International Journal of Pattern Recognition
and Artificial Intelligence
Vol. 36, No. 7 (2022) 2250027 (24 pages)

DOI: 10.1142/S0218001422500276

World Scientific
www.worldscientific.com

# Lightweight Hardware Architecture for Object Detection in Driver Assistance Systems

Bhaumik Vaidya*
*Gujarat Technological University*
*Ahmedabad, India*
*vaidya.bhaumik@gmail.com*

Chirag Paunwala
*Electronics and Communication Department*
*Sarvajanik College of Engineering and Technology, Surat, India*
*cpaunwala@gmail.com*

Received 14 August 2021
Accepted 5 February 2022
Published 6 April 2022

Object detection on hardware platforms plays a very significant role in developing driver assistance systems (DASs) with limited computational resources. Object detection for DAS is a multiclass detection problem that involves detecting various objects like cars, auto, traffic lights, bicycles, pedestrians, etc. DAS also requires accuracy, speed, and sensitivity for detecting these objects in various challenging conditions. The lighting and weather conditions pose a serious challenge for accurate object detection for DAS. This paper proposes a speed-efficient and lightweight fully convolutional neural network (CNN) architecture for object detection in adverse rainy conditions. The proposed architecture uses a CNN-based deraining network with a custom SSIM loss function in the object detection pipeline, which can give an accurate performance using limited computational and memory resources. The object detection architecture contains some architectural modifications to the existing single shot multibox detector (SSD) architecture to make it more hardware efficient and improve accuracy on small objects. It uses a trainable color transformation module using $1 \times 1$ convolutions for handling the adverse lighting conditions encountered in DAS. The architecture uses feature fusion and the dilated convolution approach to enhance the accuracy of the proposed architecture on small objects. The datasets available for object detection in DAS are very imbalanced with cars as a predominant object. The class weight penalization technique is used to improve the performance of the architecture on scarcely present objects. The performance of the architecture is evaluated on well-known datasets like Kitti, Udacity, Indian Driving Dataset (IDD), and DAWN. The architecture achieves satisfactory performance in terms of mean average precision (mAP) and detection time on all these datasets. It requires three times fewer hardware resources compared to existing architectures. The lightweight nature of the proposed architecture and modification of CNN architecture with TensorRT allow the efficient implementation on the jetson nanohardware platform for prototyping, which can be integrated with other intelligent transportation systems.

*Corresponding author.

*Keywords*: Driver assistance system (DAS); deraining network, dilated convolution; feature fusion network; structural similarity index measure (SSIM); class weight penalization; Jetson nanodevelopment board.

## 1. Introduction

Object detection is an evolving field that has attracted so many researchers in the last decade. It has immense potential in driver assistance systems (DASs) or autonomous driving solutions in intelligent transportation systems as it provides a low-cost solution compared to Radar and Lidar.[11,16] Object detection for DAS can be challenging as it needs to detect various objects like cars, auto, bicycles, pedestrians, traffic lights, etc. in challenging road and weather conditions. This system can help drivers in following traffic rules and avoid accidents if implemented accurately.

Object detection is not as trivial for computers as it is for humans. It is a complex computer vision problem. Speed, accuracy, and sensitivity are the main requirements for using object detection for DAS.[64] The challenges like occlusion, illumination, motion blur, clutter, rain, fog, etc. can affect the accuracy of object detection in DAS. Recent advancements in deep learning-based algorithms have improved the accuracy of object detection despite these challenges. This advancement is fueled by the availability of large annotated datasets, which can be useful for training these deep learning algorithms. There are datasets like Kitti,[17] Udacity,[55] and IDD,[59] which can be used to train algorithms for object detection in DAS.

Most of these algorithms are resource hungry and require large computational power for achieving good detection accuracy.[43] This prevents their implementation on embedded platforms for real-time inference. These algorithms are designed to process low-resolution images, which require resizing of high-definition images acquired with the latest cameras. Most of the objects in DAS are small in size and occupy a very small portion of the entire image. These objects will not be detected accurately when images are resized to smaller resolutions needed by deep learning architectures.

Given the above limitations, this paper proposes a deep learning architecture for object detection in DAS, which can be implemented on a hardware platform. The rain streaks severely affect the object detection results because of the similarity in structure and orientation of objects with rain streaks. The proposed technique also incorporates a fully convolutional deraining network in the object detection pipeline, which allows object detection in rainy conditions. The proposed architecture is very lightweight and speed-efficient, which allows its implementation on a resource-constrained jetson nanodevelopment platform.

This research paper has the following contributions:

- It has a lightweight feature extractor compared to existing object detection architectures, which allow processing of high-resolution images. It also preserves features of small objects by using shallow architecture, which can help in accurately detecting the small object.

- It incorporates a fully convolutional deraining network in the object detection pipeline, which allows accurate object detection in rainy conditions. It also proposes a modified structural similarity index (SSIM)[62] loss function, which improves the performance of the deraining network.
- It uses a trainable color transformation module using $1 \times 1$ convolution layers for adapting to various lighting conditions encountered in DAS. It also uses concepts of dilated convolution[3] and feature fusion[32] for improving the accuracy of small object detection.
- It proposes a class weight penalization technique for avoiding class imbalance in the dataset used for training.
- The network reduces memory usage and floating-point operations per second (FLOPs) by almost three times compared to similar architectures proposed in the literature with comparable results in terms of accuracy.
- The lightweight nature of the proposed architecture allows efficient implementation on the jetson nanohardware platform.[28] The convolutional neural network (CNN) architecture is further pruned by using TensorRT,[54] which helps in acceleration on jetson nano.

The organization of the paper is as follows: The related work in image deraining and object detection for DAS is summarized in Sec. 2. The proposed deraining and object detection architecture is explained in Sec. 3. The implementation results and performance analysis of the proposed architecture are shown in Sec. 4.

## 2. Related Work

This paper deals with image deraining as well as object detection so this section will summarize all the related work done for both problems. Video-based methods use interframe information and temporal features for removing rain.[3,5] This paper focuses on incorporating deraining architecture in the object detection pipeline so no temporal information can be utilized. Many researchers have proposed methods by considering a rainy image as a linear combination of clear images and rain streaks. They used methods like the gaussian mixture model,[34] representation in the low-rank form,[7] and representation in the sparse form[67] for separating rain streaks from the image. These methods did not perform well in removing rain streaks and object features are also removed when they have the same orientation as rain streaks.

Many deep neural network-based algorithms are proposed, which learn the mapping between rainy images and clean images. The algorithm proposed in Refs. 12 and 13 decomposes the rainy image into a base and a detail layer and tries to learn the mapping between the detail layer and the negative residual. This approach is fairly accurate but they used deep ResNet[23] architecture to learn the mapping, which requires large computational resources and memory. Some lightweight architectures like residual guide FFN[10] and pyramid network[14] are also proposed but they are not very accurate. Recent architectures use variations of generative adversarial networks

to generate deraining images.[34,45] These architectures provide good performance but they are difficult to train and require large datasets. This paper uses a combination of negative residual mapping and lightweight architecture for accurate deraining on a hardware platform with limited computational resources.

There are many algorithms proposed for object detection in DAS. Some of them use the classical machine learning approach, while others use end-to-end deep learning architectures. Pedestrian and vehicle detection is an important part of any DAS. Adaboost with the Haar Cascade[18,60] algorithm has been proposed for pedestrian detection. It gave real-time performance but it was not very accurate. Histogram of oriented gradients (HOG) with support vector machine (SVM)[9,65] has been widely used for accurate pedestrian and vehicle detection in DAS. It requires applying a sliding window at multiple scales on the original image, which slows down the performance. These limitations of machine learning approaches gave rise to deep learning-based solutions.

The first deep learning architecture for object detection was region-based CNN (R-CNN),[19] which significantly improved the performance of object detection in DAS. It required region proposals given by selective search[56] and edge boxes[68] as the first step. CNN was used to extract features from these proposals and SVM was used for classifying these features. It required each proposed region to go through the CNN architecture, which significantly slowed down its performance. The training of CNN and SVM had to be done separately, which also caused computational challenges. This challenge was overcome by fast R-CNN,[20] which used softmax classification instead of SVM. The performance was further improved in faster R-CNN,[50] which removed the requirement of a separate region proposal algorithm and incorporated the region proposal network in the CNN architecture itself. The faster R-CNN gave a very good performance on object detection but it was very bulky that cannot be implemented efficiently on embedded platforms.

The demand for faster inference leads to the development of You only look once (YOLO)[47] and single shot multibox detector (SSD)[36] algorithms, which detected objects by passing the image only once through the CNN architecture without the requirement of an explicit region proposal network. They treated detection as a regression problem that can learn the boundary by regressing through the default bounding boxes. The YOLO algorithm was very fast but less accurate compared to SSD because SSD used multiple feature layers at different scales for prediction. SSD achieved good performance in terms of accuracy and speed for DAS but the performance on small objects was not very good. Many researchers tried to solve this problem by suggesting some architectural modifications. Deconvolutional SSD (DSSD) architecture was proposed, which used the feature fusion concept for improving performance on small objects. BaNET architecture[64] was also proposed, which used dilated convolution for improving performance on small objects. These algorithms also performed poorly when the driving environment is unconstrained like Indian roads. Prajjwal *et al.*[4] proposed a combination of domain-specific classifiers

and effective transfer learning techniques over faster R-CNN, which improved the performance of object detection in unconstrained environments. Anay *et al.*[39] evaluated performances of YOLO-v3,[49] Poly-YOLO,[26] mask R-CNN,[22] Retina-Net,[35] and Faster-RCNN[20] on the IDD and also proposed a few-shot learning method for training on new classes.

Most architectures described above used deep CNN architectures as feature extractors that consume large memory and slow down the speed for computation. Many researchers have proposed lightweight feature extractors, which can improve inference speed and memory utilization. The architectures like MobileNet-SSD,[25] MobileNetv2-SSDLite,[52] Pelee,[63] and Tiny-DSOD[33] used lightweight feature extractors, which allowed implementation on mobile devices with a low frame rate. Riah *et al.*[2] proposed lightweight separable convolution neural network-based architecture for pedestrian detection in DAS, which could be implemented on embedded devices. Huy-Hung *et al.*[44] implemented various recent lightweight deep learning architectures like EfficientDet Lite[37] and Yolov3-Lite[40] on the Jetson TX2 development board. The accuracy of these algorithms was not sufficient for using them in DAS. The proposed architecture tries to use a shallow feature extractor with some architectural modifications, which allow its implementation on the hardware platform and give sufficient accuracy to be used in DAS.

## 3. Proposed Architecture

The proposed architecture will be explained in two separate sections with the first section describing the deraining network and the second section describing object detection architecture.

### 3.1. *Proposed image deraining architecture*

The proposed image deraining architecture is illustrated in Fig. 1.

Most of the earlier neural network architecture tried to learn the mapping between clean and rainy images, which has a large mapping range that can reduce the performance of the CNN architecture. The architecture uses similar concepts as suggested in Ref. 13 for reducing the mapping range from input to output. The rainy image is decomposed into base and detail layers based on prior knowledge of the image. The base layer is obtained by passing the rainy image through a low pass filter, while the detail layer is obtained by subtracting the base layer from the original

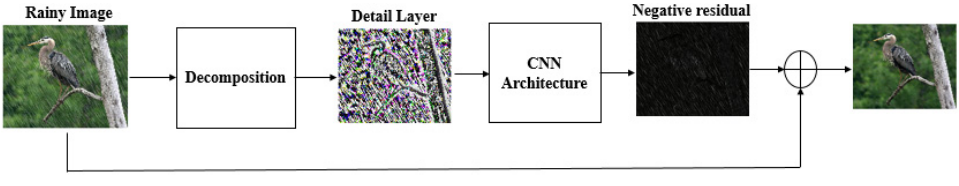


Fig. 1. Proposed deraining architecture.

rainy image. The detail layer will be devoid of any background information and only contain the finer details in the image with most of the pixel values nearer to zero. This detail layer will serve as an input to the trainable CNN architecture. Instead of predicting the clean image directly, the trainable layer tries to predict the difference between the clean image and the rainy image. This difference is called a negative residual, which only requires learning of the rain streaks. This significantly reduces the mapping range for the learnable CNN network, which helps in the improvement of the performance.

The architecture proposed in Ref. 13 used deep ResNet architecture for learning the mapping, which requires large computational resources, which does not serve the purpose of the proposed implementation. So the proposed architecture uses eight-layer simple CNN architecture without any skip connection to learn the mapping between the detail layer and the negative residual. The mathematical formulation for the proposed architecture can be explained with the following equation. The input rainy image is denoted by '$I$' and the output image is denoted by '$O$'.

$$I^{0}_{\text{detail}} = I - I_{\text{base}}, \tag{1}$$

$$I^{l}_{\text{detail}} = \text{RelU}(W^{l} * I^{l-1}_{\text{detail}} + b^{l}) \quad \text{where } l = 1 \text{ to } L-1, \tag{2}$$

$$O_{\text{approx}} = \text{RelU}(W^{L} * I^{L-1}_{\text{detail}} + b^{L}) + I, \tag{3}$$

where '$L$' indicates the total number of layers, '$W^{l}$' is the weights at each layer, and '$b^{l}$' is the biases. '$*$' indicates convolution operation at each layer. The kernel size at each layer is taken as $3 \times 3$ with the number of feature maps equal to 32. The training of the proposed architecture is done on patches with a size equal to $128 \times 128$. The output of each convolution layer is passed through batch normalization for faster training and the ReLU activation layer for nonlinearity. As described earlier, the trainable network learns the negative residual between clear and rainy images so the original rainy image is added to the output of the last layer for obtaining the final clean image. We started with using MSE as a loss function for training the architecture as suggested in Ref. 13. The function is given in the equation below:

$$L = \sum_{i=1}^{N} \| O_{\text{approxi}} - O_i \|_F^2. \tag{4}$$

The equation indicates that loss function is the summation of the Frobenius norm between the predicted image ($O_{\text{approxi}}$) and clean image ($O_i$). The accuracy of the proposed deraining architecture is measured by using SSIM as a performance metric.[62] SSIM is a very good indicator for identifying the structural similarity between two images.[62] SSIM is a matrix that gives a quantitative idea about the perceptual quality of distorted images with respect to the reference image. SSIM is better at replicating the human tendency of identifying structural information and using that for differentiating between two images. It uses luminance, contrast, and structure

features for calculation. The mathematical equation for calculating SSIM is shown below.

$$\mathrm{SSIM}(x, y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)}, \tag{5}$$

$$L = 1 - \mathrm{SSIM}(x, y), \tag{6}$$

where '$x$' is a distorted image, '$y$' is a reference image. '$\mu$' indicates the mean operation of intensity values, and '$\sigma$' is a standard deviation. '$C_1$' and '$C_2$' are numeric constants that are used for the mathematical stability of the equation. This paper proposes to use the modified SSIM function as an objective function for training the deraining architecture, which is indicated by 'L'. SSIM needs to be maximized while training the network and it is in the range of 0 to 1. So 1 – SSIM can be used as a loss function, which will be minimized while training the architecture. The use of SSIM significantly improves the performance of deraining architecture, which will be discussed in the implementation result section of the paper.

### 3.2. *Proposed object detection architecture*

The proposed object detection architecture is illustrated in Fig. 2.

The proposed object detection architecture is derived from the concept of SSD[36] architecture as SSD provides a nice tradeoff between accuracy and speed. It is similar to the architecture proposed in Ref. 58, which was used for traffic sign detection with minor modifications in dilated convolution and feature fusion layers. The architecture uses feature maps of the last four convolution layers at different scales for object detection and classification. The nonmaximum suppression block is used to filter out multiple boxes at the same location along with detections with low confidence. Some architectural changes are proposed by taking the end goal of implementing the object detection pipeline on an embedded platform with sufficient accuracy. The trainable color transformation technique[41] is introduced as the first few layers of the architecture, which decide the best color channels needed for the given application. This technique avoids choosing between different color spaces and improves the performance in various illumination conditions.

The performance of SSD architecture was not good for small objects.[36] There were a couple of reasons for that. It uses VGG architecture as a base feature extractor,

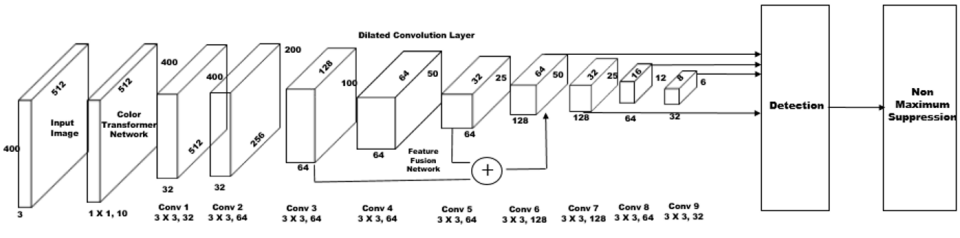


Fig. 2. Proposed object detection architecture.

which is a very deep architecture. The receptive field of deep layers is very large in the original image, which is not able to represent small objects. The careful consideration of image size, object size, and receptive field calculation leads to the development of a feature extractor with five convolutional layers, dilated convolution in the fourth layer, and feature fusion layer. The dilated convolution technique helps in increasing the receptive field size without any increase in trainable parameters, which helps in detecting small objects in shallow networks.[64] The feature fusion module[32] helps in detecting small objects, which will be explained in the next section. The proposed architecture uses exponential rectified linear unit (EReLU)[8] as an activation function after all the convolutional layers.

The choice of scale and aspect ratios for default bounding boxes play a very important role in the performance of the architecture. They are chosen based on the size of the feature map, resolution of the images, and size of objects in the dataset. The proposed architecture uses six default bounding boxes with aspect ratios of [0.20, 0.33, 0.5, 1.0, 1.0, and 2.0] at each prediction layer. The architecture has four prediction layers with scale values of [0.08, 0.16, 0.32, 0.64, and 0.96]. The current feature map scale and next feature map scale are used to decide the scale for the second bounding box with the aspect ratio of 1.

The architecture uses classical cross-entropy loss with hard negative mining and class weight penalization technique, which can avoid the class imbalance in positive and negative boxes and classes in the dataset. The proposed architecture is targeted towards an embedded platform so hardware optimization techniques using TensorRT[54] are performed for better performance. It is explained in the next section. Various concepts related to the proposed architecture are explained in the following sections.

#### 3.2.1. *Feature fusion network*

It has been observed that shallow layers in deep CNN architectures have rich detail information related to an image, while deep layers have more semantic feature information.[57] So, it can be concluded that features of shallow layers can help significantly in detecting small objects, while deeper layers can help in large objects. It was observed during the visualization of learned feature maps in the proposed architecture that the conv3 layer had rich feature details for objects that have the size of 1–5% compared to the original image. If the conv3 layer is directly used as the prediction layer, then it will increase the computation time for default bounding boxes, which will slow down the performance. So, the feature fusion[32] technique, which allows fusion of different features maps, is used for improving the performance of architecture on small objects. The layers in the modules are as shown in Fig. 3.

The features can be fused by either concatenating maps along channel dimensions or summing up the feature maps. The dimensions of the feature maps are made equal before fusing by using the concept deconvolution,[15] which converts the feature map

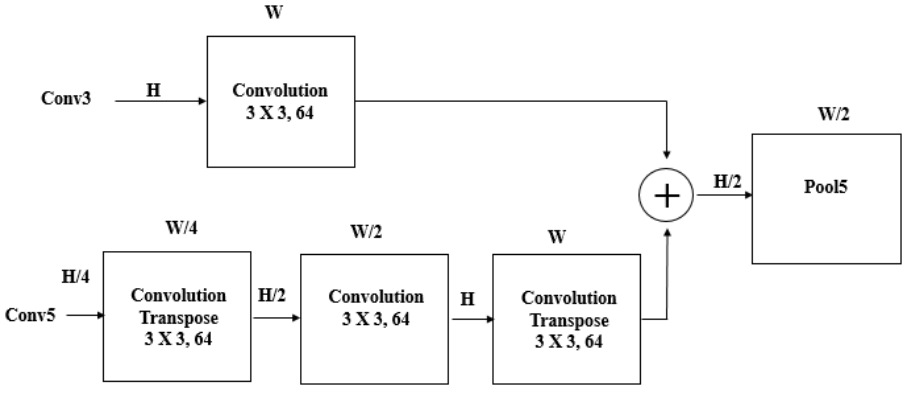


Fig. 3. Feature fusion network.

to a high dimension space. The proposed method used transposed the convolution layer for up-sampling before concatenation along the channel dimensions, which preserves the features of both layers. The max-pooling layer is added at the last to reduce the feature map dimensions, which can help in the reduction of computational requirements.

### 3.2.2. *Data augmentation techniques*

The motivation behind using various data augmentation techniques to train the proposed architecture is to allow the model to be robust in real-life conditions. The translation augmentation is applied, which randomly shifts the image vertically as well as horizontally by a few pixels. The brightness of the image is varied randomly to make the model more robust to lighting changes. The scaling is applied, which randomly scales the image by a factor of 0.7 to 1.4, which allows size invariance. Some images are randomly flipped, which allows the model to perform well in case of rotation in objects.

### 3.2.3. *Loss function with hard negative mining and class weight penalization*

The loss function for the proposed architecture consists of summation between L1 loss for localization and cross-entropy or softmax loss for classification.[36] Most of the default bounding boxes will not contain any objects, which tilt the bias of the detection algorithm. To overcome that, this paper also uses a hard negative mining technique but with a ratio of negative to positive examples as 2:1. It improved accuracy by almost 1% compared to the 3:1 value taken in the original paper. The second type of imbalance can occur when the dataset contains more examples of some classes and very few examples of other classes. All datasets used for DAS contain many examples of cars compared to other classes. The class weight penalization technique is used to overcome this effect. It assigns weights to each class, which is learned from the dataset. The loss on classes with fewer examples is given

more weightage compared to a class with more examples. The weighted cross-entropy loss[24] is used for optimization, which gives better performance compared to the original loss function. The equation for weighted cross-entropy is given below:

$$L = -\frac{1}{M}\sum_{k=1}^{K}\sum_{m=1}^{M} w_k * y_m^k * \mathrm{Log}(\hat{y}_m^k), \tag{7}$$

where $M$ is the number of training examples, $K$ is the number of classes, $w_k$ is a weight for class $k$, $y$ is the true label l, and $\hat{y}$ is the predicted label.

#### 3.2.4. *Acceleration on the Jetson nanoembedded platform*

The goal of this paper is to implement a prototype of the proposed architecture on a hardware platform that can be integrated with any intelligent transportation system. It is imperative for DAS that videos are processed on the edge only rather than transmitting to the cloud and receiving the decision. There are boards like jetson tx1,[27] tx2,[27] nano,[27] google coral,[21] and raspberry pi[46] available in the market, which can help in the edge deployment of deep learning architecture. The jetson nanodevelopment board was chosen as it provides the best tradeoff between cost and computational power.[28] It has 128 core Maxwell GPU and an ARM CPU running at 1.43 GHz. The computational power is 472 GFLOPs with a RAM of 4 GB.[28]

The proposed architecture can be directly deployed on jetson nano but this paper uses the TensorRT runtime optimization engine for high-performance inference.[54] It converts model parameters to the floating-point 16-bit resolution, which helps in the reduction of memory and power. This allows faster inference time. There are some modifications done in the computational graph of the CNN. The elimination of unused output layers and fusion of convolution, bias, and relu operation are done to speed up the computation in CNN. The use of TensorRT[54] allowed faster and memory-efficient implementation of the proposed architecture on Jetson nano.

## 4. Implementation and Results

The deraining object detection architecture is implemented and tested on a computing platform with 8 GB of RAM and an Intel i5 processor running at a frequency of 2.5 GHz. The computing platform also has an NVIDIA GeForce 940 GPU with 4 GB of dedicated GPU memory, which helps in the acceleration of the proposed architecture. This section will contain the results for deraining and object detection architectures individually and then the combined results for end-to-end architecture.

### 4.1. *Implementation results for the deraining network*

It is very difficult to obtain the pair of rainy and clean images from real word to train the model so the proposed deraining architecture was trained on a dataset provided by authors of Refs. 12 and 13, which have randomly chosen images from UCID dataset[53] with rain streaks added in random orientations to mimic real-life rain

scenarios using Photoshop. The training dataset is enhanced by using rain streak added images of the IDD. The architecture is trained using randomly selected $128 \times 128$ patches from training images for three million iterations with a batch size of 16. ADAM[31] optimizer is used to train the model with the starting learning rate of 0.1, which is reduced by a factor of 10 after every million iterations. The models were trained using MSE loss initially and then trained using custom SSIM loss for performance comparison. The performance of the model is compared qualitatively using visual inspection and quantitatively with SSIM and PSNR. The architecture is also implemented on Jetson nano. The inference time on laptop GPU and Jetson nano are compared with other architectures designed with similar objectives.

Initially, we will see the qualitative and quantitative performance of three synthesized rainy images from the test data. The qualitative results are shown in Fig. 4 below.

As can be seen from the figure that the results using the Discriminative sparse coding[38] method contain lots of rain streaks, while the method using layer priors[34] also leaves behind some rain artifacts. Both of these methods use low-level image features, which are not able to remove rain completely. The deep detail network in Ref. 18 uses deep Resnet architecture so it performs exceedingly well on these images. The proposed method is also able to remove rain streaks and artifacts from the images completely while preserving details from the image that is useful for object detection. The quantitative comparisons in terms of SSIM for these images and average SSIM on test data of 1400 images are shown in Table 1.

The performance of the proposed architecture improves if we use custom SSIM loss compared to MSE loss. It performs well compared to the method in the Layer prior method[34] and DSC.[38] The weaker performance compared to the method using DDN-ResNet[13] can be attributed to the lighter architecture used for embedded platforms. It only uses eight convolutional layers as compared to more than 50 used in DDN-ResNet.[13] The performance of the architecture is further evaluated in terms

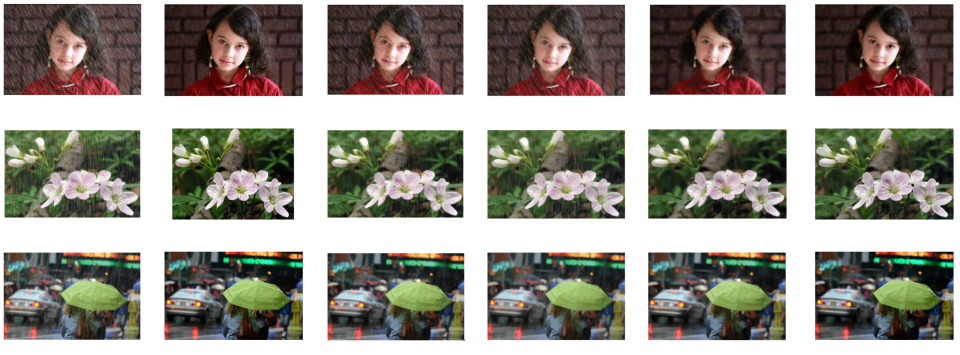


Fig. 4. Qualitative comparison of different image deraining methods.

Table 1. SSIM comparison on test images.

| Images | DSC[38] | Layer Priors[34] | Resnet[13] | Proposed Method with MSE Loss | Proposed Method with SSIM Loss |
|---|---|---|---|---|---|
| Umbrella | 0.80 | 0.82 | 0.86 | 0.862 | 0.912 |
| Flower | 0.77 | 0.81 | 0.92 | 0.819 | 0.844 |
| Girl | 0.71 | 0.80 | 0.90 | 0.812 | 0.899 |
| Test Data | $0.78 \pm 0.12$ | $0.87 \pm 0.07$ | $0.90 \pm 0.05$ | $0.87 \pm 0.04$ | $0.88 \pm 0.04$ |

Table 2. Quantitative comparison between various deraining architectures.

| | ID[29] | AMNF[30] | DSC[38] | CCRR[66] | Resnet[13] | U-net GAN[45] | Proposed Method |
|---|---|---|---|---|---|---|---|
| SSIM | 0.82 | 0.85 | 0.87 | 0.85 | 0.90 | 0.85 | 0.88 |
| PSNR | 22.23 | 22.33 | 25.79 | 25.05 | — | 24.52 | 30.94 |

of PSNR metrics and compared with other well-known approaches for deraining available in the literature. The comparison is shown in Table 2.

The proposed architecture performs well in terms of both metrics. The PSNR value for the method in Ref. 13 is not available. The performance in terms of execution time for various image sizes is done to see the effect of lighter architecture. It is shown in Table 3 below.

The proposed architecture significantly improves the performance in terms of execution time on Desktop GPU. It can be easily implemented on Jetson nano because of its lightweight and the performance on Jetson nano is also better compared to other deraining methods.

### 4.2. *Implementation results of object detection architecture*

The proposed object detection architecture is trained and evaluated on various object detection datasets like Kitti,[17] Udacity self-driving car dataset,[55] and IDD.[59] The results of the proposed architecture are compared with other architectures available in the literature. The training and testing setup for all these datasets is shown in Table 4.

The implementation results of various datasets are shown in separate sections below.

Table 3. Execution time comparison on test images (in seconds).

| Image Sizes | DSC[59] | Layer Priors[34] | Resnet[13] | Proposed Work [NVIDIA Geforce 940] | Proposed Work [Jetson nano] |
|---|---|---|---|---|---|
| $250 \times 250$ | 54.9 | 169.6 | 0.2 | 0.035 | 0.12 |
| $500 \times 500$ | 189.3 | 674.8 | 0.3 | 0.14 | 0.46 |
| $750 \times 750$ | 383.9 | 1468.7 | 0.5 | 0.303 | 1.02 |

Table 4. Training setup for object detection architecture.

| | Kitti[17] | Udacity[55] | IDD[59] |
|---|---|---|---|
| No. of training images | 7481 | 18727 | 31569 |
| No. of testing images | 501 | 4341 | 10255 |
| Image size | $400 \times 1200$ | $400 \times 512$ | $400 \times 512$ |
| No. of classes | 8 | 6 | 15 |

#### 4.2.1. *Results on the Kitti dataset*

The Kitti[17] is a widely used object detection dataset in the literature that has eight different object classes. The architecture is trained for 20 epochs using the ADAM optimizer.[31] The object detection results on various test images are shown in Fig. 5 below.

As can be seen from the figure, the architecture can detect objects that are occluded as well as objects that are small in size. The performance is measured by using Mean Average Precision (mAP) as a performance metric for easy, moderate, and hard detections. The easy detection is defined by a minimum 40 pixels bounding box height with no occlusion and maximum 15% truncation. Moderate detection is defined by a minimum 25 pixels bounding box height with partial occlusion and maximum 30% truncation. Hard detection is defined by a minimum 25 pixels bounding box height with a difficult occlusion level and maximum 50% truncation. The values of mAP for all classes are shown in Table 5.

Fig. 5. Object detection results of the proposed architecture on the Kitti dataset.

Table 5. Average precision of all classes in the Kitti dataset (in %).

| Class | Easy | Moderate | Hard |
|---|---|---|---|
| Car | 91.59 | 90.17 | 84.73 |
| Pedestrian | 75.07 | 72.29 | 62.71 |
| Truck | 93.01 | 92.23 | 85.09 |
| Cyclist | 80.46 | 78.78 | 61.18 |
| Tram | 93.62 | 92.45 | 81.92 |
| Van | 81.62 | 81.3 | 63.98 |
| Person-sitting | 71.03 | 70.29 | 59.88 |
| Misc | 74.58 | 73.19 | 43.41 |
| mAP | 82.62 | 81.39 | 66.73 |

The architecture can give high precision for all classes even though the training examples are very low for some of the classes due to data augmentation and class weight penalization techniques. This performance is compared with other well-known architectures in the literature. The comparison is shown in Table 6 below. Few of the results of other methods are taken from Ref. 43.

As can be seen from the Table, the architecture performs well on easy, moderate, and hard examples. Some architectures perform better than the proposed architecture, which can be attributed to the complex or deep nature of those architectures, which will make them hard to deploy on hardware platforms. The proposed architecture is designed to be efficient on embedded platforms, which can be proven by comparing the computational complexity of the architecture with other architectures, as shown in Table 7. Some results of other methods are taken from Refs. 43 and 44.

The computational complexity is compared in terms of overall memory size, floating points operation per second, and the number of frames processed per second. The architecture consumes three times lower memory compared to other architectures. There is almost a 3× reduction in floating-point operations per second, which signifies the lightweight nature of the architecture. The architecture takes around 0.06 s to process a single image on laptop GPU, which amounts to processing around 17 frames per second. This architecture was also deployed on Jetson nano

Table 6. Average precision comparison between different architectures.

| | Average Precision (%) | | | | | |
|---|---|---|---|---|---|---|
| | Vehicle | | | Pedestrian | | |
| Method | Easy | Moderate | Hard | Easy | Moderate | Hard |
| Faster R-CNN[20] | 86.71 | 81.84 | 71.12 | 76.21 | 62.14 | 60.33 |
| SSD[47] | 77.71 | 64.06 | 56.17 | 25.12 | 18.20 | 16.21 |
| YOLO V2[48] | 76.79 | 61.31 | 50.25 | 22.16 | 16.16 | 15.82 |
| MS-CNN[6] | 90.03 | 89.02 | 76.11 | 84.12 | 74.98 | 63.48 |
| ShuffleNetv2[43] | 88.46 | 84.31 | 73.12 | 80.63 | 72.18 | 64.02 |
| Proposed method | 88.06 | 87.06 | 75.38 | 75.07 | 72.29 | 62.71 |

Table 7. Performance comparison between different architectures in terms of computational complexity.

| Method | Model Size (MB) | GFLOPs | FPS | |
|---|---|---|---|---|
| | | | Laptop GPU | Jetson TX2/Nano |
| Faster R-CNN[20] | 520 | 150 | 1 | 0.8 |
| SSD[47] | 35 | 1.2 | 3 | 8 |
| ShuffleNetv2[43] | 30 | 1.12 | 17 | 11 |
| EfficientDet-Lite[44] | 60 | 48 | — | 1.7 |
| Yolo3-tiny[44] | 35 | 33 | — | 10.3 |
| Yolo3-tiny-3I[44] | 36 | 43 | — | 8.0 |
| Proposed method | 10.68 | 0.399 | 17 | 7 |

where it could process around 7 frames per second. The other architectures in the literature were deployed on Jetson tx2, which has almost three times compute capacity compared to Jetson nano so if the proposed architecture is implemented on Jetson tx2 it will give comparable or better performance compared to other architectures in the literature.

The architecture is also evaluated on other datasets to validate its performance. This is explained in the next two sections.

#### 4.2.2. *Results on the Indian driving dataset*

This dataset is specially built by taking Indian driving scenarios into account along with objects that are only encountered while driving on Indian roads.[59] The dataset has 15 different classes of objects. The architecture is trained for 15 epochs using the ADAM optimizer.[31] The object detection results on various test images are shown in Fig. 6.

The IDD has many classes compared to the other two datasets with classes like motorcycles and autorickshaws, which are mostly seen on Indian roads. The architecture can detect most of the objects even with random orientations and occlusions. The performance is calculated by measuring mAP for classes in the test set. The performance of the architecture is compared with other architectures on the same dataset in Table 8.

The performance of the proposed architecture is evaluated for two scenarios. The version 1 architecture is simple CNN architecture without feature fusion, dilated convolution, and class weights in the loss function.

The second version includes these two modifications. The addition of feature fusion and class weights improves the performance for almost all the classes in the dataset. The proposed architecture performs better than most of the architecture. It performs poorly compared to the Retinanet and faster R-CNN variant, which can be attributed to the complex nature of their architecture, which makes them computationally inefficient on embedded platforms.


Fig. 6. Object detection results of proposed architecture on IDD.

Table 8. Performance comparison of different architectures on IDD.

| Architecture | mAP (%) |
|---|---|
| YOLO-V3 with DarkNet-53[26] | 11.7 |
| Poly-YOLO with SE-DarkNet[26] | 15.2 |
| Mask R-CNN with ResNet-50[26] | 17.5 |
| Retina-Net[39] | 22.1 |
| Faster R-CNN[4] | 18.45 |
| Faster R-CNN with domain transfer[4] | 24 |
| Proposed architecture (version 1) | 16.42 |
| Proposed architecture (version 2) | 19.31 |

#### 4.2.3. *Results on the Udacity self-driving car dataset*

The Udacity self-driving dataset[55] contains images of five different classes, which includes car, pedestrians, truck, bicyclist, and traffic light. The autorickshaw images in Indian conditions[1] were combined with the Udacity dataset to add a sixth class in this dataset. The architecture is trained for 25 epochs using an ADAM optimizer.[31] The object detection results on various test images are shown in Fig. 7.

As can be seen from Fig. 7, the architecture can detect objects with various sizes and aspect ratios. It can detect some of the traffic lights but its performance is not

Fig. 7. Object detection results of proposed architecture on the Udacity dataset.

good as car class because there are very few training annotations provided for the traffic light class. The performance is calculated by measuring mAP for images in the test data. The values of mAP for all classes are shown in Table 9 below.

The performance is again measured for both versions of the architecture. The second version improves the mAP values by almost 3% with an almost 8% increase in the Pedestrian class, which has a small size and fewer examples in the training set.

#### 4.2.4. *Analyzing detection failure on images*

The proposed object detection algorithm fails to detect objects in certain cases. This section analyzes a few of these scenarios by visual inspection. The images on which the object detection algorithm fails are displayed in Fig. 8. The failed object is surrounded by a green rectangle.

The motorcycle on the top-left image is not detected because of a large amount of truncation. The algorithm also fails to detect sitting or truncated pedestrians, as

Table 9. Average precision of various classes in the Udacity dataset for different architectures.

| Architecture | Car | Truck | Pedestrian | Autorickshaw | mAP |
|---|---|---|---|---|---|
| Version 1 | 74 | 22 | 19 | 77 | 48.09 |
| Version 2 | 76 | 24 | 27 | 79 | 51.47 |

Fig. 8. Object detection failure in the proposed architecture.

shown in the top-right as well the bottom-left image. When the object is very far away from the camera, then the algorithm fails to detect the object, as shown in the bottom-right image. So, it can be concluded that the performance of the algorithm is not good when there is a large amount of truncation or deformation or the size of an object is very small.

### 4.3. *Implementation results of the combined architecture*

The rain removal and object detection architectures are concatenated for the overall detection of objects in rainy conditions. The weights of the rain removal network are fine-tuned by training them on images from IDD with added rain streaks. The detection results for combined architecture are shown in Fig. 9.

Dense streaks are added to images in IDD compared to the earlier UCID dataset to simulate real-time rainy conditions. The output of the deraining network and object detection network is also in the figure. The proposed object detection architecture without a deraining network achieved mAP of 12.36% on the rainy images, which increased to 18.1% after incorporating a deraining network. The combined architecture achieved similar performance in rainy conditions as compared to without rain conditions. The performance of the combined architecture is compared with other architectures in the literature by measuring its performance on the

(a) Rainy Image (b) Derained image (c) Object detection results

Fig. 9. Object detection results on rainy images.

Table 10. Performance comparison of proposed architecture on the DAWN dataset.

| Architecture | mAP (%) |
|---|---|
| Resnet50 with standard data augmentation[51] | 23.3 |
| Resnet50 with AMDA[51] | 25.55 |
| Resnet50 with Ensemble AMDA[51] | 25.8 |
| RoHL[51] | 24.9 |
| RetinaResnet50 with affirmative voting strategy[31] | 32.75 |
| Proposed architecture | 25.41 |

DAWN dataset,[42] which has a collection of 1000 images for real traffic conditions in adverse weather conditions.

It can be observed from Table 10 that the performance of the proposed architecture is comparable in terms of detection accuracy to the other architectures in the literature with a significant reduction in computation requirement and memory usage. The architectures in Refs. 51 and 61 used Resnet50 as the base architecture, which itself has a 98 MB of memory size. They also use an ensemble approach and voting strategy for arriving at the final detection result, while the proposed architecture only uses one network for prediction, which makes it speed efficient. The proposed combined architecture takes around 0.11 s to process the entire image. The overall architecture takes around 12 MB of storage and about one million trainable parameters, which makes it ideal for implementation on embedded platforms.

## 5. Conclusion and Future Work

Object detection can play a very crucial role in any DAS, which can be integrated with an intelligent transportation system. It is a complex computer vision problem with challenges like illumination, size of the object, adverse weather conditions, occlusion, and clutter affecting the performance of the system. The implementation

of the system on hardware is even more difficult as accurate object detection algorithms based on deep learning require large computational resources and memory. This paper proposes a speed-efficient and lightweight CNN architecture for object detection, which is implemented on a Jetson nanoembedded platform. It also proposes the integration of a deraining network in the object detection pipeline, which improves the performance of the system in rainy conditions. The proposed deraining network is a shallow computationally efficient network that gives an equivalent performance to a deep neural network with a modified SSIM loss function. The proposed object detection architecture has a shallow feature extractor compared to the original SSD with feature fusion and dilated convolution concepts, which improve the performance on small objects. The performance of the architecture is evaluated on well-known object detection datasets and it achieves good detection accuracy for all of them. The architecture is computationally very efficient with a three times reduction in memory usage and FLOPs. The overall object detection pipeline is evaluated on synthetically added rain streaks on the IDD dataset, which takes around 0.11 s for inference and 12 MB of memory usage. The performance of the proposed architecture can be improved further in the future on the IDD dataset in terms of detection accuracy by making further architectural modifications.

## References


1. Autorickshows Detection Dataset, Available at http://cvit.iiit.ac.in/autorickshaw_detection/.
2. R. Ayachi, Y. Said and A. B. Abdelaali, Pedestrian detection based on light-weighted separable convolution for advanced driver assistance systems, *Neural Process. Lett.* **52**(3) (2020) 2655–2668.
3. P. C. Barnum, S. Narasimhan and T. Kanade, Analysis of rain and snow in frequency space, *Int. J. Comput. Vis.* **86**(2) (2010) 256–274.
4. P. Bhargava, On generalizing detection models for unconstrained environments, in *Proc. IEEE/CVF Int. Conf. Computer Vision Workshops* (Seoul, Korea (South), 2019), pp. 4296–4301.
5. J. Bossu, N. Hautiere and J. P. Tarel, Rain or snow detection in image sequences through use of a histogram of orientation of streak, *Int. J. Comput. Vis.* **93**(3) (2011) 348–367.
6. Z. Cai, Q. Fan, R. S. Feris and N. Vasconcelos, A unified multi-scale deep convolutional neural network for fast object detection, in *European Conf. Computer Vision* (Springer, Cham, 2016), pp. 354–370.
7. Y. L. Chen and C. T. Hsu, A generalized low-rank appearance model for spatio-temporally correlated rain streaks, in *Proceedings of 4th International Conference on Learning Representations in Puerto Rico* (2016), pp. 1968–1975.
8. D. A. Clevert, T. Unterthiner and S. Hochreiter, Fast and accurate deep network learning by exponential linear units (elus), arXiv:1511.07289 (2015).
9. N. Dalal and B. Triggs, Histograms of oriented gradients for human detection, in *2005 IEEE Comput. Soc. Conf. Computer Vision and Pattern Recognition (CVPR'05)* (2005), Vol. 1, pp. 886–893.
10. Z. Fan, H. Wu, X. Fu, Y. Hunag and X. Ding, Residual-guide feature fusion network for single image deraining, arXiv:1804.07493 (2018).

11. P. Felzenszwalb *et al.*, Object detection with discriminatively trained part-based models, *IEEE Trans. Pattern Anal. Mach. Intell.* **32**(9) (2009) 1627–1645.
12. X. Fu, J. Huang, X. Ding, Y. Liao and J. Paisley, Clearing the skies: A deep network architecture for single-image rain removal, *IEEE Trans. Image Process.* **26**(6) (2017) 2944–2956.
13. X. Fu, J. Huang, D. Zeng, Y. Huang, X. Ding and J. Paisley, Removing rain from single images via a deep detail network, in *Proc. IEEE Conf. Computer Vision and Pattern Recognition* (2017), pp. 3855–3863.
14. X. Fu, B. Liang, Y. Huang, X. Ding and J. Paisley, Lightweight pyramid networks for image deraining, *IEEE Trans. Neural Netw. Learn. Syst.* **31**(6) (2019) 1794–1807.
15. C. Y. Fu, W. Liu, A. Ranga, A. Tyagi and A. C. Berg, Dssd: Deconvolutional single shot detector, arXiv:1701.06659 (2017).
16. H. Gao, C. Liu, L. Youhuizi and Y. Xiaoxian, V2VR: Reliable hybrid-network-oriented V2V data transmission and routing considering RSUs and connectivity probability, *IEEE Trans. Intell. Trans. Syst.* **22**(6) (2020) 3533–3546.
17. A. Geiger, P. Lenz and R. Urtasun, Are we ready for autonomous driving? The kitti vision benchmark suite, in *2012 IEEE Conf. Computer Vision and Pattern Recognition* (IEEE, 2012), pp. 3354–3361.
18. D. Gerónimo, A. D. Sappa, A. López and D. Ponsa, Adaptive image sampling and windows classification for on-board pedestrian detection, in *Int. Conf. Computer Vision Syst.* (2007).
19. R. Girshick, J. Donahue, T. Darrell and J. Malik, Region-based convolutional networks for accurate object detection and segmentation, *IEEE Trans. Pattern Anal. Mach. Intell.* **38**(1) (2015) 142–158.
20. R. Girshick, Fast r-cnn, in *Proc. IEEE Int. Conf. Computer Vision* (2015), pp. 1440–1448.
21. Google Coral Board, https://coral.ai/products/dev-board/.
22. K. He, G. Gkioxari, P. Dollár and R. Girshick, Mask r-cnn, in *Proc. IEEE Int. Conf. Computer Vision* (2017), pp. 2961–2969.
23. K. He, X. Zhang, S. Ren and J. Sun, Deep residual learning for image recognition, in *Proc. IEEE Conf. Computer Vision and Pattern Recognition* (2016), pp. 770–778.
24. Y. Ho and S. Wookey, The real-world-weight cross-entropy loss function: Modeling the costs of mislabeling, *IEEE Access* **8** (2019) 4806–4813.
25. 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:1704.04861 (2017).
26. P. Hurtik, V. Molek, J. Hula, M. Vajgl, P. Vlasanek and T. Nejezchleba, Poly-YOLO: higher speed, more precise detection and instance segmentation for YOLOv3, arXiv:2005.13243 (2020).
27. Jetson Modules, https://developer.nvidia.com/embedded/jetson-modules.
28. Jetson Nano Development Board Help Document, https://developer.nvidia.com/embedded/learn/get-started-jetson-nano-devkit.
29. L. W. Kang, C. W. Lin and Y. H. Fu, Automatic single-image-based rain streaks removal via image decomposition, *IEEE Trans. Image Process.* **21**(4) (2011) 1742–1755.
30. J. H. Kim, C. Lee, J. Y. Sim and C. S. Kim, Single-image deraining using an adaptive nonlocal means filter, in *2013 IEEE Int. Conf. Image Process.* (IEEE, 2013), pp. 914–917.
31. D. P. Kingma and J. Ba, Adam: A method for stochastic optimization, arXiv:1412.6980 (2014).
32. J. Leng and Y. Liu, An enhanced SSD with feature fusion and visual reasoning for object detection, *Neural Comput. Appl.* **31**(10) (2019) 6549–6558.

33. Y. Li, J. Li, W. Lin and J. Li, Tiny-DSOD: Lightweight object detection for resource-restricted usages, arXiv:1807.11013 (2018).
34. Y. Li, R. T. Tan, X. Guo, J. Lu and M. S. Brown, Rain streak removal using layer priors, in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (Las Vegas, NV, USA, 2016), pp. 2736–2744.
35. T. Y. Lin, P. Goyal, R. Girshick, K. He and P. Dollár, Focal loss for dense object detection, in *Proc. IEEE Int. Conf. Computer Vision* (2017), pp. 2980–2988.
36. W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C. Y. Fu and A. C. Berg, Ssd: Single shot multibox detector, in *Eur. Conf. Computer Vision* (Springer, Cham, 2016), pp. 21–37.
37. R. Liu, Higher accuracy on vision models with EfficientNet-Lite. TensorFlow Blog. Available at https://blog.tensorflow.org/2020/03/higher-accuracy-on-visionmodels-with-efficientnet-lite.html [Accessed 30 Apr. 2020].
38. Y. Luo, Y. Xu and H. Ji, Removing rain from a single image via discriminative sparse coding, in *Proc. IEEE Int. Conf. Computer Vision* (2015), pp. 3397–3405.
39. A. Majee, K. Agrawal and A. Subramanian, Few-shot learning for road object detection, arXiv:2101.12543 (2021).
40. R. Mehta and C. Ozturk, Object detection at 200 frames per second, in *Proc. European Conf. Computer Vision (ECCV) Workshops* (Munich, Germany, 2018), pp. 659–675.
41. D. Mishkin, N. Sergievskiy and J. Matas, Systematic evaluation of convolution neural network advances on the imagenet, *Comput. Vis. Image Understand.* **161** (2017) 11–19.
42. K. Mourad, "DAWN", Mendeley Data, V3 (2020), doi: 10.17632/766ygrbt8y.3.
43. H. Nguyen, Real-time vehicle and pedestrian detection on embedded platforms, *J. Theor. Appl. Inf. Technol.* **98**(21) (2020) 3405–3415.
44. H. H. Nguyen, D. N. N. Tran and J. W. Jeon, Towards real-time vehicle detection on edge devices with nvidia jetson tx2, in *2020 IEEE Int. Conf. Consumer Electronics-Asia (ICCE-Asia)* (IEEE, 2020), pp. 1–4.
45. O. A. Ramwala, C. N. Paunwala and M. C. Paunwala, Image de-raining for driver assistance systems using U-Net based GAN, in *2019 IEEE Int. Conf. Signal Processing, Information, Communication & Systems (SPICSCON)* (2019), pp. 23–26.
46. Rapberry Pi Board, https://www.raspberrypi.org/.
47. J. Redmon, S. Divvala, R. Girshick and A. Farhadi, You only look once: Unified, real-time object detection, in *Proc. IEEE Conf. Computer Vision and Pattern Recognition* (2016), pp. 779–788.
48. J. Redmon and A. Farhadi, YOLO9000: Better, faster, stronger, in *Proc. IEEE Conf. Computer Vision and Pattern Recognition* (2017), pp. 7263–7271.
49. J. Redmon and A. Farhadi, Yolov3: An incremental improvement, arXiv:1804.02767 (2018).
50. S. Ren, K. He, R. Girshick and J. Sun, Faster r-cnn: Towards real-time object detection with region proposal networks, *Adv. Neural Inf. Process. Syst.* **28** (2015) 91–99.
51. T. Saikia, C. Schmid and T. Brox, Improving robustness against common corruptions with frequency biased models (2021), https://arxiv.org/abs/2103.16241.
52. M. Sandler, A. Howard, M. Zhu, A. Zhmoginov and L. C. Chen, Mobilenetv2: Inverted residuals and linear bottlenecks, in *Proc. IEEE Conf. Computer Vision and Pattern Recognition* (2018), pp. 4510–4520.
53. G. Schaefer and M. Stich, UCID: An uncompressed color image database, in *Storage and Retrieval Methods and Applications for Multimedia 2004*. Vol. 5307 (International Society for Optics and Photonics, 2003), pp. 472–480.
54. TensorRT, "https://docs.nvidia.com/deeplearning/frameworks/tf-trt-user-guide/index.html".

55. Udacity self driving car Dataset, https://github.com/udacity/self-driving-car/tree/master/annotations.
56. J. R. Uijlings, K. E. Van De Sande, T. Gevers and A. W. Smeulders, Selective search for object recognition, *Int. J. Comput. Vis.* **104**(2) (2013) 154–171.
57. B. Vaidya and C. Paunwala, Deep learning architectures for object detection and classification, in *Smart Techniques for a Smarter Planet* (Springer, Cham, 2019), pp. 53–79.
58. B. Vaidya and C. Paunwala, Hardware efficient modified CNN architecture for traffic sign detection and recognition, *Int. J. Image Graph.* **21** (2021) 2250017.
59. G. Varma, A. Subramanian, A. Namboodiri, M. Chandraker and C. V. Jawahar, IDD: A dataset for exploring problems of autonomous navigation in unconstrained environments, in *2019 IEEE Winter Conf. Applications of Computer Vision (WACV)* (IEEE, 2019), pp. 1743–1751, https://idd.insaan.iiit.ac.in/dataset/details/.
60. P. Viola, M. J. Jones and D. Snow, Detecting pedestrians using patterns of motion and appearance, *Int. J. Comput. Vis.* **63**(2) (2005) 153–161.
61. R. Walambe, A. Marathe, K. Kotecha and G. Ghinea, Lightweight object detection ensemble framework for autonomous vehicles in challenging weather conditions, *Comput. Intell. Neurosci.* (2021) 1–12.
62. Z. Wang, A. C. Bovik, H. R. Sheikh and E. P. Simoncelli, Image quality assessment: From error visibility to structural similarity, *IEEE Trans. Image Process.* **13**(4) (2004) 600–612.
63. R. J. Wang, X. Li and C. X. Ling, Pelee: A real-time object detection system on mobile devices, arXiv:1804.06882 (2018).
64. X. Xu, J. Zhao, Y. Li, H. Gao and X. Wang, BANet: A balanced atrous net improved from SSD for autonomous driving in smart transportation, *IEEE Sens. J.* **21**(2) (2020) 25018–25026.
65. L. Zhang, B. Wu and R. Nevatia, Pedestrian detection in infrared images based on local shape features, in *2007 IEEE Conf. Computer Vision and Pattern Recognition* (IEEE, 2007), pp. 1–8.
66. H. Zhang and V. M. Patel, Convolutional sparse and low-rank coding-based rain streak removal, in *2017 IEEE Winter Conf. Applications of Computer Vision (WACV)* (IEEE, 2017), pp. 1259–1267.
67. L. Zhu, C. W. Fu, D. Lischinski and P. A. Heng, Joint bi-layer optimization for single-image rain streak removal, in *Proc. IEEE Int. Conf. Computer Vision (ICCV)* (Springer, Venice, Italy, 2017), pp. 2526–2534.
68. C. L. Zitnick and P. Dollár, Edge boxes: Locating object proposals from edges, in *Eur. Conf. Computer Vision* (Springer, Cham, 2014), pp. 391–405.

**Bhaumik Vaidya** has received his B.E. degree in Electronics and Communication from Dharmsinh Desai University, Nadiad, in 2011 and M.E. Degree in VLSI and Embedded Systems from Gujarat Technological University, Ahmedabad in 2013. He is currently pursuing his Ph.D. degree from Gujarat Technological University, Ahmedabad, India. His current research interests include machine learning, deep learning, computer vision, and GPU programming.

**Chirag Paunwala** has received his B.E. degree in Electronics and Communication from South Gujarat University, Surat, in 1999 and M.E. Degree in Digital Techniques from Rajiv Gandhi Technological University in 2006. He has completed his Ph.D. degree from SVNIT, Surat in 2012. He is serving as a professor in EC Department, SCET, Surat. His current research interests include image processing, pattern recognition, computer vision, deep learning, medical signal processing, and machine learning.