A Survey of Computer Vision Research for Automotive Systems

Cheng-Yang Fu cyfu@cs.unc.edu
Chun-Kun Wang amos@cs.unc.edu
Eunbyung Park eunbyung@cs.unc.edu

Computer Science
University of North Carolina at Chapel Hill

May 5, 2015

1 Computer Vision Research and Benchmarks for Automonous Driving

1.1 Intuition

Current autonomous driving technology heavily relies on a special sensor, Velodyne Sensor, and a detailed precision map. From Darpa Grand Challenge in 2005 to current Google Car, Velodyne Lidar sensors are used widely. They could provide 360 degree real-time 3D distance map by illuminating environment with lasers and analyzing the reflect light. The 3D distance map can be used in obstacle detection and navigation. However, according to the marketing report, Velodyne LIDAR scanner used in Google Car costs 70,000 dollars which is much higher than the average budget people expect. Another constrain Google Car facing is a high definition inch-precision map. For example, the map should include the height of traffic light. That means Google Car can only drive when the precision map is provided. It limits usage of Google Car to only small areas. Although these two technologies provide important functions for autonomous driving, they also make universalizing autonomous car become impossible.

1.2 KITTI Vision Benchmark Suite

In 2012, a computer vision benchmark, the KITTI vision benchmark suite[13], is created to encourage computer vision research communities to develop algorithms to reduce the reliance on Velodyne Lidar sensors and detailed maps. The benchmark includes stereo, optical flow, visual odometry, 3D object detection and 3D tracking. To achieve this goal, they equipped a standard station wagon with two high-resolution color and grayscale video cameras. Figure 1 shows the recording platform with sensors. Accurate 3D information and position used as ground truth are provided by a Velodyne laser scanner and a GPS localization system. Besides, providing the raw data, KITTI suite provides preprocessed data as benchmarks for each task. Evaluation metric and website are also included to let users have same evaluation standard and a platform to easily compare their results. Compared with other benchmarks, the KITTI provides more realistic data recorded in real-world instead of in laboratory environment.

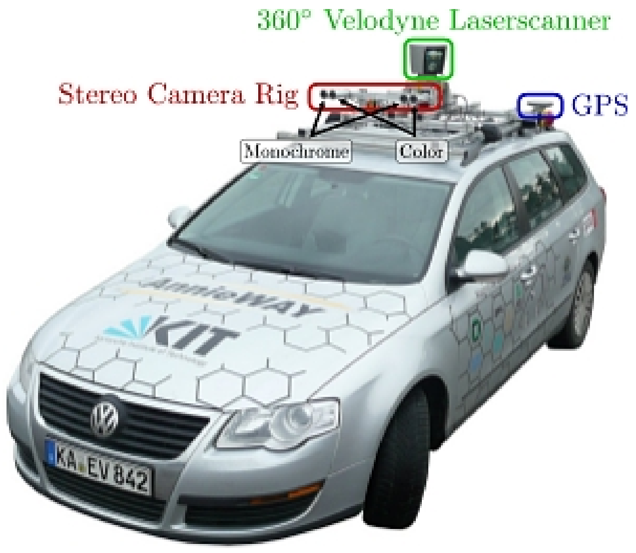
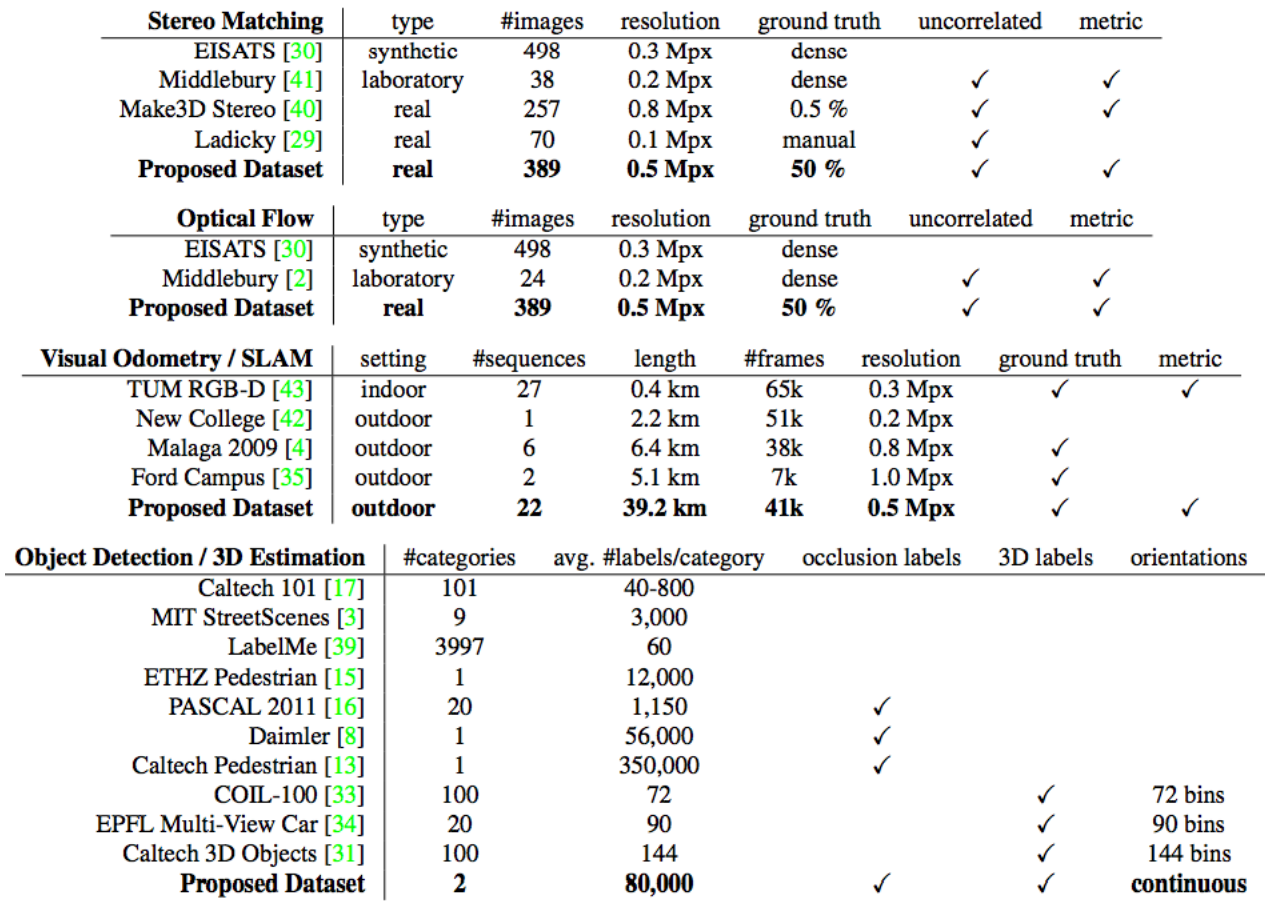


Figure 1: The standard station wagon is equipped with color stereo camera, grey stero camera, Velodyna Laser Scanner, and GPS system.

1.2.1 Sensors and Data Acquisition

The recording system is built from current autonomous driving platform. The aim is to extract suitable data for stereo, optical flow, visual odometry and 3D object detection tasks. The whole benchmarks are captured by driving around a mid-size city of Karlsruhe, in rural areas and on highways. In [13], the authors provides the detailed parameters of all equipments they used: the recording platform is equipped with two color and two grayscale PointGrey Flea2 video cameras (10 Hz, resolution: 1392512 pixels, opening: 90 35), a Velodyne HDL-64E 3D laser scanner (10 Hz, 64 laser beams, range: 100 m), a GPS/IMU localization unit with RTK correction signals (open sky localization errors ¡ 5 cm). Figure 2 shows the summary of KITTI benchmarks and a comparison to other datasets.



Stereo Matching	type	#images	resolution	ground truth	uncorrelated	metric
EISATS [30]	synthctic	498	0.3 Mpx	dcnsc		
Middlebury [41]	laboratory	38	0.2 Mpx	dense	✓	✓
Make3D Stereo [40]	real	257	0.8 Mpx	0.5 %	✓	✓
Ladicky [29]	real	70	0.1 Mpx	manual	✓	
Proposed Dataset	**real**	**389**	**0.5 Mpx**	**50 %**	✓	✓

Optical Flow	type	#images	resolution	ground truth	uncorrelated	metric
EISATS [30]	synthetic	498	0.3 Mpx	dense		
Middlebury [2]	laboratory	24	0.2 Mpx	dense	✓	✓
Proposed Dataset	**real**	**389**	**0.5 Mpx**	**50 %**	✓	✓

Visual Odometry / SLAM	setting	#sequences	length	#frames	resolution	ground truth	metric
TUM RGB-D [43]	indoor	27	0.4 km	65k	0.3 Mpx	✓	✓
New College [42]	outdoor	1	2.2 km	51k	0.2 Mpx		
Malaga 2009 [4]	outdoor	6	6.4 km	38k	0.8 Mpx	✓	
Ford Campus [35]	outdoor	2	5.1 km	7k	1.0 Mpx	✓	
Proposed Dataset	**outdoor**	**22**	**39.2 km**	**41k**	**0.5 Mpx**	✓	✓

Object Detection / 3D Estimation	#categories	avg. #labels/category	occlusion labels	3D labels	orientations
Caltech 101 [17]	101	40-800			
MIT StreetScenes [3]	9	3,000			
LabelMe [39]	3997	60			
ETHZ Pedestrian [15]	1	12,000			
PASCAL 2011 [16]	20	1,150	✓		
Daimler [8]	1	56,000	✓		
Caltech Pedestrian [13]	1	350,000	✓		
COIL-100 [33]	100	72		✓	72 bins
EPFL Multi-View Car [34]	20	90		✓	90 bins
Caltech 3D Objects [31]	100	144		✓	144 bins
Proposed Dataset	**2**	**80,000**	✓	✓	**continuous**

Figure 2: Comparison of current State-of-the-Art Benchmarks and Datasets. Proposed Dataset means KITTI benchmark.

1.3 Tasks

In this section, we introduce tasks provided by KITTI benchmark suites and the leading methods in each task.

1.3.1 Stereo Matching

Stereo Matching [1] is the extraction of 3D information from digital images. Two cameras displaced horizontally from one another are used to obtain different views on a scene. By comparing these two images, the relative depth information can be obtained in the form of disparities.

The state-of-the-art method of stereo matching in KITTI benchmark is Displets[5]. Stereo Matching is a traditional topic and has tremendous progress in last decades. However, certain problems still exist. The disparities of reflecting or textureless surfaces are not easily recovered. The reason is the points in these surface are less discriminative. Normally, the points with specific edges are easily found correspondence in another image and vice versa. Once we can not match points in these two images accurately, the disparity map will not be correct. Displets [5] proposes to use object knowledge to conquer this problem. The most common object in the street is a vehicle. By using this observation, Displets uses a set of 3D CAD models of vehicle as knowledge and make the output disparity map fit these models acting as a long distance regularizers. Figure 3 shows that the disparity map is improved and smoother when class-specific, car object knowledge is given.

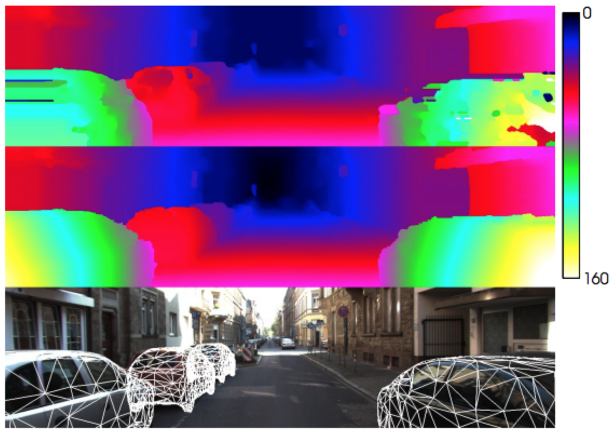


Figure 3: Resolving Stereo Matching Ambiquities: without using object knowledge,stereo methods often fail at reflecting, textureless surface(top). By using 3D object models, both quantitatively and qualitatively are improved(mid) while the 3D geometry of the objects are recovered in the scene(bottom).

1.3.2 Optical Flow

Optical Flow [2] is the pattern of apparent motion of objects, surfaces, and edges in a visual scene caused by the relative motion between an observer and the scene. Optical Flow can construct motion flow and 3D information by comparing temporarily adjacent image sequences captured by a single moving camera. Normally, optical flow is consider as a low-level operation for motion analysis. It can be used for segmentation of moving object or tracking.

The state-of-the-art method of Optical Flow in KITTI benchmark is VC-SF [6] using multiple consecutive frames and stereo video simultaneously. Traditionally, the computation of Optical Flow only consider two temporarily adjacent frames. However, it can be easily interfered by occlusions or view-dependent bias. In order to solve the noise phenomenon, VC-SF [6] use a longer time period and multiple views generated from the stereo camera. By setting constrains forcing estimation of Optical Flow be consistent across the different views in a longer time period, the results show the effects of outliers will be smaller. In Figure 4, the frame at time t=0 suffered severe disturbance. However, by using multiple frames and viewpoint of the right camera, the results is robust against such noises.

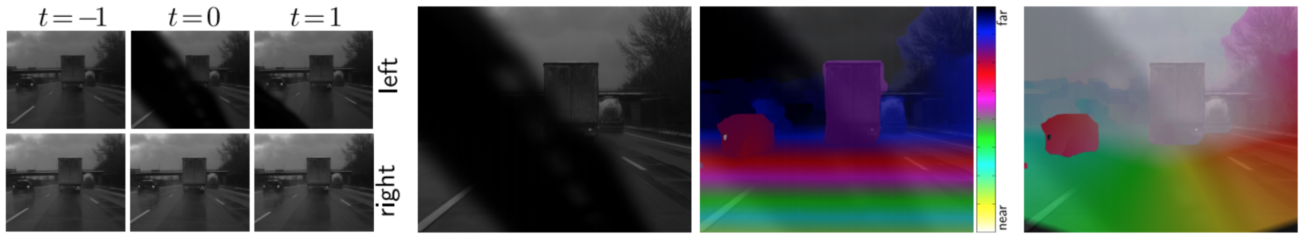


Figure 4: Consistency over multiple frames makes scene flow estimation robust against severe disturbances like the windscreen wiper. (left) Input frames. (middle) The left view at time t = 0. (right) Our scene flow estimate for that viewpoint (shown, from left to right, as disparity and reprojected 2D flow field).

1.3.3 Visual Odometry

In current autonomous driving, localization is crucial. Autonomous vehicles need precision localization to figure out the lane they can drive. Most adopted localization systems are GPS-based guidance system, specially in urban areas. However, GPS signals are not available under certain circumstances, such as tunnels or indoor environment. Another one is map-based system. Such system needs a pre-built databases containing street views of cities. When the autonomous vehicle is running, it can query the scene to know the current location. Due to the pre-built database is need, the areas which autonomous vehicle can drive are limited. Therefore, computer vision community propose Visual Odometry as a second source to aid localization.

Visual Odometry [3] is the process of determining the position and orientation by analyzing the associated camera images. Most existing approaches contain following stages. First, feature extraction is applied on every frame of the input video. The goal of feature extraction is to match the point across frames and use these points to construct the optical flow. Then the optical flow can be analyzed to estimate the motion of camera.

Most methods submitted to KITTI evaluation website use not only traditional videos but also 3D information from LIDAR system or stereo camera. The key step in Visual Odometry is constructing the optical flow. By using 3D information, a more accurate optical flow result can be expected.

Currently, the leading approach V-LOAM [7] can run in real-time (at least 30 frames per second) and provide accurate results. Figure 5 is the result of V-LOAM. First, the distance difference between ground truth and the estimation from Visual Odometry is really small. Another one is that the loop is detected. If the localization system is not accurate enough, redundant paths will be generated.

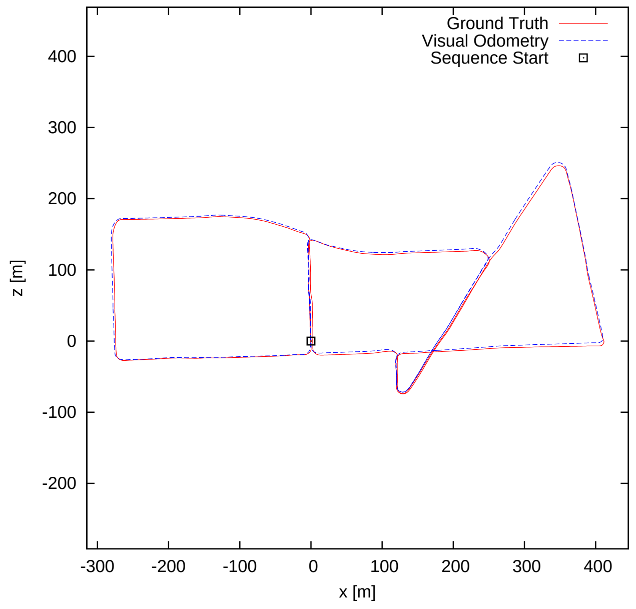


Figure 5: A example generated by V-LOAM shows that the loop can also be detected instead creating multiple redundant paths

1.3.4 3D Object Detection

Object Detection [4] is a technology dealing with detecting instances of semantic objects of a certain class. In KITTI benchmark, Object Detection focuses on three kinds of object: Car, Cyclist, and Pedestrian. Most methods submitted in KITTI evaluation website only use a single monocular image without 3D information from LIDAR system or stereo camera. Ideally, the 3D information can provide a better shape feature and improve the detection result. We think it is worth to further investigate this phenomenon.

Regionlets [8] achieves great results on these 3 classes. Because the exist of local parts of object is an useful feature for object detection, the intuition is finding discriminative local parts of object then deciding the possible locations of these parts. Figure 6 is an example illustrating how it works. In pedestrian detection, hands can be a good discriminative local part of pedestrian due to the color. Because of physical composition, when searching hands, we only need to consider the upper side of bounding box represented as blue rectangles. The distribution of location of hand can be discovered by calculating through all training images. In this case, the most N possible locations are extracted, called regionlets, and shown as orange rectangles. So, in testing time, given a bounding box, we only need to detect the hand appear or not in these regionlets. Multiple models are generated to represent different parts of pedestrian. The output of each model will be a feature for pedestrian detection. Regionlets use a boosting learner to choose useful features and the coefficient of them.

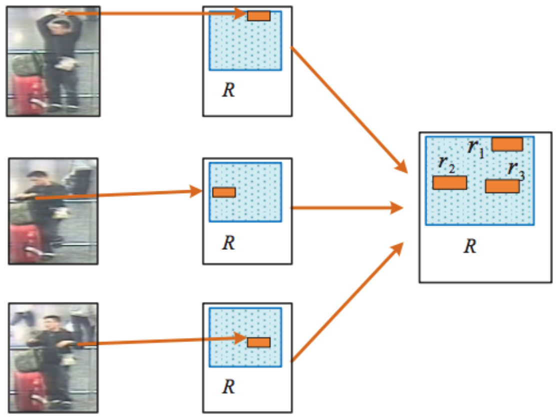


Figure 6: Illustration of the relationship among a detection bounding box, a feature extraction region and regionlets. A feature extraction region R, shown as a light blue rectangle, is cropped from a fixed position from 3 samples of a person. Inside R, several small sub-regions denoted as r1, r2 and r3 (in orange small rectangules) are the regionlets to capture the possible locations of the hand for person detection.

1.4 Conclusion

KITTI benchmark suites provide useful resource for autonomous driving development in computer vision field. KITTI evaluation website provides a platform to let users submit their methods and compare them in different ways: runtime, environment, extra input information, and coding language.

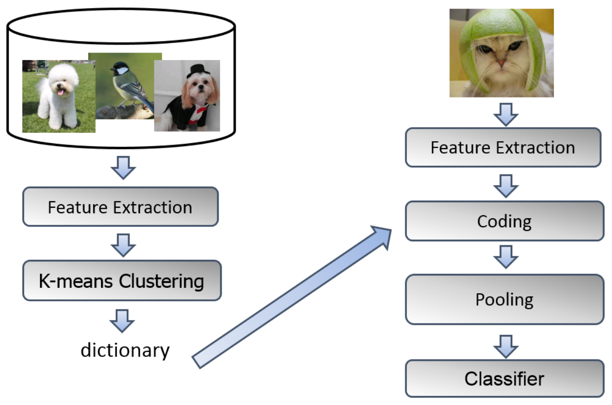


Figure 7: A common computer vision pipeline for image classification. Slide credit: http://ufldl.stanford.edu/eccv10-tutorial/

2 Performance of Computer Vision

2.1 Background of computer vision and real-time computing

In this section, we will introduce basic technique of computer vision which is related to performance issue and the basic concept of real-time computing. In terms of automotive, pedestrian detection becomes the most relevant topic. Also, how pedestrian detection operates in real time is an essential part.

2.1.1 Computer Vision

Here the technique of computer vision we focus on is pedestrian detection in terms of pedestrian. Figure 7 shows that A common computer vision pipeline for image classification. First, we analyze basic images and then extract these images' feature. With these image feature, we can construct the dictionary which helps to search the specific image in the frame we want or classify the image whether is the object we are looking for or not. Typically, the training flowchart can be separated by four steps, such as extract training sample, feature extraction, classifier selection, and validation. We will go through these four step and introduce these basic four techniques for pedestrian detection as following.

Extract Training Sample In extract training sample [9] in Fig 8, it shows an excerpt from the Daimler pedestrian detection benchmark data set. The benchmark data set has three parts: Pedestrian training samples consider as positive samples which provided as training examples is 15,660. And, these samples are got from manually extracting 3,915 rectangular position labels from video images. Second part is negative

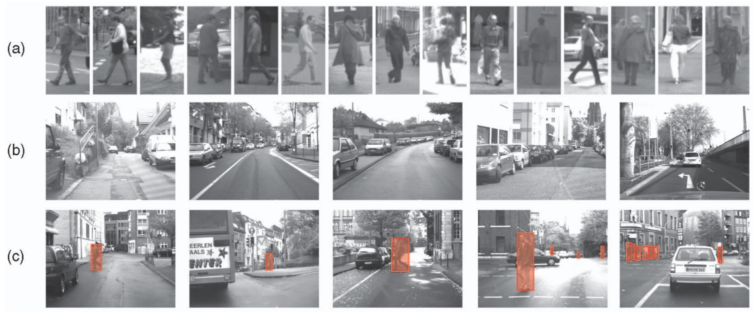


Figure 8: Daimler pedestrian detection benchmark data set: (a) Pedestrian training samples, (b) nonpedestrian training images, (c) test images with annotations. Source from [9].

samples which contained 6,744 full images without any pedestrians. With the positive and negative samples, we can detect pedestrian in the images shown as Fig 8 (c) with annotations.

Feature Extract In feature extract, one of the essential techniques is Histogram of Oriented Gradients (HOG), see Fig 9. HOG are feature descriptors used in computer vision and image processing for the purpose of object detection. Paper [9] shows that local gradients are packed by using their orientation, then weighted by their magnitude. That confined in a spatial grid of cells with overlapping blockwise contrast normalization. With overlapping block, we extracted a feature vector by sampling the histograms from the contributing spatial cells. Finally, we get the feature vectors for all blocks which are concatenated to produce a final feature vector, which is subject to classification using a linear support vector machine (linSVM).

Classifier Selection In classifier selection, support vector machines (SVM) can be used as supervised learning models which associated learning algorithms. These algorithms can analyze data and recognize patterns for classification and regression analysis. Moreover, a SVM yields a hyperplane or set of hyperplanes in a high- or infinite-dimensional space. We use the hyperplane for classification, regression, or other tasks. That is, a good separation is achieved by the hyperplane that can create largest distance to the nearest training-data point of any class.

2.1.2 Real-time Computing

Real-time computing provides hardware and software systems a ”real-time constraint”, for example operational deadlines from event to system response. Real-time programs have to guarantee response within specified time constraints, called as ”deadlines”. And real-time responses are often known as in the order

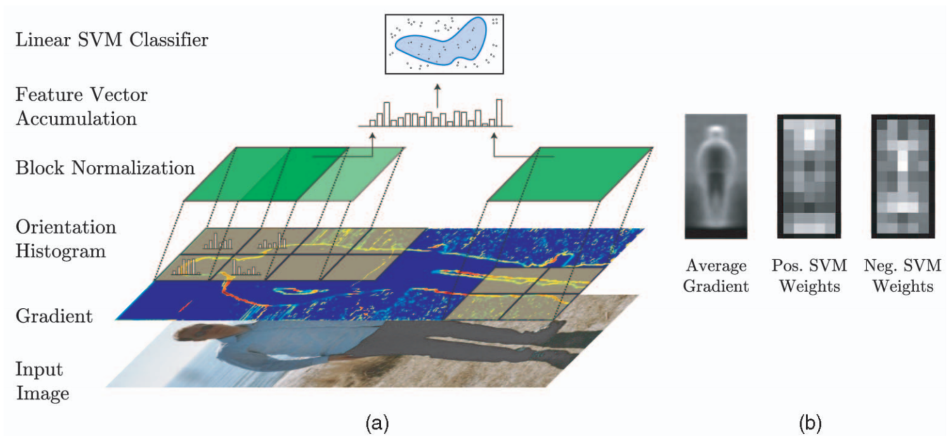


Figure 9: (a) Overview of HOG/linSVM architecture. Cells on a spatial grid are shown in yellow, whereas overlapping normalization blocks are shown in green. (b) Average gradient image along with visualization of positive and negative SVM weights, which highlight the most discriminative regions for both the pedestrian and nonpedestrian classes. Source from [9].

of milliseconds, and sometimes microseconds. A system is said to be real-time if the system satisfies two a dual notion of correctness, such as logical correctness and temporal correctness. Logical correctness means that real-time system does the right thing; temporal correctness is that system finishes its job on time. Otherwise, in real-time computing, there are two standards classified by the consequence of missing a deadline: hard real-time, firm real-time, and soft real-time. In terms of hard real-time, if any hard deadline is ever missed, then the system is incorrect. Requires a means for validating that deadlines are met. Soft real-time, however, may occasionally be missed. The usefulness of a soft real-time with missing its deadline, thereby degrading the system's quality of service. Otherwise, firm real-time is that a soft deadline such that the corresponding jobs usefulness function goes to 0 as soon as the deadline is reached.

2.2 Computer Vision and Real-time

When it comes to computer vision in real time, people usually set real-time constraint as 30 fps (frames per second). In order to succeed 30 fps, the research use different approaches to achieve the real-time goal, such as software level, hardware level, and hybrid. In the following, we talk about three papers [10, 11, 12] and exploit the basic how they achieve real-time computing of pedestrian detection.

2.2.1 Software Level

The research [10] talks about pedestrian detection in driver-assistance systems. They propose a monocular vision system for real-time pedestrian detection and tracking during nighttime driving with a near-infrared (NIR) camera, see Fig. 10. They use three modules, such as region-of-interest (ROI) generation, object classification, and tracking. These three modules are integrated in a cascade, and each utilizes complementary visual features which can distinguish the objects from the cluttered background in the range of from 20 to 80 meters. In Fig. 10, the ROI generation module contains two steps: image segmentation and candidate selection. The image segmentation step contains two thresholds in order to calculate each pixel the horizontal scan line.

That is able to determine the foreground based on the fact that pedestrians which appear brighter than the nearby background in NIR images. In pedestrian classification, a tree-structured, two-stage detector, based on Haar-like, digital image features used in object recognition, and HOG features, is used to deal with the problem that high performance and real-time constraint against the large intra class variability in the pedestrian class. Otherwise, AdaBoost here is used to select the critical features and learn the classifiers from the training images.

This paper balances the robustness and efficiency at a high performance level. There are some novel approaches have been proposed which are the efficient adaptive dual-threshold segmentation for candidate generation, the tree-structured two-stage detector to reduce the complexity of pedestrian classification caused by large intraclass variability, and template-matching-based tracking for multiframe validation. Hence, results show that, on a Pentium IV 3.0-GHz personal computer, the algorithm can produce a detection rate of more than 90% at the cost of about 10 false alarms/h and perform as fast as 30 fps.

2.2.2 Hardware Level

We discuss the pedestrian detection achieve real-time computing by using software scheme. For this part, we introduce a paper [11] which takes advantage of hardware performance to reach above 30 fps. [11] try to find out the workload of pedestrian detection which can be parallel executed by hardware and distribute jobs to FPGAs.

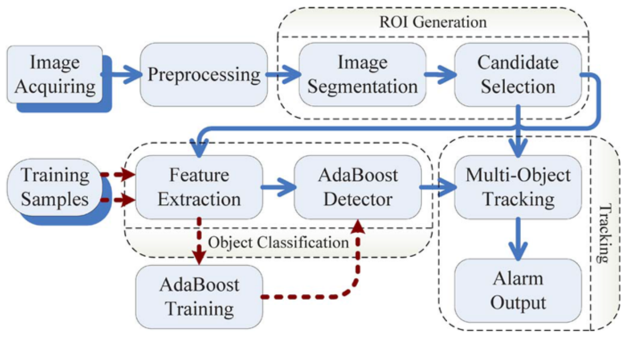


Figure 10: Overview of the system modules. Source from [10].

According to paper [11], Co-occurrence histograms of oriented gradients (Co-HOG) is a powerful feature descriptor for pedestrian detection. Co-HOG uses uses pairs of gradient orientations. However, its calculation cost is large because the feature vector for the CoHOG descriptor is very highdimensional. Each co-occurrence matrix C with an offset (x, y) can be expressed as a 64-dimensional vector $f_{k,x,y}$, see below:

$$C_{i,j} = \sum_{p=0}^{n-1}\sum_{q=0}^{m-1}\begin{cases}1, if I(p,q) = i and I(p+x,q+y) = j,\\ 0, otherwise\end{cases}$$

According to CoHOG feature vector F, K 31 co-occurrence matrices for all combinations of block k and offset (x, y) are calculated. This part is key component that we can use that for parallel computing, since no overlaps and calculation dependencies between any two blocks exist. Hence, all cooccurrence matrices $f_{k,x,y}$ can be calculated in parallel. Paper shows that it can exploit $K \times 31$ parallelism at its maximum with this parallelization approach.

Paper [11] shows the proposed architecture for pedestrian detection based on Co-HOG. Main components of Fig. 11 are a module for generating images which represent gradient orientations and a merged module for executing histogram calculation and SVM prediction at the same time.

This proposed algorithm is implemented on Virtex-5 FPGA and the result shows that hardware architecture can process 139,166 sub-windows per second and it achieves real-time pedestrian detection on 38 fps 320240 video.

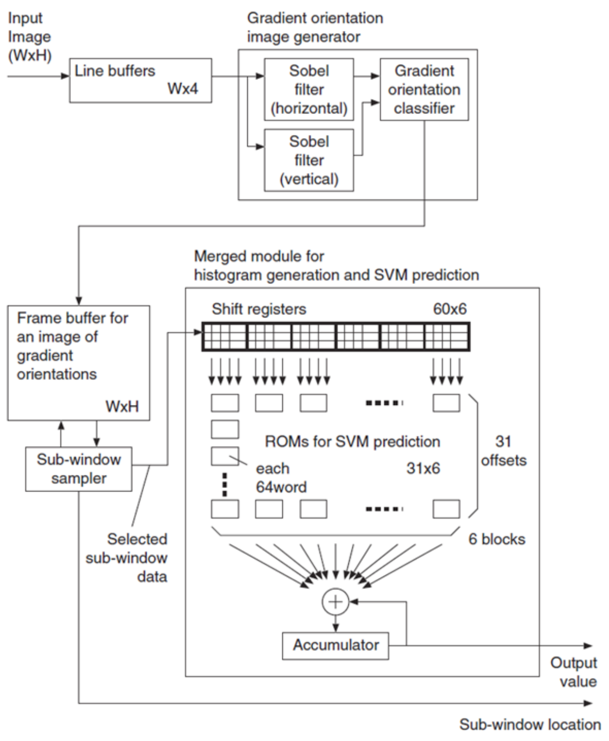
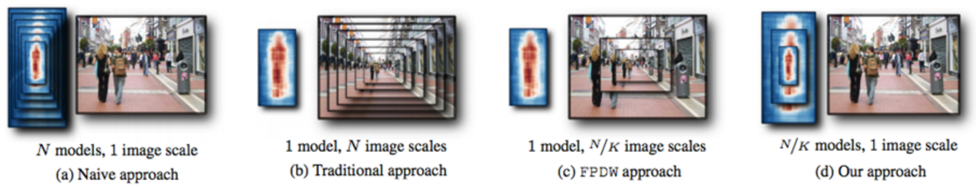


Figure 11: Overview of the proposed architecture. Source from [11].

2.2.3 Hybrid

In this paper [12], they use not only software but also hardware to increase the performance of pedestrian detection. This paper presents a new pedestrian detector that improves both in speed and quality. They modified the process of detection. A basic process of detection contains multiple scales, sliding window, feature extraction, and classify. First, we generate multiple images on different scales and then use sliding window on all locations which find out the region of interest. When the region to analyze is ready, We extract HoG from current window. Final step is using LinearSVM to classify whether the region of interest is the target object or not.

Fig. 12 shows different approaches to detecting pedestrians at multiple scales. Figure 12 (a) is a traditional approach which scales N is usually in the order of 50 scales. This way need a lot of data for this method and also tend to cause overfitting problem. Second, the figure 12 (b) is to train a single model for one canonical scale, and then rescale the image N times. However, this way causes two problems: one is that training a canonical scale is delicate; the other is that one needs to resize the input image 50 times at run-time, and



N models, 1 image scale
(a) Naive approach
1 model, N image scales
(b) Traditional approach
1 model, N/K image scales
(c) FPDW approach
N/K models, 1 image scale
(d) Our approach

Figure 12: Different approaches to detecting pedestrians at multiple scales. Source from [12].

recompute the image features 50 times, too. The third way is that without rescaling the input image N times, one rescales image only N=K times, see figure 12 (c). But, this way can not apply on Histogram of Oriented Gradients (HoG) since HoG need to recalculate. Final, figure 12 (d) shows the way is proposed by this paper. It skips some feature extraction by using N/K model and get approximated models quickly since they scale the features not the images.

With the high parallelism of their solution [12], it will directly benefit from future hardware improvements. The result shows that the proposed approach presented a novel pedestrian detector running at 135 fps in one CPU+GPU enabled desktop computer.

2.2.4 Conclusion

Although pedestrian detection has many challenges, such as various style of clothing in appearance, the presence of occluding accessories, and frequent occlusion between pedestrians, we only focus on performance issue in this report and we do not discuss how detection accuracy is. As we discuss above, we introduce three major papers to exploit how these paper achieve real-time computing in pedestrian detection. However, these papers fail to give their approach a lower bound that guarantee the performance in real-time. As we know the real-time constraint of computer vision is 30 fps, actually 30 fps is not a specific and object requirement which is a threshold human eyes cannot tell. Hence, in terms of computer vision, real-time computing transforms to high performance problem. We can find that these paper try to run detection as fast as they can to reach real-time requirement, but never satisfy real-time guarantee, since the most important requirement of a real-time system is predictability and not performance.

3 Deep Convolutional Neural Network in Computational Perspective

3.1 Introduction

Convolutional neural network (CNN) has been remarkably successful in many computer vision tasks, such as object recognition, detection, segmentation, and so on. Its mainly because of good representation (or image features) learnt by CNNs, and the key enabling factors are the increase of computational power and big labeled datasets, which allow us to scale up the networks to tens of millions of parameters.

It is known that the large networks would consume a lot of computational power, which might not be appropriate for the resource constrained environment, such as mobile or embedded system. However, nowadays we can easily see the powerful and energy efficient embedded systems thanks to the advancement in computer architecture. So, people starts to pay attention to running deep CNN on them and come up with the solutions that is highly optimized in those architecture.

In this report, we presents some explanation in computational perspective so that we will be able to come up with the system that support deep CNN and satisfy real-time constraints.

3.2 The components of deep convolutional neural network.

The deep CNN is constructed by several components. It includes convolutional, pooling, non-linear function, normalization and fully connected layers. It might contain other layers, but these are the common components. Out of those layers, convolutional layer and fully connected layers are main part of computation. So, in this report we focused on the two layers.

3.3 Fully connected layer

Figure 13 is how fully connected layer looks like. Another name of fully connected layer is feed forward neural network. The reason why it is called as fully connected layer is each neurons are densely fully connected.

In order to compute the output of unit h(1), we compute dot product between weights that are connected to h(1) and previous layers outputs, in this case x. h(2) and h(3) also can be computed in this way. So, if we build a weight matrix W, computing h vector would be simple matrix multiplication. There might be

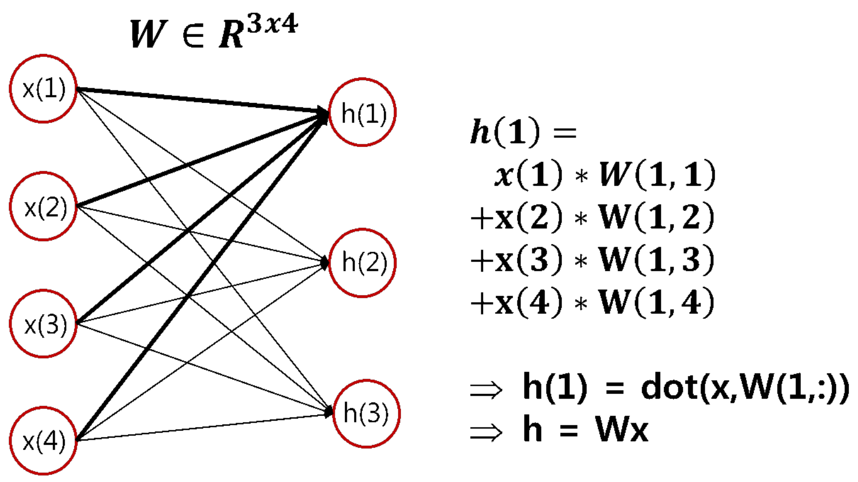

$W \in R^{3x4}$
x(1)
x(2)
x(3)
x(4)
h(1)
h(2)
h(3)
$h(1) = x(1) * W(1,1) + x(2) * W(1,2) + x(3) * W(1,3) + x(4) * W(1,4)$
$\Rightarrow$ h(1) = dot(x,W(1,:))
$\Rightarrow$ h = Wx

Figure 13: Fully connected layer or feed forward neural network

many cases that we need to compute multiple inputs. In that case, we can simply iterate for loops and each iteration, we can simply compute matrix multiplication.

3.4 Convolutional Layer

Convolutional Layer is heart of the deep CNN. Figure 14 shows graphical explanation of how convolution looks like. The input is 2 dimension matrix(actually, it doesnt have to be 2 dimension) and the output is also 2 dimension matrix. The size of output is not necessarily same as the size of input. It could be smaller or even larger, however, in practice we usually pad zeros on the outside of input matrix so that we can make them same size. The size of padding depends of the filter size.

There is the filter that convolve on the input. In other words, to compute the value of outputs ith row and jth col, we compute the dot product between the filters and corresponding inputs region. So, computational complexity is the size of filters times the size of outputs. The size of filters can vary, for example 3x3, 5x5, or 7x7. There is no rules about how large the filters should be. In practice, lower layers tend to have larger filters(7x7) and higher layers tend to have smaller filters(3x3).

3.5 Lowering convolution to matrix multiplication

There are many ways to compute convolutions. The obvious way is to compute convolution directly as previous section explained. For example, we can write some cuda program that each cuda kernel compute

$$x_{ij}^{\ell} = \sum_{a=0}^{m-1}\sum_{b=0}^{m-1} \omega_{ab} y_{(i+a)(j+b)}^{\ell-1}$$

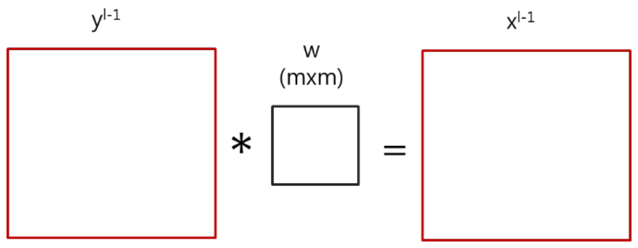


Figure 14: Convolutional layer

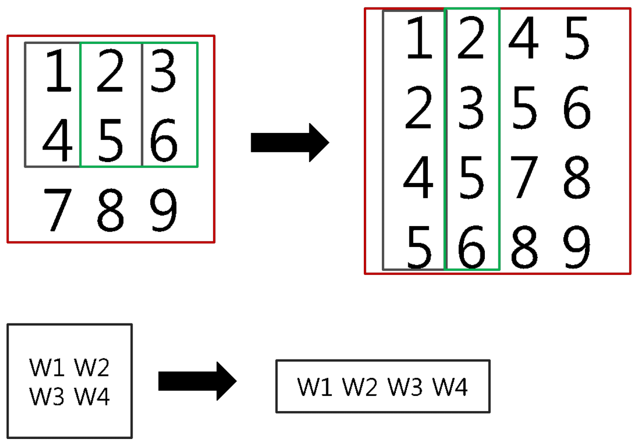


Figure 15: Lowering convolution

each output elements. However, it turns out that it is not easy to implement this in efficient way for every corner cases. We should be able to deal with different size of filters, outputs, inputs, padding, and so on since our computing architectures are very sensitive to memory hierarchy.

Another way is to take advantage of existing efficient computation routines, such as matrix multiplication. This routines have been highly optimized in several programming library, for example BLAS, cuBLAS, and so on. So, people came up with the trick to translate convolutions into matrix multiplication, which sometimes is called as Lowering to matrix multiplication.

Red box in figure 15 is the input and black box is the filters(or weights). Here, the size of input is 3x3, there is no padding, the size of filters are 2x2, so the size of output is going to be 2x2. Instead of computing convolution directly, we change the form of input(left) into right side. For example, we take 2x2 patch out

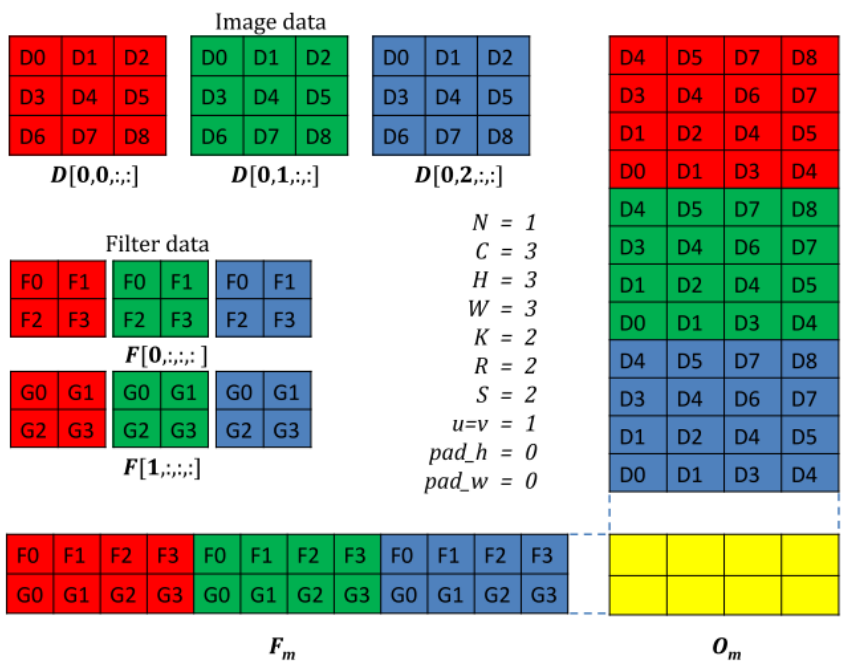


Figure 16: Lowering convolution from cuDNN [14]

of inputs, (1,2; 4,5), and make it as one column vector, (1;2;4;5). And similarly (2,3; 5,6) -¿ (2;3;5;6). Once we change the form in this way, we can easily compute convolution with matrix multiplication.

In real deep CNN, there are multiple input channels. For example, the first layers input channel might be R,G,B color channels in image case. Our filters are going to be 7x7x3, here 7 is the height and width of the filters and 3 is the number of channels. The way to compute the output is first to compute convolution on each input channels, and then sum of 3 outputs to get one output. In addition to multiple input channels, we can also have multiple filters. If the number of filters is 10, our filters are going to be 7x7x3x10 tensor. In this case, the number of output channels is 10. Obviously, the output channels is input channels of next layer, so next layers the number of input channels is 10. Given all these things together, we can lower the convolution to large one matrix multiplications.

In this example, we have 3 input channels, 2 filters, the size of input is 3x3, the size of filters is 2x2. Therefore, the number of output channels is 2, the size of one output channel is 2x2 and the total dimension of the output is 2x2x2. As you can see in the figure 16, we can make it as one big matrix multiplication. Here output matrix O is 2x4, which can be easily converted to 2x2x2.

3.5.1 Discussion

One might ask about the cost of lowering the convolutions. It turns out that it can be implemented in very efficient way and no acatual computation is involved. Another issue is the size of converted matrix. It might be some problem because sometimes it is impossible to allocate a large chunk of memory. So, we might need to have some ways to divide the computation into several steps.

3.6 Other Layers

There are some important layers we havent covered. One is non-linear activation function. It is simple element wise operation, so we can easily parallelize this layer. Another important layer is pooling layer. It is very similar to convolutional layer in terms of computation. For average pooling layer, we can easily design convolution filters that exactly do the same thing. For max pooling layer, we can easily replace the dot product with max operation.

References

[1] ”http://en.wikipedia.org/wiki/Computer_stereo_vision.”

[2] ”http://en.wikipedia.org/wiki/Optical_flow.”

[3] ”http://en.wikipedia.org/wiki/Visual_odometry.”

[4] ”http://en.wikipedia.org/wiki/Object_detection.”

[5] Fatma Gney and Andreas Geiger,”Displets: Resolving Stereo Ambiguities using Object Knowledge,” *Conference on Computer Vision and PatternRecognitionRecognition (CVPR)*, 2015.

[6] Christoph Vogel and Stefan Roth and Konrad Schindler, ”View-Consistent 3D Scene Flow Estimation over Multiple Frames,” *ECCV (4)’14*

[7] Ji Zhang and Sanjiv Singh, ”Visual-lidar Odometry and Mapping: Low-rift, Robust, and Fast,” *IEEE International Conference on Robotics and Automation(ICRA)*, 2015.

[8] Xiaoyu Wang and Ming Yang and Shenghuo Zhu and Yuanqing Lin,”Regionlets for Generic Object Detection,” *International Conference on Computer Vision*, 2013.

[9] Enzweiler, Markus, and Dariu M. Gavrila., “Monocular pedestrian detection: Survey and experiments”, *Pattern Analysis and Machine Intelligence*, IEEE Transactions on 31.12 (2009): 2179-2195.

[10] Ge, Junfeng, Yupin Luo, and Gyomei Tei., “Real-time pedestrian detection and tracking at nighttime for driver-assistance systems,” *Intelligent Transportation Systems*, IEEE Transactions on 10.2 (2009): 283-298.

[11] Hiromoto, Masayuki, and Ryusuke Miyamoto. “Hardware architecture for high-accuracy real-time pedestrian detection with CoHOG features,” *Computer Vision Workshops (ICCV Workshops)*, 2009 IEEE 12th International Conference on. IEEE, 2009.

[12] Benenson, Rodrigo, Markus Mathias, Radu Timofte, and Luc Van Gool. “Pedestrian detection at 100 frames per second,” *In Computer Vision and Pattern Recognition (CVPR)*, 2012 IEEE Conference on, pp. 2903-2910. IEEE, 2012.

[13] Andreas Geiger and Philip Lenz and Raquel Urtasun, ”Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite,” *Conference on Computer Vision and PatternRecognition (CVPR)*, 2012.

[14] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro and Evan Shelhamer., “cuDNN: Efficient Primitives for Deep Learning”, *http://arxiv.org/abs/1410.0759*,