International Journal of Pattern Recognition
and Artificial Intelligence
Vol. 36, No. 7 (2022) 2250027 (24 pages)
© World Scientific Publishing Company
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 1×1 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 1×1 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

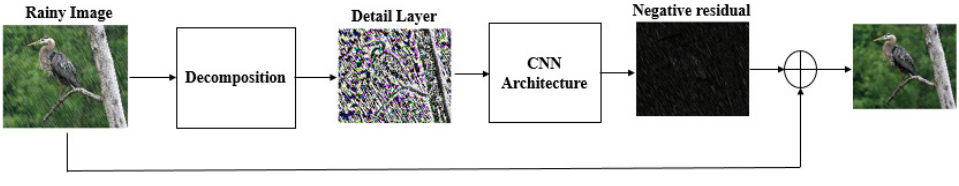


Fig. 1. Proposed deraining architecture.

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 3×3 with the number of feature maps equal to 32. The training of the proposed architecture is done on patches with a size equal to 128×128. 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_{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. 'μ' indicates the mean operation of intensity values, and 'σ' 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,

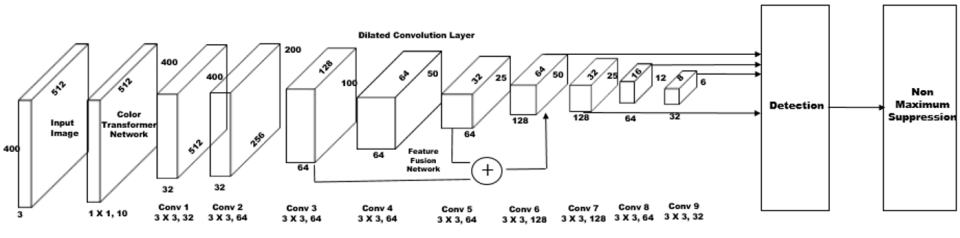


Fig. 2. Proposed object detection architecture.

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

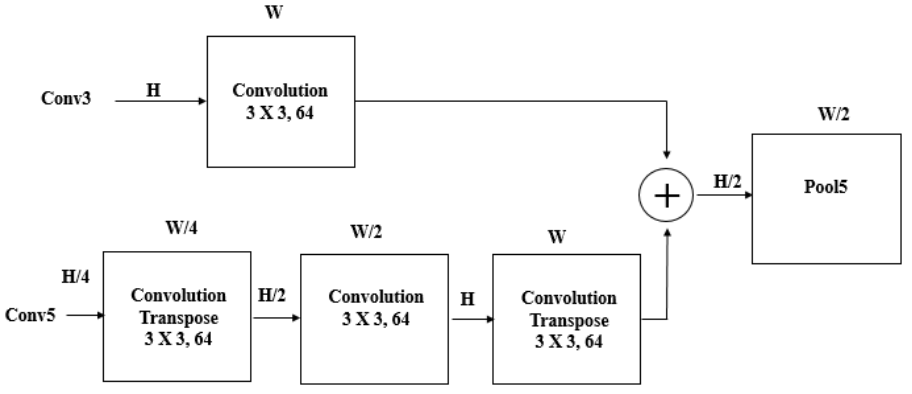


Fig. 3. Feature fusion network.

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 128×128 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

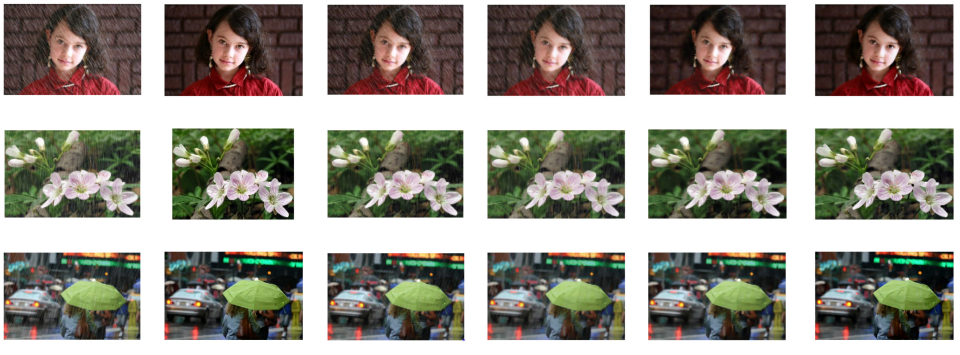


Fig. 4. Qualitative comparison of different image deraining methods.

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 ± 0.12	0.87 ± 0.07	0.90 ± 0.05	0.87 ± 0.04	0.88 ± 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]
250×250	54.9	169.6	0.2	0.035	0.12
500×500	189.3	674.8	0.3	0.14	0.46
750×750	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	400×1200	400×512	400×512
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.

Real Time Object Detection and Tracking

Bhaumik Vaidya and Chirag N. Paunwala
Electronics and Communication Department, Sarvajanik College of Engineering and Technology, Surat, India

Abstract

Object detection and tracking is an ongoing research topic in computer vision that makes efforts to detect, recognize, and track objects through a series of frames. It has been found that object detection and tracking in the video sequence is a challenging task and time consuming process. In this entry, various feature-based and silhouette-based object detection techniques are discussed along with their pros and cons. Object tracking techniques such as absolute difference, background subtraction, optical flow, and mean shift are explained in detail. The challenges in implementing these algorithms on hardware and few implementations of object detection algorithms are discussed in the last part of the article.

Keywords: Background subtraction; FAST; LBP; Object detection; Optical flow; SIFT; Silhouette-based detection; SURF; Tracking.

INTRODUCTION

During the past few years, there has been a rapid advancement in computer vision technologies. One area in computer vision that has attained great focus from researchers around the world is object detection, recognition, and tracking.

In many computer vision systems, object detection is the first step in building a larger system. A lot of information can be derived from the detected object: (i) object can be classified into a particular class, (ii) it can be tracked in image sequence, and (iii) more information about the scene or other object inferences can be derived from the detected object.

Object tracking can be defined as the task of detecting objects in every frame of the video and establishing correspondence between these detected objects from one frame to another in all frames of the video in the video file. The next part of the entry consists of a detailed description of object detection and object tracking methods with their pros and cons and applications.

APPLICATIONS OF OBJECT DETECTION AND TRACKING

Many factors such as the application of machine learning algorithms to solve computer vision problem, the development of new mathematical models for representation, and an increase in the computational power of processors have helped in rapid advancement of object detection applications in recent years.[1]

Object detection and tracking has many applications such as video surveillance system to track suspicious activities, events, persons, or group of persons in real time;[2,3] and intelligent traffic system to track vehicle,[4–7] to detect traffic signs,[8] and visual traffic surveillance (Fig. 1).

Object detection is essential in autonomous robots to give them information about their surroundings and planning for their navigation.[9] It can be used for real-time human–computer interaction based on face and hand gesture recognition;[10] industrial assembly and quality control in production lines;[11] and biomedical image analysis, for example, to detect cervical vertebra in X-ray images, breast cancer, or brain tumor.[12] It can be used in facial expression analysis which includes marketing, perceptual user interfaces, human–robot interaction, drowsy driver detection, telenursing, pain assessment, analysis of mother–infant interaction, social robotics, facial animation, and expression mapping for video gaming, among others.[13] It is also useful for pedestrian detection or head pose estimation in automatic driver assistance system,[14] image retrieval in search engine, and photo management.[15] Nowadays, object detection is widely used in consumer electronics specifically in smartphones.[16]

The object detection system can be categorized by the number of classes of objects they recognize accurately in challenging conditions posed by camera view. The categories are single class from a single view, for example, face detection from frontal views as shown in Fig. 2; single class from multiple views, for example, human tracking system in visual surveillance[17] The example for this is shown by taking video from PETS database[18] as shown in Fig. 3; multiple class from single view, for example, detection of

Encyclopedia of Image Processing, First Edition
DOI: 10.1201/9781351110273-140000130

Copyright © 2018 by Taylor & Francis. All rights reserved.

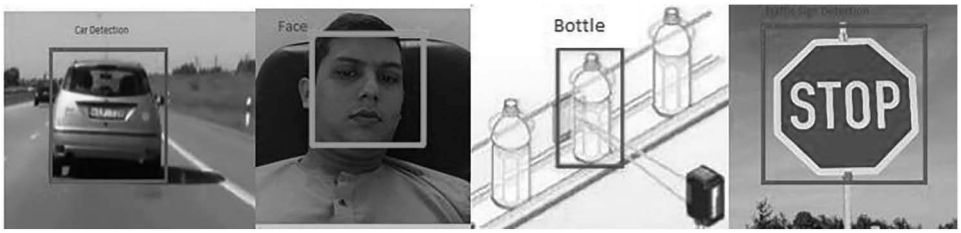


Fig. 1 Object detection application

Fig. 2 Single-class single-view object detection

Fig. 4 Multiple-class single-view object detection

Fig. 3 Single-class multiple-view object detection

Fig. 5 Multiple-class multiple-view object detection

humans and mobile in a single scene as shown in Fig. 4; and multiple class from multiple views, for example, intelligent traffic management system to detect vehicles, pedestrians,[19,20] traffic sign, etc. from a video taken from PETS database[18] as shown in Fig. 5.

CHALLENGES IN OBJECT DETECTION

Object detection is challenging task because images in real life is affected by many factors such as noisy image, illumination changes (Fig. 6), dynamic backgrounds (Fig. 7), shadowing effect (Fig. 8), and camera jitter (Fig. 9).

Object detection is difficult when object is rotated, scaled, under occlusion condition and cluttering. Many applications require detecting more than one object class. If a large number of classes are being detected, the processing speed becomes an important issue, as well as the kind of classes that the system can handle without accuracy loss.[21]

Most methods cannot handle multiple views or large pose variations apart from deformable part-based methods. Efficiency and computational power are still the issues for implementation of some techniques in real-time applications. By using a specialized hardware (e.g., field

Fig. 6 Illumination problem

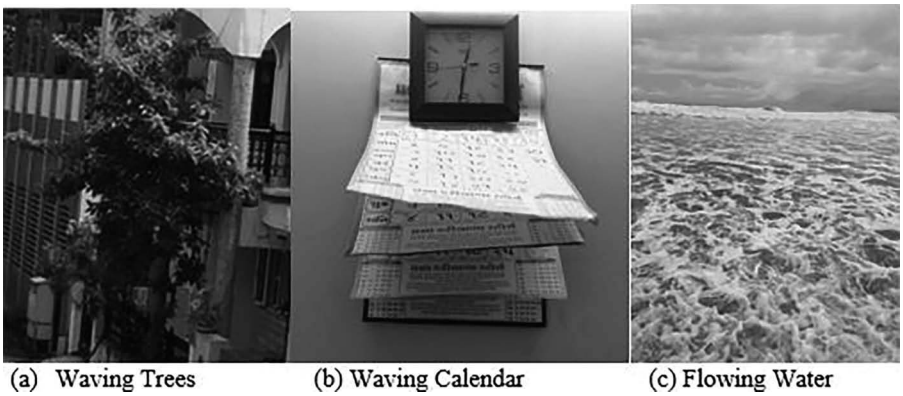


Fig. 7 Dynamic background

Fig. 8 Shadowing effect

programmable gate array [FPGA], graphics processing unit [GPU], application specific integrated circuit [ASIC]), some methods can run in real time.[21]

Fig. 9 Camera jitter

OBJECT DETECTION STRATEGIES

This section mainly deals with different types of object detection techniques.

Feature-Based Object Detection

The block diagram of feature-based object detection system is shown in Fig. 10. As shown in block diagram, feature points or key points are derived first from the input image that can describe unique features of the image and that are invariant to all transformation. Subsequently, invariant feature vector representation, also called region descriptors, are derived for each feature point, each representing the image information available in a local neighborhood around one interest point. Object recognition can then be performed by matching the region descriptors to the model database.

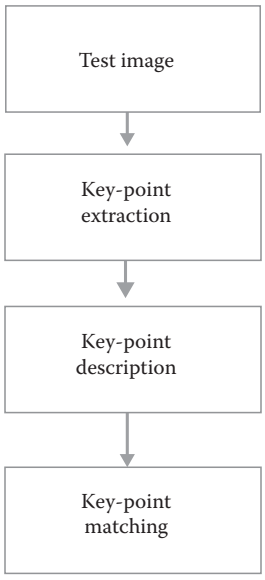


Fig. 10 Block diagram of feature-based methods for object detection

Features detected can be global features or local features. Global features such as color, texture, and histogram aim to describe an image as whole, whereas local features detect key points in an image and describe them. Global features are much faster, are easy to compute, and requires small amount of memory, but they can only be used when some amount of information related to object of interest is available. Also global features are sensitive to clutter and occlusion.[22] Local features require significant amount of memory because of large amount of image features, but they result in better performance.[22] To reduce the memory problem, dimensionality of vectors can be reduced and compact, localized key points can be found. For a good feature detection algorithm, the features found should be robust, accurate, and repeatable.

This section describes various feature-based algorithms for object detection with their pros and cons.

Harris Corner Detection

Corners are regions in the image with large variation in intensity in all the directions. In Harris corner detection, the difference in intensity for a displacement of (u, v) is found in all directions. Window function can be either a rectangular window or Gaussian window. It is used to give weights to pixels underneath.[23] It can be expressed in mathematical form as follows:

$$E(u, v) = \sum_{x,y} w(x,y) * \left[I(x+u, y+v) - I(x,y) \right]^2 \quad (1)$$

This function $E(u, v)$ should be maximize for corner detection. That means, the second term should be maximum. After applying Taylor expansion to Eq. 1 and using some mathematical steps, a final equation is derived as follows:

$$E(u, v) \approx [u \ v] * M \begin{bmatrix} u \\ v \end{bmatrix} \quad (2)$$

where

$$M = \sum w(x,y) \begin{bmatrix} I_x I_x & I_x I_y \\ I_x I_y & I_y I_y \end{bmatrix} \quad (3)$$

Here, I_x and I_y are the local image derivatives in x and y directions, respectively. After this, cornerness measure C is found for each pixel using an equation, which will determine if a window contains a corner or not.

$$R = \det(M) - k\left[\text{trace}(M)\right]^2 \quad (4)$$

where

$$\det(M) = \hat{\lambda}_1 \hat{\lambda}_2 \quad (5)$$

$$\text{trace}(M) = \hat{\lambda}_1 + \hat{\lambda}_2 \quad (6)$$

$\hat{\lambda}_1$ and $\hat{\lambda}_2$ are the eigenvalues of M, and k is an adjusting parameter. Eigenvalues are computationally expensive, so Harris suggested using a measure that combines the two eigenvalues in a single measure. So the value of cornerness measure decide whether a region is corner, edge, or flat. For smaller $|R|$, the region is flat. For $R < 0$, the region is edge. When R is large, the region is a corner. Figure 11 shows the output obtained using Harris detector.

Harris corner detectors exhibit high-speed performance and requires less memory, so it is suitable for real-time video processing applications specifically in the embedded environment. However, it is not scale invariant and not robust to noise.

Features from Accelerated Segment Test Algorithm

Features from accelerated segment test (FAST) is a corner detector in which corner points are detected by applying a segment test to every pixel by considering a circle of 16 pixels around the candidate pixel.[24] A pixel p in the image is selected which is to be identified as an interest point or not. Let its intensity be I_p. Select appropriate threshold value t. Now the pixel p is classified as a corner if there exists a set of n contiguous pixels in the circle of 16 pixels which are all brighter than $I_p + t$, or all darker than $I_p - t$.

A high-speed test was proposed to exclude a large number of non-corners. This test examines only the four pixels at positions 1, 9, 5, and 13 (First, 1 and 9 are tested if they are too brighter or darker. If so, then 5 and 13 are checked.)[24] If p is a corner, then at least three of these must be brighter than $I_p + t$ or darker than $I_p - t$. If this is not the case, then p cannot be a corner. The full segment

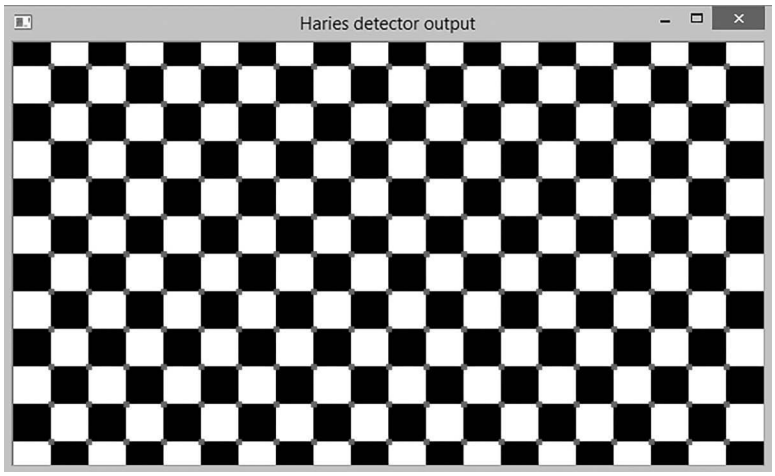


Fig. 11 Output of Harris corner detector

test criterion can then be applied to the passed candidates by examining all the pixels in the circle.

This detector itself exhibits high performance, but multiple features are detected adjacent to one another. It does not reject as many candidates for $n < 12$. The choice of pixels is not optimal because its efficiency depends on the ordering of the questions and the distribution of corner appearances. Multiple features are detected adjacent to one another. An improvement for addressing these limitations is achieved with a machine learning approach and non-maximal suppression.[25] The results of FAST with and without non-maximal suppression are shown in Fig. 12. It can be observed that non-maximal suppression considerably reduces the features adjacent to each other.

Scale-Invariant Feature Transform

A corner in a small image within a small window is flat when it is zoomed in the same window but still will be a corner when the image will be rotated. So corner detectors described earlier are not only rotation invariant but also scale invariant. To overcome this, a new feature detection and description algorithm is developed which is called scale-invariant feature transform (SIFT).[26] The flowchart for SIFT is shown in Fig. 13 below.

As shown in the figure, there are mainly five steps involved in SIFT feature detection, description, and matching.

1. Scale-space extrema detection

   To detect key points with different scale in SIFT, scale-space filtering is used. Laplacian of Gaussian (LoG) is found for the image with various σ values. LoG is basically a blob detector which detects blobs in various sizes due to change in σ. In short, σ acts as a scaling parameter. Gaussian kernel with low σ gives high value for small corners, whereas Gaussian kernel with high σ fits well for larger corners.

Fig. 12 Output using FAST detector with and without maximal suppression

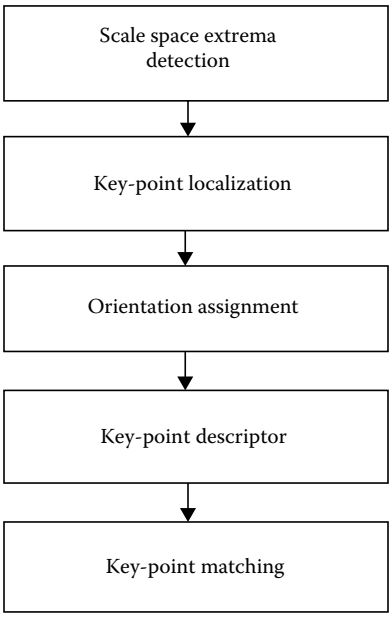


Fig. 13 Algorithmic flowchart for SIFT

To optimize LoG operation, SIFT uses difference of Gaussians (DoG). Mathematically, it can be shown that LoG can be approximately equal to DoG. DoG is obtained as the DoG blurring of an image with two different σ's: σ and $k\sigma$.

$$\frac{\partial G}{\partial \sigma} = \sigma\Delta^2 G = \frac{G(x,y,k\sigma) - G(x,y,\sigma)}{k\sigma - \sigma} \tag{7}$$

where

$$G(x,\ y,\ k\sigma) = \frac{1}{2\pi(k\sigma)^2} e^{-\left(\frac{x^2+y^2}{2k^2\sigma^2}\right)} \tag{8}$$

This process is done for different octaves of the image in Gaussian Pyramid.

Once this DoG are found, images are searched for local extrema over scale and space. For example, one pixel in an image is compared with its eight neighbors as well as nine pixels in next scale and nine pixels in previous scales. If it is a local extrema, it is a potential key point. It basically means that key point is best represented in that scale. Regarding different parameters, there are some empirical data values which can be summarized as follows: number of octaves = 4, number of scale levels = 5, initial $\sigma = 1.6$, and $k = \sqrt{2}$ as optimal values.[26]

2. Key point localization

   To refine the key points found in the previous stage and get more accurate results, further computations are made. In this step, key points with low contrast and edge key points that were detected in the previous stage because of DoG sensitiveness to it are removed. Taylor series expansion of scale space is used to remove low contrast point. If the intensity of Taylor series expansion at this extrema is less than a contrast threshold value, then it is rejected.

   To remove edges 2×2, Hessian matrix (H) is used to compute the principal curvature. From Harris corner detector, it is known that for edges, one eigenvalue is larger than the other. This concept is used here. If this ratio is greater than the edge threshold, the key point is discarded.

3. Orientation assignment

   To make SIFT rotation invariant, a neighborhood is taken around the key point location depending on the scale, and the gradient magnitude and direction are calculated in that region. An orientation histogram with 36 bins covering 360° is created. It is weighted by gradient magnitude and Gaussian-weighted circular window. The highest peak in the histogram is taken, and any peak above 80% of it is also considered to calculate the orientation. It creates key points with same location and scale, but different directions. It contributes to stability of matching.

4. Key point descriptor

   The description stage of SIFT starts by taking a 16×16 neighborhood around the key point using its scale to select the level of Gaussian blur for the image. It is divided into 16 sub-blocks of 4×4 size. For each sub-block, eight-bin orientation histogram is created. So a total of 128 bin values are available. It is represented as a vector to form key point descriptor. Finally, it is normalized to gain invariance to changes in illumination.

5. Key point matching

   Key points between two images are matched by brute force matching or by identifying their nearest neighbors. But in some cases, the second closest match may be very near to the first. It may be due to noise or some other reasons. In that case, the ratio of the closest distance to the second closest distance is taken. If it is >0.8, they are rejected.

The result for SIFT algorithm for object detection is shown in Fig 14.

SIFT gives good result in case of transformation and changes in illumination but it is slow. SIFT is more suitable in case of images that are effected by translation, rotation, and scaling.[26] To overcome that, speeded up robust features (SURF) are developed.

Speeded-Up Robust Features

The SURF scheme is designed as an efficient and fast alternative to SIFT. The flowchart of SURF is shown in Fig. 15.

SURF approximates LoG with computation based on a simple 2-D box filter. The main advantage of this approximation is that convolution with box filter can be easily

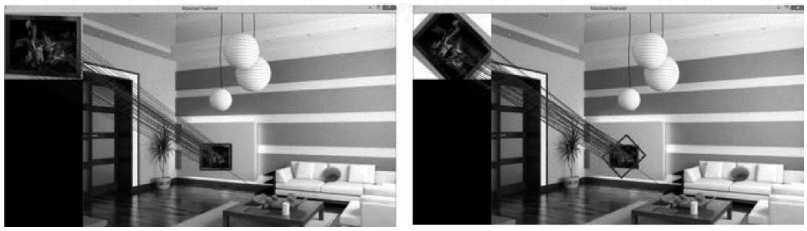


Fig. 14 SIFT object detection results

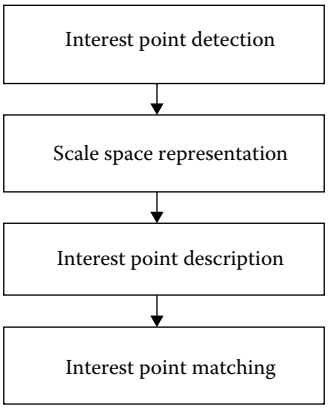


Fig. 15 Flowchart for SURF

calculated with the help of integral images. And it can be done in parallel for different scales. Also the SURF rely on determinant of Hessian matrix for both scale and location.[27] The approximated determinant of Hessian can be expressed as follows:

$$\mathrm{Det}\left(H_{\mathrm{approx}}\right) = D_{xx}D_{yy} - \left(wD_{xy}\right)^2 \tag{9}$$

where w is a relative weight for the filter response and used to balance the expression for the determinant. For orientation assignment, SURF uses wavelet responses in horizontal and vertical directions using integral image approach for a neighborhood of size 6s, where s is the scale at which interest point is detected. Adequate Gaussian weights are also applied to it. The dominant orientation is estimated by calculating the sum of all responses within a sliding orientation window of angle 60°.

For feature description, SURF uses Haar wavelet responses in horizontal and vertical directions in a neighborhood of size 20s×20s, where s is the scale at which interest point is taken. Interest region is divided into 4×4 subregions. For each subregion, horizontal and vertical wavelet responses are taken which is denoted by d_x and d_y. These responses are again weighted with Gaussian window centered at interest point. Then, they are summed up for each subregion and a feature vector is formed as follows:

$$v = \left(\sum d_x, \sum d_y, \sum |d_x|, \sum |d_y|\right) \tag{10}$$

This is computed for all 4×4 subregions resulting in SURF feature descriptor with total 64 dimensions. The lower the dimension, the higher the speed of computation and matching. For more distinctiveness, SURF feature descriptor has an extended 128 dimension version. The sums of d_x and $|d_x|$ are computed separately for $d_y < 0$ and $d_y \geq 0$. Similarly, the sums of d_y and $|d_y|$ are split up according to the sign of d_x, thereby doubling the number of features.

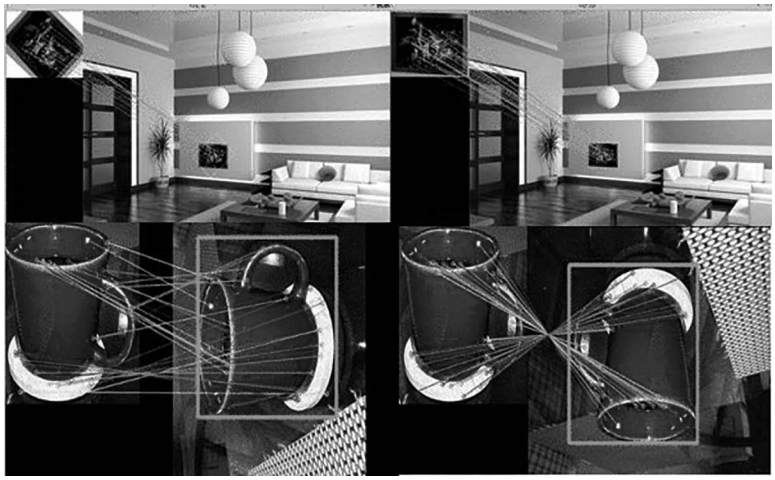


Fig. 16 Result of SURF for object detection

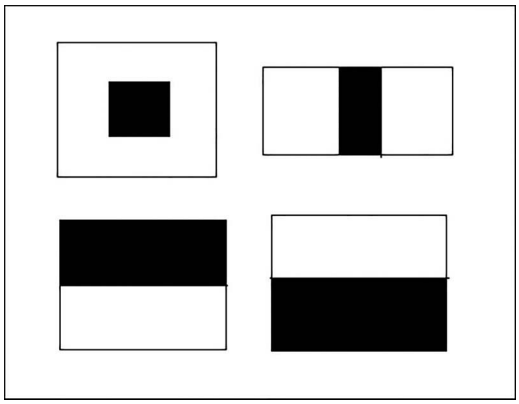


Fig. 17 Rectangular Haar-like features

The sign of the Laplacian can be used to distinguish bright blobs on dark backgrounds from the reverse situation without adding any computational cost. The results of SURF algorithm for object detection is shown in Fig. 16. It can be seen that SURF is rotation invariant and scale invariant.

SURF has a higher processing speed compared to SIFT because it uses a 64-dimensional feature vector compared to SIFT which uses 128-dimensional feature vector. SURF is good at handling images with blurring and rotation, but not good at handling viewpoint change and illumination change.[27]

Haar-Like Features

The principle of Haar-like features is based on detection of features encoding of some information about the class to be detected. They are adjacent rectangles in a particular position of an image. Figure 17 shows the types of Haar-like features depending on the number of adjacent rectangles. There are three types of Haar-like features: edge feature (Fig. 17 two bottom squares), line feature (Fig. 17 upper left square), and center-surround feature (Fig. 17 upper right square).[28]

The idea behind the Haar-like feature selection algorithm is simple. It lies on the principle of computing the difference between the sum of white pixels and the sum of black pixels. The main advantage of this method is the fast sum computation using the integral image. It is called Haar-like because it is based on the same principle of Haar wavelets.

The result of face detection using the Haar-like feature is shown in Fig. 18.

Fig. 18 Face detection using Haar-like features

Local Binary Pattern

Local binary pattern (LBP) compute a local representation of textures. This local representation is constructed by comparing each pixel with its surrounding neighborhood of pixels. The first step in constructing the LBP texture descriptor is to convert the image to gray scale. The LBP algorithm slides its processing window over the grayscale image for evaluating the successive stages of the cascade algorithm by scoring their features. Each feature is described by 3×3 neighboring rectangular areas.

The value of each feature is computed by comparing the central area with the neighboring area around it (eight neighbors). The result is in the form of 8-bit binary value called LBP as can be seen in Fig. 19. A number of features represent a stage of the cascade algorithm. Every feature has positive and negative weights associated with it. For the case where the feature is in consistence with the object to be detected, the positive weight is added to the sum. For the case where the feature is inconsistent with the object, the negative value is added to the sum. The sum is then compared to the threshold of the stage. If the sum is below the threshold, the stage fails and the cascade terminates early, and thus, the processing window moves to the next window. If the sum is above the threshold, the next stage of the cascade is attempted. In general, if no stage rejects a candidate window, it is assumed that the object has been detected.

In order to avoid the redundancy of computing the integral of rectangles, the integral images are calculated to speed up the calculation of the feature.

Binary Robust Independent Elementary Features

SURF and SIFT feature descriptors take lots of memory because they have high-dimension feature vectors. But all these dimensions may not be needed for actual matching. It can be compressed using several methods such as PCA and LDA. Even other methods such as locality sensitive hashing is used to convert these SIFT descriptors in floating point numbers to binary strings. These binary strings are used to match features using Hamming distance. This provides better speedup because finding hamming distance is just applying XOR and bit count, which are very fast in modern CPUs with SSE instructions. But here also the descriptors are found first then hashing is applied, which does not solve the problem on memory.[29]

Binary robust independent elementary features (BRIEF) provide a shortcut to find the binary strings directly without finding descriptors. It takes smoothened image patch

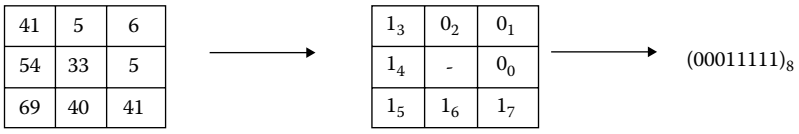


Fig. 19 Method to calculate LBP

and selects a set of $n_d(x, y)$ location pairs in a unique way. Then, some pixel intensity comparisons are done on these location pairs. For example, let the first location pairs be p and q. If $I(p) < I(q)$, then its result is 1, else it is 0. This is applied for all the n_d location pairs to get an n_d-dimensional bit string.[29]

One important point is that BRIEF is a feature descriptor, it does not provide any method to find the features. So other feature detectors such as SIFT and SURF have to be used. In short, BRIEF is a faster method for feature descriptor calculation and matching. It also provides high recognition rate unless there is large in-plane rotation.[29]

Feature Matching

For feature matching, many methods such as brute force matching, Fast Library for Approximate Nearest Neighbors, and fast directional chamfer matching can be used. Brute force matcher is simple. It takes the descriptor of one feature in first set and is matched with all other features in second set using some distance calculation such as L1 norm, L2 norm, or hamming distance. And the closest one is returned. It contains a collection of algorithms optimized for fast nearest neighbor search in large datasets and for high-dimensional features. It works faster than brute force matcher for large datasets.

Contour- or Silhouette-Based Object Detection

Different appearances and articulations of the object are difficult to model without a very large training set for feature-based method. Humans use many visual cues to recognize an object like shape information retrieved from image contours. Contour-based segmentation lead to a robust, flexible, and efficient system, invariant to object appearance such as color, texture, and illumination, without the need for learning from a large set of images.[30]

In this method, hand-drawn sketches can be used as the model(s), which can preserve the significant amount of discriminative object structural information required for liable detection. The sketch model is then segmented into multiple parts, which are expected to capture certain genuine object parts. It is important to identify those parts, which are likely to describe certain actual parts of an object and therefore is expected to be aligned to the choices of a human observer. For part decomposition, different types of algorithms can be used based on convexity measures of an object. Each part is segmented into contour fragments and then matched using different matching algorithms such as fast directional chamfer matching (FDCM). Figure 20 indicates the flowchart and the result of one example of silhouette-based object detection and tracking system.

From the given input image or sketch, shape segmentation and part decomposition is done automatically by using concept of convexity defect[31] and information regarding parts is stored in the database on the server side. On the client side from a test video, the first frame is extracted and parts are matched using FDCM. Then using SURF, key points are extracted and matched. As can be seen from the results in the flowchart, deformation in the parts can be easily handled by this method.

Comparison between Feature-Based and Contour-Based Approaches

Contour based methods are robust, flexible, and invariant to object deformation and they require no training. While they have the limitation that it is not so easy to distinguish the object contours from other edges due to the presence of object and/or background texture. Also the gradient along the object boundary may become very weak in certain portions due to the presence of a matching background.[32,33] Feature-based methods are more popular because of its accuracy and speed but they require large training set to deal with different appearances and articulation of the object.[30]

OBJECT TRACKING

Object detection involves locating objects in frames of a video sequence, whereas tracking is the process of locating moving objects over a period of time. There are two key steps in object tracking: detection of moving objects and tracking of such objects from frame to frame.

Tracking objects can be complex due to loss of information caused by projection of the 3D world on a 2D image, noise in images, complex object motion, nonrigid or articulated nature of objects, partial and full object occlusions, complex object shapes, scene illumination changes, and real-time processing requirements.[34] Many object tracking strategies have been designed that can overcome some of the above issues and can work in real life. All techniques are described one by one in Section "Absolute Difference Method."

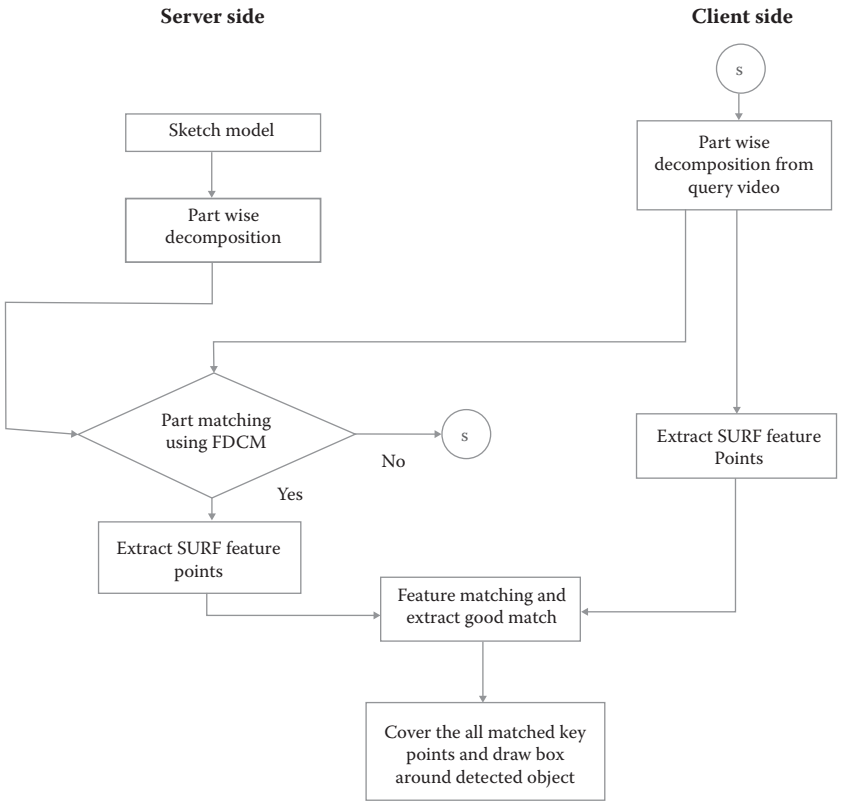


Fig. 20 Flowchart of silhouette-based object detection and tracking system

Absolute Difference Method

In this method, a pixel-by-pixel difference in consecutive frames or in sequence of images is calculated. It is basically image subtraction. It can be mathematically represented as follows:

$$D(t) = I\left(t_i\right) - I\left(t_j\right) \tag{11}$$

where $I(t_i)$ and $I(t_j)$ are the pixel intensity values in two different frames. Difference of intensities are stored in D. The value of D gives an image frame with changed and unchanged regions. It gives better result for only dynamic scene changes; in other words, this method fail to detect a nonmoving object in the scene. This method also fails when the object is moving slowly because there will not be much difference in the pixel map. This method is easy to understand but involves lots of computations per frame. So it will be hard to use this method for real-time applications.

Background Subtractions

Absolute difference method completely fails to detect objects that are moving slowly or not moving at all. For that conditions, background subtraction method is used. As the name suggests, background subtraction is the process of separating out foreground objects from the background in a sequence of video frames. The major steps in a background subtraction method are preprocessing, background modeling, foreground detection, and data validation.

i. Preprocessing is done to remove any kind of noise in the image.
ii. Background modeling is very important for this method. In basic model, background is modeled using an average, a median, or a histogram analysis over time.[35] In this case, once the model is computed, the pixels of the current image are categorized as foreground by thresholding the difference between the background image and the current frame. The statistical models are more robust models to illumination changes and for dynamic backgrounds.[35] They can be categorized as Gaussian models,[36] support vector models,[37,38] and estimation models.[39,40] Gaussian is the easiest way to represent a background.[36] It expects the history over the time of pixel's intensity values. But a single model cannot deal with the dynamic background, for example, waving trees and rippling water. To overcome this problem, the mixture of Gaussians (MoG) or Gaussian mixture model is used.[36]

The second category utilizes more sophisticated statistical models, for example, support vector machine (SVM)[37] and support vector regression (SVR).[38] In estimation models, the filter is used to estimate the background. The filter may be a

Wiener filter,[39] Kalman filter,[40] or Chebyshev filter.[41] The Wiener filter works well for periodically changing pixels and it produces a larger value of the threshold for random changes that are utilized in the foreground detection. The main advantage of the Wiener filter is that it lessens the uncertainty of a pixel value by representing how it varies with time. A drawback occurs when a moving object corrupts the history values. The Chebyshev filter slowly updates the background for changes in lighting and scenery while making the use of a small memory footprint with low-computational complexity. In Cluster model, K-means models, codebook models, neural network models can be used.[42] Table 1 shows the comparison between different background modeling techniques in the challenging environment.

iii. After modeling background, foreground is detected by taking the absolute difference between the current frame and the modeled background.

iv. Threshold is used to validate that object is present or not. The background initialization and updating of background after specific time is necessary and important in this method.

The result for background subtraction using two different methods is shown in Fig. 21.

Optical Flow Technique

Optical flow is the pattern of apparent motion of image objects between two consecutive frames caused by the movement of object or camera. It is a 2D vector field where each vector is a displacement vector showing the movement of points from first frame to second. Optical flow has many applications in areas such as structure from motion, video compression, and video stabilization. Optical flow works on the assumptions that the pixel intensities of an object do not change between consecutive frames and neighboring pixels have similar motion.[43]

Consider a pixel $I(x, y, t)$ in the first frame at time t. It moves by distance (d_x, d_y) in the next frame taken after time d_t. So, from the assumption in calculating the optical flow, it can be concluded that

Table 1 Comparison of different algorithms for background modeling

No	Challenges	Solution
1	Noisy image	Clusters models: K-means, codebook[42] Features: blocks, clusters[58]
2	Camera jitter	Statistical models: MoG[36] Advanced statistical models: DMM Fuzzy models
3	Camera automatic adjustments	Background maintenance: MoG[36] Features: edges
4	Illumination changes	Filter models: Wiener filter[39] Sparse models Features: textures
5	Bootstrapping	Background initialization: consecutive frames Cluster models: K-means, codebook[42] Features: blocks
6	Moved background objects	Background maintenance: MoG[36]
7	Inserted background objects	Background maintenance: MoG[36]
8	Dynamic backgrounds	Fuzzy models Features: texture Features: histogram
9	Beginning moving objects	Background maintenance: MoG[36]
10	Sleeping foreground objects	Background maintenance: MoG[36]

$$I(x, y, t) = I(x + dx, y + dy, t + dt) \tag{12}$$

If Taylor series approximation is taken of the right-hand side, common terms are removed and equation is divided by d_t to get the following equation:

$$f_x u + f_y v + f_t = 0 \tag{13}$$

where

Fig. 21 Background subtraction using the KNN subtraction technique and the adaptive Gaussian mixture model

$$f_x = \frac{\partial f}{\partial x};\ f_y = \frac{\partial f}{\partial y} \tag{14}$$

$$u = \frac{dx}{dt}; v = \frac{dy}{dt} \tag{15}$$

The preceding equation is called optical flow equation. In this equation, f_x and f_y are the image gradients. Similarly, f_t is the gradient along time. But (u, v) are unknown. Two unknown variables cannot be solved with one equation. So, several methods are provided to solve this problem and one of them is Lucas–Kanade. It takes a 3×3 patch around the point. All the nine points have the same motion. An image gradient (f_x, f_y, f_t) is found for these nine points. Now from the assumptions that pixel intensities do not change and neighboring pixels have the same motion, problem becomes solving nine equations with two unknown variables which is overdetermined. A better solution is obtained with the least squares fit method. The final solution which is two-equation, two-unknown problem and solved to get the solution is given as follows:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \sum_i f_{x_i}^2 & \sum_i f_{x_i} f_{y_i} \\ \sum_i f_{x_i} f_{y_i} & \sum_i f_{y_i}^2 \end{bmatrix}^{-1} \begin{bmatrix} -\sum_i f_{x_i} f_{t_i} \\ -\sum_i f_{y_i} f_{t_i} \end{bmatrix} \tag{16}$$

The idea of an optical flow is simple: given some points to track, the optical flow vectors of those points are received and solved for u and v, but it fails when there is large motion. So, the concept of image pyramids is used. When we go up in the pyramid, small motions are removed and large motions become small motions. After applying Lucas–Kanade at every scale, optical flow along with the scale information is received. The output of optical flow algorithm for two different videos is shown in Figs. 22 and 23. Figure 22 indicates the optical flow output of two different frames of traffic on road. Green arrows on frame indicates magnitude and direction of vectors. Results are color coded for better visualization. Direction corresponds to the hue value of the image. Magnitude corresponds to the value plane.

Figure 23 indicates optical flow for face tracking from webcam.

Dense Optical Flow Technique (Gunner Farneback's Optical Flow)

Lucas–Kanade method computes optical flow for a sparse feature set. Sparse techniques only need to process some pixels from the whole image, while dense techniques process all the pixels. Dense techniques are slower but can be more accurate. An example of a dense optical flow algorithm is Gunner Farneback's optical flow. It computes the optical flow for all the points in the frame.[44] The results for algorithms are shown in Fig. 24. Results are color coded for better visualization. Direction corresponds to the hue value of the image. Magnitude corresponds to the value plane.

Meanshift and Camshift Method

The meanshift algorithm is an efficient approach to track objects whose appearance is defined by histograms. For a given set of points (it can be a pixel distribution like histogram) and a small window, you have to move that window to the area of maximum pixel density:[45]

Algorithm 1: Meanshift method

```
Do the iteration till
    {
    C1 = Starting window
    C1_o = center of window
    C1_r = centroid of window
    If ( C1_o == C1_r)
            Break;
    else
            Move to Next window such that C2_o = C1_r where
C2_o is center of new window
    }
```

So in this method, the histogram back-projected image and initial target location is passed. When the object moves, obviously the movement is reflected in histogram back-projected image. As a result, the meanshift algorithm moves window to the new location with maximum density.[45]

One problem with the meanshift algorithm is window always has the same size when object is farther away or very close to camera. This problem can be solved by using camshift (continuously adaptive meanshift).[46]

In this method, the meanshift is applied first. Once the meanshift converges, it updates the size of the window using the following equation:

$$s = 2 * \sqrt{\frac{M_{00}}{256}}$$

It also calculates the orientation of best-fitting ellipse to it. Again, it applies the meanshift with new scaled search window and previous window location. The process is continued until required accuracy is met. The result for mobile tracking using camshift is shown in Fig. 25. To improve the accuracy of tracking further, camshift is used with different features such as SIFT features, color feature, and texture information, or Camshift algorithm is combined with the Kalman filter.

Fig. 22 Optical flow output for two different frames of road

Fig. 23 Face tracking using optical flow

HARDWARE IMPLEMENTATION CHALLENGES OF OBJECT DETECTION AND TRACKING METHODS

Normally, general-purpose processors and specialized architectures such as digital signal processors do not provide the flexibility required to achieve the real-time performance of object detection and tracking algorithms, so there is always need of dedicated hardware architectures for these algorithms. Most of the applications of object detection and tracking are associated with real-time performance constraints. The problem is further aggravated in an embedded systems environment, where most of these applications are deployed. The high computational complexity and power consumption makes implementing an embedded object detection system with real-time performance a challenging task.[47] Consequently, there is a strong need for dedicated hardware architectures capable of delivering high detection accuracy within an acceptable processing time given the available hardware resources. Almost all algorithms discussed above for object detection and tracking have inherent parallelism associated with it. If hardware with more parallel resources is used to implement this algorithms, there can be considerable speed up. Same way it can also be implemented on graphical processing units with multiple cores with high computational tasks divided between multiple cores.

Fig. 24 Result of dense optical flow for two frames in a video

Fig. 25 Mobile tracking using camshift

The platforms with more number of DSP blocks (embedded multipliers and accumulators) and more amount of memory are ideally suitable for these algorithms. While implementing these algorithms, there are strict timing constraints, memory constraints, and resource sharing constraints. The data rate requirements of real-time applications impose a strict timing constraint. At video rates, all required processing for each pixel must be performed at the pixel clock rate (or faster).[48] This generally requires low-level pipelining to meet this constraint. Another form of timing constraint occurs when asynchronous processes need to synchronize to exchange data or synchronization within multiple clock domains. Some operations require images to be partially or wholly buffered. While current high-capacity devices have sufficient on-chip memory to buffer a single image, in most applications it is poor use of this valuable resource to simply use it as an image buffer. For this reason, image frames and other large datasets are more usually stored in off-chip memory.[48] The simplest way to increase memory bandwidth is to have multiple parallel memory systems. If each memory system has separate address and data connections to the FPGA, then each can be accessed independently and in parallel. A further approach to increasing bandwidth is to increase the word width of the memory. Each memory access will then read or write several pixels. This is effectively connecting multiple banks in parallel but using a single set of address lines common to all banks. Every hardware platform will have a limited amount of resources, so there will always be a need to share resources among different processes.

The majority of the published implementations of object detection are on CPU and GPU platforms. The implementation by Benenson et al.[49] achieves higher throughput on a GPU at 100 fps using histogram gradient approach presented by Dollar et al.[50] but with a resolution of 640×480 pixels. For higher throughput and resolution,[51] FPGA-based implementations have been reported recently. Different parts of the detector are implemented on different platforms: HOG feature extraction is divided between an FPGA and a CPU, and SVM classification is done on a GPU.[52] It can process 800×600 pixels at 10 fps for single-scale detection. The entire HOG-based detector is implemented on

FPGA by Mizuno et al.[53] and can process 1080 HD video (1920×1080 pixels) at 30 fps. However, the implementation only supports a single image scale. An ASIC version of this design is presented by Takagi et al.,[54] with dual cores to enable voltage scaling for power consumption of 40.3 mW for 1080 HD video at 30 fps, but still only supports a single image scale. It should be noted that these hardware implementations have relatively large on-chip memory sizes (e.g., Takagi et al.[54] uses 1.22 Mbit on ASIC, Hahnle et al.[55] uses 7 Mbit on FPGA), which contributes to increased hardware cost. Thus, from the discussion, it can be concluded that it is hard to satisfy all the desired requirements for accurate and robust object detection in embedded systems, which include real-time, high resolution (1,080 HD), high frame rate (>30 fps), multiple image scale, low power and low hardware costs.[56,57] There has to be trade-off according to the application. Performance requirements can vary quite a lot depending on application. Consider the scenarios of analyzing the footage of a security camera at the restricted area in a busy airport and demographic analysis of visitors in a retail store.

In the second scenario, algorithm can run at just five frames per second (FPS), but in the first scenario, a higher FPS is critical. The first scenario also places a much greater demand for accuracy. So, after knowing the performance requirement, suitable device and implementation technique can be chosen.

CONCLUSION

Due to high-speed computers and low-cost cameras, the demand for object detection and tracking algorithms that works in real time is increasing day by day. Although great progress has been observed in the last years, we are still far from achieving human-level performance. Object detection and tracking is an important task in computer vision field. Object detection is the first stage in many computer vision applications. This entry focused on the methods of real-time object detection. A comparison of different object detection methods in challenging environment is discussed. For object detection, SURF works well in embedded applications with its high accuracy of results with low feature size and high speed. Object tracking involves detecting object in every frame, tracing its path from frame to frame, and analyzing its behavior. For object tracking, combination of optical flow and meanshift techniques work well in almost all conditions. Hardware implementation constraints along with trade-offs in designing algorithms for real-time applications are also discussed.

REFERENCESS

1. Rodrigo, V.; Javier, R. Object detection: Current and future directions. Front. Rob. AI **2015**, *2*, 29.
2. Kalirajan, K.; Sudha, M. Moving object detection for video surveillance. Sci. World J. **2015**, *2015*, 10. doi:10.1155/2015/907469.
3. Haritaoglu, I.; Harwood, D.; Davis, L. W4: Real time surveillance of people and their activities. IEEE Trans. Pattern Anal. Mach. Intell. **2000**, *22* (8), 809–830.
4. Beymer, D.; McLauchlan, P.; Coifman, B.; Malik, J. A real-time computer vision system for measuring traffic parameters. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, San Juan, PR, 1997.
5. Bramberger, M.; Pflugfelder, R.P.; Rinner, B.; Schwabach, H.; Strobl, B. Intelligent traffic video sensor: Architecture and applications. In *Telecommunications and Mobile Computing Workshop on Wearable Computing*, Graz, Austria, March 2003.
6. Koller, D.; Weber, J.; Malik, J. Towards real time visual based tracking in cluttered traffic scenes. In *IEEE Intelligent Vehicles Symposium*, Paris, 1994, 201–206.
7. Pflugfelder, R. *Visual Traffic Surveillance Using Real-Time Tracking*; Diploma Thesis; Technical Report PRIP-TR-071, PRIP, TU Wien, 2002.
8. Yang, Y.; Luo, H.; Xu, H.; Wu, F. Towards real-time traffic sign detection and classification. In *IEEE 17th International Conference on Intelligent Transportation Systems (ITSC)*, Qingdao, October 8–11, 2014.
9. Pauly, M.; Surmann, H.; Finke, M.; Liang, N. Real-Time object detection for autonomous robots. In *Autonome Mobile Systeme* ; Wörn, H.; Dillmann, R.; Henrich, D.; Eds.; Informatik aktuell; Springer: Berlin and Heidelberg, 1999.
10. Azad, R.; Azad, B.; Nabil, K.; Shahram, J. Real-time human-computer interaction based on face and hand gesture recognition. Int. J. Found. Comput. Sci. Technol. **2014**, *4* (4), 37–48.
11. Herakovic, N. Robot vision in industrial assembly and quality control processes. In *Robot Vision*; Ude, A.; Ed.; 2010, In Tech. Available at http://www.intechopen.com/books/robotvision/robot-vision-in-industrial-assembly-and-quality-control-processes.
12. Lecron, F.; Benjelloun, M.; Mahmoudi, S. Descriptive image feature for object detection in medical images. In *Image Analysis and Recognition*, ICIAR 2012. Lecture Notes in Computer Science; Campilho, A.; Kamel, M.; Eds.; Springer: Berlin and Heidelberg, Vol. 7325, 2012.
13. De la Torre, F.; Cohn, J.F. *Facial Expression Analysis. Visual Analysis of Humans*; Springer: London, 2011, 377–409.
14. Geronimo, D.; Lopez, A.M.; Sappa, A.D.; Graf, T. Survey of pedestrian detection for advanced driver assistance systems. IEEE Trans. Pattern Anal. Mach. Intell. **2010**, *32* (7), 1239–1258.
15. Kumar, N.; Belhumeur, P.; Nayar, S. Facetracer: A search engine for large collections of images with faces. In *Computer Vision—ECCV 2008. ECCV 2008*, Vol. 5305, Lecture Notes in Computer Science; Forsyth, D.; Torr, P.; Zisserman, A.; Eds.; Springer: Berlin and Heidelberg, 2008, 340–353.
16. Jeong, K.; Moon, H. Object detection using FAST corner detector based on smartphone platforms. In *First ACIS/JNU International Conference on Computers, Networks,*

Systems and Industrial Engineering (CNSI), Jeju Island, 2011, 111–115
17. Blauensteiner, P.; Kampel, M. Visual surveillance of an airport's Apron—An overview of the AVITRACK project. 2004.
18. PETS. *Benchmark Data*, 2009. Available at http://www.cvg.reading.ac.uk/PETS2009/a.html.
19. Daniele, T. FPGA-based pedestrian detection under strong distortions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, Boston, MA, 2015, 66–71.
20. Roig, G.; Boix, X.; Shitrit, H.B.; Fua, P. Conditional random fields for multi-camera object detection. In *IEEE International Conference on Computer Vision (ICCV)*, Barcelona, 2011, 563–570.
21. Bouwmans, T. Traditional and recent approaches in background modeling for foreground detection: An overview. Sci. Direct J. Comput. Sci. Rev. **2014**, *11–12*, 31–66.
22. Hassaballah, M.; Abdelmgeid, A.; Hammam, A. Image features detection, description and matching. In *Image Feature Detectors and Descriptors: Foundations and Applications*, Part of the Studies in Computational Intelligence; Awad, A.I.; Hassaballah, M.; Eds.; Springer: Cham, 2016, 630. doi:10.1007/978-3-319-28854-3_2.
23. Harris, C.; Stephens, M. A combined corner and edge detector. In *Proceedings of the 4th Alvey Vision Conference*; University of Manchester: Manchester, 1988, 147–151.
24. Rosten, E.; Drummond, T. Machine learning for high-speed corner detection. In *Computer Vision—ECCV 2006*, Vol. 3951, ECCV 2006. Lecture Notes in Computer Science; Leonardis, A.; Bischof, H.; Pinz, A.; Eds.; Springer: Berlin and Heidelberg, Germany, 2006, 430–443.
25. Rosten, E.; Porter, R.; Drummond, T. Faster and better: A machine learning approach to corner detection. IEEE Trans. Pattern Anal. Mach. Intell. **2010**, *32*, 105–119.
26. Lowe, D. Distinctive image features from scale-invariant keypoints. Int. J. Comput. Vision **2004**, *60*, 1–28.
27. Bay, H.; Ess, A.; Tuytelaars, T.; Van Gool, L. Speeded-up robust features (SURF). Comput. Vis. Image Underst. **2008**, *110* (3), 346–359.
28. Guennouni, S.; Ahaitouf, A.; Mansouri, A. A comparative study of multiple object detection using haar-like feature selection and local binary patterns in several platforms. Modell. Simul. Eng. **2015**, *2015*, 8. doi:10.1155/2015/948960.
29. Michael, C.; Lepetit, V.; Strecha C.; Fua, P. BRIEF: Binary robust independent elementary features. In *11th European Conference on Computer Vision (ECCV)*, LNCS; Springer: Heraklion, 2010.
30. Bhattacharje, S.; Mittal, A. Part-base deformable object detection with a single sketch. Comput. Vision Image Underst. **2015**, *139*, 73–87.
31. Gopalan, R.; Turaga, P.; Chellappa, R. Articulation invariant representation of non-planer shapes. In *Proceedings of the European Conference of Computer Vision*; Verlag: Berlin and Heidelberg, 2010.
32. Dickinson, S. Object representation and recognition. In *What is Cognitive Science?*; Lepore, E.; Pylyshyn, Z.; Eds.; Basil Black Well Publishers: New Brunswick, NJ, 1999, 172–207.
33. Treiber, M. *An Introduction to Object Recognition*; Springer: London.
34. Hemalatha, C.; Muruganand, S.; Maheswaran, R. A survey on real time object detection, tracking and recognition in image processing. Int. J. Comput. Appl. (0975–8887) **2014**, *91* (16), 38–42.
35. Bouwmans, T.; El-Baf, F.; Vachon, B. Statistical background modeling for foreground detection: A survey. Handb. Pattern Recognit. Comput. Vision **2010**, *4* (2), 181–199.
36. Grimson, W.; Stauffer, C. Adaptive background mixture models for real-time tracking. In *IEEE Conference on Computer Vision and Pattern Recognition*, Fort Collins, CO, 1999, 246–252.
37. Lin H.; Liu T.; Chuang, J. A probabilistic SVM approach for background scene initialization. In *International Conference on Image Processing*, Rochester, NY, September 2002, 893–896.
38. Wang, J.; Bebis, G.; Miller, R. Robust video-based surveillance by integrating target detection with tracking. In *IEEE Workshop on Object Tracking and Classification beyond the Visible Spectrum in Conjunction,* New York, June 2006.
39. Toyama, K.; Krumm, J.; Brumitt, B.; Meyers, B. Wallflower: Principles and practice of background maintenance. In *International Conference on Computer Vision*, Kerkyra, September 1999, 255–261.
40. Ridder, C.; Munkelt, O.; Kirchner, H. *Adaptive Background Estimation and Foreground Detection using Kalman-filtering*; Bavarian Research Center for Knowledge-Based Systems: Erlangen, 1995, 193–199.
41. Chang, R.; Gandhi, T.; Trivedi, M. Vision modules for a multi sensory bridge monitoring approach. In *IEEE Intelligent Transportation Systems Conference*, Washington, WA, October 2004, 971–976.
42. Kim, K.; Chalidabhongse, T.; Hanuood, D.; Davis, L. Background modeling and subtraction by codebook construction. In *IEEE International Conference on Image Processing*, Singapore, 2004.
43. Barron, J.; Fleet, D.; Beauchemin, S. Performance of optical flow techniques. Int. J. Comput. Vision **1994**, *12*, 43–77.
44. Farnebäck, G. Two-frame motion estimation based on polynomial expansion. Image Anal. **2003**, *2749*, 363–370.
45. Gorry, B.; Chen, Z.; Hammond, K.; Wallace, A.; Michaelson, G. Using Mean-Shift Tracking algorithms for real-time tracking of moving images on an autonomous vehicle tested platform. In *Conference of the World Academy of Science Engineering and Technology*, Venice, November 2007, 23–25.
46. Bradski, G. Computer vision face tracking for use in a perceptual user interface. Intel Technol. J. Q2, **1998**, *1*, 1–15.
47. Jinwook, O.; Gyeonghoon, K.; Injoon, H.; Junyoung, P.; Seungjin, L.; Joo-Young, K.; Jeong-Ho, W.; Hoi-Jun, Y. Low-power, real-time object-recognition processors for mobile vision systems. IEEE Micro **2012**, *32* (6), 38–50.
48. Bailey, D.G. *Design for Embedded Image Processing on FPGAs*, 1st Ed.; John Wiley & Sons (Asia) Pte Ltd.: Singapore, 2011.
49. Benenson, R.; Mathias, M.; Timofte, R.; Van Gool, L. Pedestrian detection at 100 frames per second. In *Proceedings IEEE Conference on Computer Vision and Pattern Recognition*, Providence, RI, 2012, 2903–2910.

50. Dollar, P.; Belongie, S.; Perona, P. The fastest pedestrian detector in the west. In *Proceedings of the British Machine Vision Conference*, *Aberystwyth, GBR*, 2010, 68.1–68.11.
51. Wei, Z.; Zelinsky, G.; Samaras, D. Real-time accurate object detection using multiple resolutions. In *Proceedings IEEE International Conference on Computer Vision*, *Rio de Janeiro*, Brazil, 2007, 1–8.
52. Bauer, S.; Kohler, S.; Doll, K.; Brunsmann, U. FPGA-GPU architecture for kernel SVM pedestrian detection. In *Proceedings IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, San Francisco, CA, 2010, 61–68.
53. Mizuno, K.; Terachi, Y.; Takagi, K.; Izumi, S.; Kawaguchi; H.; Yoshimoto, M. Architectural study of HOG feature extraction processor for real-time object detection. In *Proceedings IEEE Workshop on Signal Processing Systems*, Quebec City, QC, 2012, 197–202.
54. Takagi, K.; Mizuno, K.; Izumi, S.; Kawaguchi, H.; Yoshimoto, M. A sub-100-milliwatt dual-core HOG accelerator VLSI for real-time multiple object detection. In *Proceedings IEEE International Conference on Acoustics, Speech and Signal Processing*, Vancouver, BC, 2013, 2533–2537.
55. Hahnle, M.; Saxen, F.; Hisung, M.; Brunsmann, U.; Doll, K. FPGA-based real-time pedestrian detection on high-resolution images. In *Proceedings IEEE Conference on Computer Vision and Pattern Recognition Workshops*, Portland, OR, 2013, 629–635.
56. Dollár, P.; Belongie, S.J.; Perona, P. The fastest pedestrian detector in the west. Br. Mach. Vision Conf. **2010**, *2* (3), 7.
57. Qasaimeh, M. *Fpga-based parallel hardware architecture for real-time object classification*; A Thesis Presented to the Faculty of the American University of Sharjah, June 2014.
58. Bove, V.M. Jr.; Sridharan, S.; Butler, D.E. Real time adapadaptive foreground/background segmentation. EURASIP **2005**, *14*, 2292–2304.

Silhouette-Based Real-Time Object Detection and Tracking

Bhaumik Vaidya, Harendra Panchal and Chirag Paunwala

Abstract Object detection and tracking in the video sequence is a challenging task and time-consuming process. Intrinsic factors like pose, appearance, variation in scale and extrinsic factors like variation in illumination, occlusion, and clutter are major challenges in object detection and tracking. The main objective of the tracking algorithm is accuracy and speed in each frame. We propose the best combination of detection and tracking algorithm which performs very efficiently in real time. In proposed algorithm, object detection task is performed from given sketch using Fast Directional Chamfer Matching (FDCM) which is capable of handling some amount of deformation in edges. To deal with the articulation condition, part decomposition algorithm is used in the proposed algorithm. Combination of these two parts is capable enough to handle deformation in shape automatically. Amount of time taken to perform this algorithm depends on the size and edge segment in the input frame. For object tracking, Speeded up Robust Features (SURF) algorithm is used because of its rotation invariant and fast performance features. The proposed algorithm works in all situations without the prior knowledge about number of frames.

Keywords Convexity defects · Fast directional chamfer matching
Part decomposition · Speeded up robust features

1 Introduction

The ongoing research topic in computer technology that makes efforts to detect, recognize, and track objects through a series of picture frames is known as object detection and tracking. Along with that, it makes an attempt to determine and

B. Vaidya (✉)
GTU, Ahmedabad, India
e-mail: vaidya.bhaumik@gmail.com

H. Panchal · C. Paunwala
EC Department, SCET, Surat, India

© Springer Nature Singapore Pte Ltd. 2018

B. B. Chaudhuri et al. (eds.), *Proceedings of 2nd International Conference on Computer Vision & Image Processing*, Advances in Intelligent Systems and Computing 703, https://doi.org/10.1007/978-981-10-7895-8_16

narrate attributes of an object, thereby superseding the obsolete traditional method of supervising cameras manually. Object detection and tracking is consequential and challenging task in large number of vision applications such as vehicle navigation system, self-govern robot navigation and surveillance, traffic control.

Accurate object detection is an important step in object tracking. It needs an object detection step either in every frame or the frame in which the object appears first. While going for object tracking, basic operation is to make separation of interested object called 'foreground' from 'background' [1]. There are three important steps in video analysis—detection of objects, tracking of these objects from every frame to frame, and analysis of their tracks to determine their attributes. Some work in literature focused on developing algorithms for automatic detection and tracking which reduces the need of human surveillance. In some environments, the background is not available and can always be changed under critical situations like illumination changes, objects being introduced or removed from the scene. So, the background representation model must be more robust and adaptive [1].

Representation of shapes for matching is well-studied problem, and there are existing two types of categories: (1) Appearance-related cues of objects and (2) Contour or silhouettes or voxel sets [2]. The proposed algorithm is based on second type of category as it can deal with deformation in object shape easily [2].

The remaining paper is arranged as follows: Sect. 2 summarizes the related work done in object detection and tracking; Sect. 3 describes problem formulation; Sect. 4 describes FDCM algorithm in detail; Sect. 5 describes SURF algorithm in detail; Sect. 6 summarizes experiments and results of proposed algorithm.

2 Related Work

For detection of object, background modeling is very important. In basic model, background is modeled using an average, a median, or a histogram analysis over time [3]. In this case, once the model is computed, the pixels of the current image are categorized as foreground by thresholding the difference between the background image and the current frame. The statistical models are more robust to illumination changes and for dynamic backgrounds [4]. They can be categorized as Gaussian models, support vector models, and subspace learning models. Gaussian is the easiest way to represent a background [4]. It expects the history over the time of pixel's intensity values. But a single model cannot deal with the dynamic background, for example, waving trees, rippling water. To overcome this problem, the Mixture of Gaussians (MoG) or Gaussian Mixture Model (GMM) is used [4].

The second category utilizes more sophisticated statistical models, for example, Support Vector Machine (SVM) [5], Support Vector Regression (SVR) [6]. In estimation models, the filter is used to estimate the background. The filter may be a Wiener filter [7] or a Kalman filter [8] or Chebychev filter [9]. The Wiener filter works well for periodically changing pixels and it produces a larger value of the threshold for random changes that are utilized in the foreground detection. The

main advantage of the Wiener filter is that it lessens the uncertainty of a pixel value by representing how it varies with time. A drawback occurs when a moving object corrupts the history values [7]. The Chebychev filter slowly updates the background for changes in lighting and scenery while making the use of a small memory footprint with low computational complexity [9]. For cluster modeling K-means models, codebook models, neural network models can be used [10, 11].

It can be seen that most of the algorithms detect the moving object with different accuracy and conditions. The background initialization and updating of background after specific time are necessary in all of them. Some of the other techniques are based on contour-based approach where background initialization is not needed. Contour-based approach gives best result in most of the conditions like illumination change, bootstrapping, dynamic backgrounds [12]. Shape band algorithm [12] which is a contour-based model detects an object within a bandwidth of its sketch/contour. However, it not only detects/matches the identical template and input image but also captures the reasonable variation and deformation of object in the same class. Sreyasee and Anurag [2] presented an algorithm for object detection using part-based deformable template. They are using part-wise hierarchical structure decomposition for matching the template with the parts of object. For this, Gopalan's algorithm [13] is used for estimating the parts of the shape through approximate convex decomposition by measure of convexity and decomposed convex shape with junction of concave region.

The proposed algorithm uses convexity defect for finding junction points from where object convex parts are decomposed. Then parts are matched using FDCM and then SURF is used for tracking.

3 Proposed Algorithm

Figure 1 shows the flowchart of proposed algorithm for object detection and tracking. From the given input image or sketch, shape segmentation and part decomposition are done automatically and stored in the database on the server side. On client side from a test video, first frame is extracted and parts are matched using FDCM. Then using SURF, key points are extracted and matched. Individual steps are explained in detail, and results for individual steps are shown in next section.

3.1 Part-wise Decomposition

Figure 2 shows the flowchart for part-wise decomposition.

A. Input Sketch Model

The object sketches of human or animal are common articulation sketches used. It has large deformation if the viewpoint changes or part of object is in action and

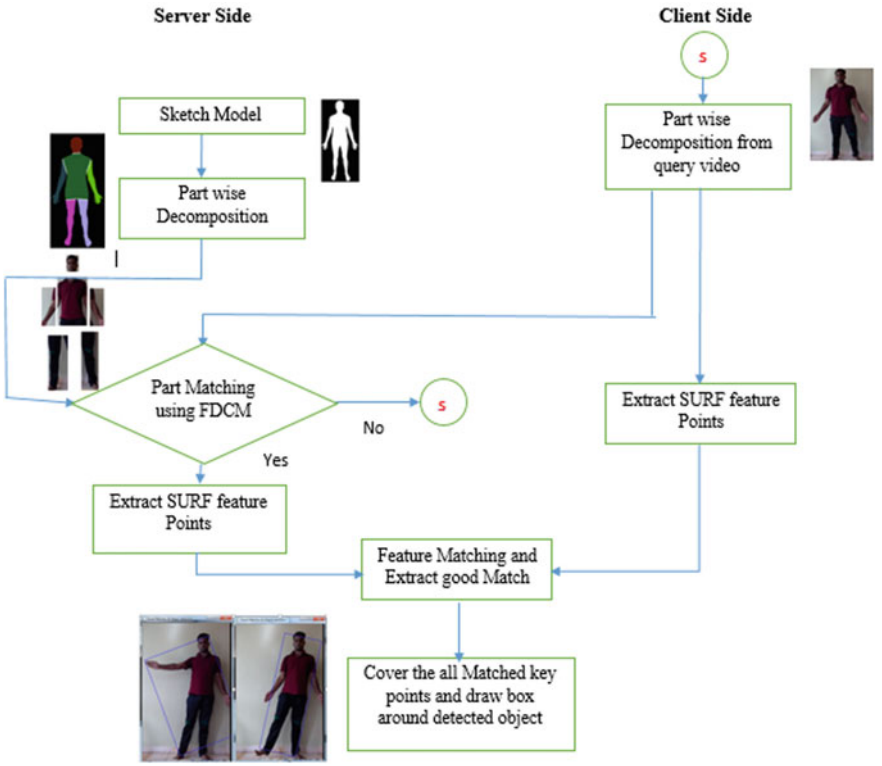
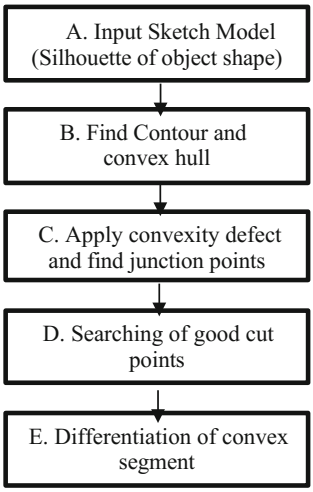


Fig. 1 Flowchart of proposed algorithm

Fig. 2 Flowchart for part-wise decomposition



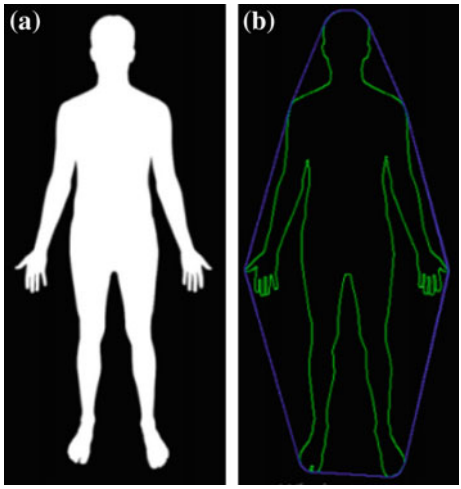


Fig. 3 **a** Human silhouette **b** green line denotes contour and blue line convex hull

movement. To deal with this situation, the proposed algorithm uses convexity defect to identify the joint from where the articulating shape can undergo movement.

B. Contour extraction and Convex Hull Creation

Contours can be explained simply as a curve joining all the continuous points (along the boundary), having same color or intensity. The contours are useful tool for shape analysis, object detection, and recognition. The Suzuki's algorithm [14] for retrieving contours from the binary image is used in proposed algorithm which gives only the outermost borders.

The convex hull [15] of a planar shape P is defined by finding the smallest convex region enclosing shape P. For example, to fit shape P into a polygon, it is necessary and sufficient that the convex hull CH(P) of shape P fits. It is usually the case that CH(P) has less points than shape P.

C. Convexity defects

The convexity defect [16] is actually the space between the convex hull and actual object. As shown in Fig. 5, human body silhouette shown in Fig. 3 can be described by five defect triangles (A, B, C, D, and E). Each triangle represents three coordinate points: (i) defect starting point (x_{ds}, y_{ds}), (ii) defect position point (x_{dp}, y_{dp}), and (iii) defect end point (x_{de}, y_{de}). Identification of defect points is a crucial and important task. If triangle A is considered, the defect is defined by considering each points of object contour in triangle A and making perpendicular online passes from points (x_{ds}, y_{ds}) and (x_{de}, y_{de}) and that is called as depth of that point. So the defect points of the triangle A can be described as the following vector:

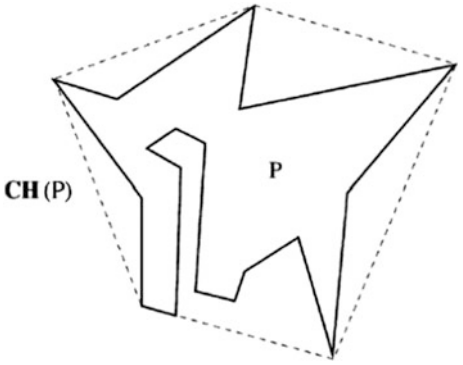


Fig. 4 Non-convex polygon P and its convex hull CH(P) (dashed line) [20]

$$V_d^A = <A(x_{ds}, y_{ds})\mathrm{A}\left(x_{dp}, y_{dp}\right)\mathrm{A}(x_{de}, y_{de})>$$

In real world, we have to deal with different shapes with uneven surface and also more complicated shapes. So finding only convexity defect position is not sufficient. It needs some modifications to deal with those artifacts. In proposed algorithm, for removing the unnecessary defect points from computation, all defect points which have less depth than depth threshold are discarded and remaining points are taken into account for further computation (Fig. 4).

D. Cut Points search

This step is used to find a set of points from the defect points that can be used to segment different parts from the sketch. The algorithm to find this set of cut point is given below:

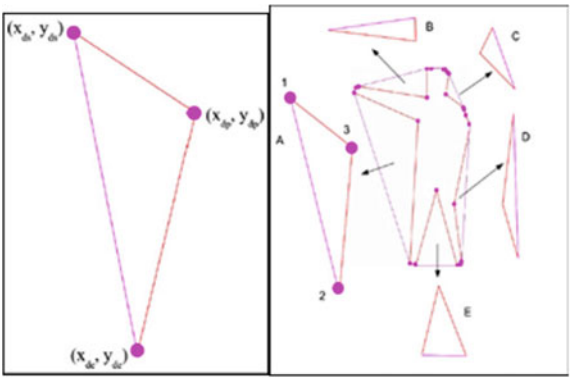


Fig. 5 Convexity defects human body silhouette

```
Program  Cut-Points (Defect Points)

d1= defect point1
ROI =  region of interest (ROI) around the defect point
M = Appropriate Mask Size
Repeat till No_of_contors > 1 in ROI :
   if No_of_contour < 1:
       Find d2= defect point 2 on Same Contour
   elseif No_of_defect_point == 2
       Joint (d1,d2) with line
   Else:
       M = 0.1*M
d2= Make first guess of defect point on another contour
d3= starting or end point of convexity defect in ROI
f1=  minimum Euclidean distance defect point
   if d2 is present :
       Join(d1,d2) with line
    elseif d3 is present:
       join(d1,d3) with line
   else :
       join(d1,f1) with line
   Move to Next Defect Point
end
```

Follow same procedure for each defect point and differentiate all segment from each other

E. Differentiation of convex segment

This step is used to segment different parts from the sketch by using defect positions and cut points found in previous steps. The algorithm for part segmentation is explained below, and results are shown in Fig. 6a and b:

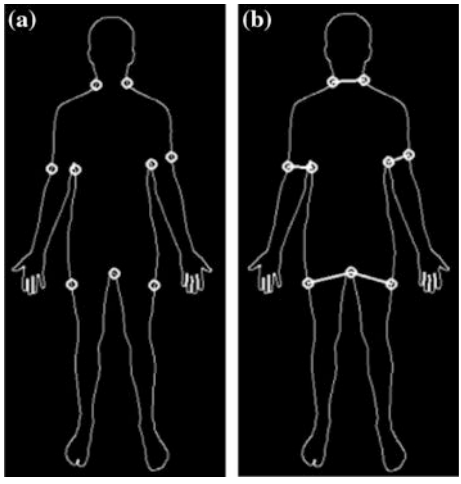


Fig. 6 **a** Defect position with cut point **b** differentiation of convex segment

```
Program Part Segmentation (Defect Points, Cut-Points)
for all cut points c1:
   Find nearest cut point c2
   Joint (c1, c2) with a line
   Repeat for all part segments:
   D1 = starting defect point
   D2 = Iterating defect point
   Contour points = [D1]
   While D2 != D1:
       Contour Points += D2
       D2= Next Defect Point
end
```

Contour points array represent close contour part. At the end fill different colors to represent different segment parts.

All experiment results for part identification and segmentation are shown in Fig. 7. The performance of proposed system is evaluated on ETHZ dataset [17] which is suitable for investigating performance of this algorithm. The dataset contains sketches of various objects like animals, mug, apple, and bottle. Main focus of work is to decompose the whole structure into the part-wise structure as the human can visualize.

Table 1 shows the experimental setup for ETHZ shape class dataset which contains sketches and images for five classes of objects: bottle, apple, mug, giraffe,

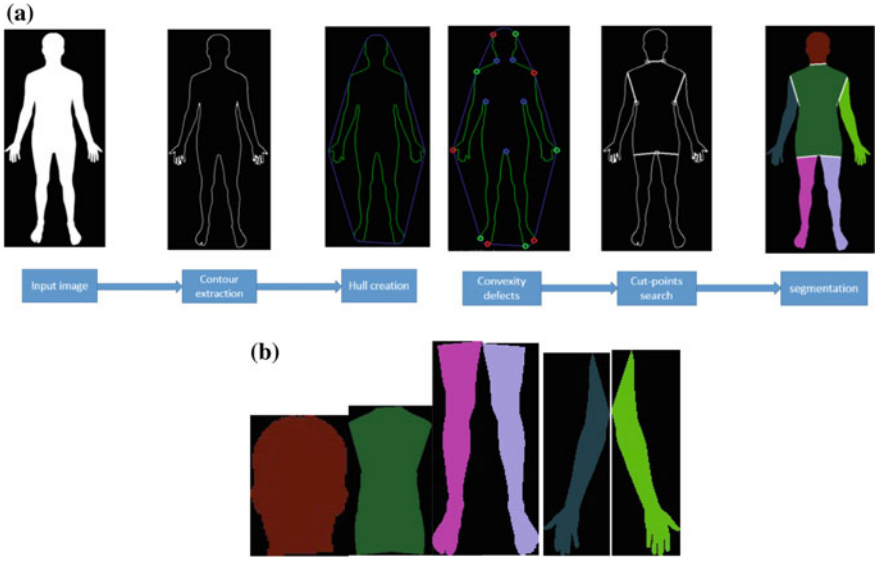


Fig. 7 **a** Step-wise analysis for human of proposed algorithm **b** segmentation of object in part-wise structure

Table 1 Experimental setup data

Sketch	Image size	Depth threshold	Mask size	Sketch
Human	294 × 549	20	150 × 150	Human
Apple	93 × 114	10	40 × 40	Apple
Bottle	200 × 300	7	60 × 60	Bottle
Giraffe	250 × 250	20	130 × 130	Giraffe
Mug	200 × 200	15	40 × 40	Mug

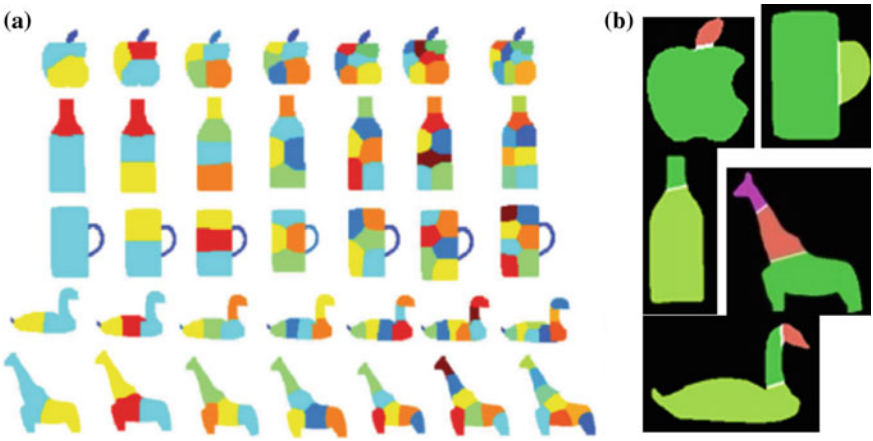


Fig. 8 **a** Result of part decomposition using the method by Gopalan [13], each row represents the results for specific shape as the user estimate for the number of part varies from 2 to 8. **b** Results using the proposed part decomposition method without user input

and swan [17]. As shown in the table, depth threshold and mask size values are decided based on the structure and convex hull area. First input data is silhouette of human body which is not from ETHZ dataset.

Figure 8 shows comparison between the results of part decomposition using Gopalan's algorithm [13] and the proposed algorithm. It can be observed that proposed algorithm outperforms Gopalan's algorithm because it gives different parts of object as human can visualize automatically without user explicitly specifying the number of parts as it was required in case of Gopalan's algorithm [13]. So the proposed algorithm will work well with objects of different shapes without worrying about the number of parts for decomposition.

4 Part Matching Using Fast Directional Chamfer Matching (FDCM)

After segmenting sketch into several parts, there is a need to match the parts obtained from the client-side sketch to the parts that are already stored in server database. For real-time operation, this function should be as fast as possible. There are many shape-matching algorithms available, but the chamfer matching is the most preferred one based on speed and robustness compared to other algorithms [18]. There are different types of variations in Chamfer Matching algorithm which are described in literature like (i) Chamfer matching (ii) Directional Chamfer Matching (iii) Fast Directional Chamfer Matching. Chamfer matching provides a fairly smooth measure of fitness and can tolerate small rotations, misalignments, occlusions, and deformations. Chamfer matching becomes less reliable in the presence of background clutter [18]. To improve robustness, Directional Chamfer matching has been introduced by incorporating edge orientation information into the matching cost. In DCM, match cost is more in terms of time and memory and to overcome that FDCM was introduced. In that directional matching cost is optimized in three stages by linear representation of the template edges, describing a three-dimensional distance transform representation and presenting a directional integral image representation over distance transforms. Using these intermediate structures, exact computation of the matching score can be performed in sublinear time in the large number of edge points. In the presence of many shape templates, the memory requirement also reduces drastically. In addition, smooth cost function allows binding the cost distribution of large neighborhoods and skipping the bad hypotheses [18]. Experimental results for fast direction chamfer matching are shown in Fig. 9.

Hand-drawn sketch shown in white box and green box indicates detected object using given sketch. After detecting parts using FDCM, SURF is used for continuous detection and tracking extracted feature point in image sequences.

5 Speeded Up Robust Features (SURF)

One of the most important tasks is to select the ‘interest points’ at distinct location in an image. An output of FDCM is considered as region of interest (ROI). SURF detector and descriptor is used for finding minimum feature points in ROI [19].

SURF uses Hessian matrix approximation for interest point detection which performs well in terms of accuracy [19]. SURF detects blob-like structure at locations with the maximum value of determinant. For reducing the computational complexity, integral images are used which uses only four memory locations and three addition operations to calculate the intensities inside a rectangular region of any size. Hence, the calculation time of this is independent from the size of an image [19]. For making SURF descriptor invariant to image rotation, Haar wavelet

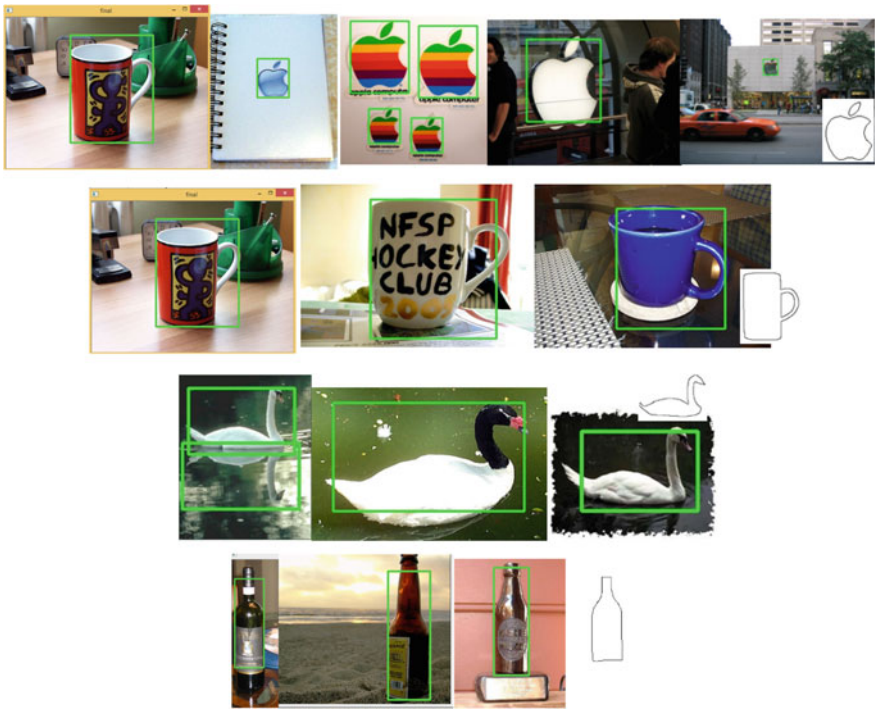


Fig. 9 Output of FDCM algorithm on ETHZ dataset

response is calculated in x- and y-direction with the radius of 6 s around the interest point, where 's' is the scale at which the interest point is detected. Then wavelet response is computed and Gaussian weight is given centered at interested point. The sum of all responses within the window $\pi/3$ is calculated, and with use of this, the dominant orientation is estimated. The horizontal and vertical responses are summed, and longest vector of the window defines the orientation of the interest point. Figure 10 shows the experimental results using SURF algorithm for different conditions like scaling and rotation.

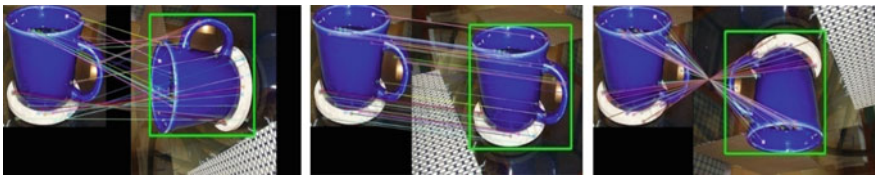


Fig. 10 Output of surf, left-side image is input image, and right-side image shows detected object image

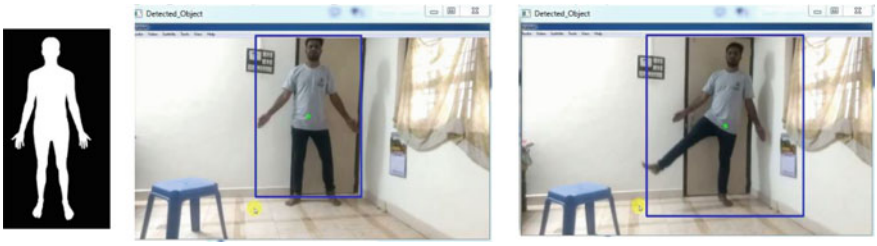


Fig. 11 Results of human tracking in two different frames with different shapes

6 Experiments and Results

Experiment results for object tracking for different objects like human, book, and bottle are shown in Figs. 11, 12, and 13. The results are taken in different environments to prove the effectiveness of the proposed algorithm. Figure 11 shows human tracking in two different frames, where shape of the body is different and there is deformation in parts of body. In Fig. 12, the object of interest is book so the algorithm tracks book in different frames even when bottle is introduced or moved in the frame. In Fig. 13, bottle is tracked but book is not tracked. Both results are shown with start frame, two intermediate frames, and end frames. The results indicate that proposed algorithm performs well in challenging conditions.

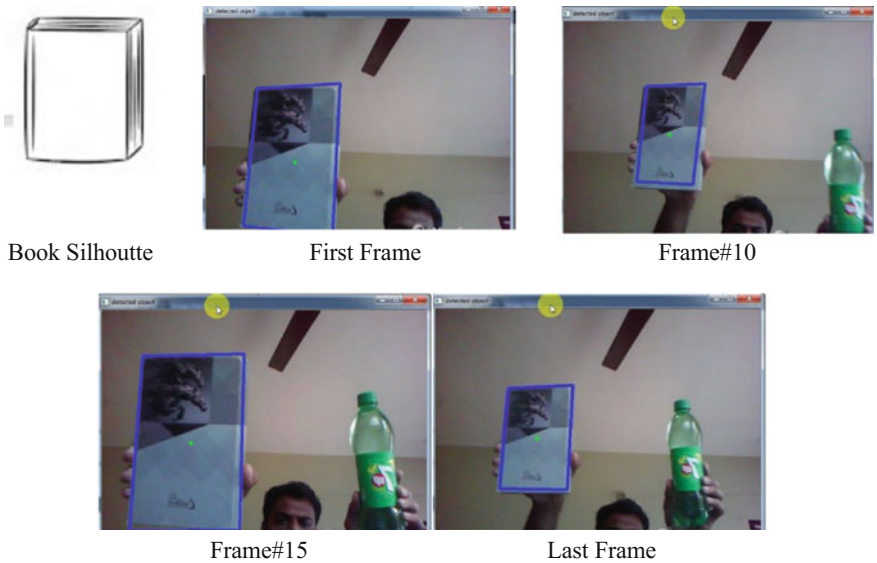


Fig. 12 Results of book tracking in the presence of bottle movements in different frames

Fig. 13 Results of bottle tracking in the presence of book movement in different frames

7 Conclusion

In this paper, novel algorithm has been proposed for object detection and tracking using part decomposition and SURF. A method to decompose non-planar structure in part-wise planar structure using silhouette of object is also proposed. The method works very efficiently when input silhouette is without distortion. The computation complexity is also less than state-of-the-art algorithms. As in Gopalan's [13] algorithm and other state-of-the-art algorithms required prior knowledge about number of parts in given structure or user has to decide number of parts to be decomposed, but this method works in all situation without the prior knowledge about number of parts. As shown in experiment results, proposed algorithm gives better results in terms of part decomposition for all input sketches of ETHZ dataset. SURF can fulfill the requirement of real time though part decomposition and matching take more time, so in future work time taken for part decomposition can be modified.

Informed consent Additional informed consent was obtained from all individual participants for whom identifying information is included in this article.

References

1. Bouwmans, T.: Traditional and recent approaches in background modeling for foreground detection: An overview. Computer Science Review, 11, pp. 31–66 (2014).
2. Bhattacharjee, S. D., Mittal, A.: Part-based deformable object detection with a single sketch. Computer Vision and Image Understanding, vol. 139, pp. 73–87 (2015).
3. Bouwmans, T.: Recent advanced statistical background modeling for foreground detection-a systematic survey. Recent Patents on Computer Science, vol. 4 no. 3, pp. 147–176 (2011).
4. Stauffer, C., Grimson, W. E. L.: Adaptive background mixture models for real-time tracking. In: IEEE Computer Society Conference on. Computer Vision and Pattern Recognition, Vol. 2, pp. 246–252, IEEE (1999).
5. Lin, H. H., Liu, T. L., & Chuang, J. H.: A probabilistic SVM approach for background scene initialization. In: International Conference on Image Processing Vol. 3, pp. 893–896. IEEE (2002).
6. Wang, J., Bebis, G., Miller, R.: Robust video-based surveillance by integrating target detection with tracking. In: Conference on Computer Vision and Pattern Recognition Workshop, CVPRW'06, IEEE (2006).
7. Toyama, K., Krumm, J., Brumitt, B., Meyers, B.: Wallflower: Principles and practice of background maintenance. In: The Proceedings of the Seventh IEEE International Conference on Computer Vision, Vol. 1, pp. 255–261, IEEE (1999).
8. Ridder, C., Munkelt, O., Kirchner, H.: Adaptive background estimation and foreground detection using kalman-filtering. In: Proceedings of International Conference on recent Advances in Mechatronics, pp. 193–199 (1995).
9. Chang, R., Gandhi, T., & Trivedi, M. M.: Vision modules for a multi-sensory bridge monitoring approach. In: Proceedings of 7th International IEEE Conference on Intelligent Transportation Systems, pp. 971–976, IEEE (2004).
10. Butler, D. E., Bove, V. M., & Sridharan, S.: Real-time adaptive foreground/background segmentation. In: EURASIP Journal on Advances in Signal Processing (2005).
11. Kim, K., Chalidabhongse, T. H., Harwood, D., Davis, L.: Background modeling and subtraction by codebook construction. In: International Conference on Image Processing (ICIP'04), Vol. 5, pp. 3061–3064, IEEE., 2004.
12. Bai, X., Li, Q., Latecki, L. J., Liu, W., Tu, Z.: Shape band: A deformable object detection approach. In: IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2009, pp. 1335–1342, IEEE (2009).
13. Gopalan, R., Turaga, P., Chellappa, R.: Articulation-invariant representation of non-planar shapes. In: Computer Vision–ECCV 2010, 286–299 (2010).
14. Suzuki, S.: Topological structural analysis of digitized binary images by border following. In: Computer vision, graphics, and image processing, vol 30 no. 1, pp. 32–46 (1985).
15. Graham, R. L., Yao, F. F.: Finding the convex hull of a simple polygon. In: Journal of Algorithms, vol 4 no. 4, pp. 324–331 (1983).
16. Youssef, M. M., Asari, V. K.: Human action recognition using hull convexity defect features with multi-modality setups. In: Pattern Recognition Letters, vol. 34 no. 15, pp. 1971–1979 (2013).
17. ETHZ datasets, http://www.vision.ee.ethz.ch/datasets/.
18. Liu, M. Y., Tuzel, O., Veeraraghavan, A., Chellappa, R.: Fast directional chamfer matching. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 1696–1703. IEEE (2010).
19. Bay, H., Ess, A., Tuytelaars, T., Van Gool, L. Speeded-up robust features (SURF). In: Computer vision and image understanding, vol. 110 no. 3, pp. 346–359 (2008).
20. Zunic, J., Rosin, P. L.: A new convexity measure for polygons. In: IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 26 no. 7, pp. 923–934 (2004).

Proceedings of the International Conference on Intelligent Computing and Control Systems (ICICCS 2019)
IEEE Xplore Part Number: CFP19K34-ART; ISBN: 978-1-5386-8113-8

CNN based Traffic Sign Classification using Adam Optimizer

[1]Smit Mehta,[2]Chirag Paunwala,[3]Bhaumik Vaidya
[1,2,3] Department of Electronics and Communication
Sarvajanik College of Engineering and Technology Surat, India
[1]mehtasmith19@gmail.com,[2]chirag.paunwala@scet.ac.in

Abstract—**An automatic detection and classification of traffic signs is an important task in Advanced Driver Assistance System (ADAS).Convolutional Neural Network (CNN) has surpassed the human performance and shown the great success in detection and classification of traffic signs. The paper proposes an approach based on the deep convolutional network for classifying traffic signs. The Belgium traffic sign dataset (BTSD) is used for evaluation and experiment results shows that the proposed method can achieve competitive results compared with state of the art approaches. Different activations and optimizers are used to evaluate the performance of proposed architecture and it is observed that Adam (Adaptive Moment Estimation) optimizer and softmax activation performs well.**

Keywords- Adam optimizer, Belgium traffic sign dataset (BTSD), Convolutional Neural Network (CNN), Dropout, Traffic Sign Classification.

I. INTRODUCTION

Traffic sign detection is considered to be one of the most important part in the advance driver assistance system as it is necessary to detect traffic signs before they can be identified.traffic signs are designed with definite shape and color to provide the valuable information like traffic rules,route directions and different road conditions to drivers for safe driving.The main objective of designing the advance driver assistance system is to reduce the number of road accidents and wrong decisions. Designing smart vehicles for detecting traffic signs from the environment is one of the hot topics in today's traffic sign detection systems. Traffic sign detection and recognition system is mainly divided into two stages. First stage is the localization of traffic signs and second stage is the classification of detected traffic signs. Classification of traffic signs can be accomplished by using neural networks.

Traffic sign detection system reminds and warns the driver about upcoming traffic signs coming in a way. These systems have the ability to detect traffic signs even in worst case scenarios, but still sometimes it is difficult to detect traffic signs in challenging conditions .Some of the most common problems that may be encountered while classifying and detecting traffic signs are color fading, partial occlusion by surrounding obstacles, variation in lighting and weather conditions, shadowing, reflections from the sign boards during day hours and motion blurring. The detection and recognition of traffic signs has a variety of important application which includes advanced driver assisting systems, road surveying, autonomous driving, building and maintaining maps of signs, mobile mapping systems, vehicle navigations system, surveillance and self-govern robot navigation systems.

The rest of the paper is organized as follows. Section 2 reviews previous works on traffic sign classification and detection. The details of proposed system and proposed network architecture are described in Section 3.Section 4 shows the experiment results and Section 5 concludes the paper.

II. RELATED WORK

In general, traffic sign detection and classification methods are divided into two types. First is traditional or conventional methods and second is end to end learning or deep learning based methods. Most of the research works conducted on traffic sign classification and detection are based on these methods.

As traffic signs have definite shape and color ,color based methods and shape based methods are widely used for the detection and of the traffic signs. Color thresholding [1],color invariants [2],color segmentation [3] are the most common color based methods of detection. These methods convert the RGB color space [3] to other color space like, HIS[4], YCbCr [5] because RGB color space is very sensitive to environment. Drawbacks of these methods are Color fading and variation in lighting conditions. To overcome the problems of color based method, shape based methods like Hough transform [6] and EDCircle [1] are used widely. But these methods are very slow and takes more computational time for detection. So these methods can't applied for the practical use.

To improve the results sliding window based method [7] was proposed which uses HOG [8] and Viola-Jones [9] for traffic sign detection. These methods are complex and very time consuming. By knowing the fact that traffic signs are located in the two sides of roadways researchers have found out the new approach based on region proposal [10,11,12].Region based object proposal methods are widely used as compared to sliding window based methods as it reduces the search area.

Most of the traffic sign detection method uses different classifiers like neural networks [13],support vector machines

[14],convolutional neural network [15,16], random forests [13] and nearest neighbor [17] to classify the traffic signs from the environment. Out of these classifiers convolutional neural network is widely used for image classification problem.

In [2], a method based on color invariants and pyramid histogram of oriented gradients is used for traffic sign detection.

Approach based on the fully convolution network guided traffic sign proposals followed by the deep Convolutional Neural Network (CNN) has been used in [10] which reduces the search area classifies the traffic sign proposals.

In the study by Dan Ciresan,Ueli Meier,Jonathan Masci,Jurgen Schmidhuber [18], various Deep Neural Networks (DNNs) trained on differently preprocessed data were combined into Multi Column Deep Neural Network (MCDNN) for boosting the performance of the classification and Detection.

Most of the research work focuses on the CNN [15,19,20] for classifying traffic signs as they have the ability of learning features in a hierarchical way.

This work proposes an approach for traffic sign detection which uses the CNN for classifying the traffic signs from the publicly available Belgium traffic sign dataset (BTSD) [21].Experiment results based on this network architecture shows that CNN works efficiently and gives better accuracy.

III. PROPOSED METHOD

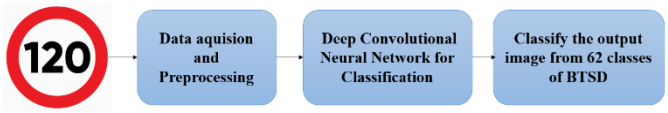


Figure 1 Pipeline of the proposed method

Fig.1 shows the the pipeline for the proposed method which includes three stages :data acquisition, preprocessing and classification..

A. Data Acquisition::

Traffic sign images in the BTSD database are extracted from the video sequences. The database does not take account of the disturbance of motion blur,fogy or rainy weather.

B. Convolutional Neural Network:

A convolutional neural network is a kind of feed forward neural network widely used for the image based classification object detection and object recognition.The basic principle behind the working of CNN is the idea of using convolution, which produces the filtered feature maps stacked over each other. Fig. 2 shows the key operations involved in CNN which includes convolution, non-linearity spatial pooling and convolved feature maps generated by this operation.

A CNN is made up of different Layers as shown in the Fig.3.Each layer has a simple application of transforming an input 3D volume to an output 3D volume that may or may not have trainable parameter.

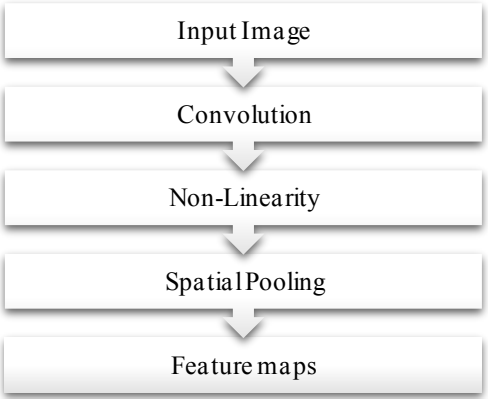


Figure 2 Key operations in CNN

C. CNN Architecture :

Fig.3 shows the architecture of CNN which this paper has proposed for classifying traffic signs from the Belgium traffic sign dataset (BTSD). In Convolutional neural network neurons are arranged in 3 dimensions width, height and depth where depth refers to the total number of filters. The network consists of three convolution layers each followed by the maxpooling layers and two fully connected layers. The dropout layer has been used in between two fully connected layers.The proposed network takes the color image of size 64 × 64 as an input and classifies it into RGB image as an input and classify it into one of the 62 classes from the dataset.

Convolutional layer gives convolved feature map as an output after applying the dot product between the weights of the filter and a small region of input to which they all are connected to. The network uses 32 filters with a size 3×3, so the output volume of a first convolution layer is given by [62×62×32]. After that pooling layer is used which is basically used to perform downsampling operation.Output volume of this layer has a size of [31×31×32] as it uses maxpooling with a stride 2 and filter size 2×2. Likewise the output volume of each layer can be calculated. In Convolutional neural network, the size of the output volume for each layer can be calculated using following formula:

$$\frac{W - F + 2P}{S} + 1 \qquad (1)$$

Where W is the size of input ,F is the size of filter ,S is the value of stride used in maxpooling layer,P is the amount of zero padding used on the border. After performing convolution and pooling operation, the output of maxpooling layer is flatten into a single vector as shown in Fig.3.Then fully connected layer is used to compute the scores for 62 classes as there are 62 classes present in the Belgium traffic sign dataset (BTSD).

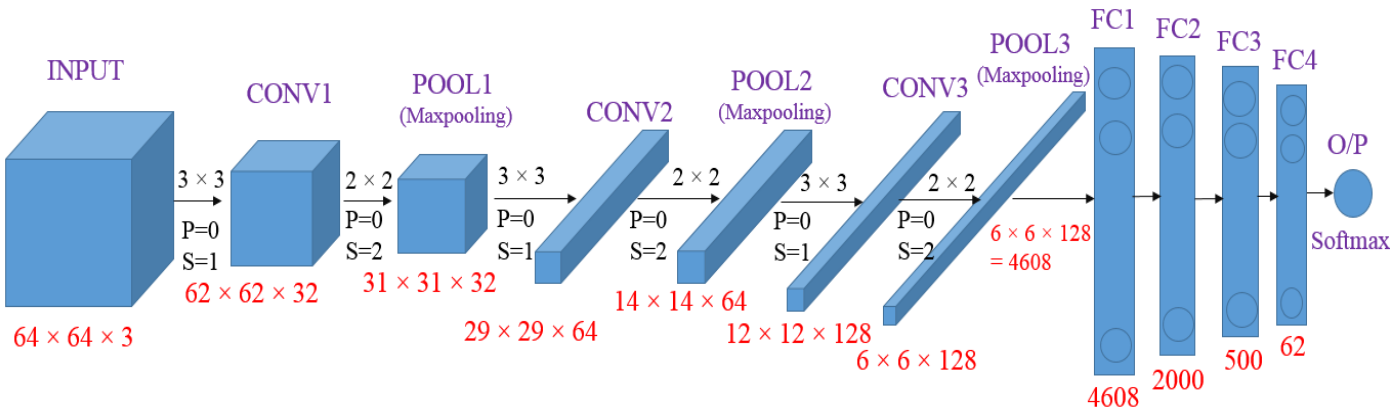


Figure 3 Proposed network architecture of convolutional neural network

IV. EXPERIMENT RESULTS

A. Dataset:

In this paper Belgium traffic sign dataset (BTSD) [21] is used as a database which has been adopted widely in most of the research work for classifying and detecting traffic signs. The dataset set contains 4575 training images and the 2520 test images.Traffic sign images of these database extracted from the environment does not take account of the disturbance occurred due to motion blur and rainy weather. As shown in the Fig.4 the traffic signs of Belgium traffic sign database are divided into five categories,i.e.,Prohibitive signs with red color and circular shape, Mandetory signs with blue color and circular shape, Danger signs with red color and triangular shape,Derestriction signs and rest of the traffic signs fall under the category of unique signs.

Figure 4 Different classes of Belgium traffic sign dataset (BTSD)

B. Experimental setup:

Proposed system is implemented with Keras library and using Tensorflow as backend engine. The experiments are conducted on Intel (R) Core (TM) i5–7500 CPU @ 3.40 GHz with 8GB RAM , 64-bit operating system. Implementation of CNN (Convolutional Neural Network) was performed on Belgium traffic sign dataset (BTSD) [21] dataset. Adam (Adaptive Moment Estimation) optimizer [25] is used to train the network.Relu activation function is used for each neuron of CNN. Loss function used is sparse categorical cross entropy.

C. Results on Belgium traffic sign Dataset (BTSD):

The proposed architecture is applied on Belgium traffic sign dataset (BTSD). This paper emphasizes on the solution to traffic sign classification and detection. This work shows an effective implementation of classification with the network architecture that performs well. Table I Shows the layer wise details of proposed network architecture like output volume of the each layer and no of parameters to be handled by each layer.

TABLE I SPECIFICATIONS OF CNN

Layer (Type)	Output shape	Parameters
Conv_1	62,62,32	896
Max Pooling_1	31,31,32	0
Conv_2	29,29,64	18496
Max pooling_2	14,14,64	0
Conv2D_3	12,12,128	73856
Max pooling_3	6,6,128	0
Flatten	4608	0
Dense _1	2000	9218000
Drop out_1	2000	0
Dense _2	500	1000500
Drop out_2	500	0
Dense_3	62	31062

Total parameters to be handled by the proposed network is 13,89,543.ReLu activation function has been used in between the hidden layers of the network.ReLu does not activate all neurons at the same time which is considered to be the major advantage of using it as an activation function compared to other activation function.

TABLE II RESULTS ON BTSD BEFORE DROPOUT

Dataset	Total Trainable Parameters	Training Accuracy	Testing Accuracy	Epochs
BTSD	10342810	98.26	92.18	10

As can be seen from the Table II, training accuracy is higher than testing accuracy. This issue is called overfitting. There are lots of ways to improve the network performance.In case of overfitting: dropout [22], batch normalization [23], size of datasets, maxpooling, relu layer (non linearity) are the widely used methods. Table III shows the results on BTSD after applying the dropout as a regularization.Dropout randomly drops some of the neurons.

TABLE III RESULTS ON BTSD AFTER APPLYING DROPOUT

Dataset	Total Trainable Parameters	Training Accuracy	Testing Accuracy	Epochs
BTSD	10342810	96.61	97.06	10

TABLE IV DIFFERENT OPTIMIZERS USED IN PROPOSED ARCHITECTURE

Dataset	Training Accuracy	Testing Accuracy	Epochs	Optimizer	Dropout
BTSD	82.95	87.26	10	SGD	0.2
BTSD	97.40	96.31	10	Adam	0.2
BTSD	85.73	90.40	10	SGD	0.3
BTSD	96.61	97.06	10	Adam	0.3

Table IV shows the different optimizers used to train the network with dropout values.Adam optimizer with 0.3 dropout value shows the better result compared to SGD (Stochastic Gradient Descent) optimizer as it adaptively learn weights. In SGD ,weights are updated incrementally after each epoch (means passing over the whole training dataset).Adam uses following equation to update weights.

$$w = w - \alpha J_{dw} \quad (2)$$

$$J_{dw} = \beta J_{dw}(1-\beta)dw \quad (3)$$

Where α is learning rate and β is hyper parameter and its recommended value is 0.9.

TABLE V DIFFERENT ACTIVATION USED IN PROPOSED ARCHITECTURE

Dataset	Training Accuracy	Testing Accuracy	Epochs	Activation	Dropout
BTSD	95.08	94.68	10	Sigmoid	0.3
BTSD	96.61	97.06	10	Softmax	0.3

Table V shows the different activation applied on the network and furthermore dropout has been used to improve the results.Training and testing accuracy using softmax activation shows the better result compared to sigmoid.The main advantage of using softmax as an activation compared to other activation is it computes the probability of each and every class.And output in terms of probability is more helpful in predicting and determining the target class.

Fig.5 shows the result of training and testing accuracy plotted against number of epochs without using dropout. And results of training and testing accuracy with dropout is shown in Fig.6.In a same way result of training and testing loss plotted against epochs are shown in Fig.7 and Fig.8.

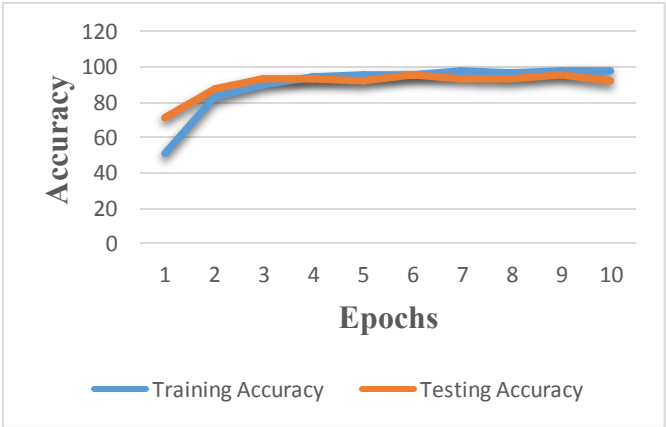


Figure 5 Training and testing accuracy without dropout

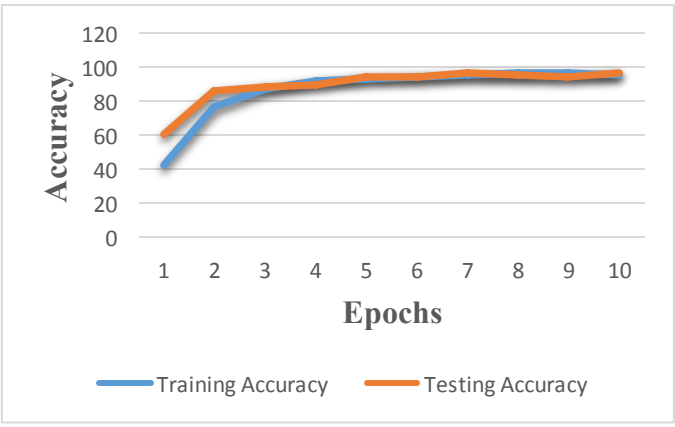


Figure 6 Training and testing accuracy with dropout

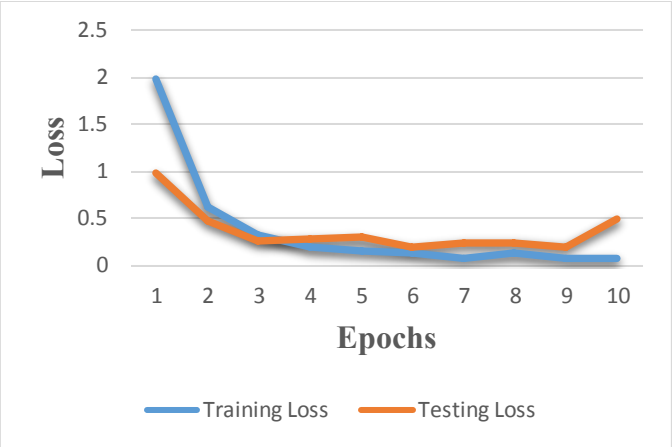


Figure 7 Training and testing loss without dropout

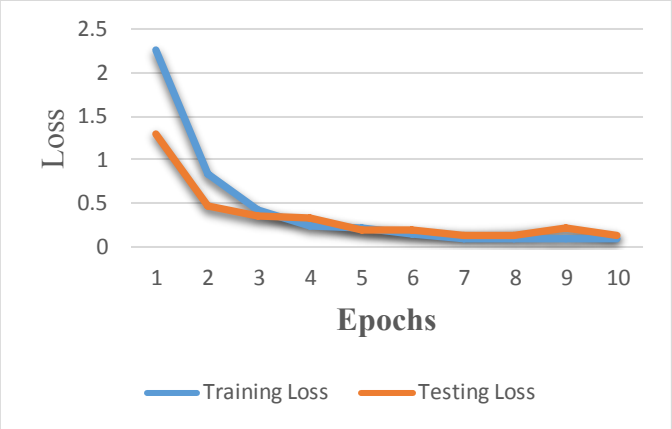


Figure 8 Training and testing loss with dropout

CONCLUSION

In this paper, an approach based on the Convolutional Neural Network (CNN) for classifying traffic signs is proposed. Evaluation was carried out on the publicly available Belgium traffic sign dataset (BTSD) and the architecture showed the better accuracy. Furthermore, it uses dropout to overcome the problem of overfitting as it randomly drops some of the units from neural network and it is also considered to be the most efficient way of model averaging. The proposed network architecture uses softmax as an activation in an output layer because it calculates a probability of every possible class.For training the network Adam optimizer is used and it is observed that it adapts faster compared to other optimizers like SGD.

ACKNOWLEDGMENT

We thank VISICS, ESAT, KU Leuven for providing us the valuable sources like Belgium traffic sign dataset (BTSD) and allowing us to use it for our research work.

REFERENCES

[1] Selcan Kaplan Berkaya, Huseyin Gunduz, Ozgur Ozsen, Cuneyt Akinlar, Serkan Gunal, "On circular traffic sign detection and recognition," Journal of Expert Systems with Applications, Elsevier publication ,Volume 48, pp. 67-75, ISSN 0957-4174, 2016

[2] Haojie Li, Fuming Sun, Lijuan Liu, Ling Wang, "A novel traffic sign detection method via color segmentation and robust shape matching", Journal of Neurocomputing, Elsevier publication, Volume 169, pp. 77-88, ISSN 0925-2312, 2015

[3] A.De La Escalera,L.E.Moreno, M.A.Salichs, J.M.Armingol, Road traffic sign detection and classification, IEEE Trans. Ind. Electron. 44 (6) (1997) 848–859.

[4] S. Maldonado-Bascon, S. Lafuente-Arroyo, P. Gil-Jimenez, H. Gomez-Moreno, and F. Lopez-Ferreras, "Road-sign detection and recognition based on support vector machines," IEEE Transactions on Intelligent Transportation Systems, vol. 8, no. 2, pp. 264–278, 2007.

[5] HERBSCHLEB E, Real-time traffic sign detection and recognitio[C] IS &T／SPIE Electronic Imaging. International Society for Optics and Photonics, 2009 : 72570A－72570A-12

[6] N. Barnes, G. Loy, D. Shaw, and A. Robles-Kelly, "Regular polygon detection," in Proceedings of the Tenth IEEE International Conference on Computer Vision, 2005, pp. 778–785.

[7] G. Wang, G. Ren, Z. Wu, Y. Zhao, and L. Jiang, "A robust, coarse-to-fine traffic sign detection method," in Proceedings of IEEE International Joint Conference on Neural Networks, 2013.

[8] Y. Xie, L.-f. Liu, C.-h. Li, and Y.-y. Qu, "Unifying visual saliency with hog feature learning for traffic sign detection," in Proceedings of the IEEE Intelligent Vehicles Symposium, 2009, pp. 24–29.

[9] C. G. Keller, C. Sprunk, C. Bahlmann, J. Giebel, and G. Baratoff, "Realtime recognition of u.s. speed signs," in Proceedings of the Intelligent Vehicles Symposium, 2008, pp. 518–523.

[10] Yingying Zhu, Chengquan Zhang, Duoyou Zhou, Xinggang Wang, Xiang Bai, Wenyu Liu, "Traffic sign detection and recognition using fully convolutional network guided proposals",Journal of Neurocomputing, Elsevier publication ,Volume 214, pp. 758-766, ISSN 0925-2312, 2016

[11] P.Krahebuhl, V.Koltun, Geodesic object proposals,in: proceeding of ECCV,Springer,Zurich,Switerland,2014,pp.725-729.

[12] R.Girshick, Fastr-cnn,in:Proceedings of ICCV,2015,pp.1440–1448.

[13] D. Cireşan, U.Meier, J.Masci, J.Schmidhuber, Multi-column deep neural network for traffic sign classification, Neural Netw.32 (2012) 333–338.

[14] S. Maldonado-Bascon, S. Lafuente-Arroyo, P. Gil-Jimenez, H. Gomez-Moreno, and F. Lopez-Ferreras, "Road-sign detection and recognition based on support vector machines," IEEE Transactions on Intelligent Transportation Systems, vol. 8, no. 2, pp. 264–278, 2007.

[15] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," Proceedings of the IEEE, vol. 86, no. 11, pp. 2278–2324, November 1998.

[16] Hamed Habibi Aghdam, Elnaz Jahani Heravi, Domenec Puig, "A practical approach for detection and classification of traffic signs using Convolutional Neural Networks," Journal of Robotics and Autonomous Systems, Elsevier publication ,Volume 84, pp. 97-112,ISSN 0921-8890, 2016

[17] A.Ruta,Y.Li,X.Liu, Real-time traffic sign recognition from video by class-specific discriminative features,Pattern Recognit.43(1)(2010) 416-430.

[18] Dan Ciresan,Ueli Meier,Jonathan Masci,Jurgen Schmidhuber, "Multi-Column Deep Neural Network for traffic sign classification" Journal of Neural Networks, Elsevier publication, pp. 333-338, ISSN 0893-6080, 2012

[19] S. Behnke, Hierarchical Neural Networks for Image Interpretation,ser. Lecture Notes in Computer Science. Springer, 2003, vol. 2766.

[20] P. Simard, D. Steinkraus, and J. Platt, "Best practices for convolutional neural networks applied to visual document analysis," in Seventh International Conference on Document Analysis and Recognition,2003.

[21] Timofte R, Zimmermann K, Gool LJV (2009) Multi-view traffic sign detection, recognition, and 3d localisation. In: WACV. USA, pp 1–8

[22] T. Wang, J. Huan and B. Li, "Data Dropout: Optimizing Training Data for Convolutional Neural Networks," 2018 IEEE 30th International Conference on Tools with Artificial Intelligence (ICTAI), Volos, 2018, pp. 39-46.

[23] V. Thakkar, S. Tewary and C. Chakraborty, "Batch Normalization in Convolutional Neural Networks — A comparative study with CIFAR-10 data," 2018 Fifth International Conference on Emerging Applications of Information Technology (EAIT), Kolkata, 2018, pp. 1-5.

[24] P.Sermanet,Y.LeCun, Traffic sign recognition with multi-scale convolutional networks,in: The 2011 International Joint Conference on Neural Networks (IJCNN), IEEE, SanJose, California, USA ,2011,pp.2809–2813.

[25] Kingma, Diederik P., and Jimmy Ba. "Adam: A method for stochastic optimization." arXiv preprint arXiv:1412.6980 ,2014, pp. 1-15.

PERFORMANCE OF DIFFERENT OPTIMIZERS FOR TRAFFIC SIGN CLASSIFICATION

Ishita Joshi
Student Member, IEEE
Electronics and Communication Dept.
Sarvajanik College of Engineering and Technology
Surat, India
ishitahjoshi@gmail.com

Milauni Desai
Electronics and Communication Dept.
Sarvajanik College of Engineering and Technology
Surat, India
desai.milauni23@gmail.com

Sannidhi Bookseller
Electronics and Communication Dept.
Sarvajanik College of Engineering and Technology
Surat, India
sannidhibookseller@gmail.com

Rucha Mohod
Electronics and Communication Dept.
Sarvajanik College of Engineering and Technology
Surat, India
ruchavmohod01@gmail.com

Chirag N. Paunwala
SMIEEE
Electronics and Communication Dept.
Sarvajanik College of Engineering and Technology
Surat, India
chirag.paunwala@scet.ac.in

Bhaumik Vaidya
Electronics and Communication Dept.
Sarvajanik College of Engineering and Technology
Surat, India
bhaumik.vaidya@scet.ac.in

Abstract*— The use of traffic signs is vital especially when travelling in the highways or the hills in adverse environmental conditions. The pace of advancements in the field of Machine learning has opened doors for scopes of improvement in the performance of Convolutional Neural Network Architectures dedicated to classification of traffic signs. Speed is as important as accuracy for such problems in the field of Advance Driver Assistance Systems and the use of GPU instead of CPU gives the benefit of parallel processing. Gradient Descent helps navigating towards the minima of the loss function. Purpose of various gradient descent optimizing algorithms is to help in quicker convergence. This proposed algorithm comprises of a compact Convolutional Neural Network architecture that was trained on GPU using RMSProp, Adam and Nadam optimizers on the BelgiumTS dataset. RMSProp and Adam caused either under-fitting or over-fitting that was resolved by Nadam used with an appropriate dropout with 97.51 training accuracy and 96.78 testing accuracy. The predictions on test images convey that the architecture trained using Nadam works perfectly for blurry images, positionally challenging images and images with uneven illumination.

Keywords*— convolutional neural networks, GPU, machine learning, optimizers, Traffic sign classification

I. INTRODUCTION

Traffic signs serve the purpose to maintain traffic discipline. But the use of traffic signs becomes more crucial on the highways, especially during night; on hilly regions with sharp curves; during uneven visual conditions as driving at night or during foggy climate; to indicate construction areas or diversions on expressways. In these discussed cases, traffic signs do more than just maintaining discipline and correct interpretation of traffic sign is extremely important. Quick identification of traffic signs is vital as it will give more time for taking required action which was the motivation for implementing the proposed CNN Architecture on a multi-core GPU. The aim is to accomplish traffic sign classification and to train the proposed architecture using three optimizers. There are many challenges in the identification of traffic signs such as position, occlusion, obstacles, illumination and low image quality at high speed.

In the domain of traffic sign detection and classification, the research began with use of conventional algorithms derived using the concept of image processing. Lately, because of the easy availability of data, the research has inclined more towards the use of Convolutional Neural Networks derived using the concepts of Machine Learning that focuses greatly on building an architecture, training it using a huge collection of data and optimizing it so as to obtain desired results. Software implementation might have been done a lot many times but implementation on a hardware component that is highly computationally powerful is not undertaken as frequently.

Initial section includes brief overview of the optimizers used. The result section includes an outline of the datasets available and feature description of hardware used. Experimentation results comprise of accuracy plots and a summarized table that help to interpret and compare the performance of the three optimizers in a better way.

II. RELATED WORK

Researches on traffic sign detection and recognition are being continued from last decade has achieved ample amount of theoretical achievements; to put these achievements in actual practice seems quite difficult. Most accidents occur due to the avoidance (unintentionally or otherwise) or false interpretation of traffic signs in various environments.

A. Traffic Sign Detection and Recognition

If a part of an object is somehow covered (not properly visible) then successful detection and classification requires proper recognition task. Traffic sign recognition is an active research area in computer vision, color codes, shapes and pictograms used for traffic signs are different for countries. So, well defined color and shapes are main cues for detection of traffic signs. Also to attract the driver's attention easily, physical properties of traffic signs are so important. Due to these reasons the characteristics can be divided into 3 sub-classes:

1) Color based detection and recognition:

Color can be easily affected by lightning, as we discussed above different colors are used as background in different signs in different countries. Due to these differences, color spaces used nowadays are RGB (Red Green Blue), HSI (Hue Saturation and Intensity), HSV (Hue, Saturation, and Value), CIELAB, YUV [1]. The advantage of using color based approach is that it has fast computing speed. Whether HSL or RGB are better suited for traffic sign recognition has no hard proof. Both have their own pros and cons [2]. In HSV color of area proportional to brightness intuitive; color based on artist idea of tint, saturation and tone. Hue in HSV and CIELAB color space depends on distance and weather condition of traffic signs [3]. The most common approaches are using color and space based filtering to select the region of interest (ROIs).

2) Shape based detection and recognition:

Shape features plays key role for detection as shape is an important attribute for traffic sign detection. Reliability of shape detection depends on boundary detection or matching algorithm. As we discussed earlier shape based detection is important in a condition i.e. In Germany 'one-way street' represented by Long, Laying blue rectangular box whereas in US it is represented by white rectangular box. It is more robust to changes in illumination conditions as it depends on shape, which is based on edge or boundary of traffic signs. Imperfect shape of signs, occlusions of the object may cause the task somewhat challenging.

3) Both colour and shape based detection and recognition:

In order to improve the performance of detection process, a joint implementation of shape and color based algorithm is utilized.

B. Identification of Traffic Signs

One should have to set features properly in order to get good results. Identification of traffic signs consist of two steps; Pre-processing and classification. Well known classifiers named MLPs (Multi Layer Perceptions), SVMs [20] (Support Vector Machine), Radial basis functions [21], k nearest neighbors [22] are very sensitive to 2D input data. Normalized input data is required which makes it important to use feature extraction techniques. Feature extraction is a crucial part to distinguish multiclass probabilities as correctness of algorithm depends on feature vectors.

C. Motivation

We desire to reach the minima of the loss function in order to gain maximum accuracy. So, the intent is to identify which Gradient Descent optimizing algorithm out of RMSProp, Adam and Nadam converges to the local minima the earliest.

As far as Traffic sign detection and recognition is concerned, most of the earlier studies comprise of experimenting with different architectures. Whereas this paper is centered on a compact Convolutional Neural Network architecture that remains unchanged throughout the study and instead compares three optimizers used to train the architecture.

III. Overview of Optimizers

A convolutional neural network architecture consists of convolution layers designed for feature extraction, pooling layers that reduce the size of feature maps, flattening layers for converting 2-dimensional feature maps into 1-dimensional arrays followed by fully connected layers dedicated to *make sense* from the extracted feature maps, i.e. the decision making classifiers, activation functions used to maintain non-linearity and evaluating probabilities of the output classes. Dropout layers are used to randomly drop-off certain neurons in order to reduce the co-dependency among neurons in order to make the algorithm more robust by resolving the issue of over-fitting (a state where the algorithm works good for training set but fails to perform well on the test set). There exists no assured method to figure if a specific architecture will perfectly work for the given set of data until and unless practical experimentation is carried out. Hence we need to build our architectures intuitively which makes it very important to understand the backend mathematics.

Once we have designed our neural network architecture, the most important step is optimization. Sometimes a change as simple as switching to the correct optimizer can give improved accuracy instead of adding layers, increasing layer size and making the model bulky.

Understanding of Gradient Descent is essential to interpret the theory of optimizers. Gradient descent serves the purpose to minimize the loss function *J(θ).* In other words, Gradient Descent helps us navigate towards the local minima. It gives the gradient and then takes a descent i.e. moves in opposite direction of the gradient. Hence it was named as Gradient Descent. The batch gradient descent [6] or vanilla gradient descent computes the gradient of the loss function,

$$\theta = \theta - \eta \nabla_{\theta} J(\theta) \qquad (1)$$

Choosing a learning rate is quite a task which motivates to optimize the Gradient Descent.

Vanilla Gradient considered present gradients to evaluate the next step towards local minima. It faces problems navigating ravines that occur frequently around the local

minima. Momentum [7] considers a fraction of update vector of the past time step along with the current update vector [8]

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta) \quad (2)$$

$$\theta = \theta - v_t \quad (3)$$

Momentum increases dimensions of gradients in same directions and reduces updates in all other directions. This results into faster convergence.

RMSProp [9] i.e. Root Mean Square Propagation is a method for adaptive optimization. It divides the learning rate by exponential decaying average of squared gradients [6]

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t \quad (4)$$

Adaptive moment estimation [10] **(**Adam) is said to converge more quickly. By quick convergence, it means that it will require less number of epochs to reach the local minima of the los function. RMSProp considers exponentially decaying averages of past squared gradients and momentum considers exponentially squared averages of past gradients. Adam can be interpreted as a combination of RMSProp and Momentum. [6]

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \; ; \quad (5)$$

Exponentially decaying averages of past gradients

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \; ; \quad (6)$$

Exponentially decaying averages of past squared gradients

Bias correction $\hat{m}_t = \frac{m_t}{1-\beta_1^t}$ and $\hat{v}_t = \frac{v_t}{1-\beta_2^t}$ and then,

update parameters using Adam's update rule:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}} \hat{m}_t \quad (7)$$

with default values β_1 =0.9 and β_2=0.999 as proposed by authors.

Nesterov Accelerated Gradient [11] (NAG) calculates the gradient not w.r.t. the current position of parameters like momentum did, but w.r.t. approximate future position of the parameters:

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta - \gamma v_{t-1}) \quad (8)$$

Where $\theta = \theta - v_t$

Nesterov-accelerated Adaptive Moment Estimation (Nadam)[12] combines Adam and NAG by updating the momentum term, m_t. Recalling the momentum update rule: $\theta_{t+1} = \theta_t - (\gamma m_{t-1} + \eta g_t)$. It illustrates that momentum takes a step in direction of previous momentum vector and a step in the direction of the current gradient [6]

Modifying the gradient term as per NAG: [6]

$$g_t = \nabla_{\theta_t} J(\theta_t - \gamma m_{t-1}) \quad (9)$$

$$m_t = \gamma m_{t-1} + \eta g_t \quad (10)$$

$$\theta_{t+1} = \theta_t - m_t \quad (11)$$

As we have seen above, NAG is advanced than vanilla momentum so Nadam is expected to work out better than Adam, as per theory. Again, the results may seem otherwise, as the type of data plays a major factor. In this paper, we have compared the performance of RMSProp, Adam and Nadam on the BelgiumTS dataset for classification.

IV. RESULTS

A. Benchmarks/ Datasets

Plenty of benchmarks are available for traffic sign detection and recognition algorithms like United states traffic sign dataset [25], Italy traffic sign dataset, Croatia traffic sign dataset, Chinese traffic sign dataset, BTSD[4] (Belgium traffic sign dataset), GTSRB[23] (German traffic sign recognition benchmark), STSD[24] (Swedish traffic sign dataset) etc. There are multiple reasons to choose one dataset over another, includes various facts depending on requirements. The reduced BelgiumTS Dataset including 62 classes comprised of warning, priority, prohibitory, mandatory, parking and direction signs has been used to implement the proposed CNN architecture. If the provided dataset is not sufficient to train the models, then to increase the volume of images some images can be created synthetically by changing contrast and by adding noise using image editors.

B. NVIDIA Jetson Tx1/Tx2

A GPU can be termed as a co-processor that performs requested operations by CPU [5]. We are fulfilling our purpose of traffic sign classification using Convolutional Neural Networks that mainly demands to perform operations on matrices. For the feature extracting convolutional layers, the task is to convolve feature maps inputted by previous layers with the weight matrices, and to do it repeatedly for each layer. CPUs are efficient for complex mathematical tasks whereas GPUs are proficient in carrying out repetitive tasks.

The NVIDIA Jetson Tx1 Development board (pre-flashed with a linux environment) that contains 256 CUDA (Computer Unified Device Architecture) Maxwell cores at 998MHz, a quad-core 64-bit ARM Cortex–A57 at 1.73GHz CPU. Jetson Tx2 is twice as energy efficient for deep learning inference than Tx1. 256 CUDA cores Pascal GPU at 1300MHz. Quad-core ARM Cortex-A57 at 2GHz and dual-core NVIDIA Denver2 at 2 GHz. Both the kits have 5 MP Fixed Focus MIPI CSI Camera that captures 30 fps, 4 power buttons(power, reset, force recovery and user defined), port connectivity (USB 3.0, USB 2.0 Micro AB, HDMI, M.2 Key E, PCI-E x4, Gigabit Ethernet), connectivity to 802.11ac Wi-Fi and Bluetooth enabled devices.[19]

C. Experimentation Results

This section experimentally compares the performance of RMSProp, Adam and Nadam optimizers.

Layer (type)	Output Shape	Param #
conv2d_10 (Conv2D)	(None, 62, 62, 64)	1792
max_pooling2d_10 (MaxPooling	(None, 31, 31, 64)	0
conv2d_11 (Conv2D)	(None, 29, 29, 128)	73856
max_pooling2d_11 (MaxPooling	(None, 14, 14, 128)	0
conv2d_12 (Conv2D)	(None, 12, 12, 128)	147584
max_pooling2d_12 (MaxPooling	(None, 6, 6, 128)	0
flatten_4 (Flatten)	(None, 4608)	0
dense_13 (Dense)	(None, 256)	1179904
dropout_7 (Dropout)	(None, 256)	0
dense_14 (Dense)	(None, 256)	65792
dropout_8 (Dropout)	(None, 256)	0
dense_15 (Dense)	(None, 128)	32896
dense_16 (Dense)	(None, 62)	7998

Total params: 1,509,822
Trainable params: 1,509,822
Non-trainable params: 0

Table 1 : Summary of proposed CNN architecture

Table 1 depicts the proposed CNN architecture that was built for the classification of traffic signs on the BelgiumTS classification dataset. The aim here was to make the architecture as compact as possible. The architecture contains three convolutional layers and their respective pooling layers, flattening layer, three fully connected layers followed by dropout layer after first and second fully connected layer and an output layer with dimensions equal to the number of output classes. The performance of RMSProp, Adam and Nadam using this architecture on the BelgiumTS classification dataset is portrayed by figure 1 to figure 6.

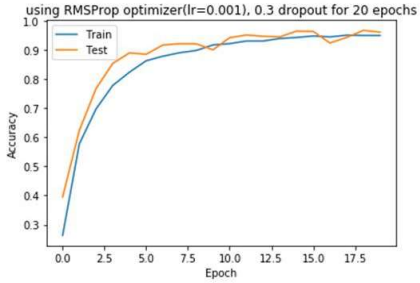


Figure 1 : Accuracy plot for RMSProp optimizer for dropout 0.3

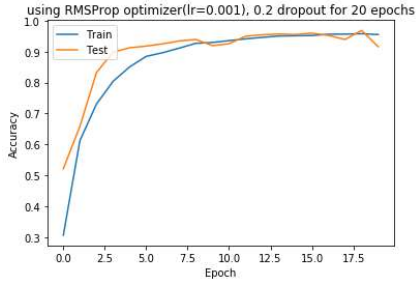


Figure 2 : Accuracy plot for RMSProp optimizer for dropout 0.2

Using RMSProp optimizer with dropout of 0.3 gave 94.91% training accuracy and 96.07 testing accuracy at 20 epochs. For majority of the training period, under-fitting can be observed from figure 2. Using the dropout of 0.2 gave comparatively better graphs. As per figure 2, under-fitting is observed initially, which was minimized after 10 epochs. With accuracies 95.98% and 91.63% for train and test respectively, over-fitting is observed towards the end.

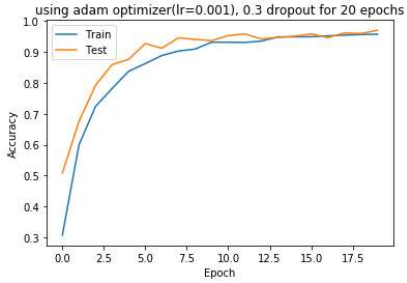


Figure 3 : Accuracy plot for Adam optimizer for dropout 0.3

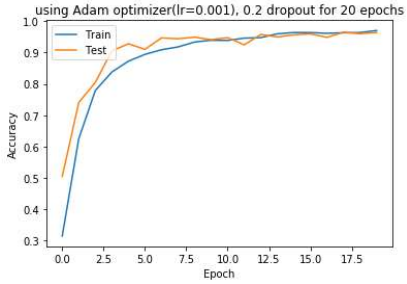


Figure 4 : Accuracy plot for Adam optimizer for dropout 0.2

Even after using Adam, under-fitting was observed for dropout of 0.3 accuracies 95.73% and 96.99% for train and test respectively were noted. Dropout of 0.2 gave better results compared to 0.3 as per the accuracy graphs. The training and testing accuracies were 97.05% and 96.43% respectively.

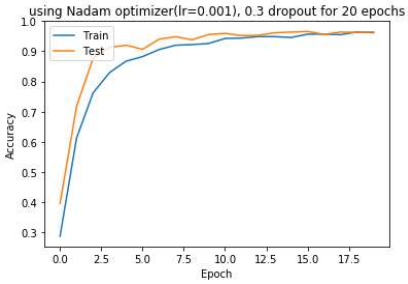


Figure 5 : Accuracy plot for Nadam optimizer for dropout 0.3

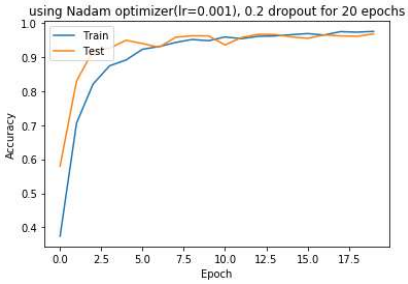


Figure 6 : Accuracy plot for Nadam optimizer for dropout 0.2

Using the Nadam optimizer for dropout of 0.3 boosted the training accuracy to 99.19%, but the testing accuracy was 97.62%. It can be seen that even though over-fitting is observed, Nadam improved the overall accuracy of the model. Using the dropout 0.2 with Nadam improved the training and testing accuracies to 97.51% and 96.78% respectively. The learning rate α was set to 0.001 for training all the models.

Clearly, the Nadam optimizer when used with the dropout of 0.2 was the best out of all the combinations stated above and predictions on test images were carried out using it.

Figure 7 shows our predictions on test images as implemented on the NVIDIA Jetson Tx1/Tx2. Figure 7(a) is a prediction of a visually clear traffic sign. Figure 7(b) and 7(c) are unevenly illuminated and improperly positioned signs that were predicted correctly by our model. Parts of the image in Figure 7(d) were purposely blurred and that too, was predicted correctly.

Identifying incorrect predictions is more important than the correct ones as it hints about further improvements. Figure 8 (left image) is a correctly predicted sign named 'Bumpy road'. Figure 8 (right) was intentionally tilt using editing tools and the architecture gave an incorrect prediction.

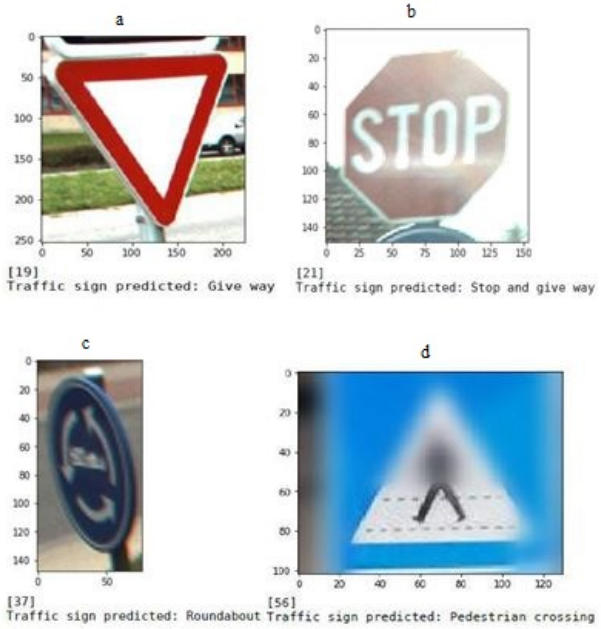


Figure 7

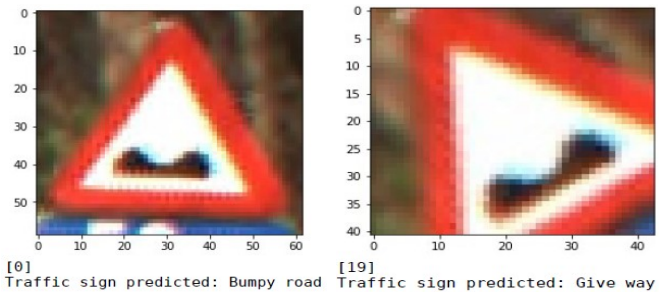


Figure 8

V. SUMMARY

DROPOUT	OPTIMIZER	EPOCHS	TRAIN	TEST
0.3	RMSProp	20	94.91	96.07
	Adam		95.31	95.75
	Nadam		96.19	96.43
0.2	RMSProp	20	95.98	91.63
	Adam		97.05	96.43
	Nadam		97.51	96.78

Table 2 : Accuracies for different optimizers on BelgiumTS dataset

As stated above, we have used dropout regularization of 0.3 and 0.2 and compiled the model using RMSProp, Adam and Nadam optimizers.

As to compare the optimizers, RMSProp is the *slowest* to converge. When used with momentum i.e. Adam, it converged *quicker*. When Adam used with nestrov momentum i.e. Nadam, it gave *quickest convergence* and hence best results. When dropout of 0.2 used with Nadam optimizer, it gave the best accuracy; Training accuracy = 97.51, Testing accuracy = 96.78

VI. CONCLUSION

As per theory, the Nestrov momentum makes Nadam intuitive and suggests better performance compared to other optimizers. We conclude with that Nadam works the best with Training accuracy = 97.51, Testing accuracy = 96.78 and converges quicker and better than RMSProp and Adam for classification on the BelgiumTS dataset.

Upon experimenting with many images, it can be concluded that the proposed architecture can predict blurred images well but it fails to predict tilted images. The scope of improvement is in expanding the training set and including misaligned images in order to make the algorithm more robust.

VII. ACKNOWLEDGEMENTS

The research was financially supported by the Student Start-up and Innovation Policy (SSIP), an initiative by the Government of Gujarat. After clearing the screenings conducted by GTU Innovation Council, we were eligible for the financial grant for Jetson Tx2, Display device (MarQ monitor) and other peripheral devices (keyboard, mouse, SSD drive, etc.)

VIII. REFERENCES

[1] W.G. Shadeed, D.I. Abu-Al-Nadi, and M.J. Mismar, "Road traffic sign detection in color images", Proceedings of the 2003 10th IEEE International Conference on Electronics, Circuits and Systems (ICECS). IEEE, 2003

[2] Kardkovács, Zsolt & Paróczi, Zsombor & Varga, E & Siegler, Adam & Lucz, P. (2011), "Real-time traffic sign recognition system".

[3] Emre Ulay, "Color and Shape Based Traffic Sign Detection". A thesis submitted to the Graduate School of Natural and Applied Sciences, 2008

[4] M. Mathias, R. Timofte, R. Benenson, L.J.V. Gool, "Traffic sign recognition - how far are we from the solution?" in: International Joint Conference on Neural Networks (IJCNN), 2013.

[5] Nathan Otterness[1] , Ming Yang[1] , Sarah Rust[1] , Eunbyung Park[1] , James H. Anderson[1] , F. Donelson Smith[1] , Alex Berg[1] , and Shige Wang[2], "An Evaluation of the NVIDIA TX1 for Supporting Real-time Computer-Vision Workloads", [1]Department of Computer Science, University of North Carolina at Chapel Hill [2]General Motors Research.

[6] Sebastian Ruder, "An overview of gradient descent optimization algorithms" arXiv:1609.04747

[7] Ning Qian, "On the momentum term in gradient descent learning algorithms", Neural networks : the official journal of the International Neural Network Society, 12(1):145–151, 1999.

[8] John Duchi, Elad Hazan, and Yoram Singer, "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization", Journal of Machine Learning Research, 12:2121–2159, 2011.

[9] H. Brendan Mcmahan and Matthew Streeter. "Delay-Tolerant Algorithms for Asynchronous Distributed Online Learning", Advances in Neural Information Processing Systems (Proceedings of NIPS), pages 1–9, 2014.

[10] Diederik P. Kingma and Jimmy Lei Ba., "Adam: a Method for Stochastic Optimization". International Conference on Learning Representations, pages 1–13, 2015.

[11] Yurii Nesterov, "A method for unconstrained convex minimization problem with the rate of convergence o(1/k2)", Doklady ANSSSR (translated as Soviet.Math.Docl.), 269:543–547.

[12] Timothy Dozat, "Incorporating Nesterov Momentum into Adam", ICLR Workshop, (1):2013–2016, 2016.

[13] François Chollet, "Keras - https://keras.io/", Neural Networks platform written in python, stable release: 2.2.4 / 3 October 2018.

[14] John D. Hunter, "Matplotlib: A 2D Graphics Environment", Computing in Science & Engineering, **9**, 90-95 (2007), DOI:10.1109/MCSE.2007.55 (publisher link)

[15] Travis E, Oliphant, "A guide to NumPy", USA: Trelgol Publishing, (2006).

[16] Stéfan van der Walt, S. Chris Colbert and Gaël Varoquaux, "The NumPy Array: A Structure for Efficient Numerical Computation", Computing in Science & Engineering, 13, 22-30 (2011), DOI:10.1109/MCSE.2011.37

[17] Python Software Foundation, "Python Language Reference, version 3.6.8" Available at http://www.python.org

[18] A. Collette, "HDF5 for Python", 2008 (http://h5py.alfven.org)

[19] Dustin Franklin, "NVIDIA Jetson TX2 Delivers Twice The Intelligence To the Edge", March 7, 2017 NVIDIA Developer Blog.

[20] Muthukumaresan.T, Kirubakaran.B, Kumaresan.D, Akhil Satheesan, Jaya Prakash.A., "Recognition of Traffic Sign using Support Vector Machine and Fuzzy Cluster", IJSTE - International Journal of Science Technology & Engineering, Volume 2, Issue 10, April 2016.

[21] D.M. Gavrila, "Traffic Sign Recognition Revisited", Proc. of the 21st DAGM Symposium für Mustererkennung, pp. 86-93, Springer Verlag, 1999.

[22] Fatin Zaklouta, Bogdan Stanciulescu and Omar Hamdoun, "Traffic Sign Classification using K-d trees and Random Forests".

[23] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel, "The German Traffic Sign Recognition Benchmark: A multi-class classification competition", submitted to International Joint Conference on Neural Networks, 2011.

[24]Fredrik Larsson and Michael Felsberg, **"**Using Fourier Descriptors and Spatial Models for Traffic Sign Recognition**",** In Proceedings of the 17th Scandinavian Conference on Image Analysis, SCIA 2011, LNCS 6688, pp. 238-249. doi:10.1007/978-3-642-21227-7_23

[25] A. Møgelmose, M. M. Trivedi, and T. B. Moeslund, "Vision based traffic sign detection and analysis for intelligent driver assistance systems: Perspectives and survey", IEEE Trans. Intell. Transp. Syst., vol. 13, no. 4, pp. 1484–1497, Dec. 2012.

[26] Felipe Sisido[1], Jonas Goya[2], Guilherme S. Bastos[3] and Audeliano W. Li[4] , "Traffic Signs Recognition System with Convolutional Neural Networks", 2018 Latin American Robotic Symposium, 2018 Brazilian Symposium on Robotics (SBR) and 2018 Workshop on Robotics in Education (WRE)

[27] Liu Wei, Lu Runge and Liu Xiaolei, "Traffic Sign Detection and Recognition via Transfer Learning", Chinese Control and decision Conference (CCDC) 9-11 June 2018, Shenyang, China, doi: 10.1109/CCDC.2018.8408160

[28] Yan Han and Erdal Oruklu, "Traffic Sign Recognition Based on the NVIDIA Jetson TX1 Embedded System using Convolutional Neural Networks", IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS), 6-9 Aug. 2017, Boston, MA, USA.

[29] Chen, Z., Huang, X., Ni, Z., & He, H. (2014) "A GPU-based real-time traffic sign detection and recognition system" 2014 IEEE Symposium on Computational Intelligence in

Vehicles and Transportation Systems *(CIVTS)*.doi:10.1109/civts.2014.7009470

[30] Schaul, T.; Zhang, S.; and LeCun, Y. 2013, "No More Pesky Learning Rates", International Conference on Machine Learning (ICML).

[31] Tieleman, T., sand Hinton, G. 2012. Lecture 6.5 "RMSprop: Divide the Gradient by The Running Average of its recent magnitude", In COURSERA: Neural Networks for Machine Learning.

Traffic Sign Recognition Using Color and Spatial Transformer Network on GPU Embedded Development Board

Bhaumik Vaidya[1](✉) and Chirag Paunwala[2]

[1] Gujarat Technological University, Ahmedabad, India
vaidya.bhaumik@gmail.com
[2] Electronics and Communication Department, Sarvajanik College of Engineering and Technology, Surat, India

Abstract. Traffic sign recognition is an integral part of any driver assistance system as it helps the driver in taking driving decisions by notifying about the traffic signs coming ahead. In this paper, a novel Architecture is proposed based on Convolutional Neural Network (CNN) for traffic sign classification. It incorporates Color Transformer Network and Spatial Transformer Network (STN) within CNN to make the system invariant to color and affine transformation invariant. The aim of this paper is to compare the performance of this novel architecture with the existing architectures in constrained road scenarios. The performance of the algorithm is compared for two well-known traffic sign classification dataset: German Traffic Sign dataset and Belgium Traffic Sign dataset. The paper also covers the deployment of the trained CNN model to Jetson Nano GPU embedded development platform. The performance of the model is also verified on Jetson Nano development Board.

Keywords: Traffic sign classification · Convolutional Neural Networks · Spatial Transformer Network · Color transformer network · Jetson Nano development board

1 Introduction

Traffic sign detection and classification is the essential part of any driver assistant or autonomous driving system. Traffic signs are used to maintain traffic discipline on Roads and avoid road accidents. The use of traffic signs is prominent of highways during night time and on hilly regions to identify curves on the road. The traffic signs can aide driving in adverse visual conditions like foggy or rainy environments. The correct interpretation of traffic signs can help reducing road accidents drastically.

Traffic signs are designed using a specific shape and color to indicate valuable information like traffic rules, speed limits, turns on the road, road conditions, construction going on etc. This shape and color information can be used to detect and classify Traffic signs. Many systems have been built to detect traffic signs using color and shape

© Springer Nature Singapore Pte Ltd. 2020
N. Nain et al. (Eds.): CVIP 2019, CCIS 1147, pp. 82–93, 2020.
https://doi.org/10.1007/978-981-15-4015-8_8

information. This systems have their limitation. Otherwise also detecting and classifying traffic signs is a complex computer vision problem. The problems like color fading, partial occlusion, illumination variation, motion blur etc. are major hurdles in detecting and classifying traffic signs.

The availability of large datasets and machines with high computational power has revolutionized the research in the domain of Traffic Sign Classification. The traditional method of using shape and color for identification is being replaced by various Architectures of CNN. Though still many CNN architectures need Image Preprocessing to make it Color and Transform invariant. This paper introduces an end to end CNN architecture which is invariant to color and affine transformation and helps in classifying traffic signs accurately. We need some embedded platform to deploy this model on hardware, if we want to include this system in vehicles. Jetson Nano development board is used in this paper to deploy the trained CNN model on hardware.

The rest of the paper is organized as follows. Section 2 summarizes related work on traffic sign classification and detection. The details of the proposed system and theoretical background of algorithms used is explained in Sect. 3. Section 4 shows the experiment results of the proposed system on two datasets. The last sections contain the conclusion and references used for the work.

2 Related Work

Traffic sign Classification is the process of identifying the class of the sign from an Image while Traffic Sign detection is to localize the sign by adding a bounding box around the detected signs. There are two approaches for Traffic Sign Detection and Classification in the literature. The first approach uses traditional image processing techniques while the other approach uses different deep neural network architectures.

Traffic signs have definite color and shapes so they were used traditionally to detect and classify traffic signs. Shadeed *et al.* [1] in their paper proposed a traffic sign classification system using color segmentation in YUV color space. Li *et al.* [2] in their paper proposed a technique based on color segmentation and shape matching. De La Escalera *et al.* [3] in their paper prosed a system based on color segmentation and color thresholding. Most of the color based method uses HIS, HSV and YCbCr color spaces for segmentation as RGB color space is very sensitive to change in illumination. The advantage of using color based techniques is that they are fast but they are very sensitive to color information. These systems can fail after the color fades.

Traffic signs are mostly circular or triangular. They have definite shape so this information can be used to classify traffic signs from an Image. Barnes *et al.* [4] used Hough transform to detect traffic signs which have shape of polygons. Hough transform is a very simple and fast algorithm to identify regular shapes from an image. Berkaya *et al.* [5] used EDCircle algorithm to identify circular traffic signs. Keller *et al.* [6] used Histogram of oriented gradients (HOG) features to identify traffic signs from an Image. The shape based methods are not as fast as color based methods but still they can be used in real time. They fail when traffic sign is having deformation or occluded by some other objects.

The color and shape based methods used to detect traffic signs uses some classification algorithm for classifying different types of traffic signs. Maldonado-Bascon *et al.*

[7] used Support Vector Machine (SVM) for classification. Cireşan et al. [8] used Random Forest Algorithm for classification and Ruta *et al.* [9] used K-Nearest Neighbor algorithm for Classification.

The second approach for detecting and classifying is to use deep neural networks. Ciresan *et al.* [8] proposed a multi column deep neural network for classifying traffic signs from a color image. Sermanet *et al.* [10] proposed a multi scale CNN architecture for Traffic Sign Classification. Aghdam *et al.* [11] proposed an end to end CNN architecture for Traffic sign detection and Classification. The use of CNN architectures have significantly increased the accuracy of traffic sign detection and classification. Most CNN based architectures needs a preprocessing steps like color space conversion, histogram equalization, data augmentation etc. to make it color, transformation and illumination invariant.

The motivation of this paper was to eliminate this preprocessing step and make an end to end CNN architecture which is invariant to affine transformation, color and illumination. The CNN architectures takes a raw image as input and gives corresponding class as an output without any extra steps. The details of a proposed system is explained in the next section.

3 Proposed System

The simplified flow chart for the proposed system is shown in Fig. 1 below.

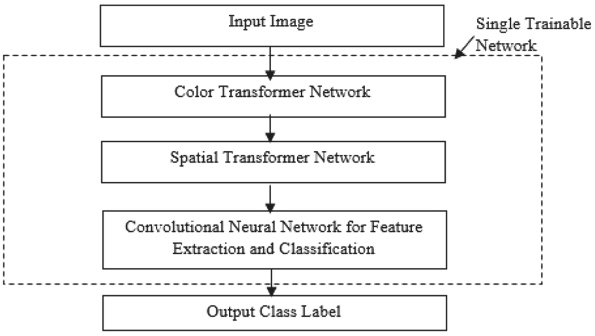


Fig. 1. Flow chart for proposed system

As can be seen from the figure, the input image is given to a single neural network which internally contained Color transformer network [12] for color space conversion, Spatial Transformer Network [13] for learning spatial transformation and CNN for feature extraction and classification. The beauty of the system that entire network is trainable in a one go. The color transformer network and STN will learn their parameters in the same way like regular CNN using gradient descent and back propagation. The entire system is trained using German Traffic Sign Classification dataset [14] or Belgium Traffic Sign dataset [15]. The trained model is deployed on Jetson Nano development board [16] for inference in real time on embedded platform. The individual constituents of the system are explained one by one below:

3.1 Color Transformer Network

Color images are stored in RGB format inside computer. This format is very sensitive to change in illumination so to make a robust system, researchers use HSV or YCbCr format to train CNN model. This requires deciding on a color channel format which is optimal for the problem and explicit color space conversion step.

It would be great if this color conversion parameters can also be learned by the network during training according to the dataset. Mishkin et al. proposed a technique for leaned color space conversion using 1×1 convolution [12]. This paper adopts the best technique from that paper for a color transformer network which passes RGB values through ten 1×1 convolutions followed by three 1×1 convolutions. The ReLU activation function is used. This network will learn color transformation parameters according to the dataset while training.

3.2 Spatial Transformer Network

Jaderberg et al. proposed STN [13] which is a learnable module that allows spatial transformation of the data within the network. This module can be integrated with existing CNN architecture to give them ability to learn spatial transformation parameters conditional on the training dataset without the need of any additional training. The architecture of STN is shown in Fig. 2 below:

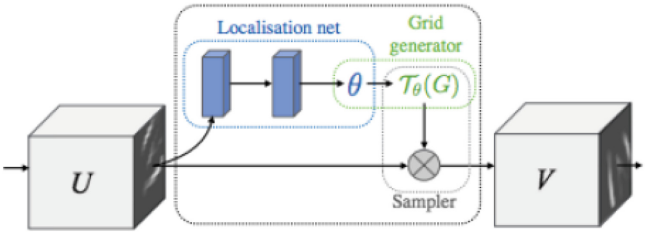


Fig. 2. Spatial Transformer Network [13]

The input feature map U is passed through a localization network which itself is a CNN to find transformation parameters θ through regression. The sampling grid $T_\theta(G)$ is generated by transforming regular spatial grid G over V. This sampling grid is applied to feature map using image resampling to produce output feature map V.

The input to the localization network is $32 \times 32 \times 3$ image. The architecture of Localization network used in the paper is shown in table below (Table 1):

Table 1. Localization network architecture

Layer name	Filter size	Number of filters or neurons
Convolution	3×3	32
Max-pooling with stride 2×2	–	–
Convolutional	3×3	64
Max-pooling with stride 2×2	–	–
Convolutional	3×3	64
Max-pooling with stride 2×2	–	–
Dense	–	128
Dense	–	64
Dense	–	6

The localization network generates six parameters for affine transformation matrix shown in Eq. 1 as output.

$$A_\theta = \begin{bmatrix} \theta_{11} & \theta_{12} & \theta_{13} \\ \theta_{21} & \theta_{22} & \theta_{23} \end{bmatrix} \tag{1}$$

Then grid generator will generate a grid of coordinates in the input image corresponding to each pixel from the output image using the Eq. 2 below:

$$\begin{pmatrix} x_i^s \\ y_i^s \end{pmatrix} = T_\theta(G_i) = A_\theta \begin{pmatrix} x_i^t \\ y_i^t \\ 1 \end{pmatrix} \tag{2}$$

Where (x_i^s, y_i^s) are co-ordinates of source image and (x_i^t, y_i^t), are co-ordinates of affine transformed image. A_θ is the transformation matrix generated in Eq. 1. The bilinear interpolation is used to generate the final pixel value from the grid values.

3.3 Convolutional Neural Network

CNN is used for feature extraction and classification in the system. It consists of series of convolutional and pooling layers to learn hierarchical features from the image. The ReLU (Rectified Linear Unit) activation function is used throughout the architecture as it is easy to compute and makes learning faster by avoiding vanishing gradient problem which is prominent in sigmoid or tanh activation functions. The spatially transformed feature map is given as input to the CNN. It has the same size of input image. The layer wise CNN architecture used in the paper is shown in Table 2.

Table 2. CNN layer-wise architecture

Layer name	Filter size	Number of filters or neurons
Convolution	3 × 3	16
Convolutional	3 × 3	32
Max-pooling with stride 2 × 2	–	–
Convolutional	3 × 3	64
Convolutional	3 × 3	96
Max-pooling with stride 2 × 2	–	–
Convolutional	3 × 3	128
Convolutional	3 × 3	64
Max-pooling with stride 2 × 2	–	–
Dense	–	128
Dense	–	43

Batch Normalization is used after every convolutional layer output. It helps in faster training and increases accuracy by normalizing output of each layer [17]. The Input image size is 32 × 32 × 3 for German Traffic sign dataset and output neurons are 43 as it has 43 classes. The Input image size is 64 × 64 × 3 for Belgium Traffic sign dataset and output neurons are 62 as it has 62 classes. The training setup and results are explained in the next section.

3.4 GPU Embedded Development Board

A GPU are much more efficient in computing parallel operations like convolutions then simple CPUs. When the proposed system needs to be deployed in real life scenarios it has to be deployed in embedded board which contains GPU for faster inference. There are several GPU Embedded board available in market like NVIDIA Jetson TX1, TX2, Nano and Google Coral. Jetson TX1 and TX2 are very costly for the given application and Google Coral only supports TensorFlow Lite at this point so Jetson Nano Development board is used to deploy the system on hardware. Jetson Nano also consumes less power (Less than 5 W) compared to other boards [16].

NVIDIA Jetson nano board which is a small powerful computer that allows faster computation of traffic sign classification is used for deployment in this paper. It has a 128 core Maxwell GPU along with Quad-core ARM A57 CPU running at 1.43 GHz. It has 4 GB of RAM and runs from a MicroSD card. It has 4 USB 3.0 port for connecting external peripherals and a HDMI connector for connecting displays. It can also be interfaced with Camera via a CSI connector or a USB port. It delivers performance of 472 Giga floating point operations per second (GFlops) [16].

The CNN model for traffic sign classification using TensorFlow and Keras is deployed on Jetson nano for inference. The board comprises of a Jetpack installed over Ubuntu Operating System which uses TensorRT for faster inference.

4 Implementation and Results

The proposed system is implemented using Python and OpenCV using Anaconda Python distribution. Tensorflow and keras library is used to implement CNN architecture. The system is tested both on CPU and GPU hardware platform. The GeForce 940 GPU is used for training which has a dedicated RAM of 4 GB. The CPU has i5 processor with 8 GB of RAM and 2.2 GHz clock speed. The system is also deployed on Jetson nano development board. The dataset used are BTSD (Belgium traffic sign dataset) [15] and GTSRB (German traffic sign recognition benchmark) [14]. They both contains large amount of image data collected in real life scenarios and ideal for training CNN.

Training setup for GTSRB is shown in Table 3.

Table 3. Training setup for GTSRB

Parameter	Value
Input image size	$32 \times 32 \times 3$
Number of classes	43
Batch size	256
Number of epochs	50
Learning rate	0.0005
No. of training images	34799
No. of validation images	4410
No. of testing images	12630
Training accuracy	99.94%
Validation accuracy	99.16%
Test accuracy	98.40%
Classification time	3.9 ms/image

The increase in training and validation accuracy after every epoch is shown in Fig. 3 below. Adaptive moment estimation (Adam) [18] is preferred as an optimization function while training because of its faster convergence and lower fluctuations compared to RMSProp and Nadam Optimizer as shown in Fig. 3. The equations for updating weights using ADAM optimizer while training are given below: Exponentially decaying averages of past gradients is calculated by:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{3}$$

Exponentially decaying averages of past squared gradients is calculated by:

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{4}$$

Bias correction $\hat{m}_t = \frac{m_t}{1-\beta_1^t}$ and $\hat{v}_t = \frac{v_t}{1-\beta_2^t}$ and then, update parameters using Adam's update rule:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}\hat{m}_t \tag{5}$$

$\beta_1 = 0.9$ and $\beta_2 = 0.999$ is used in this paper as proposed by authors.

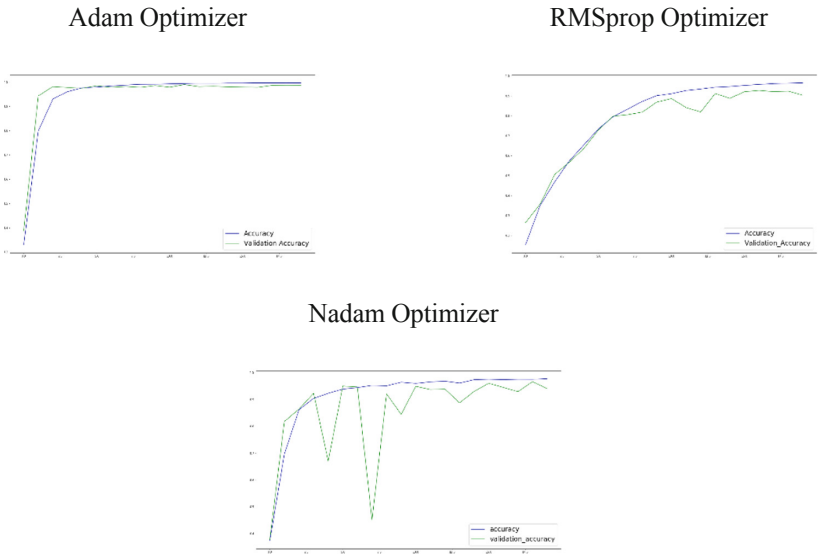


Fig. 3. Training and validation accuracy vs Number of epochs for different optimizers

Training setup for Belgium Traffic sign dataset is shown in Table 4.

Table 4. Training setup for BTSD

Parameter	Value
Input image size	$64 \times 64 \times 3$
Number of classes	62
Batch size	256
Number of epochs	50
Learning rate	0.0005
No. of training images	4575
No. of validation images	2520
Training accuracy	98.26%
Validation accuracy	96.31%
Classification time	3.9 ms/image

The increase in training and validation accuracy after every epoch using Adam optimizer is shown in Fig. 4 below.

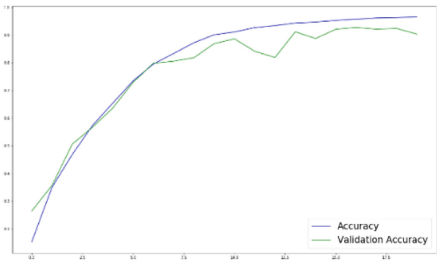


Fig. 4. Training and validation accuracy vs Number of epochs

Initially, it was found that training accuracy is very high compared to validation accuracy as shown in Fig. 4. So model was over fitting on training data. To overcome that, dropout of 0.3 was used in dense layers. It randomly removes neurons with a probability of 30% while training so we are training a different architecture every time. It removes the over dependence on any neuron and by that avoids over fitting.

The few correctly identified traffic signs from the test set of germen dataset using the proposed method is shown in Fig. 5.

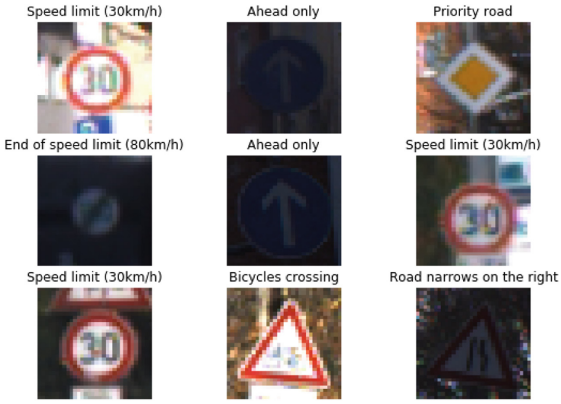


Fig. 5. Correctly classified traffic signs

The few correctly identified traffic signs from the random images downloaded from internet using the proposed method is shown in Fig. 6.

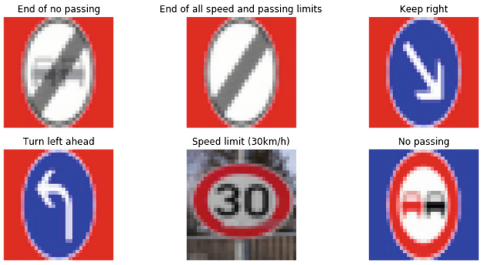


Fig. 6. Correctly classified images from Internet

It is important to look at images on which the algorithm fails to identify traffic signs.

Fig. 7. Incorrectly classified images

As can be seen from the Fig. 7, the traffic signs are very difficult to classify even for a human as they are hardly visible due to illumination and blurriness. The performance of the proposed algorithm on German Traffic Sign dataset is compared with other algorithms in Table 5.

Table 5. Performance of the proposed system

Algorithm used	Classification accuracy (%)
Human (Average) [14]	98.84
CNN with Spatial Transformer (Proposed Method)	98.40
Multi-scale CNN [10]	98.31
CNN without STN	97.04
Random Forest [19]	96.14
LDA on HOG2 [20]	95.68
LDA on HOG1 [20]	93.18
LDA on HOG3 [20]	92.34

As can be seen, the performance of the proposed algorithm is very close to human performance. By tweaking few parameters and training for a longer time may help it in reaching human level performance as training and validation error is already surpassing human performance.

The system is also deployed on Jetson Nano development board. It takes around 11 ms to classify traffic signs in Jetson Nano which is faster than 40 ms taken by CPU only computation unit. So this system can be easily used for deploying in vehicles for real time applications.

5 Conclusion

Traffic sign classification is very essential for building driver assistant or autonomous driving system. In this paper, a novel Architecture is proposed based on Convolutional Neural Network (CNN), Color Transformer Network and Spatial Transformer Network (STN) is proposed. It removes the need for preprocessing to make CNN invariant to color and affine transformation. The architecture is end to end which can be trained in one go and learn the parameters for color transformation and spatial transformation. The performance of proposed architecture is compared with the existing architectures and it can be seen that it can reach a human level performance. The performance of the algorithm is compared for two well-known traffic sign classification dataset. The performance of the model is also verified on Jetson Nano development Board.

References

1. Shadeed, W.G., Abu-Al-Nadi, D.I., Mismar, M.J.: Road traffic sign detection in color images. In: Proceedings of the 2003 10th IEEE International Conference on Electronics, Circuits and Systems (ICECS). IEEE (2003)
2. Li, H., Sun, F., Liu, L., Wang, L.: A novel traffic sign detection method via color segmentation and robust shape matching. J. Neurocomput. **169**, 77–88 (2015). ISSN 0925-2312
3. De La Escalera, A., Moreno, L.E., Salichs, M.A., Armingol, J.M.: Road traffic sign detection and classification. IEEE Trans. Ind. Electron. **44**(6), 848–859 (1997)
4. Barnes, N., Loy, G., Shaw, D.: Regular polygon detection. In: Proceedings of the Tenth IEEE International Conference on Computer Vision, pp. 778–785 (2005)
5. Berkaya, S.K., Gunduz, H., Ozsen, O., Akinlar, C., Gunal, S.: On circular traffic sign detection and recognition. J. Expert Syst. Appl. **48**, 67–75 (2016). ISSN 0957-4174
6. Keller, C.G., Sprunk, C., Bahlmann, C., Giebel, J., Baratoff, G.: Realtime recognition of U.S. speed signs. In: Proceedings of the Intelligent Vehicles Symposium, pp. 518–523 (2008)
7. Maldonado-Bascón, S., Lafuente-Arroyo, S., Gil-Jimenez, P., Gómez-Moreno, H., López-Ferreras, F.: Road-sign detection and recognition based on support vector machines. IEEE Trans. Intell. Transp. Syst. **8**(2), 264–278 (2007)
8. Cireşan, D., Meier, U., Masci, J., Schmidhuber, J.: Multi-column deep neural network for traffic sign classification. Neural Netw. **32**, 333–338 (2012)
9. Ruta, A., Li, Y., Liu, X.: Real-time traffic sign recognition from video by class-specific discriminative features. Pattern Recognit. **43**(1), 416–430 (2010)
10. Sermanet, P., LeCun, Y.: Traffic sign recognition with multi-scale convolutional networks. In: The 2011 International Joint Conference on Neural Networks (IJCNN), SanJose, California, USA, pp. 2809–2813. IEEE (2011)
11. Aghdam, H.H., Heravi, E.J., Puig, D.: A practical approach for detection and classification of traffic signs using Convolutional Neural Networks. J. Robot. Auton. Syst. **84**, 97–112 (2016). ISSN 0921-8890

12. Mishkin, D., Sergievskiy, N., Matas, J.: Systematic evaluation of CNN advances on the ImageNet. arXiv preprint arXiv:1606.02228 (2016)
13. Jaderberg, M., Simonyan, K., Zisserman, A.: Spatial transformer networks. In: Advances in Neural Information Processing Systems, pp. 2017–2025 (2015)
14. German Traffic Sign Recognition Dataset. http://benchmark.ini.rub.de/?section=gtsrb&subsection=dataset
15. Belgium Traffic Sign Dataset. https://btsd.ethz.ch/shareddata/
16. Jetson Nano Development Board Help Document. https://developer.nvidia.com/embedded/learn/get-started-jetson-nano-devkit
17. Ioffe, S., Szegedy, C.: Batch normalization: accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167 (2015)
18. Kingma, D. P., Ba, J.: Adam: a method for stochastic optimization. In: International Conference on Learning Representations, pp. 1–13 (2015)
19. Zaklouta, F., Stanciulescu, B., Hamdoun, O.: Traffic sign classification using K-d trees and Random Forests. In: International Joint Conference on Neural Networks (IJCNN), pp. 2151–2155 (2011)
20. Stallkamp, J., Schlipsing, M., Salmen, J., Igel, C.: Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. Neural Netw. **32**, 323–332 (2012)

Chapter 4
Deep Learning Architectures for Object Detection and Classification

Bhaumik Vaidya and Chirag Paunwala

Abstract Object detection and classification have observed large amount of transformation and research after the advances in machine learning algorithms. The advancement in the computing power and data availability is complimenting this transformation in object detection. In recent times, research in the field of object detection is dominated by special type of neural network called Convolutional Neural Network (CNN). The object detection system has to localize objects in an image and accurately classify it. CNN is well suited for this task as it can accurately find features like edges, corners and even more advanced features needed to detect object. This chapter provides detailed overview on how CNN works and how it is useful in object detection and classification task. After that popular deep networks based on CNN like ResNet, VGG16, VGG19, GoogleNet and MobileNet are explained in detail. These networks worked well for object classification task but needed sliding window technique for localizing object in an image. It worked slowly as it needed to process many windows for a single image. This led to more advanced algorithms for object detection based on CNN like Convolutional Neural Network with Region proposals (R-CNN), fast R-CNN, faster R-CNN, Single shot multi-box detector (SSD) and You Only Look Once (YOLO). This chapter provides a detail explanation of how these algorithms work and comparison between them. Most of the deep learning algorithms require large amount of data and dedicated hardware like GPUs to train. To overcome this, the concept of transfer learning is discovered. In that pretrained models of popular CNN architecture are used to solve new problems. So in the last part of the chapter this concept of transfer learning and when it is useful is explained.

B. Vaidya (✉)
Research Scholar, Gujarat Technological University, Ahmedabad, India
e-mail: bhaumik.vaidya@scet.ac.in; vaidya.bhaumik@gmail.com

C. Paunwala
SCET, Surat, India

© Springer Nature Switzerland AG 2019

M. K. Mishra et al. (eds.), *Smart Techniques for a Smarter Planet*,
Studies in Fuzziness and Soft Computing 374,
https://doi.org/10.1007/978-3-030-03131-2_4

Keywords Deep learning · Convolutional neural network (CNN) · CNN with region proposals (R-CNN) · You only look once (YOLO) · Single shot multi-box detector (SSD) · Transfer learning

4.1 Introduction

The amount of image and video data available in the world is increasing day by day. It is estimated that billions of images taken from personal mobile devices are uploaded on social networking sites daily [1]. Other sources of image and video data are commercial imaging devices which are widely used for automation in almost all fields of life.

Doctors are using images like MRI, X-Rays etc. to diagnose various diseases. Engineers are using images to find out faulty products after manufacturing or development of autonomous vehicle where camera mounted on it will continuously monitor surrounding and take decisions based on that. Scientist uses images for space exploration or for research at molecule levels. In smart cities, widespread deployment of traffic monitoring cameras continuously captures images or videos for traffic surveillance or crime monitoring. CCTV cameras installed at various important locations also capture large amount of video data. It can be seen from above examples that enormous amount of data is produced every day and there is a need to extract information from this data. Computer vision is a field that extracts information from the images and helps to convert pixel level information to information that can be understood by human. The field of computer vision is different than simple image processing. Image processing deals with manipulating visual information on the pixel level while computer vision is concerned with extracting useful information from an image. Computer vision has a widespread application in object detection, image classification, machine vision, augmented reality and automation [2].

One of the basic problems faced in computer vision is object detection. Object detection is the problem in which objects in an image are localized and identified automatically [3]. Identifying a class of a particular object is called object classification. It is a widely explored and researched topic in computer vision. It is very trivial for human to identify object in an image but for machines it can be very difficult. Various challenges affect the performance of a machine in object detection task. It involves creating an object detection system that is invariant to shape changes, change in lighting or illumination, translation and rotation. Sometimes camera jitter or noisy images also affect the performance of the system. When color of background is similar to object to be detected or when background is dynamic makes object detection task even more difficult. Object detection creates a problem like chicken and egg, where to identify the location from object, shape of the object should be known and to know the shape, location should be known [4].

Research interest in the field of object detection and classification is increasing day by day due to availability of large- scale image database and computing resources that are capable to process these data. In last decade, research in object detection was

driven by many machine learning algorithms like Shape-Invariant Feature Transform (SIFT) [5], Speeded up Robust Features (SURF) [6] and Histogram of Oriented Gradient (HOG) [7]. They involved feature extraction and feature description task which finds unique features from the image that can accurately describe the object. These features were given as an input to the classifiers like Naïve Bayes or Support Vector Machines (SVM) [7] to locate and classify object from the image. It uses sliding window approach where window slides over an image at different location and different scales. For each of these sub-windows features are extracted and given to classification algorithms for object detection and classification [7].

The end of the last decade and start of this decade saw rise of neural network because of availability of large amount of image dataset and powerful computers due to Graphics Processing Unit (GPU). Modern neural networks are deep neural networks which have multiple hidden layers between input and output layers to learn important features from data [8]. These sub-domain popularly known as deep learning has transformed the research in the field of object detection and classification. The invention of CNN [9] has been instrumental in increasing accuracy of object detection and classification task.

Deep learning has removed the need to find hand-coded features from the image as was the case in previous machine learning algorithms. In deep learning, lower layer learn to recognize simple features like edges or colors from the image which are fed to advance layers to find high level features specific to image. Many algorithms are developed for object detection based on CNN. Some algorithms like R-CNN [10], fast R-CNN [3] and faster R-CNN [11] need a separate region proposal network other than CNN to detect and classify object from image whereas other like YOLO [12] and SSD [13] detects and classifies object in single pass through CNN.

This chapter starts with explaining difference between traditional machine learning and deep learning algorithms and what made deep learning very popular. Then theoretical explanation of CNN is given. The popular deep learning architecture used for image classification like Lenet-5 [14], AlexNet [9], GoogleNet [15], ResNet [16], VGGNet [17], ZFNet [18] and MobileNet [19] are described in the next section. Then object detection algorithms based on CNN are described in detail. The last section describes transfer learning in detail.

4.2 Need of Deep Learning

To illustrate how deep learning works, animal classification system which classified cat or dog from an image is taken as an example. Conventional machine learning algorithm requires finding out or hand coding various features to identify both the objects and these features need to be trained and classified.

Now, deep learning takes this to higher level of abstraction by automatically finding out the features which are relevant for classification and by that removing the need of giving features manually, as is the case in machine learning.

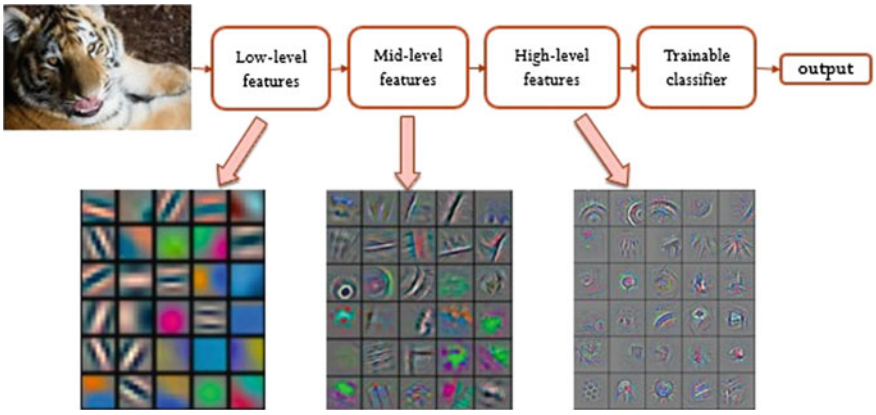


Fig. 4.1 Block diagram of object classification using deep learning approach

Deep learning works as follows:

- The algorithm first identifies lower level features such as edges that are important in distinguishing between a cat and a dog.
- It then hierarchically builds on these lower level features to find what combination of these features is relevant. For example, image contains whiskers or not, or image contains ears or not etc.
- After consecutive hierarchical building of complex features, algorithm then decides which of these features are most relevant in finding out the answer.

The whole process of object classification described above is depicted in Fig. 4.1

To learn relevant features for a given application deep learning approach needs a large amount of data. Also to train such a system on large image data it needs high-end computers with GPUs whereas machine learning tasks normally can be completed on normal CPUs [20]. The module problem-solving approach of first extracting features and then classifying it in machine learning is replaced with end-to-end approach in deep learning. Though deep learning systems take large amount of time in training, its test time is comparable or sometimes better than some machine learning approach [8]. As it will be seen later on, it removes the need of processing large amount of windows to detect an object from a single image as in the case of machine learning algorithms with sliding window approaches. The main challenge that deep learning system designer face is its very hard to interpret which features are learnt by the system for solving the given problem. The summary of difference between deep learning algorithms and traditional machine learning algorithms is given in Table 4.1.

The most popular deep learning approach for solving computer vision problems using deep learning approach is convolutional neural network (CNN). The next section explains the concept of CNN and how it has revolutionized the field of object detection and classification.

Table 4.1 Difference between deep learning and machine learning approaches

Parameter	Deep learning	Machine learning
Data dependencies	Large data	Small data
Hardware dependencies	High end machine	Normal machines
Feature engineering	Automated	Hand coded
Problem solving approach	End to end	Modular
Execution time training	Long	Short
Testing time	Short	Long
Interpretability	Low	High

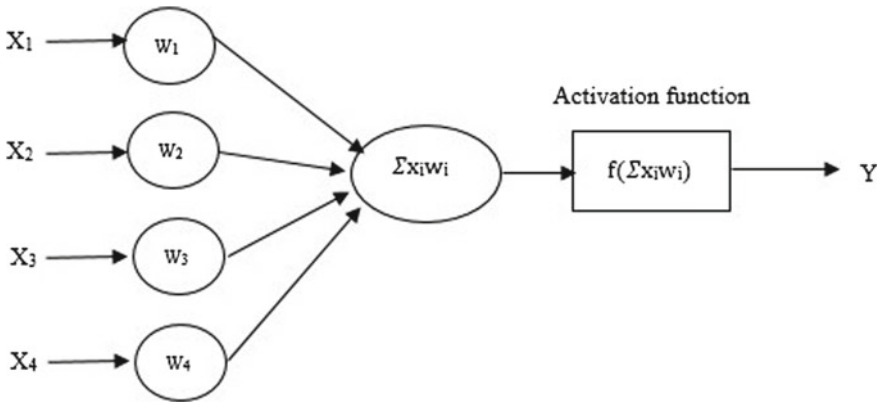


Fig. 4.2 Artificial neuron architecture

4.3 Convolutional Neural Network (CNN)

Neural networks or popularly known as artificial neural networks (ANN) for its mimicking the neural function of the human brain are popular machine learning models. Pioneering research in neural network includes the threshold logic and the perceptron [21].

The architecture of artificial neuron is shown in Fig. 4.2. Each neuron receives one input parameter xi and it also has one weight parameter wi. The dot product between the input vector and weight vector is taken. The answer of dot product is then fed to a nonlinear activation function that produces the output y of the neuron. Sometimes bias term is included which is matched with dummy input with value 1 in input parameters. If neuron has m inputs the output y can be represented as:

$$y = f\left(\sum_{i=0}^{m} x_i w_i\right) \tag{4.1}$$

In ANN, many of these neurons are combined to form a larger network. Many times it has layers of neurons between the input and output which are called hidden layers. The more the hidden layers, deeper the network gets. In modern day neural networks, there are more than hundred hidden layers; therefore, it is sometimes referred as deep neural networks. In some networks, every neuron is interconnected with neurons in the next layer. These networks are referred as fully connected Network.

The choice of activation function f is very important in designing the neural network. The neuron without an activation function f is a simple linear network which cannot be used to solve nonlinear problems like XOR problem. When the network is linear, adding more layers or making network deeper won't help as network will still remain linear [8]. So, to introduce some kind of nonlinearity in the network different types of activation functions like sigmoid function, tanh function or Rectified Linear Unit (ReLU) function is used [8]. The mathematical equations for all these functions are given below.

Sigmoid: $f\,(z) = \frac{1}{1+e^{-z}}$

tanh: $f\,(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$

ReLU: $f\,(z) = \max(0, z)$

The sigmoid and tanh or hyperbolic tangent functions have more or less similar shape but range of sigmoid function is between 0 and 1, while range of tanh function is between -1 and 1. Almost always other than output layer tanh function works better because it has zero mean so it has an effect of centering the data which makes training better for the next layer. For output layer, sigmoid function will work better as it has a range between 0 and 1 that will indicate the probability of output and probability cannot be negative. ReLU is a lightweight and very easy way to create a nonlinear network, so it is becoming increasingly popular for all layers other than output layer. ReLU function is not differentiable at zero, but it solves the problem of gradient saturation and slower computation of sigmoid and tanh function [8]. For multi-class classification problems, output layer uses a special activation function called softmax activation function [22] which takes output vector of k arbitrarily large values as input and converts it to values between 0 and 1 so that sum of these values is 1.

Softmax: $f\,(z) = \frac{e^{z}k}{\sum_{k=1}^{k} e^{z}k}$

The weights of the network are iteratively selected during the training of neural network to achieve the desired output. Gradient descent algorithm and its variants are used to arrive at optimal value of weights [8, 22]. The detail of these algorithms is explained later on in training of CNN.

The solving of object detection and classification problem using these neural networks involves processing so much data and train billions of weights for a modest image size. If monochrome image of 600×600 is considered and each pixel in the image is given as input to the fully connected network each neuron will require

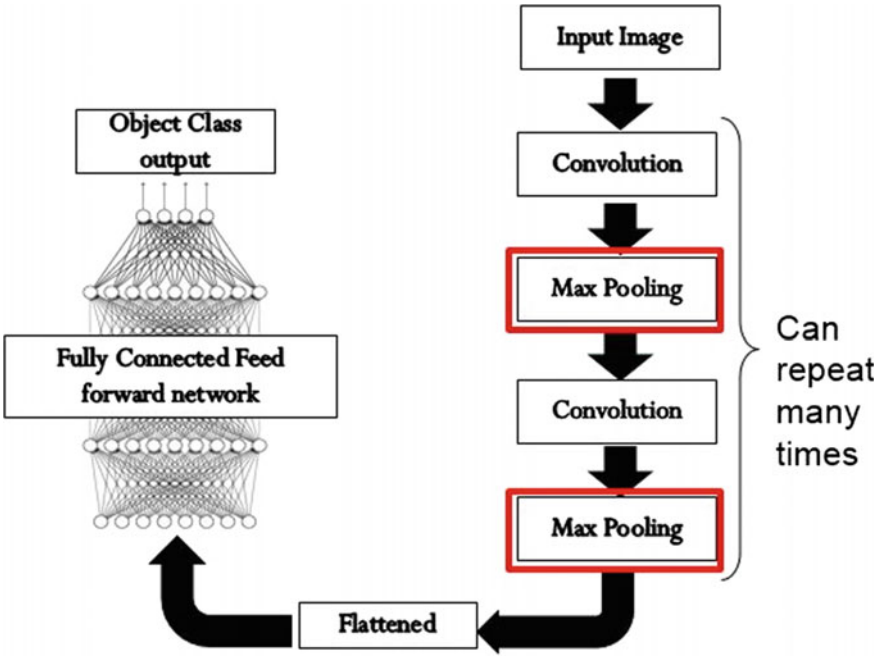


Fig. 4.3 CNN architecture

3,60,000 weights. If it is a color image with R, G and B channels than number of parameters are multiplied by three. Nowadays most of the image data is in high definition and it is impractical to have each pixel as input to the neural network. So there is a need to reduce these trainable parameters in neural network.

Second case specific to computer vision applications is that it is almost always translation invariant. The filter that can find edges in the top left corner can be used to find edges in bottom right corner so it is impractical to train separate weights to find similar features [23]. A fully connected network fails to recognize local neighborhood patterns that are very important to find features in an image. So in the case of 600×600 image if 6 filters of 5×5 size are used to find feature maps from an image it only needs 25 parameters of filter and 1 bias parameters so total 26 parameters per filter. Total parameters needed are 156 which is drastically lower than first layer of artificial neural network. Therefore, convolutional neural network has two advantages: one is parameter sharing where same filter can be used to find similar features throughout an image and second is scarcity of weight connection, as number of weights to be learnt per layer is reduced. These two advantages led to development of CNN [24]. The architecture of CNN is shown below in Fig. 4.3.

The input image given to CNN passes through series of convolution and pooling layers. Total number of these layers and number of filters in each layer will vary depending on the network architecture and these parameters are called hyper-

parameters. The final output of these layers is passed through flattening layer to convert the output in to a single column vector. This vector is passed through a fully connected layer similar to ANN seen previously. The output layer of this network will consist of neurons that are equal to number of classes for classification. It will use a softmax activation that will give class probabilities of each class. Each layer of CNN and its importance is explained below.

4.3.1 Convolution Layer

Convolution layers are the main work horse in CNN. The basic idea of convolutions evolved from the similar idea in biology called receptive field where it is sensitive to some part in image and insensitive to other part [24, 25]. It can be mathematically represented as:

$$g\,(x, y) = f\,(x, y) * h\,(x, y) = \sum_{n} \sum_{m} f\,(n, m)\, h(x - n, y - m) \tag{4.2}$$

In simplified form, this equation is a dot product between filter h and a sub-image of image f centered around (x, y) point. The answer of this product is equal to (x, y) point in image g. To illustrate working of convolution operation on an image, example of 3×3 filter applied to an image of size 6×6 is shown in Fig. 4.4. To find the first point in an image, dot product is taken between leftmost window is shown in red with the filter. The answer of the dot product will be $3(1 * 1 + -1 * 0 + -1 * 0 + -1 * 0 + 1 * 1 + -1 * 0 + -1 * 0 + -1 * 0 + 1 * 1)$. Same operation is repeated after moving 3×3 window by 1 pixel. It is called window stride of one pixel. The number of pixels traveled in both directions between two windows is indicated by window stride. The sizes of the filter and window strides are hyper-parameters that can be chosen by user according to application.

This dot product can be repeated for all windows in an image. The result will be a 4×4 image as shown in Fig. 4.5. The blue line and blue box image indicate the concept of receptive field where value of the output is maximum when pattern similar to filter is detected in input image. The output image is called feature map or feature detector in CNN.

The size of output image is determined by following equation.

$$(\mathrm{x}, \mathrm{y}) = \left(\frac{n + 2p - f}{s} + 1, \frac{m + 2p - f}{s} + 1 \right) \tag{4.3}$$

Where (x, y) indicates height and width of output image and (n, m) is the height and width of input image. f $\times$ f indicates the filter size. p indicates the value of padding applied before the convolution operation. The number of pixels added on the boundary of an image is indicated by padding. If no padding is added then size of

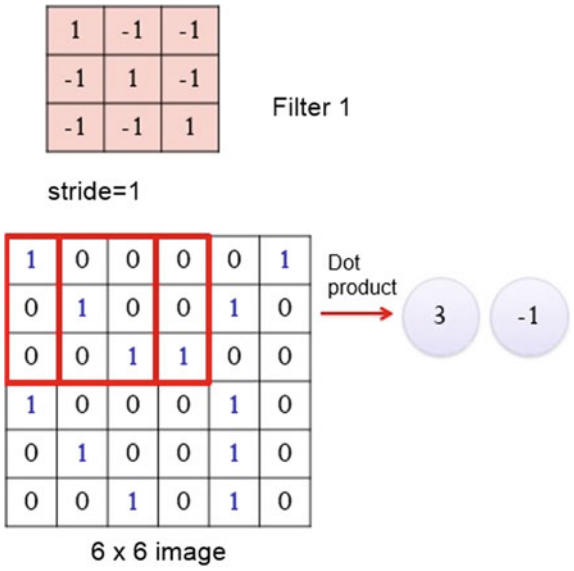


Fig. 4.4 Simplified convolution operation

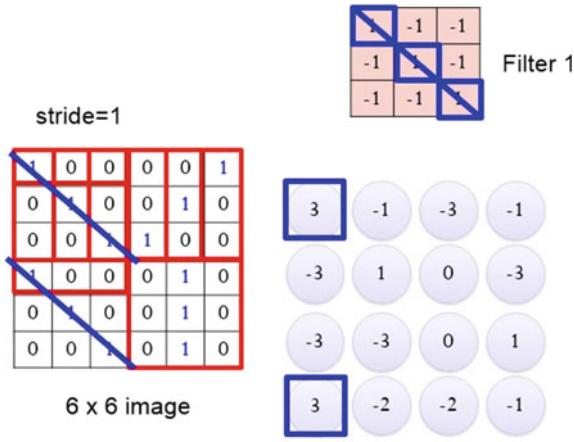


Fig. 4.5 Result of convolution operation on simple image

output image will decrease. Sometimes normal padding is applied that keeps output image size similar to input image. s indicates window stride which indicate number of pixel that is traveled in both directions between two windows. Large value of window stride will drastically reduce output image size. So if this equation is applied to above problem then (m, n) is (6, 6). No padding is applied and window stride is 1. Filter size f is equal to 3. If these values are put into Eq. (4.3), value of (x, y) will be (4, 4).

The above example illustrates convolution operation in monochrome image which contains single channel. For color RGB image with three channels, these filters are applied on all three channels. Filter sizes will be a $3 \times 3 \times 3$ volume. Again the resulting feature map after dot product between 27 overlapping elements will be of

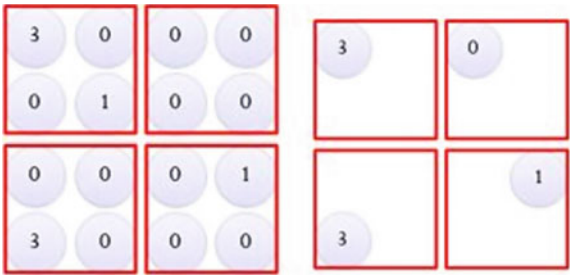


Fig. 4.6 Maximum pooling operation applied on image

size 4 × 4. If n such filters are applied in one convolutional layer output feature map will be 4 × 4× n.

As in ANN sometimes to introduce nonlinearity in computation output of CNN is passed through an activation function. Here again ReLU is most famous because of its simplicity and ease of computation. This output of activation function is applied to pooling layer.

4.3.2 Pooling Layer

CNN uses pooling layers to reduce the size of the feature vectors, to speed up the computation, as well as make some of the features that are detected a bit more robust [26]. How pooling is applied to same 4 × 4 image which was outputted by convolution layer in Fig. 4.5. Pooling layer uses output of the convolution after applying ReLU activation to the output.

Pooling follows a simple idea that by subsampling pixels or making image smaller will not change the object but it will reduce the parameters that characterize the image [26]. Pooling operation can be maximum pooling or average pooling. As maximum value in the window indicates maximum receptive field or high similarity with the filter pattern, max pooling is used more often. In this pooling operation, image is divided into 2 × 2 windows with a window stride of 2 as can be seen from red windows in Fig. 4.6. After that maximum value in this 2 × 2 window is kept and other 3 values are thrown away. So in a way pooling operation reduces the image size by 75% when window size is 2 × 2 and stride is 2 which is common. So output of a 4 × 4 image shown in Fig. 4.6 will be a 2 × 2 image.

The thing to note is even after applying pooling layer feature with maximum value detected at top right and bottom right is preserved. One more thing to notice is that there are no parameters that have to be trained for pooling operation. It only requires two hyper- parameters window size and strides. So pooling layers make CNN translation invariant and reduce image size without adding any training cost [27].

4.3.3 Flattening Layer

Fully connected layer requires single column vector as an input, flattening layer is a very simple layer sandwiched between convolutional plus pooling layer and fully connected layer. It basically converts output volume from pooling layers to a single column vector that can be used as input feature vector for fully connected network. For this layer, learning of parameters is not required. Also there are no hyper-parameters to be decided.

4.3.4 Fully Connected Layer

The final layers of a CNN are fully connected layers which are used to capture interesting relationships that parameter-sharing convolutional layers cannot [11]. The size of feature vector reaching this layer should be sufficiently small. Pooling and stride settings in the previous layers are used to reduce the size of feature vector. The final output of this layer will be a layer with softmax activation which will give probabilities of each class in classification. These layers have maximum parameters to train in CNN.

4.3.5 Additional Layers

Apart from these four layers described above, some architectures also use additional layers according to application. Most of these layers are used to prevent over-fitting. The simple technique to prevent over-fitting is to penalize arbitrarily large parameter values in cost function that will prevent these values getting too large.

Local response normalization (LRN) [17] can be used to prevent over- fitting or regularization. It is normally used after convolutional layer to normalize the response. It normalizes activity of every neurons in that layer.

The second popular technique used for regularization is called dropout [28]. In this technique, random neurons are dropped during training which changes the architecture of network each time during training. This will make the network independent of any single neuron value which is essential to prevent over-fitting. It is also computationally inexpensive [28].

4.3.6 Training of CNN

CNN is trained by selecting the values of all weights so that it can approximate target output from the known input values. This is called supervised learning [22]. CNN

learns these parameters by a simple iterative method called backpropagation and gradient descent as optimization method [17, 22].

In the first phase, all the weights are initialized with small random values and input is passed through CNN to approximate the output value. The output is called predicted value and this step is called forward propagation. Next step is to calculate the difference between the predicted value and true value in terms of error function. This error function can be mean square error, absolute error or binary cross entropy. Mean square error is more popular in regression problems, while cross entropy is popular in classification task. After calculating error function which is also called cost function, gradient of this error is calculated which is passed backwards to update the value of weights. The rate at which weights are updated is called learning rate and it is very important hyper- parameter [22]. If it is too small, it will take long time for training to converge at minimum cost function. If it is large, it may never reach minimum but it will be zigzagging around the minimum value [22]. The gradient is calculated by taking derivative of cost function with respect to parameters and derivative is back propagated backwards by using chain rule [29].

Learning methods are divided based on when the parameters are updated in training stage. If weights are updated after every input it is called online learning. It has advantage as it will require low memory and less computing resource because it needs to process single input at a time. But it has disadvantage of gradient zigzagging around the minimum value. It is also called stochastic gradient descent. The opposite of this method is full batch learning where training parameters are updated after processing all input data. It can be slow and requires more memory and computing resource but will find minimum value. To overcome difficulties in both methods mini-batch learning [30] is used where training parameters are updates after a set of input called batch size. Again this batch size is a hyper-parameter that has to be chosen with proper care.

4.4 Case Study of CNN Architectures

In this section, building dedicated CNN architectures are discussed using the building boxes explained in the last section. The need of case study of basic CNN architectures can be understood by the example of people learn coding by seeing someone else code and more importantly CNN architecture that worked for one computer vision problem will work for other with no or minor modification. So in this section, a brief case study of some famous CNN architectures used for object detection and classification are explained. The detailed architecture of every network is beyond the scope of this chapter and can be found in respective papers. Main contributions of each architecture are discussed in this section.

4.4.1 LeNet-5

LeNet-5 [14] was first CNN-based architecture introduced and it was mainly used for handwritten digit classification [14]. The name itself indicates it has 5 layers. Two layers combined of convolution and padding, two fully connected layers and output softmax layer. It used convolution filter of 5×5 and stride of 1 in convolution layer. For pooling layers size of the window is 2×2 with window stride of 2 [14].

4.4.2 AlexNet

AlexNet [9] was introduced in 2012 and it won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC-2012) challenge [31] of classifying 1.2 million high resolution images from ImageNet database [32]. Main features of AlexNet are as follows:

1. It has achieved top-5 error rate of 15.3% on ImageNet dataset.
2. It has eight trainable layers with five convolution layers and three fully connected layers.
3. It has used ReLU activation for the first time in CNN architectures.
4. It also used local response normalization and data augmentation techniques to avoid over-fitting.
5. It used dropout of 0.5 in fully connected layers.
6. It was trained on GTX 580 GPU with only 3 GB of memory. The network spread across 2 GPUs, half the neurons (feature maps) on each GPU [9].

4.4.3 ZFNet

ZFNet [18] was introduced in 2013 and won the ILSVRC-2013 challenge [31]. The architecture of ZFnet is very similar to AlexNet with minor modifications. Main features of ZFNet are as follows:

1. It reduced the 15.3% top-5 test error rate of AlexNet to 11.7%.
2. It again has eight layers.
3. The first convolution layer of 11×11 and window stride 4 in AlexNet is changed to convolution layer of 7×7 and window stride 2 in ZFNet.
4. In Conv3, 4, 5 layers instead of 384, 384, 256 filters in AlexNet it uses 512, 1024, 512 filters [18].

4.4.4 VGGNet

VGGNet [17] was introduced in 2014 and it won the ILSVRC-2014 localization challenge and came second in classification challenge [31]. It has two variant of 16 and 19 layers called VGG16 and VGG19, respectively. Main features of VGGNet are as follows:

1. It reduced the 11.7% top-5 test error rate of ZFNet to 7.3%.
2. It introduced the concept of smaller filters and deeper networks by removing the need of using large filter sizes. It only used filter size of 3×3 filter as opposed to filter sizes of 11×11 and 7×7 in the earlier networks.
3. It compensated reduction in filter size with increase in number of layers. The concept that stack of three 3×3 convolution layers with a stride of 1 has same effective receptive field as one 7×7 convolution layer. This type of deeper network has advantage of more nonlinearities and reduction in number of parameters from $72 * n^2$ to $3 * (3 * n^2)$ [17].

4.4.5 GoogleNet

GoogleNet [15] was introduced in 2014 and it won the ILSVRC-2014 classification challenge [31]. Main features of GoogleNet are as follows:

1. It reduced the 11.7% top-5 test error rate of ZFNet to 6.7%.
2. It has 22 layers. It is a deep network with low computational complexity.
3. It has no fully connected layer and has 5 million parameters which is 12x less than AlexNet.
4. It achieved this simplification because of the introduction of inception module which optimally utilized computing resource with network within network architecture.

As shown in Fig. 4.7, the inception module contains application of multiple parallel filter sizes for convolution (1×1, 3×3, 5×5) on the input from previous layer and then application of pooling operation (3×3). Sometimes 1×1 convolution layers are stacked in between for dimensionality reduction. After this, output of all filters is concatenated together in depthwise manner. These inception modules are stacked together to build overall GoogleNet architecture [31].

4.4.6 ResNet

ResNet [16] was introduced in 2015 and it won the ILSVRC-2015 classification challenge [31]. Main features of ResNet are as follows:

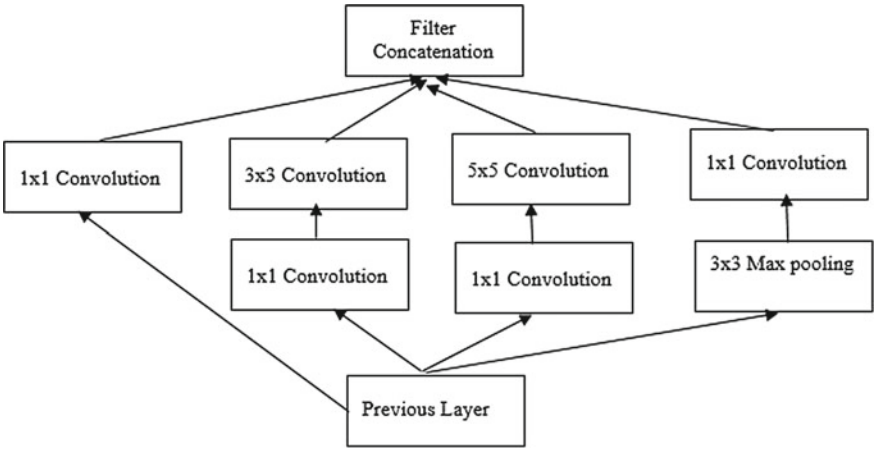


Fig. 4.7 Basic inception module [15]

1. It reduced the 6.7% top-5 test error rate of GoogleNet to 3.57%.
2. It is a very deep network using residual connections.
3. It brought about revolution in depth with 152 layer architecture. If more and more layers are stacked on a plain CNN then after some layers it will start to perform worse than shallow network. This is not because of over-fitting but it is optimization problem. Deeper networks are difficult to optimize. They have gradient vanishing problems.

A solution was found in ResNet as shown in Fig. 4.8 by copying the learned layers from the shallower model and setting additional layers to fit a residual mapping [16].

4.4.7 MobileNet

MobileNet [19] was introduced in 2017 to deploy CNN architectures on embedded systems or mobile environment. Main features of MobileNet are as follows:

1. It achieved 70.6% accuracy compared to 69.8% of GoogleNet for ImageNet [32] dataset.
2. It introduced the concept of depthwise separable convolution which reduces number of parameters for training without sacrificing accuracy. For example, a regular 3×3 convolution over 16 input channels and 32 output channels does the following: every single channel of the 16 channels is traversed by 32, 3×3 kernels resulting in a total of 4608 ($16 \times 32 \times 3 \times 3$) parameters. Thirty-two different feature maps for each of the 16 channels will be generated. Now take one feature map out of every 16 input channels and add them together. Since it can be done 32 times, 32 output channels will be generated. For depth wise separable convo-

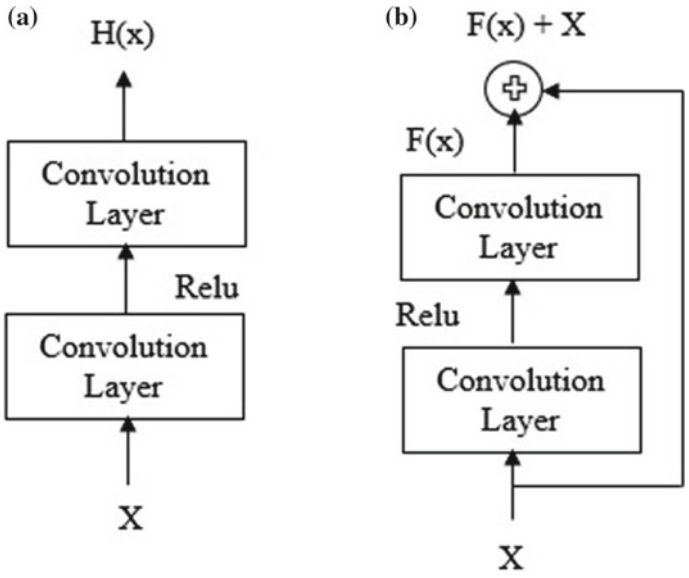


Fig. 4.8 **a** Plain layer **b** Residual block [16]

lutions on the same setup each of the 16 channels with 1 3x3 kernel is traversed resulting in 16 feature maps. Each of these feature maps is then traversed by 32 (1×1) convolutions resulting in 512 (16×32) feature maps. Now one feature map out of each of the 16 input channels is taken and added. Since it can be done 32 times, 32 output channels will be generated. The total number of parameters can be calculated by $(16 \times 3 \times 3) + (16 \times 32 \times 1 \times 1) = 656$ parameters.

3. This simplification results in lower training and testing time for Mobilenet.
4. It also reduces the size of memory needed to store this model which is very important for embedded devices [19].

4.4.8 Comparison of Deep Learning Architectures for Object Classification

In this part, the comparison between different CNN-based architecture used for image classification is made in terms of top-5 test error rate on ImageNet dataset of 1000 classes. Basically Table 4.2 and Fig. 4.9 signify total objects not correctly classified in top 5 choices given by algorithm.

Table 4.2 Comparison of well-known CNN architectures for object classification

Deep learning architecture	No of layers	Top-5 test error rate (%)
Shallow networks(2011)	–	25.2
AlexNet (2012) [9]	8	16.4
ZFNet (2013) [18]	8	11.5
VGG19 (2014) [17]	19	7.3
GoogleNet (2014) [15]	22	6.7
ResNet (2015) [16]	152	3.57

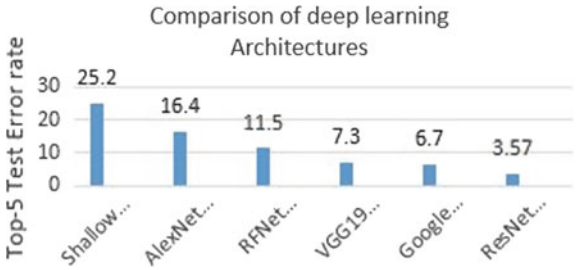


Fig. 4.9 Comparison of deep learning architectures in terms of top-5 test error rate

4.5 Object Detection Based on CNN

In this section, object detection methods that utilized CNN-based architectures are discussed and compared. Most methods like R-CNN [10], fast R-CNN [3] and faster R-CNN [11] which combines CNN with region proposal networks are discussed. Then methods such as Single Shot multi-box Detector (SSD) [13] and You Only Look Once (YOLO) [12] are discussed which detects object in a single pass of CNN without explicit region proposals.

4.5.1 R-CNN

In 2013, Girshick et al. published a method [10] called R-CNN: Regions with CNN features by generalizing the results of image classification method published by Krizhevsky et al. [9] for object detection.

Figure 4.10 shows several computation stages for R-CNN. In the first stage, region of interest also called region proposals are generated that has high likelihood of containing interesting object. Selective search algorithm [33] is used for generating region proposals. The warped sub-image matched with input image size of CNN is extracted from region proposals and fed to the network. CNN extracts features from each of the warped sub images of region proposals. These extracted features are fed

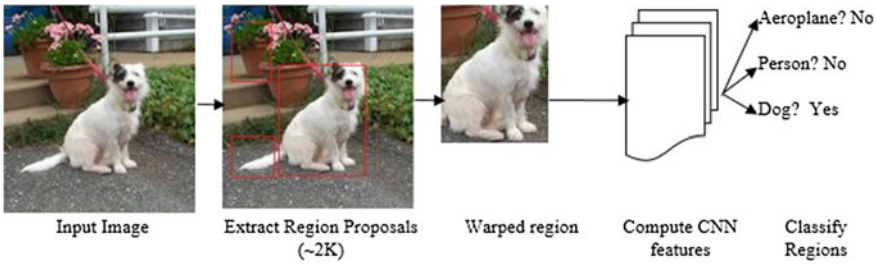


Fig. 4.10 Stages in R-CNN computations

to the SVM classification algorithm which classifies the object in to one of several classes.

R-CNN contains three separate sub-network which has to be trained separately. First CNN has to be trained to extract features from the image. Then SVM has to be trained that fit features extracted by CNN to number of classes. Region proposal algorithm like selective search has to be trained separately. So training of R-CNN is very expensive both in terms of time and computing resource utilization [3]. As features extracted from each region proposal is used to train SVM and region proposal network, it needs to be stored in memory. It requires large memory (in terms of GB) for a large database. This training will take many days to complete [3].

Detection of object is also slow during testing stage. It will take almost a minute to detect objects from image even with high-end GPUs [10]. Each region proposal has to be processed separately through forward computation of CNN even if regions are overlapping so it will take lots of time to extract features from each sub-image.

The main advantage of this method is that it improves mean average precision (mAP) by more than 30% on VOC 2012 achieving a mAP of 53.3%. R-CNN with VGGNet took 84 h to train and around 47 s per image during testing.

Region proposal method plays important role in performance of R-CNN object detection system. Normally region proposals are generated to minimize recall so that each bounding box that contains the probable object is generated in region proposals [34]. It cares less about precision as false positives can be removed later on in the object detection pipeline. Selective search [33] is preferred method for generating region proposals. It helps to utilize a hierarchical partitioning and iterative merging of super pixel of an image.

4.5.2 Fast R-CNN

As mentioned earlier, R-CNN has drawback of processing each sub regions separately through CNN. It is computationally inefficient even though regions overlapped and similar features needed to be computed. This drawback was removed in fast R-CNN as it processes entire image in a single pass through CNN [3].

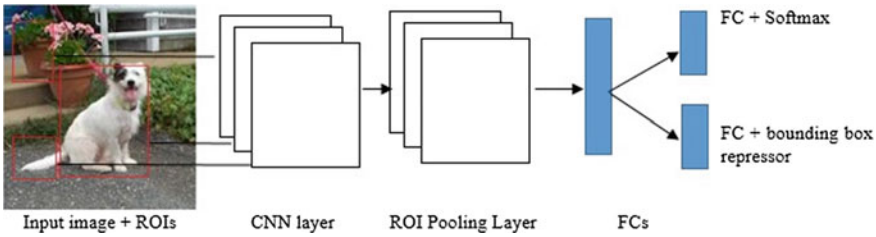


Fig. 4.11 Stages of Fast R-CNN forward computation [3]

Fast R-CNN still needs a separate region proposal network (RPN). The computation stages for fast R-CNN is shown in Fig. 4.11. It takes an image and region proposals generated by RPN as input. It passes entire image through convolution neural network which applies series of convolution and pooling operations on this image. Fast R-CNN contains one special layer called RoI pooling layer which extracts fix size feature vector for each region proposal from a feature map given by CNN. These vectors are given as input to next FCN. Fully connected layer is connected to two separate output layer. One output layer is a softmax layer that predicts the class of object. Second is the regression layer that is used to predict bounding box offset.

Training of fast R-CNN is simple than R-CNN. It does not have a separate SVM classification stage. So entire training of CNN and RoI pooling layers can be completed in one go. RoIs from the same image share computation resource and memory. So it reduces training time by a factor of nine [3].

As it passes entire image through CNN, testing time for an image also decreases than R-CNN. It takes less than a second to detect object in an image on a computer with GPU [3]. The overall computation time depends largely on the performance of the region proposal network. Also computation time increases with increase in RoIs which takes more time to pass through fully connected network. Fast R-CNN achieves 66.9% accuracy [3] on VOC 2009 dataset [35].

4.5.3 Faster R-CNN

The authors of Faster R-CNN thought of using same feature maps for object detection and generation region proposals [11]. So they proposed an integrated approach which uses shared convolution layers for object detection and region proposal generation. The part of CNN network which is used for generating regions of interest is called Region proposal Network (RPN) [11].

The shared convolutional layers in faster R-CNN generates feature maps from the input image. These features are passed through region proposal network which generates region proposals. After that both feature maps and region proposals are passed through ROI pooling layers and output layers same as the case in fast R-CNN.

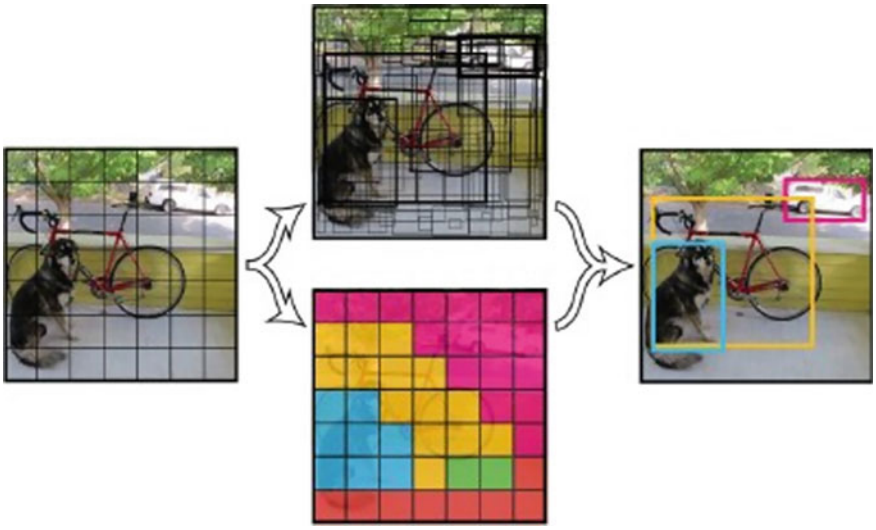


Fig. 4.12 Steps in object detection using YOLO algorithm

Training procedure is little bit tricky in faster CNN as same network is used for two purposes. So this network is trained by switching between region proposal generation and object detection.

It is important to note that because region proposal network utilizes CNN architecture it can be realized on GPU because of its parallel nature which was not the case with selective search method that can only be realized on CPU. This removes the bottleneck of region proposals taking maximum computing time in fast R-CNN. In faster R-CNN because of the shared convolutional layers, region proposal generation is almost cost-free.

This method also removed the need of generating pyramid of scaled images or filters by introducing concept of anchor boxes which deals with detection windows of different shapes and sizes. Faster R-CNN achieves mAP of 66.9% on pascal VOC 2009 dataset [35]. Faster R-CNN with VGG16 network achieves almost 7 FPS when running on high-end GPUs [11].

4.5.4 You Only Look Once (YOLO)

YOLO [12] is CNN-based object detection algorithm which utilizes different approach for detection than networks seen earlier. It applies single neural network to entire image without the need of separate region proposal generation.

This algorithm divides image into several subregions as shown in Fig. 4.12. Then it predicts bounding boxes and class probabilities for each of this subregion. The

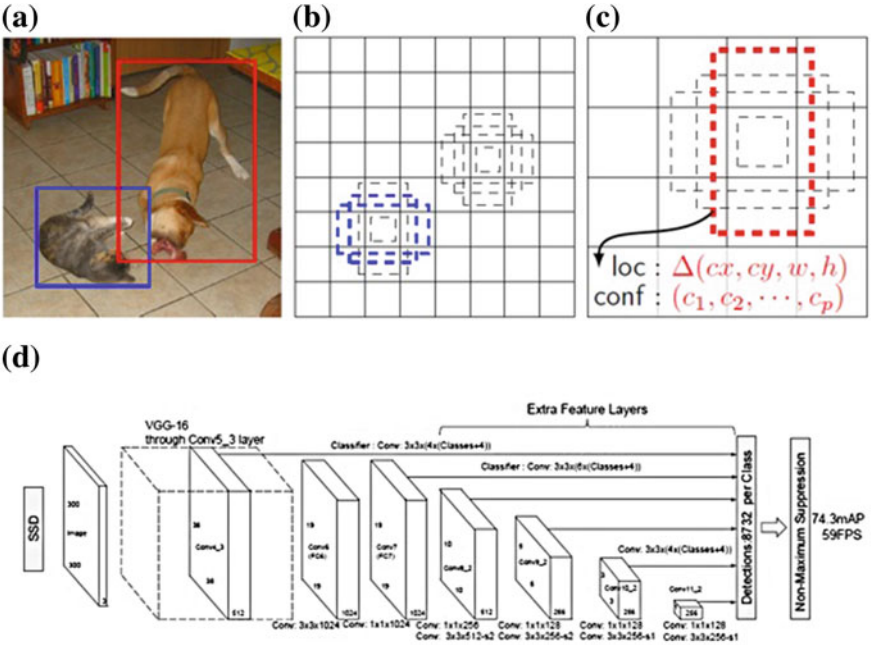


Fig. 4.13 **a** Image with GT boxes **b** 8×8 feature map **c** 4×4 feature map **d** whole SSD architecture [13]

algorithm divides image in to cells of 7×7. For each of these cells, two anchor boxes of different aspect ratios are used to detect objects of different size and scale. The anchor box which contains the center of object is marked as positive. For each anchor boxes, YOLO algorithm predict two values. One value is the class probabilities of each class and second is bounding box coordinates in terms of (x, y, w, h) of rectangle.

YOLO achieves 63.4% mean average precision at 45 frames per second on Pascal VOC 2007 dataset [12]. However by incorporating batch normalization, new fully convolutional network and multiple anchor boxes this accuracy can be increased, but it comes at the cost of decrease in frame rate. YOLO works in real time but it has some disadvantage that it fails to detect smaller objects. It also fails when there is lot of occlusion [12].

4.5.5 Single Shot Multi-box Detector (SSD):

SSD is similar to YOLO in terms that it does not generate region proposals and detect object in single pass through neural network [13].

The architecture of SSD algorithm is shown in Fig. 4.13d. This algorithms begins with default set of rectangular boxes predicted on entire image. Basically a

rectangular grid is overlaid on entire image. To make algorithm invariant to size of object, rectangular boxes of different aspect ratios centered at same origin are used as shown in Fig. 4.13b. For all of these boxes, class predictions probabilities and offset parameters that indicate offset of predicted boxes to default ground truth bounding boxes is predicted as shown in Fig. 4.13c. Loss function for SSD will be the summation of loss in these two parameters: Softmax loss and localization loss (Smooth L1 loss). So the training objective to be minimized for SSD algorithm can be defined by following equation:

$$L(x, c, l, g) = \frac{1}{N}(L_{conf}(x, c) + \propto L_{loc}(x, l, g)) \tag{4.4}$$

Where N indicates number of default set bounding boxes, x is 1 if the default box is matched to a determined ground truth box and 0 otherwise, l is predicted bb parameters, g is ground truth bounding box parameters and c is class probability.

This algorithm deals with different sizes of object by combining output feature maps from different convolutional layers as input to final classifier. It can be seen that this algorithm generate dense set of bounding boxes. To remove boxes with low confidence score or high intersection over union with other boxes, Non-Maximum suppression (NMS) stage is used.

Two versions of SSD algorithms were proposed in the paper. One with input size of 300×300 (SSD300) and other with input with 512×512 (SSD512) [13]. SSD300 achieves Mean average precision (mAP) of 74.3% on PASCAL VOC2017 dataset at 46 fps while SSD512 achieved mAP of 76.8% on the same dataset at 19 fps [13]. There is a trade-off between precision and frame rate. If more rectangular boxes are used then precision will increase but fps will decrease. Same way if feature maps of many layers are used for classification then precision will increase but fps will decrease.

4.5.6 Comparison Between Deep Learning Algorithms for Object Detection

Normally the performance of object detection algorithms is evaluated based on two well-known parameters: mAP and speed of algorithms measured in terms of frames per second. So in this section all the algorithms stated above is compared in terms of frames per second and mean average precision. The performance is measured for common dataset and results are shown in Table 4.3 and Fig. 4.14.

As can be seen from the graph and table, methods like YOLO and SSD works better both in terms of speed and accuracy. These results are taken from original papers in which they were introduced. Some of these algorithms may have improved speed and accuracy in later version which has not been taken in to account.

Table 4.3 Comparison of object detection algorithms

Detection framework	Train dataset	mAP (%)	FPS
Fast R-CNN [3]	VOC 2007 + 2012	70.0	0.5
Faster R-CNN VGG16 [11]	VOC 2007 + 2012	73.2	7
Faster R-CNN ResNet [16]	VOC 2007 + 2012	76.4	5
YOLO [12]	VOC 2007 + 2012	63.4	45
SSD300 [13]	VOC 2007 + 2012	74.3	46
SSD512 [13]	VOC 2007 + 2012	76.8	19

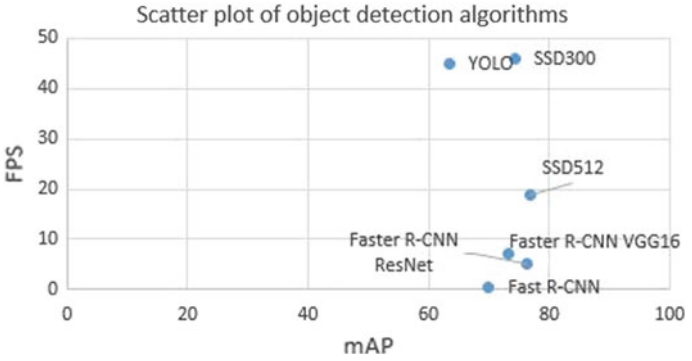


Fig. 4.14 Scatter plot comparing FPS and mAP of different object detection algorithms

4.6 Transfer Learning

Most of the deep learning models used for real-time application such as object detection or speech recognition requires large amount of data. So it will require large computing resources in terms of RAM as well as GPUs. Large RAMs are available for supercomputers relatively cheaply and easily. However, access to GPUs with hundreds of GBs of virtual RAMs is hard for normal researchers and it comes with a high cost. The second important thing to note that even if one has access to large computing resources still it will take days to train these deep learning models. There is a large turnaround time. It is very costly to run algorithms for days after making small changes particularly in research scenarios [36].

To overcome these difficulties, concept of transfer learning is developed which enables one to use pre-trained model by making small changes in it and retrain it. Basically it transfers the learning that solved one problem to learning that solves one similar problem. It can be best understood with a teacher–student analogy. Teacher with years of experience in subject he/she can give concise and brief overview of all the knowledge accumulated in fixed duration of a lecture. It is a transfer of

Table 4.4 List of available pre trained models for object detection on COCO dataset

Model name	Speed (ms)	COCO mAP (%)	Outputs
ssd_mobilenet_v1_coco	30	21	Boxes
ssd_inception_v2_coco	42	24	Boxes
faster_rcnn_inception_v2_coco	58	28	Boxes
faster_rcnn_resnet50_coco	89	30	Boxes
faster_rcnn_resnet50_lowproposals_coco	64	–	Boxes
rfcn_resnet101_coco	92	30	Boxes
faster_rcnn_resnet101_coco	106	32	Boxes
faster_rcnn_resnet101_lowproposals_coco	82	–	Boxes
faster_rcnn_inception_resnet_v2_atrous_coco	620	37	Boxes
faster_rcnn_inception_resnet_v2_atrous_lowproposals_coco	241	–	Boxes
faster_rcnn_nas	1833	43	Boxes
faster_rcnn_nas_lowproposals_coco	540	–	Boxes

knowledge from experienced teacher (Pre-trained model) to a newbie (New model) trying to study similar topic.

For deep neural networks, the knowledge will be in terms of weights or feature maps that it has learnt previously so in transfer learning these weights will get transferred to new network. For example, if somebody is working on autonomous vehicle application then it will take years if one start from scratch. By using transfer learning, one can use pre-trained models of inception or mobilenet with SSD algorithm to solve similar problem. It will not give optimal result for a particular application like autonomous vehicle but still it will be way better than starting from scratch.

The list of pre-trained deep learning models for object detection is given in Table 4.4 with its speed and accuracy. They are trained on COCO dataset which contains object of 80 classes [37]. These models are open source and can be downloaded from github [38].

Though transfer learning is a great idea but it should be used with proper care. One should be very careful in choosing which pre-trained model to use for a particular application. A pre-trained deep learning model of speech recognition taken to solve an object detection problem will give horrible results whereas pre-trained network for detecting car will give good result when it is transferred to detect bus or auto rickshaw.

This idea of transfer learning is particularly useful for object detection applications because in most of the models lower layers will find features such as edges and corners which are independent of object detection. So one can transfer weights of these layers to other network and only train higher layers.

There are many ways in which existing pre-trained model can be fine-tuned to make it usable in other applications via transfer learning. If output softmax layer of CNN architecture is removed then CNN can also be used as feature extractor. One

can reuse entire CNN network as feature extractor and only add output softmax layer according to application. This method will be very useful when size of dataset for new application has high similarity with dataset of pre-trained network. Again this is very useful for object detection, as many pre-trained networks are available that are trained on ImageNet dataset which contains objects with 1000 classes. So more or less, these models can be used to detect new objects just by modifying output layer. When large dataset is available for new application then entire network can also be retrained on new dataset by taking pre-trained weights as initial values of weights. This will take short time to converge then random initialization of weights and give optimal result for new application.

One can also freeze some layers of pre-trained models and choose to train some other layers according to application. Most of the deep learning libraries provide this facility of freezing and unfreezing. Again this will help in reduction of training time for a new application. When there is no data similarity with pre-trained models and size of new dataset is large then it is better to train new model from scratch then use a pre-trained networks.

4.7 Conclusion

The basic building block for deep learning models in computer vision application is CNN. It is explained in detail along with difference between conventional ANN and CNN. It can be observed that after invention of CNN for object detection and classification, the mean average precision for object detection has almost doubled while classification accuracy has overtaken human performance. The winners of well-known ILSVRC object detect and classification challenge in last 5–6 years have used CNN-based architectures. The comparison between famous CNN architectures for object classification is done in terms of classification accuracy. ResNet provides highest classification accuracy but it takes more time to classify while GoogleNet uses inception module to speed up the operation. MobileNet uses depthwise convolution to reduce number of parameters for training which reduces memory size and computational cost. It is suitable for deploying CNN-based models on mobile and embedded devices. Comparison between CNN-based object detection algorithms is done in terms of speed and accuracy. One can conclude from comparison that SSD is optimal solution at the moment for object detection using deep learning both in terms of speed as well as accuracy.

In the last section, the topic of transfer learning which is gaining lot of popularity among deep learning researchers is discussed along with details of how and when to use it for new applications.

Though it may seem that optimal real-time performance has been achieved by using these algorithms, lots of improvement can be made. Most of these algorithms need a high end costly GPUs for real-time operation. Research in direction of how to use CNN-based algorithms on consumer laptops or mobile and embedded applications can be accelerated. Though classification algorithms have reached human

level performance, detection algorithms have far way to go. Still there is a scope for large amount of improvement in mean precision value. One thing is sure that deep learning will continue to be an active area of research in next decade and it will be instrumental in transforming life of human at large.

References

1. Kpcb Internet Trends Report 2014. http://www.kpcb.com/blog/2014-internet-trends. Accessed 20 June 2017
2. Szeliski, R.: Computer Vision: Algorithms and Applications. Springer Science & Business Media, Berlin (2010)
3. Girshick, R.: Fast R-CNN. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 1440–1448 (2015)
4. Walther, D., Itti, L., Riesenhuber, M., Poggio, T., Koch, C.: Attentional selection for object recognition - a gentle way. In: International Workshop on Biologically Motivated Computer Vision, pp. 472–479. Springer (2002)
5. Lowe, D. G.: Object recognition from local scale-invariant features. In: The Proceedings of the Seventh IEEE International Conference on Computer Vision, 1999, vol. 2, pp. 1150–1157. IEEE (1999)
6. Bay, H., Tuytelaars, T., Van Gool, L.: Surf: speeded up robust features. In: Computer Vision ECCV 2006, pp. 404–417 (2006)
7. Dalal, N., Triggs, B.: Histograms of oriented gradients for human detection. In: 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2005), vol. 1, pp. 886–893. IEEE (2005)
8. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press, Cambridge (2016). http://www.deeplearningbook.org
9. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems, pp. 1097–1105 (2012)
10. Girshick, R., Donahue, J., Darrell, T., Malik, J.: Rich: feature hierarchies for accurate object detection and semantic segmentation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 580–587 (2014)
11. Ren, S., He, K., Girshick, R., Sun, J.: Faster R-CNN: towards real-time object detection with region proposal networks. In: Advances in Neural Information Processing Systems, pp. 91–99 (2015)
12. Redmon, J., Divvala, S., Girshick, R., Farhadi, A.: You only look once: unified, real-time object detection. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 779–788 (2016)
13. Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., Berg, A.C.: Ssd: single shot multibox detector. In: European Conference on Computer Vision, pp. 21–37. Springer, Berlin (2016)
14. LeCun, Y.: LeNet-5, Convolutional Neural Networks (2015). http://yann.lecun.com/exdb/lenet
15. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Rabinovich, A.: Going deeper with convolutions. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1–9 (2015)
16. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 770–778 (2016)
17. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition (2014). arXiv:1409.1556

18. Zeiler, M.D., Fergus, R.: Visualizing and understanding convolutional networks. In: European Conference on Computer Vision, pp. 818–833. Springer, Cham (2014)
19. Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Adam, H.: Mobilenets: efficient convolutional neural networks for mobile vision applications (2019). arXiv:1704.04861
20. Steinkraus, D., Buck, I., Simard, P.: Using GPUs for machine learning algorithms. In: Proceedings of the Eighth International Conference on Document Analysis and Recognition, pp. 1115–1120. IEEE (2005)
21. Rojas, R.: Neural Networks - A Systematic Introduction. Springer, Berlin (1996)
22. Bishop, C.M.: Pattern recognition and machine learning. Information Science and Statistics. Springer, New York Inc, Secaucus (2006)
23. LeCun, Y., Boser, B., Denker, J.S., Henderson, D., Howard, R.E., Hubbard, W., Jackel, L.D.: Backpropagation applied to handwritten zip code recognition. Neural Comput. **1**, 541–551 (1989)
24. Fukushima, K.: Neocognitron: a hierarchical neural network capable of visual pattern recognition. Neural Netw. **1**, 119–130 (1988)
25. Hubel, D.H., Wiesel, T.N.: Receptive fields and functional architecture of monkey striate cortex. J. Physiol. **195**, 215–243 (1968)
26. Chatfield, K., Simonyan, K., Vedaldi, A., Zisserman, A.: Return of the devil in the de-tails: delving deep into convolutional nets (2014). arXiv:1405.3531
27. Springenberg, J.T., Dosovitskiy, A., Brox, T., Riedmiller, M.A.: Striving for simplicity: the all convolutional net (2014). CoRR labs/ arXiv:1412.6806
28. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: a simple way to prevent neural networks from overfitting. J. Mach. Learn. Res. **15**, 1929–1958 (2014)
29. Rumelhart, D.E., Hinton, G.E., Williams, R.J., et al.: Learning representations by back-propagating errors. Cognitive Modeling (1988)
30. Sebe, N.: Machine learning in computer vision, vol. 29. Springer Science & Business Media, Berlin (2005)
31. Imagenet large scale visual recognition challenge. http://image-net.org/challenges/LSVRC/
32. Imagenet database statistics. http://image-net.org/about-stats
33. Van de Sande, K.E., Uijlings, J.R., Gevers, T., Smeulders, A.W.: Segmentation as selective search for object recognition. In: IEEE International Conference on Computer Vision (ICCV), pp. 1879–1886. IEEE (2011)
34. Zitnick, C.L., Dollar, P.: Edge boxes: locating object proposals from edges, pp. 391–405. Springer, Berlin (2014)
35. PASCAL VOC image dataset. http://host.robots.ox.ac.uk/pascal/VOC/
36. Huang, J., Rathod, V., Sun, C., Zhu, M., Korattikara, A., Fathi, A., Murphy, K.: Speed/accuracy trade-offs for modern convolutional object detectors (2016). arXiv:1611.10012
37. COCO object detection dataset. http://cocodataset.org/#home
38. Google object detection API. https://github.com/tensorflow/models/tree/master/research/object_detection

International Journal of Image and Graphics
Vol. 21 (2022) 2250017 (19 pages)
© World Scientific Publishing Company
DOI: 10.1142/S0219467822500176

World Scientific
www.worldscientific.com

Hardware Efficient Modified CNN Architecture for Traffic Sign Detection and Recognition

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 3 January 2021
Accepted 29 March 2021
Published

Traffic sign recognition is a vital part for any driver assistance system which can help in making complex driving decision based on the detected traffic signs. Traffic sign detection (TSD) is essential in adverse weather conditions or when the vehicle is being driven on the hilly roads. Traffic sign recognition is a complex computer vision problem as generally the signs occupy a very small portion of the entire image. A lot of research is going on to solve this issue accurately but still it has not been solved till the satisfactory performance. The goal of this paper is to propose a deep learning architecture which can be deployed on embedded platforms for driver assistant system with limited memory and computing resources without sacrificing on detection accuracy. The architecture uses various architectural modification to the well-known Convolutional Neural Network (CNN) architecture for object detection. It uses a trainable Color Transformer Network (CTN) with the existing CNN architecture for making the system invariant to illumination and light changes. The architecture uses feature fusion module for detecting small traffic signs accurately. In the proposed work, receptive field calculation is used for choosing the number of convolutional layer for prediction and the right scales for default bounding boxes. The architecture is deployed on Jetson Nano GPU Embedded development board for performance evaluation at the edge and it has been tested on well-known German Traffic Sign Detection Benchmark (GTSDB) and Tsinghua-Tencent 100k dataset. The architecture only requires 11 MB for storage which is almost ten times better than the previous architectures. The architecture has one sixth parameters than the best performing architecture and 50 times less floating point operations per second (FLOPs). The architecture achieves running time of 220 ms on desktop GPU and 578 ms on Jetson Nano which is also better compared to other similar implementation. It also achieves comparable accuracy in terms of mean average precision (mAP) for both the datasets.

Keywords: Traffic sign detection; Driver Assistant System (DAS); Convolutional Neural Network (CNN); feature fusion Network; Color Transformer Network (CTN); focal Loss; receptive field; Jetson Nano Development Board.

*Corresponding author.

1. Introduction

Traffic sign detection (TSD) and recognition gained a lot of attraction among computer vision researcher in the recent years as it is an integral part for any autonomous vehicle or driver assistant systems.[1–6] Traffic signs have definite shape and color. They are placed on the sides of the road for indicating traffic rules, speed limits and curvature on the roads. They are used for guiding the drivers in maintaining traffic rules and avoiding accidents on the road.[7] They are placed for notifying drivers about what to expect on the road and making driving decisions accordingly still many drivers misses the sign and get into trouble.[8]

If a system which can detect traffic signs automatically is integrated in the vehicle, then it can help the drivers immensely.[7] The TSD system can also be placed in existing vehicles where it notifies the drivers when they miss the traffic sign boards. It will help in maintaining traffic rules and avoiding accidents particularly in adverse weather conditions of rain and fog.[7] The system can also help in driving on mountains where early detection of curve signs will help in making driving decisions.

TSD and recognition is a very complex computer vision problem. It has still not reached the satisfactory performance levels despite getting a lot of attraction in the recent years. There are many challenges in detecting traffic signs from the images. The challenges due to illumination, occlusion, clutter and deformation are similar to other computer vision problems. One specific problem in traffic signs is that it occupies a very small portion of the entire image and typically it is less than 1% of the image size so detecting such a small object on a large image is not trivial.[9] The second problem is due to intra class variance. Many traffic signs, such as speed limits or curves, have very small intra class variance so detecting and recognizing them from a large images can be a daunting task. The third problem arises in the deployment of the system on edge platforms for real-time performance[9] as they have very limited memory and computational resources. The final challenge is the introduction of motion blur due to capturing image from a high speed moving camera.

The emergence of Machine Learning and Deep Learning has improved the performance of TSD system remarkably despite the challenges listed above. The emergence of Convolutional Neural Network (CNN) has seen a human level performance in traffic sign recognition and has improved the performance in detection task but they require large computing resources, memory and communication bandwidth.[1–9] These requirements are hard to satisfy when the deep learning models are deployed on Edge platforms. The deep learning models also require large labeled dataset which is difficult to find for many applications. Most deep learning architecture processes images with low resolution which needs acquired images to be resized to lower dimensions which further reduces the size of object to be detected.

In the light of the above limitations of deep learning models, the motivation behind the proposed research is to build TSD system which can be deployed on embedded platform with limited memory and computational resources without

sacrificing detection accuracy. The proposed system will help in early driving decisions which can avoid accidents on the road.

The proposed architecture has the following contributions:

(1) It can process images at resolutions of 1360×800 which is very high compared to earlier approaches which needed images to be resized to 300×300 or 512×512.[36] This helps in accurately detecting small traffic signs from large resolution images taken from high definition cameras.
(2) It introduces the trainable CTN for making the network invariant to illumination.
(3) It introduces feature fusion module which is integrated for enhancing the detection performance on small traffic signs. The convolution layers used for feature aggregation are chosen based on receptive field calculation.
(4) It uses focal loss[43] as a loss function instead of classical categorical crossentropy used earlier which avoids class imbalance in the dataset which improves the performance of classes with low training examples.
(5) The network drastically reduces the number of trainable parameters, memory storage and floating point operations per second (FLOPs) so that it can be implemented on Jetson Nano Embedded Development Board.

The rest of the paper is organized as follows. Section 2 summarizes related work on TSD using traditional and learning based methods. The details of the proposed CNN architecture and theoretical aspect are explained in Section 3. Section 4 shows the experiment results and performance calculation for the proposed architecture along with its edge deployment. The last sections contain the conclusion and references used for the work.

2. Related work

There are two main approaches in Computer Vision which are called recognition and detection. Traffic sign recognition deals with identifying class of traffic signs. It gives a label to an image while detection incorporates bounding box around the location of the traffic signs. This paper deals with detection as well as recognition task. There are two major approaches in this domain: The first approach uses traditional texture based methods of using color and shape for detecting and learning based algorithm for classifying traffic signs. The second approach uses end to end learning based methods and in particular CNN based methods for solving the problem. Many researchers have tried to combine these approaches as well.

Traditional methods used color and shapes of traffic signs for detection and classification. Shadeed *et al.*[10] used color segmentation in YUV color space for traffic sign classification. Haojie Li *et al.*[11] combined color and shape features for detecting and classifying traffic signs from the images. De La Escalera *et al.*[12] used color thresholding technique for traffic signs as they come in definite colors.

Shape information can be easily fetched using Hough Transform[13] so Barnes *et al.*[14] used that for detecting traffic signs. Selcan Kaplan Berkaya *et al.*[15] used EDCircle algorithm[16] for detecting traffic signs but it only worked for circular TSD.

Histogram of oriented features (HOG)[17] and Support Vector Machine (SVM)[18] are very popular algorithms for detecting and classifying objects of fix shapes and sizes. Keller *et al.*[19] and Maldonado-Bascon *et al.*[20] used them for detecting as well as classifying traffic signs from images. Random Forest[21] and K Nearest Neighbor[22] can also be used as classification algorithms for traffic sign classification.[23] The advantages of using color- and shape-based methods are that they are computationally efficient and easy to implement. The disadvantages of these methods are that they are not illumination invariant and also fail when the color of the traffic sign fades. These methods will also fail when there is some occlusion in the image or signs have deformation. The second approach based on deep neural network was discovered to overcome these disadvantages.

The Deep learning-based approaches use CNN for end to end TSD and recognition. The traffic sign recognition approaches using CNN have touched almost human level performance in the constrained scenarios. Dan Ciresan *et al.*[24] used a multi-column CNN[25] for traffic sign recognition. Vaidya *et al.*[26] proposed spatial and color transformation network with CNN to make traffic sign invariant to color and affine transformation. Sermanet *et al.*[27] proposed a multi-scale CNN architecture[28] which helped in classifying traffic signs of multiple sizes and shapes. The recognition approach for traffic sign is easy compared to detection as no other object or sign is present in the image so CNN-based approaches have achieved great results.

CNN can also be used for TSD. The first major breakthrough in object detection using deep learning was achieved by R-CNN.[29] Though it improved accuracy drastically but it had many drawbacks. It used selective search[30] and Edge boxes[31] for making region proposals, CNN for extracting features from the proposed regions and then SVM for recognition. The R-CNN[29] required separate training for CNN and SVM.[18] It also required multiple pass through CNN for all regions which slowed the detection performance. The SPP-Net[32] proposed speeding up this process by calculating features for all regions simultaneously but still it required multiple training. The further improvement was done in Fast R-CNN[33] which replaced SVM by softmax classification layer. It still required selective search for region proposals which made the detection process slower. This problem was solved in Faster R-CNN[34] which used Region proposal Network instead of selective search which allowed the network to be trained end to end. This family of methods treated region proposals and classification as two different parts.

These parts were merged into one by two methods called You Only Look Once (YOLO)[35] and Single Shot Detector (SSD).[36] Both the algorithms detected the objects in a single pass through the network which improved the speed of the network. They both divided the image in rectangular grid or default bounding boxes and then used CNN for classification as well as regression. The YOLO used a single

prediction layer which made it faster but less accurate.[17] The SSD[36] used multiple feature maps at different scales for prediction which made it more accurate. Both SSD and YOLO were less accurate for detecting small objects. Fu *et al.*[37] proposed Deconvolutional SSD (DSSD) which fused the feature maps of earlier layer with later layers for more accurate detection of small objects. This improved accuracy is at the cost of speed. It was observed from the literature that most of the architectures require high computing resources and memory which makes them hard to deploy on edge platforms with limited resources for real-time performance. SSD and YOLO are less computationally complex but they process images with smaller resolution which makes their performance poorer on small objects. These are the main challenges overcome by using the proposed architecture.

3. Proposed Architecture

The proposed CNN architecture resembles more to the SSD[36] family of architectures than R-CNN family.[20–25] The motivation behind using SSD family is that it will require a single pass through the network which will result in less computation time and less memory resources which serves the requirement for embedded implementation. The complexity of detecting small traffic signs from high-resolution images in various illumination conditions with reduced computational complexity motivated the design of the proposed architecture. The proposed TSD architecture is shown in Fig. 1.

The proposed architecture uses feature pyramids of last four convolutional layers for detection and classification. It uses CTN[38] for making the architecture invariant to illumination and change in the lighting conditions at night. It was observed that SSD architectures do not provide good results for small objects.[36] The reason behind is that as we go deeper into the architecture, each feature in the feature map corresponds to large receptive field in the input image which cannot represent small objects. The traffic signs in the datasets used are from sizes 16×16 to 128×128 in the image sizes of 1360×800 or 2048×2048.[39,47] If we resize the image, then the sign dimensions will get even smaller which will make it very hard to detect. The proposed architecture can process the image without resizing. The choice of number of convolution layers is done based on the receptive field

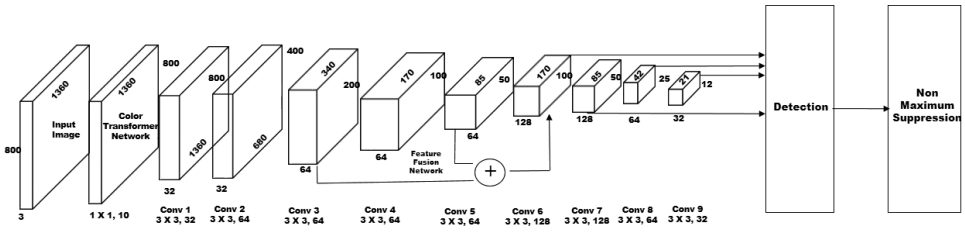


Fig. 1. Proposed CNN architecture.

calculation which will be explained later. The architecture also uses feature fusion module[40] to further boost the performance of the architecture on small objects. The architecture first measures performance using classical cross entropy loss with hard negative mining[36] for avoiding class imbalance. The performance is also measured using focal loss[43] which can effectively reduce the effect of class imbalance. The proposed architecture is targeted towards embedded platform so how the architecture fairs in terms of memory and computational cost is also of paramount importance. The choice of layers and trainable parameters is done by keeping this final goal in mind. The concepts used in the architecture are explained in detail in what follows.

3.1. *Color transformer network (CTN)*

Generally, images are captured in RGB format which is very sensitive to changes in illumination and lighting conditions. Many researchers have converted RGB images to HSV or YCbCr for making the system invariant to color.[10,12] This requires trial and error for deciding on which color space is best for the given problem. To avoid this explicit conversion, the proposed architecture incorporates the learnable module using 1×1 convolution which automatically learns the color channel conversion from input RGB color image to the new color space. This inclusion improves the performance of the architecture in various lighting conditions. The learned conversion will be based on the data provided in the training set so it should incorporate the lighting conditions encountered after deployment.

3.2. *Computations in convolutional neural network*

The main limitation of deploying CNN architecture on embedded platforms is the amount of computation it requires. CNN consists of convolution and pooling layers arranged in sequential fashion for feature extraction. The proposed architecture uses Batch Normalization after every convolution layer and before pooling layer for normalizing the activations which helps in faster convergence.[41] Exponential Linear Unit (ELU)[42] is used as an activation function which is explained in the later section.

The main computation in CNN happens inside the convolution layer. If c is the kernel index, f is the kernel, b_c is the bias term for each kenel, C_i represents the total number of input channels, I_c represent the input feature map and O_c represents the output feature map after convolution, then Convolution operation can be summarized as follows:

$$O_c = \sum_{c=1}^{C_i} (I_c * f_c, c) + b_c. \tag{1}$$

This formula is used for computing convolution operation at each layer. It basically computes dot product between input feature map and kernel at each location.

Table 1. Notations used for parameters of CNN.

Notation	Description
n_H	Height of output feature map
n_W	Width of output feature map
n_i	Number of channels in input feature map
n_o	Number of channels in output feature map
f_H	Height of the kernel
f_W	Width of the kernel

3.2.1. *Computational cost calculation*

The notations used for calculating computational cost are summarized in Table 1.

Given the notation specified in Table 1 and if the stride used for Convolution is one then the total number of Multiply and Accumulate (MAC) operations per layer can be computed using the following formula:

$$\text{MAC} = f_H * f_w * n_H * n_W * n_i * n_o. \tag{2}$$

Sometimes floating points operations per second (FLOPs) is used instead of MAC for calculating the computational cost of the CNN architecture. The number of trainable parameters at each layer can be calculated as multiplication of kernel size at each layer and number of kernels applied. This can be summarized as follows:

$$P = f_H * f_w * n_i * n_o. \tag{3}$$

The computational efficiency of a CNN architecture can be measured by how much computation is done per each parameter. This parameter can be called MAC per parameter (MPP) which should be high for a computationally efficient architecture.

$$\text{MPP} = n_H * n_w. \tag{4}$$

The high value of MPP indicates that we are doing more computations per parameter which results in effective utilization of each parameter. The value of MPP is 1 for fully connected layers which is not so good in terms of computational efficiency so we are not using any fully connected layer in the proposed architecture. Normally, the calculation in CNN is performed in batches so if the batch size is B then the MPP value will also be multiplied by B. If we increase the batch size, then amount of memory required for storing the intermediate results will also be high so there should be a trade-off between these parameters.

3.2.2. *Receptive field calculation*

Receptive field is the region in the input image where a particular feature at a deeper layer is looking at. It is very important in choosing the size of network as deeper layer will have a very large receptive field which will not be helpful in detecting small objects. The receptive field calculation is also helpful in choosing the right scales for default bounding boxes in each feature map. It is also important to note that not all points in the receptive field are important and their importance is decreased

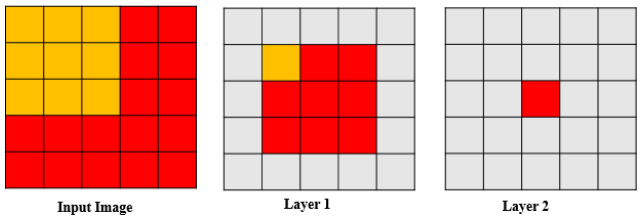


Fig. 2. Receptive field size.

exponentially as we move away from the center. This can be observed from Fig. 2, where receptive field is shown after applying 3×3 convolution for two times on the input feature map. The receptive field is indicated by the same color in the figure.

The receptive field calculations can be done using the following equations. The gap between two features or jump is represented by j, filter size is represented as f, stride is represented by s and padding is represented by p.

$$\text{Size of output feature map: } n_o = \left(\frac{n_i + 2p - f}{s}\right) + 1. \tag{5}$$

$$\text{Jump in output feature map: } j_o = j_i * s. \tag{6}$$

$$\text{Receptive field size: } r_o = r_i + (f-1) * j_i. \tag{7}$$

$$\text{Center of the receptive field: } r_{co} = r_{ci} + \left(\frac{f-1}{2} - p\right) * j_i. \tag{8}$$

The initial center is taken as the left most pixel of the input image and then the receptive fields are calculated using above formulas recursively. If width and height are different, then the above calculation is done separately for both the dimensions. Normally, smaller dimension out of two is taken for calculation. The size of receptive field is calculated for all convolutional layers. This calculated receptive field size and size of traffic signs in the dataset are used in the proposed work to determine that the convolution layers 6–9 can be chosen for detection with appropriate scales of default bounding boxes. These parameters are also used to determine fusion between convolution3 and convolution5 layers. Three aspect ratios, 0.5, 1 and 2, are chosen for default bounding boxes with two boxes at aspect ratio 1 because most of the traffic signs have square shape.

3.3. *Feature fusion network*

It is known that features in lower layers in the CNN architectures contain rich detail information which is useful for detecting small objects from an image while later layers detect advanced semantic features which are useful in detecting large objects. If somehow both of these features can be concatenated, then we can enhance the performance of the CNN architecture in object detection of various sizes.[40] It was observed that features of convolution3 layer in the architecture contain rich features

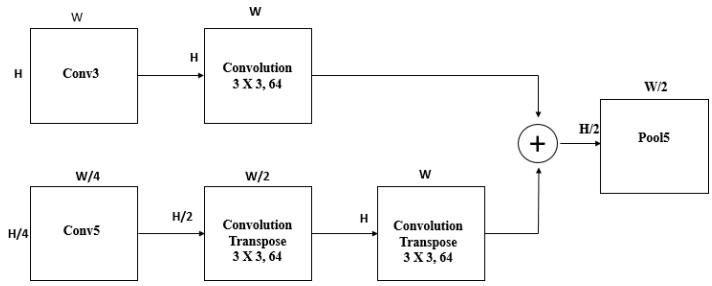


Fig. 3. Feature fusion network.

for detecting objects with smaller than 0.01% size with respect to original image so these features were aggregated with convolution5 layer features for detecting small traffic signs efficiently. The proposed architecture uses feature fusion network, as shown in Fig. 3.

There are two ways in which features can be fused. The first method uses concatenation along the channel dimension while the second method uses addition of two feature maps. These methods require height and width of two feature maps to be same which requires deconvolution37 or up-sampling of the features from a lower dimension feature map to higher dimension feature map. Transpose convolution layer is used for up-sampling in the proposed architecture with concatenation method for fusion which will keep features of both layers intact. Max pooling of the final fused layer is performed to reduce the computation and number of default bounding boxes for detection.

3.4. *Activation functions*

The proposed architecture uses ELU[42] as an activation function for all layer except the final classification layer which uses softmax as an activation function. The ELU is very similar to ReLU activation function but it treats negative activations efficiently compared to RelU which just truncates it to zero. The equation of ELU is given as follows:

$$\begin{aligned} F(x) &= x \quad \text{if } x > 0, \\ &= \alpha * (e^x - 1) \quad \text{if } x < 0, \end{aligned} \tag{9}$$

where α is a hyper parameter which was chosen as 0.5 after experimenting with different values and seeing the convergence performance. The Softmax activation function gives the class probability of each class. If there are C classes and a_i is the activation for ith class, then the softmax activation function is given as follows:

$$F(a_i) = \frac{e^{a_i}}{\sum_C e^{a_i}}. \tag{10}$$

3.5. *Loss function*

The loss function for optimization will consist of two parts, which are localization loss and classification loss. Smooth L1 loss[36] is used as a loss function for localization. Two scenarios are checked for classification with first one using classical crossentropy or softmax loss and online hard negative mining while second one uses focal loss.[43] The overall loss is the summation of localization and classification loss.

Original SSD method uses Softmax loss function for classification which is given as follows:

$$L(w) = -\frac{1}{B}\sum_{i} \log F(\alpha_i), \tag{11}$$

where F is a Softmax activation function. The problem in using softmax is that it is affected by class imbalance because of the large amount of negative examples in comparison to positive examples. To avoid that hard negative mining with negative to positive ratio of 2 is used where only first two negative examples according to confidence score are used to calculate loss function. The SSD paper uses 3 as a ratio but it was observed that loss function did not converge well as loss function was dominated by loss from the negative images.

The focal loss[43] avoids the class imbalance by tunable focusing parameter. The focal loss function is given as follows:

$$FL(\alpha_i) = -(1-\alpha_i)^{\gamma} \log \alpha_i, \tag{12}$$

where γ is the focusing parameter which helps in down weighting the easy example and if $\gamma = 0$ then FL will be similar to softmax loss. The value of γ is taken as 2 for the proposed paper which is same as proposed in the original paper.[43]

3.6. *Jetson nano GPU embedded development board for edge deployment*

There are two paradigms in deploying deep learning architectures for real-time applications. One uses cloud for all the computations where all the video data is transmitted to the cloud and decision is transmitted back. This requires large network bandwidth and security of the data is also a concern. So, there is a trend in moving towards a second paradigm which is deploying on the edge platforms where all the computation is performed on the device itself and there is no need for network connection. It also allows the use of other sensor devices connected to the board itself.

There are many low power edge deployment boards like Jetson Nano, Jetson TX1, Jetson TX2, Google Coral, etc. available in the market for edge deployment prototyping. Jetson TX1 and Jetson TX2 have large computational power but they are very costly. Google coral needs a host system and only allows Tensorflow Lite model for deployment while Jetson Nano can work standalone so it is used for deploying the proposed architecture. Jetson Nano is also cost efficient which will make it easy to deploy the system on vehicles with low cost. It comprises of 128 core Maxwell GPU with Quad core ARM CPU running at 1.43 Ghz. The RAM capacity is

Table 2. Training setup for GTSDB.

Parameter	Value
Input image size	$800 \times 1360 \times 3$
Batch size	2
Number of epochs	150
Learning rate	0.005

4 GB and it can run from SD Card. It has a performance of 472 GFLOPs and has USB as well as CSI ports for connecting camera with the device. The proposed model is converted to TensorRT for faster inference on the device.[44]

4. Implementation and Results

The proposed architecture is implemented using TensorFlow and Keras Library in Python. It is implemented on a Laptop system with 8 GB of RAM and Intel i5 processor running at 2.5 GHz. The system also has GeForce 940 NVIDIA GPU which can leverage the parallel computation in deep learning architecture. The implementation uses GTSDB[39] and Tsinghua-Tencent 100K[47] for evaluating the performance of the architecture. The architecture is also deployed on Jetson Nano for real-time inference. The training setup for the architecture is shown in Table 2.

The training of the proposed architecture required choosing appropriate optimization algorithm for faster convergence. It was found in the experimentation that Stochastic gradient descent was very slow to converge while NADAM optimizer zigzagged around the minima[26] but Adaptive Moment Estimation (ADAM)[45] algorithm provided faster convergence.[26] ADAM uses exponentially decaying average of past gradients for updating weights which is given as follows:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t, \tag{13}$$

where g_t is the current gradient and $\beta_1$is the hyperprameter which is taken as 0.9. ADAM also uses average of past squared gradients which can be calculated as follows:

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2, \tag{14}$$

where β_2 is the hyperprameter which is taken as 0.90. Bias correction $\hat{m}_t = \frac{m_t}{1-\beta_1^t}$ and $\hat{v}_t = \frac{v_t}{1-\beta_2^t}$ is applied for initial average calculation and then weights are updated using the following formula:

$$W_{t+1} = W_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t, \tag{15}$$

where η is the learning rate and ε is a very small constant taken to avoid division by zero.

4.1. *Results on GTSDB dataset*

The GSTDB[39] dataset consists of images with traffic signs belonging to 43 classes. The traffic signs in the dataset are very small in size ($< 1\%$) compared to the original image size. The images also have varied illumination and transformation. This makes the detection task even more difficult. Total 739 images from GSTDB[39] dataset were used for training and 131 images were used for testing. The results obtained after training using ADAM optimizer for 150 epochs are shown in Fig. 4.

The performance of the proposed architecture is measured in terms of Mean Average Precision (mAP) on classes with more than 50 instances in training set. The performance is measured for small ($< 0.01\%$ of image size), medium (0.01–0.05% of image size) and large object sizes ($> 0.05\%$ of image size) from the test dataset. The mAP achieved is 96.13% for large, 85.68% for medium and 73.03% for small objects. The performance of the proposed algorithm on GTSDB39 is compared with other algorithms in Fig. 5. The results for other algorithms are taken from Arcos-Garcia *et al.*[46]

The network performs very well on large objects and its performance is equivalent or partially better compared to faster R-CNN. The addition of CTN adds around 1% to mAP value. The addition of feature fusion network adds 4% to mAP value for smaller objects which makes the proposed architecture best performer in smaller objects. It is also observed that use of focal loss for classification improves the mAP.

Fig. 4. Proposed results for TSD and recognition on GTSDB.

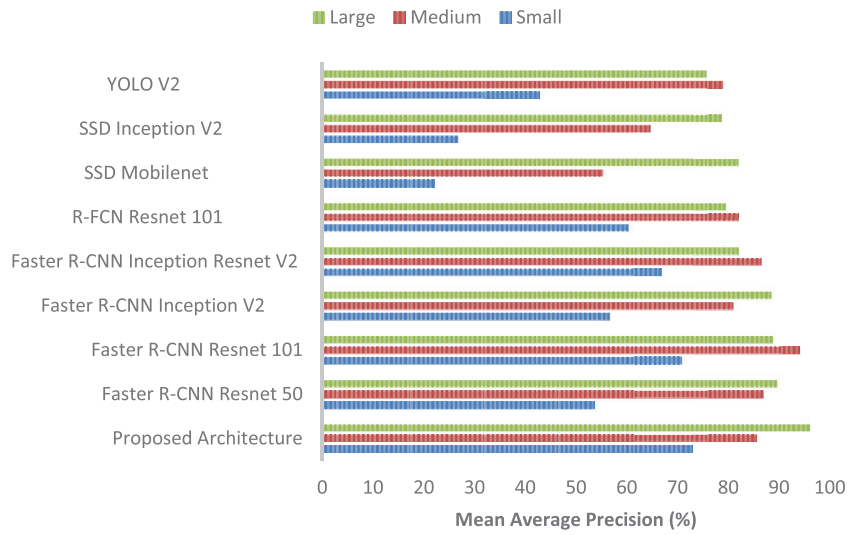


Fig. 5. mAP vs image size for different architectures on GTSDB dataset.

4.2. *Results on Tsinghua-Tencent 100K dataset*

The performance of the proposed architecture is also measured on Tsinghua-Tencent 100K dataset which is even more challenging and contains more images compared to

Fig. 6. Proposed results for TSD and recognition on Tsinghua-Tencent 100K dataset.

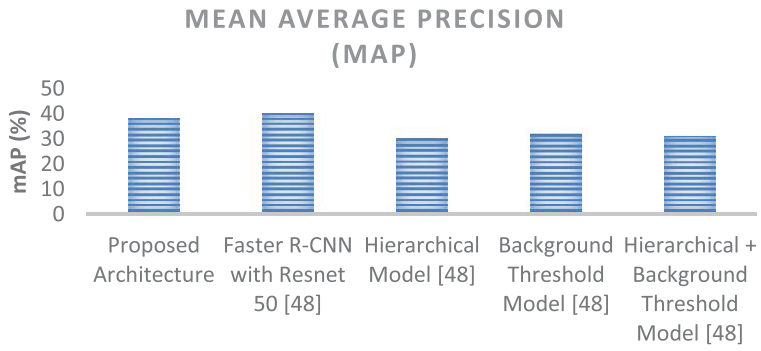


Fig. 7. mAP comparison of different architectures.

GTSDB dataset. The detection results after training the architecture on Tsinghua-Tencent 100K dataset for 150 epochs using ADAM optimizer are shown in Fig. 6.

The performance of the proposed architecture is measured in terms of mAP. The performance of the proposed algorithm on Tsinghua-Tencent 100K is compared with other algorithms in Fig. 7.[48]

The proposed architecture gives 38% mAP value which is slightly less than the performance given by Faster R-CNN because of its complex ResNet 50 feature extractor. This architecture is designed to be more computationally efficient compared to other architectures which will be shown in the section in what follows.

4.3. *Computational efficiency of the proposed architecture*

The architecture is targeted towards embedded platforms for edge deployment so the architecture is implemented on Jetson Nano which is an edge deployment board for performance evaluation. The performance of architecture in terms of forward

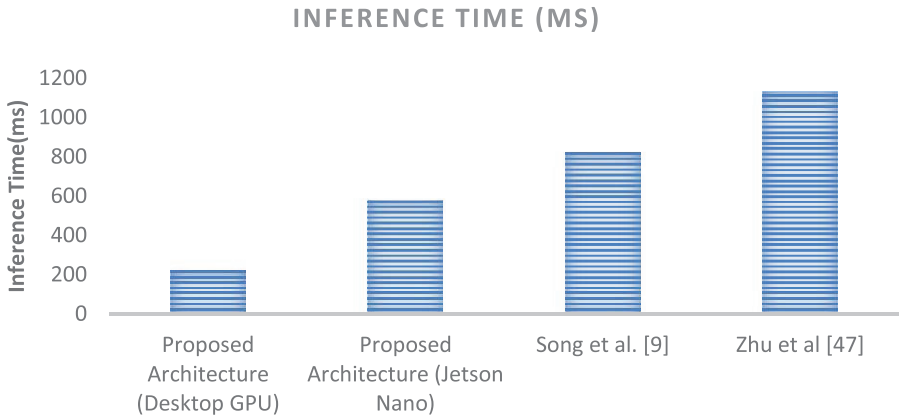


Fig. 8. Running time for different architectures.

Table 3. Computational performance of different architectures.

	Parameters	FLOPS (Giga)
Proposed architecture	945088	0.042
Faster R-CNN Resnet 50[46]	43337242	533
Faster R-CNN Resnet 101[46]	62381593	625
Faster R-CNN Inception V2[46]	12891249	120
Faster R-CNN Inception Resnet V2[46]	59412281	1837
R-FCN Resnet 101[46]	64594585	269
SSD Mobilenet[46]	5572809	2.3
SSD Inception V2[46]	13474849	7.59
YOLO V2[46]	50588958	62.78

inference time is compared with other architectures designed with the same objectives in mind which is shown in Fig. 8.

The proposed architecture performs far better than other similar architectures both on Desktop GPU Nvidia GTX 940 with running time of 220 ms and on Jetson Nano with 578 ms. The comparison in terms of model size or memory requirement is shown in Fig. 8.

The proposed architecture only requires 11 MB for storing model and its parameters which is 10 times better than other architectures. The system's performance is compared in terms of number of trainable parameters and FLOPs, which is shown in Table 3.

The architecture has one sixth parameters than the best performing architecture and 50× less FLOPs.

The system is deployed on Jetson Nano development board. It takes around 578 ms to detect traffic signs in Jetson Nano which is still 1.5 times faster than other similar architectures implemented earlier. The implementation results on Jetson Nano are shown in Fig. 10.

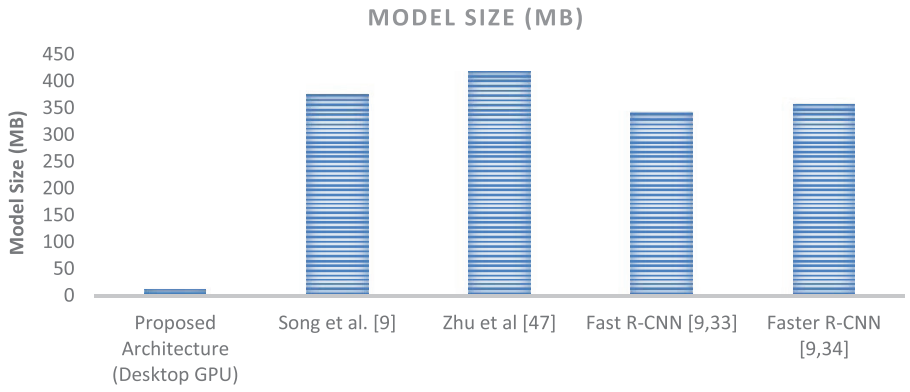


Fig. 9. Memory requirement for different architectures.

Fig. 10. Proposed TSD results on Jetson Nano.

5. Conclusion and Future Work

TSD is a vital ingredient for any driver assistant or autonomous driving system. Traffic signs occupy a very small portion of the entire image which makes the detection task difficult. This paper proposed a deep learning architecture for TSD which can be deployed on embedded platforms with limited memory and computational resources without sacrificing performance on detection accuracy. It allowed processing of the high resolution images without any resizing which preserves the dimensions of traffic signs. It used trainable CTN for making the system invariant to illumination changes and feature fusion network for detecting traffic signs for various sizes. The inclusion of feature fusion network improved the accuracy by 4% on small objects. The proposed architecture used one sixth parameters and fifty times less FLOPS. The memory footprint of the model was also very less compared to other architectures. The paper covered the deployment of the model on Jetson Nano Embedded platform. The performance on Jetson Nano was also faster compared to earlier architecture proposed for the same problem. The performance of the architecture in terms of detection accuracy can be further improved in future particularly for Tsinghua-Tencent 100K dataset. The inference time still needs an improvement as this architecture falls short of achieving real-time performance.

References

1. U. Kamal, T. I. Tonmoy, S. Das and M. K. Hasan, "Automatic traffic sign detection and recognition using SegU-Net and a modified Tversky loss function with L1-constraint," *IEEE Trans. Intel. Transport. Syst.*, 1467–1479 (2019).

2. D. Tabernik and D. Skočaj, "Deep learning for large-scale traffic-sign detection and recognition," *IEEE Trans. Intel. Transport. Syst.* **21**(4), 1427–1440 (2019).
3. J. Jin, K. Fu and C. Zhang, "Traffic sign recognition with hinge loss trained convolutional neural networks," *IEEE Trans. Intel. Transport. Syst.* **15**(5), 1991–2000 (2014).
4. J. M. Lillo-Castellano, I. Mora-Jiménez, C. Figuera-Pozuelo and J. L. Rojo-Álvarez, "Traffic sign segmentation and classification using statistical learning methods," *Neurocomputing* **153**, 286–299 (2015).
5. M. Haloi, "A novel plsa based traffic signs classification system," arXiv preprint arXiv:1503.06643, 2015.
6. Y. Zhu, C. Zhang, D. Zhou, X. Wang, X. Bai and W. Liu, "Traffic sign detection and recognition using fully convolutional network guided proposals," *Neurocomputing* **214**, 758-766 (2016).
7. V. Balali and Golparvar-Fard, "Evaluation of multiclass traffic sign detection and classification methods for US roadway asset inventory management," *J. Comput. Civil Eng.* **30**(2), 04015022 (2016).
8. A. Alam and Z. A. Jaffery, "Indian Traffic Sign Detection and Recognition. *Int. J. Intel. Transport. Syst. Res.* **18**, 98–112 (2020).
9. S. Song, Z. Que, J. Hou, S. Du and Y. Song, "An efficient convolutional neural network for small traffic sign detection," *J. Syst. Architect.* **97**, 269–277 (2019).
10. W. G. Shadeed, D. I. Abu-Al-Nadi and M. J. Mismar, "Road traffic sign detection in color images," in *Proc. 2003 10th IEEE Int. Conf. Electron. Circ. Syst.*, Vol. 2, 2003, pp. 890–893.
11. H. Li, F. Sun, L. Liu and L. Wang, "A novel traffic sign detection method via color segmentation and robust shape matching," *Neurocomputing* **169**, 77–88 (2015).
12. A. De La Escalera, L. E. Moreno, M. A. Salichs and J. M. Armingol, "Road traffic sign detection and classification," *IEEE Trans. Ind. Electron* **44**(6), 848–859 (1997).
13. D. H. Ballard, "Generalizing the Hough transform to detect arbitrary shapes," *Read. Comput. Vision*, 1987, pp. 714–725.
14. G. Loy, N. Barnes, D. Shaw and A. Robles-Kelly, "Regular polygon detection," in *Proc. 10th IEEE Int. Conf. Comput. Vision*, Vol. 1, 2005, pp. 778–785.
15. S. K. Berkaya, H. Gunduz, O. Ozsen, C. Akinlar and S. Gunal, "On circular traffic sign detection and recognition," *Expert Syst. Appl.* **48**, 67–75 (2016).
16. C. Akinlar and C. Topal, "EDCircles: A real-time circle detector with a false detection control," *Pattern Recogn.* **46**(3), 725–740 (2013).
17. N. He, J. Cao and L. Song, "Scale space histogram of oriented gradients for human detection," in *2008 Int. Symp. Inform. Sci. Eng.*, Vol. 2, 2008, pp. 167–170.
18. L. Wang, *Support Vector Machines: Theory and Applications* (Springer Science & Business Media), Vol. 177.
19. C. G. Keller, C. Sprunk, C. Bahlmann, J. Giebel and G. Baratoff, "Real-time recognition of US speed sign," *IEEE Intel. Vehicles Symp.* **8**(2), 518–523 (2008).
20. S. Maldonado-Bascón, S. Lafuente-Arroyo, P. Gil-Jimenez, H. Gómez-Moreno and F. López-Ferreras, "Road-sign detection and recognition based on support vector machines," *IEEE Trans. Intel. Transport. Syst.*, 2007, pp. 264–278.
21. M. Pal, "Random forest classifier for remote sensing classification," *Int. J. Remote Sens.* **26**(1), 217–222 (2007).
22. J. M. Keller, M. R. Gray and J. A. Givens, "A fuzzy k-nearest neighbor algorithm," *IEEE Trans. Syst. Man Cybern.* **15**(4), 580–585 (1985).
23. A. Ruta, Y. Li and X. Liu, "Real-time traffic sign recognition from video by class-specific discriminative features," *Pattern Recogn.* **43**(1), 416–430 (2010).

24. D. CireşAn, U. Meier, J. Masci and J. Schmidhuber, "Multi-column deep neural network for traffic sign classification," *Neural Netw.* **32**, 333–338 (2012).
25. Y. Zhang, D. Zhou, S. Chen, S. Gao and Y. Ma, "Single-image crowd counting via multi-column convolutional neural network," in *Proc. IEEE Conf. Comput. Vision pattern Recogn.*, 2016, pp. 589–597.
26. B. Vaidya, C. Paunwala, "Traffic Sign Recognition using Color and Spatial Transformer Network on GPU Embedded Development Board," in *Int. Conf. Computer Vision Image Processing (CVIP – 2019)* IAPR (Int. Association of Pattern Recognition), Scopus Indexed - Springer, Communications in Computer and Information Science series (CCIS), (2019), pp. 82–93.
27. P. Sermanet and Y. LeCun, "Traffic sign recognition with multi-scale convolutional networks," in *2011 Int. Joint Conf. Neural Netw.*, 2011, pp. 2809–2813.
28. Z. Cai, Q. Fan, R. S. Feris and N. Vasconcelos, "A unified multi-scale deep convolutional neural network for fast object detection," in *Eur. Conf. Comput. Vision*, 2016, pp. 354–370.
29. R. Girshick, J. Donahue, T. Darrell and J. Malik, "Region-based convolutional networks for accurate object detection and segmentation," *IEEE Trans. Pattern Anal. Machine Intel.* **38**, 142–158 (2015).
30. J. R. Uijlings, K. E. Van De Sande, T. Gevers and A. W. Smeulders, "Selective search for object recognition," *Int. J. Comput. Vision* **104**, 154–171 (2013).
31. C. L. Zitnick and P. Dollár, "Edge boxes: Locating object proposals from edges," in *Eur. Conf. Comput. Vision*, 2014, pp. 391–405.
32. K. He, X. Zhang, S. Ren and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," *IEEE Trans. Pattern Anal. Mach. Intel.* **37**, 1904–1916 (2015).
33. R. Girshick, "Fast r-cnn," in *Proc. IEEE Int. Conf. Comput. Vision*, 2015, pp. 1440–1448.
34. S. Ren, K. He, R. Girshick and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *Adv. Neural Inform. Proces. Syst.*, 2015, pp. 91–99.
35. J. Redmon, S. Divvala, R. Girshick and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proc. IEEE Conf. Comput. Vision Pattern Recogn.*, 2016, pp. 779–788.
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. Comput. Vision*, 2016, pp. 21–37.
37. C. Y. Fu, W. Liu, A. Ranga, A. Tyagi and A. C. Berg, "Dssd: Deconvolutional single shot detector," *arXiv preprint arXiv:1701.06659*, (2017).
38. D. Mishkin, N. Sergievskiy and J. Matas, "Systematic evaluation of convolution neural network advances on the imagenet," *Comput. Vision Image Understand.* **161**, 11–19 (2017).
39. German Traffic Sign Detection Dataset, "http://benchmark.ini.rub.de/?section=gtsdb&subsection=news".
40. J. Leng and Y. Liu, "An enhanced SSD with feature fusion and visual reasoning for object detection," *Neural Comput. Appl.* **31**, 6549–6558 (2019).
41. S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, (2015).
42. D. A. Clevert, T. Unterthiner and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (elus)," *arXiv preprint arXiv:1511.07289*, (2015).
43. T. Y. Lin, P. Goyal, R. Girshick, K. He and P. Dollár, "Focal loss for dense object detection," in *Proc. IEEE Int. Conf. Comput. Vision*, 2017, pp. 2980–2988.
44. Jetson Nano Development Board Help Document, https://developer.nvidia.com/embedded/learn/get-started-jetson-nano-devkit.

45. D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, (2014).
46. A. Arcos-Garcia, J. A. Alvarez-Garcia and L. M. Soria-Morillo, "Evaluation of deep neural networks for traffic sign detection systems," *Neurocomputing*, 2018, pp. 332–344.
47. Z. Zhu, D. Liang, S. Zhang, X. Huang, B. Li and S. Hu, "Traffic-sign detection and classification in the wild," in *Proc. IEEE Conf. Comput. Vision Pattern Recogn.*, 2016, pp. 2110–2118.
48. A. Pon, O. Adrienko, A. Harakeh and S. L. Waslander, "A hierarchical deep architecture and mini-batch selection method for joint traffic sign and light detection," in *2018 15th Conf. Comput. Robot Vision (CRV)*, 2018, pp. 102–109.

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

Chirag Paunwala received his B.E. degree in Electronics and Communication from the South Gujarat University, Surat, in 1999 and ME Degree in Digital Techniques from the Rajiv Gandhi Technological University in 2006. He completed his Ph.D. degree from SVNIT, Surat in 2012. He is serving as 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.

Comparative Analysis Of Motion Based And Feature Based Algorithms For Object Detection And Tracking

Bhaumik Vaidya
Research Scholar,
GTU, India

Chirag Paunwala
Professor, EC Department,
SCET, Surat, India

Abstract—**Object detection and tracking in the video sequence is a challenging task and time consuming process. Intrinsic factors like pose, appearance, variation in scale and extrinsic factors like variation in illumination, occlusion and clutter are major factors effecting this task. The main aim of this work is to implement and compare different algorithms in challenging conditions and find the algorithm that performs very efficiently on real time videos. In this paper, two motion based algorithms Zivkovic Adaptive Gaussian Mixture Model (ADGMM) and Grimson Gaussian Mixture Models (GGMM) and two feature based algorithms Speeded up Robust features (SURF) and Haar Cascade are implemented. The comparison of these algorithms in real life challenges and application is done to find out suitable algorithm for a particular application.**

Keywords— Background substraction; background Modelling; Zivkovic Adaptive Gaussian Mixture Model; Grimson Gaussian Mixture Models; Haar Cascade; Speeded up robust features.

I. INTRODUCTION

Object identification and tracking is a widely explored subject in computer vision. Object identification and tracking is consequential and challenging task in large numbers of vision applications such as automatic vehicle or robot navigation and video surveillance, augmented reality, behavioral analysis applications, traffic control and monitoring etc.

Accurate object detection is the stepping stone for object tracking applications. It can be done by performing object detection in each frame or the frame in which the object appears for the first time. While going for object tracking, basic operation is to make separation of interested object called 'foreground' from 'background' [1]. Some work in literature has focused on developing algorithms for automatic detection and tracking which reduces the need of human surveillance. Most surveillance systems include static cameras and fixed backgrounds [3] that provide clues to object discovery in video via background subtraction techniques. The background representation model should be robust and adaptive because in some conditions background is not available or it is constantly changing [1].

For object tracking motion based and feature based approaches are widely used. Both techniques have their pros and cons. So in this paper both types of algorithms are implemented and compared in various challenging conditions. In motion based algorithms Zivkovic Adaptive Gaussian Mixture Model (ADGMM) [2] and Grimson Gaussian Mixture Model (GGMM) [4] are implemented whereas in feature based techniques Speeded up Robust features (SURF) [8] and Haar Cascade [9] are implemented.

The remaining paper is organized as follows: section II summarizes the related work done in this domain; section III describes motion based algorithm in detail; section IV describes feature based algorithm in detail; Section V summarizes experiments and results of the implemented algorithms.

II. RELATED WORK

For motion based approach modeling of background is a very important step. In basic background modeling techniques an averaging operation, median operation or a histogram analysis of frames is done over time [3]. Thresholding is used to distinguish between foreground and background in these techniques. The statistical models like Gaussian models [4] and support vector models [5] are more robust to illumination changes and for dynamic backgrounds [4]. Gaussian models uses pixel's intensity values over the period of time. A single model can't deal with the dynamic background conditions like waving trees and rippling water. To overcome this Mixture of Gaussians (MoG), Gaussian Mixture Model (GMM) or Adaptive Gaussian Mixture Model can be used [4].
The motion based algorithms may not work properly when object stops moving or movement is very slow as well as shadows are not removed properly [6]. For that different feature based techniques like Scale invariant feature transform (SIFT) [7], Speeded up robust features (SURF) [8], Haar Cascade [9], Harris [19] and SUSAN corner detectors, Features from Accelerated Segment test (FAST) [10] are used. Haar Casacde is very popular in human detection and tracking. SURF and SIFT are other very popular algorithms in terms of detection accuracy on different classes of objects in different challenging conditions.

These algorithms differ in terms of detection speed and computational complexity. Table 1 shows the comparison between different feature based algorithms.

Table 1 Comparison between different features based Algorithms
(+++ - Good Performance, ++ - Average Performance, + - Poor Performance)

Feature Detector	Corner	Rotation invariant	Scale invariant	Repeatability	Localization accuracy	Robustness	Efficiency	Computational Complexity
Harris [19]	√	√		+++	+++	+++	++	+++
Hessian [8]		√		++	++	++	+	+++
SUSAN [10]	√	√		++	++	++	+++	++
FAST [10]	√	√		+++	+++	+++	++	++
SIFT [7]	√	√	√	+++	+++	+++	+++	+
SURF [8]	√	√	√	++	++	++	+++	++
Haar [9]	√	√	√	+++	++	++	+++	+++

In Table 1, algorithms are compared in terms of whether the algorithm is rotation or scale invariant, repeatability of feature extracted for similar objects, localization of features in an object, robustness, detection efficiency and computational complexity. All the corner detectors like Harris, Hessian, SUSAN and FAST are computationally efficient but they are not scale invariant. It can be seen from the table that SURF and SIFT performs well in terms of all parameters. SIFT performs better than SURF in detection accuracy but it is computationally very complex and it is hard to use SIFT in real time application. The Haar cascade [9] has high speed of detection and detection accuracy because of many classifiers are arranged in cascaded structure. So in this paper SURF and Haar cascade which are less complex and more accurate is implemented for object detection and tracking.

III. MOTION BASED ALGORITHM FOR OBJECT TRACKING

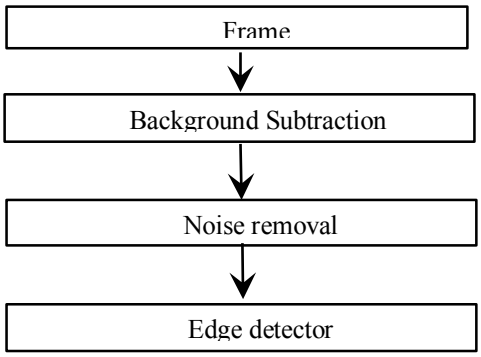


Fig 1 Flow chart for Motion based object tracking

The flow chart for motion based object tracking algorithm is shown in Fig 1 below. As shown in the flow chart, from input frame background is subtracted using Gaussian Mixture Model. In this paper two GMM algorithms ADGMM [2] and GGMM [4] are used.

In Grimsion method each pixel is a mixture of Gaussians. To calculate probability of observing the current pixel value model history of every pixel in mixture of K Gaussian distributions is used. It is given in Equation 1.

$$P(X_t) = \sum_{i=1}^{K} w_{i,t} * \eta(X_t, \mu_{i,t}, \Sigma_{i,t}) \quad (1)$$

In Equation 1, $w_{i,t}$ is an estimate that indicates which fraction of the data is used for in calculating ith Gaussian at time t, $\mu_{i,t}$ is the mean value, $\Sigma_{i,t}$ is the covariance matrix and where η is a Gaussian probability density function

$$\eta(X_t, \mu_{i,t}, \Sigma_{i,t}) = \frac{1}{(2\pi)^{\frac{n}{2}} |\Sigma|^{\frac{1}{2}}} e^{-\frac{1}{2}(X_t-\mu_t)^T \Sigma^{-1} (X_t-\mu_t)} \quad (2)$$

The value of K is chosen based on memory availability and computational power. [4].

In real life scenarios illumination can very gradually or suddenly in the scene. Objects in the scene can also change from time to time. In order to adapt to these kinds of changes Zivkovic ADGMM can be used.

In Zivkovic ADGMM model after reasonable Time period T training set is updated. For each new sample the training data set X_T is updated and probability $\hat{P}(\vec{x}|X_T, BG)$ is

reestimated. The estimate of foreground object in the recent history of samples is denoted as $\widehat{P}(\vec{x}|X_T, BG + FG)$, Where BG indicates background and FG indicates foreground.

$$\widehat{P}(\vec{x}|X_T, BG + FG) = \sum_{m=1}^{M} \widehat{\pi_m} \mathcal{N}(\vec{x}; \widehat{\vec{\mu_m}}, \widehat{\sigma_m^2} I) \quad (3)$$

In equation 3, μ_m indicates the estimation of the means and σ_M signifies estimation of variances. $\widehat{\pi_m}$ indicates mixing weights. This mixing weights must add up to one and they all should be positive. All parameters are regularly updated for all new data samples.

Methods such as GMM and AGMM perform well when the background is stable. These methods are best compromise between speed, simplicity and efficiency. To improve the efficiency of GMM post processing filter is implemented. It contains noise removal and edge detection blocks. For removing unwanted noise morphological opening which is erosion followed by dilation is applied [18]. To get a more accurate segmentation boundary, the contour of the object is extracted by using Canny edge pixel classification method [17].

IV. FEATURE BASED ALGORITHM FOR OBJECT TRACKING

The motion based algorithm may not work properly when object stops moving or movement is very slow. Also shadows are not removed properly [6]. For that feature based techniques are used.

A. Speeded up Robust Features (SURF)

SURF is most common and widely used feature based algorithm for object detection and tracking. SURF uses Hessian-matrix approximation for interest point detection which performs well in terms of accuracy [8]. SURF detects blob-like structure at locations with the maximum value of determinant. For reducing the computational complexity integral images are used. The computational time for SURF is independent from the size of image [8]. For making SURF descriptor invariant to image rotation, Haar-wavelet response is calculated in horizontal and vertical direction with the suitable radius according to scale around the interest point. Then wavelet response is computed and Gaussian weight is given centered at interested point. After that sum of all responses within the window $\pi/3$ is calculated and with use of this the dominant orientation is estimated. [8].

Fig 2 shows the flow chart for implemented SURF algorithm. As shown in Fig 2, key-points are identified and SURF descriptor is extracted from object to be detected and stored on the server side. On client side when video frame arrives, SURF key-points are extracted from frame and descriptor is found. These descriptor is matched with descriptor on server side using Fast Library for Approximate Nearest Neighbors (FLANN) [14].

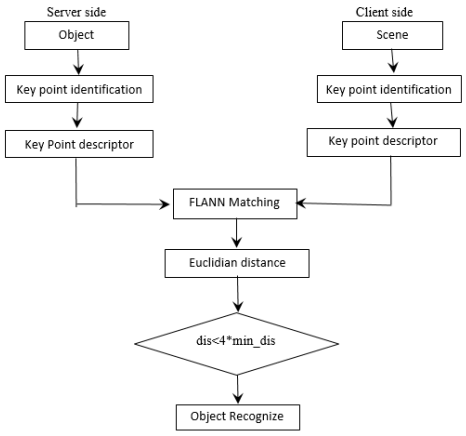


Fig 2 Flow Chart for SURF Algorithm

FLANN is a software library written in C++ for performing fast searches in high dimensional spaces. For FLANN based matching algorithm to be used for matching and the number of times the trees in the index should be recursively traversed needs to be specified. The higher values of trees will give better accuracy but it will also take more time. It gives displacement between keypoints as the output. If a distance between keypoints is less than some minimum threshold which is decided by experimentation object is recognized.

B. Haar Cascade:

Haar Cascade gives very good result for Human detection [9]. Haar uses Rectangular Features as shown in Figure. The pixels which lies within the white rectangles are subtracted from the pixels in black rectangles. The concept of integral image is used to speed up computation. Rectangular features of different sizes are shown in figure 3.

The rectangle features used in Haar Cascade provides very robust features for image representation which helps in effective learning from the features. The combination of integral image calculation and the efficiency of the rectangle feature set compensate their limited flexibility by reducing computation time [9].

Cascading of small classifiers achieves high performance in terms of detection accuracy and it greatly reduces computation time because non promising regions from the images are eliminated from early in the cascaded stage.

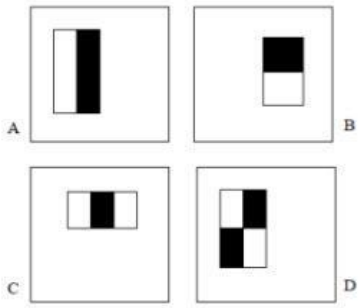


Fig 3 Rectangle features of different sizes used for Haar Cascade

The main reason for this is small boosted classifiers constructed rejects many of the negative sub-windows while detecting almost all positive instances.

The flow chart of implemented Haar Cascade is shown in Fig 4.

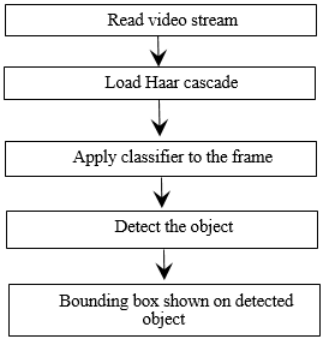


Fig 4 Flow chart for Haar cascade

V. EXPERIMENTS AND RESULTS

All the algorithms described above are implemented using OpenCV library and C++. The database used for measuring performance are PETS database [13] and POOL database [16]. The algorithms are also tested on real time videos taken using Webcam.

A. Comparison of Motion based Algorithm:

Results of ADGMM and GGM on a particular frame of each database is shown in Fig 5.

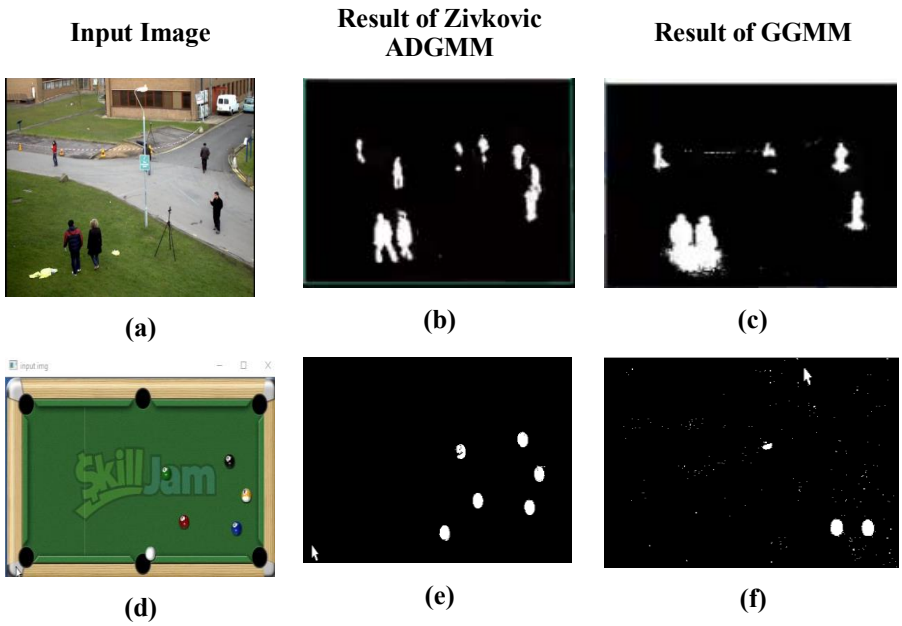


Fig 5 (a,b,c) ADGMM and GGMM method PETS2009 database (d, e,f) Result of ADGMM and GGMM method on Pool database

As can be seen from Fig 5 performance in terms of segmentation and Shadow removal is better in Zivkovic ADGMM than Grimson GMM. Table 2 shows the comparison between computation times of these algorithms.

Table 2 Analysis of two method's processing time (FPS) on particular frame of Real-time videos.

	PETS2009 DATABASE (FPS)	POOL_DATABASE (FPS)	Cup object(Home Scene) (FPS)	Remote object (Classroom Scene) (FPS)
Grimson GMM	5.5	3.7	6.1	6.2
Zivkovic AGMM	7.1	4.5	7	6.9

As can be seen from Table 2, in Zivkovic ADGMM processing time is reduced because of its computational simplicity. Fig. 6 indicates results for Grimson GMM after noise removal and edge classification using canny edge detector. After performing these steps segmentation results are comparable to Zivkovic ADGMM.

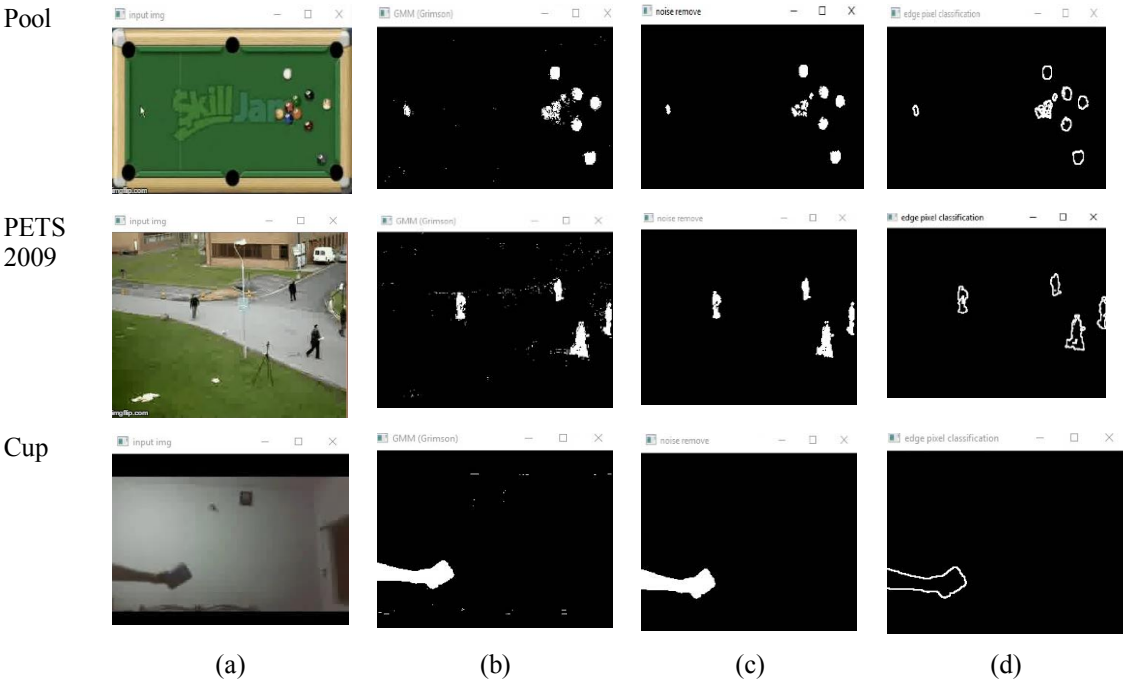


Fig 6 Motion based object tracking method (a) Input (b) Background subtraction (c) Noise Removal (d) Edge pixel Classification on various datasets

B. Result for Feature based Algorithms:

Fig 7 shown below indicates results of SURF algorithm on different objects and challenges.

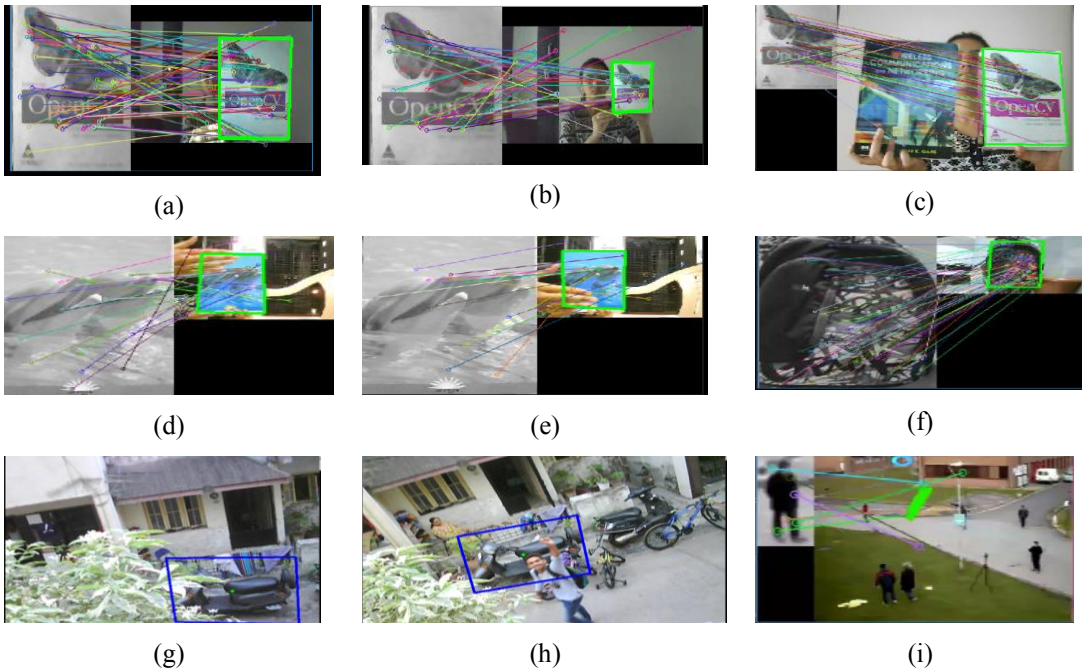
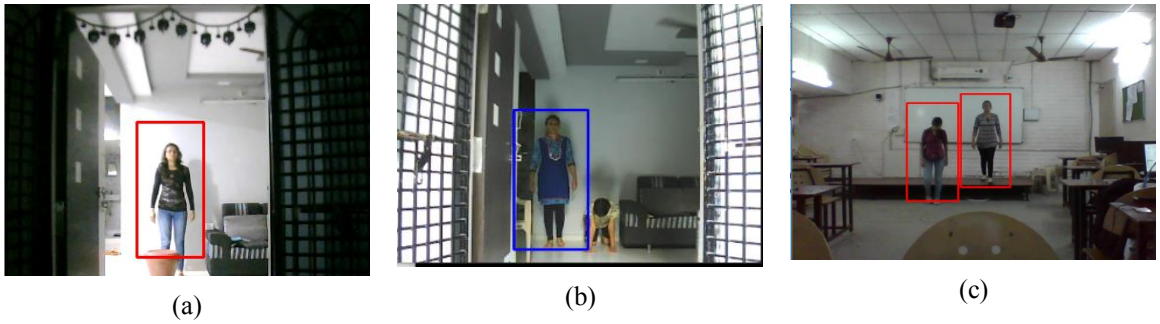


Fig 7 SURF algorithm in different challenges (a,b) different scaling (c) multiple object scenario (d,e) occlusion and crossing (f) shadow condition (g,h) occlusion (i) human detection

It can be seen from the results that SURF is rotation invariant, scale invariant and robust to occlusion and clutter. SURF was not good in Human detection whereas Haar Cascade gives good results in human detection. Fig 8 indicates results of Haar Cascade on different videos.



(a) (b) (c)

(d) (e)

Fig 8 Human detection using Haar Cascade (a) Single object (b) Multiple object but one with bending body (c) Multiple objects (d) Single object rotated (e) PETS2009 database from different videos under different conditions

C. Comparison between different Algorithms on Different datasets:

(a) (b) (c)

(d) (e)

Fig 9 Comparison of results obtained of all algorithms on PETS Database (a) Input Frame (b) GMMM (c) ADGMM (d) Haar Cascade (e) SURF

Fig 9 shows the comparison of all four algorithms implemented on one frame of PETS database. As can be seen from figure, in motion based approach ADGMM works better in terms of segmentation and shadow removal. SURF is not able to detect humans from the video whereas Haar Cascade is able to detect multiple objects which are at short distance (Approximate 4-10 meters) from camera.

Table 3 Timing comparison between algorithms on different databases

Database		Moving object detection			SURF algorithm			Haar cascade algorithm		
		Time (s)	frames	FPS	Time (s)	frames	FPS	Time (s)	frames	FPS
PETS2009 database	Grimson	38.3	398	12	30.21	398	13	21.26	398	18
	Zivkovic	39.1	398	10						
Real-time Video (Single Object)		20.50	269	13	19.49	269	14	9.98	269	26
Real-time Video (Multiple Object)		20.90	143	6	20.10	143	7	9.73	143	14

Table 3 shows the comparison of computation speed between algorithms in terms frames per second. Total processing time for a video and Number of frames in the video is also shown in the table. As can be seen that Haar Cascade out performs the other algorithms in terms of speed of processing.

VI. CONCLUSION

In this paper, motion based and feature based algorithms are implemented for object detection and tracking. These algorithms are compared in terms of speed of operation and detection accuracy in challenging environment. In motion based algorithm Zivkovic ADGMM is faster as compared to GGMM and give appropriate result for shadow removal and various speed of object movement. In feature based algorithm, SURF algorithm works well in different lighting condition, static and moving objects, rotation, scaling, occlusion and crossing. For human detection Haar cascade classifier results are better than SURF. Haar cascade algorithm works well in distance of 4 meter to 10 meter. Processing time for Haar cascade algorithm is also less compared to motion based algorithm and SURF algorithm.

VII. REFERENCES

[1] T. Bouwmans, "Traditional and recent approaches in background modeling for foreground detection: An overview," Science Direct journal on Computer Science Review, vol. 11–12, pp. 31-66, may 2014.

[2] Z. Zivkovic, "Improved adaptive Gaussian mixture model for background subtraction," Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on. Vol. 2. IEEE, 2004.

[3] T. Bouwmans, F. El-Baf and B. Vachon, "Statistical background modeling for foreground detection: A survey," Handbook of Pattern Recognition and Computer Vision, vol. 4, no. 2, p. 181–199, 2010.

[4] W. Grimson and C. Stauffer, "Adaptive background mixture models for real-time tracking," IEEE Conference on Computer Vision and Pattern Recognition, p. 246–252, 1999.

[5] H. Lin, T. Liu and J. Chuang, "A probabilistic SVM approach for background scene initialization," International Conference on Image Processing, p. 893–896, September 2002.

[6] P. Divyani, and H. J. Galiyawala. "A review on moving object detection and tracking." Int J Comput Appl 5.3 (2015): 168-175.

[7] D. Lowe, "Distinctive image features from scale-invariant keypoints," International journal of computer vision 60.2 (2004): 91-110.

[8] B. Herbert, T. Tuytelaars, and L. Gool, "Surf: Speeded up robust features," Computer vision–ECCV 2006 (2006): 404-417.

[9] L. Rainer, and J. Maydt, "An extended set of haar-like features for rapid object detection," Image Processing. 2002. Proceedings. 2002 International Conference on. Vol. 1. IEEE, 2002.

[10] R. Edward, and T. Drummond, "Machine learning for high-speed corner detection," Computer vision–ECCV 2006 (2006): 430-443.

[11] P. Massimo, "Background subtraction techniques: a review," Systems, man and cybernetics, 2004 IEEE international conference on. Vol. 4. IEEE, 2004.

[12] Z. Dongxiang, and H. Zhang, "Modified GMM background modelling and optical flow for detection of moving objects," Systems, Man and Cybernetics, 2005 IEEE International Conference on. Vol. 3. IEEE, 2005.

[13] PETS 2009 Benchmark Data, "http://www.cvg.reading.ac.uk/PETS2009/a.html"

[14] FLANN Library, "http://www.cs.ubc.ca/research/flann/"

[15] S. Guennouni, A. Ahaitouf, and A. Mansouri, "A Comparative Study of Multiple Object Detection Using Haar-Like Feature Selection and Local Binary Patterns in Several Platforms," Modelling and Simulation in Engineering, vol. 2015.

[16] POOL Database, "http://www.imageprocessingplace.com/root_files_V3/image_databases.htm"

[17] C. John, "A computational approach to edge detection." IEEE Transactions on pattern analysis and machine intelligence 6 (1986): 679-698.

[18] R. Gonzalez and R. E. Woods. "Digital image processing prentice hall." Upper Saddle River, NJ (2002).

[19] C. Harris and M. Stephens. "A combined corner and edge detector." Alvey vision conference. Vol. 15. No. 50. 1988.