Driving Lane Detection on Smartphones using Deep Neural Networks

RAVI BHANDARI, Indian Institute of Technology Bombay, India
AKSHAY UTTAMA NAMBI and VENKATA N. PADMANABHAN, Microsoft Research, India
BHASKARAN RAMAN, Indian Institute of Technology Bombay, India

Current smartphone-based navigation applications fail to provide lane-level information due to poor GPS accuracy. Detecting and tracking a vehicle's lane position on the road assists in lane-level navigation. For instance, it would be important to know whether a vehicle is in the correct lane for safely making a turn, or whether the vehicle's speed is compliant with a lane-specific speed limit. Recent efforts have used road network information and inertial sensors to estimate lane position. While inertial sensors can detect lane shifts over short windows, it would suffer from error accumulation over time. In this article, we present DeepLane, a system that leverages the back camera of a windshield-mounted smartphone to provide an accurate estimate of the vehicle's current lane. We employ a deep learning–based technique to classify the vehicle's lane position. DeepLane does not depend on any infrastructure support such as lane markings and works even when there are no lane markings, a characteristic of many roads in developing regions. We perform extensive evaluation of DeepLane on real-world datasets collected in developed and developing regions. DeepLane can detect a vehicle's lane position with an accuracy of over 90%, and we have implemented DeepLane as an Android app.

CCS Concepts: • **Human-centered computing** → *Mobile computing*;

Additional Key Words and Phrases: Transfer learning, lane detection, convolutional neural networks

ACM Reference format:
Ravi Bhandari, Akshay Uttama Nambi, Venkata N. Padmanabhan, and Bhaskaran Raman. 2020. Driving Lane Detection on Smartphones using Deep Neural Networks. *ACM Trans. Sen. Netw.* 16, 1, Article 2 (January 2020), 22 pages.
https://doi.org/10.1145/3358797

1 INTRODUCTION

Sensors in a driver's smartphone have been used for various applications such as navigation assistance, road condition assessment, traffic congestion detection, driving safety monitoring, and so on [15, 27, 37]. An important input to many applications is the physical location or position of

Ravi Bhandari would like to thank Microsoft Research India and Confederation of Indian Industry (CII) for providing joint financial support under the Prime Minister's fellowship for doctoral research.
Authors' addresses: R. Bhandari and B. Raman, Dept. of Computer Science and Engineering, Indian Institute of Technology Bombay, Mumbai, Maharashtra, India, 400076; emails: {ravib, br}@cse.iitb.ac.in; A. U. Nambi and V. N. Padmanabhan, Microsoft Research India, Lavelle Road, Bengaluru, India; emails: akshay.uttama@gmail.com, padmanab@microsoft.com.

1550-4859/2020/01-ART2 $15.00
https://doi.org/10.1145/3358797

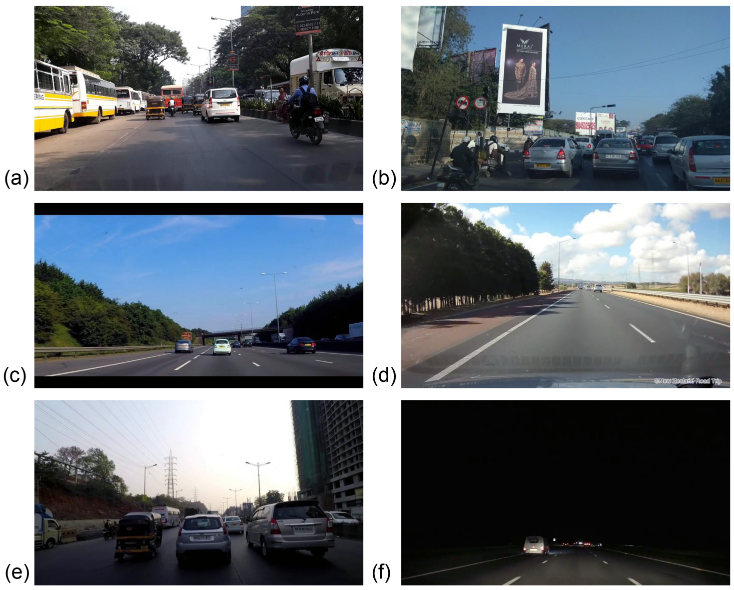

Fig. 1. Sample images used in DeepLane.

the vehicle. While GPS-based positioning is typically adequate for longitudinal positioning, i.e., positioning along the direction of the road, it is typically not for lateral positioning, i.e., lane positioning, since an error of even just a few meters could mean that the positioning is off by a lane or more.

Lane positioning is important for many applications, especially safety-related ones. For instance, it would be important to know whether a vehicle is in the correct lane for safely making a turn, perhaps even alerting the driver in advance if it is not, or whether the vehicle's speed is compliant with a lane-specific speed limit. Note that determining which lane the vehicle is in is different from knowing the correct lane that vehicle *should* be in for making the turn; the lane assist feature in the popular Google Maps application [4] only addresses the latter but not the former. If the vehicle is not in the correct lane to make the turn, then the driver may miss the turn, or worse, make a hurried attempt to turn [2], endangering the maneuvering vehicle as well as others around. In 2015, 2% of road fatalities in the US were caused due to making an improper turn [3]. Hence, automated detection of the lane that a vehicle is in is an important problem to consider.

Traditionally, lane detection has been attempted using road lane markings. For instance, high-end cars [1] are often equipped with a camera-assisted active lane keeping system that uses lane markings to detect whether the car is within a lane's boundaries. However, such an approach might not work in some challenging environments. For instance, sometimes there are no lane markings, e.g., in some developing country settings, as shown in Figure 1(a). Even if lanes are marked, vehicles might be packed together more tightly than the markings would suggest, e.g., three vehicles driving astride on a two-lane road (Figure 1(b)). Since the "effective" lane that the vehicle is in is what matters for applications such as knowing whether it is safe to turn, lane position in such a setting would have to be defined in a relative sense, by considering the presence or absence of vehicles to the left or right.

In this article, we present DeepLane, a camera-based system for lane detection, which we dub as "camera-assisted GPS." We use the back camera of a windshield-mounted smartphone, which gives a 70-degree view of the scene in front.

We first show the limitations of hand-crafted features and then turn to deep learning techniques to classify the lane of the vehicle. We build a three-way lane classifier to detect whether the vehicle is in the left, right, or middle lane. (Note that if the vehicle is found to be in neither the left-most nor the right-most lane, we deem it to be in the middle lane.) `DeepLane` is trained and tested using an extensive dataset composed of over 29,000 images from developed and developing countries in both daytime and nighttime conditions, and ranging from well-marked highways to unmarked city roads with chaotic traffic. Some sample images from our dataset are shown in Figure 1.

As extensive as our dataset is, it is insufficient to train a Deep Neural Network (DNN) from scratch. Therefore, we employ transfer learning, wherein a pre-trained network (we employ either the 8-layer AlexNet or the 16-layer VGG16, trained on 1.2M images from ImageNet [5] dataset) is first used to generate a compact feature representation of an image. In this article, we present two architectures towards lane detection, (i) without temporal features (here, the image features are independently evaluated to determine the lane position) and (ii) with temporal features (here, features from an image sequence are aggregated to determine the lane position). Using the above approaches, we see an overall lane classification accuracy of over 90%.

A specific form of "lane" detection is determining whether the vehicle is driving on the wrong side of the road, i.e., against the flow of traffic instead of with it. Such wrong-side driving is not uncommon in developing regions [12], with drivers sometimes choosing to take the shortcut of driving—say, on the shoulder—against the flow of traffic, to avoid a much longer drive the correct way and then taking a U-turn to come back. Needless to say, this is a significant safety hazard. In 2016, 5% of road fatalities in India were caused due to wrong-side driving [8]. The problem is so severe in some cases that authorities have considered infrastructure heavy and destructive "solutions" such as the installation of tire-killers [9]. While it is difficult to prevent this, if we are at least able to detect wrong-side driving of a vehicle using the same setup as above (i.e., a windshield-mounted camera), then it might be possible to discourage such behavior; say, in a fleet setting, where the supervisor is interested in overseeing each driver's overall safe driving behavior. We develop a binary classification system to indicate whether the vehicle is driving in the correct direction or the wrong direction. While it would appear that the problem is different from lane detection in that we would see the front-view of vehicles rather than the back-view, we find that hand-crafting features corresponding to the front- vs. back-view does not help due to heterogeneous vehicles, manually identifying relevant features, and the requirement of significant diverse data to train the network. Instead, employing the same DNN pipeline as for lane detection, with just the final classification step being retrained, yields an accuracy of over 90%.

We present an implementation of `DeepLane` optimized for Android smartphones, which can operate at 5fps on a Quad-core CPU and up to 15fps on an Adreno 530 GPU.

2 MOTIVATION AND PRIOR WORK

To motivate our work on `DeepLane`, we describe some of the existing approaches to lane detection along with their limitations.

Navigation apps today fail to give lane-level information, because they depend on GPS, whose location resolution is inadequate for resolving lane position. To establish this methodically, we took a ride through a city road, while changing lanes from time to time and also recording the GPS coordinates all along using a OnePlus 3 Android smartphone. The raw GPS points recorded in a section of the drive are shown in Figure 2(a), with the points recorded while in the right lane being shown in brown with a "star" marker and those in the middle lane in green with a "location" marker. Due to the inherent error of GPS, all the GPS points lie outside of the actual road segment. Furthermore, when we use the snap-to-roads [19] API from Google Maps, the points get mapped

(a) Raw GPS points (b) GPS points snapped to road

Fig. 2. Limitations of GPS for lane detection.

to the correct road segment but are all placed at the middle of the road, as shown in Figure 2(b). All of this points to the inadequacy of GPS for resolving lane position.

While improvements to GPS such as differential GPS (D-GPS) or dual-frequency GNSS could yield a high enough accuracy for lane positioning, these systems are not widely available. For instance, D-GPS stations are not present in large parts of developing regions, and dual-frequency GNSS is very new (e.g., just recently, on May 31, 2018, Xiaomi launched the "world's first dual-frequency GNSS smartphone" providing up to decimeter-level accuracy [11]).

Several past works [17, 34] have employed inertial sensors such as accelerometer and gyroscope in conjunction with GPS to detect lanes. While effective in detecting discrete lane-change events, such an approach suffers from an accumulation of error over time, which would impede accurate tracking of lane position. This effect is more pronounced if the initial state estimation is erroneous, too.

With the rising popularity of autonomous vehicles, LIDAR (Light Detection and Ranging), in combination with IMU (Inertial Measurement Unit) and GPS sensing, are being employed for lane detection [28, 30]. Although LIDAR-based systems have reported positioning errors under 1m [28], the high cost of LIDAR [6] prevents it from being adopted for general use. However, we just use a smartphone in DeepLane.

There has been much work on camera-based lane detection. Kim et al. [24] use the Hough transform to detect lane markings, which are then used to arrive at an estimate of left and right lane boundaries, thereby detecting the current lane. Wang et al. [35] use a spline-based road model and CHEVP (Canny/Hough estimation of vanishing points) to detect and track lanes. Yu et al. [38] estimate lane boundaries using multi-resolution Hough transform with a parabolic model. Such approaches have several shortcomings. First, most of the above methods depend on either prior knowledge of the road model or on detecting lane markings, which may not be present on all roads, especially in developing countries. Second, during nighttime, or even due to shadows in daytime, lane markings may not be visible and hence be hard to detect. Third, edge detection algorithms perform poorly on curved roads. Although there are several curve detection algorithms that assume the road curve to follow a parabolic model [21] or a hyperbolic model [26], it is hard to estimate the radius of curvature while the vehicle is driving on the curve. This is because of the bias in difference between the heading direction of the vehicle's body and its actual heading angle [23].

Yet another downside of depending on lane markings is that even when such markings exist, these might not be respected by the dense and heterogeneous traffic that is quite common in developing regions. For instance, on a road with two marked lanes, there could be three or more vehicles driving astride, as shown in Figure 1(b).

3 STRAWMAN DESIGN AND ITS LIMITATIONS

In view of the above, we present a strawman design that is also camera-based but does not depend on lane markings. The idea, instead, is to look at the pattern of movement of vehicles around. The intuition here is as follows: If we only see vehicles moving to our right, we can conclude that we are in the "left" lane, and likewise for the "right" lane. If vehicular movement is seen on both the left and the right sides, then we can conclude that we are in a "middle" lane.

Optical flow (OF) [33] is the basic technique we use for detecting motion. We tried two approaches based on optical flow:

1. Pixel-level OF: In this approach, we calculate the optical flow for each pixel in the image. The angle and magnitude of the flow vector corresponding to each pixel are then used as input features to an SVM classifier.

2. Vehicle-level OF: In this approach, we first identify all the vehicle objects in the frame using an object detector such as DNN-based YOLO [13]. We then find OF for only the detected vehicles. The idea here is to restrict motion analysis to objects of interest, i.e., vehicles, thus eliminating noise from other static objects in the scene. For each vehicle, we then calculate the mean angle and mean magnitude of the flow vectors, which are passed to the SVM classifier as features.

We tried these approaches on a subset of our dataset (described in Section 5). We computed OF on images extracted at 10 frames per second (fps) and the overall classification accuracy for pixel-level and vehicle-level techniques were 69.3% and 72.6%, respectively. On further inspection, we found that the majority of the misclassifications arose due to static objects and corresponding noisy OF values, and inaccuracies in detecting vehicle boundaries. While we could further feature engineering by hand, this is a tedious task that needs to be done with care to ensure generality. Therefore, we move to a DNN-based approach for feature extraction in `DeepLane`, which we present next.

4 DEEPLANE DESIGN

We now present the design of `DeepLane` to detect a vehicle's lane position. `DeepLane` uses the back camera of a windshield-mounted smartphone to detect the lane position. In this article, we assume, without loss of generality, that the vehicles are right-hand driven (with steering wheel on the right side) and follow the left-hand traffic rule (i.e., the vehicles are driven on the left side of the road).

Deep Neural Network (DNN)–based techniques [10, 13, 25] have shown a lot of promise for identifying and classifying objects in a scene by learning relevant features, without the need for feature engineering by hand. However, these DNNs require a significant amount of data for training. For instance, various DNN architectures such as AlexNet [25] and VGG [10] have been designed and trained on datasets such as ImageNet [5], which consists of 1.2M images belonging to 1K separate object categories such as vehicle types, sign posts, buildings, various household objects, dogs, cats, and so on.

Obtaining such a large volume of labelled diverse data to train a DNN for the lane detection problem is challenging. With the lack of adequate amount of data, the DNN model often overfits on the training data and would not adapt to diverse scenarios originating in the real world. In Section 5.2.2, we shed light on the efficacy of such an approach.

To circumvent this, we employ *transfer learning* [29, 31, 36], wherein models trained for one task capture relations in the data that can be reused for different problems in the same domain. Recent works have shown that pre-trained models have a strong ability to generalize to images outside the ImageNet dataset via transfer learning [22, 36]. For example, a model trained for classification on ImageNet data can be reused to detect various car types such as sedan, SUV, and hatchback,

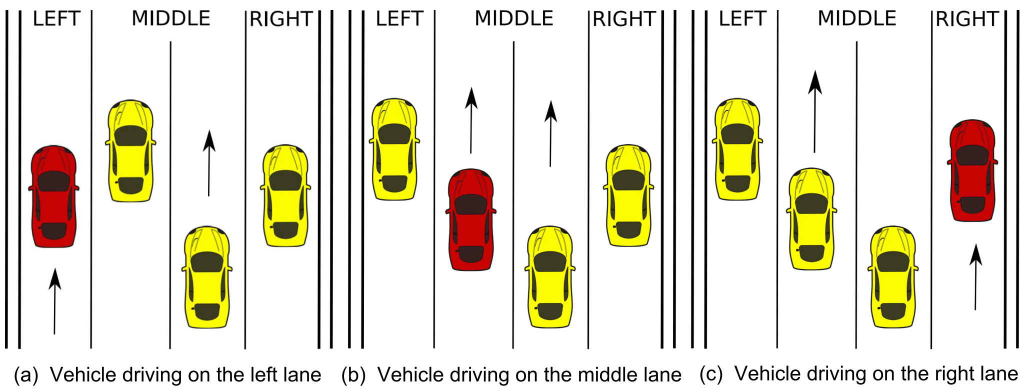
LEFT
MIDDLE
RIGHT
(a) Vehicle driving on the left lane
(b) Vehicle driving on the middle lane
(c) Vehicle driving on the right lane

Fig. 3. Lane scenarios showing left, middle, and right lanes where the interest vehicle is shown in red.

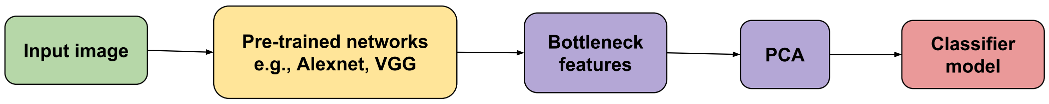

Fig. 4. DeepLane architecture towards lane detection.

without the need to train a new model from scratch. Features from intermediate layers of pre-trained DNN networks are used to train a much smaller network for classifying an image into three categories of cars instead of a 1K object categories. The intuition is that while the lower (i.e., initial) layers of the DNN architecture learn fine-grained information such as edges and texture, the higher layers learn coarse-grained information such as object parts and the final output layer is a classification layer [29]. Transfer learning extracts features from the higher layers that capture object parts and other patterns, since it can be reused for other tasks in the same domain.

4.1 DeepLane Architecture without Temporal Features

DeepLane's objective is to classify the lane that a vehicle is currently in as one of left, middle, or right, as defined in Section 3. We define the current lane as *left lane* when the vehicle is driving in the left-most permitted lane and sees vehicles ahead of it only towards its right. Similarly, if the vehicle is driving in the right-most permitted lane and sees vehicles ahead of it only towards its left, we define it as *right lane*. If the vehicle is found to be in neither the left-most nor the right-most lane, then we deem it to be in the *middle lane*, as shown in Figure 3.

To take advantage of pre-trained networks for image processing, DeepLane works with images (i.e., individual frames) rather than a video. It takes an image as input, extracts relevant features automatically, and classifies it as either left, middle, or right lane. DeepLane employs transfer learning where pre-trained models are used as *feature extractors*. The hypothesis is that the features extracted from pre-trained models trained on ImageNet have information on various objects of interest on the road such as vehicles, trees, sign posts, and so on, that can be used for lane detection, thus eliminating the need for manual feature identification and training from scratch, and the requirement of a large volume of training data. Figure 4 shows the architecture of DeepLane towards lane detection. The input image is fed to a pre-trained DNN model to extract features, which are called *bottleneck features*. These features are then used to build a classifier model that classifies an input image to either left, middle, or right. We now describe the building blocks of DeepLane.

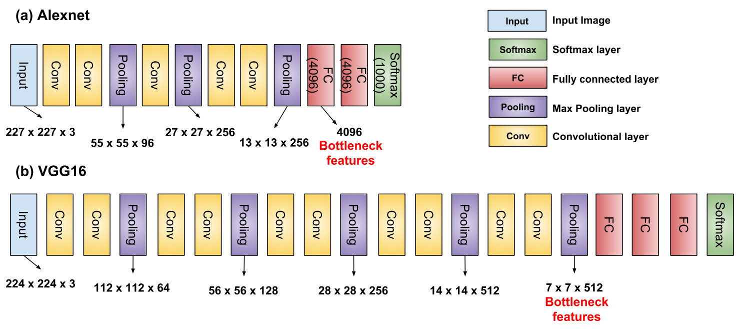

Fig. 5. Pre-trained DNNs: (a) AlexNet and (b) VGG 16.

4.1.1 DNN Models. There exist several DNN models, such as AlexNet [25], VGG [10], ResNet [20], and Inception [32], that are trained on the ImageNet dataset for classification. Since our goal is also to run DeepLane as an Android app that can detect the current lane in real-time, we select DNN models that can run efficiently on a smartphone. In this article, we leverage AlexNet and VGG16 as our pre-trained models due to the network design simplicity and performance. In Section 7, we describe how we implement DeepLane on an Android app and present performance results.

AlexNet [25]: AlexNet has five convolutional layers followed with three fully connected layers and a final softmax layer for classification as shown in Figure 5(a). The network takes an input of image size 227×227×3, where the image resolution is 227×227 pixels and there are three color channels (RGB). This is then compressed down to 4,096-dimensional feature vectors at the first fully connected layer and finally reduced to a 1K-dimensional feature vector, which is used for classification.

VGG16 [10]: VGG16 consists of 16 layers, which includes 13 convolutional layers and 3 fully connected layers followed by a softmax classifier as shown in Figure 5(b). VGG16 takes a 224×224×3 image as input and the convolutional layers compresses it to 7×7×512, i.e, 25,088 features. This is then fed to the fully connected layers to obtain a 1K-dimensional feature vector.

4.1.2 Bottleneck Features. The features extracted after the final convolutional layers are called the bottleneck features. In AlexNet, these features are of length 4,096 and in VGG16 it is 25,088 for each frame. These features encode information on various objects of interest on the road, such as vehicles, trees, sign posts, and so on, from the input frame. The extracted features are then fed to Principal Component Analysis (PCA) to reduce the dimensionality of the feature vector. PCA performs a linear mapping of the data to a lower-dimensional space such that the variance of the data in the low-dimensional representation is maximized. We apply PCA on bottleneck features to primarily speed up DeepLane performance on smartphones. In Section 5.2.1, we show what parts of the image bottleneck features concentrate to extract relevant features.

4.1.3 Classifier Model. The feature vectors are then used to build a classifier model that classifies an input image to either left, middle, or right lane. We use linear Support Vector Machine (SVM) as our 3-way classification model. During training, the classifier takes the label associated with the input image along with the bottleneck features. In testing phase, for each image DeepLane extracts the bottleneck features using pre-trained models, which are then evaluated on the trained classifier.

It is to be noted that in this architecture each input frame is classified independently to either left, middle, or right lane. However, typically a lane change event occurs over a short time span

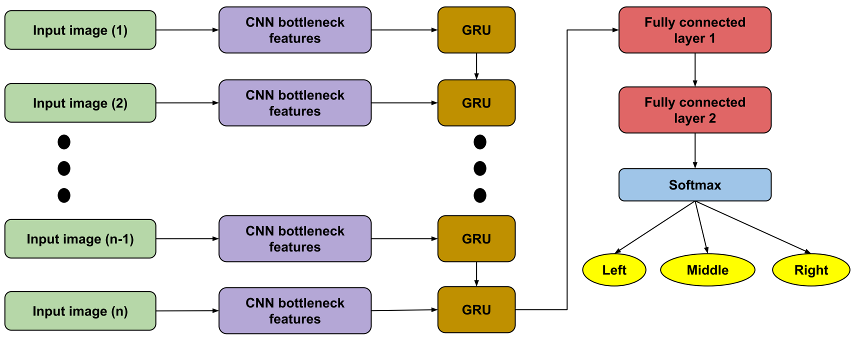

Fig. 6. DeepLane architecture towards lane detection with temporal features.

say few seconds. To this end, we extend the current architecture to handle temporal features as described next.

4.2 DeepLane **Architecture with Temporal Features**

Until now, each individual frame was used to detect the driving lane of the vehicle. However, in the real-world, DeepLane will be continuously recording and analyzing video frames. Can we leverage this temporal information to our benefit? The intuition here is that a vehicle is unlikely to change lanes abruptly, and a misclassification on a single frame can be phased out by incorporating the knowledge of the previous frames. Also, a lane change is a smooth maneuver that takes a few seconds to happen. Hence it is imperative to include temporal features across image frames for accurate lane detection. One challenge that arises is how to aggregate temporal features across images to make a driving lane prediction? To this end, we leverage the use of CNN + recurrent neural networks (RNN) as our basic building block to aggregate temporal features [18] as shown in Figure 6. In a nutshell, CNN derives the image level features and RNN aggregates the individual images across time. We now explain this architecture in detail.

For a given input image, we extract the bottleneck features as described in Section 4.1.2. This feature vector is then fed to an RNN for aggregation across consecutive frames. In our architecture we employ gated recurrent unit (GRU), which is like a long short-term memory (LSTM) but with two gates instead of three. The GRU block will combine both the current image features along with previous image features to derive a temporal feature vector for a given sequence of images. The aggregated temporal features for a sequence of images is passed to two fully connected layers followed with a softmax layer for classification. The output dimension of GRU was set to 256, while the fully-connected layers had an output dimension of 1,024 and 256, respectively. The final softmax layer gives the probability of the image sequence belonging to each of the three classes, viz. left, middle, or right lane. The classification, unlike the previous model, considers all the images in a sequence to determine the labels for each image, thus eliminating spurious misclassifications in a sequence.

Finally, it is important to identify the number of frames required to extract temporal features for lane detection. The longer the sequence, the more time required for lane detection and vice versa. This tradeoff depends on the application requirement; for example, it is imperative to have a short sequence of frames to determine lane position to accurately provide navigation information in GPS navigation systems. However, longer sequences can be employed to determine lane position if the application is to analyze lane changing patterns of a driver over time. We found $n = 5$ to work well in practice, i.e., frames of previous 5s were used for predicting the driving lane of the vehicle.

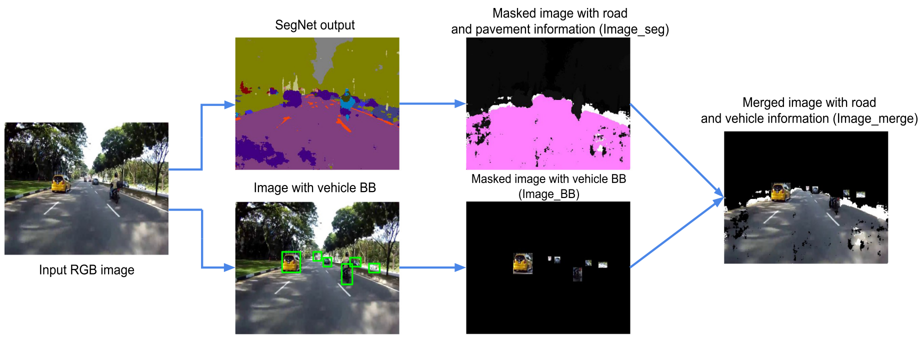
SegNet output
Masked image with road and pavement information (Image_seg)
Merged image with road and vehicle information (Image_merge)
Image with vehicle BB
Masked image with vehicle BB (Image_BB)
Input RGB image

Fig. 7. LD_{seg} stages: input image, road segmentation, vehicle BB detection, and merged image.

4.3 Image Segmentation–based Lane Detection

While DeepLane either takes an entire raw image or a sequence of images as input for automated feature extraction, we also tried a modified version that relies on input image segmentation for lane detection (LD_{seg}), the intuition being that having information just about the vehicle and road context may improve accuracy by ignoring other information that can contribute to the noise. To this end, we first segment the input image to extract information about objects of interest, *viz.,* (i) road surface and (ii) vehicles. We then mask the remaining part of the image to reduce noise. Figure 7 shows the input image and its corresponding segmented image for the road and vehicles, along with the merged input image. The merged input image with only road and vehicle information is fed to the DeepLane architecture to extract features and classify the current lane position. We now describe the steps involved.

4.3.1 Road Segmentation. Image segmentation techniques aim to partition an input image into multiple segments, i.e., set of pixels representing different object categories. Recent advances in deep learning have enabled pixel-wise segmentation of an image, not just to identify different objects but also their precise contours [14]. In this work, we employ SegNet [14], which is a popular pixel-wise image segmentation technique based on a deep encoder-decoder architecture. The encoder network in SegNet compresses the input image into a low-resolution feature map. The objective of decoder network is to map the low-resolution encoder feature maps to full input resolution feature maps for pixel-wise classification. SegNet is trained on a dataset that has pixel-wise annotations for various objects such as buildings, vehicles, roads, pavement, poles, and traffic signs.

In typical road scenes, the majority of the pixels belong to large classes such as road and buildings. On these classes SegNet performs significantly better than identifying pixels for vehicles in a scene. Thus, we use SegNet towards broad segmentation such as roads and pavement (see Figure 7) and apply customized techniques for vehicle detection described next. We call the image obtained with road and pavement information as $image_{seg}$.

4.3.2 Vehicle Segmentation Using Bounding Box (BB). Since SegNet segmentation masks are noisy for vehicles, we use techniques that can detect just the vehicles in the scene. To this end, we use the You Only Look Once (YOLO) [13] deep network to detect bounding box for all vehicles such as cars, motorbikes, and trucks in the scene. Figure 7 shows all the vehicles detected along with their bounding boxes. We call this image as $image_{bb}$.

At this stage, we combine $image_{seg}$ and $image_{bb}$, which has road and vehicle information, respectively. We create a mask on the original image to retain only those pixels that either identified as road/pavement in $image_{seg}$ or as a vehicle in $image_{bb}$, while the rest of the pixels are blacked

Table 1. Training and Test Images for Chaotic and Non-chaotic Datasets

	Chaotic		**Non-chaotic**	
Lane	Training	Test	Training	Test
Left	2,477	983	2,307	1,048
Middle	3,255	1,751	3,183	2,126
Right	4,398	2,230	3,520	1,885

out. The merged image is then fed as input to the transfer learning pipeline described in Section 4.1 to extract bottleneck features. We next present a detailed evaluation of DeepLane towards our goal of lane detection.

5 EXPERIMENTAL EVALUATION

In this section, we first describe the various real-world datasets we use to evaluate DeepLane. We then present an experimental evaluation of the accuracy of lane detection. We defer an evaluation of the computational performance on smartphones to Section 7.

5.1 Datasets

To test the efficacy of DeepLane, we have considered two datasets (i) *chaotic:* a dataset from roads in a developing region, where the traffic was rather chaotic without much lane discipline and (ii) *non-chaotic:* a dataset from various developed regions where traffic mostly followed lane discipline. In total, we have collected around 10h of video data across the two datasets. To reduce correlation between successive frames, we extracted the images at 1fps, resulting in 29,163 images (15,094 images from chaotic and 14,069 from non-chaotic). These images were labeled manually by us according to their respective class, *viz.,* left, middle, or right as defined in Section 4.1. In both the datasets, vehicles followed the left-hand driving rule (i.e., the vehicles were driven on the left side of the road). We now provide a detailed description of our datasets.

5.1.1 Chaotic Traffic Dataset. This dataset consists of video samples taken from four different cities in a developing region during both daytime and nighttime. The video samples were collected from two sources, (i) YouTube videos and (ii) manually by use, using a windshield-mounted smartphone on vehicles. In our manual data collection setup, we deployed smartphones in a fleet of 10 cabs across multiple days and sampled video frames at 1fps to extract images.

This dataset includes data from locations where lane markings are often absent, and even if present, vehicles usually do not follow lane discipline, often straddling lanes. The data includes curvy roads, shadows, and the presence of heterogeneous vehicles in the scene. Further, the data was collected at both highways and city roads. Some sample images from this dataset are shown in Figures 1(a), (b), and (e). The dataset consists of a total of 15,094 images including both daytime and nighttime images, which is then split into training and testing as shown in Table 1.

5.1.2 Non-chaotic Traffic Dataset. This dataset consists of video samples collected using YouTube videos from the following countries: Singapore, UK, Australia, South Africa, and Japan. The data represent non-chaotic traffic images with proper lane markings and well organized traffic. The images represent a mix of motorways and city roads. Figures 1(c), (d), and (f) show sample images belonging to this dataset in both daytime and nighttime conditions. As with other datasets, we extracted frames at 1fps from the video samples. The dataset consists of a total of 14,069 images, which is split into training and testing data as shown in Table 1.

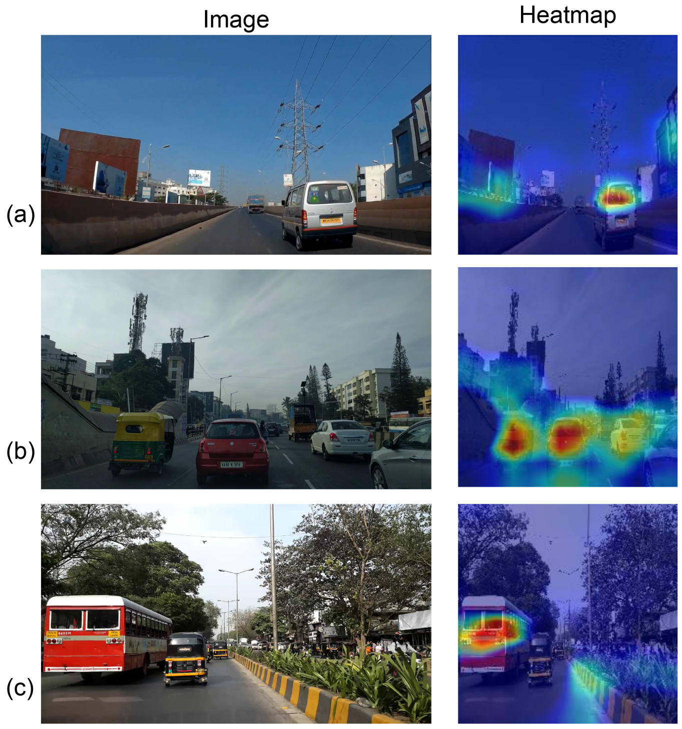

Fig. 8. Bottleneck features for left, middle, and right lane images.

5.2 Lane Detection Results Using DeepLane

We present lane detection results on the two datasets described in Section 5.1 for both the DeepLane architectures, *viz.,* with temporal features (TF) as well as without TF. In our evaluation, we considered both AlexNet and VGG for feature extraction. Further, based on empirical evaluation, in our implementation, we reduce the dimensions to 300 feature vectors for both AlexNet and VGG. The different configurations considered are, *viz.,*

AlexNet-4096: It uses 4,096 bottleneck features for classification.

AlexNet-300: It reduces 4,096 features to 300 features using PCA.

VGG-25088: It uses 25,088 bottleneck features for classification.

VGG-300: It reduces 25,088 features to 300 features using PCA for classification.

5.2.1 *Evaluation of Bottleneck Features.*

We first present some qualitative results showing what parts of the image are identified as relevant using bottleneck features extracted from pre-trained networks. As mentioned earlier, since these networks are trained on large datasets such as ImageNet [5], the bottleneck features concentrate on various object parts and other relevant patterns in the image. The key idea is to eliminate manual identification of features and allowing the DNN to identify relevant features of interest.

Figure 8 shows sample images when a vehicle is in the left lane, middle lane, and right lane. The figure also shows a heatmap, which highlights the parts of the image the bottleneck features represent. These heatmaps can be viewed as activation maps obtained as a result of stacking up convolutional filter output from each layer. Intuitively, the network will learn filters that activate when they see some type of distinct visual feature such as object parts.

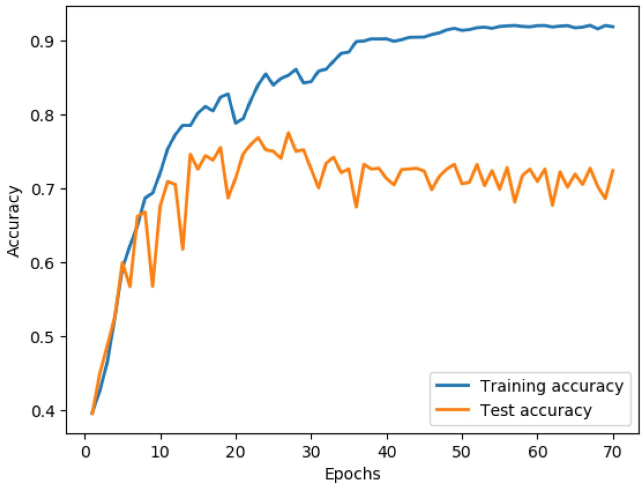

Fig. 9. Training accuracy of a network (VGG-16) trained from scratch on chaotic dataset.

In Figure 8(a), the heatmap shows the parts of the vehicle to the right and other static objects such as pavement to the left being activated when the vehicle's current lane position is left. Similarly, Figure 8(b) shows the heatmap of an image when the vehicle's current lane is middle. It can be seen that the bottleneck features ignore background information such as blue sky, and concentrate on vehicles to the front, left, and right. Finally, in Figure 8(c), bottleneck features focus on the vehicle to the left and road divider to the right for images when the vehicle is in the right lane. Intuitively, since bottleneck features are extracting different features for each class of images, (i.e., left lane, right lane, and middle lane), one can use these features to detect the current lane position. We now present lane detection results from both chaotic and non-chaotic datasets.

5.2.2 Training Accuracy of DNN Networks Trained from Scratch. A straightforward approach for lane detection would be to train a DNN network from scratch. Figure 9 shows the performance of such an approach on the chaotic traffic dataset. We employed VGG-16 as the baseline DNN network. It could be seen that the model shows signs of overfitting on the training data, where the model achieves high accuracy from early epochs itself. The training accuracy flattens out to around 92% after 40 epochs, whereas the test accuracy lingers at a value around 70%. This is mainly due to shortage of labelled data across various classes leading to overfitting based on the training images. To overcome this, DeepLane uses transfer learning–based architecture that enables usage of DNN with fewer labeled images.

5.2.3 Chaotic Traffic Dataset. We now present the case in which we use individual image frames only, i.e., without temporal features (without TF) for lane classification. For this, we trained a linear SVM classifier model on bottleneck features extracted from the training data in chaotic traffic dataset (see Table 1).

Table 2 shows the confusion matrix (without TF) when the current vehicle's lane is classified to either left or middle or right lane. The diagonal elements represent the instances for which the predicted label matches the ground truth label, while off-diagonal elements correspond to mislabeling by the classifier. It can be seen that the majority of the classifications lie on the diagonal of the table, indicating correct classifications. Further, the number of misclassifications for VGG have reduced as compared to AlexNet due to the deeper architecture, where the former has 16 layers and the latter has 8.

Table 2. Lane Detection Confusion Matrix in Chaotic Dataset without TF

		Left	Middle	Right
AlexNet-4096	Left	**0.88**	0.09	0.03
	Middle	0.08	**0.74**	0.18
	Right	0.03	0.08	**0.89**
AlexNet-300	Left	**0.84**	0.11	0.05
	Middle	0.06	**0.77**	0.17
	Right	0.03	0.08	**0.89**
VGG-25088	Left	**0.90**	0.08	0.02
	Middle	0.05	**0.89**	0.06
	Right	0.01	0.04	**0.95**
VGG-300	Left	**0.85**	0.12	0.03
	Middle	0.04	**0.85**	0.11
	Right	0.01	0.05	**0.94**

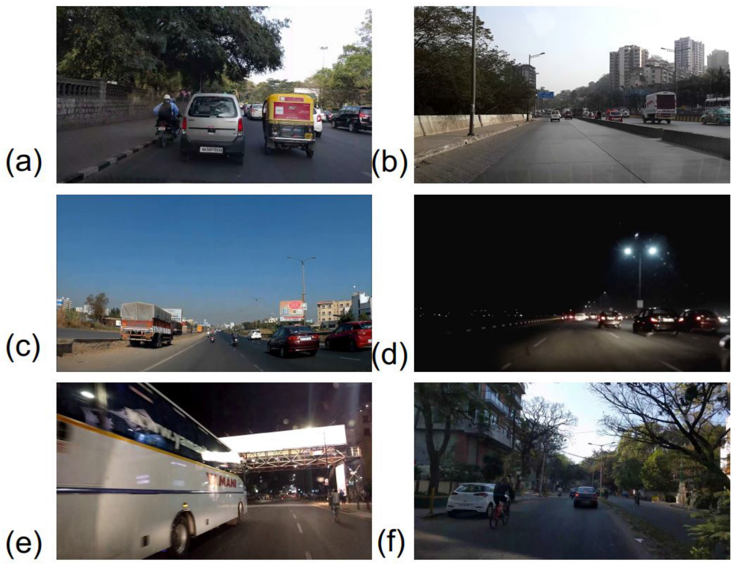

Fig. 10. Lane misclassifications in chaotic dataset.

Upon analysis of misclassifications, we found that several instances were misclassified due to parked vehicles on either side of the road (mostly on pavement) or instances where the pavement's surface looked similar to the road's surface. Figure 10 shows a few examples of misclassification. In the first two images, the vehicle is in the left lane but was classified as being in the middle lane. This was mainly because DeepLane could not distinguish between the pavement's surface and road surface as they look similar (see Figures 10(a) and (b)). In Figure 10(c), the vehicle is in the left lane (ground truth) but was classified as being in the middle lane, due to the parked vehicle off the road to the left. Figure 10(d) shows an instance where the left lane is not adequately illuminated, due to which DeepLane misclassifies the middle lane (ground truth) as the left lane. Figure 10(e) shows a scenario where the entire left lane view is blocked by a bus, resulting in the classifier detecting

Table 3. Lane Detection Confusion Matrix in Chaotic Dataset with TF

		Left	Middle	Right
AlexNet-4096	Left	**0.89**	0.08	0.03
	Middle	0.02	**0.89**	0.09
	Right	0.03	0.09	**0.88**
AlexNet-300	Left	**0.93**	0.16	0.01
	Middle	0.03	**0.89**	0.08
	Right	0.02	0.08	**0.90**
VGG-25088	Left	**0.94**	0.06	0.00
	Middle	0.02	**0.95**	0.03
	Right	0.00	0.03	**0.97**
VGG-300	Left	**0.92**	0.08	0.00
	Middle	0.03	**0.92**	0.05
	Right	0.00	0.03	**0.97**

Table 4. Lane Detection Accuracy in Chaotic Dataset with/without TF

	AlexNet-4096	AlexNet-300	VGG-25088	VGG-300
Accuracy without TF	83.6%	83.8%	91.3%	88.0%
Accuracy with TF	88.4%	90.0%	95.4%	93.1%

the current vehicle's lane as left as opposed to middle (ground truth). Finally, Figure 10(f) shows an instance where the road divider to the right of the vehicle looks similar to road surface. Hence, `DeepLane` classifies it as the middle lane as opposed to the right lane (ground truth).

Next, we present the corresponding confusion matrix for `DeepLane` with temporal features in Table 3. We can see that overall across all scenarios the accuracy of detection improves compared to the without TF case. Upon closer observation, we can see that all the extreme cases (left classified as right, or right classified as left) are non-existent for VGG configurations and reduced significantly for the Alexnet configurations.

Table 4, row 1, shows the overall accuracy of lane detection per image (averaging the diagonal elements from Table 2) across various configurations. VGG-25088 has highest accuracy of 91.3% in detecting the current lane position, whereas AlexNet-4096 has 83.6% accuracy. In row 2, we show the results using temporal features. An overall accuracy improvement of 6% was observed for the Alexnet configurations, and over 5% for the VGG configurations. Further, it can be seen that by reducing the feature vectors to 300 using PCA for both AlexNet and VGG, the accuracy drop is minimal, with corresponding benefits in computational efficiency (see Section 7). From now on, we show evaluation results on AlexNet-300 and VGG-300 throughout the article. We also trained separate SVM models for day (or night) separately, and the resulting accuracy was similar to combined day and nighttime data.

5.2.4 Non-chaotic Traffic Dataset. Table 5 shows the confusion matrix for lane detection without TF on the non-chaotic dataset for AlexNet-300 and VGG-300 (see Table 1 for training and test data split). It can be seen from Table 5 that majority of the classifications lie on the diagonal of the table, indicating correct classifications.

In this dataset, we found that majority of the misclassifications were due to the presence of shoulders on the left or right side of the road, or due to no vehicles being present in the scene.

Table 5. Lane Detection Confusion Matrix
in Non-chaotic Dataset without TF

		Left	Middle	Right
AlexNet-300	Left	**0.91**	0.04	0.05
	Middle	0.04	**0.86**	0.10
	Right	0.01	0.12	**0.87**
VGG-300	Left	**0.94**	0.04	0.02
	Middle	0.03	**0.92**	0.05
	Right	0.03	0.13	**0.84**

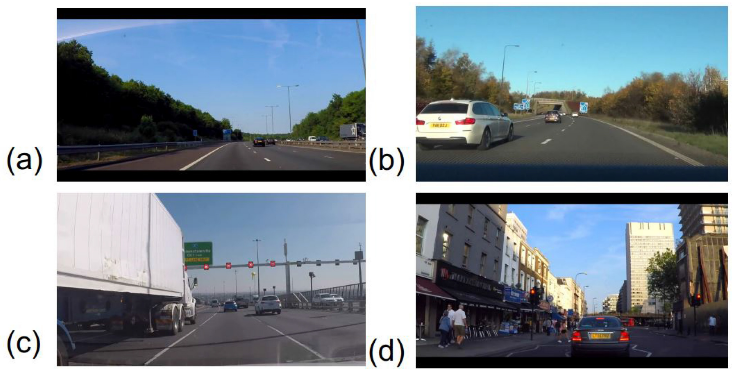

Fig. 11. Lane detection misclassifications in non-chaotic dataset.

Table 6. Lane Detection Confusion Matrix
in Non-chaotic Dataset with TF

		Left	Middle	Right
AlexNet-300	Left	**0.89**	0.06	0.05
	Middle	0.07	**0.86**	0.07
	Right	0.01	0.01	**0.98**
VGG-300	Left	**0.95**	0.05	0.00
	Middle	0.02	**0.93**	0.05
	Right	0.00	0.01	**0.99**

Some misclassifications are shown in Figure 11. Figures 11(a) and (b) show instances where there is a shoulder present towards the left and right side of the road, respectively. Due to this, DeepLane misclassifies the vehicle's lane position as middle instead of left or right, respectively. As in the case of the chaotic traffic dataset, in some of the instances the entire left lane view is blocked by a bus/truck, leading to a misclassification (see Figure 11(c)). Finally, few instances get misclassified from right lane to middle lane, as shown in Figure 11(d), due to absence of road dividers. One way to reduce these misclassifications would be to add the misclassified images back to the training set to improve accuracy, which is known as hard positive/negative sampling [16].

Table 6 shows the confusion matrix for DeepLane architecture with temporal features. It can be seen that across diagonal the classification accuracy is significantly improved for both VGG and Alexnet configurations. Further, due to consideration of temporal features across image sequence

Table 7. Lane Detection Accuracy in Non-chaotic Dataset with/without TF

	AlexNet-4096	AlexNet-300	VGG-25088	VGG-300
Accuracy without TF	87.0%	86.0%	91.1%	90.2%
Accuracy with TF	88.4%	88.8%	94.6%	93.7%

Table 8. Lane Detection Accuracy in Merged Dataset with/without TF

	AlexNet-4096	AlexNet-300	VGG-25088	VGG-300
Accuracy without TF	83.3%	83.1%	91.6%	88.4%
Accuracy with TF	94.0%	92.1%	94.8%	92.5%

Table 9. Lane Detection Confusion Matrix
in Merged Dataset without TF

		Left	Middle	Right
AlexNet-300	Left	**0.85**	0.09	0.06
	Middle	0.06	**0.80**	0.14
	Right	0.03	0.12	**0.85**
VGG-300	Left	**0.88**	0.09	0.03
	Middle	0.05	**0.86**	0.09
	Right	0.01	0.08	**0.91**

the misclassifications in extreme cases such as left to right or right lane are negligible. Table 7 shows the overall lane detection accuracy with and without TF for various configurations. An overall accuracy of 86% was found for AlexNet-300, which further improved to around 89% when temporal features (TF) were considered. Similarly, an accuracy of 90.2% and 93.7% was found for VGG-300 without and with TF, respectively.

5.2.5 Merged Dataset. Finally, we also show the lane detection accuracy when we combine both the chaotic and non-chaotic traffic datasets, we call it *merged dataset.* Table 8 shows the lane detection accuracy with and without temporal features (TF). It can be seen that the overall accuracy on the merged dataset is similar to individual datasets for the VGG configurations, but it shows a significant improvement for Alexnet configurations. This may be attributed to the improved diversity in the data, thus helping a shallower network (Alexnet in our case) in generating distinctive features. Overall, some of the misclassifications found in individual datasets were eliminated due to the diversity in the data. Table 9 shows the confusion matrix for the merged data (without TF) and it can be seen that the off-diagonal elements are lower compared to individual datasets, showing reduction in misclassifications. Further, we also performed 10-fold cross-validation, which resulted in similar overall accuracy numbers showing the robustness of the classifier. It was also observed that considering temporal features for the merged dataset leads to total removal of extreme cases for both the Alexnet and VGG configurations, as shown in the confusion matrix in Table 10.

5.3 Segmentation-based Results

To test the efficacy of the LD_{seg} approach described in Section 4.3, we evaluated it on our non-chaotic traffic dataset. With VGG-25088 as the DNN model, the overall accuracy obtained was 84.3%. For the same dataset, we found `DeepLane` has higher accuracy, i.e., 91.1%.

Table 10. Lane Detection Confusion Matrix in Merged Dataset with TF

		Left	Middle	Right
AlexNet-300	Left	**0.94**	0.06	0.00
	Middle	0.05	**0.89**	0.06
	Right	0.00	0.03	**0.97**
VGG-300	Left	**0.95**	0.05	0.00
	Middle	0.04	**0.90**	0.06
	Right	0.00	0.03	**0.97**

Table 11. Lane Detection Confusion Matrix Using LD_{seg}

	Left	Middle	Right
Left	**0.95**	0.03	0.02
Middle	0.07	**0.81**	0.12
Right	0.01	0.16	**0.83**

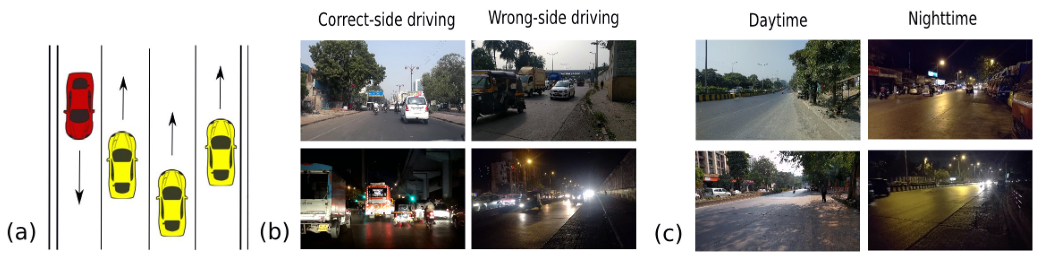

Fig. 12. (a) Wrong-side driving scenario, (b) Sample images from our dataset, (c) Misclassifications.

Table 11 shows the confusion matrix for lane detection using LD_{seg}. The majority of the misclassifications was due to (i) inaccuracy in accurately detecting road surface and pavement and (ii) inaccuracy in detecting accurate vehicle bounding boxes, especially when the vehicles were far away. Hence, the resulting segmented image was noisy and included other background information leading to poor accuracy in detecting the vehicle's lane. Further, LD_{seg} requires use of three separate DNNs for (i) road segmentation, (ii) vehicle segmentation, and (iii) feature extraction, which are computationally expensive and not feasible to run on smartphones. Due to the above reason, we discard the segmentation-based approach for lane detection.

6 WRONG-SIDE DRIVING DETECTION

In the previous section, we showed the efficacy of DeepLane in detecting the current lane that a vehicle is in. A specific form of lane detection arises when the vehicle is driving on the wrong side of the road (henceforth abbreviated as WD), i.e., against the flow of traffic. Such wrong-side driving is not uncommon in the chaotic road conditions in developing regions. Here, we focus on the specific case where the vehicle in which the camera is mounted is itself driving on the wrong side, along the edge of the road. Figure 12(a) depicts the wrong-side driving scenario, where the interest vehicle in red is driving on the wrong side.

Table 12. Training and Test Images for Wrong-side Driving

	Training data	Test data	Time of Day
Correct-side	3,000	2,213	day
Wrong-side	3,000	3,614	day
Correct-side	2,200	1,618	night
Wrong-side	3,000	2,717	night

The GPS available in smartphones typically lacks the resolution needed to detect wrong-side driving on a two-way road, just as it is inadequate for lane detection (Section 4). (A one-way road is a different case, where GPS might suffice when coupled with map annotations indicating the direction restrictions.) Therefore, we focus on a camera-based approach, just as with lane detection. The intuition here is if a vehicle travels in the correct direction, the camera view would largely be composed of the rears of the vehicles in front, including such features as the tail-lights. However, while going in the wrong direction, the camera would mostly see a frontal view of vehicles, including their headlights.

While we could build a detector to identify the front view versus the rear view of vehicles, this would require a significant amount of labeling, identifying features of interest, and collecting a sufficiently diverse set of data. To avoid these difficulties, we follow the same `DeepLane` architectures as described in Section 4.1 and Section 4.2. The idea is, given an input image along with a label (correct-side driving or wrong-side driving), the `DeepLane` architecture extracts relevant features using pre-trained networks and then uses these features to classify the scene into correct-side versus wrong-side driving (a two-way classification as opposed to three-way for lane detection).

To train the network, we took a subset of labeled data corresponding to correct-side and wrong-side driving in both daytime and nighttime conditions. The bottleneck features extracted from the pre-trained networks were fed to a classifier to build a model for two-way classification. This classifier, as described in Sections 4.1 and 4.2 could either be a linear SVM or an RNN (GRU), depending on whether temporal features were used or not. As in the case of lane detection, we believe the bottleneck features extracted before the final fully connected layers in the DNN include the most relevant features with respect to object parts such as headlights, tail-lights, and other vehicle parts that can be used to detect wrong-side driving.

Dataset: To evaluate the accuracy of our approach, we collected real-world data on various road segments in developing regions. Since we did not wish to risk driving on the wrong side of the road, we parked at the edge of the road, facing the wrong side, to capture videos. Though we remained stationary during data collection, we were nevertheless in constant motion relative to the incoming traffic. Since the relative speed would be low, we sampled the videos at 1fps to reduce the correlation between successive sampled images. The dataset consists of a total of 6h of video data, with 2.5h and 3.5h of data for correct-side and wrong-side driving, respectively. We extracted frames at 1fps from this video, resulting in a total of 21,362 images, with 9,031 and 12,331 images corresponding to correct- and wrong-side driving, respectively. Some sample images of correct- and wrong-side driving are shown in Figure 12(b). The data split for training and testing the network is shown in Table 12 for both day and nighttime images.

We now present our evaluation of WD using the architecture described in Section 4.1. Tables 13 and 14 show the accuracy in detecting wrong-side driving along with false positives (FPs) and false negatives (FNs) using the pre-trained networks, VGG and AlexNet, during daytime and nighttime, respectively. False positives (FPs) represent cases when the vehicle is traveling in the correct direction (with the flow of the traffic) but predicted as going in the wrong direction. Conversely, false

Table 13. Wrong-side Driving Detection During Daytime

	FPs (without TF)	FNs (without TF)	Accuracy (without TF)	FPs (with TF)	FNs (with TF)	Accuracy (with TF)
AlexNet-4096	2.1%	13.1%	90.5%	2.0%	11.7%	91.8%
AlexNet-300	2.7%	23.1%	84.5%	2.5%	14.2%	89.1%
VGG-25088	2.1%	13.1%	90.5%	1.8%	9.7%	94.6%
VGG-300	2.2%	13.8%	90.3%	1.8%	10.5%	94.3%

Table 14. Wrong-side Driving Detection During Nighttime

	FPs (without TF)	FNs (without TF)	Accuracy (without TF)	FPs (with TF)	FNs (with TF)	Accuracy (with TF)
AlexNet-4096	1.7%	8.4%	93.8%	1.5%	7.3%	95.3%
AlexNet-300	1.9%	9.0%	93.2%	1.7%	7.9%	94.7%
VGG-25088	1.9%	10.1%	91.8%	1.5%	6.8%	95.5%
VGG-300	2.1%	10.3%	91.6%	1.9%	8.5%	93.8%

Table 15. Wrong-side Driving on Merged Data (Day+Night)

	FPs (without TF)	FNs (without TF)	Accuracy (without TF)	FPs (with TF)	FNs (with TF)	Accuracy (with TF)
AlexNet-4096	2.5%	10.1%	92.1%	1.8%	7.9%	94.3%
AlexNet-300	2.4%	15.0%	88.8%	2.7%	9.4%	92.3%
VGG-25088	2.1%	13.3%	90.7%	1.5%	7.7%	96.1%
VGG-300	2.9%	13.3%	90.3%	1.8%	8.6%	94.8%

negatives (FNs) represent cases when a vehicle is traveling in the wrong direction but is predicted as going in the correct direction. The tables also show evaluation on the architecture described in Section 4.2.

It can be seen that the accuracy is generally over 90% in detecting wrong-side driving during both daytime as well as nighttime. Further, when temporal features are aggregated with a window of five frames, the overall accuracy improves by up to about 4%, as shown in Tables 13 and 14.

While the number of false positives is under 3%, most of the misclassifications happen to be due to false negatives, with an FN rate close to 15% across daytime and nighttime. On closer examination, we found that most of the misclassifications happen when there were few or no vehicles in the scene. Some of those cases, corresponding to daytime and nighttime, are shown in Figure 12(c), where even the few vehicles seen are very far away. In such cases, the relevant features could not be obtained, hence the misclassification. However, in such cases, the use of temporal features leads to a reduction in the number of false negatives, as can be seen in Tables 13 and 14.

Besides evaluating daytime and nighttime conditions separately, we also built a classifier that pools together both daytime and nighttime images for wrong-side driving detection. The accuracy of the classifier on the merged data is more than 92% when temporal features are used, as shown in Table 15. The false negatives, too, have fallen below 10%. The fact that accuracy is comparable to the case where daytime and nighttime are considered separately suggests that the relevant features (e.g., the pattern of vehicles around) are independent of the lighting conditions.

7 MOBILE IMPLEMENTATION

We now describe the implementation of DeepLane without temporal features to detect a vehicle's current lane position on an Android platform. Further, we benchmark the performance of DeepLane running on CPU and GPU for various Android phones such as OnePlus 3 (USD 350) and Lenovo Zuk 2 (USD 155). DeepLane has three building blocks: (i) extracting features using pre-trained DNN networks such as AlexNet and VGG, (ii) reducing the dimensionality of the features using PCA to improve performance, and (iii) classifier model to detect lane position.

7.1 DeepLane Implementation on Android

The entire pipeline on Android is written in native C++ code based on OpenCV, TensorFlow, and custom libraries interfaced with Java using JNI wrappers.

1. Porting AlexNet and VGG on Android phones: We have written the AlexNet and VGG architecture shown in Figure 5 using TensorFlow. We save this model as a protocol buffer (PB) file, which can later be used to load the model and visualize the structure of the network.

To run DNN networks on Android, we import the saved model file, *viz.,* the PB file, into our Android app. Currently, the TensorFlow models saved as PB files can run only on NVidia GPUs. Most of the smartphones have Qualcomm-based Adreno GPUs and TensorFlow models on these can only use the CPU. To overcome this, we employ Qualcomm's Snapdragon Neural Processing Engine (SNPE) [7], which is an SDK for running neural network models trained in Caffe or TensorFlow on Snapdragon mobile platforms. SNPE takes the PB file and converts it into a DLC file (Deep Learning Container). The DLC file can now be used by SNPE runtime to run on either CPU or GPU of the smartphone. Thus, for each image, the Android app sends a request to the DLC, which returns with the feature vectors corresponding to the image.

2. Dimensionality reduction: The feature vectors returned are then fed to the PCA for dimensionality reduction and for faster inference. We used OpenCV implementation of PCA in our Android application.

3. Classifier model: We used linear SVM to build our classifier model for lane detection and wrong-side driving. We employed libSVM for Android along with Android NDK library to implement linear SVM model.

7.2 Benchmarking DeepLane on Android

We now present benchmark results of DeepLane using both AlexNet and VGG on CPU and GPU of Android smartphones. We have tested the Android app on multiple smartphones such as OnePlus 3 and Lenovo Zuk Z2, both having Qualcomm Snapdragon 820 chipset with Quad-core CPU and Adreno 530 GPU.

Table 16 shows the model size and time taken for inference on CPU and GPU, for AlexNet-4096 and VGG-25088, on the Lenovo ZUK Z2 phone. As described in Section 4.1, AlexNet has 5 convolutional layers followed by 3 fully connected layers and has 60M parameters. Similarly, VGG16 has 16 layers with 13 convolutional layers and 3 fully connected layers, resulting in 138M parameters. While both the models yield similar accuracy for lane detection and wrong-side driving, the model size and inference time of AlexNet is significantly lower than VGG16 due to the much smaller number of parameters. Thus, DeepLane on Android employs AlexNet as its pre-trained model for feature extraction.

As shown in Table 16, for both AlexNet and VGG16, the inference time drops significantly when the smartphone's GPU is used for feature extraction. For AlexNet-4096 the inference time is 140ms and 38ms when CPU and GPU is used, respectively. Further, the time taken for feature reduction

Table 16. Inference Time for Bottleneck Feature Extraction

	Model Size (MB)	Inference time CPU (ms)	Inference time GPU (ms)
AlexNet-4096	5MB	140	38
VGG-25088	20MB	1,300	340

from 4,096 to 300 using OpenCV PCA implementation is around 10ms. Finally, the time taken for classification using libSVM classifier model is around 30ms.

The DeepLane Android app using AlexNet currently supports processing at 5 frames per second (fps) (i.e., 140ms+10ms+30ms = 180ms per frame) when running on a CPU and can support up to 15fps when GPU on the smartphone is used. Since vehicles are unlikely to be changing lanes very frequently, the supported frame processing rate of 5fps on CPU is suitable for detecting lane position in real-world settings. The CPU usage with the app running was around 20%–25% on both the smartphone models, thus enabling other apps such as navigation to be used concurrently with DeepLane.

8 CONCLUSION

In this article, we have presented DeepLane, a smartphone-based system for real-time lane detection. DeepLane employs deep learning techniques to classify the lane of the vehicle to either the left, middle, or right lane. We have evaluated DeepLane in both developed and developing regions. DeepLane can detect a vehicle's lane position with an accuracy of over 90% in both daytime and nighttime conditions using temporal features, and does not depend on any infrastructure support such as lane markings. We have implemented DeepLane as an Android app and can assist navigation applications with lane-level information.[1]

REFERENCES

[1] BMW USA. 2018. Active Lane Keeping and Traffic Jam Assistant. Retrieved from https://www.youtube.com/watch?v=w24HYJvaCl0.

[2] DC Nation. 2018. Driver turns wrong and gets hit. Retrieved from https://tinyurl.com/y9kkytq2.

[3] National Highway Traffic Safety Administration (NHTSA). 2018. FARS Encyclopedia: People—Drivers. Retrieved from https://www-fars.nhtsa.dot.gov/People/PeopleDrivers.aspx.

[4] Fox Van Allen. 2018. Google Maps 3.0 with Lane Assist. Retrieved from https://www.techlicious.com/blog/google-maps-3-0-lane-assist-uber-savable-maps/.

[5] Stanford Vision Lab. 2018. ImageNet. Retrieved from http://www.image-net.org/.

[6] Los Angeles Times. 2018. Lidar costs $75,000 per car. Retrieved from http://www.latimes.com/business/la-fi-hy-ouster-lidar-20171211-htmlstory.html.

[7] Qualcomm Technologies Inc. 2018. Qualcomm Neural Processing SDK for AI. Retrieved from https://developer.qualcomm.com/software/qualcomm-neural-processing-sdk.

[8] Ministry of Road Transport and Highways, Government of India. 2018. Road accidents in India, 2016. Retrieved from https://tinyurl.com/y7j86kuh.

[9] Shashank Agarwal. 2018. Tyre Killer. Retrieved from https://www.youtube.com/watch?v=L9EZHDYE_e8.

[10] University of Oxford. 2018. Visual Geometry Group Home Page. Retrieved from http://www.robots.ox.ac.uk/~vgg/research/very_deep/.

[11] European Global Navigation Satellite Systems Agency. 2018. World's first dual-frequency GNSS smartphone hits the market. Retrieved from https://www.gsa.europa.eu/newsroom/news/world-s-first-dual-frequency-gnss-smartphone-hits-market.

[12] Ravi Bhandari. 2018. Wrong-side Driving. Retrieved from https://youtu.be/TfwB6kP1ByM.

[13] Joseph Chet Redmon. 2018. YOLO: Real-Time Object Detection. Retrieved from https://tinyurl.com/m9ml6fx.

[1]Sample videos of DeepLane: https://www.youtube.com/playlist?list=PLlJkASwMpdW1Byp1TzYYfLwHd32yMqxKS.

[14] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. 2017. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.* 39, 12 (2017), 2481–2495.

[15] Ravi Bhoraskar, Nagamanoj Vankadhara, Bhaskaran Raman, and Purushottam Kulkarni. 2012. Wolverine: Traffic and road condition estimation using smartphone sensors. In *Proceedings of the COMSNETS*. IEEE, 1–6.

[16] Olivier Canévet and François Fleuret. 2014. Efficient sample mining for object detection. In *Proceedings of the ACML*.

[17] Dongyao Chen, Kyong-Tak Cho, Sihui Han, Zhizhuo Jin, and Kang G. Shin. 2015. Invisible sensing of vehicle steering with smartphones. In *Proceedings of the ACM MobiSys*.

[18] Isha Dua, Akshay Uttama Nambi, Venkat Padmanabhan, and C. V. Jawahar. 2019. AutoRate: How attentive is the driver? In *Proceedings of the FG*. IEEE.

[19] Google. 2018. Snap to Roads API. Retrieved from https://developers.google.com/maps/documentation/roads/snap.

[20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE CVPR*. 770–778.

[21] Albert S. Huang, David Moore, Matthew Antone, Edwin Olson, and Seth Teller. 2009. Finding multiple lanes in urban road networks with vision and Lidar. *Auton. Robots* 26, 2–3 (2009), 103–122.

[22] Minyoung Huh, Pulkit Agrawal, and Alexei A. Efros. 2016. What makes ImageNet good for transfer learning? Retrieved from *arXiv preprint arXiv:1608.08614* (2016).

[23] Yan Jiang, Feng Gao, and Guoyan Xu. 2010. Computer vision-based multiple-lane detection on straight road and in a curve. In *Proceedings of the IASP*. IEEE, 114–117.

[24] ZuWhan Kim. 2008. Robust lane detection and tracking in challenging scenarios. *IEEE Trans. Intell. Trans. Syst.* 9, 1 (2008), 16–26.

[25] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Proceedings of the NIPS*. 1097–1105.

[26] R. Labayrade, J. Douret, J. Laneurit, and R. Chapuis. 2006. A reliable and robust lane detection system based on the parallel use of three algorithms for driving safety assistance. *IEICE Trans. Inf. Syst.* 89, 7 (2006), 2092–2100.

[27] Prashanth Mohan, V. N. Padmanabhan, and R. Ramjee. 2008. Nericell: Rich monitoring of road and traffic conditions using smartphones. In *Proceedings of the ACM SenSys*.

[28] Michael Montemerlo, Jan Becker, Suhrid Bhat, et al. 2008. Junior: The Stanford entry in the urban challenge. *J. Field Robot.* 25, 9 (2008), 569–597.

[29] Maxime Oquab, Leon Bottou, Ivan Laptev, and Josef Sivic. 2014. Learning and transferring mid-level image representations using convolutional neural networks. In *Proceedings of the CVPR*. IEEE, 1717–1724.

[30] Christopher Rose, Jordan Britt, John Allen, and David Bevly. 2014. An integrated vehicle navigation system utilizing lane-detection and lateral position estimation systems in difficult environments for GPS. *IEEE Trans. Intell. Trans. Syst.* 15, 6 (2014), 2615–2629.

[31] Hoo-Chang Shin, H. R. Roth, M. Gao, Le Lu, Ziyue Xu, Isabella Nogues, Jianhua Yao, Daniel Mollura, and Ronald M. Summers. 2016. Deep convolutional neural networks for computer-aided detection: CNN architectures, dataset characteristics. and transfer learning. *IEEE Trans. Med. Imag.* 35, 5 (2016), 1285–1298.

[32] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich, et al. 2015. Going deeper with convolutions. In *Proceedings of the CVPR*.

[33] Ashit Talukder and Larry Matthies. 2004. Real-time detection of moving objects from moving vehicles using dense stereo and optical flow. In *Proceedings of the IROS*, Vol. 4. IEEE, 3718–3725.

[34] Rafael Toledo-Moreo and Miguel A. Zamora-Izquierdo. 2009. IMM-based lane-change prediction in highways with low-cost GPS/INS. *IEEE Trans. Intell. Trans. Syst.* 10, 1 (2009), 180–185.

[35] Yue Wang, Eam Khwang Teoh, and Dinggang Shen. 2004. Lane detection and tracking using B-Snake. *Image Vis. Comput.* 22, 4 (2004), 269–280.

[36] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. 2014. How transferable are features in deep neural networks? In *Proceedings of the NIPS*. 3320–3328.

[37] Chuang-Wen You, Nicholas D. Lane, Fanglin Chen, Rui Wang, Zhenyu Chen, Thomas J. Bao, Yuting Cheng, Lorenzo Torresani Mu Lin, and Andrew T. Campbell. 2013. CarSafe app: Alerting drowsy and distracted drivers using dual cameras on smartphones. In *Proceedings of the ACM MobiSys*.

[38] Bin Yu and Anil K. Jain. 1997. Lane boundary detection using a multiresolution Hough transform. In *Proceedings of the ICIP*, Vol. 2. IEEE, 748–751.

Received March 2019; revised August 2019; accepted August 2019