Monitoring Traffic Control

Lecture	Data Science SoSe 2021
Participants	Das, Devikalyan (7007352), deda00002@stud.uni-saarland.de Ekmen, Beste (2572855), s8beekme@stud.uni-saarland.de Mundra, Akshay (7011525), akmu00001@stud.uni-saarland.de Sakhalkar, Kaustubh (2581456), s8kasakh@stud.uni-saarland.de
Submission date	23-07-2021
Chair	Univ.-Prof. Dr.-Ing. Wolfgang Maaß Chair in Information and Service Systems, Campus A5 4, 66123 Saarbrücken

Executive Summary

Availability of proper mechanisms for traffic monitoring can avoid various socio-economic impacts on the citizens as a reason due to chaos and delays [1]. Also, with the number of vehicles plying on the roads have manifolded since the last decades. Hence having a deeper understanding of traffic density is essential for effective monitoring and managing of the traffic. But availability of properly annotated data to give an idea of the density of traffic on roads at any point of time in a day is a challenge. Even though some datasets [2] are available, these have issues such as being captured from a fixed sensor mounted on some immobile surfaces, hence they become effective only for a particular road and setup but fail for other roads. Also a unified end-to-end framework starting from image acquisition to outputting the traffic density of a road through smartphone or web is not available.

In this project we tried to address this issue. Our focus is traffic density estimation from real time images mostly captured from smartphones or camera sensors mounted on the vehicles and outputting the density level through a smartphone or web. For this density estimation we used a Deep learning based approach to train with a diverse dataset created by considering various levels of traffic densities as well as occlusions during night. The trained model is finally deployed in a smartphone compatible application for inferencing. Also there will be freedom to the user regarding how to choose images whether from gallery or capture from camera.

Contents

List of Abbreviation

CNN - Convolutional Neural Network
SGD - Stochastic Gradient Descent

List of Figures

List of Tables

1 Introduction

Availability of proper mechanisms for traffic monitoring can avoid various socio-economic impacts on the citizens as a reason due to chaos and delays [1]. Also, with the number of vehicles plying on the roads have manifolded since the last decades. Hence having a deeper understanding of traffic density is essential for effective monitoring and managing of the traffic. But availability of properly annotated data to give an idea of the density of traffic on roads at any point of time in a day is a challenge. Even though some datasets [2] are available, these have issues such as being captured from a fixed sensor mounted on some immobile surfaces, hence they become effective only for a particular road and setup but fail for other roads. Also a unified end-to-end framework starting from image acquisition to outputting the traffic density of a road through smartphone or web is not available.

In this project we tried to address this issue. Our focus is traffic density estimation from real time images mostly captured from smartphones or camera sensors mounted on the vehicles and outputting the density level through a smartphone or web. For this density estimation we used a Deep learning based approach to train with a diverse dataset created by considering various levels of traffic densities as well as occlusions during night. The trained model is finally deployed in a smartphone compatible application for inferencing. Also there will be freedom to the user regarding how to choose images whether from gallery or capture from camera.

2 Data Set

For this project, we have considered data from two datasets CULane [2] and TrafficNet [3]. Individual dataset were not sufficient for providing enough information regarding the level of traffic density on various roads. Dataset [3] had images taken from unconditional angles and also had ambiguous labels such as fire on the roads. Hence, we have to manually select images from these datasets, annotate them as per the required class to create a diverse dataset that provides information of different levels of traffic density on different roads at different points of time. Also, care was taken while creating this dataset that there was sufficient data in each class to prevent class imbalance. The annotation was done based on three classes:

1. Dense Traffic
2. Sparse Traffic
3. No Traffic

3 Procedure and Analysis

We use convolutional neural networks (CNN) as a choice of our models for classifying images whether they have sparse, dense or empty traffic. CNNs have been the state-of-the-art models in classifying im-

ages and we choose CNNs because they outperform handcrafted models by a large margin. The case with neural networks is we have to ensure that they work for the given dataset. We initially checked our results with LeNet [4]. Since our final objective is to deploy the model on a smartphone, we do not use bigger networks which generally give better results at the cost of increased computation and memory usage. Our choice of model for computation is mobilenetv2 [5] which is the most widely chosen model for mobile deployment. We train the network on a combination of datasets for traffic net, cuLane and combination of both. We realized that the trafficnet [3] dataset contains some images that are not taken from the road. We trained the models anyway to check if such images turn out to be detrimental to the performance. It turned out that the trafficnet images were easy to classify but some part of the dataset was not relevant to road traffic congestion because the road images were taken from an aerial view therefore degraded the performance of the combined dataset. Hence we chose to discard the trafficnet dataset and train our model only on CuLane [2] dataset since the images of CuLane were more aligned towards real world traffic congestion detection scenarios.

3.1 Model

We use three settings in our training code with three datasets and two backbone architectures. The results of the test accuracy on each dataset and model are summarized in table 1.

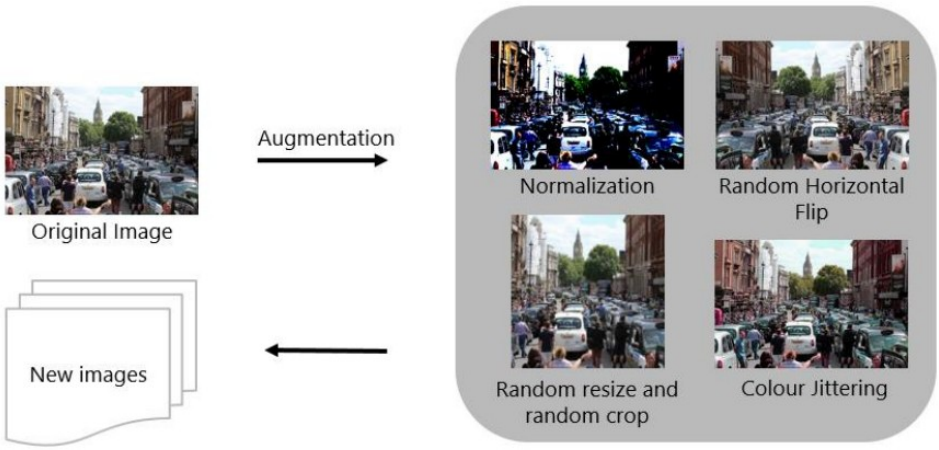

Figure 1: Illustrated augmentations (shown on the original image from Traffic-Net)

Since dataset size is insufficient to train a CNN model, we use data augmentation to artificially increase the number of images by applying image transformations such that the labels aren't changed. The key idea of applying augmentations is to transform our sample image in such a way that their target label doesn't change. We use the following random augmentations for our training which are illustrated in figure [1].

- Random resize and crop: Randomly crop a portion of the image and resize the image to input size of the network (224x224)
- Random horizontal flip: Randomly mirror the image horizontally with a probability of 0.5. Note that we do not apply vertical flip because it does not fit the road congestion detection problem

- Random color jitter: Change the color of the image by adjusting the hue, saturation, brightness parameters so that color of the image is affected.

During test time we only crop the image from the centre. We also normalize the image from [-1, 1] because it helps the network to learn parameters which are very similar to the pretrained dataset. We use ImageNet pretrained model mobilenetv2 because fine tuning on existing pretrained models is known to benefit the downstream task.

3.1.1 LeNet

LeNet-5 [4] CNN architecture is made up of 7 layers. The layer composition consists of 3 convolutional layers, 2 subsampling layers and 2 fully connected layers. It utilizes two significant types of layers, that are convolutional layers and subsampling layers as illustrated in the architecture. The architecture is illustrated in figure [2]

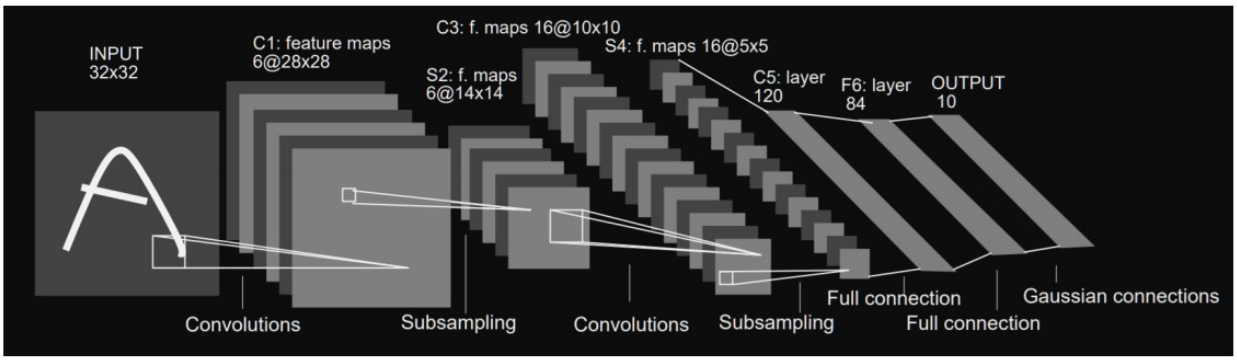

Figure 2: LeNet-5 Architecture (Pic Credit [6])

3.1.2 MobileNetV2

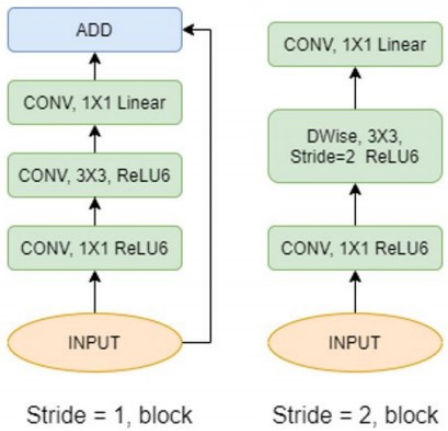

Figure 3: Convolution Blocks with Different Strides

Exam The MobileNetV2 [5] model was used which used inverted bottleneck blocks and residual connections. The reason for selecting Mobilenet V2 was its smartphone compatibility. This architecture has two types of blocks: one is a bottleneck block with stride 1 and the other with stride 2. Three layers of each block are stacked on top of each other. The blocks for different strides are illustrated in …

3.1.3 Training Details

We initialize the model with random weights for LeNet and use ImageNet pretrained weights for MobileNet. We train with stochastic gradient descent (SGD) with an initial learning rate of 0.01. We progressively decrease the learning rate at epochs [30, 60, 90] for LeNet and [10, 20, 30 , 40] for MobileNet with a factor of 0.1. This avoids overfitting. We also use nesterov momentum based averaging to get smoother results over time and have a weight decay of 0.0001. We also use tensorboard to log our training results which are given in the appendix. We trained all our experiments on a 6gb 1060 qmax GPU so the largest batch size we could fit was 64.

3.2 Android Application

TrafficMode is a traffic congestion detection application using the PyTorch Android API. It runs the trained model on images selected by the user and classifies as dense traffic, sparse traffic or no traffic. The application was developed on Android Studio 4.4.2 using the Java programming language. The application supports a minimum SDK version of 21 and targets the SDK version of 30. The code for the application as well as the apk file has been provided with the submitted code files. The details of the implementation can be found under the Appendix.

(a) (b) (c) (d) (e) (f)

Figure 4: Mobile Application Screens

The main purpose of the app was planned as detecting the traffic status by using the mobile pictures. Figure 4 shows the flow of the application screens. After the explanation of the functionalities (b), we have allowed users to pick a picture from the gallery or take a picture with the camera for classification (c). When the user picks a picture with either method (d-e) and clicks on the appearing button, the app passes the image to the lite model and obtains the classification result, which is later used to show the respective message (f).

4 Results and Discussion

In table 1, we compare the accuracy results of training the two architectures on Traffic-Net dataset, CuLane dataset and a combination of the two. As evident, MobileNetV2 outperforms LeNet across all datasets. Another critical point to note is that while TrafficNet gives a higher accuracy score, the underlying data is not very suitable for our case as we discussed earlier, and its performance is relatively poor during deployment. This is also evident when we train the model on the combined data, where it degrades the performance of CuLane dataset by 12.43%, suggesting that it is not suitable if we add it with a more realistic scenario. Hence we only use CuLane dataset for training. We give the graphs for loss and accuracy progression in the Appendix. Moreover, we use pytorch JIT to optimize the model for mobile devices.

Table 1: Results of model architectures on each dataset setting. Numbers reported are the accuracy obtained in the respective setting. The rows correspond to the model used and the columns correspond to the dataset used.

	Traffic Net	Trafficnet + CuLane	CuLane
LeNet	64.29	60.930	73.82
MobilenetV2	96.03	69.63	82.06

Using a smartphone application to detect traffic can be thought of as the first step towards smart city traffic monitoring. Using a model of experts or ensemble we can learn the traffic statistics of different places in the city. Our work can also include geo tagged image coordinates to measure statistics of country road traffic. Since the advent of autonomous cars, we see a future where road traffic scheduling will become commonplace and hope that others take efforts towards traffic congestion detection, mitigation and road management.

Bibliography

1. Matheus S Quessada, Rickson S Pereira, William Revejes, Bruno Sartori, Euclydes N Gottsfritz, Douglas D Lieira, Marco AC da Silva, Geraldo P Rocha Filho, & Rodolfo I Meneguette (2020). ITSMEI: An intelligent transport system for monitoring traffic and event information. *Internation al Journal of Distributed Sensor Networks, 16(10), 1550147720963751.*
2. Xingang Pan, & Xiaoou Tang (2018). Spatial As Deep: Spatial CNN for Traffic Scene Understanding. In *AAAI Conference on Artificial Intelligence (AAAI).*
3. Olafenwa M, Traffic-Net, (2019), GitHub repository, https://github.com/OlafenwaMoses/Traffic-Net
4. LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., & Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural computation*, *1*(4), 541-551.
5. Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, & Liang-Chieh Chen. (2019). MobileNetV2: Inverted Residuals and Linear Bottlenecks.
6. Richmond Alake. (2020). Understanding and Implementing LeNet-5 CNN Architecture (Deep Learning).
7. Tang J, HelloWorldApp, (2021), GitHub repository, https://github.com/pytorch/android-demo-app/tree/master/HelloWorldApp
8. Tamada, R (2017) IntroSlider, https://www.androidhive.info/2016/05/android-build-intro-slider-app/
9. Olafenwa J, PytorchMobile, (2019), GitHub repository, https://github.com/johnolafenwa/PytorchMobile
10. Kapoor T C, Object-Detector-Android-App-Using-PyTorch-Mobile-Neural-Network, (2021), GitHub repository, https://github.com/tusharck

Appendix

Code Description

We write our entire code in python with the pytorch framework. We describe what each file does in the following list

- dataloader/trafficnet.py: Loads the data and preprocesses the images and labels with the augmentations
- dataset/: Has the three dataset folders of cuLane, trafficnet and combined dataset in cuLane_trafficnet
- experiments/: Here we store all the model files and checkpoints of the experiments we run, including the tensorboard loggers. It is divided into three dataset folders and each dataset folder has a model file in which we store the graphs and checkpoints of the experiments
- scripts/ here you will find the sample script to run. make sure that you have pytorch installed along with tensorboard. ./script/run.sh will run all the experiments again but you have to change the datapath in the file appropriately.
- utils/ has two files for calculating the accuracy of the model and storing the running average during training. We acknowledge taking the functions from https://gist.github.com/weiaicunzai/2a5ae6eac6712c70bde0630f3e76b77b and https://discuss.pytorch.org/t/meaning-of-parameters/10655
- _for_mobile.py takes the best model and converts the checkpoint into one that can be used for mobile deployment. Change the path inside the code if you intend to use it. We take the code from https://pytorch.org/mobile/android/
- .py is the main training and testing code used. It has some arguments whose description can be obtained by $python train.py –help

Data Description

We have three dataset folders trafficnet, cuLane and a combination of both in culane_trafficnet. Each dataset folder is split into a train and test folder which contains the train and test set. Each train and test folders has images with the folders dense_traffic, sparse_traffic and empty_traffic

which represent the file for annotations. The cuLane dataset didn't come with these annotations so we manually annotated the images. The following is the number of images.

Table 2: The number of images

Modality	Label	traffic net	cuLane	cuLane+trafficnet
train	Sparse traffic	900	1765	2665
train	dense traffic	900	1016	1916
train	empty road	-	1757	1757
test	sparse traffic	200	268	468
test	dense traffic	200	128	328
test	empty road	-	205	205

Android Application

To make the model suitable for Android, we have used torch and torchvision libraries in order to serialize and optimize the model with "optimize_for_mobile" method.

We have saved the optimized model for the lite interpreter which is used on mobile applications with the "_save_for_lite_interpreter" method. This model is packaged inside the android application as an asset to be used on the device.

The main functions of the application were accessing the gallery to pick a picture, taking a picture with camera and accessing it, slider screen for app introduction and using the lite model to perform classification. For each of these tasks, we have developed our project by referring to some tutorials with respective repos for model usage [7], slider screen [8], camera usage [9] and gallery usage [10].

We have provided the respective messages for classification results inside the ModelClasses file which refer to dense traffic, sparse traffic and no traffic.

Learning Graphs

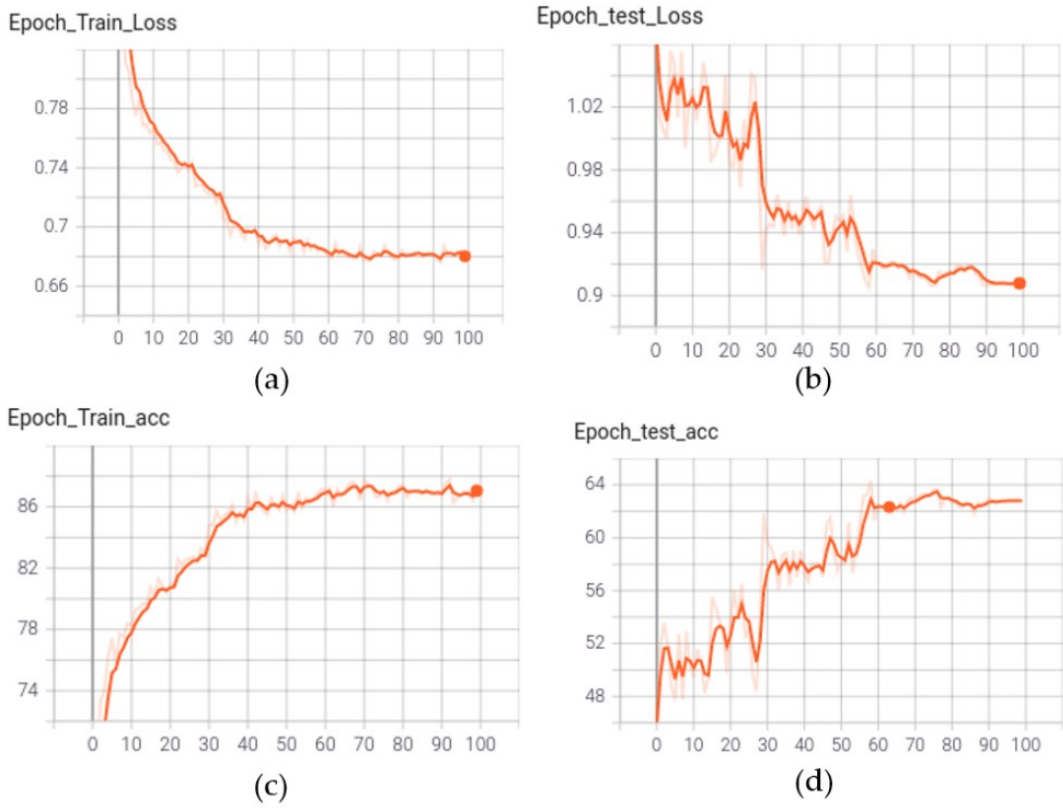

Figure 5: LeNet accuracy loss on training set (a), loss on test set (b), accuracy on training set (c), and accuracy on test set (d) of TrafficNet

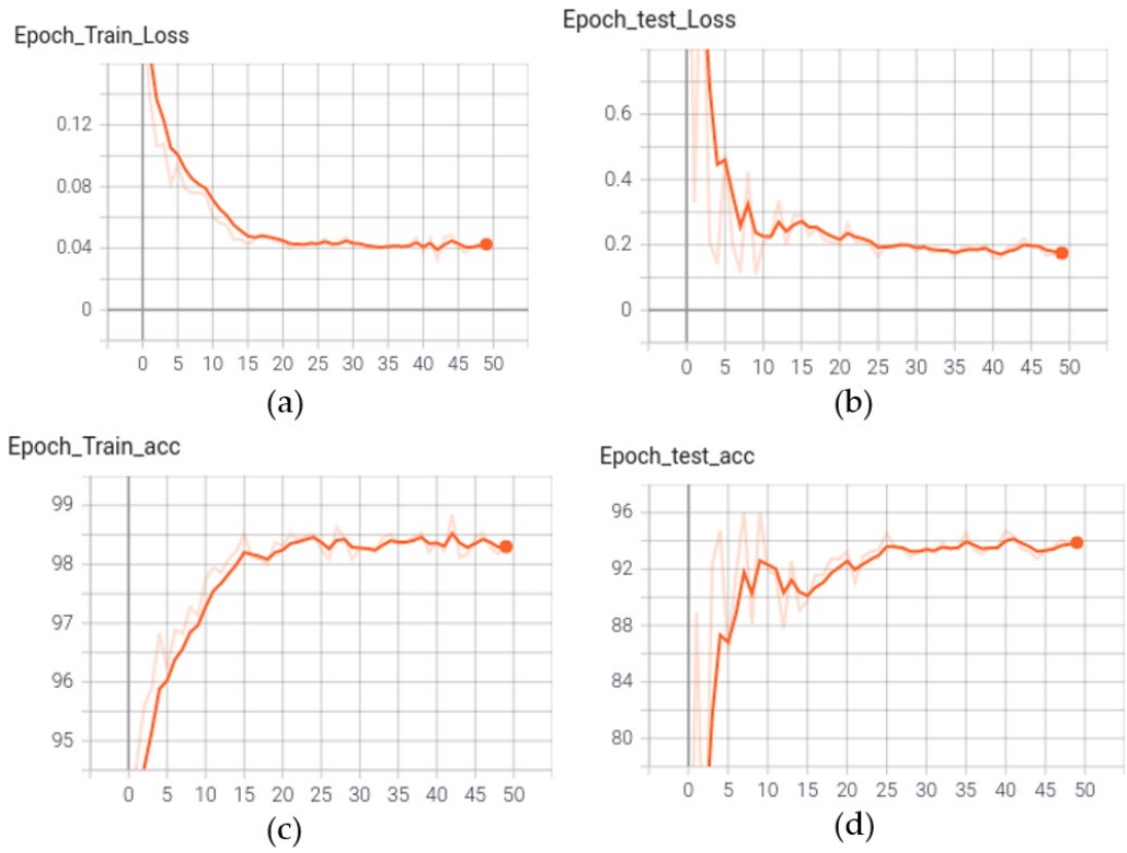

Figure 6: MobileNet accuracy loss on training set (a), loss on test set (b), accuracy on training set (c), and accuracy on test set (d) of TrafficNet

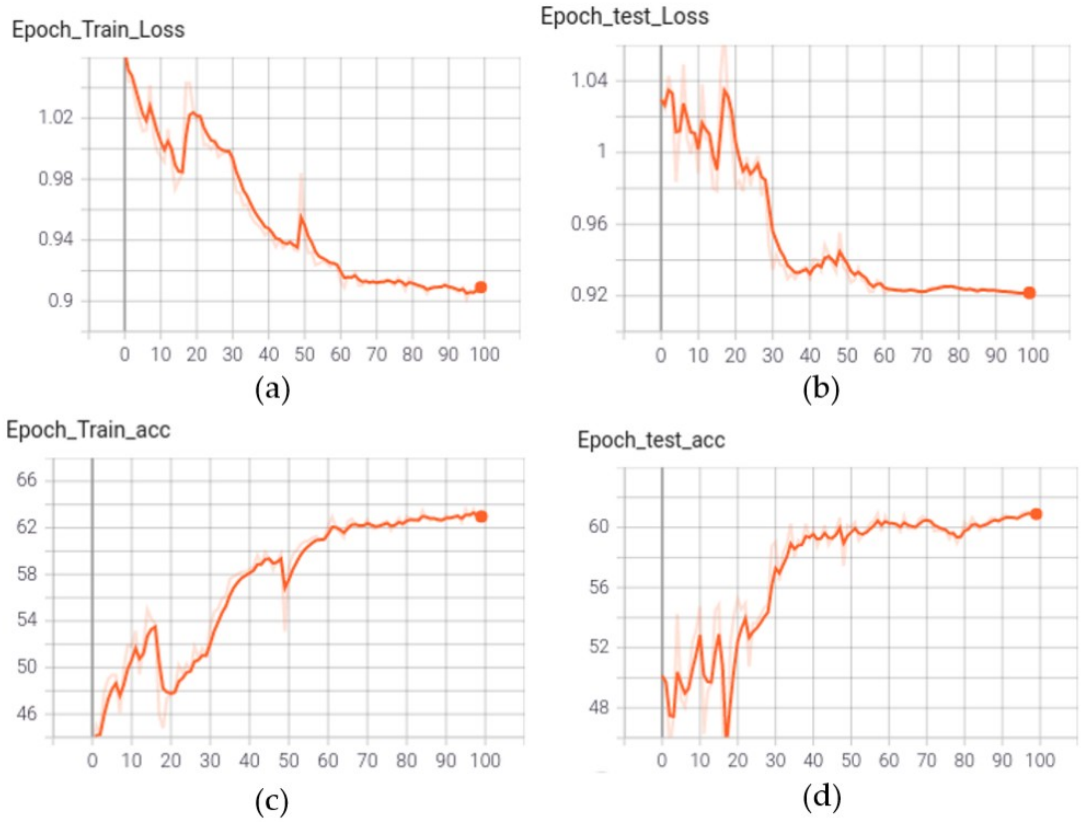

Figure 7: LeNet accuracy loss on training set (a), loss on test set (b), accuracy on training set (c), and accuracy on test set (d) TrafficNet + CuLane

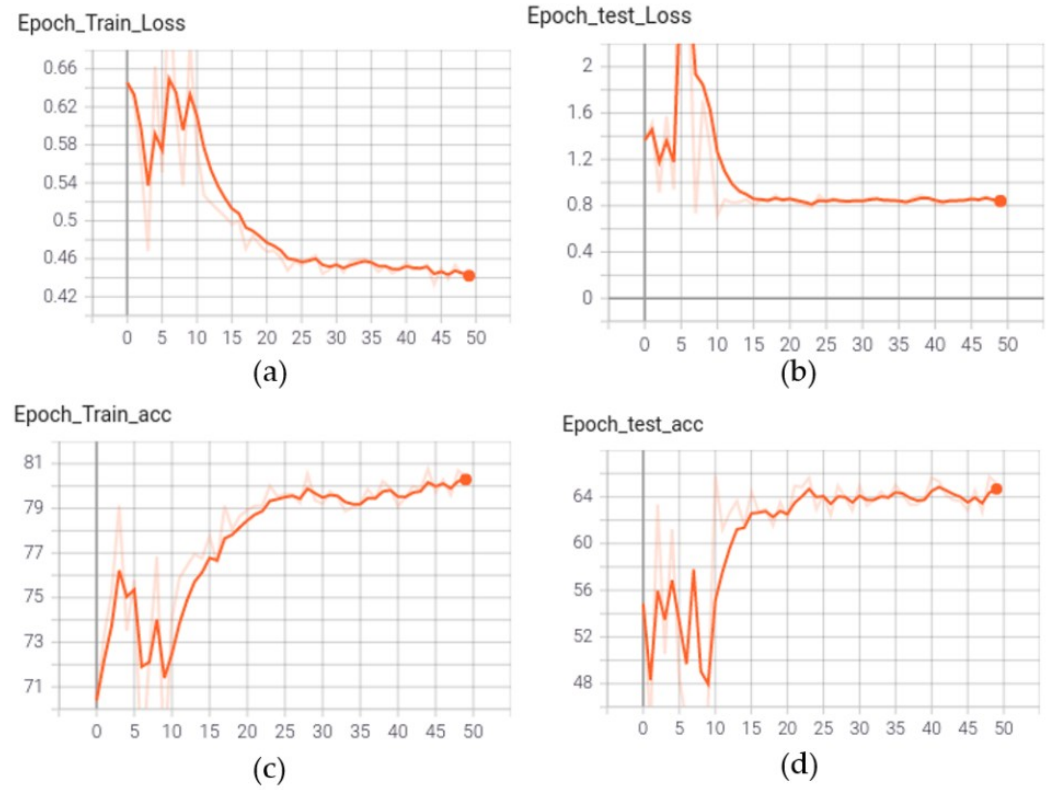

Figure 8: MobileNet accuracy loss on training set (a), loss on test set (b), accuracy on training set (c), and accuracy on test set (d) of TrafficNet + CuLane

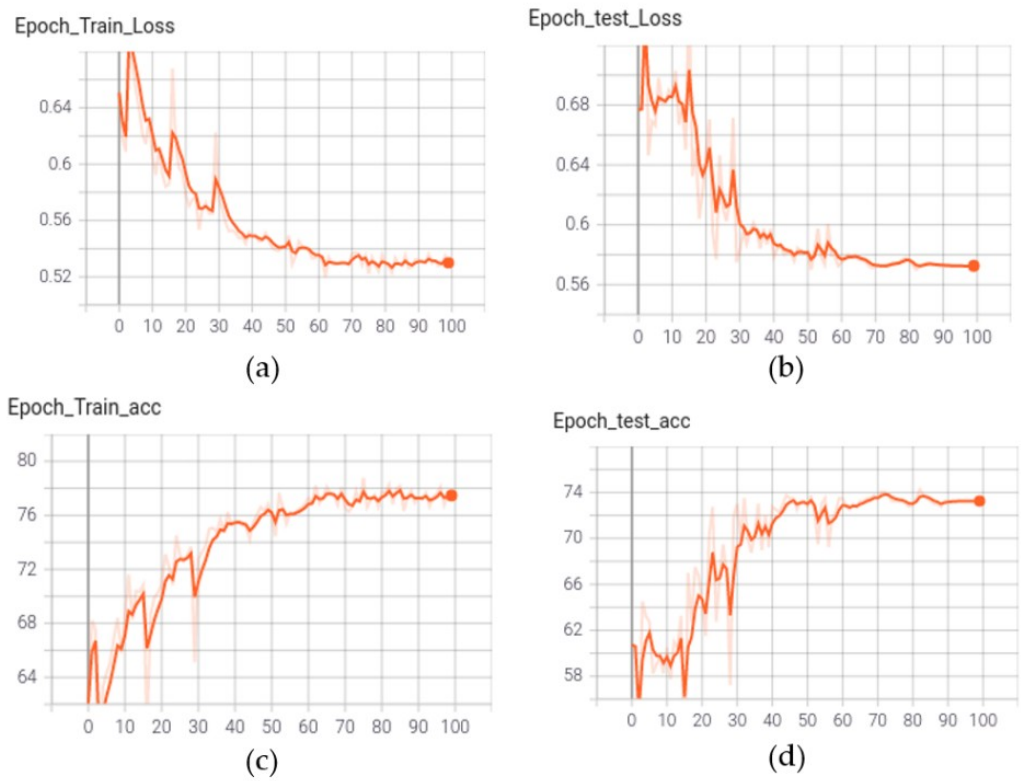

Figure 9: LeNet accuracy loss on training set (a), loss on test set (b), accuracy on training set (c), and accuracy on test set (d) of CuLane

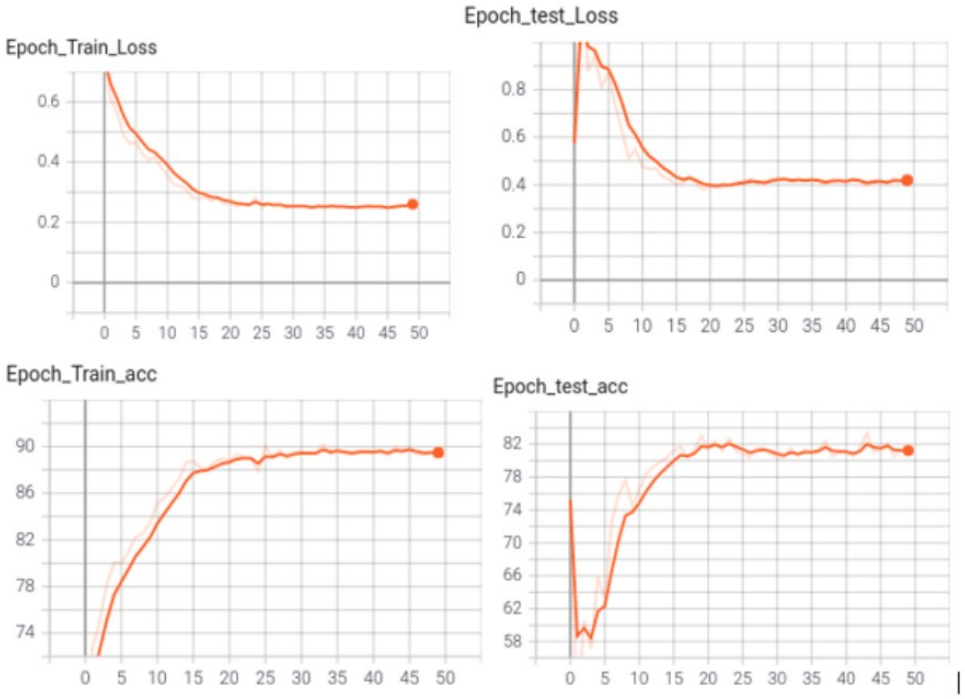

Figure 10: MobileNet accuracy loss on training set (a), loss on test set (b), accuracy on training set (c), and accuracy on test set (d) of CuLane

Declaration of Authorship

I affirm that I have produced the work independently, that I have not used any aids other than those specified and that I have clearly marked all literal or analogous reproductions as such.

Saarbrücken, 23.07.2021

Das, Devikalyan

Ekmen, Beste

Mundra, Akshay

Sakhalkar, Kaustubh