21] X. Du, M. El-Khamy, J. Lee, and L. Davis, "Fused dnn: A deep neural network fusion approach to fast and robust pedestrian detection," in *2017 IEEE winter conference on applications of computer vision (WACV)*, IEEE, 2017, pp. 953–961.

[22] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *European conference on computer vision*, Springer, 2016, pp. 21–37.

[23] S. Zhang, J. Yang, and B. Schiele, "Occluded pedestrian detection through guided attention in cnns," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 6995–7003.

[24] J. Wei, J. M. Snider, T. Gu, J. M. Dolan, and B. Litkouhi, "A behavioral planning framework for autonomous driving," in *2014 IEEE Intelligent Vehicles Symposium Proceedings*, IEEE, 2014, pp. 458–464.

[25] M. Ilievski, S. Sedwards, A. Gaurav, A. Balakrishnan, A. Sarkar, J. Lee, F. Bouchard, R. De Iaco, and K. Czarnecki, "Design space of behaviour planning for autonomous driving," *arXiv preprint arXiv:1908.07931*, 2019.

[26] W. Tabone, J. de Winter, C. Ackermann, J. Bärgman, M. Baumann, S. Deb, C. Emmenegger, A. Habibovic, M. Hagenzieker, P. Hancock, *et al.*, "Vulnerable road users and the coming wave of automated vehicles: Expert perspectives," *Transportation Research Interdisciplinary Perspectives*, vol. 9, p. 100 293, 2021.

[27] X. Li, L. Li, F. Flohr, J. Wang, H. Xiong, M. Bernhard, S. Pan, D. M. Gavrila, and K. Li, "A unified framework for concurrent pedestrian and cyclist detection," *IEEE transactions on intelligent transportation systems*, vol. 18, no. 2, pp. 269–281, 2016.

[28] X. Li, F. Flohr, Y. Yang, H. Xiong, M. Braun, S. Pan, K. Li, and D. M. Gavrila, "A new benchmark for vision-based cyclist detection," in *2016 IEEE Intelligent Vehicles Symposium (IV)*, IEEE, 2016, pp. 1028–1033.

[29] K. Saleh, M. Hossny, A. Hossny, and S. Nahavandi, "Cyclist detection in lidar scans using faster r-cnn and synthetic depth images," in *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, IEEE, 2017, pp. 1–6.

[30] F. Foroozandeh Shahraki, "Cyclist detection, tracking, and trajectory analysis in urban traffic video data," 2017.

[31] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[32] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 3213–3223.

[33] F. Yu, H. Chen, X. Wang, W. Xian, Y. Chen, F. Liu, V. Madhavan, and T. Darrell, "Bdd100k: A diverse driving dataset for heterogeneous multitask learning," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2020.

[34] A. Kuznetsova, H. Rom, N. Alldrin, J. Uijlings, I. Krasin, J. Pont-Tuset, S. Kamali, S. Popov, M. Malloci, A. Kolesnikov, *et al.*, "The open images dataset v4," *International Journal of Computer Vision*, pp. 1–26, 2020.

[35] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Y. Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng, *TensorFlow: Large-scale machine learning on heterogeneous systems*, Software available from tensorflow.org, 2015. [Online]. Available: https://www.tensorflow.org/.

[36] C. Shorten and T. M. Khoshgoftaar, "A survey on image data augmentation for deep learning," *Journal of Big Data*, vol. 6, no. 1, pp. 1–48, 2019.

[37] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," *arXiv preprint arXiv: 1406.2661*, 2014.

[38] Wikipedia contributors, *Training, validation, and test sets — Wikipedia, the free encyclopedia*, [Online; accessed 26-June-2021], 2021. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Training,_validation,_and_test_sets&oldid=1025312311.

[39] P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for activation functions," *arXiv preprint arXiv:1710.05941*, 2017.

[40] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, "Activation functions: Comparison of trends in practice and research for deep learning," *arXiv preprint arXiv: 1811.03378*, 2018.

[41] Wikipedia contributors, *Cross entropy — Wikipedia, the free encyclopedia*, [Online; accessed 26-June-2021], 2021. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Cross_entropy&oldid=1024917567.

[42] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[43] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[44] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization.," *Journal of machine learning research*, vol. 12, no. 7, 2011.