Accelerating neural networks embedded on a microcomputer trained on KITTI.

Arnoud Jonker

Master of Science Thesis

TU Delft Delft University of Technology

Department of Cognitive Robitics

Accelerating neural networks embedded on a microcomputer trained on KITTI.

Master of Science Thesis

For the degree of Master of Science in Vehicle Engineering at Delft University of Technology

Arnoud Jonker

4170431

April 7, 2020

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

Delft University of Technology
Department of
Cognitive Robotics (CoR)

The undersigned hereby certify that they have read and recommend to the Faculty of Mechanical, Maritime and Materials Engineering (3mE) for acceptance a thesis entitled

Accelerating neural networks embedded on a microcomputer trained on KITTI.

by

Arnoud Jonker

in partial fulfillment of the requirements for the degree of

Master of Science Vehicle Engineering

Dated: April 7, 2020

Supervisor(s):

Dr. ir. W. Pan

Reader(s):

Dr.ir. R. Happee

ir. H. Zhou

Abstract

Due to a high contribution of human error in fatal traffic accidents, efforts in research and industry for automating vehicles steadily increased the last 3 decades. To reduce accidents with other road users, automated recognition and classification of road users is crucial. The most accurate models are large convolutional neural networks, however they require expensive and powerful hardware to be implemented. The hardware requirement hinders wide embedded use in education and industry, slowing down the potential increase of safety on the road. In pursue of reduced inference time on GPUs, research has focused to reduce the computational demand of neural networks. However the accelerations on microcomputers are not reported, potentially leading to different results.

To contribute to increased insight of the applicability of previous research on microcomputers, this thesis compares accelerations on a microcomputer and GPU. This is pursued by using the pretrained neural network "Squeezedet", trained on the "KITTI" road user dataset. The neural network is pruned in 3 different ways: by reducing the number of filters, by reducing the size of filter kernels along a layer and by reducing the amount of layers. The accelerations are measured on a Raspberry Pi 3b+ and a Tesla K80 GPU. The process of embedding the neural network on the Raspberry Pi is described in great detail, to promote further research and educational use.

Acceleration of the network is up to 12 % better on a Raspberry Pi than a GPU, when pruning filters from the network. When pruning full layers, or decreasing filter size the accelerations are up to 4% worse than on a GPU. This means when working with a microcomputer, the choice of pruning method can not be based on reported accelerations on GPUs. Therefore the recommendation is to do more research on accelerating neural networks for microcomputers. This will lead to wider use in industry and education and eventually safer roads.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1-1 Automated vehicles

The potential of automated vehicles capable of reducing traffic accidents is widely accepted in industry and research [8]. One of the reasons for this potential is the high contribution of human error in fatal accidents. A study performed in the United Stated showed 94% of fatalities on the road are estimated to be caused by human error [9]. To take the human out of the loop in controlling vehicles the efforts for developing automated vehicles have steadily increased in the last three decades [10]. One of the factors fueling this development is the potential of increasing safety on the road. A report on traffic safety statistics by the European Commission [11], showed a decrease of 18% in traffic deaths from 2010 until 2015, continuing a longer trend. However a worrying conclusion was drawn as well. Vulnerable road users like pedestrians and cyclists showed a decrease of 11% and 8%, which is low compared to the mean decrease of 18%. Where the goal of the European Union is to promote green and healthy ways of transport, the safety of these vulnerable road users should have extra attention they concluded. One of the aspects to interact with road users is the ability to correctly identify the road user using the sensed information. According to [11] 69% of the accidents concerning vulnerable road users happen in urban areas. This leads to the challenge to be able to accurately identify road users, especially in complex in urban areas.

1-2 Neural networks

A survey showed about pedestrian detection showed the best performing techniques in terms of accuracy originate from deep learning [12]. The revolution in image recognition caused by deep learning started with the presentation of the AlexNet [13], now cited over 40000 times. They introduced a convolutional neural network architecture in the 2011 ImageNet competition, being able to classify images in 1000 categories. Their approach resulted in a top1 error of 36.7% and top5 error of 15.3%, the second best contender had a top5 error of 26%. To achieve these results they created a network with 60 million parameters, while dealing with

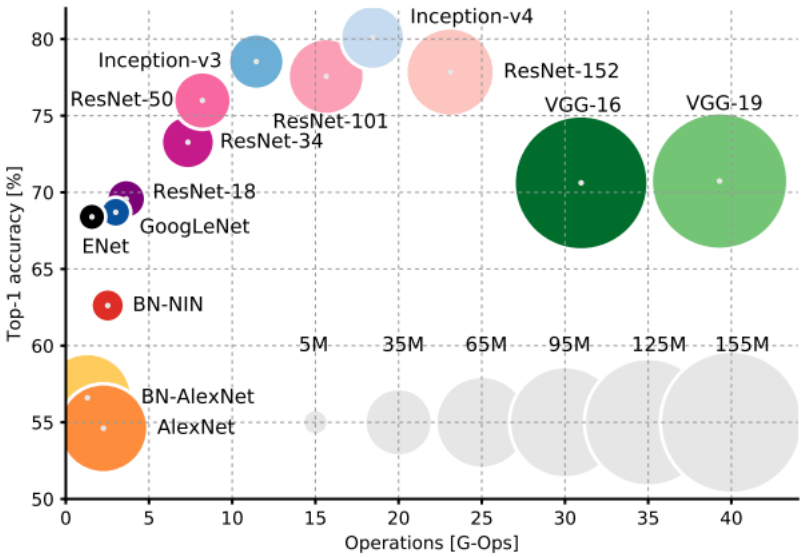

Figure 1-1: Top1 accuracy vs operations vs size in million parameters. 14 state off the art neural networks, with their ImageNet top1 accuracy, the amount of operations of 1 forward pass and the amount of parameters in the network. A larger or more computational complex network does not guarantee a better accuracy.

overfitting with dropout and long training times by applying efficient convolution operations on a Graphical Processing Unit (GPU). Since the AlexNet more convolutional neural network architectures have been presented. As seen in figure 1-1 other networks have reached even higher accuracy's, with varying needs of operations and varying amounts of parameters. So while demanding varying amounts of computational load and memory, convolutional neural networks keep proving themselves as accurate networks for classifying images.

However, according to the survey [12], the techniques origination from deep learning also lead to an inference frequency below 15 frames per second (fps), which is too low. The article "Scaling for edge inference of deep neural networks" [14] discusses the gap between the capacity of hardware and the growing computational demand of deep learning algorithms. Data from 2012 until 2017 shows that the hardware does not meet the demands on energy efficiency and computational density. Another study on the implications of hardware in automated vehicles [15], showed a driving range reduction of 3 to 12% by the computing engine including the cooling. The range reduction depended on the type and amount of processors. They concluded that after hardware acceleration computational capability remains the bottleneck, so a reduction in computational demand is necessary. So to make use the potential of deep learning in an Automated vehicle (AV), leading to a reduction in accidents, a decrease in the computational demand of deep learning algorithms is needed.

1-3 Compression for embedded neural networks

The survey "A Survey of network Compression and Acceleration for Deep Neural Networks" [12] cited over 60 works on network compression and acceleration. More than 90% of these

works are published after 2010. Next to pruning, research also focuses on other techniques, such as quantization and low-rank factorization. From the cited work in the survey focused on pruning only 2 studies show their results on mobile devices. One of the conclusions of the survey therefore was: "Hardware constraints in various of small platforms (e.g.,mobile, robotic, self-driving car) are still a major problem to hinder the extension of deep CNNs. How to make full use of the limited computational source and how to design special compression methods for such platforms are still challenges that need to be addressed." [12]. So more knowledge needs to be gained on the effect of compression methods like pruning, when deploying networks on small platforms.

1-4 Thesis objectives

Following from the previous sections there is a clear need for more knowledge about the performance of pruned neural networks on small platforms. In automated vehicles neural networks can contribute to effective recognition of road users. Deploying neural networks with state of the art accuracy leads to problems with resources in the vehicle. To contribute to better road user detection in automated vehicles and ultimately to less accidents with vulnerable road users, the thesis has the following 2 objectives:

- To compare the acceleration of pruned neural networks for road user detection on a microcomputer in with the acceleration on a GPU.
- To support an increase in use of microcomputers in research, education and industry.

The thesis will revolve around an experimental setup, pruning neural networks in different manors, to be able to compare the results on the different devices. Next to that, to contribute to the second objective, the methods for deploying and testing on the microcomputer will be described in high detail. The thesis will answer the following questions:

- Does pruning different structures from a convolutional neural network lead to the same acceleration in road user detection on a microcomputer as a GPU?
- Which structure is the best to remove for accelerating a neural network on a microcomputer?

1-5 Thesis outline

First in this thesis, in chapter 2, all background of used theories, network and methods is explained. Also the dataset used for this thesis, KITTI, will be discussed. In chapter 3 the methods for the experiment are explained. This consists of the used methods for pruning, an explanation on the SqueezeDet model, the preprocessing of the KITTI dataset and how to embed the networks on the microcomputer. In chapter 4 the results are shown of the pruning methods and the accelerations on the microcomputer and GPU. In this chapter the results will also be discussed. In chapter 5 The conclusions and recommendations of this thesis are found.

Chapter 2

Background

This chapter will give background on used theory and techniques in this thesis. In section 2-1 the foundations of the Deep Neural Network (DNN) are explained. In section 2-2 the used neural network for this thesis SqueezeDet will be explained. In section 2-3 methods for compressing and accelerating neural networks are discussed. In section 2-4 the used dataset is introduced and analysed.

2-1 Deep learning

Since the DNN Alexnet [13] has been published as winner of the ImageNet [16] competition of 2012 DNNs have been the state of the art when in comes to image classification [5]. Since the AlexNet has been published more DNNs have been published, like VGG-16 [17], ResNet [18] and GoogLeNet [19] increasing the accuracy of image detection and introducing object detection through the years that followed. In this section the theory of the deep neural networks used for object detection is explained.

2-1-1 Network building

The DNN used for object detection is a Convolutional Neural Network (CNN). Convolutional neural networks inhibit their name for the convolutional operation used throughout the network. In a CNN also other types of operations can be incorporated , like pooling operations and fully connected layers.

Neuron

A neural network, as the name implies, is a network of neurons. So first the neuron itself is explained. A neuron can have an infinite amount of inputs and has one output. All the inputs are multiplied by a corresponding weight. Also each input may have an added bias.

The neuron sums all these inputs and then applies an activation function. In 2-1 a visual representation of a neuron is shown. Before the activation function each neuron outputs a linear combination of the inputs. The activation function can introduce non-linearaties. The activation function determines when a neuron should fire, or in other words should propagate a value to the next neuron. Depending on the activation function also the height of the propagated value is determined. The choice of activation function can lead to a difference in accuracy of factor 4 [20].

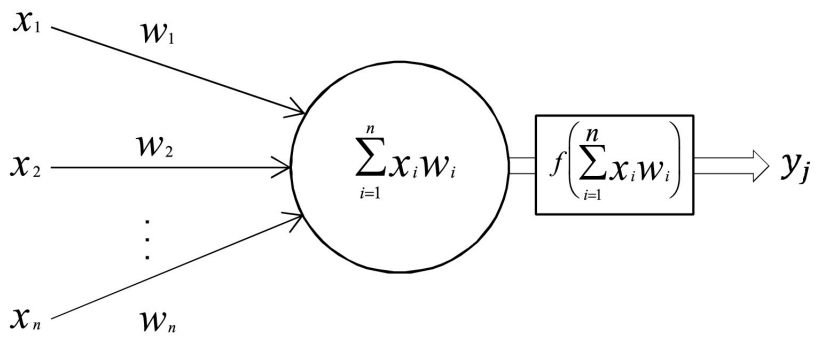

Figure 2-1: One neuron as used in a neural network. Before the neuron each input is multiplied by a corresponding weight, then the results are added together. On the result of the summation a function is applied called the activation function. The output of this function is the output of one neuron.[1]

Fully Connected layer

A fully connected layer in a neural network connects all outputs of the previous layer to each neuron in the current layer. Each connection between 2 neurons resembles a trainable parameter in form of a weight and optional bias. In figure 2-2 3 fully connected layers are shown to visualize this type of layer. A fully connected layer is typically used to classify data based on extracted input features. However research has shown that for object detection fully connected layers may not be necessary for classifying detected objects. [21]

Convolutional layer

A convolutional layer in a neural network has a convolution operation at its core. In the layer a set of filters is sliding over the input, where each filter produces a feature map as output. When convolutional filters are applied on image data, the input of the layer typically is a stack of channels with 2 dimensions. For the input layer this could be the 2 dimensional image with 3 input channels, being the red green blue (RGB) values. The convolutional filter will consist of a stack of kernels, equal to the number of channels from the input. The width an height are free to choose, typically between 1 and 10. The filters slides along the input width and height, producing a value for each location and mapping this to a feature output. The number of convolutional filters determines the number of feature output maps, which will be the number of input channels for the next layer. In this explanation of a convolutional layer the neurons are in the feature maps and are connected via the filters. The filters therefore are the weights

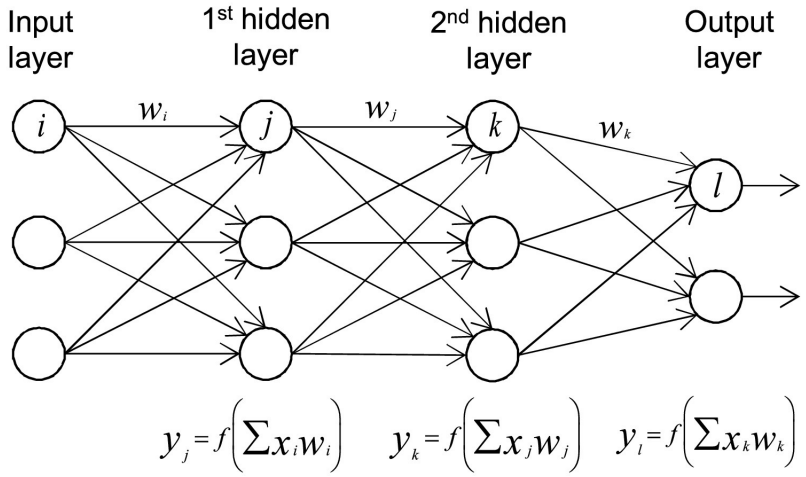

Figure 2-2: In the figure a neural network with 2 fully connected hidden layers are shown and one fully connected output layer. The name hidden layer is used for layers between the input and output. The 2 hidden layers and output layer are fully connected because eacht of the neurons are connected to all neurons of the previous layer. [1]

and resemble the way the neurons are connected. Each neuron will be connected to a part of the input, however all neurons across one feature map share the same weights as they all use the same filter. In figure 2-3 a schematic view is given on the working of one convolutional filter.

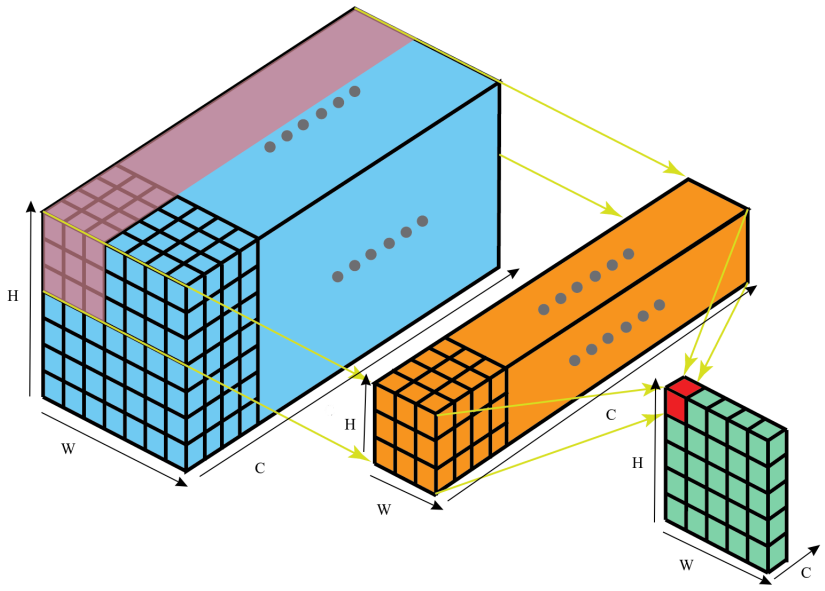

Figure 2-3: The image shows a schematic interpretation of a convolutional operation. The blue box represents the input for the convolutional layer, which usually is the network input or feature map of the previous layer. The input has a width W, height H and C channels. The convolutional filter is shown in orange. The filter is a stack of C kernels, aligned with the input channels. The kernels have their own height H and width W. The green box represents the feature map of the convolutional filter. The filter slides along the input and gives one value for each location, resulting in a feature map with a height and width depending on the layer input height and width and filter height and width. The number of channels of the feature map depends on the number of filters in the convolutional layer. [2]

Convolutional layers in object detection networks are used to extract features from the input. The feature extraction therefore is incorporated in the training of the network. The training is explained in 2-1-2. When the convolutional layer has been trained the filters can be visualized. The visualization resembles the trained weights of the filter. In figure 2-4 a set of trained kernels from a filter is shown. This is from an early layer in the network, where the network will learn basic edge and pattern detectors. The ability of these layers to learn feature extractors, together with neurons sharing each feature extractor along the whole image dimension make convolutional layers a powerful tool in object detection.

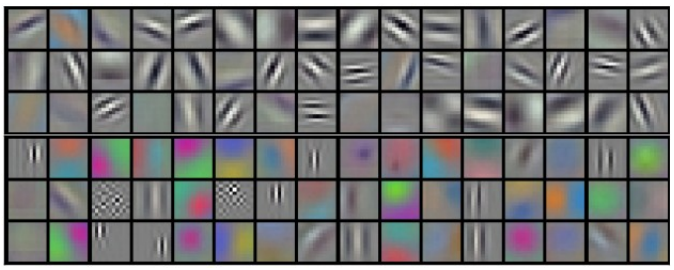

Figure 2-4: The image shows multiple kernels from a filter out of a convolutional layer. This visualization of the kernels gives insight in how a convolutional layer learns to extract features. [3]

Pooling layer

Pooling layers in a neural network are used for reducing the spatial resolution of the input from the layer. In other words this means that the height and width of the layer input are scaled down. The pooling layer slides a pooling region of $k \times k$ over the input with a stride s. The stride determines the step size when sliding the pooling region and therefore the reduction in resolution. A stride of 2 results in half the spatial resolution. From the pooling region for example the average of the input values or maximum input value can be chosen, resulting respectively in average pooling and max pooling.

2-1-2 Network training

Training the neural network starts with a forward pass through the network with an input. This results in the output. Then the neural network output is compared with the desired output, called a label. A value, representing the error, is calculated with a loss function. Then using this error, the network weights are updated using back propagation.

Forward pass

In the forward pass the output of the network is calculated. From the input to the output of the network, the output of each neuron in calculated as explained in section 2-1-1. This results in an output of the network. This output is used to calculate the loss.

Loss function

The loss function uses the output of the forward pass and the desired output called the label to produce one value. This value represents how wrong the network is, so a lower value is a better performing network. An example of a loss function is Cross Entropy (CE). CE can be used for classification problems, where the class is represented by a vector of zeros and a one. The location of the one in the vector indicates the class. For M classes the CE loss is given by equation 2-1. The loss function has to be chosen specified on the application of the network.

$$L_{ce} = -\sum_{c=1}^{M} label_{o,c} \log(pred_{o,c}) \tag{2-1}$$

Back propagation

Based on the value of the loss function the weights are updated in the network. The weights need to be updated so the loss function is minimized. To update the weights the gradient in equation 2-2 is computed.

$$\frac{\partial L}{\partial w}, \frac{\partial L}{\partial b} \tag{2-2}$$

When the weights are updated in the direction of the negative gradient, the loss function will decrease. To calculate the gradient for each weight in the network, the error is propagated from the back of the network to the front. This enables the reuse of variables by applying the chain rule. This is best explained by an example. In figure 2-5 2 graphs of the same neural network are shown. This network is used to illustrate the backpropagation.

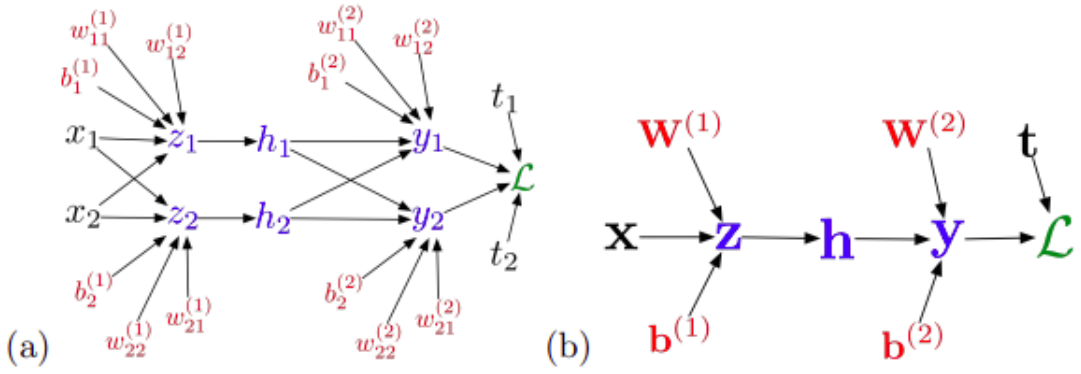

Figure 2-5: 2 graphs of the same neural network, (a) is written out completely for reference of (b).(b) is the vectorized version. x is input, w is weights, b is biases, z is the first layer, h is output of z, y is the output of network, t is the label and Λ is the loss. [4]

The loss function in this example is half of the squared error. When doing the forward pass, the values of all nodes and the loss function are described in equations 2-3 to 2-6.

$$z_i = \sum_j w_{ij}^{(1)} x_j + b_i^{(1)} \tag{2-3}$$

$$h_i = \sigma(z_i) \tag{2-4}$$

$$y_k = \sum_i w_{ki}^{(2)} h_i + b_k^{(2)} \tag{2-5}$$

$$L = \frac{1}{2} \sum_k (y_k - t_k)^2 \tag{2-6}$$

Then by applying the chain rule, the derivative of each node in the graph is a function of the forward pass values of its parent node and the derivative of its child node. So to calculate the derivative of a node, the derivative of its child node is required. This means you have to start at the final child of the network, in this case the L node. From there we can propagate the derivatives, to calculate all the derivatives. When using the bar notation from R. Grosse [4], $\frac{\partial L}{\partial w}$ becomes $\bar{w}$. With this notation the backpropagation through the network is shown in equations 2-7 to 2-14.

$$\bar{L} = 1 \tag{2-7}$$

$$\bar{y_k} = \bar{L}(y_k - t_k) \tag{2-8}$$

$$\bar{w_{ki}^{(2)}} = \bar{y_k} h_i \tag{2-9}$$

$$\bar{b_k^{(2)}} = \bar{y_k} \tag{2-10}$$

$$\bar{h_i} = \sum_k \bar{y_k} w_{ki}^2 \tag{2-11}$$

$$\bar{z_i} = \bar{h_i} \sigma'(z_i \tag{2-12}$$

$$\bar{w_{ij}^{(1)}} = \bar{z_i} x_j \tag{2-13}$$

$$\bar{b_i^{(1)}} = \bar{z_i} \tag{2-14}$$

After this backpropagation we have values for 2-2, so the weights can be updated. The gradient could be used to directly update the weights. However to optimize the training, certain optimization to this weight update can be applied.

Optimizations

For updating the weights and biases the calculated gradients are used, however the update of the weights can be further optimized. The size of the weight update is influenced by a hyperparameter, the learning rate. Each gradient is multiplied by this learning rate, before it is added to the weights and biases. This influences the step size in searching for the optimal combination of weights and biases. A low learning rate results in slow convergence and risk of being stuck in local optima. A high learning rate is of risk of not converging at all.

To run the update through the network by calculating the gradient for all the training examples is of high computational cost. To solve this the optimization function Stochastic Gradient Descent (SGD), also known as mini-batch gradient descent can be used. The gradients are calculated and averaged over samples of training data instead of the complete dataset. The size of the batch size influences the generalizibilty of the computed gradients as well, because more training data means a better representation of the data. SGD is the most used basis for optimizing the neural network. However extensions of this algorithm and different algorithms are available. An overview of these is discussed in [22].

2-2 SqueezeDet network

The network used for this thesis is the SqueezeDet [7]. The name is a contraction of the [5] and a layer they added to the network called "convdet". This section will discuss the architecture of the SqueezeDet in section 2-2-1 and the used training algorithm and parameters in section 2-2-2.

2-2-1 Architecture

SqueezeNet backbone

The backbone of the SqueezeDet network is the SqueezeNet. This network introduced a set of layers called the 'fire module'. Using this module the relatively small network SqueezeNet was created, containing 50x less parameters than the AlexNet [13]. The fire module incorporates a layer of 1x1 filters, meant to downsample the input of the layer. After this layer a set of 1x1 and 3x3 filters are connected to the downsampled input features of the layer. The downsampling 1x1 filter layer is called the squeeze layer and the connected set of 1x1 and 3x3 filter layer is called the expanding layer. The combination of using a lot of small 1x1 filters and downsampling leads to a light weight structure. For an illustration of the fire module see figure 2-6.

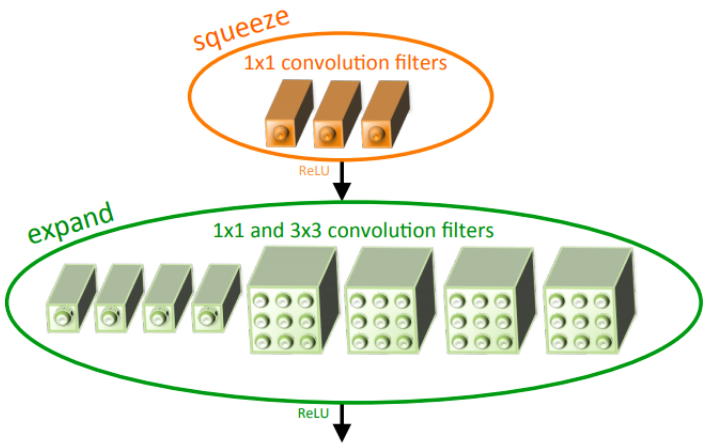

Figure 2-6: A fire module, as introduced in the SqueezeNet network. The squeeze layer downsamples the input features with trained 1x1 filters. The expand layer extracts features with 1x1 and 3x3 filters. The downsampling and partial use of 1x1 filters for feature extraction leads to a light module to build a network with. [5]

The SqueezeDet has been published in different sizes, based on different sizes of the SqueezeNet. The difference in sizes are dependent on the amount of filters in each layer. More information on the different sizes can be found in [5]. The SqueezeDet plus layout is shown in figure 2-7. The layout is equal to the SqueezeNet, except for the last layer, the ConvDet layer.

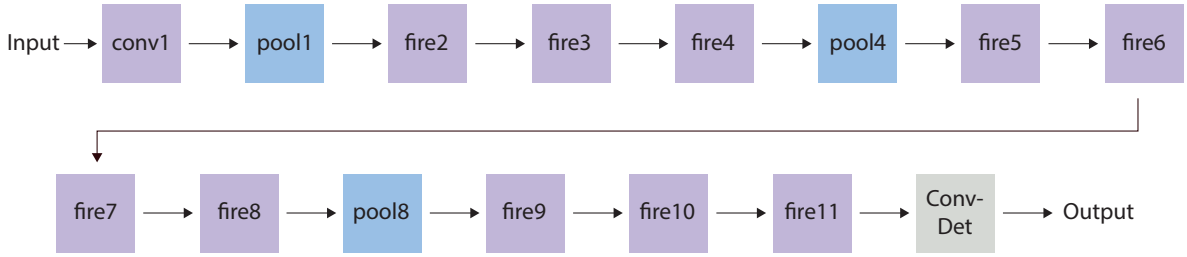

Figure 2-7: Overview of the SqueezeDet plus layout. The architecture is a combination of 11 fire modules, 3 max pooling layers and the final layer is the ConvDet layer.

ConvDet layer

In the published paper on the SqueezeDet a final convolutional layer was introduced to output boxes with classes and confidence scores, this layer is called the ConvDet layer [7]. This last layer of the network is developed in such a way that it can output proposals for bounding boxes for an object, a confidence score for a bounding box and a class for the object in the bounding box. The idea to let the same network in one step propose regions for objects and classify those regions comes from the You Only Look Once (YOLO) network [23]. Where the YOLO network used a fully connected layer to classify the bounding boxes, for the SqueezeDet the classification is also done by the convolutional layer proposing the regions. Before the YOLO network introduced region proposal and classification by one network a technique introduced by 'Faster R_CNN' was used [24]. Then the region proposal for a bounding box and classification of the region proposal was done by 2 separate networks.

The ConvDet layer works, just as a normal convolutional layer, like a sliding box over the input feature map. The ConvDet layer computes bounding boxes around a distributed grid of points. The grid has size $W \times H$ in horizontal and vertical direction. At each grid point $K \times (4 + 1 + C)$ values are given as output to a feature map. K is the number of different possible box shapes, this is defined by how the dataset is labeled. The number 4 resembles 4 scalars $\hat{x}_i, \hat{y}_j, \hat{w}_k, \hat{h}_k$, these describe the coordinate of where the box is located and the width and height of the box. For all these box proposals 1 value is added with a confidence score, describing the probability of containing an object of interest. Next to that all box proposals contain C scalars representing the conditional class probability of the content of the box.

The number of grid positions is dependent on the resolution of the input feature map. The feature map is a lower resolution than the input, due to resolution downsampling in the network. So the predicted values need to be upsampled again, to give the actual locations of the bounding boxes on the input image. To compute these locations four relative coordinates are computed $(\delta x_{ijk}, \delta y_{ijk}, \delta w_{ijk}, \delta h_{ijk})$. Using these relative coordinates the projected location on the original image are computed by equations 2-15 to 2-18.

$$x_i^p = \hat{x}_i + \hat{w}_k \delta x_{ijk} \tag{2-15}$$

$$y_j^p = \hat{y}_j + \hat{h}_k \delta y_{ijk} \tag{2-16}$$

$$w_k^p = \hat{w}_k + exp(\delta w_{ijk}) \tag{2-17}$$

$$h_k^p = \hat{x}_i + exp(\delta h_{ijk}) \tag{2-18}$$

2-2-2 Training

The SqueezeDet can be trained end to end, learning the object locations and classifications simultaneously. To train the network to output box locations, a confidence score and class the loss function in 2-19 has been defined.

$$
\begin{aligned}
L_{sq} =& \frac{\lambda_{bbox}}{N_{obj}} \sum_{i=1}^{W} \sum_{j=1}^{H} \sum_{k=1}^{K} I_{ijk}((\delta x_{ijk} - \delta x_{ijk}^{G})^2 + (\delta y_{ijk} - \delta y_{ijk}^{G})^2 + (\delta w_{ijk} - \delta w_{ijk}^{G})^2 \\
& + (\delta h_{ijk} - \delta h_{ijk}^{G})^2) + \sum_{i=1}^{W} \sum_{j=1}^{H} \sum_{k=1}^{K} \frac{\lambda_{conf}^{+}}{N_{obj}} I_{ijk}(\gamma_{ijk} - \gamma_{ijk}^{G})^2 + \frac{\lambda_{conf}^{-}}{WHK - N_{obj}} \bar{I}_{ijk} \gamma_{ijk}^{2} \\
& + \frac{1}{N_{obj}} \sum_{i=1}^{W} \sum_{j=1}^{H} \sum_{k=1}^{K} \sum_{c=1}^{C} I_{ijk} l_{c}^{G} log(p_c)
\end{aligned}
\tag{2-19}
$$

The part of the loss function multiplied by λ_{bbox} is to penalize bad predictions on bounding boxes. The bounding boxes are defined by relative coordinates to the grid point. This part of the loss is the sum of all boxes at each grid point ($W \times H \times K$). The ground truth (label), is calculated by taking the bounding box on the input image and translating it to the feature map input of the ConvDet layer. This is the inverse operation of equations 2-15 to 2-18. Only the closest grid point to a ground truth is responsible for detection, so I_{ijk} acts as a mask and assigns 1 to grid points responsible for a bounding box and 0 for grid points not responsible for a box. In this way the loss only rises if the closest grid point to a box is bad at predicting the box location. This part of the loss is divided by the number of objects in the image, to compensate for high penelization when more objects are in an image.

The part of the loss function multiplied by λ_{conf}^{+} and λ_{conf}^{-} is to penalize predicting a bad confidence score. The confidence score is defined as the probability that an object exists in the box times the Intersection Over Union (IOU) between the predicted box and ground truth box. The same I_{ijk} is used to only penalize confidence scores compared to a ground truth of grid points closest to a ground truth bounding box. All other confidence scores are scaled down by penalizing the confidence score squared, achieved by applying the opposite mask $\bar{I}_{ijk}$

The final term of the loss function is a cross entropy term to minimize the error on predicted classes. The classes output is normalized using softmax. Again only wrong predicted classes of grid points close to a ground truth bounding box are penalized.

The network is trained using back propagation with SGD. Momentum is applied to SGD for optimization. A learning rate with decay is applied as well. All hyperparameters for training can be found in table 2-1

Table 2-1: Parameters used for training the SqueezeDet

Parameter	Learning rate	Weight decay	Momentum	Learning rate decay	Learnin rate decay step
Value	0.01	0.0001	0.9	0.5	10000

2-3 Network compression and acceleration

A neural network can be compressed and accelerated during or after training [25]. Before this thesis a literature review has been done on various network compression and acceleration

techniques [26]. In section 2-3-1 a summary of different types of techniques will be given from the literature review. In sections 2-3-2 and 2-3-3 the background of applied theories for this thesis will be explained in more detail.

2-3-1 Literature review

The techniques for compressing and accelerating networks had been divided in 3 categories. The first is matrix factorization. A fully connected layer in a neural network can be represented by a matrix multiplication, and convolutional layers for 2d images are modelled by 4D tensors. The matrix multiplications can be sped up for example by low rank matrix factorization or convolution flattening. These methods have to work layer by layer and it has been shown to lack in giving stable results [25].
The second type is pruning. Pruning tries to remove parameters from the network. Many different methods have been published to find or create sparsities in a network, so parameters can be removed [27], [28], [29], [30]. Pruning gives the possibility to retrain the smaller structure, to regain possible lost performance.
The third type is quantization. Quantization is focused on saving the parameters with low precision. Quantization can be done after training, as well as during training [31]. Quantization can be taken to the level where a network only has binary parameters [32].

2-3-2 Structured sparsity learning

To achieve actual speedups in a neural network, parameters need to be removed from the network. This can be achieved by representing matrices in a sparse matrix form or to reduce the size of the original matrix. For a convolutional layer this means the 3 dimensional tensor representing the filter needs to be reduced in size [33]. In [33] these two approaches are compared. To search for structured sparsities, they apply group lasso on different structures. The filter can be reduced in size by pruning rows and columns from the kernel size of all filters in the layer. Also the amount of filters in one layer can be pruned by removing the filters. When filters are removed, connected channels and therefore connected kernels in the next layer can be removed as well. Finally they demonstrate removing a layer completely. Compared to speedups with unstructered sparsity, structured sparsity achieves 2 times higher acceleration.

2-3-3 Network slimming by filter factors

The paper 'Learning Efficient Convolutional Networks through Network Slimming' [6], introduces an efficient way to prune filters and their channels from a network. Each filter is multiplied by a factor γ. Then to the loss function a term is added with the l1 regularization over the filter factors. This means the l1-norm of all the filter factors is added to the loss. When the training is applied, this results in many low factors, implying their connected filters can be removed. In figure 2-8 a schema of the removal is shown.

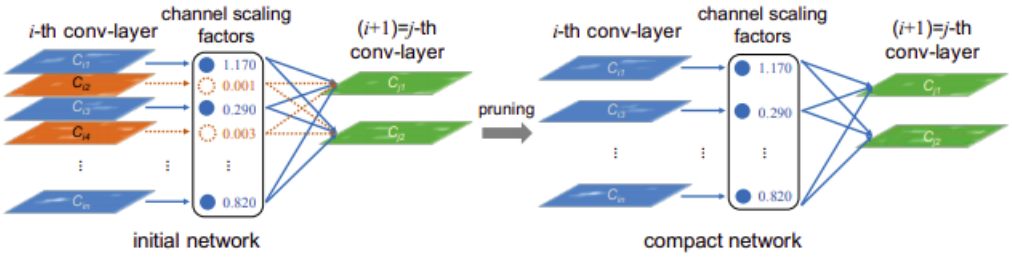

Figure 2-8: Schematic overview of applying scaling factors to convolutional layers. After pruning filters with a low factor, the compact networks has less filters. [6]

The loss function is shown in equation 2-20. The original loss is a function of the input, weights and label. To this a function g of the factors γ is added. The function g is the l1 regularization.

$$L = \sum_{(x,y)} l(f(x, W), y) + \lambda \sum_{\gamma \in \Gamma} g(\gamma) \tag{2-20}$$

This method had results ranging from 30 to 88.5 % pruned parameters on 3 different networks. The use of factors for their desired structure, the filters, can be easily applied to other structures as well. However they only applied it to pruning filters.

2-4 KITTI dataset

To train the network a labeled dataset is required. In this thesis the KITTI dataset [34] is used for training. The KITTI dataset is a dataset released by the Karlsruhe Institute of Technology (KIT) in cooperation with Toyota Technical Institute (TTI). They split the dataset into a training set of 7481 images and a test set of 7518 images. The training set can be downloaded, the test set is kept at the institute and is used to test detection models of published works. For unpublished work, they advice the split up the training set and test on that. In figure 2-9 four examples of the data are shown.

Figure 2-9: Four examples of data from the KITTI dataset. The environment where the images are taken differs from city centre, to suburbs to highway.

In the images the cars, cyclists and pedestrians are labeled with 16 values. In table 2-2 all the labels are listed. Each image can contain any number of objects. The dataset has been gathered using a car with cameras and laser scanners. Spatial information about the objects comes from the laser scanners. Regions in the image that have been missed by the laser scanner, because it was too far away have been labeled as 'DontCare'. These regions are not taken into account in the official test script. The distribution of classes is 80% cars, 4% cyclists and 16% pedestrians.

Table 2-2: The values describing each object in the image. From the objects information about the class, visibility and spatial information is available.

Name	Values	Description
Type	1	Word like 'Car', 'Cyclist', 'Pedestrian' describing class
Truncated	1	float [0,1] indicating if object is completely in the image
Occluded	1	Int(0,1,2,3) inidaction if object is occluded 0=fully visible, 1=partly occluded, 2=largely occluded, 3=unknown
alpha	1	float[-pi,pi] indicating the angle of object
bbox	4	2D bounding box of of object in pixel coordinates
dimensions	3	3D object dimensions in meters
location	3	3D object location in camera coordinates in meters
rotation_y	1	float[-pi,pi] Rotation around y axis
score	1	Float[0,1]. For results indicating confidence

Chapter 3

Methods

In this chapter a motivation and or description of the used methods is given. The descriptions will be given in code, pseudocode and or supporting mathematical equations. More information on the background and origine of the methods can be found in chapter 2. In the first section of this chapter the general experimental setup is discussed in the sections that follow the steps of the experiment will be discussed.

3-1 Experimental setup and overview

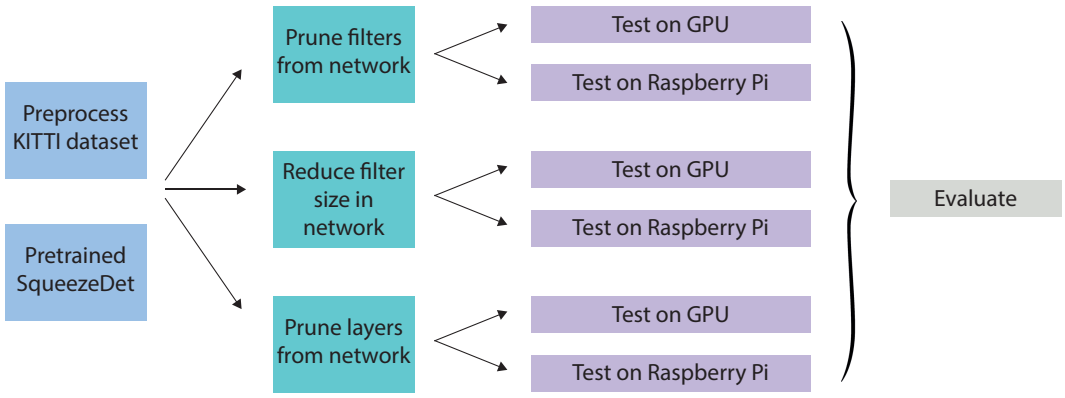

Figure 3-1: Overview of the experimental setup. The first step is to acquire and preprocess the pretrained model and dataset. Then using the data 3 different networks structures are pruned from the pretrained model. Then each model is tested on the Raspberry Pi and GPU. This is all compared and evaluated.

In figure 3-3 an overview of the experimental setup is shown. The experiment consists of 4 general steps. The first step is the preperation. Both the model and dataset are prepared for the experiment. In section 3-2 the choice for the KITTI dataset is explained. Also the applied

preprocessing and split of training and test set are discussed. In 3-3 the choice of model is explained. The second step in the experiment is pruning the 3 different structures from the network and is described in section 3-4. The 3 methods are discussed in the same section, due to the similar approach in pruning the different structures. The method of pruning is given by equations and in pseudo code. The third step describes how the pruned networks are tested on the Graphical Processing Unit (GPU) and Raspberry Pi. The method of testing on the Raspberry Pi is described extensively. This is to promote further research, making use of a Raspberry Pi. The last step is the evaluation of tests on the GPU and Raspberry Pi. In section 3-7 the methods of evaluation are explained.

3-2 KITTI Dataset

The dataset used for training and evaluation is the KITTI dataset on 2d object detection [34]. This dataset is used as benchmark for at least 150 published works and has been cited for over 4000 times. The dataset has 7481 labeled Portable Network Graphics (PNG) images available for training. 7518 images are kept at the Karlsruhe Institute of Technology (KIT) used for testing of published works. The images are labeled with bounding boxes, and a class per bounding box. The 3 available classes are pedestrians, cyclists and cars. A more detailed background and analysis of the dataset can be found in chapter 2 section 2-4. At least over 20 other public datasets, published by research institutes, are available. The KITTI dataset has bounding boxes on cyclists available, which is relevant because the fatalities under pedestrians and cyclists have been decreasing slower than other road users [11]. The availability of cyclists and the dataset being used by over 150 published works have made the KITTI dataset the basis of this research.

3-2-1 Division to training and test set

The 7481 images of the dataset is split evenly into a training and validation set according to the same data split used in the original paper on the SqueezeDet [7]. This ensures that the pruned networks will not be evaluated on data which may have been used for training the pretrained model. In appendix A-1-1 a list of image numbers used in the training set set can be found. All the steps in the experiment use the same training and validation set.

3-2-2 Preprocessing

Preprocessing is applied to artificially create more training data and to ensure stable results. All the preprocessing steps are described in this section, to ensure similar results can be reproduced. To be able to work with the pretrained network, the preprocessing steps for the dataset are kept similar to the steps used for training the SqueezeDet [7]. The following preprocessing steps are applied to each image.

1. Load PNG image to array with red green blue (RGB) values
2. Subtract dataset mean RGB values from the image

3. Let the image drift randomly in x and y direction
4. Randomly mirror the image
5. Rescale the image

Table 3-1: Parameters used for preprocessing of the data.

Parameter	Mean red value	Mean green value	Mean blue value	Max pixels drift in X	Max pixels drift in Y	Chance of mirroring
Value	123.68	116.779	103.939	150	100	0.5

All the parameters for preprocessing the images can be found in table 3-1. The first step converts the data to a compatible array for the neural network. The second step, subtracting the mean value of each RGB value from the image RGB values achieves centered data. The third step shifts the image for a random amount of pixels, the maximum drift for both directions is determined by the corresponding parameter. The empty space in the image is filled with zeros. To make sure bounding boxes do not drift of the image, the maximum drift parameter is lowered when bounding boxes are too close to the edge. The fourth step is randomly mirroring the image, the chance can be set by the corresponding parameter. The final step is rescaling the image to the input size of the network.

3-3 Squeezedet model

The model chosen to do this research is the Squeezedet plus model published in the paper on Squeezedet [7]. This model is based on the Squeezenet [5]. The Squeezenet is a well known model in research (over 2000 citations), which introduced an architecture for object recognition with 50 times fewer parameters as the network AlexNet [13]. More information on the architecture of these networks and the performance can be found in chapter 2 in section 3-3.

The Squeezedet plus model has been chosen for this research, because the architecture is based on the well known Squeezenet and the code and pretrained parameters have been published. The availability of the pretrained parameters and code makes this work easier to reproduce and will also make it suitable for comparison when other compression methods will be applied to the same pretrained model.

The Squeezedet is implemented using an adaptation of the source code from github of the paper [7]. Two scripts are used to implement the network. The script 'nn_skeleton.py', found in A-2-2, contains all the functions required to create a network. The functions for adding the loss graph to the network and the function to create the convolutional layer have been adapted to enable pruning. More information on these adaptations can be found in section 3-4-2. Then a script for creating the network class is run, to create an instance of the model. The scripts for the different network classes can be found in A-2-3. The differences between the network classes are explained in section 3-4-3.

3-4 Network pruning

3-4-1 Method overview

In figure 3-3 the steps are shown for pruning the network. At the start the original network is transferred into a sparse network by creating structured sparsities. The method to obtain these sparsities is explained in section 3-4-2. Then the sparsities are exploited to remove structures from the network, this is explained in section 3-4-3. Then the network is retrained in the same way the original network was retrained, the details can be found in section 3-4-4. Then a second pruning iteration can be started with the retrained network, or the process is stopped. The choice of iterations can be found in section 3-4-5.

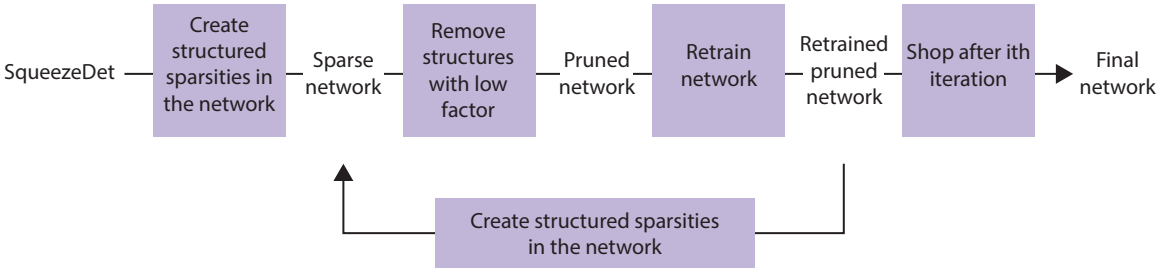

Figure 3-2: Schematic overview of pruning the network. The first step is to create structured sparsities in the network. This step is different for the different pruning methods. Then the sparse structures are removed, this also depends on the pruning method. The network is then retrained, this is done in the same way is described in [7]. Finally a new iteration is started or the process is stopped

3-4-2 Creating structured sparsities in the network

l1 regularization on structure multiplication factors

The method to prune the networks for different structures is based on the method introduced in [6]. The method from the paper is discussed in detail in 2-3-3. In the paper filters in the network are multiplied by factors, on which l1 regularization is applied. To prune other structures than filters, a similar approach is used in this research. The 3 reasons for using this method instead of many different other pruning methods are the following: the wide use of l1 regularization for inducing sparsity, the possibility of using them equally on the different structures and the proven ability of getting decent results.

The wide use of l1 regularization for sparsity increases the ease of reproducibility. Tensorflow, the used framework for this research, implements l1 regularization. The possibility to apply the same method on different structures, by multiplying each structure with a factor, makes it suitable for comparing the effect of removing different structures. The method has shown decent results, however after 2017 works as mentioned in [25] have shown better results. However for this thesis the difference in networks with pruned structures on the different hardware is researched and not the single best method for a microcomputer.

Definition of the to regularize factors and loss functions

From the network 3 different types of structures need to be removed, with minimal effect on the performance in the network. The 3 structures being filters, a row or column reduction of filter kernels along a layer, or a complete layer. The training algorithm, making use of backpropagation, is used to restructure the network, leading to the desired sparsities. In this section new parameters, referred to as γ are defined. During the pruning phase only the γ parameters are updated, the original parameters are kept constant. The training algorithm updates the network according to the loss function. An additional term is added to the original loss function. The original loss function was written only to optimize the mean average precision of the network, the added term will also induce optimization for sparsity.

The original loss function (3-1) for training the network on object detection was given in section 3-3:

$$\begin{aligned} L_{sq} =& \frac{\lambda_{bbox}}{N_{obj}} \sum_{i=1}^{W}\sum_{j=1}^{H}\sum_{k=1}^{K} I_{ijk}((\delta x_{ijk} - \delta x_{ijk}^{G})^2 + (\delta y_{ijk} - \delta y_{ijk}^{G})^2 + (\delta w_{ijk} - \delta w_{ijk}^{G})^2 \\ &+ (\delta h_{ijk} - \delta h_{ijk}^{G})^2) + \sum_{i=1}^{W}\sum_{j=1}^{H}\sum_{k=1}^{K} \frac{\lambda_{conf}^{+}}{N_{obj}} I_{ijk}(\gamma_{ijk} - \gamma_{ijk}^{G})^2 + \frac{\lambda_{conf}^{-}}{WHK - N_{obj}} \bar{I}_{ijk}\gamma_{ijk}^{2} \\ &+ \frac{1}{N_{obj}} \sum_{i=1}^{W}\sum_{j=1}^{H}\sum_{k=1}^{K}\sum_{c=1}^{C} I_{ijk} l_c^{G} log(p_c) \end{aligned} \tag{3-1}$$

For sparsifying the different structures the three different additional terms are defined as followed.

For pruning filters: Let F be the set containing all filters $f_a, f_b, ...f_i$ in the network, containing i filters. Then let Γ_{fil} be the set containing all variables $\gamma_{fa}, \gamma_{fb}, ...\gamma_{fi}$ each multiplied by the corresponding filter $f_a, f_b, ...f_i$. Then the absolute sum of each γ_f in Γ_{fil}, multiplied by λ_{fil}, is added to the loss L_{sq}. Resulting in the new function 3-2.

$$L_{pr} = L_{sq} + \lambda_{fil} \sum_{\gamma \in \Gamma_{fil}} |\gamma| \tag{3-2}$$

For pruning rows and columns of filters along a layer: Let L be the set containing all sets $F_a, F_b, ...F_l$ of the network of l layers. Then each set $F_a, F_b, ...F_l$ contains all $m_a, m_b, ...m_l$ by $n_a, n_b, ...n_l$ filters from the layer. Then let Γ_{row} be the set containing all variables

$\gamma_{ra1}, \gamma_{ra2}, ...\gamma_{rama}, \gamma_{rb1}, \gamma_{rb2}, ...\gamma_{rbmb}, , \gamma_{rl1}, \gamma_{rl2}, ...\gamma_{rlml}$

where each factor is multiplied by the corresponding row in the corresponding filter. Then let Γ_{column} be the set containing all variables

$\gamma_{ca1}, \gamma_{ca2}, ...\gamma_{cama}, \gamma_{cb1}, \gamma_{cb2}, ...\gamma_{cbmb}, , \gamma_{cl1}, \gamma_{cl2}, ...\gamma_{clml}$

where each factor is multiplied by the corresponding column. Then the absolute sum of each γ in Γ_{row} and the absolute sum of each γ in Γ_{col} multiplied by λ_{colrow}, is added to the loss L_{sq}. Resulting in the new function 3-3.

$$L_{pr} = L_{sq} + \lambda_{colrow}(\sum_{\gamma \in \Gamma_{row}} |\gamma| + \sum_{\gamma \in \Gamma_{row}} |\gamma|) \tag{3-3}$$

For pruning the expanding layers from the network: Let E be the set containing all the expanding layers of the network. Then let Γ_{lay} be the set containing all variables $\gamma_{ea}, \gamma_{eb}, ...\gamma_{ei}$ each multiplied by the corresponding expanding layer. Then the absolute sum of each γ in Γ_{lay}, multiplied by λ_{lay} is added to the loss L_{sq}. This results in the loss function 3-4.

$$L_{pr} = L_{sq} + \lambda_{lay} \sum_{\gamma \in \Gamma_{lay}} |\gamma| \tag{3-4}$$

Implementing the to regularize factors and loss functions

The different sets Γ defined in the previous section are coded in the convolutional layer function of the network. In the algorithm 1 the logic is given how the convolutional layer is created, according to the applied pruning method. Depending on the pruning method in each layer new gammas are created and multiplied by the desired structures. In algorithm 2 the logic for defining the loss functions is defined. Depending on the pruning method the loss funtion of the network is updated. The choice of λ for the loss functions is, together with other hyperparameters, given in section 3-4-2.

Algorithm 1 Create convolutional layer with pruning logic

filter rows = r
filter columns = c
number of filters = f
channels = ch
Layer number = n
create Filters tensor F with dim $r \times c \times ch \times f$

if pruning filters **then**
  Fill array with Γ_{filn} of length f with ones
  $F_n = F_n \odot \Gamma_{filn}$ {F is pointwise multiplied along dimension f}
  F_n is not trainable
  Γ_{filn} is trainable
  create convolutional layer L_n with F_n
end if

if pruning filter rows and columns **then**
  create Identity($r \times r$) Γ_{rown}
  create Identity($c \times c$) Γ_{coln}
  $F_n = \Gamma_{rown} \times F_n \times \Gamma_{coln}$ {a matrix multiplication along dimension r,c for F}
  F_n is not trainable
  Γ_{rown} and Γ_{coln} are trainable
  create convolutional layer L_n with F_n
end if

if pruning layers **then**
  create scalar γ_{layn}
  F_n is not trainable
  γ_{layn} is trainable
  create convolutional layer L_n with F_n
  $L_n = L_n \odot \gamma_{layn}$
end if

if not pruning **then**
  F_n is trainable
  create convolutional layer L_n with F_n
end if

Algorithm 2 Add l1 regularization loss to loss function

Original loss from Squeezedet = L_{sq}
Function for taking l-norm of array numbers = $N_{l1}()$

if pruning filters **then**
$L_{pr} = L_{sq} + \lambda_{fil} \times N_{l1}(\Gamma_{fil})$
end if

if pruning filter rows and columns **then**
$L_{pr} = L_{sq} + \lambda_{rowcol} \times N_{l1}(array[\Gamma_{row}, \Gamma_{col}])$
end if

if pruning layers **then**
$L_{pr} = L_{sq} + \lambda_{lay} \times N_{l1}(\Gamma_{lay})$
end if

Using the training algorithm for pruning

The network is pruned using the training algorithm. The introduced gammas in Γ_{fil}, Γ_{rowcol} and Γ_{lay} are defined as trainable. The weights and biases of the filters in the network are not trainable in this stage. Because the number of gammas in Γ_{fil}, Γ_{rowcol} and Γ_{lay} differ by a factor over 30, the loss terms differ by the same factor. To compensate for this, λfil, $\lambda rowcol$ and λlay are chosen as such to compensate for this difference. This results equal additional loss terms for all structures. After initial experimenting the height of the additional loss term has been set to 4.5. The lambdas for each iteration are given by equation 3-5.

$$\lambda = \frac{4.5}{N} \tag{3-5}$$

Where N is the total number of gammas to be optimized. In table 3-2 the resulting values for lambda for the first iteration are given.

Table 3-2: The learning rates and lambdas during the 1st iteration of network pruning. The values follow from 3-5 and 3-6. In later iterations the values are updated according to the same equations

Parameter	LR_{fil}	LR_{rowcol}	LR_{lay}	λ_{fil}	λ_{rowcol}	λ_{lay}
Value	0.1	0.001	0.0003	0.00071	0.061	0.225

$$L_{pr} = L_{sq} + \lambda_{colrow}(\sum_{\gamma \in \Gamma_{row}} |\gamma| + \sum_{\gamma \in \Gamma_{row}} |\gamma|) \tag{3-6}$$

3-4-3 Removing structures from the network

When the pruning algorithm finished creating sparsity in the network by setting certain gamma factors to low numbers, the structures need to be removed from the network to achieve

actual acceleration. When removing structures, the weights of the network are deleted and the network needs to be rebuild with the new smaller set of weigths. To maintain the same output from the network, the new network has to connect the parameters in the same way as they were connected in the larger network. To achieve the removal of structures and maintaining the same output the method is split into five steps.

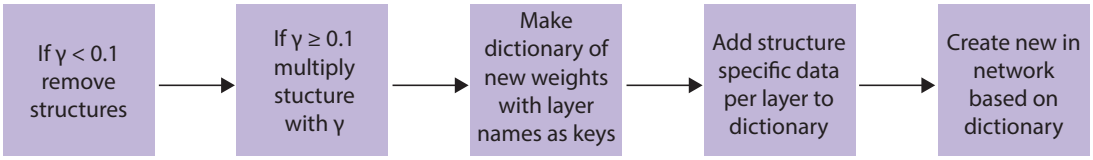

Figure 3-3: This scheme shows the method in 5 steps to remove structures from the network. For pruning 3 different structures the same steps are followed. For the each structure the final 2 steps differs slightly from the others, due to the different nature of the structures.

In figure 3-3 the five steps are shown. The first step is to remove the structures, the second step multiplies the other structures with the gamma factor, to maintain the same network output. Then a dictionary is created, where the parameters are stored per layer. Then for pruning the rows and columns of the filters, extra information needs to be stored to indicate an offset. Then the network classes create new smaller networks based on the stored parameters in the dictionary. In appendix A-2-4 the code for removing the structures can be found.

3-4-4 Retraining

Each network is retrained when the structures are removed. For training the network the same code and parameters, except for learning rate, are used as described in [7]. The learning rate in the original paper started at 0.01 and decayed 8 times with 0.5. This means it ended with approximately 0.00004. As the weights now only need to be fine tuned to the new structure, the learning rate starts at 0.00008 and decays after 2000 steps to 0.00004. Then the training is run for another 2000 steps.

Table 3-3: Parameters used for retraining the networks

Parameter	Learning rate	Weight decay	Momentum	Learning rate decay	Learnin rate decay step
Value	0.00004	0.0001	0.9	0.5	2000

3-4-5 Number of iterations

For this thesis the amount of iterations for all 3 methods has been set to 3. In [6] one iteration already leads to significant accelerations. 3 iterations are chosen to make sure significant accelerations are achieved for all methods.

3-5 Embedding and testing network on Raspberry Pi

In this section a detailed description is given on how to set up the Raspberry Pi for research. The level of detail is important for future research and education as a detailed method description for these tests is not available yet. First the specifications of the used Raspberry Pi are given. Then the instructions on how to set up the Raspberry Pi for running Tensorflow neural networks are given. Then the code and instructions are given how to convert Tensorflow models for usage on the Raspberry pi. At last the data and code for the Raspberry Pi are given to run the tests.

3-5-1 Raspberry pi technical information

For this research the Raspberry pi 3b+ was provided as microcomputer for testing. The specifications of the Raspberry pi 3b+ are listed below.

- CPU: Broadcom BCM2837B0 quad-core A53 (ARMv8) 64-bit @ 1.4GHz
- RAM: 1GB LPDDR2 SDRAM
- Power: 5V/2.5A DC
- Price: 40 euro

3-5-2 Raspberry pi set up

In this section the following steps to set up the Raspberry pi for testing neural networks from Tensorflow are described:

- Operating system isntallation on Raspberry Pi 3b+
- Required package installation
- Installation of tflite_runtime

The following equipment is required to set up the Raspberry Pi:

- Raspberry pi 3b+
- Computer with internet connection
- Micro SD card
- Micro SD card reader
- Screen with HDMI cable
- keyboard with USB cable
- 5V/2.5A DC micro USB power cable

Table 3-4: Installed packages and the installed version

Package	pillow	numpy	tflite_runtime
Version	6.1.0	1.17.3	1.14.0

Operating system installation

First the OS needs to be installed on the memory card of the Raspberry Pi. The OS Raspbian Buster Lite can be downloaded following the link https://www.raspberrypi.org/downloads/raspbian/. There a link to download the ZIP for Raspbian Buster Lite can be found. Unzip the archive to get the .img file of Raspbian Buster Lite.

To write the image to the SD card the software package balenaEtcher is used. This can be downloaded and installed via this link https://www.balena.io/etcher/. When the software is installed connect the sd card with the sd card reader. Then open balenaEtcher and select the unzipped .img file. Select the connected sd card to write to. Then click flash to write the OS to the sd card. When the image is written, the micro sd can be unplugged and put into the Raspberry Pi.

First plug in the screen and keyboard to the Raspberry Pi, then plug in the power cable. The Raspberry Pi will now boot. Run through the installation process and set up a wireless internet connection during this process. This is necessary to install additional software. When the Raspberry Pi finished its installation process you can login using the default credentials:

Username: Pi

Password: Raspberry

Required software and package installation

The Raspbian Buster Lite OS comes installed with python3.7. python3.7 is used for running the models. To install additional required packages for python3.7, pip3 needs to be installed by:

```
$ sudo apt install python3-pip
```

Then the 3 required packages from table 3-4 need to be installed. In the table the versions are listed, which have been used during this research. To install run the following commands on the Raspberry Pi:

```
$ pip3 install Pillow==6.1.0
$ pip3 install numpy==1.17.3
$ pip3 install https://dl.google.com/coral/python/
    tflite_runtime-1.14.0.post1-cp37-cp37m-linux_armv7l.whl
```

This installs the specific version of the packages. The newest version can be installed if preferred by leaving out the "==x.y.z" in the command. However this could lead to mismatches in packages and therefore malfunction.

Convert Tensorflow model to tflite model

To be able to run a Tensorflow model efficiently on a Raspberry Pi, Tensorflow has developed the tflite library. Using this library the Tensorflow model can be converted to a tflite format, so an embedded device only needs the tflite interpreter to run the model. In this section is explained how to convert the Tensorflow model, which is a graph, to a tflite model. This is done in 2 steps as the model needs to be converted to a Tensorflow savedModel first. Then the conversion to a tflite format can be executed.

Before converting the the Tensorflow model graph is needs to be made suitable for conversion. Only a small selection of operations is supported in tflite. The list of supported operations can be found following this link: https://www.tensorflow.org/lite/guide/ops_compatibility. For the Squeezedet this meant removing the dropout operation and all the graph nodes used for training and evaluating the network.

The python code to convert the Tensorflow graph to a Tensorflow savedModel is given in 3.1. This conversion is needed to define the input and output tensors of the network, without these the eventual tflite model does not have information on what the input and output of the model is. From lines 16 to 21 the model graph is loaded, for this thesis this is done with the function SqueezeDetPlusBNEval(). When the model is loaded a checkpoint from training can be loaded, however in this case pretrained weights are already initialised. For the conversion to work, all variables need to be initialised, so from lines 27 untill 35 all variables are checked on initialization and initialized if they are not. From lines 37 untill 53 a prediction signature is created, so the model has its input and output defined. Lines 56 untill 64 are used to save the Tensorflow savedModel information.

```
1  import os
2  import tensorflow as tf
3  from config import kitti_squeezeDetPlus_config
4  from nets import SqueezeDetPlusEval
5  import pdb
6  from dataset import kitti
7
8  weights_path = 'Optional path to pretrained weights array for initialization'
9  export_dir = os.path.join('/path where saved model data is stored', '0')
10
11
12 graph = tf.Graph()
13 with tf.compat.v1.Session(graph=graph,
       config=tf.ConfigProto(allow_soft_placement=True)) as sess:
14     # This part is used to build a graph of the model, when using your own
15     # model, replace this code to build your own tensorflow graph
16     mc = kitti_squeezeDetPlus_config()
17     mc.IS_TRAINING = False
18     mc.IS_PRUNING = False
19     mc.BATCH_SIZE = 1
20     mc.PRETRAINED_MODEL_PATH = weights_path #pretrained weights are loaded
21     model = SqueezeDetPlusEval() # Function to build the graph of the model
22
23     # When using a tensorflow checkpoint from training
24     # Here the checkpoint can be restored
```

```
25
26      # All vars need to be initialized to create saved model
27      uninitialized_vars = []
28      for var in tf.all_variables():
29          try:
30              sess.run(var)
31          except tf.errors.FailedPreconditionError:
32              uninitialized_vars.append(var)
33
34      init_new_vars_op = tf.initialize_variables(uninitialized_vars)
35      sess.run(init_new_vars_op)
36
37      #Input and output tensors need to be defined and are searched by name
38      input_tensor_info = tf.saved_model.build_tensor_info(
39          graph.get_tensor_by_name('image_input:0'))
40      output_tensor_bbox_info = tf.saved_model.build_tensor_info(
41          graph.get_tensor_by_name('bbox/trimming/bbox:0'))
42      output_tensor_class_info = tf.saved_model.build_tensor_info(
43          graph.get_tensor_by_name('probability/class_idx:0'))
44      output_tensor_conf_info = tf.saved_model.build_tensor_info(
45          graph.get_tensor_by_name('probability/score:0'))
46
47      # Define inputs and output of model and create signature
48      prediction_signature = tf.saved_model.signature_def_utils.build_signature_def(
49                             inputs={'image': input_tensor_info},
50                             outputs={'bbox': output_tensor_bbox_info,
51                                      'class': output_tensor_class_info,
52                                      'conf': output_tensor_conf_info},
53                             method_name=tf.saved_model.signature_constants.PREDICT_METHOD_NAME)
54
55      # Export to SavedModel
56      builder = tf.saved_model.builder.SavedModelBuilder(export_dir)
57      builder.add_meta_graph_and_variables(sess,
58                                           [tf.saved_model.tag_constants.SERVING],
59                                           signature_def_map={
60                                           'predict_images': prediction_signature},
61                                           clear_devices=True,
62                                           strip_default_attrs=True)
63      builder.save()
64      builder.save(as_text=True)
```

Listing 3.1: Python code to convert a Tensorflow graph to a Tensorflow saved model. This is the first step for creating a tflite model from a Tensorflow graph.

The python code to convert the Tensorflow Savedmodel to a tflite model is showed in the listing 3.2. The savedModel directory is given to the tflite converter. Then the tflite interpreter is used to check if the input and output tensors are found. Then the model is saved as tflite model.

```
1   import os
2   import tensorflow as tf
3
```

```
loaded_graph = tf.Graph()
with tf.Session(graph=loaded_graph,
    config=tf.ConfigProto(allow_soft_placement=True)) as sess:
    export_dir = os.path.join('Use the export dir of savedmodel here', '0')

    converter = tf.lite.TFLiteConverter.from_saved_model(export_dir,
        signature_key='predict_images')
    tflite_model = converter.convert()

    # Load TFLite model and allocate tensors.
    interpreter = tf.lite.Interpreter(model_content=tflite_model)
    interpreter.allocate_tensors()

    # Get input and output tensors.
    input_details = interpreter.get_input_details()
    output_details = interpreter.get_output_details()

    open("pathtomodel.tflite", "wb").write(tflite_model)
```

Listing 3.2: Python code to convert Tensorflow saved model to tflite model. This is the second and final step to convert a Tensorflow graph to a tflite model. The model will be saves as .tflite file and can be invoked by the tflite interpreter.

Setting up data and code to test

To test the tflite model on the Raspberry Pi, test data and a test script is needed. For testing the images can be stored on the Raspberry Pi as .png images. The code makes use of the tflite_runtime interpreter. This package only uses the necessary Tensorflow code for running this interpreter and therefore is quicker to load on a Raspberry Pi.

```
from PIL import Image
import numpy as np
import time
import tflite_runtime.interpreter as tflite

model_names = [ 'path_to_model.tflite',]

for mname in model_names:
  interpreter = tflite.Interpreter(model_path=mname)
  interpreter.allocate_tensors()

  input_details = interpreter.get_input_details()
  output_details = interpreter.get_output_details()

  #KITTI image numbers to test can be given here
  im_nums=['000005','000243', '000709', '001002', '001051', '001137',
  '005047', '004995', '006762', '006778']
  times = []

  #Run model on KITTI images
  for n in im_nums:
```

```
22      img = Image.open('/data/KITTI/training/image_2/' + n + '.png')
23
24      input_data = np.expand_dims(img, axis=0)
25      input_data = np.array(input_data, dtype=np.float32)
26
27      interpreter.set_tensor(input_details[0]['index'], input_data)
28
29      start = time.time()
30      interpreter.invoke()
31      end = time.time()
32      times.append(end-start)
33      print(end-start)
```

Listing 3.3: Python code to run on Raspberry Pi. The code loads the tflite model using the tflite_runtime.interpreter. Then the code loops through a list of test images and runs the model on the image.

3-6 Testing network on GPU

In this section the description is given how the speed of the networks is tested on the GPU. For this project training and testing is done with GPUs from cloudservice Floydhub. Technical information of this GPU is given in this section about this GPU and the code for testing on the GPU is given as well.

3-6-1 GPU technical information

For this thesis the GPU NVIDIA TESLA K80 is used. This is the GPU available when making use of the cloudservice Floydhub. The specifications are listed below:

- GPU Name: GK210 @ 5 GHz
- RAM: 12 GB GDDR5
- Power: 300W
- Price: 830 euro

3-6-2 Speed test

The testing is performed in the python environment using Tensorflow. Tensorflow Lite is not used in this case, as it is not compatible with the GPU. The speed test is done using the evaluation script found in appendix A-2-5. This script is run with the same 100 examples as the speed test on the Raspberry Pi. The evaluation time of the network on each image is measured with the function process_time from the python time package. The measured times are saved and exported for further evaluation.

3-7 Evaluation methods

The results of the pruning methods will be shown to add context to the acceleration results. The location of the pruned structures will be shown and the remaining accuracy of the network. This does not directly contribute to comparing the acceleration on the devices, however it gives context in whether the compression technique would be suitable for further use in compressing neural networks for microcomputers.

The compression results will be given in terms of reduction in parameters, flops and size on disk. This indicates the effectiveness of the pruning method and helps with interpreting the acceleration results.

The measurements of speed on the devices will be done in seconds. To compare the acceleration, the measurements will be normalized to the speed of the original model on the respective device. To test if the acceleration is statistically significant, the unequal variances t-test and the two-tailed p-test will be used. When the p-value is below 0.05 the difference in acceleration is statistically significant.

Chapter 4

Results and Discussion

In this chapter first all the results are shown and in the end will be discussed.

First in section 4-1, the results of the 3 pruning methods are shown. This will give insight in how the pruning methods have performed, to give context for the actual measured speedups. From the pruning method the distribution of structure factors is shown first, to indicate how many structures were qualified to be removed. The location of the removed structures in the network is given. Finally in this first section a table is given where the performance of the original networks is compared with the pruned networks from all iterations.

In the second section (4-2), the compression and acceleration results of the methods are shown. The compression is expressed in amount of parameters and size on disk. The acceleration in flops and inference time on the hardware. Finally the normalized accelerations are shown on the different devices and the p-values are given for the difference in speed for each pruned network on the different devices.

Finally in section 4-3, the results will be discussed.

4-1 Pruning results

In figures 4-1 and 4-2 the results of pruning filters from the network are found. The distribution of factors for the filters is shown in a histogram. According to the method all filters with a factor lower than 0.1 were removed, this corresponds to the first bin in the histogram. The location of the removed filters is indicated per layer.

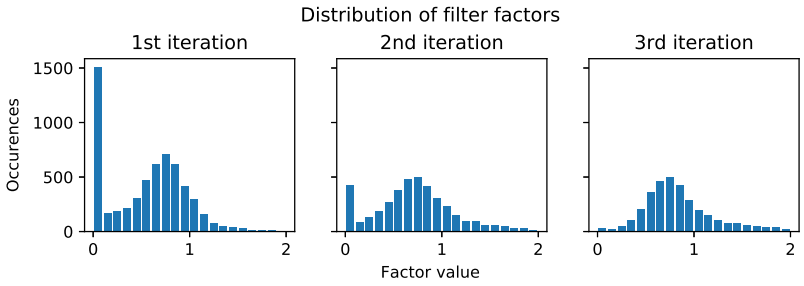

Figure 4-1: The figure shows a histogram of factors that the filters were multiplied with. The distribution is shown after each iteration of creating sparsity in the network. Based on these factors the filters were removed. The first bin of the histogram shows the amount of factors qualifying their structures for removal (<0.1).

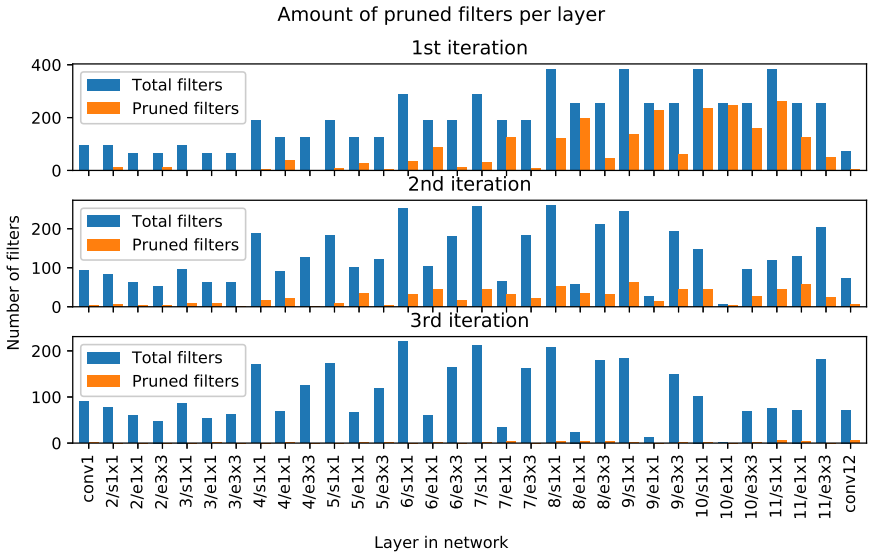

Figure 4-2: The figure shows the location of the removed filters in the network. On the horizontal axis the layer names are shown. From the fire module layers, the first part indicates the fire module number, the second part the layer in the fire module. From each layer the total amount of filters is shown and the amount of removed filters in that iteration.

In figures 4-3 and 4-4 the results of pruning rows and columns from filter kernels along a layer are found. The distribution of factors for the rows and columns is shown in a histogram. According to the method all rows and columns with a factor lower than 0.1 were removed, this corresponds to the first bin in the histogram. The location of the removed rows and columns is indicated per layer. Only the layers with a filter size larger than 1x1 were suitable for this method, so these are the layers shown in the figure.

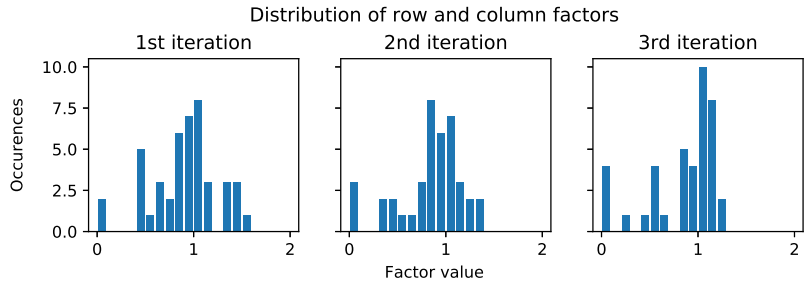

Figure 4-3: The figure shows a histogram of factors that the rows and columns were multiplied with. The distribution is shown after each iteration of creating sparsity in the network. Based on these factors the kernel row of column from the filter was reduced. The first bin of the histogram shows the amount of factors qualifying their structure for removal (<0.1).

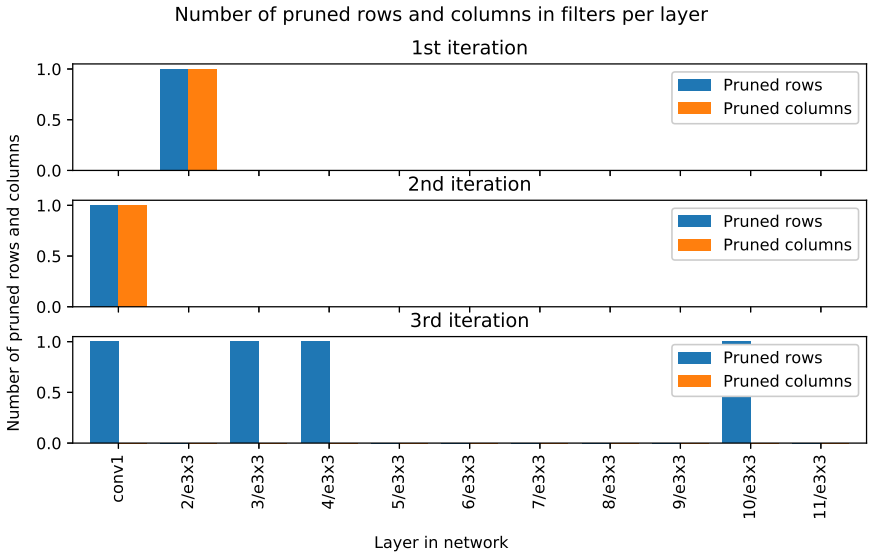

Figure 4-4: The figure shows the location of the removed rows and columns in the network. On the horizontal axis the layer names are shown. Only layers with a filter size suitable for size reduction($>$1x1) are shown. From the fire module layers, the first part indicates the fire module number, the second part the layer in the fire module. From each layer the total amount of filters is shown and the amount of removed filters in that iteration.

In figure 4-5 and table 4-1 the results of pruning layers from the networks are shown. The distribution of factors for the layers is shown in a histogram. According to the method all rows and columns with a factor lower than 0.1 were removed, this corresponds to the first bin in the histogram. The location of the removed layers is shown in a table. Following the method for removing the layers only the expanding layers in the fire modules were eligible for removal. These layers are shown in the table.

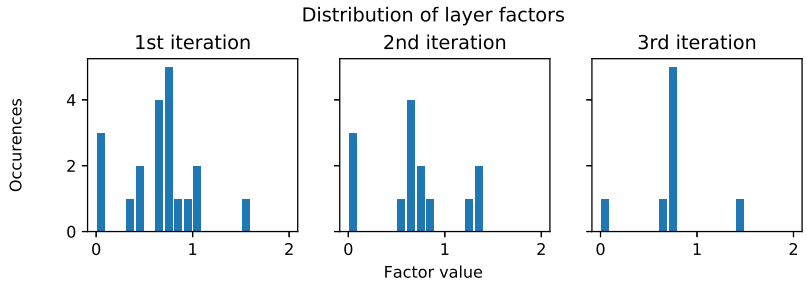

Figure 4-5: The figure shows a histogram of factors that the layers were multiplied with. The distribution is shown after each iteration of creating sparsity in the network. Based on these factors the layers were removed. The first bin of the histogram shows the amount of factors qualifying their structure for removal (<0.1).

Table 4-1: In this table is shown which layers have been pruned from the network in which iteration. Only the layers eligible for removal are listed. The first part of the layer name is the number of the fire module, the second part if it is the expanding layer with 1x1 filter of expanding layer with 3x3 filters. A total of 7 layers have been pruned, over 3 iterations.

Layer name	Pruned in iteration
2/e1x1	
2/e3x3	2
3/e1x1	
3/e3x3	
4/e1x1	
4/e3x3	
5/e1x1	
5/e3x3	
6/e1x1	3
6/e3x3	
7/e1x1	2
7/e3x3	
8/e1x1	2
8/e3x3	
9/e1x1	1
9/e3x3	
10/e1x1	1
10/e3x3	
11/e1x1	1
11/e3x3	

In table 4-2 from the original network and all the pruned networks the precision on each class divided in different difficulties and the mean over all precisions is shown.

Table 4-2: In the table the precisions on the three classes are shown. The KITTI evaluation metrics are divided in easy (E), medium (M) and hard (H), depending on the minimum bounding box size. In the final column the mean over the precisions of all categories in all difficulty levels is shown. Horizontally the original network and the pruned networks from all 3 iterations are listed.

network		car			cyclist			pedestrian			mAP
		E	M	H	E	M	H	E	M	H	
Original network		0.90	0.87	0.78	0.87	0.80	0.78	0.81	0.71	0.68	0.80
full	iter1	0.88	0.85	0.76	0.86	0.80	0.76	0.83	0.75	0.68	0.80
filters	iter2	0.84	0.82	0.72	0.84	0.76	0.74	0.80	0.71	0.66	0.77
pruned	iter3	0.85	0.82	0.72	0.84	0.76	0.73	0.80	0.70	0.65	0.76
filter rows	iter1	0.88	0.82	0.69	0.83	0.75	0.74	0.83	0.73	0.67	0.77
and columns	iter2	0.77	0.67	0.56	0.63	0.57	0.54	0.70	0.60	0.54	0.62
pruned	iter3	0.59	0.49	0.43	0.47	0.41	0.39	0.54	0.46	0.41	0.47
layers pruned	iter1	0.91	0.87	0.78	0.86	0.82	0.77	0.83	0.74	0.68	0.81
	iter2	0.90	0.85	0.76	0.84	0.77	0.76	0.81	0.71	0.67	0.79
	iter3	0.89	0.85	0.74	0.83	0.76	0.75	0.81	0.71	0.66	0.78

4-2 Compression and acceleration results

In figures 4-6 and 4-7 the size of the networks can be found after pruning. The size of the networks is given in the number of parameters and the Mb on disk.

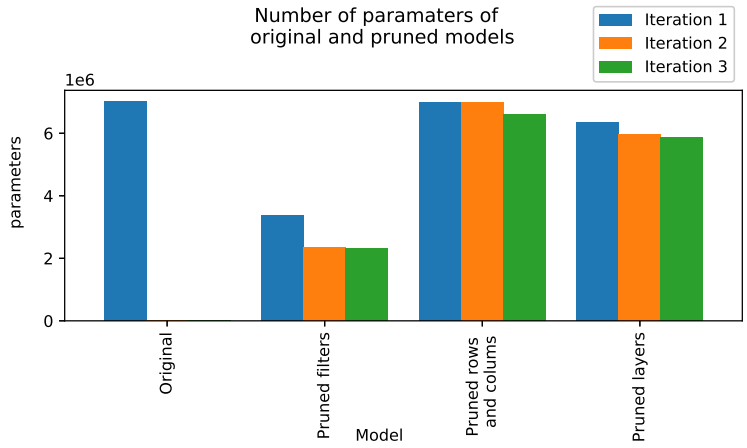

Figure 4-6: The figure shows the number of parameters. The parameters of the original network and the pruned network are shown.

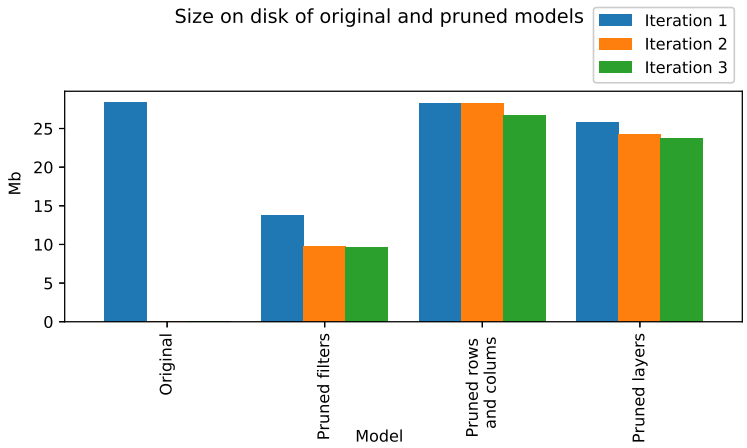

Figure 4-7: The size of the original network and of the pruned networks according to the 3 different techniques. From the pruned networks all the pruning iterations are shown.

In figure 4-8 the number of flops for one forward pass of the network is shown. In figures 4-9, 4-10 the absolute speed of a forward pass in seconds on a Raspberry Pi and GPU are shown. Then in the last figure, figure 4-11, the speeds normalized to the original model are shown, to indicate the acceleration on the different devices. In table 4-3 the p-values are shown for the normalized speed differences per device as shown in figure 4-11.

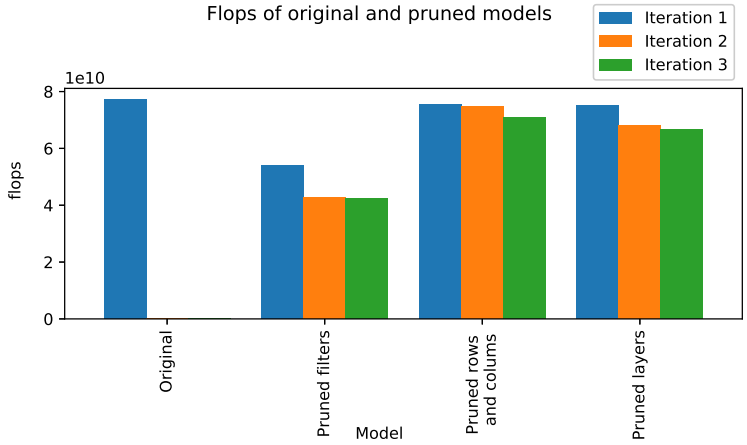

Figure 4-8: The figure shows the number of flops from one forward pass in the network. The flops of the original network and the pruned networks are shown.

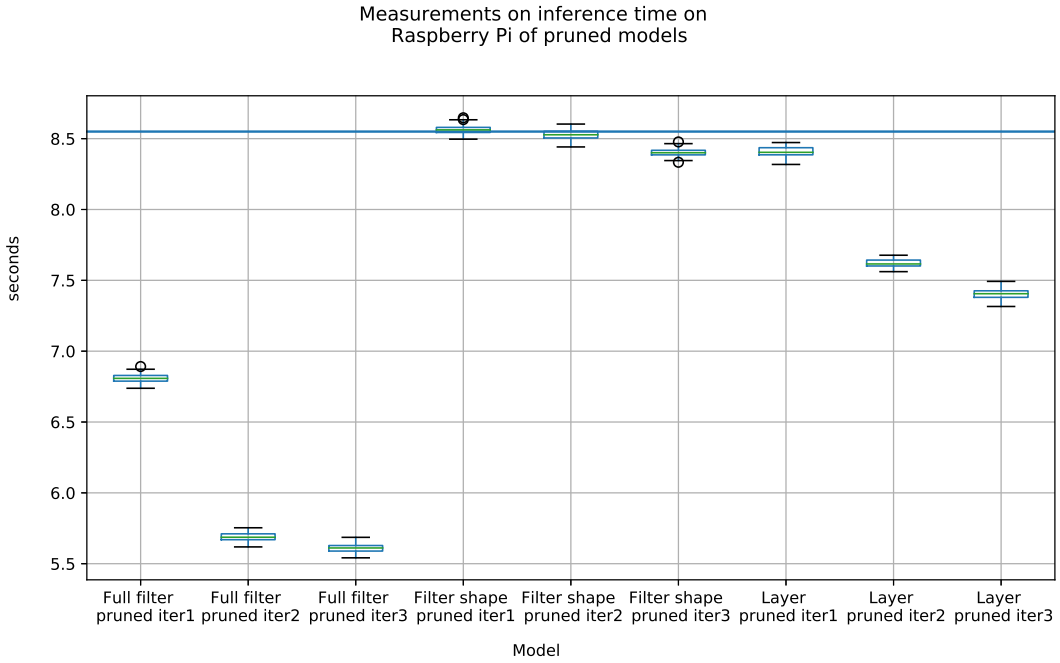

Figure 4-9: The inference time on a Raspberry Pi of the pruned networks according to the 3 different techniques compared to the original network. From the pruned networks all the pruning iterations are shown. The mean of the original inference time is indicated by the blue horizontal line. The measured inference times differs between 0 and 3 seconds per technique. Also the inference time decreases each iteration

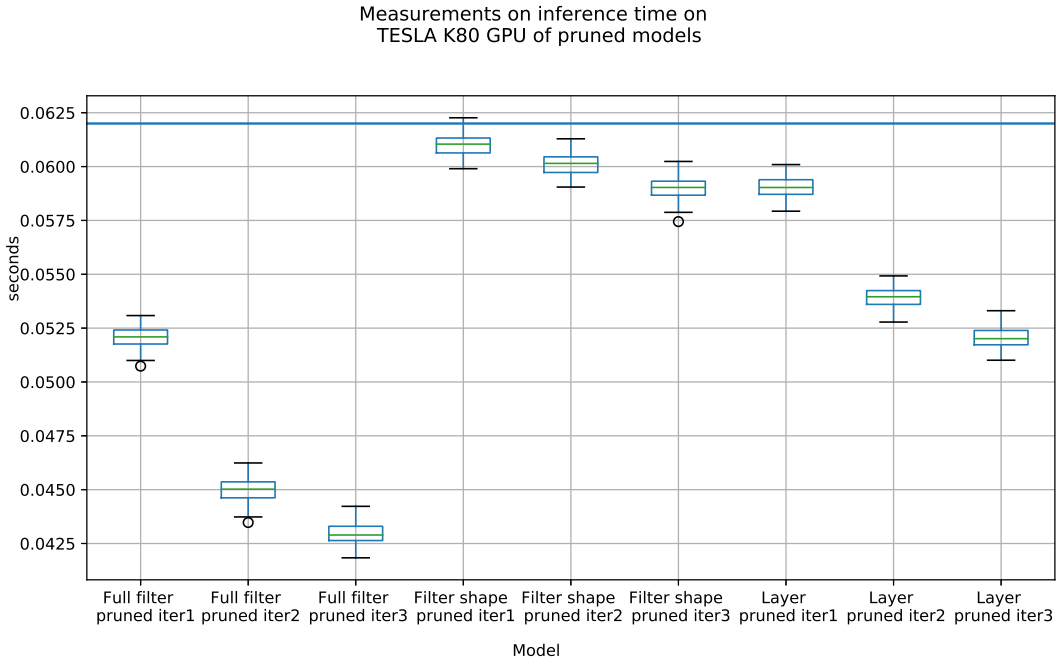

Figure 4-10: The inference time on a Tesla K80 GPU of the pruned networks according to the 3 different techniques compared to the original network.The mean of the original inference time is indicated by the blue horizontal line. The measured inference times differs between 0 and 0.02 seconds per technique. Also the inference time decreases each iteration

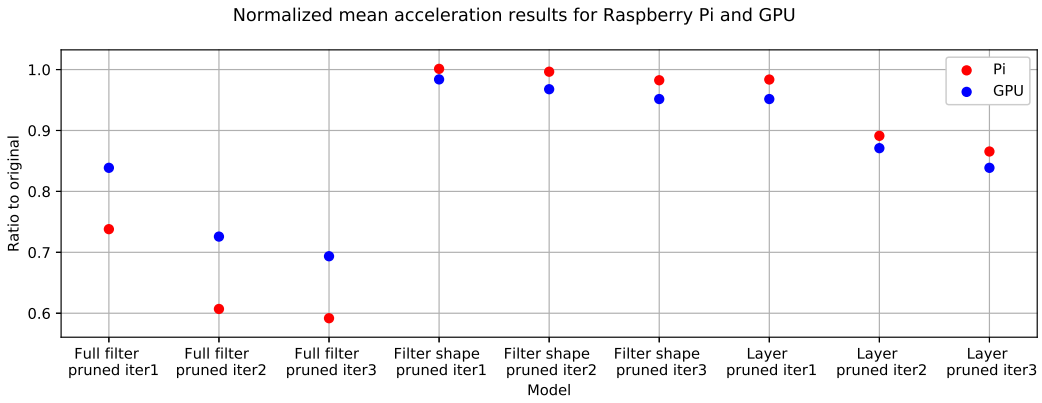

Figure 4-11: The normalized acceleration for the methods prune filters, prune filter shape and prune layer are shown. The 3 methods lead to different behaviour in terms of acceleration on the different hardware. For the full filter method the acceleration on the raspberry pi is between 10 and 12 % better, for the filter shape method and for pruning full layers method the acceleration on the gpu is 2-4 % better.

Table 4-3: In the table the p-values using the unequal variances t-test and two-tailed p-test are listed. The p-values are calculated from the normalized differences as shown in figure 4-11. All differences between the acceleration on the Raspberry Pi and GPU are statistically relevant.

Model	Full filter pruned			Filter shape pruned			Layers pruned		
	iter 1	iter 2	iter 3	iter 1	iter 2	iter 3	iter 1	iter 2	iter 3
p-value	<0.0001	<0.0001	<0.0001	0.0371	0.0009	0.0006	0.0002	0.0208	0.002

4-3 Discussion

4-3-1 Observations on pruning

From figures 4-1 and 4-5 can be seen that pruning the filters and pruning the layers from the network show similar behaviour in the distribution of structure factors trough the iterations. Both end with a relative low number of structure factors below 0.1 after the 3rd iteration. This indicates not much sparsity could be created anymore in the network. The distribution of the layer factors is off course more grainy, due to the number of total factors being 3 orders lower in magnitude. From figure 4-2 and table 4-1 can be seen both methods results in more sparsity in the later layers, with the layer '2/e3x3' as exception. The layer '2/e3x3' is removed in the layer pruning method, however in the filter pruning method it is almost unaffected.

The method of pruning rows and filters from kernels per layer, shows quite different behaviour, see figure 4-3 and 4-4. The row and column factors below 0.1 increases per iteration and also the earlier layers in the network seem to be preferred by this method.

When looking at table 4-2, we can see the pruning of filter rows and columns heavily affects the accuracy of the network. Together with the previous observations, this indicates reducing the size of the kernel shape is not effective in this way. The acceleration can still be compared, however this lack in remaining performance should be kept in mind when applying this method.

When looking at the accuracy of the other methods some interesting observations can be made. After the removal of 3 layers in the first iteration, the mean average precision had increased slightly. This can be explained by the layers being completely redundant and only adding noise to the training process, hindering the training process. Especially the cyclist (M) and pedestrian (M) performance show a significant increase. The leaderboard from [34] even does not even show this high performance on the cyclist precision. However unfortunately testing on the official test is not possible, so these results can not be confirmed.

4-3-2 Observations on compression and acceleration

When observing the compression of the networks, figures 4-6 and 4-7 show that pruning filters lead to the smallest networks and pruning layers lead to the second smallest networks. From the three structures the filters are the smallest and have the highest frequency (shown by most factors) in the networks. This could explain the much lower network sizes of this method, due to more flexibility in pruning structures. However, the method for rows and columns also had more options to prune, but the compression results are lacking.

When comparing the compression to the absolute accelerations in figure 4-9 and 4-10, it is interesting to observe that the difference in acceleration between for example pruned filters and pruned layers is not as big as the difference in compression. In absolute terms, pruning filters reaches the best acceleration.

When comparing the inference times normalized to the original model in figure 4-11, the method of pruning affects the acceleration differently on a GPU and Raspberry Pi. Especially pruning filters leads to up to 12% better acceleration on a Raspberry Pi than a GPU. For the other 2 methods the accelerations on a GPU are up to 4% better. Interestingly is that pruning filters was especially doing well on compression and is also clearly getting better acceleration on a Raspberry Pi.

Chapter 5

Conclusions and Recommendations

5-1 Conclusions

The aim of this thesis was to measure if pruning different structures from a convolutional neural network lead to different accelerations on a microcomputer and GPU. Next to that the aim was to conclude what the best structure is for removal to accelerate a neural network on a microcomputer.

Pruning filters leads to better accelerations on a microcomputer than on a GPU. A difference up to 12%. Pruning layers leads to slightly better accelerations on a GPU. A difference up to 4%. For pruning the rows and columns also the acceleration on the GPU is up to 4% better. Therefore the conclusion is that pruning different structures does not lead to the same accelerations when compared on a microcomputer and GPU. This shows that the choice of best pruning strategy is dependant on the device it will be deployed to.

The best structure to remove for acceleration on a microcomputer is also the method which works better on for microcomputer than for a GPU. This is the method focused on pruning filters. However the methods of pruning layers and filter shapes were dependant on the structure of the SqueezeNet backbone. So when training a pretrained model with the SqueezeNet as backbone, pruning filters will lead to the best results for a microcomputer as well as a GPU.

5-2 Recommendations

It has been shown that the performance of a pruning strategy can differ on the applied hardware. Next to pruning different structures, also the method for creating sparsities in the network can be different. Also methods to search for neural network architectures are available to search for light weight networks. It is recommended to do more research on the effect of those methods on microcomputers, as the different architectures could yield different accelerations on a microcomputer than the accelerations measured on a GPU.

Also in this thesis a pretrained model with the SqueezeNet as backbone has been pruned. This limits the options for pruning layers. Also the filters in the fire models are already too small (1x1) or quite small (3x3). This limits the pruning capabilities when pruning this structures. If shortcuts, like in the RESNET [18] are used, more layers can be pruned leading to better absolute accelerations. Also an architecture with larger filters could have better absolute accelerations for reducing filter size. So when dealing with these types of networks, pruning these structures could still be a good option for a microcomputer and should be researched. However the accelerations will probably be slightly worse than the accelerations on a GPU.

An interesting observation was made, being the accuracy on detecting cyclists. Without it being the aim of the thesis, the accuracy on cyclists was higher than the leader board entries of the official KITTI website. Unfortunately testing on the official test set is not possible for unpublished work, however the results on this data split are already promising. It is recommended to focus more research on detecting cyclists, as there is probably much to win. Especially when taking in to account that the decrease of fatal accidents with vulnerable road users is going slower than fatal accidents with other road users.

Finally it is recommended to switch the focus of research from expensive high performance hardware to cheap low performance hardware. This thesis contributed to this by explaining how these methods can be tested on cheaper hardware. Now the search for acceleration methods is constantly tested on expensive hardware, maybe hiding effective methods for cheap hardware. When the speed and performance increase on cheap hardware, mass adoption of these techniques will accelerate. This enables more educational and industrial use of convolutional neural networks and could accelerate the development of safer traffic.

Appendix A

The Back of the Thesis

A-1 Dataset

A-1-1 Split for training and testing

000000 000002 000003 000004 000006 000007 000010 000011 000017 000018 000019 000021 000025 000028 000030 000031 000032 000033
000034 000036 000039 000040 000042 000043 000044 000045 000046 000047 000049 000050 000057 000058 000061 000063 000065 000067
000069 000070 000071 000072 000074 000076 000079 000080 000081 000082 000084 000085 000086 000088 000089 000094 000095 000096
000098 000104 000108 000110 000111 000113 000115 000118 000121 000122 000124 000127 000129 000130 000131 000132 000135 000136
000142 000144 000145 000150 000153 000154 000158 000164 000165 000166 000167 000168 000170 000176 000177 000179 000186 000187
000188 000190 000191 000192 000193 000195 000196 000201 000202 000203 000205 000208 000209 000210 000211 000214 000215 000217
000220 000222 000223 000224 000228 000230 000231 000233 000234 000235 000236 000239 000241 000243 000245 000246 000247 000249
000252 000254 000256 000257 000258 000260 000262 000263 000266 000268 000269 000270 000274 000278 000287 000293 000295 000297
000300 000301 000302 000306 000307 000310 000311 000312 000313 000322 000324 000325 000326 000327 000328 000329 000335 000340
000342 000345 000346 000348 000349 000350 000351 000353 000354 000355 000356 000358 000361 000363 000366 000369 000371 000372
000373 000374 000377 000378 000380 000382 000383 000385 000386 000388 000389 000390 000391 000392 000395 000396 000397 000398
000402 000403 000407 000410 000414 000415 000416 000418 000421 000423 000424 000425 000426 000428 000429 000432 000434 000435
000436 000437 000440 000441 000442 000443 000445 000446 000448 000449 000451 000453 000457 000458 000461 000462 000463 000464
000470 000472 000476 000477 000478 000479 000481 000482 000486 000487 000488 000489 000496 000498 000499 000500 000502 000503
000505 000506 000507 000510 000511 000512 000513 000515 000516 000518 000519 000522 000526 000527 000528 000536 000537 000544
000546 000547 000550 000551 000553 000554 000555 000556 000557 000562 000566 000568 000570 000573 000575 000576 000578 000579
000580 000581 000582 000585 000587 000588 000590 000591 000593 000594 000595 000596 000597 000599 000600 000601 000603 000607
000610 000612 000613 000618 000620 000624 000626 000628 000629 000632 000638 000640 000641 000642 000646 000647 000648 000649
000652 000655 000657 000658 000659 000661 000662 000663 000664 000667 000669 000672 000673 000674 000675 000676 000677 000680
000681 000683 000685 000687 000688 000689 000690 000691 000692 000697 000700 000701 000703 000705 000707 000711 000716 000718
000722 000724 000725 000726 000727 000730 000731 000734 000735 000739 000742 000743 000744 000750 000751 000752 000756 000757
000758 000760 000762 000765 000773 000774 000775 000776 000778 000779 000780 000781 000782 000786 000789 000790 000791 000794
000798 000799 000801 000802 000803 000804 000805 000807 000808 000809 000810 000811 000813 000814 000816 000818 000819 000820
000821 000824 000827 000828 000829 000830 000834 000835 000836 000838 000842 000845 000846 000848 000850 000858 000859 000864
000868 000870 000872 000873 000876 000877 000878 000880 000883 000887 000888 000889 000892 000893 000894 000895 000905 000906

000907 000909 000910 000911 000914 000916 000917 000918 000920 000922 000924 000925 000926 000929 000932 000934 000938 000939
000940 000941 000944 000947 000948 000951 000954 000955 000959 000961 000962 000964 000969 000970 000971 000972 000974 000975
000977 000978 000979 000980 000981 000982 000983 000985 000986 000987 000990 000991 000992 000994 000995 000998 001000 001001
001003 001004 001005 001008 001010 001011 001012 001013 001015 001016 001019 001020 001021 001023 001024 001026 001027 001028
001030 001031 001033 001035 001036 001037 001039 001042 001044 001045 001048 001049 001051 001054 001055 001057 001058 001059
001061 001063 001065 001066 001067 001069 001070 001072 001073 001075 001077 001078 001079 001081 001084 001089 001090 001092
001093 001096 001097 001101 001102 001106 001109 001110 001111 001115 001116 001117 001127 001128 001129 001130 001133 001134
001135 001140 001142 001143 001144 001145 001148 001149 001150 001151 001152 001153 001155 001156 001161 001167 001168 001171
001173 001174 001178 001179 001180 001182 001184 001187 001190 001191 001193 001196 001197 001198 001202 001205 001210 001211
001213 001214 001215 001216 001218 001219 001220 001222 001224 001225 001232 001233 001235 001236 001237 001240 001243 001245
001249 001253 001254 001256 001257 001260 001263 001264 001265 001266 001268 001271 001275 001276 001278 001280 001281 001283
001284 001287 001290 001292 001299 001301 001304 001305 001306 001307 001309 001311 001312 001314 001315 001316 001317 001318
001319 001320 001321 001322 001329 001330 001345 001346 001348 001350 001352 001353 001354 001356 001361 001364 001366 001367
001369 001373 001374 001376 001377 001378 001379 001381 001382 001383 001384 001385 001387 001388 001389 001390 001392 001393
001394 001397 001401 001403 001406 001410 001413 001414 001415 001420 001422 001426 001430 001431 001432 001434 001435 001438
001440 001442 001443 001444 001445 001446 001447 001448 001451 001452 001457 001458 001459 001461 001465 001467 001468 001469
001470 001471 001473 001476 001479 001480 001481 001482 001483 001484 001486 001487 001488 001492 001493 001494 001496 001497
001499 001500 001502 001504 001506 001508 001509 001510 001511 001513 001514 001515 001519 001522 001523 001524 001525 001528
001529 001530 001532 001534 001536 001541 001544 001546 001550 001551 001553 001554 001555 001557 001558 001561 001562 001563
001568 001569 001570 001571 001572 001573 001575 001576 001577 001579 001581 001584 001585 001586 001587 001588 001590 001591
001594 001598 001599 001604 001606 001609 001616 001620 001622 001624 001625 001626 001627 001632 001634 001637 001638 001639
001640 001642 001645 001649 001651 001654 001659 001660 001661 001662 001663 001666 001667 001668 001669 001671 001672 001674
001677 001683 001686 001687 001688 001691 001693 001696 001697 001698 001700 001710 001711 001715 001716 001717 001718 001719
001720 001721 001722 001724 001727 001731 001732 001733 001737 001738 001739 001740 001744 001746 001747 001748 001750 001751
001752 001753 001759 001763 001767 001770 001771 001773 001774 001776 001777 001778 001779 001780 001781 001782 001783 001784
001785 001787 001790 001794 001795 001798 001800 001807 001810 001813 001814 001815 001816 001817 001818 001820 001822 001823
001825 001826 001827 001828 001829 001832 001834 001837 001840 001842 001844 001846 001850 001852 001854 001856 001857 001858
001859 001861 001862 001863 001864 001869 001870 001872 001873 001874 001877 001878 001885 001887 001888 001890 001893 001895
001896 001902 001905 001907 001908 001911 001913 001914 001921 001922 001923 001925 001931 001932 001933 001934 001937 001938
001942 001943 001944 001947 001948 001949 001950 001952 001954 001955 001956 001957 001958 001959 001960 001963 001969 001974
001976 001983 001992 001995 001997 002000 002005 002006 002013 002016 002019 002022 002024 002028 002029 002031 002036 002037
002038 002039 002040 002043 002044 002046 002047 002048 002049 002052 002054 002055 002056 002057 002059 002060 002062 002065
002066 002068 002069 002072 002074 002075 002076 002079 002080 002082 002085 002087 002088 002090 002091 002092 002093 002097
002099 002100 002102 002104 002105 002106 002109 002111 002112 002115 002117 002119 002120 002121 002122 002123 002124 002125
002126 002128 002131 002133 002135 002138 002140 002143 002145 002146 002148 002149 002152 002153 002157 002158 002160 002161
002164 002165 002166 002167 002168 002172 002173 002176 002177 002178 002183 002184 002185 002186 002189 002190 002191 002192
002199 002201 002202 002204 002205 002206 002209 002214 002216 002218 002224 002225 002228 002232 002233 002234 002238 002242
002243 002245 002246 002249 002254 002256 002258 002262 002263 002264 002265 002268 002269 002270 002274 002276 002278 002279
002281 002283 002286 002291 002293 002299 002301 002304 002305 002308 002309 002311 002312 002317 002318 002319 002320 002323
002326 002329 002330 002334 002335 002336 002337 002338 002339 002342 002343 002344 002349 002350 002353 002355 002356 002360
002361 002363 002365 002367 002371 002373 002374 002378 002380 002382 002383 002385 002386 002390 002391 002392 002396 002400
002403 002405 002406 002407 002408 002410 002414 002415 002417 002419 002420 002421 002422 002424 002425 002426 002427 002428
002432 002437 002439 002441 002444 002447 002448 002455 002461 002463 002465 002466 002469 002470 002473 002474 002475 002476
002477 002478 002483 002485 002486 002489 002490 002491 002494 002495 002496 002501 002506 002509 002510 002512 002513 002514
002515 002517 002518 002522 002523 002524 002527 002529 002531 002532 002533 002535 002537 002539 002541 002542 002543 002544
002545 002546 002547 002548 002549 002550 002553 002554 002555 002557 002560 002561 002562 002565 002567 002569 002571 002572
002573 002574 002576 002580 002581 002582 002583 002584 002585 002591 002592 002594 002597 002598 002599 002601 002602 002604

002605 002607 002609 002610 002613 002614 002619 002623 002624 002628 002629 002630 002631 002633 002635 002636 002639 002640
002641 002644 002645 002646 002647 002648 002649 002652 002654 002655 002657 002659 002661 002667 002670 002674 002675 002679
002680 002684 002685 002689 002690 002691 002692 002695 002696 002698 002701 002702 002704 002706 002707 002708 002710 002712
002713 002714 002716 002718 002719 002721 002726 002727 002729 002731 002733 002737 002738 002739 002740 002742 002743 002745
002746 002747 002748 002749 002751 002755 002759 002760 002762 002765 002766 002771 002773 002774 002775 002777 002779 002780
002781 002782 002783 002785 002786 002787 002790 002792 002795 002797 002798 002799 002800 002808 002809 002810 002811 002813
002814 002818 002819 002820 002822 002826 002827 002828 002829 002830 002834 002836 002839 002840 002841 002842 002844 002846
002848 002849 002852 002854 002855 002856 002860 002863 002864 002866 002867 002869 002870 002871 002872 002873 002874 002875
002876 002877 002878 002879 002880 002881 002882 002883 002884 002887 002888 002889 002890 002893 002896 002899 002900 002901
002905 002907 002909 002910 002911 002912 002914 002917 002919 002922 002923 002924 002926 002928 002930 002931 002932 002933
002934 002935 002937 002941 002945 002946 002948 002949 002950 002951 002952 002954 002959 002962 002963 002964 002965 002966
002967 002968 002970 002973 002974 002975 002978 002980 002981 002982 002983 002984 002986 002988 002990 002993 002996 002997
003000 003001 003002 003003 003005 003006 003008 003009 003015 003017 003018 003019 003020 003025 003026 003027 003028 003031
003034 003035 003036 003039 003040 003042 003045 003046 003047 003050 003053 003054 003056 003058 003060 003066 003067 003068
003070 003071 003073 003074 003076 003081 003082 003083 003088 003089 003090 003092 003094 003095 003096 003101 003102 003103
003104 003105 003109 003112 003113 003114 003117 003119 003122 003124 003125 003127 003128 003129 003130 003131 003132 003135
003139 003140 003141 003142 003143 003145 003149 003153 003157 003161 003162 003166 003167 003168 003170 003173 003174 003175
003176 003177 003178 003179 003181 003187 003188 003189 003190 003191 003192 003194 003195 003197 003198 003200 003202 003203
003204 003205 003209 003210 003212 003213 003214 003215 003220 003224 003226 003228 003229 003232 003240 003241 003242 003247
003248 003249 003250 003257 003258 003259 003260 003261 003262 003268 003272 003278 003281 003285 003286 003288 003290 003291
003292 003294 003296 003297 003298 003300 003302 003305 003308 003309 003314 003315 003316 003317 003318 003322 003324 003326
003332 003333 003335 003338 003341 003344 003354 003359 003365 003366 003367 003370 003378 003379 003383 003387 003389 003392
003396 003398 003400 003401 003402 003403 003404 003406 003407 003408 003409 003411 003413 003414 003417 003420 003421 003423
003424 003425 003427 003430 003431 003432 003433 003434 003441 003445 003446 003447 003451 003453 003454 003455 003456 003460
003461 003463 003465 003468 003474 003475 003476 003477 003480 003483 003486 003487 003488 003489 003490 003491 003494 003495
003496 003497 003498 003500 003501 003503 003504 003505 003506 003507 003508 003510 003513 003519 003520 003521 003526 003528
003529 003531 003533 003536 003537 003543 003544 003546 003547 003549 003550 003551 003553 003554 003556 003557 003558 003559
003560 003565 003566 003567 003570 003572 003575 003576 003577 003578 003580 003581 003582 003584 003585 003589 003590 003593
003596 003599 003604 003605 003607 003609 003613 003615 003619 003620 003624 003626 003627 003628 003630 003634 003640 003646
003647 003648 003652 003653 003654 003659 003661 003662 003664 003665 003666 003667 003669 003671 003672 003673 003675 003676
003677 003682 003684 003685 003686 003688 003690 003692 003695 003696 003697 003700 003701 003705 003706 003707 003708 003709
003710 003712 003714 003715 003717 003719 003722 003725 003726 003728 003729 003732 003733 003735 003736 003737 003739 003741
003745 003750 003753 003754 003755 003759 003760 003763 003764 003766 003769 003770 003773 003774 003775 003777 003780 003781
003782 003783 003785 003787 003788 003790 003792 003797 003800 003802 003803 003804 003805 003806 003807 003808 003809 003810
003812 003813 003817 003818 003819 003820 003821 003824 003825 003827 003829 003832 003837 003840 003841 003843 003845 003846
003847 003848 003849 003850 003853 003855 003856 003858 003860 003862 003863 003865 003867 003871 003872 003873 003876 003877
003878 003880 003882 003883 003885 003886 003889 003890 003894 003899 003900 003906 003908 003909 003911 003914 003915 003917
003919 003921 003923 003925 003926 003928 003930 003933 003934 003935 003936 003938 003939 003942 003944 003945 003947 003948
003949 003951 003952 003953 003955 003957 003961 003968 003969 003970 003972 003975 003976 003977 003978 003980 003981 003984
003986 003987 003988 003989 003990 003991 003992 003994 003995 003996 004001 004004 004005 004006 004009 004010 004012 004013
004015 004016 004018 004021 004026 004030 004034 004036 004037 004038 004040 004042 004045 004050 004053 004056 004057 004059
004060 004061 004062 004066 004069 004070 004073 004077 004079 004081 004082 004084 004086 004087 004088 004089 004091 004093
004096 004102 004105 004108 004109 004113 004115 004117 004118 004124 004125 004126 004127 004129 004131 004132 004133 004134
004135 004138 004140 004141 004143 004144 004145 004148 004149 004150 004151 004152 004153 004155 004157 004158 004160 004161
004162 004163 004164 004167 004168 004169 004170 004171 004172 004175 004176 004177 004178 004180 004182 004185 004187 004188
004189 004190 004192 004193 004199 004200 004201 004202 004203 004204 004205 004206 004207 004208 004210 004213 004218 004224
004225 004227 004233 004234 004237 004240 004241 004243 004245 004247 004248 004252 004257 004260 004262 004263 004264 004265

004267 004268 004269 004272 004275 004276 004277 004279 004281 004282 004283 004285 004288 004293 004299 004300 004301 004302
004305 004306 004307 004309 004310 004313 004314 004316 004318 004319 004320 004321 004326 004327 004328 004330 004333 004336
004338 004339 004340 004341 004342 004343 004344 004345 004350 004352 004355 004357 004358 004359 004361 004364 004367 004368
004369 004370 004372 004374 004378 004379 004381 004383 004384 004385 004387 004388 004389 004392 004394 004395 004396 004397
004399 004400 004401 004405 004408 004409 004410 004411 004412 004413 004414 004417 004420 004421 004423 004426 004432 004434
004435 004436 004437 004438 004441 004442 004443 004444 004446 004449 004452 004453 004457 004461 004462 004464 004465 004467
004468 004474 004479 004480 004482 004483 004484 004487 004491 004498 004499 004500 004502 004503 004504 004506 004510 004512
004513 004514 004515 004518 004519 004521 004524 004525 004529 004530 004535 004536 004537 004540 004543 004544 004545 004547
004548 004549 004553 004554 004555 004556 004558 004559 004561 004562 004563 004565 004566 004567 004568 004569 004571 004572
004574 004575 004578 004580 004583 004584 004585 004589 004590 004591 004592 004594 004596 004598 004599 004600 004601 004605
004607 004610 004611 004615 004616 004620 004621 004622 004623 004624 004625 004627 004629 004633 004634 004637 004640 004644
004646 004647 004649 004650 004651 004652 004653 004655 004656 004661 004664 004666 004669 004671 004672 004674 004675 004679
004680 004681 004684 004686 004689 004690 004694 004696 004697 004699 004700 004702 004705 004708 004709 004710 004711 004712
004713 004716 004722 004723 004724 004725 004726 004727 004731 004734 004736 004737 004738 004739 004740 004741 004743 004744
004747 004748 004751 004754 004756 004759 004760 004764 004766 004767 004769 004771 004773 004775 004778 004781 004784 004785
004788 004793 004794 004796 004797 004798 004800 004801 004803 004804 004806 004808 004810 004813 004814 004815 004817 004818
004819 004820 004821 004823 004826 004829 004830 004831 004832 004835 004836 004837 004838 004839 004840 004842 004846 004847
004850 004851 004852 004855 004857 004861 004865 004866 004868 004869 004872 004873 004875 004879 004880 004881 004882 004883
004884 004887 004888 004889 004890 004892 004893 004895 004897 004901 004910 004911 004913 004914 004917 004918 004919 004920
004921 004922 004926 004929 004930 004932 004933 004934 004935 004936 004938 004945 004946 004948 004949 004951 004952 004956
004958 004961 004962 004963 004964 004966 004967 004968 004970 004971 004973 004976 004981 004982 004984 004985 004986 004991
004993 005000 005002 005003 005004 005005 005007 005009 005010 005011 005013 005014 005015 005020 005022 005028 005030 005031
005033 005034 005035 005036 005038 005039 005042 005043 005046 005047 005048 005050 005052 005056 005057 005059 005061 005063
005064 005065 005066 005070 005071 005073 005074 005078 005082 005083 005084 005088 005090 005091 005092 005093 005094 005095
005096 005097 005099 005101 005102 005103 005106 005107 005109 005111 005112 005114 005115 005117 005122 005123 005126 005129
005130 005131 005132 005137 005139 005140 005142 005145 005146 005147 005148 005150 005152 005154 005158 005163 005165 005167
005169 005174 005176 005177 005178 005179 005180 005181 005182 005185 005187 005188 005189 005190 005192 005193 005196 005197
005199 005201 005205 005206 005208 005209 005212 005215 005216 005217 005218 005219 005221 005222 005226 005228 005230 005232
005233 005234 005237 005238 005240 005247 005248 005250 005252 005253 005257 005260 005261 005263 005264 005268 005269 005273
005275 005278 005279 005280 005281 005282 005284 005288 005290 005294 005296 005297 005300 005303 005305 005312 005316 005317
005318 005319 005320 005321 005322 005324 005326 005327 005328 005330 005331 005333 005334 005340 005341 005342 005349 005350
005351 005355 005356 005359 005362 005364 005367 005368 005370 005372 005373 005376 005377 005378 005379 005380 005383 005384
005387 005392 005393 005394 005396 005399 005400 005401 005404 005405 005408 005409 005410 005415 005416 005418 005422 005423
005424 005426 005430 005432 005435 005437 005438 005442 005443 005444 005447 005448 005449 005450 005451 005456 005457 005461
005464 005465 005469 005470 005471 005472 005474 005475 005478 005479 005480 005482 005484 005486 005489 005490 005491 005493
005494 005496 005499 005501 005502 005504 005509 005512 005514 005515 005518 005519 005523 005524 005525 005526 005527 005529
005530 005531 005535 005538 005540 005541 005542 005545 005547 005548 005560 005561 005562 005564 005565 005566 005567 005570
005575 005576 005577 005579 005580 005581 005584 005586 005587 005589 005590 005591 005597 005600 005602 005603 005605 005607
005611 005614 005620 005622 005623 005624 005628 005631 005634 005635 005637 005638 005639 005641 005642 005643 005645 005646
005648 005649 005651 005652 005654 005655 005661 005663 005666 005669 005671 005674 005675 005678 005681 005683 005688 005689
005690 005691 005694 005695 005696 005697 005699 005702 005703 005705 005706 005708 005709 005711 005713 005716 005719 005721
005728 005730 005732 005735 005736 005742 005745 005746 005747 005753 005757 005760 005761 005763 005765 005768 005769 005772
005774 005777 005778 005779 005782 005785 005786 005787 005788 005789 005790 005792 005793 005795 005799 005805 005809 005811
005815 005817 005819 005820 005824 005826 005827 005835 005840 005842 005843 005844 005846 005847 005848 005852 005853 005854
005857 005858 005860 005861 005862 005863 005865 005866 005867 005869 005870 005872 005873 005874 005875 005877 005879 005880
005884 005886 005888 005889 005894 005896 005897 005901 005902 005903 005904 005910 005913 005914 005916 005917 005921 005922
005923 005924 005925 005927 005928 005930 005941 005942 005943 005945 005947 005949 005952 005960 005961 005964 005965 005967

005971 005974 005976 005977 005985 005988 005990 005991 005992 005993 005994 005997 006004 006005 006007 006013 006014 006016
006018 006021 006023 006024 006026 006031 006036 006038 006039 006041 006042 006048 006052 006053 006054 006055 006057 006058
006059 006061 006062 006064 006067 006070 006072 006077 006078 006080 006081 006082 006086 006087 006088 006089 006092 006096
006097 006098 006099 006101 006102 006104 006105 006106 006107 006109 006110 006112 006114 006116 006118 006122 006124 006127
006129 006130 006134 006135 006138 006139 006140 006148 006149 006152 006154 006155 006156 006157 006159 006165 006169 006170
006175 006176 006178 006180 006183 006185 006190 006193 006196 006198 006200 006203 006205 006207 006208 006210 006214 006215
006216 006217 006218 006223 006224 006226 006227 006228 006229 006230 006231 006236 006239 006240 006241 006244 006245 006246
006247 006248 006249 006252 006254 006255 006256 006258 006260 006262 006263 006265 006266 006267 006268 006269 006270 006272
006273 006274 006275 006276 006278 006281 006283 006288 006289 006290 006293 006295 006296 006297 006300 006301 006302 006304
006305 006307 006308 006310 006311 006313 006315 006321 006322 006323 006329 006333 006334 006335 006336 006337 006338 006339
006341 006345 006346 006347 006348 006349 006351 006352 006355 006358 006359 006360 006363 006364 006365 006366 006367 006369
006371 006372 006374 006379 006382 006384 006385 006386 006388 006390 006391 006392 006395 006396 006401 006402 006404 006406
006410 006412 006417 006423 006426 006427 006428 006429 006430 006431 006434 006435 006436 006437 006438 006443 006446 006448
006452 006453 006455 006457 006459 006460 006461 006462 006464 006465 006466 006470 006473 006474 006476 006478 006479 006483
006484 006487 006489 006491 006494 006495 006496 006499 006500 006501 006502 006505 006507 006508 006512 006514 006515 006518
006520 006522 006526 006527 006530 006531 006532 006533 006535 006538 006539 006542 006547 006548 006551 006553 006554 006555
006558 006559 006561 006562 006563 006566 006567 006569 006573 006574 006575 006576 006577 006578 006579 006580 006583 006585
006586 006587 006592 006594 006597 006598 006599 006600 006603 006604 006605 006606 006608 006609 006613 006614 006615 006619
006621 006623 006624 006626 006629 006632 006633 006634 006636 006638 006641 006642 006646 006647 006650 006651 006652 006654
006656 006658 006659 006663 006664 006666 006667 006668 006671 006673 006675 006676 006679 006681 006682 006683 006685 006687
006690 006692 006693 006694 006696 006701 006702 006704 006707 006708 006709 006710 006711 006715 006716 006717 006719 006720
006722 006724 006728 006730 006733 006735 006736 006737 006740 006742 006743 006744 006749 006753 006755 006756 006757 006763
006764 006765 006767 006768 006770 006772 006777 006781 006782 006783 006784 006786 006788 006789 006792 006799 006802 006804
006810 006811 006814 006815 006817 006822 006823 006826 006827 006829 006832 006836 006837 006838 006841 006843 006844 006847
006850 006852 006853 006856 006857 006858 006859 006860 006861 006862 006864 006866 006868 006869 006871 006872 006876 006880
006882 006886 006889 006891 006893 006894 006895 006897 006900 006902 006905 006906 006908 006910 006911 006912 006916 006917
006919 006921 006924 006925 006927 006928 006929 006931 006932 006933 006935 006940 006945 006946 006947 006953 006954 006956
006957 006960 006961 006962 006965 006968 006969 006970 006972 006978 006979 006982 006983 006987 006988 006989 006991 006994
006997 006998 006999 007001 007002 007004 007007 007008 007010 007011 007014 007017 007018 007020 007022 007023 007024 007026
007027 007028 007030 007033 007036 007037 007040 007041 007045 007046 007048 007049 007055 007057 007058 007060 007061 007063
007064 007065 007071 007072 007073 007075 007076 007077 007082 007083 007084 007085 007087 007088 007089 007091 007093 007094
007096 007097 007098 007099 007100 007101 007102 007103 007104 007108 007114 007115 007116 007117 007118 007120 007121 007127
007128 007129 007130 007131 007132 007138 007139 007141 007144 007145 007149 007151 007153 007154 007156 007157 007158 007159
007161 007165 007166 007167 007169 007170 007176 007177 007179 007181 007183 007184 007186 007187 007189 007190 007191 007192
007193 007195 007197 007198 007199 007201 007204 007205 007208 007211 007212 007214 007215 007216 007217 007218 007219 007220
007222 007226 007227 007228 007232 007233 007234 007236 007238 007239 007240 007246 007247 007249 007251 007252 007255 007256
007257 007260 007262 007266 007269 007271 007273 007275 007277 007278 007279 007284 007286 007288 007290 007292 007296 007297
007299 007306 007307 007308 007309 007311 007314 007316 007317 007321 007322 007323 007324 007327 007328 007335 007336 007337
007338 007341 007342 007343 007345 007346 007351 007354 007355 007360 007362 007363 007365 007366 007367 007370 007374 007376
007379 007381 007382 007383 007384 007386 007387 007399 007401 007402 007403 007404 007405 007407 007408 007409 007413 007414
007419 007420 007424 007426 007428 007430 007432 007435 007439 007441 007442 007444 007446 007447 007449 007450 007451 007453
007456 007457 007458 007459 007461 007462 007464 007468 007471 007473 007474 007475 007476 007479

A-2 Code

A-2-1 Python code

A-2-2 Core Network

```python
# Original author: Bichen Wu (bichen@berkeley.edu) 08/25/2016
# Adapted by: A.Y.A. Jonker (arnoudjonker@gmail.com) 02/11/2020

"""Neural network model base class."""

from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import os
import sys

from utils import util
import numpy as np
import tensorflow as tf

file_dir = os.path.dirname(__file__)
sys.path.append(file_dir)

# Add losses to summary for evalutaions during training and pruning
def _add_loss_summaries(total_loss):
  losses = tf.get_collection('losses')
  for l in losses + [total_loss]:
    tf.summary.scalar(l.op.name, l)

# Function to create a Tensorflow variable
def _variable_on_device(name, shape, initializer, trainable=True):
  dtype = tf.float32
  if not callable(initializer):
    var = tf.get_variable(name, initializer=initializer, trainable=trainable)
  else:
    var = tf.get_variable(
        name, shape, initializer=initializer, dtype=dtype, trainable=trainable)
  return var

# Function to create a Tensorflow variable with weight decay
def _variable_with_weight_decay(name, shape, wd, initializer, trainable=True):
  var = _variable_on_device(name, shape, initializer, trainable)
  if wd is not None and trainable:
    weight_decay = tf.multiply(tf.nn.l2_loss(var), wd, name='weight_loss')
    tf.add_to_collection('losses', weight_decay)
  return var

# Base class of the Squeezedet models
class ModelSkeleton:
  """Base class of NN detection models."""
```

```
47   def __init__(self, mc):
48     self.mc = mc
49     # Variable for dropout
50     self.keep_prob = 0.5 if mc.IS_TRAINING else 1.0
51
52     # image batch input
53     self.ph_image_input = tf.placeholder(
54         tf.float32, [mc.BATCH_SIZE, mc.IMAGE_HEIGHT, mc.IMAGE_WIDTH, 3],
55         name='image_input'
56     )
57     # A tensor where an element is 1 if the corresponding box is "responsible"
58     # for detection an object and 0 otherwise.
59     self.ph_input_mask = tf.placeholder(
60         tf.float32, [mc.BATCH_SIZE, mc.ANCHORS, 1], name='box_mask')
61     # Tensor used to represent bounding box deltas.
62     self.ph_box_delta_input = tf.placeholder(
63         tf.float32, [mc.BATCH_SIZE, mc.ANCHORS, 4], name='box_delta_input')
64     # Tensor used to represent bounding box coordinates.
65     self.ph_box_input = tf.placeholder(
66         tf.float32, [mc.BATCH_SIZE, mc.ANCHORS, 4], name='box_input')
67     # Tensor used to represent labels
68     self.ph_labels = tf.placeholder(
69         tf.float32, [mc.BATCH_SIZE, mc.ANCHORS, mc.CLASSES], name='labels')
70
71     # IOU between predicted anchors with ground-truth boxes
72     self.ious = tf.Variable(
73       initial_value=np.zeros((mc.BATCH_SIZE, mc.ANCHORS)), trainable=False,
74       name='iou', dtype=tf.float32
75     )
76
77     # Queue for feeding data during training and pruning
78     self.FIFOQueue = tf.FIFOQueue(
79         capacity=mc.QUEUE_CAPACITY,
80         dtypes=[tf.float32, tf.float32, tf.float32,
81                 tf.float32, tf.float32],
82         shapes=[[mc.IMAGE_HEIGHT, mc.IMAGE_WIDTH, 3],
83                 [mc.ANCHORS, 1],
84                 [mc.ANCHORS, 4],
85                 [mc.ANCHORS, 4],
86                 [mc.ANCHORS, mc.CLASSES]],
87     )
88
89     # Queue operation
90     self.enqueue_op = self.FIFOQueue.enqueue_many(
91         [self.ph_image_input, self.ph_input_mask,
92          self.ph_box_delta_input, self.ph_box_input, self.ph_labels]
93     )
94
95     #
96     self.image_input, self.input_mask, self.box_delta_input, \
97         self.box_input, self.labels = tf.train.batch(
98             self.FIFOQueue.dequeue(), batch_size=mc.BATCH_SIZE,
99             capacity=mc.QUEUE_CAPACITY)
```

```
100
101     # model parameters for keeping track of parameters in model
102     self.model_params = []
103
104     # model size counter
105     self.model_size_counter = [] # array of tuple of layer name, parameter size
106     # flop counter for keeping track of flops in model
107     self.flop_counter = [] # array of tuple of layer name, flop number
108     # activation counter
109     self.activation_counter = [] # array of tuple of layer name, output activations
110     self.activation_counter.append(('input', mc.IMAGE_WIDTH*mc.IMAGE_HEIGHT*3))
111
112   # Function to create network graph, to be defined by model class
113   def _add_forward_graph(self):
114     raise NotImplementedError
115
116   # Function to create interpertation graph
117   def _add_interpretation_graph(self):
118     mc = self.mc
119
120     with tf.variable_scope('interpret_output') as scope:
121       preds = self.preds
122
123       # probability
124       num_class_probs = mc.ANCHOR_PER_GRID*mc.CLASSES
125       self.pred_class_probs = tf.reshape(
126           tf.nn.softmax(
127               tf.reshape(
128                   preds[:, :, :, :num_class_probs],
129                   [-1, mc.CLASSES]
130               )
131           ),
132           [mc.BATCH_SIZE, mc.ANCHORS, mc.CLASSES],
133           name='pred_class_probs'
134       )
135
136       # confidence
137       num_confidence_scores = mc.ANCHOR_PER_GRID+num_class_probs
138       self.pred_conf = tf.sigmoid(
139           tf.reshape(
140               preds[:, :, :, num_class_probs:num_confidence_scores],
141               [mc.BATCH_SIZE, mc.ANCHORS]
142           ),
143           name='pred_confidence_score'
144       )
145
146       # bbox_delta
147       self.pred_box_delta = tf.reshape(
148           preds[:, :, :, num_confidence_scores:],
149           [mc.BATCH_SIZE, mc.ANCHORS, 4],
150           name='bbox_delta'
151       )
152
```

```
153       # number of object. Used to normalize bbox and classification loss
154       self.num_objects = tf.reduce_sum(self.input_mask, name='num_objects')
155
156     with tf.variable_scope('bbox') as scope:
157       with tf.variable_scope('stretching'):
158         delta_x, delta_y, delta_w, delta_h = tf.unstack(
159             self.pred_box_delta, axis=2)
160
161         anchor_x = mc.ANCHOR_BOX[:, 0]
162         anchor_y = mc.ANCHOR_BOX[:, 1]
163         anchor_w = mc.ANCHOR_BOX[:, 2]
164         anchor_h = mc.ANCHOR_BOX[:, 3]
165
166         box_center_x = tf.identity(
167             anchor_x + delta_x * anchor_w, name='bbox_cx')
168         box_center_y = tf.identity(
169             anchor_y + delta_y * anchor_h, name='bbox_cy')
170         box_width = tf.identity(
171             anchor_w * util.safe_exp(delta_w, mc.EXP_THRESH),
172             name='bbox_width')
173         box_height = tf.identity(
174             anchor_h * util.safe_exp(delta_h, mc.EXP_THRESH),
175             name='bbox_height')
176
177         self._activation_summary(delta_x, 'delta_x')
178         self._activation_summary(delta_y, 'delta_y')
179         self._activation_summary(delta_w, 'delta_w')
180         self._activation_summary(delta_h, 'delta_h')
181
182         self._activation_summary(box_center_x, 'bbox_cx')
183         self._activation_summary(box_center_y, 'bbox_cy')
184         self._activation_summary(box_width, 'bbox_width')
185         self._activation_summary(box_height, 'bbox_height')
186
187       with tf.variable_scope('trimming'):
188         xmins, ymins, xmaxs, ymaxs = util.bbox_transform(
189             [box_center_x, box_center_y, box_width, box_height])
190
191         # The max x position is mc.IMAGE_WIDTH - 1 since we use zero-based
192         # pixels. Same for y.
193         xmins = tf.minimum(
194             tf.maximum(0.0, xmins), mc.IMAGE_WIDTH-1.0, name='bbox_xmin')
195         self._activation_summary(xmins, 'box_xmin')
196
197         ymins = tf.minimum(
198             tf.maximum(0.0, ymins), mc.IMAGE_HEIGHT-1.0, name='bbox_ymin')
199         self._activation_summary(ymins, 'box_ymin')
200
201         xmaxs = tf.maximum(
202             tf.minimum(mc.IMAGE_WIDTH-1.0, xmaxs), 0.0, name='bbox_xmax')
203         self._activation_summary(xmaxs, 'box_xmax')
204
205         ymaxs = tf.maximum(
```

```
206             tf.minimum(mc.IMAGE_HEIGHT-1.0, ymaxs), 0.0, name='bbox_ymax')
207         self._activation_summary(ymaxs, 'box_ymax')
208
209         self.det_boxes = tf.transpose(
210             tf.stack(util.bbox_transform_inv([xmins, ymins, xmaxs, ymaxs])),
211             (1, 2, 0), name='bbox'
212         )
213
214     with tf.variable_scope('IOU'):
215       def _tensor_iou(box1, box2):
216         with tf.variable_scope('intersection'):
217           xmin = tf.maximum(box1[0], box2[0], name='xmin')
218           ymin = tf.maximum(box1[1], box2[1], name='ymin')
219           xmax = tf.minimum(box1[2], box2[2], name='xmax')
220           ymax = tf.minimum(box1[3], box2[3], name='ymax')
221
222           w = tf.maximum(0.0, xmax-xmin, name='inter_w')
223           h = tf.maximum(0.0, ymax-ymin, name='inter_h')
224           intersection = tf.multiply(w, h, name='intersection')
225
226         with tf.variable_scope('union'):
227           w1 = tf.subtract(box1[2], box1[0], name='w1')
228           h1 = tf.subtract(box1[3], box1[1], name='h1')
229           w2 = tf.subtract(box2[2], box2[0], name='w2')
230           h2 = tf.subtract(box2[3], box2[1], name='h2')
231
232           union = w1*h1 + w2*h2 - intersection
233
234         return intersection/(union+mc.EPSILON) \
235             * tf.reshape(self.input_mask, [mc.BATCH_SIZE, mc.ANCHORS])
236
237       self.ious = self.ious.assign(
238           _tensor_iou(
239               util.bbox_transform(tf.unstack(self.det_boxes, axis=2)),
240               util.bbox_transform(tf.unstack(self.box_input, axis=2))
241           )
242       )
243       self._activation_summary(self.ious, 'conf_score')
244
245     with tf.variable_scope('probability') as scope:
246       self._activation_summary(self.pred_class_probs, 'class_probs')
247
248       probs = tf.multiply(
249           self.pred_class_probs,
250           tf.reshape(self.pred_conf, [mc.BATCH_SIZE, mc.ANCHORS, 1]),
251           name='final_class_prob'
252       )
253
254       self._activation_summary(probs, 'final_class_prob')
255
256       self.det_probs = tf.reduce_max(probs, 2, name='score')
257       self.det_class = tf.argmax(probs, 2, name='class_idx')
258
```

```
259   # Function to create graph of loss from model
260   def _add_loss_graph(self):
261     mc = self.mc
262
263     with tf.variable_scope('class_regression') as scope:
264       # cross-entropy: q * -log(p) + (1-q) * -log(1-p)
265       # add a small value into log to prevent blowing up
266       self.class_loss = tf.truediv(
267           tf.reduce_sum(
268               (self.labels*(-tf.log(self.pred_class_probs+mc.EPSILON))
269                + (1-self.labels)*(-tf.log(1-self.pred_class_probs+mc.EPSILON)))
270               * self.input_mask * mc.LOSS_COEF_CLASS),
271           self.num_objects,
272           name='class_loss'
273       )
274       tf.add_to_collection('losses', self.class_loss)
275
276     with tf.variable_scope('confidence_score_regression') as scope:
277       input_mask = tf.reshape(self.input_mask, [mc.BATCH_SIZE, mc.ANCHORS])
278       self.conf_loss = tf.reduce_mean(
279           tf.reduce_sum(
280               tf.square((self.ious - self.pred_conf))
281               * (input_mask*mc.LOSS_COEF_CONF_POS/self.num_objects
282                  +(1-input_mask)*mc.LOSS_COEF_CONF_NEG/(mc.ANCHORS-self.num_objects)),
283               reduction_indices=[1]
284           ),
285           name='confidence_loss'
286       )
287       tf.add_to_collection('losses', self.conf_loss)
288       tf.summary.scalar('mean iou', tf.reduce_sum(self.ious)/self.num_objects)
289
290     with tf.variable_scope('bounding_box_regression') as scope:
291       self.bbox_loss = tf.truediv(
292           tf.reduce_sum(
293               mc.LOSS_COEF_BBOX * tf.square(
294                   self.input_mask*(self.pred_box_delta-self.box_delta_input))),
295           self.num_objects,
296           name='bbox_loss'
297       )
298       tf.add_to_collection('losses', self.bbox_loss)
299
300     # Added functionality to add for pruning structures of the model
301     if self.mc.IS_PRUNING:
302       with tf.variable_scope('l1_regularization') as scope:
303
304         gammas = [par for par in self.model_params if 'gamma' in par.name]
305         if mc.IS_PRUNING:
306           len_gammas=0
307           for g in gammas:
308             len_gammas = len_gammas + int(g.shape[0])
309
310           lamb=4.5/len_gammas
311         else:
```

```
312           lamb = 0.0
313
314         l1_regularizer = tf.contrib.layers.l1_regularizer(scale=lamb, scope=None)
315         self.gamma_loss = tf.contrib.layers.apply_regularization(l1_regularizer,
                gammas)
316
317         tf.add_to_collection('losses', self.gamma_loss)
318
319       # add above losses as well as weight decay losses to form the total loss
320       self.loss = tf.add_n(tf.get_collection('losses'), name='total_loss')
321
322     # Function to create graph for training operations
323     def _add_train_graph(self):
324       mc = self.mc
325
326       self.global_step = tf.Variable(0, name='global_step', trainable=False)
327       lr = tf.train.exponential_decay(mc.LEARNING_RATE,
328                                       self.global_step,
329                                       mc.DECAY_STEPS,
330                                       mc.LR_DECAY_FACTOR,
331                                       staircase=True)
332
333       tf.summary.scalar('learning_rate', lr)
334
335       _add_loss_summaries(self.loss)
336
337       opt = tf.train.MomentumOptimizer(learning_rate=lr, momentum=mc.MOMENTUM)
338       grads_vars = opt.compute_gradients(self.loss, tf.trainable_variables())
339       with tf.variable_scope('clip_gradient') as scope:
340         for i, (grad, var) in enumerate(grads_vars):
341           grads_vars[i] = (tf.clip_by_norm(grad, mc.MAX_GRAD_NORM), var)
342
343       apply_gradient_op = opt.apply_gradients(grads_vars,
              global_step=self.global_step)
344
345       for var in tf.trainable_variables():
346           tf.summary.histogram(var.op.name, var)
347
348       for grad, var in grads_vars:
349         if grad is not None:
350           tf.summary.histogram(var.op.name + '/gradients', grad)
351
352       with tf.control_dependencies([apply_gradient_op]):
353         self.train_op = tf.no_op(name='train')
354
355     # Function to add graph for vizualisation during training
356     def _add_viz_graph(self):
357       mc = self.mc
358       self.image_to_show = tf.placeholder(
359           tf.float32, [None, mc.IMAGE_HEIGHT, mc.IMAGE_WIDTH, 3],
360           name='image_to_show'
361       )
362       self.viz_op = tf.summary.image('sample_detection_results',
```

```
363         self.image_to_show, collections='image_summary',
364         max_outputs=mc.BATCH_SIZE)
365
366   # Convolutional layer with pruning functionality included
367   def _conv_layer(
368       self, inputs, conv_param_name, filters, size, stride,
369       slc=[[0, 0], [0, 0]], padding='SAME', pruning=False, prune_struct='',
          relu=True,
370       stddev=0.001):
371
372     mc = self.mc
373
374     with tf.variable_scope(conv_param_name) as scope:
375       channels = inputs.get_shape()[3]
376
377       if mc.LOAD_PRETRAINED_MODEL:
378         cw = self.model_weights
379         kernel_val = np.transpose(cw[conv_param_name][0], [2,3,1,0])
380         bias_val = cw[conv_param_name][1]
381       else:
382         kernel_val = tf.truncated_normal_initializer(
383             stddev=stddev, dtype=tf.float32)
384         bias_val = tf.constant_initializer(0.0)
385
386       kernel_init = tf.constant(kernel_val , dtype=tf.float32)
387       bias_init = tf.constant(bias_val, dtype=tf.float32)
388
389       kernel = _variable_with_weight_decay(
390           'kernels', shape=[size, size, int(channels), filters],
391           wd=mc.WEIGHT_DECAY, initializer=kernel_init, trainable=(not pruning))
392
393       biases = _variable_on_device('biases', [filters], bias_init,
394                                    trainable=(not pruning))
395       self.model_params += [kernel, biases]
396
397
398       # Add gammas to filters when pruning filters
399       if pruning and prune_struct=='filter':
400         gamma_val = tf.constant_initializer(1.0)
401         gamma_filter = _variable_on_device('gamma_filter', [filters], gamma_val,
402                                            trainable=(pruning))
403         kernel = tf.multiply(gamma_filter, kernel)
404         self.model_params += [gamma_filter]
405
406       # Add gammas to filter height and width when pruning those
407       if pruning and prune_struct=='f_shape' and np.size(size)>1:
408         mask_row_val = tf.initializers.identity()
409         mask_col_val = tf.initializers.identity()
410         identity_row_val = tf.initializers.identity()
411         identity_col_val = tf.initializers.identity()
412
413         gamma_row = _variable_on_device('gamma_row', [size[1], size[1]],
414                                         mask_row_val, trainable=(pruning))
```

```
415
416         gamma_col = _variable_on_device('gamma_col', [size[0], size[0]],
417                                         mask_col_val, trainable=(pruning))
418         mask_gamma_row = _variable_on_device('id_fil_row', [size[1], size[1]],
419                                              identity_row_val, trainable=False)
420         mask_gamma_col = _variable_on_device('id_fil_col', [size[0], size[0]],
421                                              identity_col_val, trainable=False)
422         mask_row_filtered = tf.multiply(mask_gamma_row, gamma_row)
423         mask_col_filtered = tf.multiply(mask_gamma_col, gamma_col)
424         kernel = tf.transpose(tf.matmul(tf.expand_dims(tf.expand_dims(
425                             mask_row_filtered, 0), 0), tf.transpose(kernel),
426                             name = 'maskmultirow'))
427         kernel = tf.transpose(tf.matmul(tf.transpose(kernel),
428                             tf.expand_dims(tf.expand_dims(
429                               mask_col_filtered, 0), 0),
430                             name = 'maskmulticol'))
431         self.model_params += [gamma_row]
432         self.model_params += [gamma_col]
433
434       conv = tf.nn.conv2d(
435           inputs, kernel, [1, stride, stride, 1], padding=padding,
436           name='convolution')
437       conv = tf.nn.bias_add(conv, biases, name='bias_add')
438
439       # Add gamma to complete layer if pruning layers
440       if pruning and prune_struct=='layer':
441         layer_gamma_val = tf.constant_initializer(1.0)
442         layer_gamma = _variable_on_device('gamma_layer', [1], layer_gamma_val,
443                                   trainable=pruning)
444         conv = tf.multiply(conv, layer_gamma)
445         self.model_params += [layer_gamma]
446
447       out_shape = conv.get_shape().as_list()
448
449       if np.size(size)>1:
450         self.model_size_counter.append(
451             (conv_param_name, (1+size[0]*size[1]*int(channels))*filters))
452         num_flops = \
453         (1+2*int(channels)*size[0]*size[1])*filters*out_shape[1]*out_shape[2]
454       else:
455         self.model_size_counter.append(
456             (conv_param_name, (1+size*size*int(channels))*filters))
457         num_flops = \
458         (1+2*int(channels)*size*size)*filters*out_shape[1]*out_shape[2]
459
460       if relu:
461         num_flops += 2*filters*out_shape[1]*out_shape[2]
462       self.flop_counter.append((conv_param_name, num_flops))
463
464       self.activation_counter.append(
465           (conv_param_name, out_shape[1]*out_shape[2]*out_shape[3])
466       )
467
```

```
468       if relu:
469         return tf.nn.relu(conv)
470       else:
471         return conv
472
473   # Pooling layer for networks
474   def _pooling_layer(
475       self, layer_name, inputs, size, stride, padding='SAME'):
476     with tf.variable_scope(layer_name) as scope:
477       out = tf.nn.max_pool(inputs,
478                            ksize=[1, size, size, 1],
479                            strides=[1, stride, stride, 1],
480                            padding=padding)
481       activation_size = np.prod(out.get_shape().as_list()[1:])
482       self.activation_counter.append((layer_name, activation_size))
483       return out
484
485   # Filter the object box prediction by probability
486   def filter_prediction(self, boxes, probs, cls_idx):
487     mc = self.mc
488
489     if mc.TOP_N_DETECTION < len(probs) and mc.TOP_N_DETECTION > 0:
490       order = probs.argsort()[:-mc.TOP_N_DETECTION-1:-1]
491       probs = probs[order]
492       boxes = boxes[order]
493       cls_idx = cls_idx[order]
494     else:
495       filtered_idx = np.nonzero(probs>mc.PROB_THRESH)[0]
496       probs = probs[filtered_idx]
497       boxes = boxes[filtered_idx]
498       cls_idx = cls_idx[filtered_idx]
499
500     final_boxes = []
501     final_probs = []
502     final_cls_idx = []
503
504     for c in range(mc.CLASSES):
505       idx_per_class = [i for i in range(len(probs)) if cls_idx[i] == c]
506       keep = util.nms(boxes[idx_per_class], probs[idx_per_class], mc.NMS_THRESH)
507       for i in range(len(keep)):
508         if keep[i]:
509           final_boxes.append(boxes[idx_per_class[i]])
510           final_probs.append(probs[idx_per_class[i]])
511           final_cls_idx.append(c)
512     return final_boxes, final_probs, final_cls_idx
513
514   # Create summaries of layers activations
515   def _activation_summary(self, x, layer_name):
516
517     with tf.variable_scope('activation_summary') as scope:
518       tf.summary.histogram(
519           'activation_summary/'+layer_name, x)
520       tf.summary.scalar(
```

```
521           'activation_summary/'+layer_name+'/sparsity', tf.nn.zero_fraction(x))
522       tf.summary.scalar(
523           'activation_summary/'+layer_name+'/average', tf.reduce_mean(x))
524       tf.summary.scalar(
525           'activation_summary/'+layer_name+'/max', tf.reduce_max(x))
526       tf.summary.scalar(
527           'activation_summary/'+layer_name+'/min', tf.reduce_min(x))
```

A-2-3 Network classes

```
1
2  """SqueezeDet+ model."""
3
4  from __future__ import absolute_import
5  from __future__ import division
6  from __future__ import print_function
7
8  import os
9  import sys
10
11 import joblib
12 from utils import util
13 from easydict import EasyDict as edict
14 import numpy as np
15 import tensorflow as tf
16 from nn_skeleton import ModelSkeleton
17 import pdb
18
19 class SqueezeDetPlusPruneFilter(ModelSkeleton):
20   def __init__(self, mc, gpu_id=0):
21     with tf.device('/gpu:{}'.format(gpu_id)):
22       ModelSkeleton.__init__(self, mc)
23       self._add_forward_graph()
24       self._add_interpretation_graph()
25       self._add_loss_graph()
26       self._add_train_graph()
27       self._add_viz_graph()
28
29   def _add_forward_graph(self):
30     """NN architecture."""
31
32     mc = self.mc
33
34     assert tf.gfile.Exists(mc.PRETRAINED_MODEL_PATH), \
35         'Cannot find pretrained model at the given path:' \
36         '  {}'.format(mc.PRETRAINED_MODEL_PATH)
37     self.model_weights = joblib.load(mc.PRETRAINED_MODEL_PATH)
38
39     conv1 = self._conv_layer(self.image_input,
40         'conv1',
```

```
41          filters=self.flt('conv1'), size=7, stride=2,
42          padding='VALID', pruning=mc.IS_PRUNING, prune_struct='filter')
43
44      pool1 = self._pooling_layer(
45          'pool1', conv1, size=3, stride=2, padding='VALID')
46
47      fire2 = self._fire_layer(
48          'fire2', pool1, s1x1=96, e1x1=64, e3x3=64, pruning=mc.IS_PRUNING)
49      fire3 = self._fire_layer(
50          'fire3', fire2, s1x1=96, e1x1=64, e3x3=64, pruning=mc.IS_PRUNING)
51      fire4 = self._fire_layer(
52          'fire4', fire3, s1x1=192, e1x1=128, e3x3=128, pruning=mc.IS_PRUNING)
53      pool4 = self._pooling_layer(
54          'pool4', fire4, size=3, stride=2, padding='VALID')
55
56      fire5 = self._fire_layer(
57          'fire5', pool4, s1x1=192, e1x1=128, e3x3=128, pruning=mc.IS_PRUNING)
58      fire6 = self._fire_layer(
59          'fire6', fire5, s1x1=288, e1x1=192, e3x3=192, pruning=mc.IS_PRUNING)
60      fire7 = self._fire_layer(
61          'fire7', fire6, s1x1=288, e1x1=192, e3x3=192, pruning=mc.IS_PRUNING)
62      fire8 = self._fire_layer(
63          'fire8', fire7, s1x1=384, e1x1=256, e3x3=256, pruning=mc.IS_PRUNING)
64      pool8 = self._pooling_layer(
65          'pool8', fire8, size=3, stride=2, padding='VALID')
66
67      fire9 = self._fire_layer(
68          'fire9', pool8, s1x1=384, e1x1=256, e3x3=256, pruning=mc.IS_PRUNING)
69
70      # Two extra fire modules that are not trained before
71      fire10 = self._fire_layer(
72          'fire10', fire9, s1x1=384, e1x1=256, e3x3=256, pruning=mc.IS_PRUNING)
73      fire11 = self._fire_layer(
74          'fire11', fire10, s1x1=384, e1x1=256, e3x3=256, pruning=mc.IS_PRUNING)
75      dropout11 = tf.nn.dropout(fire11, self.keep_prob, name='drop11')
76
77      num_output = mc.ANCHOR_PER_GRID * (mc.CLASSES + 1 + 4)
78      self.preds = self._conv_layer(dropout11,
79          'conv12',
80          filters=num_output, size=3, stride=1,
81          padding='SAME', relu=False, stddev=0.0001,
82          pruning=False)
83
84
85    def _fire_layer(self, layer_name, inputs, s1x1, e1x1, e3x3, stddev=0.01,
86        pruning=False, prune_struct='filter'):
87      """Fire layer constructor"""
88
89      sq1x1 = self._conv_layer(inputs,
90          layer_name+'/squeeze1x1',
91          filters=self.flt(layer_name+'/squeeze1x1'), size=1, stride=1,
92          padding='SAME', stddev=stddev, pruning=pruning, prune_struct=prune_struct)
93      ex1x1 = self._conv_layer(sq1x1,
```

```
94          layer_name+'/expand1x1',
95          filters=self.flt(layer_name+'/expand1x1'), size=1, stride=1,
96          padding='SAME', stddev=stddev, pruning=pruning, prune_struct=prune_struct)
97      ex3x3 = self._conv_layer(sq1x1,
98          layer_name+'/expand3x3',
99          filters=self.flt(layer_name+'/expand3x3'), size=3, stride=1,
100         padding='SAME', stddev=stddev, pruning=pruning, prune_struct=prune_struct)
101
102     return tf.concat([ex1x1, ex3x3], 3, name=layer_name+'/concat')
103
104
105   def flt(self, layer_name):
106     return self.model_weights[layer_name][0].shape[0]
```

```
1
2   """SqueezeDet+ model."""
3
4   from __future__ import absolute_import
5   from __future__ import division
6   from __future__ import print_function
7
8   import joblib
9   import tensorflow as tf
10  from nn_skeleton import ModelSkeleton
11  import pdb
12
13  class SqueezeDetPlusPruneFilterShape(ModelSkeleton):
14    def __init__(self, mc, gpu_id=0):
15      with tf.device('/gpu:{}'.format(gpu_id)):
16        ModelSkeleton.__init__(self, mc)
17
18        self._add_forward_graph()
19        self._add_interpretation_graph()
20        self._add_loss_graph()
21        self._add_train_graph()
22        self._add_viz_graph()
23
24    def _add_forward_graph(self):
25      """NN architecture."""
26
27      mc = self.mc
28      assert tf.gfile.Exists(mc.PRETRAINED_MODEL_PATH), \
29          'Cannot find pretrained model at the given path:' \
30          '  {}'.format(mc.PRETRAINED_MODEL_PATH)
31      self.model_weights = joblib.load(mc.PRETRAINED_MODEL_PATH)
32
33      conv1 = self._conv_layer(self.image_input,
34          'conv1', slc=self.slc('conv1'),
35          filters=self.flt('conv1'), size=self.flt_size('conv1'), stride=2,
36          padding='VALID', pruning=mc.IS_PRUNING, prune_struct='f_shape')
37
38      pool1 = self._pooling_layer(
```

```
39          'pool1', conv1, size=3, stride=2, padding='VALID')
40
41      fire2 = self._fire_layer(
42          'fire2', pool1, s1x1=96, e1x1=64, e3x3=64, pruning=mc.IS_PRUNING)
43      fire3 = self._fire_layer(
44          'fire3', fire2, s1x1=96, e1x1=64, e3x3=64, pruning=mc.IS_PRUNING)
45      fire4 = self._fire_layer(
46          'fire4', fire3, s1x1=192, e1x1=128, e3x3=128, pruning=mc.IS_PRUNING)
47      pool4 = self._pooling_layer(
48          'pool4', fire4, size=3, stride=2, padding='VALID')
49
50      fire5 = self._fire_layer(
51          'fire5', pool4, s1x1=192, e1x1=128, e3x3=128, pruning=mc.IS_PRUNING)
52      fire6 = self._fire_layer(
53          'fire6', fire5, s1x1=288, e1x1=192, e3x3=192, pruning=mc.IS_PRUNING)
54      fire7 = self._fire_layer(
55          'fire7', fire6, s1x1=288, e1x1=192, e3x3=192, pruning=mc.IS_PRUNING)
56      fire8 = self._fire_layer(
57          'fire8', fire7, s1x1=384, e1x1=256, e3x3=256, pruning=mc.IS_PRUNING)
58      pool8 = self._pooling_layer(
59          'pool8', fire8, size=3, stride=2, padding='VALID')
60
61      fire9 = self._fire_layer(
62          'fire9', pool8, s1x1=384, e1x1=256, e3x3=256, pruning=mc.IS_PRUNING)
63
64      # Two extra fire modules that are not trained before
65      fire10 = self._fire_layer(
66          'fire10', fire9, s1x1=384, e1x1=256, e3x3=256, pruning=mc.IS_PRUNING)
67      fire11 = self._fire_layer(
68          'fire11', fire10, s1x1=384, e1x1=256, e3x3=256, pruning=mc.IS_PRUNING)
69      dropout11 = tf.nn.dropout(fire11, self.keep_prob, name='drop11')
70
71      num_output = mc.ANCHOR_PER_GRID * (mc.CLASSES + 1 + 4)
72      self.preds = self._conv_layer(dropout11,
73          'conv12',
74          filters=num_output, size=3, stride=1,
75          padding='SAME', relu=False, stddev=0.0001,
76          pruning=False)
77
78
79    def _fire_layer(self, layer_name, inputs, s1x1, e1x1, e3x3, stddev=0.01,
80        pruning=False, prune_struct='f_shape'):
81      """Fire layer constructor"""
82
83      sq1x1 = self._conv_layer(inputs,
84          layer_name+'/squeeze1x1',
85          filters=self.flt(layer_name+'/squeeze1x1'), size=1, stride=1,
86          padding='SAME', stddev=stddev, pruning=pruning, prune_struct=prune_struct)
87      ex1x1 = self._conv_layer(sq1x1,
88          layer_name+'/expand1x1',
89          filters=self.flt(layer_name+'/expand1x1'), size=1, stride=1,
90          padding='SAME', stddev=stddev, pruning=pruning, prune_struct=prune_struct)
91      ex3x3 = self._conv_layer(sq1x1,
```

```
92          layer_name+'/expand3x3', slc=self.slc(layer_name),
93          filters=self.flt(layer_name+'/expand3x3'),
94          size=self.flt_size(layer_name+'/expand3x3'), stride=1,
95          padding='SAME', stddev=stddev, pruning=pruning,
96          prune_struct=prune_struct)
97
98      return tf.concat([ex1x1, ex3x3], 3, name=layer_name+'/concat')
99
100
101   def flt(self, layer_name):
102     return self.model_weights[layer_name][0].shape[0]
103
104   def flt_size(self, layer_name):
105     return [self.model_weights[layer_name][0].shape[2],
106             self.model_weights[layer_name][0].shape[3]]
107
108   def slc(self, layer_name):
109     if layer_name + '/expand3x3_slicer' in self.model_weights:
110       return self.model_weights[layer_name+'/expand3x3_slicer']
111     if layer_name + '_slicer' in self.model_weights:
112       return self.model_weights[layer_name+'_slicer']
113     else:
114       return [[0, 0], [0, 0]]
```

```
1
2  """SqueezeDet+ model."""
3
4  from __future__ import absolute_import
5  from __future__ import division
6  from __future__ import print_function
7
8  import joblib
9  import tensorflow as tf
10 from nn_skeleton import ModelSkeleton
11 import pdb
12
13 class SqueezeDetPlusPruneLayer(ModelSkeleton):
14   def __init__(self, mc, gpu_id=0):
15     with tf.device('/gpu:{}'.format(gpu_id)):
16       ModelSkeleton.__init__(self, mc)
17
18       self._add_forward_graph()
19       self._add_interpretation_graph()
20       self._add_loss_graph()
21       self._add_train_graph()
22       self._add_viz_graph()
23
24   def _add_forward_graph(self):
25     """NN architecture."""
26
27     mc = self.mc
28     assert tf.gfile.Exists(mc.PRETRAINED_MODEL_PATH), \
```

```
29          'Cannot find pretrained model at the given path:' \
30          '  {}'.format(mc.PRETRAINED_MODEL_PATH)
31      self.model_weights = joblib.load(mc.PRETRAINED_MODEL_PATH)
32
33      conv1 = self._conv_layer(self.image_input,
34          'conv1',
35          filters=self.flt('conv1'), size=7, stride=2,
36          padding='VALID', pruning=False)
37
38      pool1 = self._pooling_layer(
39          'pool1', conv1, size=3, stride=2, padding='VALID')
40
41      fire2 = self._fire_layer_check(
42          'fire2', pool1, s1x1=96, e1x1=64, e3x3=64, pruning=mc.IS_PRUNING)
43      fire3 = self._fire_layer_check(
44          'fire3', fire2, s1x1=96, e1x1=64, e3x3=64, pruning=mc.IS_PRUNING)
45      fire4 = self._fire_layer_check(
46          'fire4', fire3, s1x1=192, e1x1=128, e3x3=128, pruning=mc.IS_PRUNING)
47      pool4 = self._pooling_layer(
48          'pool4', fire4, size=3, stride=2, padding='VALID')
49
50      fire5 = self._fire_layer_check(
51          'fire5', pool4, s1x1=192, e1x1=128, e3x3=128, pruning=mc.IS_PRUNING)
52      fire6 = self._fire_layer_check(
53          'fire6', fire5, s1x1=288, e1x1=192, e3x3=192, pruning=mc.IS_PRUNING)
54      fire7 = self._fire_layer_check(
55          'fire7', fire6, s1x1=288, e1x1=192, e3x3=192, pruning=mc.IS_PRUNING)
56      fire8 = self._fire_layer_check(
57          'fire8', fire7, s1x1=384, e1x1=256, e3x3=256, pruning=mc.IS_PRUNING)
58      pool8 = self._pooling_layer(
59          'pool8', fire8, size=3, stride=2, padding='VALID')
60
61      fire9 = self._fire_layer_check(
62          'fire9', pool8, s1x1=384, e1x1=256, e3x3=256, pruning=mc.IS_PRUNING)
63
64      # Two extra fire modules that are not trained before
65      fire10 = self._fire_layer_check(
66          'fire10', fire9, s1x1=384, e1x1=256, e3x3=256, pruning=mc.IS_PRUNING)
67      fire11 = self._fire_layer_check(
68          'fire11', fire10, s1x1=384, e1x1=256, e3x3=256, pruning=mc.IS_PRUNING)
69      dropout11 = tf.nn.dropout(fire11, self.keep_prob, name='drop11')
70
71      num_output = mc.ANCHOR_PER_GRID * (mc.CLASSES + 1 + 4)
72      self.preds = self._conv_layer(dropout11,
73          'conv12',
74          filters=num_output, size=3, stride=1,
75          padding='SAME', relu=False, stddev=0.0001, pruning=False)
76
77
78    def _fire_layer_check(self, layer_name, inputs, s1x1, e1x1, e3x3, pruning):
79      if not layer_name +'/expand1x1' in self.model_weights.keys():
80        print('1x1pruned')
81        fire = self.fire_layer_1x1_pruned(layer_name, inputs, s1x1, e1x1, e3x3,
```

```
                                    pruning=False)
    elif not layer_name +'/expand3x3' in self.model_weights.keys():
      print('3x3pruned')
      fire = self.fire_layer_3x3_pruned(layer_name, inputs, s1x1, e1x1, e3x3,
                                    pruning=False)
    else:
      print('not pruned')
      fire = self._fire_layer(layer_name, inputs, s1x1, e1x1, e3x3,
                            pruning=pruning)
    return fire


  def _fire_layer(self, layer_name, inputs, s1x1, e1x1, e3x3, stddev=0.01,
     pruning = False, prune_struct='layer'):
    """Fire layer constructor"""

    if self.mc.LOAD_PRETRAINED_MODEL:

      sq1x1 = self._conv_layer(inputs, layer_name+'/squeeze1x1',
          filters=self.flt(layer_name+'/squeeze1x1'), size=1, stride=1,
          padding='SAME', pruning=False, prune_struct=prune_struct)
      ex1x1 = self._conv_layer(sq1x1, layer_name+'/expand1x1',
          filters=self.flt(layer_name+'/expand1x1'), size=1, stride=1,
          padding='SAME', pruning=pruning, prune_struct=prune_struct)
      ex3x3 = self._conv_layer(sq1x1, layer_name+'/expand3x3',
          filters=self.flt(layer_name+'/expand3x3'), size=3, stride=1,
          padding='SAME', pruning=pruning, prune_struct=prune_struct)

      return tf.concat([ex1x1, ex3x3], 3, name=layer_name+'/concat')

  def fire_layer_1x1_pruned(self, layer_name, inputs, s1x1, e1x1, e3x3, stddev=0.01,
      pruning = False, prune_struct='layer'):

    sq1x1 = self._conv_layer(inputs, layer_name+'/squeeze1x1',
          filters=self.flt(layer_name+'/squeeze1x1'), size=1, stride=1,
          padding='SAME', pruning=False)
    ex3x3 = self._conv_layer(sq1x1, layer_name+'/expand3x3',
          filters=self.flt(layer_name+'/expand3x3'), size=3, stride=1,
          padding='SAME', pruning=False)

    return tf.concat([ex3x3], 3, name=layer_name+'/concat')


  def fire_layer_3x3_pruned(self, layer_name, inputs, s1x1, e1x1, e3x3, stddev=0.01,
      pruning = False, prune_struct='layer'):

    sq1x1 = self._conv_layer(inputs, layer_name+'/squeeze1x1',
          filters=self.flt(layer_name+'/squeeze1x1'), size=1, stride=1,
          padding='SAME', pruning=False)
    ex1x1 = self._conv_layer(sq1x1, layer_name+'/expand1x1',
          filters=self.flt(layer_name+'/expand1x1'), size=1, stride=1,
          padding='SAME', pruning=False)

```

```
135     return tf.concat([ex1x1], 3, name=layer_name+'/concat')
136
137   def flt(self, layer_name):
138     return self.model_weights[layer_name][0].shape[0]
```

A-2-4 Remove structures from network

```
1  from __future__ import division
2  from __future__ import print_function
3
4  import os
5
6  import joblib
7  import numpy as np
8  import tensorflow as tf
9
10 from config import kitti_squeezeDetPlus_config
11 from nets import SqueezeDetPlusPruneFilter
12 from dataset import kitti
13
14 threshold = 0.1
15
16 class remove_filters(object):
17     def __init__(self, threshold,
18                  PRETRAINED_MODEL_PATH =
                         '/home/arnoud/Documents/TU/Afstuderen/Code/squeezeDet/data/pre_trained_models/Sq
19                  checkpoint_dir =
                         '/home/arnoud/Documents/TU/Afstuderen/Code/squeezeDet/data/RESULTS/filter/iter1/
20
21         # create model
22         self.mc = kitti_squeezeDetPlus_config()
23         self.mc.LOAD_PRETRAINED_MODEL = True
24         self.mc.PRETRAINED_MODEL_PATH = PRETRAINED_MODEL_PATH
25         self.mc.BATCH_SIZE = 1
26         self.mc.IS_PRUNING = True
27         self.model = SqueezeDetPlusPruneFilter(self.mc)
28
29         self.checkpoint_dir = checkpoint_dir
30         self.image_set = 'val'
31         self.data_path = './data/KITTI'
32
33         self.threshold = threshold
34         self.dic_layers_pruned = {}
35         self.dic_n_filters = {}
36         self.gammas_values = None
37
38         self.gammas = [par for par in self.model.model_params if 'gamma' in par.name]
39         self.kernels = [par for par in self.model.model_params if
               par.name[-9:-2]=='kernels']
```

```
40          self.biases = [par for par in self.model.model_params if
                par.name[-8:-2]=='biases']
41
42          with tf.Session(config=tf.ConfigProto(allow_soft_placement=True)) as sess:
43              self.restore_checkpoint(sess)
44              self.gammas_values = sess.run(self.gammas)
45              self.set_zeros(sess)
46              self.remove_filters_and_channels(sess)
47
48          tf.reset_default_graph()
49
50      # Removes all filters and channels set to zero
51      def remove_filters_and_channels(self,sess):
52          uninitialized_vars = []
53          for var in tf.all_variables():
54              try:
55                  sess.run(var)
56              except tf.errors.FailedPreconditionError:
57                  uninitialized_vars.append(var)
58
59          init_new_vars_op = tf.initialize_variables(uninitialized_vars)
60
61          sess.run(init_new_vars_op)
62
63          weights = tf.global_variables()
64
65          #dictionary to store new paramters with layer + var name as key
66          dic = {}
67          entry = []
68
69          for var in weights:
70              if 'kernels:0' in var.name:
71                  kernel = sess.run(var)
72                  n_c_in_gone = np.sum(np.all(kernel[0][0] == 0, axis =1))
73                  n_c_out_gone = np.sum(np.all(kernel[0][0] == 0, axis =0))
74                  if n_c_in_gone > 0 or n_c_out_gone > 0:
75                      kernel_pruned = np.empty([kernel.shape[0],
                            kernel.shape[1],kernel.shape[2] - n_c_in_gone
                            ,kernel.shape[3] - n_c_out_gone])
76                      for dim1 in range(0, kernel.shape[0]):
77                          for dim2 in range(0,kernel.shape[1]):
78                              kernel_pruned[dim1][dim2] =
                                    kernel[dim1][dim2][~np.all(kernel[dim1][dim2] == 0,
                                    axis =1)].T[~np.all(kernel[dim1][dim2] == 0, axis
                                    =0)].T
79                  else:
80                      kernel_pruned = kernel
81
82                  if kernel_pruned.shape[0] > 1:
83                      kernel_new = np.zeros(kernel_pruned.T.shape)
84                      for x in range(kernel_pruned.shape[0]):
85                          for y in range(kernel_pruned.shape[1]):
86                              kernel_new.T[x][y] = kernel_pruned[y][x]
```

```
87                  kernel_pruned = kernel_new
88              else:
89                  kernel_pruned = kernel_pruned.T
90
91
92          if 'biases:0' in var.name:
93              biases = sess.run(var)
94              biases_pruned = biases[biases != 0]
95              entry = [kernel_pruned, biases_pruned]
96              dic[var.name[:-9]] = entry
97              entry = []
98
99          if 'gamma_filter:0' in var.name:
100             gammas = sess.run(var)
101             n_g_gone = np.sum(gammas == 0)
102             self.dic_layers_pruned[var.name[:-8]] = n_g_gone
103             self.dic_n_filters[var.name[:-8]] = gammas.size
104             gammas_pruned = gammas[gammas!=0]
105             dic[var.name[:-8] + '_gamma'] = gammas_pruned
106
107     joblib.dump(dic,'SqueezeDetPruned1.pkl',compress=False )
108
109 # Function to loop through gammas and set zeros in the network
110 def set_zeros(self, sess):
111
112     pruned_channels = 0
113     total_channels = 0
114     for idx, gamma in enumerate(self.gammas):
115         gam_values = sess.run(gamma)
116         mask = np.where(gam_values<self.threshold)
117         gam_values[mask] = 0
118         ass_op_g = tf.assign(self.gammas[idx], gam_values)
119         sess.run(ass_op_g)
120
121         # check layer of gamma and select appropriate function
122         if 'conv1/' in gamma.name:
123             self.set_zeros_conv1(sess, mask, idx)
124
125         elif 'fire' in gamma.name:
126             if 'squeeze1x1' in gamma.name:
127                 self.set_zeros_squeeze1x1(sess, mask, idx)
128             if 'expand1x1' in gamma.name:
129                 self.set_zeros_expand1x1(sess, mask, idx)
130             if 'expand3x3' in gamma.name:
131                 self.set_zeros_expand3x3(sess, mask, idx)
132
133         pruned_channels = pruned_channels + len(mask[0])
134         total_channels = total_channels + len(gam_values)
135
136 # Function to set zeros in first conv layer and connected channels
137 def set_zeros_conv1(self, sess, mask, gamma_idx):
138     kern_values_0 = sess.run(self.kernels[gamma_idx])
139     kern_values_1 = sess.run(self.kernels[gamma_idx+1])
```

```
        bias_values_0 = sess.run(self.biases[gamma_idx])

        kern_values_1[0][0][:][mask] = 0

        for dim1 in range(0, kern_values_0.shape[0]):
            for dim2 in range(0, kern_values_0.shape[1]):
                kern_values_0[dim1][dim2].T[mask] = 0

        bias_values_0[mask] = 0

        ass_ops = [tf.assign(self.kernels[gamma_idx], kern_values_0),
                   tf.assign(self.kernels[gamma_idx+1], kern_values_1),
                   tf.assign(self.biases[gamma_idx], bias_values_0)]

        sess.run(ass_ops)
        a=0

    # Function to set zeros in s1x1 layer and connected channels
    def set_zeros_squeeze1x1(self, sess, mask, gamma_idx):
        kern_values_0 = sess.run(self.kernels[gamma_idx])
        kern_values_1 = sess.run(self.kernels[gamma_idx+1])
        kern_values_2 = sess.run(self.kernels[gamma_idx+2])
        bias_values_0 = sess.run(self.biases[gamma_idx])

        kern_values_0[0][0].T[mask] = 0
        kern_values_1[0][0][:][mask] = 0

        for dim1 in range(0, kern_values_2.shape[0]):
            for dim2 in range(0, kern_values_2.shape[1]):
                kern_values_2[dim1][dim2][:][mask] = 0

        bias_values_0[mask] = 0

        ass_ops = [tf.assign(self.kernels[gamma_idx], kern_values_0),
                   tf.assign(self.kernels[gamma_idx+1], kern_values_1),
                   tf.assign(self.kernels[gamma_idx+2], kern_values_2),
                   tf.assign(self.biases[gamma_idx], bias_values_0)]

        sess.run(ass_ops)

    # Function to set zeros in e1x1 layer and connected channels
    def set_zeros_expand1x1(self, sess, mask, gamma_idx):
        kern_values_0 = sess.run(self.kernels[gamma_idx])
        kern_values_2 = sess.run(self.kernels[gamma_idx+2])
        bias_values_0 = sess.run(self.biases[gamma_idx])

        kern_values_0[0][0].T[mask] = 0

        for dim1 in range(0, kern_values_2.shape[0]):
            for dim2 in range(0, kern_values_2.shape[1]):
                kern_values_2[dim1][dim2][:][mask] = 0

        bias_values_0[mask] = 0
```

```
193
194         ass_ops = [tf.assign(self.kernels[gamma_idx], kern_values_0),
195                    tf.assign(self.kernels[gamma_idx+2], kern_values_2),
196                    tf.assign(self.biases[gamma_idx], bias_values_0)]
197
198         sess.run(ass_ops)
199
200     # Function to set zeros in e3x3 layer and connected channels
201     def set_zeros_expand3x3(self, sess, mask, gamma_idx):
202         kern_values_min1 = sess.run(self.kernels[gamma_idx-1])
203         kern_values_0 = sess.run(self.kernels[gamma_idx])
204         kern_values_1 = sess.run(self.kernels[gamma_idx+1])
205         bias_values_0 = sess.run(self.biases[gamma_idx])
206
207         for dim1 in range(0, kern_values_0.shape[0]):
208             for dim2 in range(0, kern_values_0.shape[1]):
209                 kern_values_0[dim1][dim2].T[mask] = 0
210
211         for dim1 in range(0, kern_values_1.shape[0]):
212             for dim2 in range(0, kern_values_1.shape[1]):
213                 kern_values_1[dim1][dim2][:][mask[0]+kern_values_min1.shape[3]] = 0
214
215
216         bias_values_0[mask] = 0
217
218         ass_ops = [tf.assign(self.kernels[gamma_idx], kern_values_0),
219                    tf.assign(self.kernels[gamma_idx+1], kern_values_1),
220                    tf.assign(self.biases[gamma_idx], bias_values_0)]
221
222         sess.run(ass_ops)
223
224     # Restore checkpoint with pruning variables
225     def restore_checkpoint(self, sess):
226         saver = tf.train.Saver(self.model.model_params)
227
228         if os.path.isfile(self.checkpoint_dir + '.meta'):
229           saver.restore(sess, self.checkpoint_dir)
230
231 Pruner = remove_filters(threshold)
```

```
1  from __future__ import absolute_import
2  from __future__ import division
3  from __future__ import print_function
4
5  import os
6
7  import joblib
8  import numpy as np
9  import tensorflow as tf
10
11 from config import *
12 from nets import *
```

```
13 from dataset import kitti
14
15
16 class remove_rows_columns(object):
17     def __init__(self, threshold,
18                  PRETRAINED_MODEL_PATH = '/path_to_pretrained_model',
19                  checkpoint_dir ='path_to_checkpoint'):
20
21         #  Create network
22         self.mc = kitti_squeezeDetPlus_config()
23         self.mc.LOAD_PRETRAINED_MODEL = True
24         self.mc.PRETRAINED_MODEL_PATH = PRETRAINED_MODEL_PATH
25         self.mc.BATCH_SIZE = 1
26         self.mc.IS_PRUNING = True
27         self.model = SqueezeDetPlusPruneFilterShape(self.mc)
28
29         self.checkpoint_dir = checkpoint_dir
30         self.image_set = 'val'
31         self.data_path = './data/KITTI'
32
33         self.threshold = threshold
34         self.dic_rows_pruned = {}
35         self.dic_cols_pruned = {}
36         self.masks_values = None
37
38         self.gammas = [par for par in self.model.model_params if 'gamma' in par.name]
39         self.kernels = [par for par in self.model.model_params if
               par.name[-9:-2]=='kernels']
40         self.biases = [par for par in self.model.model_params if
               par.name[-8:-2]=='biases']
41
42         with tf.Session(config=tf.ConfigProto(allow_soft_placement=True)) as sess:
43             self.restore_checkpoint(sess)
44             self.masks_values = sess.run(self.gammas)
45             self.remove_rows_columns(sess, threshold)
46
47         tf.reset_default_graph()
48
49     def remove_rows_columns(self, sess, threshold):
50         #get old parameters dictionary for slicer variable
51         assert tf.gfile.Exists(self.mc.PRETRAINED_MODEL_PATH), \
52         'Cannot find pretrained model at the given path:' \
53         '  {}'.format(self.mc.PRETRAINED_MODEL_PATH)
54         weights_dic_old = joblib.load(self.mc.PRETRAINED_MODEL_PATH)
55
56         weights = tf.global_variables()
57         # Dictionary to store parameters to create pruned network
58         dic = {}
59         entry = []
60         remove_col = None
61         remove_row = None
62         #Slicer variable to store offset of pruned filter
63         slicer = [[0, 0], [0, 0]]
```

```
64
65          for var_i, var in enumerate(weights):
66              # logic to check if row of next kernel needs removing
67              if 'gamma_row:0' in var.name:
68                  mask_row = sess.run(var)
69                  if min(abs(mask_row[0,0]), abs(mask_row[-1,-1])) < threshold:
70                      if abs(mask_row[0,0]) < abs(mask_row[-1,-1]):
71                          remove_row = 0
72                      else:
73                          remove_row = -1
74                  dic[var.name[:-11] + '_mask_row'] = mask_row
75
76              # logic to check if column of next kernel needs removing
77              if 'gamma_col:0' in var.name:
78                  mask_col = sess.run(var)
79                  if min(abs(mask_col[0,0]), abs(mask_col[-1,-1])) < threshold:
80                      if abs(mask_col[0,0]) < abs(mask_col[-1,-1]):
81                          remove_col = 0
82                      else:
83                          remove_col = -1
84                  dic[var.name[:-11] + '_mask_col'] = mask_col
85
86              # Remove rows and column according to remove_row and remove_col booleans
87              if 'kernels:0' in var.name:
88                  kernel = sess.run(var)
89                  kernel_pruned = kernel
90                  if 'conv1/kernels:0' in var.name or 'expand3x3/kernels:0' in
                        var.name:
91                      mask_col = sess.run(weights[var_i+3])
92                      mask_row = sess.run(weights[var_i+2])
93                      # Load slicer variable if layer has had rows or cols removed
                            before
94                      if var.name[:-10] + '_slicer' in weights_dic_old.keys():
95                          slicer = weights_dic_old[var.name[:-10] + '_slicer']
96                      else:
97                          slicer = [[0,0],[0,0]]
98
99                      #Multiply kernel with row factros
100                     kernel_pruned = np.transpose(np.matmul(np.expand_dims(
101                                                 np.expand_dims(mask_row, 0), 0),
102                                                 np.transpose(kernel)))
103
104                     #Multiply kernel with col factros
105                     kernel_pruned =
                            np.transpose(np.matmul(np.transpose(kernel_pruned),
106                                        np.expand_dims(np.expand_dims(mask_col, 0), 0)))
107
108                     #If row needs to be removed, remove and store offset in slicer
109                     if remove_row != None:
110                         self.dic_rows_pruned[var.name[:-10]] = 1
111                         mask_row[remove_row][remove_row] = 0
112                         kernel_pruned = np.delete(kernel_pruned, remove_row, 1)
113                         if remove_row == 0:
```

```
                        slicer[0][1] = slicer[0][1] + 1
                    else:
                        slicer[1][1] = slicer[1][1] + 1
                    remove_row = None
                else:
                    self.dic_rows_pruned[var.name[:-10]] = 0

                #If col needs to be removed, remove and store offset in slicer
                if remove_col != None:
                    mask_col[remove_col][remove_col] = 0
                    self.dic_cols_pruned[var.name[:-10]] = 1
                    kernel_pruned = np.delete(kernel_pruned, remove_col, 0)
                    if remove_col == 0:
                        slicer[0][0] = slicer[0][0] + 1
                    else:
                        slicer[1][0] = slicer[1][0] + 1
                    remove_col = None
                else:
                    self.dic_cols_pruned[var.name[:-10]] = 0

                dic[var.name[:-10]+'_slicer'] = slicer
                slicer = [[0,0],[0,0]]

            if kernel_pruned.shape[0] > 1:
                kernel_new = np.zeros((kernel_pruned.shape[3],
                                       kernel_pruned.shape[2],
                                       kernel_pruned.shape[0],
                                       kernel_pruned.shape[1],
                                       ))
                for x in range(kernel_pruned.shape[0]):
                    for y in range(kernel_pruned.shape[1]):
                        kernel_new.T[y][x] = kernel_pruned[x][y]
                kernel_pruned = kernel_new
            else:
                kernel_pruned = kernel_pruned.T

          if 'biases:0' in var.name:
              biases_pruned = sess.run(var)
              entry = [kernel_pruned, biases_pruned]
              dic[var.name[:-9]] = entry
              entry = []

      joblib.dump(dic,'SqueezeDetPruned1.pkl',compress=False )

  def restore_checkpoint(self, sess):
      saver = tf.train.Saver(self.model.model_params)
      if os.path.isfile(self.checkpoint_dir + '.meta'):
        saver.restore(sess, self.checkpoint_dir)

Pruner = remove_rows_columns(0.1)
```

```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import os

import joblib
import numpy as np
import tensorflow as tf
import pdb

from config import *
from nets import *


class l1_model_pruner(object):
    def __init__(self, threshold,
                 PRETRAINED_MODEL_PATH ='/path_to_weigths',
                 checkpoint_dir ='/path_to_checkpoint'):

        # Create network
        self.mc = kitti_squeezeDetPlus_config()
        self.mc.LOAD_PRETRAINED_MODEL = True
        self.mc.PRETRAINED_MODEL_PATH = PRETRAINED_MODEL_PATH
        self.mc.BATCH_SIZE = 10
        self.model = SqueezeDetPlusBNPrune(self.mc)

        self.checkpoint_dir = checkpoint_dir
        self.image_set = 'val'
        self.data_path = '../data/KITTI'

        self.threshold = threshold
        self.lay_gammas = np.array([])
        self.dic_layers_pruned = {}

        self.gammas = [par for par in self.model.model_params if
            par.name[-13:-2]=='layer_gamma']
        self.kernels = [par for par in self.model.model_params if
            par.name[-9:-2]=='kernels']
        self.biases = [par for par in self.model.model_params if
            par.name[-8:-2]=='biases']

        with tf.Session(config=tf.ConfigProto(allow_soft_placement=True)) as sess:
            self.restore_checkpoint(sess)
            self.remove_layers(sess, threshold)
        tf.reset_default_graph()

    # Remove layers from network and store information in dictionary
    def remove_layers(self,sess, threshold):

        weights = tf.global_variables()
        #dictionary for storing variables with layer name as key
```

```
        dic = {}
        entry = []
        remove_layer = False
        cut_squeeze1 = False
        cut_squeeze2 = False
        lay_gamma = 1

        for v_num, var in enumerate(weights):
            # If factor is below threshold, set layer to be removed
            if 'layer_gamma:0' in var.name:
                lay_gamma = sess.run(var)
                self.lay_gammas = np.append(self.lay_gammas, lay_gamma)
                if lay_gamma < threshold:
                    self.dic_layers_pruned[var.name[:-13]] = True
                    remove_layer = True
                    if 'expand3x3' in var.name:
                        cut_squeeze1 = True
                    elif 'expand1x1' in var.name:
                        cut_squeeze2 = True
                    else:
                        print('no cutting error')
                        pdb.set_trace()
                else:
                    self.dic_layers_pruned[var.name[:-13]] = False

            # Remove layer if boolean of layer says so
            if 'kernels:0' in var.name:
                kernel = sess.run(var)
                if 'squeeze1x1/kernels' in var.name and cut_squeeze1:
                    kernel = kernel[:, :, :int(kernel.shape[2]/2), :]
                    cut_squeeze1 = False
                elif 'squeeze1x1/kernels' in var.name and cut_squeeze2:
                    kernel = kernel[:, :, int(kernel.shape[2]/2):, :]
                    cut_squeeze2 = False
                elif 'conv12/kernels' in var.name and cut_squeeze1:
                    kernel = kernel[:, :, :int(kernel.shape[2]/2), :]
                    cut_squeeze1 = False
                elif 'conv12/kernels' in var.name and cut_squeeze2:
                    kernel = kernel[:, :, int(kernel.shape[2]/2):, :]
                    cut_squeeze2 = False

                #multiply layer with factor gamma
                kernel_pruned = kernel*lay_gamma
                lay_gamma = 1

                if kernel_pruned.shape[0] > 1:
                    kernel_new = np.zeros(kernel_pruned.T.shape)
                    for x in range(kernel_pruned.shape[0]):
                        for y in range(kernel_pruned.shape[1]):
                            kernel_new.T[x][y] = kernel_pruned[y][x]
                    kernel_pruned = kernel_new
                else:
                    kernel_pruned = kernel_pruned.T
```

```
103
104             if 'biases:0' in var.name:
105                 biases = sess.run(var)
106                 if not remove_layer:
107                     # Store all variables in dictionary
108                     entry = [kernel_pruned, biases]
109                     dic[var.name[:-9]] = entry
110                 remove_layer = False
111                 entry = []
112
113         joblib.dump(dic,'SqueezeDetPruned1.pkl',compress=False )
114
115     # Restore checkpoint
116     def restore_checkpoint(self, sess):
117         saver = tf.train.Saver(tf.global_variables())
118         if os.path.isfile(self.checkpoint_dir + '.meta'):
119           saver.restore(sess, self.checkpoint_dir)
120
121
122 Pruner = l1_model_pruner(0.1)
```

A-2-5 Evaluation on GPU

```
1  from __future__ import absolute_import
2  from __future__ import division
3  from __future__ import print_function
4
5  import os.path
6
7  import pickle
8  from six.moves import xrange
9  import tensorflow as tf
10 from time import process_time
11 import numpy as np
12
13 from config import *
14 from dataset import kitti
15 from utils.util import bbox_transform, Timer
16 from nets import *
17
18 FLAGS = tf.app.flags.FLAGS
19
20 tf.app.flags.DEFINE_string('dataset', 'KITTI',
21                            """Currently support PASCAL_VOC or KITTI dataset.""")
22 tf.app.flags.DEFINE_string('data_path', '../data/KITTI', """Root directory of
       data""")
23 tf.app.flags.DEFINE_string('image_set', 'val',
24                            """Only used for VOC data."""
25                            """Can be train, trainval, val, or test""")
26 tf.app.flags.DEFINE_string('year', '2007',
```

```
27                          """VOC challenge year. 2007 or 2012"""
28                          """Only used for VOC data""")
29 tf.app.flags.DEFINE_string('eval_dir', '',
30                          """Directory where to write event logs """)
31 tf.app.flags.DEFINE_string('pretrained_model_path',
    '/home/arnoud/Documents/TU/Afstuderen/Code/squeezeDet/data/pre_trained_models/SqueezeDetPlus/Sc
32                          """Directory where to write event logs """)
33 tf.app.flags.DEFINE_string('checkpoint_path', '/',
34                          """Path to the training checkpoint.""")
35 tf.app.flags.DEFINE_integer('eval_interval_secs', 60 * 1,
36                           """How often to check if new cpt is saved.""")
37 tf.app.flags.DEFINE_boolean('run_once', False,
38                           """Whether to run eval only once.""")
39 tf.app.flags.DEFINE_string('net', 'squeezeDet+PruneFilter',
40                          """Neural net architecture.""")
41 tf.app.flags.DEFINE_string('gpu', '0', """gpu id.""")
42
43
44 def eval_once(
45     saver, ckpt_path, imdb, model, step, restore_checkpoint):
46
47   with tf.Session(config=tf.ConfigProto(allow_soft_placement=True)) as sess:
48     if restore_checkpoint:
49       saver.restore(sess, ckpt_path)
50
51     uninitialized_vars = []
52     for var in tf.all_variables():
53         try:
54             sess.run(var)
55         except tf.errors.FailedPreconditionError:
56             uninitialized_vars.append(var)
57     init_new_vars_op = tf.initialize_variables(uninitialized_vars)
58     sess.run(init_new_vars_op)
59
60     num_images = len(imdb.image_idx)
61     all_boxes = [[[] for _ in xrange(num_images)]
62                  for _ in xrange(imdb.num_classes)]
63
64     # Detection sequence, looping through all images
65     _t = {'im_detect': Timer(), 'im_read': Timer(), 'misc': Timer()}
66     num_detection = 0.0
67     process_t = np.array([])
68     for i in xrange(num_images):
69         _t['im_read'].tic()
70         images, scales = imdb.read_image_batch(shuffle=False)
71         _t['im_read'].toc()
72         _t['im_detect'].tic()
73         t_start=process_time() #Using process time to measure detection time
74         det_boxes, det_probs, det_class = sess.run(
75             [model.det_boxes, model.det_probs, model.det_class],
76             feed_dict={model.image_input:images})
77         t_stop = process_time() #Using process time to measure detection time
78         process_t = np.append(process_t, t_stop-t_start)
```

```
79          _t['im_detect'].toc()
80          _t['misc'].tic()
81          for j in range(len(det_boxes)): # batch
82              # rescale
83              det_boxes[j, :, 0::2] /= scales[j][0]
84              det_boxes[j, :, 1::2] /= scales[j][1]
85
86              det_bbox, score, det_class = model.filter_prediction(
87                  det_boxes[j], det_probs[j], det_class[j])
88
89              num_detection += len(det_bbox)
90              for c, b, s in zip(det_class, det_bbox, score):
91                  all_boxes[c][i].append(bbox_transform(b) + [s])
92              _t['misc'].toc()
93
94      if not os.path.exists(FLAGS.eval_dir + "/" + step):
95        os.mkdir(FLAGS.eval_dir + "/" + step)
96
97      #Save all evaluation data
98      pickle.dump(all_boxes, open(FLAGS.eval_dir + "/" + step + "/all_boxes.p", "wb"))
99      pickle.dump(_t, open(FLAGS.eval_dir + "/" + step + "/_t.p", "wb"))
100     pickle.dump(num_detection, open(FLAGS.eval_dir + "/" + step +
            "/num_detection.p", "wb"))
101     pickle.dump(process_t, open(FLAGS.eval_dir + "/" + step + "/process_t.p", "wb"))
102
103 def evaluate():
104   """Evaluate."""
105   os.environ['CUDA_VISIBLE_DEVICES'] = FLAGS.gpu
106
107   if not os.path.exists(FLAGS.eval_dir):
108     os.makedirs(FLAGS.eval_dir)
109
110   with tf.Graph().as_default() as g:
111
112     #Select model to evaluate
113     if FLAGS.net == 'squeezeDet+PruneFilter':
114       mc = kitti_squeezeDetPlus_config()
115       mc.BATCH_SIZE = 1
116       mc.IS_TRAINING = False
117       mc.IS_PRUNING = False
118       mc.LOAD_PRETRAINED_MODEL = True
119       mc.PRETRAINED_MODEL_PATH = FLAGS.pretrained_model_path
120       model = SqueezeDetPlusPruneFilter(mc)
121     elif FLAGS.net == 'squeezeDet+PruneFilterShape':
122       mc = kitti_squeezeDetPlus_config()
123       mc.BATCH_SIZE = 1
124       mc.IS_TRAINING = False
125       mc.IS_PRUNING = False
126       mc.LOAD_PRETRAINED_MODEL = True
127       mc.PRETRAINED_MODEL_PATH = FLAGS.pretrained_model_path
128       model = SqueezeDetPlusPruneFilterShape(mc)
129     elif FLAGS.net == 'squeezeDet+PruneLayer':
130       mc = kitti_squeezeDetPlus_config()
```

```
131       mc.BATCH_SIZE = 1
132       mc.IS_TRAINING = False
133       mc.IS_PRUNING = False
134       mc.LOAD_PRETRAINED_MODEL = True
135       mc.PRETRAINED_MODEL_PATH = FLAGS.pretrained_model_path
136       model = SqueezeDetPlusPruneLayer(mc)
137
138     imdb = kitti(FLAGS.image_set, FLAGS.data_path, mc)
139
140     no_evaluation = True
141     # Logic to load checkpoints if available
142     for file in os.listdir(FLAGS.checkpoint_path):
143       if file.endswith(".meta"):
144         saver = tf.train.Saver(model.model_params)
145         ckpt_path = FLAGS.checkpoint_path + "/" + file[:-5]
146         step = file[11:-5]
147         eval_once(saver, ckpt_path, imdb, model, step, True)
148         no_evaluation = False
149
150     if no_evaluation:
151       saver = None
152       eval_once(saver, '-', imdb, model, '0', False)
153
154 def main(argv=None): #
155   evaluate()
156
157 if __name__ == '__main__':
158   tf.app.run()
```

Bibliography

[1] S. Vieira, W. H. Pinaya, and A. Mechelli, "Using deep learning to investigate the neuroimaging correlates of psychiatric and neurological disorders: Methods and applications," *Neuroscience & Biobehavioral Reviews*, vol. 74, pp. 58–75, 2017.

[2] "A comprehensive introduction to different types of convolutions in deep learning.." https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-6692 = Accessed: 2020-03-05.

[3] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *European conference on computer vision*, pp. 818–833, Springer, 2014.

[4] R. Grosse, "Lecture 6: Backpropagation." 2018.

[5] 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< 0.5 mb model size," *arXiv preprint arXiv:1602.07360*, 2016.

[6] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, "Learning efficient convolutional networks through network slimming," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2736–2744, 2017.

[7] B. Wu, F. Iandola, P. H. Jin, and K. Keutzer, "Squeezedet: Unified, small, low power fully convolutional neural networks for real-time object detection for autonomous driving," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 129–137, 2017.

[8] v. d. K. S. v. S. I. H.-z. M. Vissers, L., "Chauffeur aan het stuur?," tech. rep., 2016.

[9] S. Singh, "Critical reasons for crashes investigated in the national motor vehicle crash causation survey," tech. rep., 2015.

[10] B. Paden, M. Čáp, S. Z. Yong, D. Yershov, and E. Frazzoli, "A survey of motion planning and control techniques for self-driving urban vehicles," *IEEE Transactions on intelligent vehicles*, 2016.

[11] "Road safety in the european union, trends, statistics and main challenges," tech. rep., 2015.

[12] A. Brunetti, D. Buongiorno, G. F. Trotta, and V. Bevilacqua, "Computer vision and deep learning techniques for pedestrian detection and tracking: A survey," *Neurocomputing*, vol. 300, pp. 17–33, 2018.

[13] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[14] X. Xu, Y. Ding, S. X. Hu, M. Niemier, J. Cong, Y. Hu, and Y. Shi, "Scaling for edge inference of deep neural networks," *Nature Electronics*, vol. 1, no. 4, p. 216, 2018.

[15] S.-C. Lin, Y. Zhang, C.-H. Hsu, M. Skach, M. E. Haque, L. Tang, and J. Mars, "The architectural implications of autonomous driving: Constraints and acceleration," in *ACM SIGPLAN Notices*, vol. 53, pp. 751–766, ACM, 2018.

[16] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[17] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[18] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[19] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

[20] B. Karlik and A. V. Olgac, "Performance analysis of various activation functions in generalized mlp architectures of neural networks," *International Journal of Artificial Intelligence and Expert Systems*, vol. 1, no. 4, pp. 111–122, 2011.

[21] J. Dai, Y. Li, K. He, and J. Sun, "R-fcn: Object detection via region-based fully convolutional networks," in *Advances in neural information processing systems*, pp. 379–387, 2016.

[22] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[23] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "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.

[24] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "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.

[25] Y. Cheng, D. Wang, P. Zhou, and T. Zhang, "A survey of model compression and acceleration for deep neural networks," *arXiv preprint arXiv:1710.09282*, 2017.

[26] A. Jonker, "Accelerating evaluations of convolutional neural networks." 2019.

[27] V. Lebedev and V. S. Lempitsky, "Fast convnets using group-wise brain damage," *CoRR*, vol. abs/1506.02515, 2015.

[28] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," *CoRR*, vol. abs/1506.02626, 2015.

[29] K. Ullrich, E. Meeds, and M. Welling, "Soft weight-sharing for neural network compression," *arXiv preprint arXiv:1702.04008*, 2017.

[30] H. Zhou, J. M. Alvarez, and F. Porikli, "Less is more: Towards compact cnns," in *European Conference on Computer Vision*, pp. 662–677, Springer, 2016.

[31] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 6869–6898, 2017.

[32] Y. Choi, M. El-Khamy, and J. Lee, "Towards the limit of network quantization," *arXiv preprint arXiv:1612.01543*, 2016.

[33] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *Advances in neural information processing systems*, pp. 2074–2082, 2016.

[34] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

Glossary

List of Acronyms

AV	Automated vehicle
fps	frames per second
KIT	Karlsruhe Institute of Technology
RGB	red green blue
PNG	Portable Network Graphics
DNN	Deep Neural Network
CNN	Convolutional Neural Network
YOLO	You Only Look Once
CE	Cross Entropy
SGD	Stochastic Gradient Descent
IOU	Intersection Over Union
GPU	Graphical Processing Unit
TTI	Toyota Technical Institute