SqueezeMap: Fast Pedestrian Detection on a Low-power Automotive Processor Using Efficient Convolutional Neural Networks

Rytis Verbickas[1], Robert Laganiere[2]
University of Ottawa
Ottawa, ON, Canada
[1]rverb054@uottawa.ca
[2]robert@laganiere.name

Daniel Laroche, Changyun Zhu, Xiaoyin Xu, Ali Ors
NXP Semiconductors
Ottawa, ON, Canada
{daniel.laroche,changyun.zhu,christina.xu, ali.ors}@nxp.com

Abstract

Pedestrian detection for autonomous driving is a challenging task that requires careful trade-off between accuracy, storage, computation and energy requirements. In our work, we extend the recent SqueezeNet [1] architecture to pedestrian detection. We show how this network can be modified to obtain detection performance on the Caltech USA [2] pedestrian dataset that is comparable in overall log-average miss rate to other competing models while easily running at 30FPS on an automotive processor with a model size of 3.24MB and within a power envelope of 2W. The extension relies on the observation that precise knowledge of bounding box corners is not necessary to know the location of pedestrians and their approximate size. Rather, a coarse grid based localization is proposed here and acts as a kind of heatmap of pedestrian locations, relying on only one forward pass through the network. The number of new free parameters introduced is small relative to the original SqueezeNet model.

1. Introduction

In the era of autonomous vehicles and smart assist vehicle computers, a robust and above all safe driving system requires a model which can create an accurate representation of the environment. Such systems rely on sensors for its input, which can include laser, radar or camera based solutions. Camera solutions offer a cheap and simple input source for such systems.

For such camera dependent systems, the data needed to train them is readily available, foregoing the necessity of manual collection. The abundance of data is helpful, since the state-of-the-art models for object detection and classification are generally deep learning models which have a large number of tunable parameters. A subset of these deep models, convolutional neural networks (CNN's), have been especially successful for a wide range of image processing tasks, including segmentation, classification and detection. However, the majority of research in this area has focused on improving outright performance and exploring the full design space of CNN architectures while placing less emphasis on model size.

For an embedded system in an autonomous vehicle, this emphasis is paramount, because models not only have to be small in terms of run-time footprint, and therefore a small footprint in power consumption, they also small in total model parameters as changes to the model need to be pushed over network links with limited bandwidth. The decrease in footprint can help to reduce the run-time computational requirements, as any pedestrian detection system needs to ensure a processing speed sufficient for performing avoidance as well. With a decrease in model size also comes the tricky task of optimization to maintain the same level of performance relative to existing, unconstrained models. This is complicated by the fact that autonomous vehicles systems need to maintain high levels of recall and precision in order to be safely deployed.

In this paper we focus on approximating pedestrian locations using a coarse grid-based approach. We do this by adding an additional layer to an existing SqueezeNet network. This layer performs a kind of weighted voting scheme across the depth dimension of the previous (convolutional) layer to determine whether a pedestrian is present. Our approach introduces only a small number of new parameters, with a tiny final model size of 3.24MB. This model can run at 72FPS on a GTX Titan X GPU on an RGB input of 681x227 and at 30FPS on dual APEX-2 low-power processors [16] with the entire system running within a 2W power envelope. We estimate the accuracy of the approach relative to other models trained on the Caltech-USA dataset for a range of model and evaluation settings and show it has comparable performance in terms of log-average miss rate.

2. Related Work

Pedestrian detection is a special case of more general object detection. Investigation in this area has produced a wide range of solutions and a number of benchmark datasets, including Caltech-USA, KITTI [6] and ETH [7] among others. The most popular among these is arguably Caltech-USA, which stands out because it is relatively large, challenging and has a large number of solutions which have been evaluated on it. These solutions include variants of Viola&Jones [15], HOG [5], deformable part models (DPM) [11] and various types of neural networks including CNNs [19]. While the non CNN variants have steadily improved over time, it is the CNN-based solutions we are interested in due to their repeating, general structure and operations which can be trained easily end-to-end and consistently achieve state-of-the-art performance.

A number of variants for this task which use a CNN as a starting point have been proposed, including Region-based CNN's (or R-CNN's) by Girshick et al. [10] and Fully-convolutional networks (FCN's), which focus on semantic segmentation, popularized by Long et al. [13] (including variants such as R-FCN which focus on object detection). The R-CNN strategy is to identify regions of interest as a starting point for applying a CNN. Faster and more computationally efficient variants are available, such as Fast-RCNN [12] and Faster-RCNN [14], which share the computation time for proposed regions. Overall, these approaches can be slow without powerful GPU's. To address the speed shortcoming, newer approaches such and YOLO (You Only Look Once) [4] have been proposed which combine classification and region proposals into a single stage while estimating bounding boxes.

While region proposal based approaches have shown good performance on Caltech-USA, we want to avoid having to estimate the exact pixel-wise locations of bounding box corners and instead have a cluster of activation groups to indicate the approximate extent of a pedestrian (while using a bare minimum of parameters to do so). Thus instead of estimation of bounding box corners or single pixel labeling, we focus on a coarser grid-based level where a gridbox in the grid being 'on' indicates a pedestrian feature is present within that gridbox.

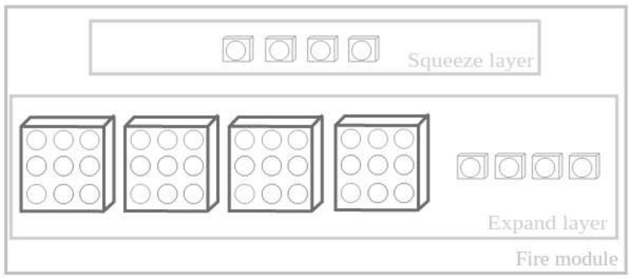

Figure 1: A fire module used to construct a SqueezeNet network. The squeeze layer is meant to show 4 FMs which each use 1x1 convolutional filters. The expand layer has 4 FMs which use 3x3 filters and 4 which use 1x1 filters.

3. Method Description

3.1. Network Structure

SqueezeNet is a small, recently developed, CNN architecture (shown in Figure 1) that was constructed to emphasize small model size while retaining classification accuracy relative to popular CNN architectures like AlexNet [9]. It is able to do this by following 3 strategies which have either been shown experimentally to work well or prevent an explosion of parameters:

1. Placing an emphasis on 1x1 convolutions
2. Restricting the number of input channels to filters larger than 1x1
3. Delayed down-sampling throughout the network

The first 2 strategies are mainly aimed at parameter reduction. By keeping the number of feature maps in squeeze layers small, and with only 1x1 convolutions, strategy 1 and 2 are achieved by saving parameters and by supplying only a small number of channels to the 3x3 'expand' feature maps. Thus as we propagate through consecutive fire modules we're repeatedly squeezing the previous modules (expand layer, fireN) feature map output through a small number of feature maps (squeeze layer, fireN+1), followed by a larger number of feature maps (expand layer, fireN+1). Strategy 3 is based on observations made in prior experimental results [17], which attempts to keep feature maps relatively large (to the input) in an attempt to learn better features. The intuition is that this should give higher accuracy (albeit at higher computational cost and runtime memory footprint). Since fire modules don't decrease the size of their input, repeated propagations through multiple fire modules mean the input is not downsampled, in the process helping to fulfill strategy 3.

Layer Name	Layer Size	Filter Size / Stride	s1x1	e1x1	e3x3	Num. params
Input	227x227x3					
conv1	113x113x64	3x3/2				1792
pool1	56x56x64	3x3/2				
fire2	56x56x128		16	64	64	11408
fire3	56x56x128		16	64	64	12432
pool3	27x27x128	3x3/2				
fire4	27x27x128		32	128	128	45344
fire5	27x27x256		32	128	128	49440
pool5	13x13x256	3x3/2				
fire6	13x13x384		48	192	192	104880
fire7	13x13x384		48	192	192	111024
fire8	13x13x512		64	256	256	188992
fire9	13x13x512		64	256	256	197184
hm0	13x13	1x1x512				86697
			Total Parameters			809193
			Total Size (MB)			3.237

Table 1: Summary of network parameters

Although SqueezeNet is amenable to compression techniques (pruning, deep compression [18]), we do not investigate this option due to the already small footprint of the network and the overhead introduced in some of these approaches. A summary of the network parameters is shown in Table 1, inspired by a similar table in the original SqueezeNet publication. Note the feature map dimensions are slightly different in our TensorFlow implementation than in the reference Caffe model.

We use SqueezeNet 1.1 as the base model, pre-trained on the ImageNet 2012 dataset [8]. This model is an extension of v1.0, achieving almost identical performance on ImageNet while reducing the number of computations by 2.4x through a small re-organization of the original model. A summary of the required multiply-accumulate (MAC) operations at each layer is shown in Table 2.

Layer Name	MACs
Input	/
conv1	22064832
pool1	/
fire2	35323904
fire3	38535168
pool3	/
fire4	32845824
fire5	35831808
pool5	/
fire6	17651712
fire7	18690048
fire8	31842304
fire9	33226752
hm0	86528
Total	266.1M

Table 2: Summary of MAC operations for regular layers

We remove the conv10 layer entirely, since it was found this does not impact performance. In the process we save ~500k parameters, leaving ~722.5k. After the addition of our partially connected layer we have just under 810k parameters or a model about 3.237MB in size. The addition of batch normalization (BN) after fire9 introduces 2 parameters for each output in fire9, resulting in 810,217 parameters (a final size of 3.24MB). The choice of base model is open and is not mandated by our approach. The only other change we investigate to the base SqueezeNet model is swapping the ReLU units after each convolutional layer to exponential ReLU units, which decay exponentially when the argument is less than 0, but found no noticeable effect to ReLU's.

Our primary goal is similar to SqueezeNet, which is to maintain a small number of parameters, while obtaining a coarse estimate of pedestrian location and size. A standard way to do this would be to introduce a fully connected layer that has the same width and height as fire9, allowing an estimate of the presence/absence of a pedestrian at each pixel of the FM. However fully connected layers, while much less computationally burdensome than convolutional layers, are generally wasteful in terms of storage. State of the art networks which have large fully connected layers can usually be pruned of about 90% or more of their parameters [18]. In addition, unless the scale of the pedestrian is large enough to fill a large portion of the fire9 activations (the pedestrian is too close), our intuition is to keep the number of connections across the width and height of fire9 small and instead focus on the depth dimension in fire9. The output of the partially connected layer can (but doesn't need to) match the height and width of the convolutional layer (given by H and W respectively) it is connected to, connecting only across the depth dimension (D). For a convolutional layer, each activation A_{hwd} can be indexed by (h,w,d) where each is in a range from 0 to H, W, D respectively.

For an output in the heatmap layer, (h,w), its input is computed as:

$$\sum_{d=0}^{D-1} w_{hwd} * A_{hwd} + b_{hw} \qquad (1)$$

The number of parameters is reduced from H*W*(H*W*D+1) to H*W*(D+1). That means only 86,528 additional parameters for this layer in our case.

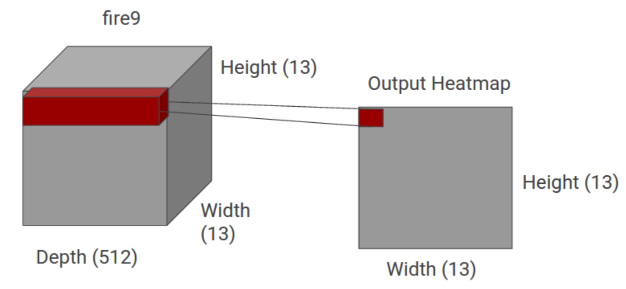

Figure 2: Connectivity along the depth dimension between fire9 and the output layer

This scheme is essentially performing a weighted voting along the depth dimension, for every pixel in the partially connected layer (Figure 2). We could ask why we would want to have different weight sets for voting at each output pixel instead of applying the same set of weights (essentially a kind of depth convolution). Taking this further, majority pooling could be performed which will activate an output if a majority of the inputs are 'on' (using some fixed threshold) saving even more weights. The benefit of having only one set of weights is tested by experiment but we leave majority pooling for future work.

We add dropout between the fire9 and heatmap layers, although we investigate its placement at other locations within the SN network, and experiment with various dropout rates <= 50%. The effect of BN after fire9 is also investigated, including placement before or after the dropout layer. At the output layer, we use tanh as our activation function although we did experiment with exponential linear units (ELU's) at the output layer but were not able to obtain consistent or comparable performance. Preliminary experiments using the output of layers upstream of fire9, followed by a heatmap layer, have shown a decrease in performance although the extent has not been fully investigated. For example, it may be possible to further trim the network depth while remaining close to the reference performance level when using the fire9 layer.

3.2. Data Generation

For generating the training dataset we overlay a set of 227x227 patches over 640x480 images. Each overlay has a 13x13 grid embedded inside. To make the 13x13 grid fit neatly we discard 3 columns of pixels on the left and right sides of the image (resulting in 17x17 pixel gridboxes). A ground truth bounding box, when intersected with the grid in a patch will intersect with a set of gridboxes, S. We leave the intersection percentage between the gridbox and the ground truth bounding box as a tunable parameter when generating our dataset. We experiment with values in the range of 35% to 75%. An example of an overlaid grid with ground truth bounding boxes is shown in Figure 3. The adjacent image shows the resulting gridboxes which are deemed to be active after applying a 50% threshold.

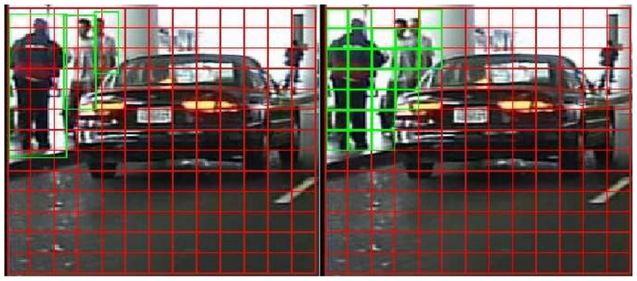
Figure 3: A grid overlaid on an image patch with ground truth bounding boxes shown (left) and the resulting target gridboxes after using a 50% threshold

We experiment with various ways to overlay the 227x227 input over the original 640x480 images. The most important region of the input images is just above (~200 pixels from the top of the image) the horizontal centerline of the image as described in [2]. We focus on the following sampling regimes for our training data:

1. Sample 3 227x227 regions along the horizontal centerline.
2. Sample 6 227x227 regions, 3 along the centerline and 3 samples offset upwards by ¼ of the sampling region height

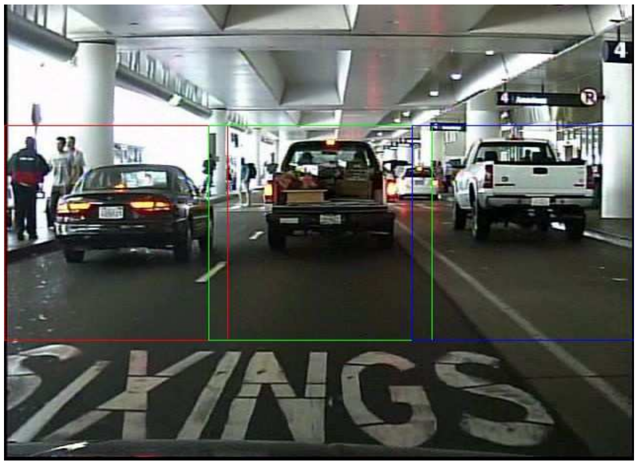
Figure 4: Example of sampled regions for '3pos'

We allow one column of gridboxes to overlap when transitioning horizontally between one grid to the next. An example of sampling for Case 1 is shown in Figure 4, with the extent of the 3 regions overlaid (red, green and blue). When generating the evaluation dataset, we follow the process detailed in [2], of generating samples every 30 frames (starting with the 30th). We investigate a number of sampling schemes for building the evaluation dataset and report performance for all:

1. Sample 3 227x227 regions along the horizontal centerline (sampling scheme named '3pos')
2. Sample 6 227x227 regions, 3 horizontally spaced starting at the top left of the image and 3 spaced similarly starting at (227,0) (named '6hilo')
3. Sample 9 227x227 regions, 3 along the horizontal centerline, 3 offset upwards to (0,0) which places 3 sampling regions along the top of the image and 3 offset downwards to (411,0) which places the last 3 regions along the bottom of the image (named '9pos')

The first scheme focuses on our main area of interest along the horizontal centerline. The second scheme attempts to cover as much of the image as possible starting along the top and is closest to achieving unique coverage over the original 640x480 image. The third scheme attempts to maintain the horizontal centerline while also covering as much of the image as possible.

3.3. Training Procedure

We use an L2 loss function to train our network. Although we experiment with L2 regularization, we found that skipping it generally produces better performing networks. During training, we apply exponential averaging to the network weights and to the computed loss, using rates of 0.9999 and 0.9 respectively. We experimented with a number of optimizers, including vanilla gradient descent, Adam, Adagrad and RMSProp. Overall, we found RMSProp to work best (in terms of resulting performance and also for speed of convergence) and most consistently.

3.4. Evaluation

The evaluation scheme for the Caltech Pedestrians dataset is extensive but focuses on bounding boxes as the unit of measure for the extent of pedestrians. Although our approach specifically foregos bounding box estimation, we still want to estimate our performance relative to other approaches applied to this dataset.

To do this we use the ground truth bounding boxes for an input image and, with the output from the heatmap layer, perform a voting procedure to estimate the bounding box predictions. This process allows us to estimate the true positive (TP), false negative (FN) and false positive (FP) rates for pedestrian detection, which then allows us to estimate the log average miss rate of our method. The log average miss rate is a measure of miss rate (false negative rate) versus the false positives per image (FPPI) over a range of FPPI values (evenly spaced in the log domain between 0.01 and 1). To obtain a set of miss rate values, we apply a range of thresholds over the range [-1, 1] to our output layer.

By introducing a number of parameters to control this estimation, we can obtain a kind of performance envelope with the 2 extremes corresponding to a difficult evaluation and an easy one. These parameters include:

- P_{OBB}: The percentage overlap between a gridbox and a ground truth bounding box
- P_{ACT}: Percentage of gridboxes belonging to a ground truth bounding box which must be on to consider the bounding box detected
- B_{INDIV}: Whether to count activations not within ground truth bounding boxes individually or to use 8-way connected components
- B_{OCC}: Whether to use the full bounding boxes or the unoccluded portion of each for the ground truth
- B_{IG}: Whether to use 'ignore' ground truth bounding boxes which fulfill certain criteria, including a minimum height of 20px for ground truth bounding boxes, filtering 'people' and 'person?' annotations and ignoring bounding boxes truncated by image boundaries.

Ideally, the most difficult parameter setting should yield performance comparable to the best performers on the Caltech benchmark. The evaluation process proceeds as follows (illustrated with concrete dimensions for clarity). For each desired activation threshold at the output layer, $T \in [-1,1]$, and the next 640x480 input image, I:

1. Extract the desired set of 227x227 grid overlays from I, as in Figure 4, to get a set of overlay regions, R.
2. For each overlay region, R_k, remap the ground truth bounding boxes in I relative to each overlay region. This produces a set of bounding boxes, $GTBB_k$ for each overlay region.
3. Intersect the 13x13 grid in each overlay region with the remapped ground truth bounding boxes. Keep the gridboxes whose percent overlap with each ground truth bounding box exceeds P_{OBB}. These are the ground truth target gridboxes, $GTTG_{k,n}$, and each overlay, k, will have some number of sets of resulting target gridboxes (one set for each bounding box in $GTBB_k$) whose number we index as a single 'n' here for simplicity.
4. If B_{IG} is 'on', the bounding boxes which don't satisfy the mentioned criteria and are meant to be ignored have their target gridboxes computed in a similar manner and are aggregated into a set, GTIG. Otherwise we leave GTIG empty.
5. For each overlay region, k, apply the current threshold T to our network output to produce a set of gridboxes of where pedestrians are predicted to be. Call this set, ACT_k.
6. For each overlay, k, iterate each set of target gridboxes in $GTTG_{k,n}$ and compute the set intersection with ACT_k. This is computing which gridboxes for each ground truth bounding box are predicted 'on' by our network. If the percentage is greater then P_{ACT} we count this as a TP. Otherwise it is a FN. Remove the current set of target gridboxes from ACT_k.
7. The remaining gridboxes in ACT_k, are those for which our network activated but were not in a target bounding box. We remove entries from this set which are in GTIG to yield $ACT2_k$. These are all of the gridboxes which the network activated for that were either incorrect predictions or were in an ignore region.
8. If counting individual entries (B_{INDIV} is 'on'), the size of set $ACT2_k$ is the number of FP's otherwise we use connected components to group the activations and count the number of components as the number of FPs.

Note that after removing each set of target gridboxes, $GTTG_{k,n}$, from ACT_k in step 6, we may be left with a number of gridbox activations along the perimeter of the ground truth bounding box. The more our model confines its output to each bounding box region the less of a problem this kind of overestimating of pedestrian location will be. How much of an impact this has can be controlled by whether connected components are used or by controlling P_{ACT}; both when performing the evaluation and also when we are generating the training data and determining the target activations for an input image. As our approach is not meant to distinguish between occluded and unoccluded pedestrians, we must be careful when comparing the result to the benchmark. This is because the evaluation is based on the full bounding boxes and not just the unoccluded portions. To be fair we evaluate performance on both sets of bounding boxes (using the B_{OCC} parameter) and show a small performance difference between the 2 cases.

4. Experimental Results

4.1. Runtime Performance

This section details the run-time evaluation we performed on our models. We use TensorFlow [3] for our model training and benchmark our model on 3 different systems, one of which is an embedded architecture:

- A laptop with an i3-6100U @ 2.3GHz, 16GB of RAM and a Samsung 850 EVO SSD
- A desktop with an i7-4790 @ 3.6GHz, 24GB of RAM, a GTX Titan X and a Samsung 850 EVO SSD
- S32V234 automotive processor

The S32V234 is a high-performance, ultra low power, automotive grade processor which supports a range of applications in vision and sensor fusion. It includes Quad ARM Cortex-A53 cores @ 1GHz, 4MB of on chip system RAM, 3D GPU and Dual APEX-2 image cognition processor cores. We evaluate and report the overall frame-rate of our model on the APEX-2 processors. In Table 3, we give the observed performance on the laptop and desktop when forward propagating 3 samples (from the '3pos' sampling scheme) which is effectively a 681x227 input region. Note that there is overhead in our unoptimized implementation. As such we show both the raw frames/sec which is the time for forward propagating the 681x227 region through our full network and also the full frames/sec which include overhead processing time of extracting and preparing frames and other overhead of the processing code.

	Laptop	Desktop	Dual APEX-2
Time (ms) / 3 patches	96.4	4.9	/
Raw Frames / sec	10.3	205	/
Time (ms) / frame	115.1	13.8	33
Full Frames / sec	8.6	72.4	30

Table 3: Summary of frames per second performance

When evaluating the '3pos' scheme, with a non BN network, on the APEX-2 processor, we observe a frame rate of 15FPS on one processor and 30FPS on both processors. The addition of BN is expected to have a negligible impact on performance, allowing for the same framerate. For our application, the entire S32V234 SoC requires about 2W with approximately 800mW for both APEX processor cores. This power is achieved because all CNN inference computations are performed by the APEX processors at 30 FPS, with minimum control by the ARM core, and the GPU is not in use in this case.

With regard to automotive safety, the current implementation is not 'aliasing' any intermediate tensor buffers. For processing a single patch, the overall required memory is 1.86 MiB which fits into the 4 MiB SRAM of on-chip memory. We note the APEX vector processing unit usage is 86%. Our GPU implementation was tested with 32-bit floating point weights and activations while our S32V234 implementation was quantized to 8-bit for everything (input, output and weights) except the tanh activations which were left as floating-point. If the desired threshold is small enough, the tanh could be substituted with a linear approximation allowing it to be 8-bit as well.

4.2. Detection Performance

Our target reference point is around 65% LAMR, putting us in the middle of the 'Overall' benchmark on the Caltech Testing data as shown in Figure 5. This includes, for example, RPN+BF at a LAMR of 65% which is a region proposal network followed by a boosted forest. Although our approach is not yet able to match the performance of SA-FastRCNN (a scale aware fast-RCNN) at 63% [20] and MS-CNN (multi-scale CNN) at 61% [21], we note that these approaches rely on significantly larger models with orders of magnitude more parameters and framerates lower by several multiples.

In general we found that increasing the dropout rate (we experimented with 0-50%) mainly helps as we began to sample positive and negative frames more frequently to build the training dataset (positive frames <= every 5th frame and negative frames <= every 10th frame). This makes sense as an increase in data redundancy should be helped by a larger dropout rate. However, in general the performance when oversampling in this manner was generally poor regardless of whether BN was used or not.

The effects of large batch sizes were observed to be negative. Going up from 32 samples per batch generally meant a small increase in training time while increasing batch sizes beyond 64 (up to 128) meant increasing the time to convergence and decreasing the performance of the best model achieved by about 8%.

Without BN, the training is sensitive to class imbalance between positive and negative frames. The number of negative frames in this case should be about 1/2 to 2/3 of the number of positive frames. This may have to do with the large number of target gridboxes which end up being off with increasing numbers of negative frames. We don't observe this class imbalance issue when using BN. When using shared weights at the partially connected layer, the best network was found to be just over 1% higher LAMR than the best network trained without shared weights.

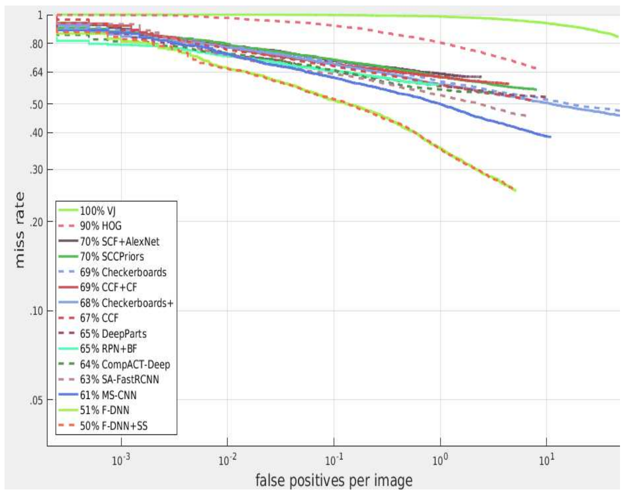

Figure 5: Top 13 (+ VJ and HOG) Results on the 'Overall' Caltech Pedestrian Testing Dataset [22]

We generally found training to be fast, requiring no more than 10 epochs to converge (training time of ~10-20min) when using 45K 227x227 training patches. In general, the training time when using BN was 3-5X faster than without, also allowing for a wider set of learning schedules to be used. We observe the performance curve shown in Figure 6 when running a batch normalized network on the test set, trained using:

- A dropout rate of 40%
- A dataset generated using a positive frame sample delay of 6 frames, a negative frame delay of 20 (sampling scheme 2 for training data) and a 50% P_{OBB} value when generating the data
- An exponential learning rate of 0.02 decayed at 0.9 every epoch
- Using the full ground truth bounding boxes (instead of taking the unoccluded subregions) as our target
- Sampling scheme '6hilo'

We can see in both cases, as we restrict P_{OBB} to be larger, a smaller number of gridboxes cover the perimeter of the ground truth bounding boxes. Note that switching to the unoccluded subregions of ground truth bounding boxes as the target results in a decrease of the LAMR by ~0.01 for each P_{OBB} value. Thus using the harder case of full bounding boxes does not have much impact on performance. For P_{OBB}=2/3, with B_{INDIV} set to True, we have a LAMR of 0.712. In general, decreasing P_{ACT} below 50% gives an increasing boost in performance. For example, decreasing to 35% for the same network with P_{OBB}=2/3 and B_{INDIV} on (original LAMR of 0.721) gives a LAMR of 0.7009 while at 25% it is 0.6764. We aim to keep this threshold at 50% in the spirit of the evaluation benchmark. When using the 3pos or 9pos sampling schemes, at P_{OBB}=2/3 (B_{INDIV} on) we see a LAMR value of 0.7281 and 0.7238 respectively. A performance curve is shown in Figure 7 for P_{OBB}=2/3 and B_{INDIV} on.

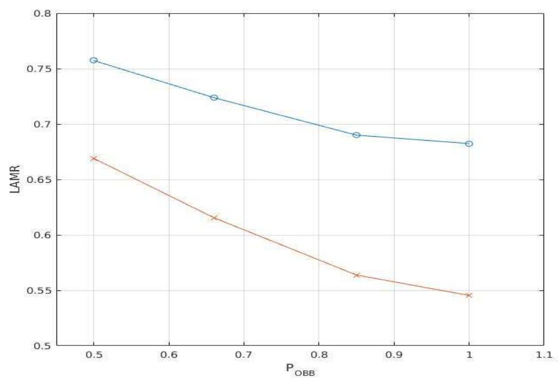

Figure 6: Plot of P_{OBB} values and the corresponding LAMR rate observed on the testing set with B_{INDIV} on (top/blue line) and using connected components/B_{INDIV} off (bottom/red line) for sampling scheme '6hilo'

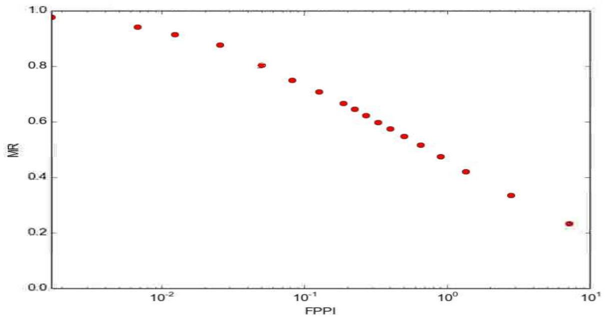

Figure 7: Example LAMR performance curve with P_{OBB}=2/3 and B_{INDIV} on

We show some example in Figures 8, 9 and 10, including results for the test dataset and a frame from the (unseen) ETH dataset for sequence LOEWENPLATZ. Note the variation in scale which seems to be captured quite well. Overall, we notice consistent performance in negative regions of the image, showing non pedestrian patterns are captured well. We do notice some moderate difficulty in regions such as trees or at the edges of cars (where people getting out of cars tend to be). The blue region is the sampling region generated by the '3pos' sampling scheme and the red boxes are the active gridboxes which have detected a pedestrian.

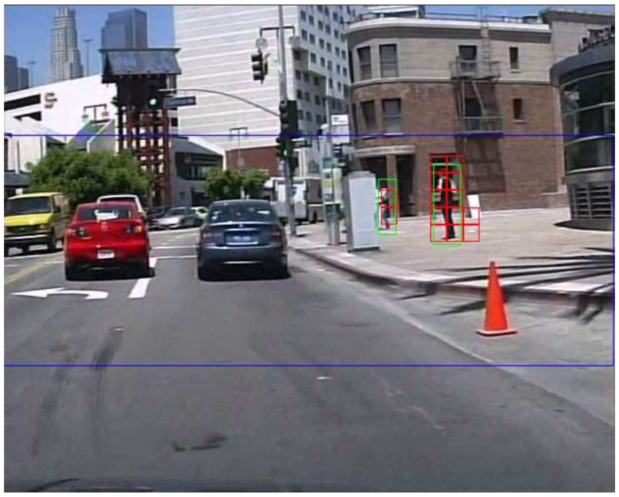
Figure 8: Sampled frame from the Caltech-USA testing dataset

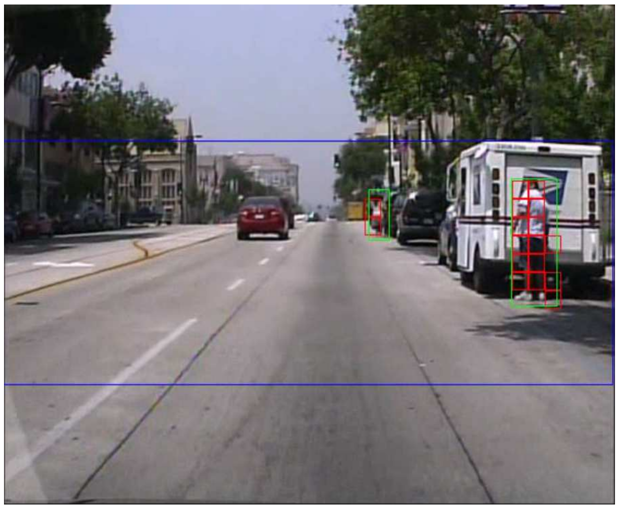
Figure 9: Sampled frame from the Caltech-USA testing dataset

5. Future Work

We've mentioned majority pooling as an extension, however the immediate goals are to incorporate scale information and to compute heatmaps for multiple classes (cars, pedestrians, road signs, etc.). Scale information can be incorporated by heatmap layers for a particular class connected at different depths in the network, each responsible for a particular scale. On the other hand, multiple classes could be incorporated by simply having multiple heatmaps at the same layer. The former extension should bring about a decrease in overall log-average miss rate while the latter extension will expand the models capability.

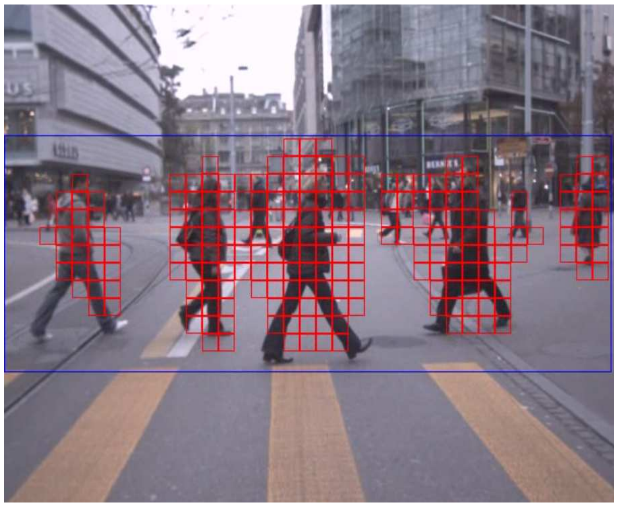
Figure 10: Sampled frame from ETH LOEWENPLATZ

6. Conclusion

In this work, we introduce an extension to the SqueezeNet architecture for performing pedestrian detection. This extension uses partially connected neurons to mimic a weighted voting scheme for the location of pedestrians, resulting in a kind of heatmap of their location. In the process, we end up with a small 3.24MB model which can run in real-time at 30FPS on an automotive processor operating within a 2W envelope. The performance of the model is comparable to state of the art models on the Caltech pedestrian testing dataset.

References

[1] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer. SqueezeNet: Alexnet-level accuracy with 50x fewer parameters and <0.5MB model size. arXiv:1602.07360, 2016.

[2] P. Dollár, C. Wojek, B. Schiele and P. Perona. Pedestrian Detection: An Evaluation of the State of the Art. In PAMI, 2012.

[3] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, R. Jozefowicz, Y. Jia, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, M. Schuster, R. Monga, S. Moore, D. Murray, C. Olah, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[4] J. Redmon, S. K. Divvala, R.B. Girshick, and A. Farhadi. You Only Look Once: Unified, Real-Time Object Detection. In CVPR, 2016.

[5] N. Dalal, and B. Triggs. Histograms of Oriented Gradients for Human Detection. In CVPR, 2005.

[6] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite. In CVPR, 2012.

[7] A. Ess, B. Leibe, and L. Van Gool. Depth and appearance for mobile scene analysis, in ICCV, 2007.
[8] J. Deng, W. Dong, R. Socher, L. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-scale Hierarchical Image Database. In CVPR, 2009.
[9] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In NIPS, 2012.
[10] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. In CVPR, 2014.
[11] P. F. Felzenszwalb, R. Girshick, D. McAllester, and D. Ramanan. Object Detection with Discriminatively Trained Part-based Models. In PAMI, 2010.
[12] R. Girshick. Fast R-CNN. In ICCV, 2015.
[13] J. Long, E. Shelhamer, and T. Darrell. Fully Convolutional Networks for Semantic Segmentation. In CVPR, 2015.
[14] S. Ren, K. He, R. Girshick, and J. Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In NIPS, 2015.
[15] P. A. Viola and M. J. Jones. Robust real-time face detection. Intl. Journal of Computer Vision, vol. 57, no. 2, pp. 137–154, 2004.
[16] NXP Research, S32V Whitepaper. http://cache.freescale.com/files/ automotive/ doc/ white_paper/ S32V230WP.pdf
[17] K. He and J. Sun. Convolutional neural networks at constrained time cost. In CVPR, 2015
[18] S. Han, H. Mao, and W. Dally. Deep compression: Compressing DNNs with pruning, trained quantization and huffman coding. arXiv:1510.00149, 2016.
[19] R. Beneson, M. Omran, J. Hosang, and B. Schiele. Ten Years of Pedestrian Detection, What Have We Learned? In ECCV, 2014.
[20] J. Li, X. Liang, S. Shen, T. Xu, J. Feng, and S. Yan. Scale-aware Fast R-CNN for Pedestrian Detection. arXiv:1510.08160, 2015.
[21] Z. Cai, Q. Fan, R. Feris, and N. Vasconcelos. A Unified Multi-scale Deep Convolutional Neural Network for Fast Object Detection. In ECCV, 2016.
[22] P. Dollar. Caltech Pedestrian Benchmark. http://www.vision.caltech.edu/Image_Datasets/CaltechPedestrians/