Master of Science Thesis in Electrical Engineering
Department of Electrical Engineering, Linköping University, 2017

Semantic segmentation

- using Convolutional Neural Networks and Sparse Dictionaries

Viktor Andersson

LINKÖPING UNIVERSITY

Master of Science Thesis in Electrical Engineering

Semantic segmentation- using Convolutional Neural Networks and Sparse Dictionaries

Viktor Andersson

LiTH-ISY-EX--17/5054--SE

Supervisor: **Mikael Persson**
ISY, Linköpings universitet
Hagen Spies
Combitech AB

Examiner: **Fahad Khan**
ISY, Linköpings universitet

Computer Vision Laboratory
Department of Electrical Engineering
Linköping University
SE-581 83 Linköping, Sweden

Abstract

The two main bottlenecks using deep neural networks are data dependency and training time. This thesis proposes a novel method for weight initialization of the convolutional layers in a convolutional neural network. This thesis introduces the usage of sparse dictionaries. A sparse dictionary optimized on domain specific data can be seen as a set of intelligent feature extracting filters. This thesis investigates the effect of using such filters as kernels in the convolutional layers in the neural network. How do they affect the training time and final performance?

The dataset used here is the Cityscapes-dataset which is a library of 25000 labeled road scene images. The sparse dictionary was acquired using the K-SVD method. The filters were added to two different networks whose performance was tested individually. One of the architectures is much deeper than the other. The results have been presented for both networks. The results show that filter initialization is an important aspect which should be taken into consideration while training the deep networks for semantic segmentation.

Acknowledgments

I would like to thank my supervisors Hagen Spies and Mikael Persson for all help during the thesis work. Thanks to Hagen Speies, David Habrman, Johnny Larsson and Glenn Hult whom have been involved and supportive in my work at Combitech. Thanks to Johnny Larsson for giving me the opportunity to do my work at Combitech and also for providing all required hardware and material. And off course thanks to my examiner Fahad Khan.

Linköping, Januari 2017
Viktor Andersson

Contents

Notation

Abbreviations

Abbreviation	Meaning
#	Number of
BN	Batch normalization
CAI	Coarse annotated images
CNN	Convolutional neural network
FAI	Fine annotated images
NN	Neural network

1 Introduction

This chapter introduces the problem and declares some terms and concepts that the reader should be familiar with. The important terms are written in *italic text*.

1.1 Introducing the problem

Semantic segmentation refers to the ability to simultaneously segment and classify various objects in images. This is useful in several applications like surveillance, medicine, autonomous driving or in any other application where object recognition is demanded. A common method to achieve this is by using *artificial deep neural networks*. This is a method that adopts the way a brain works and the goal is for the "artificial brain" to learn how to analyze the images. This is usually done by using a huge amount of input data and by trial and error, the same way as a human learn, become better and better with each iteration. These networks can be used for almost any application and are not limited to only image analysis. Dependent of the desired application the network architectures can be designed in a countless number of ways. One of the key to understand how the networks are trained is in the *weight parameters*. Those are weights on every connection in the network which are updated during each iteration in the training phase.

The main difficulty is that training a deep neural network is time consuming and requires a lot of data which might be expensive or hard to acquire. To overcome this issue, one possibility is to initialize the network with *pretrained weights* in combination with their corresponding architecture. This approach is common among researchers, but doing so removes the possibility to choose an arbitrary architecture which leads us to the development of this thesis. "Can the initial weight parameters be acquired in a better way than from another pretrained network?" Two recent approaches [11, 14] have shown that the weight initialization

is really important and affects both convergence speed and final results. The proposed solution in this thesis is to incorporate the usage of *sparse dictionaries* which is a dictionary of small convolutional filters kernels. A sparse dictionary optimized on domain specific data can be seen as a set of intelligent feature extracting filters. This thesis investigates how the training time and final performance are affected by using such filters as kernels in the convolutional layers in the neural network.

1.2 Background

Whether a computer is cognitive or not has been discussed among philosophers ever since the computers became more advanced in the middle of the 19th century. Yet, this question is as important from an engineers point view, especially in the field of artificial intelligence. What if the computer not only did the computation but also understood what it just computed and were able to interpret the meaning of the output. There are several fields in which this is an interesting as well as important feature, for example in object classification in images. Classic segmentation methods such as thresholding, region growing, water shed, and active contours do their job to segment the objects but they all lack the ability to interpret the output, there are no semantics in the algorithms. When artificial neural networks entered the scene it revolutionized the research in computer vision and today some systems can perform even better than humans [5]. *Convolutional neural networks* is a subcategory of neural networks that is commonly used for computer vision purposes. As the name indicates they are using convolution as a step in the computation. *Back propagation* has been around since 1989 and is the standard method used to update the *weight parameters*. It aims to minimize the *loss-function* which is a function of the difference between the predicted and desired output of the network. It calculates the gradient of the loss-function with respect to all weights and updates the weights accordingly to move to an optimum. *Stochastic Gradient Descend* is a simplified version of this where the gradient is calculated using only a subset of the images, a *mini-batch*, instead of the whole dataset. This method is much faster but the result is only an approximation of the gradient. Using a mini-batch may also be more efficient than training one image sample at the time since the computation can be made in parallel on modern graphic cards.

Using convolutional neural networks and Stochastic Gradient Descend have shown good results in image classification and object detection. Their suitable properties have led to successful results in pixel wise semantic segmentation, using various types of convolutional networks [2, 23, 24, 25, 29]. All those networks show decent or good performance, but the real problem is to get a perfect classifier. In those networks the number of weight parameters to be learned are in the range of 10:th or 100:th of millions. It is easy to realize that one need at least one scalar from the input data in order to put values on each weight in the network. Hence independently of the architecture a huge amount of data is always required to train these types of networks. There are several challenges

available on the Internet [6, 8, 9, 27], where researchers compete to develop the best performing semantic classifier. In these challenges the data is provided by the organizers. In most commercial cases though the data has to be mined by the company or research group themselves and the data dependency suddenly becomes a problem. Even though the computers processing power has exploded the last decades and the amount of data available online are huge, still some preprocessing is always required. For computer vision purposes the required preprocessing is usually labeling, which is an expensive method since it needs to be done by hand. Due to the lack of data the expenses arises and prevents both researchers and commercial developers from using the methods they want. To overcome this problem this thesis proposes the usage of sparse dictionaries. There are several ways to compute a sparse dictionary [1, 3, 15]. A method called *K-SVD* [1] is used in this thesis which is further described in section 3.6. The hypothesis is that the usage of such filters will affect the convergence speed and the final result of the classifier. If this is true it could possibly stimulate the future development of deep neural networks by encouraging the researchers in the field to try other potential network architectures.

1.3 Purpose

The main goal is to find out if the usage of domain specific sparse dictionaries as convolutional filters in a convolutional neural network affects the performance and accuracy. Is it possible to reduce the data amount or decrease the training time using this method?

1.4 Problem

The addressed problem in this thesis is the lack of smart parameter initialization when using convolutional neural networks for the problem of semantic segmentation. The most common initialization is either to use MSRA [14] which is the state of the art random initialization, or copying an existing architecture. This thesis proposes the following as a potential solution. (Figure 1.1)

- Use Cityscapes dataset.
- Compute sparse dictionaries using K-SVD algorithm.
- Insert the dictionary as convolutional filters in the networks.
- Train the modified networks.
- Train baseline networks.
- Compare the performance.

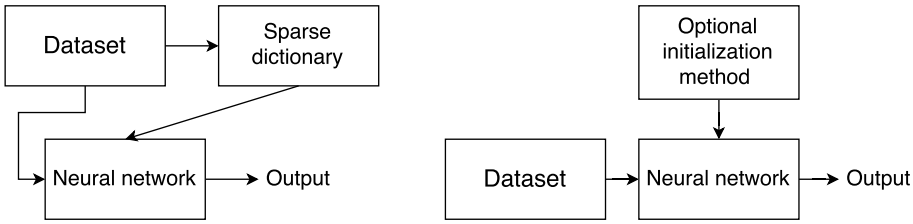


Figure 1.1: *The left flowchart shows the proposed solution for this thesis. The right flowchart shows a more common approach used as baseline in the thesis.*

1.5 Limitations

Architectures

This thesis will not consider different NN architectures but mainly focus on comparison between the reference network and two modified networks of the same type.

Training

Due to time limitations the training is performed in a rather simple way and not optimized to get the best accuracy. This thesis suggests a simple two step training process.

Data

Only one type of domain specific data is used, the Cityscapes dataset, which is a collection of road scene images.

Sparse dictionaries

K-SVD is the only method used to compute the sparse dictionaries.

2

Theory

This chapter describes the core concepts of neural networks and how they work. It will be kept short and briefly explains the fundamentals required to understand the thesis.

2.1 Artificial Neural Networks

The very first idea of neural networks was to imitate the behaviour of neurons in organisms. Each neuron has input from several other neurons and if the sum of the inputs is high enough that neuron is triggered and produces an output. Figure 2.1 shows the functionality and equations 2.1 - 2.2 describe the mathematical relationship between input and output.

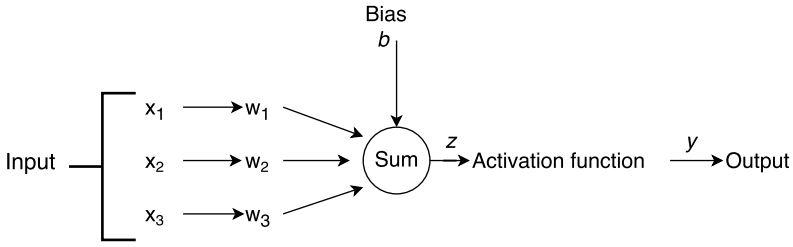


Figure 2.1: *Simple model of a node in a neural network. The input vector* $\mathbf{x}$ *are multiplied by the weights* $\mathbf{w}$*, a bias is added and then summed up. The signal is mapped in the activation function and produces an output.*

$$z = \sum_{i=1}^{n} w_i * x_i + b \tag{2.1}$$

$$y = f(z) \tag{2.2}$$

Where n is the number of input neurons, x is the feature (usually a scalar) w is the corresponding weight, b is the bias, f is the activation function and y the output. A commonly used activation function is the Rectified linear unit, ReLU, defined as:

$$f(z) = \max(0, z) \tag{2.3}$$

2.2 CNN

In a Convolutional Neural Network (CNN) the weights are defined as the scalar values in a filter kernel. The kernel is convolved with a data layer (i.e. input image or a set of feature maps) and produces an output feature map. Each filter kernel produces one feature map. A descriptive figure of how a CNN works is shown in figure 2.2

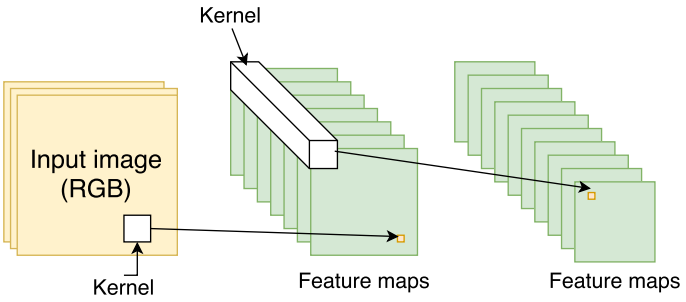


Figure 2.2: *The input image and the set of feature maps is convolved with a set of filter kernels. Each convolution produces a new feature map.*

2.3 Stochastic Gradient Descend

A predicted output map $\mathbf{p}_n$ is acquired when a sample has propagated through the network. The predicted output along with the ground truth $\mathbf{y}_n$ are passed to a differentiable loss function L which compares the data and computes an error.The loss is minimized by adjusting the weights and biases of the networks. If the set of weights and biases are defined as $\boldsymbol{W}$ then we want to minimize and find the optimal set of parameters to minimize this.

$$\min_{\boldsymbol{W}} \frac{1}{N} \sum_{n=1}^{N} L(\mathbf{p}_n, \mathbf{y}_n | \boldsymbol{W}) \tag{2.4}$$

Where N is the total number of training samples and $\mathbf{p}_n$ and $\mathbf{y}_n$ is the predicted and desired output of the sample corresponding to training sample $\mathbf{x}_n$. The method used for adjusting the parameters is referred to as *backpropagation*. With

gradient descent, backpropagation propagates the gradients of the loss function with respect to the parameters back through the network using the chain rule [13]. As mentioned in chapter 1 Stochastic Gradient Descend is an optimization algorithm used to find the solution to equation 2.4. A mini-batch of randomly selected samples is used to calculate the average gradient with respect to each weight using the loss-function. The gradients are used to update the weights according to equation 2.5. An increased mini-batch size better approximates the true gradient of the complete dataset.

$$\boldsymbol{W}^{t+1} \leftarrow \boldsymbol{W}^{t} - \frac{\alpha}{m} \sum_{n=1}^{m} \frac{\partial L(\mathbf{p}_n, \mathbf{y}_n | \boldsymbol{W})}{\partial \boldsymbol{W}} - \lambda \alpha \boldsymbol{W} \tag{2.5}$$

Where α is the learning rate, m is the batch size, λ is the weight decay and t the iteration number. Each iteration the weights are updated and moves one step closer to optimum.

3

Related work

The thesis has been inspired by several projects and articles in the same field. Some of the concepts are reused and combined to develop and evaluate a novel method for semantic segmentation. This chapter briefly describes the most relevant articles associated with this thesis and in detail describes the work of the articles from which most of the inspiration has been acquired.

3.1 Convolutional Neural Networks

Deep learning has recently seen huge success in tasks like image labeling [8] and object detection in images [29]. Other popular subjects also investigated are handwritten digit recognition[22] and speech recognition [16]. In the field of object detection, and especially semantic segmentation there are several challenges available online [6, 8, 9, 27]. Some of the participants use various types of convolutional neural networks to solve these tasks [2, 4, 10, 23, 24, 25, 29]. These networks have similar architectures but also differs a lot. The encoding part is the same for all of them whilst the main differences are how the decoding (upsampling and pixel wise classification) are performed. In some cases [4, 24], conditional random field (CRF) described in [20] is applied to the output map which has a smoothing effect on the image.

VGG [29] provides a network architecture as well as pretrained weights acquired from training on the ImageNet dataset [28]. The core is 16 or 19 layers of convolution followed by two fully connected 4096 nodes layers and a softmax layer. The method of [2, 10, 23, 24] all share the convolutional part of this architecture. One of these methods called FCN [23] faces the problem of falsely classification when the object is too big and fails to interpret the whole picture. An example is shown in figure 3.1.In paper [24] there is an example of a falsely classified buss, which seem to be a composition of several smaller objects.

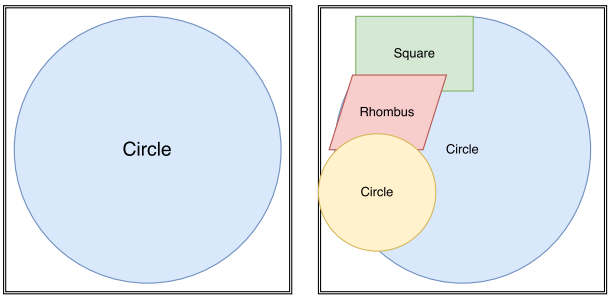
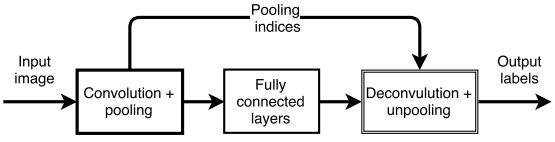
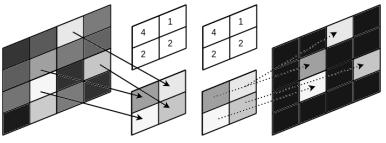


Figure 3.1: *The circle to the left is to big to be classified correctly. It is interpreted to be a composition of several other objects.*

The work of [24] tries to solve this problem by developing a novel method for decoding which they call *deconvolution*. Instead of only using the output from the previous layer as input to the next they try to add information from an earlier state in the network. Since SegNet is symmetric they create a link between each downsampling layer and the corresponding upsampling layer. The downsampling is made by a method called *max-pooling* where the pixel with the greatest value within an optional sized area is saved and the others removed. The indices of these pixels during max-pooling are stored in switch variables and put back at the same positions during unpooling. A figure of how the deconvolution and unpooling works is shown in 3.2. The switch variables provides a direct link from the current layer to the input image, even from early layers in the model, allowing each layer to be trained with respect to the image, rather than the output of the previous layer.



(a) *The general method of a deconvolutional network.*



(b) *The indices (1-4) from the max-pooling are reused for unpooling. Image inspired by [24].*

Figure 3.2: *A general description of the flow through a network and how the indices are passed between encoding and decoding.*

3.1.1 SegNet

This thesis is based on an architecture called *SegNet* [2]. It is a fully convolutional neural network using 13 encoding layers followed by 13 decoding layers. It was inspired by [24] and uses the same core concepts. One difference is however the fully connected layers which SegNet completely removes, while centers the architecture around two layers of 4096 fully connected neurons. The middle box

in figure 3.2 a) shows this. The removal of the fully connected layers greatly decreases the total number of weight parameters and consequently also training time and computational cost. The parameter number changes from 134 Millions in VGG16 to 14.7 Millions which makes SegNet a lot easier to train than some other networks

This is an interesting note as some researchers [18] has shown that the number of parameters could be reduced and still keep the same accuracy. The work in [18] reduces the 60 million parameters in AlexNet [21] to 50x fewer parameters and retain the same accuracy. Another technique is used by (S. Han) [12] witch reduces both VGG and AlexNet by 40 to 50 times without affecting the performance.

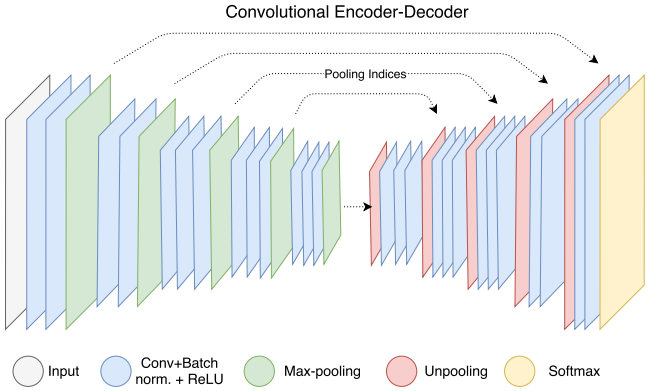


Figure 3.3: *SegNet architecture. The kernel size and the amount of layers are declared in detail in figure 4.2. Image inspired by [2].*

The specified architecture of SegNet gives a receptive field of 181 pixels. The receptive field is a crucial parameter since the field of view depends on the input image. The network will not perform on the same level if the resolution is changed. SegNet suggests an image size of 480x360 which means that the networks field of view is approximately one half and one third of the image vertically vs horizontally respectively.

3.2 Optimization and Regularization

In the SegNet architecture a layer called "batch normalization" (BN) [19] is included. BN is a method that both optimizes the training speed and reduces a phenomenon called overfitting. Overfitting causes the the classifier to perform very well on the training set, including outliers, while it lacks the ability to generalize to unseen data. Another paper [17] also pointed out this phenomenon and argues that their method called "dropout" effectively overcomes this problem by randomly turning of neurons each iteration. Their approach was however questioned by SegNet since [19] argues that their method addresses the issue which

making dropout unnecessary and only causing the training speed to drop.

3.2.1 Batch Normalization

As the name indicates this method takes a batch rather than a single sample and normalizes the data. [19] shows that using mini-batches are helpful in several ways. As mentioned in section 2.3 Stochastic Gradient Descend uses a mini-batch to approximate the gradient and optimize the weight parameters. Although Stochastic Gradient Descend is efficient the method has a couple of drawbacks. The most crucial one is the careful tuning required when initializing the learning rate parameter as well as the selection of the initial weights. The training is complicated since the input to one layer depends on the parameters in all previous layers, causing small changes in the network parameters to amplify when the data reaches deeper layers. This is a problem since the network constantly needs to adapt to the new distribution in the network. The phenomenon is refereed to as internal covariance shift. Batch normalization was developed to reduce the internal covariance shift by a normalization step that fixes the means and variances of each layer's inputs. This makes the network less sensible to the learning rate parameter which allows to increase that value. This causes batch normalization to primarily be an optimization method reducing training times. The method described by [19] is summarized below.

Let the feature vector to each layer be $x_{1...m}$ and the corrresponding normalized values be $\hat{x}_{1...m}$ and the linear transformation be $y_{1...m}$ The transformation is then described by:

$$\mathbf{BN}_{\gamma,\beta} : x_{1...m} \rightarrow y_{1...m} \tag{3.1}$$

Where γ andβ are the parameters to be learned during training. Algorithm 1 describes the the method.

The last step in the algorithm is interesting since it allows the system to get the original values of x_i back. If $\gamma = \sqrt{\sigma_B^2 + \epsilon}$ and $\beta = \mu_B$. Then $y_i = x_i$. Batch normalization uses statistics from the mini-batches during training. At test, statistics from the whole dataset should be used.

A benefit of BN apart from the improved optimization is *regularization*. When training with batch normalization, the training sample is seen in conjunction with the other samples in the mini-batch and the network does no longer produce deterministic values for a specific sample. This increases the generalization and prevents the network from relying on specific features from the previous layer. Nevertheless the generalization is dependent on the mini-batch size. The greater the mini-batch size is the greater is the generalization. If the mini-batches are small it could be of interest to enable dropout.

Input: Values of x over a mini-batch: $B = x_{1...m}$
Output: $y_i = \mathbf{BN}_{\gamma,\beta}(x_i)$
Parameters to be learned γ, β

$$\mu_B \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \tag{3.2}$$

$$\sigma_B^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_B)^2 \tag{3.3}$$

$$\hat{x}_i \leftarrow \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \tag{3.4}$$

$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \mathbf{BN}_{\gamma,\beta}(x_i) \tag{3.5}$$

Algorithm 1: Batch mean and variance are calculated in the first two steps. In equation 3.4 the current sample is normalized and in equation 3.5 the sample is scaled and shifted. Algorithm advanced from [19]

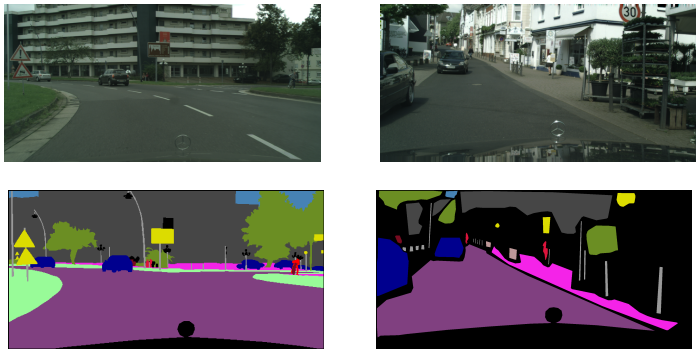
3.3 Dataset

Several datasets and challenges are available online [6, 8, 9, 27]. In this thesis the Cityscapes dataset[1] [6] was used. It consists of 20000 coarse annotated images (CAI) and 5000 fine annotated images (FAI) of which 2975 were used for training, 500 for validation and 1525 to evaluate the performance. Examples are shown in figure 3.4.

Class weights

The dataset contains 35 classes. The number of pixels representing each class has a wide distribution. This is a result of both the occurrence frequency and the size of the objects. Roads tend to be both the biggest objects in the images and are also present in all images. Traffic lights, on the other hand, are small objects and more rare. SegNet corrects for this by reducing and increasing the influence of the different classes by adding a weight to the loss-function. The weights can be calculated using a method called *median frequency balancing*, proposed by [7], described in equation 3.6 - 3.8.

[1] https://www.cityscapes-dataset.com/



(a) Data with fine annotations.

(b) Data with coarse annotations.

Figure 3.4: *Illustration of what the two different annotated images look like. The coarse annotated images are where only used during the pretraining phase and the fine annotated for fine tuning. Images from [6].*

$$\text{weight}(c) = \frac{\text{medianFrequency}}{\text{frequency}(c)} \tag{3.6}$$

Where:

$$\text{frequency}(c) = \frac{\sum_{i=1}^{I} \#\text{pixelsOfClass}(c)\text{InImage}(i)}{\#\text{imagesContainingClass}(c) * \text{imageSize}} \tag{3.7}$$

And:

$$\text{medianFrequency} = \frac{\sum_{c=1}^{N} \text{frequency}(c)}{N} \tag{3.8}$$

c = class, I = number of Images, N = number of classes.

Algorithm 2: Class weights are calculated using median frequency balancing.

Evaluation

Only 19 of the 35 represented classes are taken into account when evaluating the results. The evaluation server on Cityscapes website uses a method proposed by [8]. It measures intersection over union (IoU) which is calculated according to equation 3.9.

$$\text{IoU} = \frac{\sum \text{TP}}{\sum \text{TP} + \sum \text{FP} + \sum \text{FN}} \tag{3.9}$$

Where TP denotes true positive values for example "dogs labeled as dogs". FP denotes false positive values for example "non-dogs labeled as dogs" and FN false negative values e.g. "dogs incorrectly marked as non-dogs". To spell this out the evaluation is rewarded for the correctly labeled pixels but punished for missing pixels and mislabeled pixels.

Another method that can be used for evaluation is Accuracy. Accuracy can however be misleading since it's not punished for mislabeled pixels and the pixel accuracy can therefore be high even if the result image is just noise. Although this parameter is commonly used to monitor the learning process. It is calculated according to equation 3.10.

$$\text{Acc} = \frac{\sum \text{True positive} + \sum \text{True negative}}{\sum \text{Total population}} \tag{3.10}$$

Where true negative values are e.g. "non-dogs not labeled as dogs".

3.4 Filter initialization

Filter initialization is mainly what this thesis investigates. There have been several studies that shows that the weight initialization has big a impact on the final results of a trained network [14, 11]. Those papers also shows that good initialization allows the training to converge faster since the inner parameters are optimized. Commonly used distributions for weight initialization are constant, uniform, Gaussian, Xavier[11] and MSRA [14]. The initialization method used in this thesis is MSRA. The authors of [14] argue that a Gaussian distributed initialization has problems with convergence when the number of layers are high (e.g >8 conv layers). In VGG [29] they address this problem by pretrain a model with 8 layers to be used in deeper architectures. The drawbacks of this is the increased training time and it may also lead to a poorer local optimum. MSRA outperforms Xavier and Gaussian and it allows very deep architectures a (e.g. 30 conv. layers) to converge. According to [14] Xavier initialization assumes a linear activation function and fails to converge on deep models. The distribution of MSRA is defined in equation 3.11 and 3.12

$$x \sim N(0, \sigma^2) \tag{3.11}$$

Where the standard deviation σ depends on the number of neurons, n, in the incoming layer.

$$\sigma = \sqrt{\frac{2}{n}} \tag{3.12}$$

3.5 Sparse dictionaries

Sparse dictionaries has been proven to work well in applications like noise reduction, in painting, super resolution and compression-coding. The many successful

applications leads to the idea of using such filters in a neural network. A summary of two articles [1, 26] are descibed in the rest of this chapter. Given some input data, sparse dictionary learning aims to find a set of "basis" functions, called atoms that represent this data in some way. The set of atoms can be interpreted as an over complete optimized dictionary of small feature filters. The input data can then be represented by a linear combination of these atoms. Where the representation is be kept under a sparsity constrain. Any new input data of the same type can be expressed as a sparse linear combination of the atoms in the optimized dictionary.

The problem which sparse dictionary learning aims to solve are:

Given a dataset $\mathbf{X} = [x_1, ..., x_K]$, $x_i \in \mathbb{R}^n$
We wish to find a dictionary $\mathbf{D} \in \mathbb{R}^{n \times K} : \mathbf{D} = [d_1, ..., d_K]$
And a representation $\mathbf{R} = [r_1, ..., r_K]$, $r_i \in \mathbb{R}^K$
Such that $\|\mathbf{X} - \mathbf{DR}\|_F^2$ is minimized and r_i is sparse enough
d denotes the dimensionality of each atom and K the dictionary size.

Figure 3.5: *Sparse dictionary problem.*

Suppose that the dictionary has been optimized on patches from input images, where each column of $\mathbf{X}$ is a patch. Then figure 3.6 shows how a patch $\mathbf{y}_i$ may be described by a dictionary $\mathbf{D}$ and its sparse representation vector $\mathbf{r}_i$.

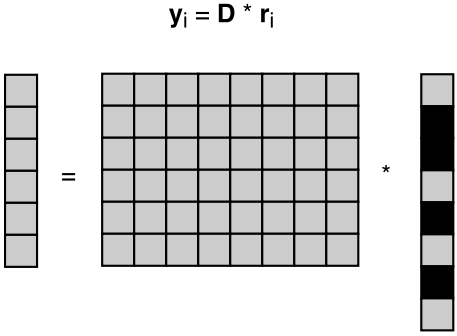


Figure 3.6: *A patch $\mathbf{y}_i$ is described by the dictionary $\mathbf{D}$ and a representation $\mathbf{r}_i$. Only a few of the atoms in the dictionary are used hence the representation is sparse.*

The problem can be formulated as an optimization problem, hence there are several possible methods to solve the problem where one commonly used is K-SVD [26], which is a patch based algorithm. A fundamental drawback of patch based algorithms is due to the assumption that input vectors (training samples) are independent of one other. The assumption leads to, when applied to natural images, basis elements that are translated versions of each other. Attempts

to avoid this was made by [3, 15], using Convolutional Sparse Coding (CSC). Excluding the translated versions of the filters seems relevant since they extract the same features anyway, and don't add any new information but only increase the computing time in the NN. In this thesis the K-SVD method was used. The correlation between the atoms was computed and the atoms was replaced if the correlation was too high.

3.6 K-SVD

K-SVD is an effective method of training sparse dictionaries for sparse signal representation via singular value decomposition. K-SVD is a generalization of the k-means clustering method and it works by iteratively alternating between updating the atoms in the dictionary and sparse coding the input data based on that dictionary [1]. This thesis adopts an optimized version of the algorithm described in [26].
Given a set of training signals $\mathbf{Y}$ we wish to find the best dictionary $\mathbf{D}$ and representation $\mathbf{R}$. The problem may be formulated as in equation 3.13, which is refereed to as our objective function.

$$\min_{\mathbf{D},\mathbf{R}} \sum_i \|\mathbf{r}_i\|_0 \quad \text{subject to} \quad \|\mathbf{Y} - \mathbf{DR}\|_F^2 \leq \epsilon \tag{3.13}$$

Where $\|.\|_0$ is the l_0-norm, counting the number of nonzero elements of a vector and ϵ is the error constrain set by the the user. Let us first consider the sparse coding step. Assume $\mathbf{D}$ is fixed and search for sparse representations with coefficients summarized in $\mathbf{R}$. The penalty term can be written as:

$$\|\mathbf{Y} - \mathbf{DR}\|_F^2 = \sum_{i=1}^{N} \|\mathbf{y}_i - \mathbf{D}\mathbf{r}_i\|_2^2 \tag{3.14}$$

The objective function 3.13 can then be decoupled into N distinct problems on the form:

$$\min_{\mathbf{r}_i} \left\{ \|\mathbf{r}_i\|_0 \right\} \quad \text{subject to:} \quad \|\mathbf{y}_i - \mathbf{D}\mathbf{r}_i\|_2^2 \leq \epsilon, \quad \text{for } i = 1, 2...., N \tag{3.15}$$

The solution of this is a NP hard problem. Hence an approximation is considered. This thesis uses orthogonal match pursuit (OMP) proposed by [26] to solve the problem.

The second step is to update the atoms in the dictionary. This process updates one column at the time, fixing all columns in $\mathbf{D}$ except one, $\mathbf{d}_k$, and tries to find new coefficients for the column that best reduce the MSE. Fixing both $\mathbf{D}$ and $\mathbf{R}$ the penalty term can be rewritten as:

$$\begin{aligned} \|\mathbf{Y} - \mathbf{DR}\|_F^2 &= \left\|\mathbf{Y} - \sum_{j=1}^{K} \mathbf{d}_j \mathbf{r}_T^j\right\|_2^F \\ &= \left\|\left(\mathbf{Y} - \sum_{j\neq k}^{K} \mathbf{d}_j \mathbf{r}_T^j\right) - \mathbf{d}_k \mathbf{r}_T^k\right\|_F^2 \\ &= \|\mathbf{E}_k - \mathbf{d}_k \mathbf{r}_T^k\|_F^2 \end{aligned} \tag{3.16}$$

Where $\mathbf{r}_T^k$ denotes the k:th row of $\mathbf{R}$. $\mathbf{E}_k$ stands for the error of all N examples when the k:th atom is removed.

For each k:

- Find the signals that currently uses $\mathbf{d}_k$ in the sparse representation (I = set of indices) and remove the other samples. This reduces the size of $\mathbf{Y}$ and $\mathbf{E}$ and the representation row $\mathbf{r}_T^k$. Figure 3.7 shows the selection of samples.

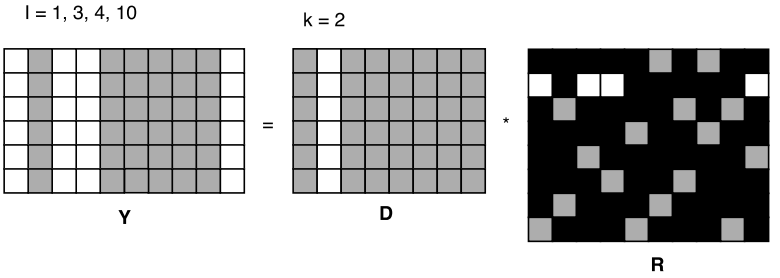


Figure 3.7: *The highlighted* ***k:th*** *row in R represents all samples in Y that uses the* ***k:th*** *atom in D for their representation.*

- Let the reduced library be expressed as $\mathbf{Y}_k^I, \mathbf{E}_k^I$ and $\mathbf{R}_k^I$. $\mathbf{R}_k^I$ suppresses the columns of $\mathbf{R}$ where $\mathbf{r}_T^k$ is zero. Let $\mathbf{r}_I^k$ be the corresponding reduced row and minimize:

$$\min_{\mathbf{d}_k, \mathbf{r}_I^k} \|\mathbf{E}_k^I - \mathbf{d}_k \mathbf{r}_I^k\|_F^2 \tag{3.17}$$

  Where $\mathbf{d}_k$ is the updated atom in $\mathbf{D}$ and $\mathbf{r}$ is the new coefficients in $\mathbf{R}_{k,I}$ and $\mathbf{R}_{k,I}$ is the k:th row in $\mathbf{R}_k^I$.

Equation 3.17 are minimized by using SVD hence the name K-SVD. Algorithm 3

shows an overview of the steps. $\mathbf{D}_k$ denotes the k:th column in $\mathbf{D}$

Input: Signal set $\mathbf{Y}$, initial dictionary $\mathbf{D}$, target error ϵ, number of iterations N.
Output: Dictionary $\mathbf{D}$ and sparse matrix $\mathbf{R}$ such that $\mathbf{Y} \approx \mathbf{DR}$
Init: Set $\mathbf{D} := \mathbf{D}_0$
for $n = 1...\ N$ **do**
  $\forall i \quad \mathbf{R}_i := \min_{\mathbf{r}} \left\{ \|\mathbf{r}_i\|_0 \right\}$ subject to: $\|\mathbf{y}_i - \mathbf{D}\mathbf{r}_i\|_2^2 \leq \epsilon$
  for $k = 1...\ K$ **do**
    $\mathbf{D}_k := 0$
    $\mathbf{E}_k^I = \mathbf{Y}_k^I - \mathbf{D}\mathbf{R}_k^I$
    $\{\mathbf{d}, \mathbf{r}\} = \min_{\mathbf{d}_k, \mathbf{r}_I^k} \|\mathbf{E}_k^I - \mathbf{d}_k \mathbf{r}_I^k\|_F^2$
    $\mathbf{D}_k := \mathbf{d}_k$
    $\mathbf{R}_{k,I} := \mathbf{r}_I^k$
  end
end

Algorithm 3: K-SVD pseudo code. Inspired by [1]

3.7 Analysis/Conclusion of related work

The work of [18] and [12] have reduced the parameters of the network and still keep the good performance. This implies that going deeper is not the only option to create better NN. As mentioned in 3.4 the initialization also affect the convergence speed and final performance. As K-SVD filters are good feature extractors the hypothesis of this thesis is to use them as initial weight in order to enhance the performance.

The choice of network is mainly based on the architecture and the compatibility with sparse dictionaries and not on the overall performance of the network. The main purpose however is to compare the difference in accuracy between initializing and not initializing with sparse dictionaries. SegNet does not perform at state of the art level but has a simple architecture where each convolutional layer can be initialized with optimized filters. The best performing networks are [31, 30] which were not available at the time this thesis started.

As mentioned there have been studies showing constant performance with reduced network parameters. An additional approach would be to not only exchange the filters but also reduce the network size. Each convolutional layer of SegNet has a constant width but varying depth. A less deep architecture but with wider filters are interesting to train and evaluate.

To reduce the number of parameters that might affect the results all networks should be trained using the same method. It seems preferable to use a fixed learning rate and momentum and use a predefined number of iterations. Many other networks uses different training methods, and a host of supporting techniques which makes it hard to evaluate and compare the actual performance. Sometimes

the test and training accuracy of a network are monitored in order to optimize the training parameters meanwhile. In this thesis the training is kept as simple as possible and the method is not used.

The authors of [19] argues in their paper that their method (BN) removes the need of dropout. I would say this is true only when the mini-batch size is big enough. A small mini-batch size in the BN layer does not support a sufficient generalization of the data set, which in turn decreases the regulating effect of the BN layer. Although the training is faster when all neurons are active and updated each iteration so dropout should be avoided if it's not necessary.

4

Method

In this chapter, specific details of implementations and adopted approaches will be explained. The Caffe framework[1] was used for training, and the Caffe-matlab interface, *matcaffe*, was used to modify the networks. Matlab was also used for general calculations and to optimize the sparse dictionaries. It start with a description of the data preprocessing. Followed by the method to calculate and insert the K-SVD filters into the network. Thereafter an explanation of how the class weights was calculated and incorporated in the training phase. And finally the network architectures and training are described.

Hardware specifications

CPU: Intel Core i7-6700K, 4 cores @ 4.00GHz
GPU: GeForce GTX 1070, 8GB
RAM: 32 GB

4.1 Preprocessing of images

The images were initially too big to be used directly in the system. This was both due to limits in the given hardware, but more importantly is the SegNet architecture optimized for images with size 360×480 (4:3 format). The images provided by Cityscapes have the 1024×2048 (2:1 format). Those images were downsampled to a size of 341×682. Due to the different formats the dimensions couldn't be matched perfectly. Hence a balancing between the receptive field in the horizontal vs the vertical direction had to be considered. It seems reasonable to have a bigger receptive field in the vertical direction. This is because of the

[1] `http://caffe.berkeleyvision.org/`

composition of objects in the image. The upper part of the images usually contains sky or building and seldom vehicles or people. If the network makes use of that information it may decrease misclassification in the vertical direction. In the horizontal direction the objects are more randomly distributed and its harder to see an intuitive generalization. Before training the data was normalized by setting the mean to zero and variance to 1 for each color channel.

4.2 K-SVD

To calculate the filters the whole training set of 2975 images were used. The filters had to be optimized iteratively starting with the first layer. The input images were used to optimize filters for the first convolutional layer. New data was propagated through the network producing an output. This output from the first layer was used to optimize filters in the second layer. A subset of 300 images was selected at each iteration to compute filters. In the first step the network is very small and has only one set of Conv/BN/ReLU layers. Each iteration the network expands and the iteration ends when filters for all layers are computed. Step 1-7 describes the method and a flowchart in figure 4.1 shows the same thing

1. Start with no network.
2. Use output data from the previous layer (images the at first iteration) as input data to compute filters for the current layer.
3. Expand the network.
4. Insert the filters into the network.
5. Propagate new data through network.
6. Save the output of the network.
7. Restart from 2.

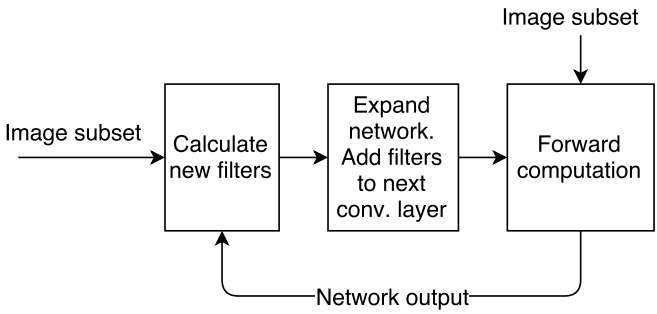


Figure 4.1: *The iterative method of how the K-SVD filters are calculated and added to the network.*

The filters were scaled to have the same mean and variance as proposed in [14] which is declared in equation 3.11 and 3.12
Since the main goal is to find filters that represent the data well the sparsity doesn't need to be low. But the training may suppress the unnecessary filter instead. The correlation factor was set to 0.9, atoms are replaced if the correlation to another atom is higher than 0.9. The usage factor to 4, atoms are replaced if they're used by less than 4 samples.

4.3 Class weights and loss

The class weights were calculated by using *median frequency balancing* as proposed by [7], see Algorithm 2. All 2975 training images were used to obtain these results which are shown in table 6.1. The class weights are passed to the Loss-function, which in this thesis is the Multinomial Logistic Loss, defined in equation 4.1. The total loss L for the current sample is the sum of the loss at each pixel position k.

$$L = -\frac{1}{N_p}\sum_{k=1}^{N_p}\sum_{i=1}^{C} c_i * \mathbf{y}_i^{(k)} \log(\mathbf{p}_i^{(k)}) \tag{4.1}$$

Where c_i is the weight given for the specific class and C is the number of output classes, in this case $C = 20$. $\mathbf{y}^{(k)}$ is the desired output vector, for the specific pixel, $\mathbf{y}_i^{(k)} \in \{0, 1\}$. N_p is the number of prediction vectors, in this case the same amount as pixels in the original image. $\mathbf{p}^{(k)}$ is the probability vector at pixel position k which is calculated using the Softmax function, defined as:

$$\mathbf{p}_i^{(k)} = \frac{e^{\mathbf{x}_i^{(k)}}}{\sum_{j=1}^{C} e^{\mathbf{x}_j^{(k)}}} \tag{4.2}$$

Where $\mathbf{x}^{(k)}$ is the prediction vector for the pixel position k and n and i indicates the index in the vectors. By definition: $\forall k \quad \sum_{i=1}^{C} \mathbf{p}_i^{(k)} = 1$.

4.4 Network architectures

The network architectures in this thesis are mainly based on SegNet. As mentioned in section 3.7 there is room for some kind of parameter reduction. This thesis investigates that by using a less deep architecture with wider filters along with the original SegNet. The two networks are called "Deep-Model" and "Wide-Model". There is also a third experiment called "Baseline" used as reference. Figure 4.2 shows a schematic representation of SegNets architecture, which is also used in "Deep-Model". Figure 4.3 shows the less deep architecture with wider filters called "Wide-Model".

4.5 Accuracy and IoU

In accordance with Cityscapes and Pascal, IoU was used to evaluate the final performance. During training the test-accuracy was plotted to analyze the convergence speed. The accuracy and loss was tested each 50 iterations using a subset of 50 images from the validation. An additional measure *Mean square error* was used to compare the filters before and after training.

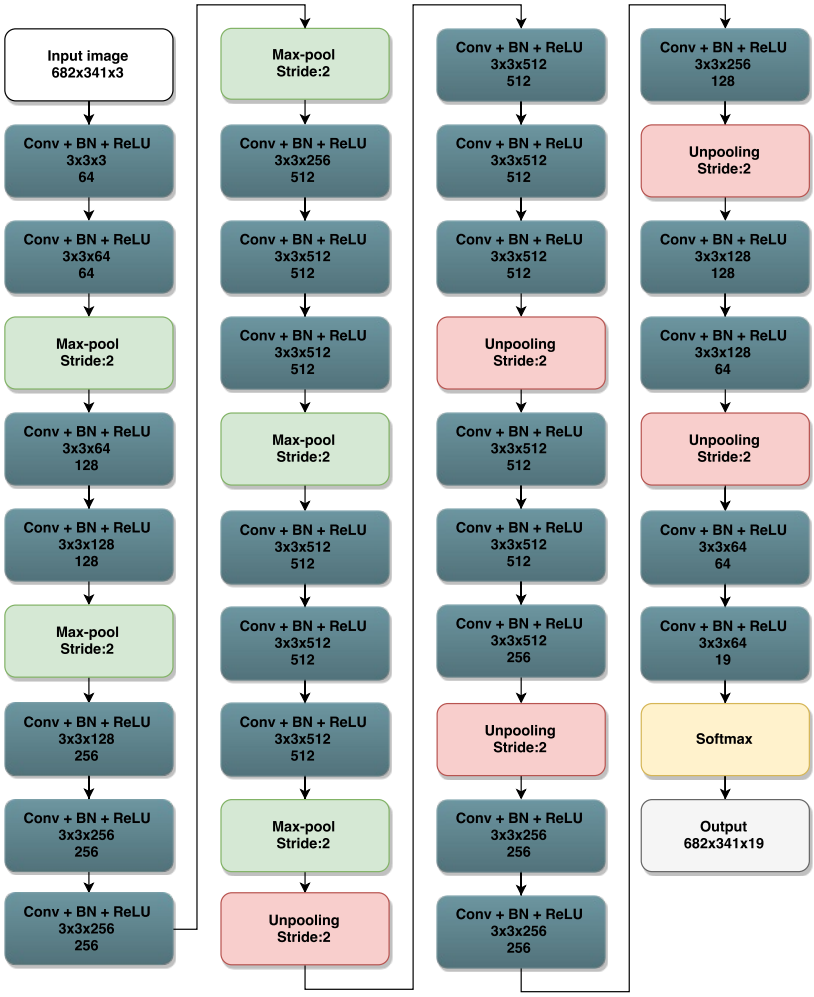


Figure 4.2: *Layer structure of SegNet and Deep-Model. Each blue blob contains three steps. Convolution, Batch normalization and ReLU activation. The indices of the Max-pooling are saved and put back in the unpooling step. The output contains a prediction for each pixel, which is passed to the loss function mentioned in equation 4.1.*

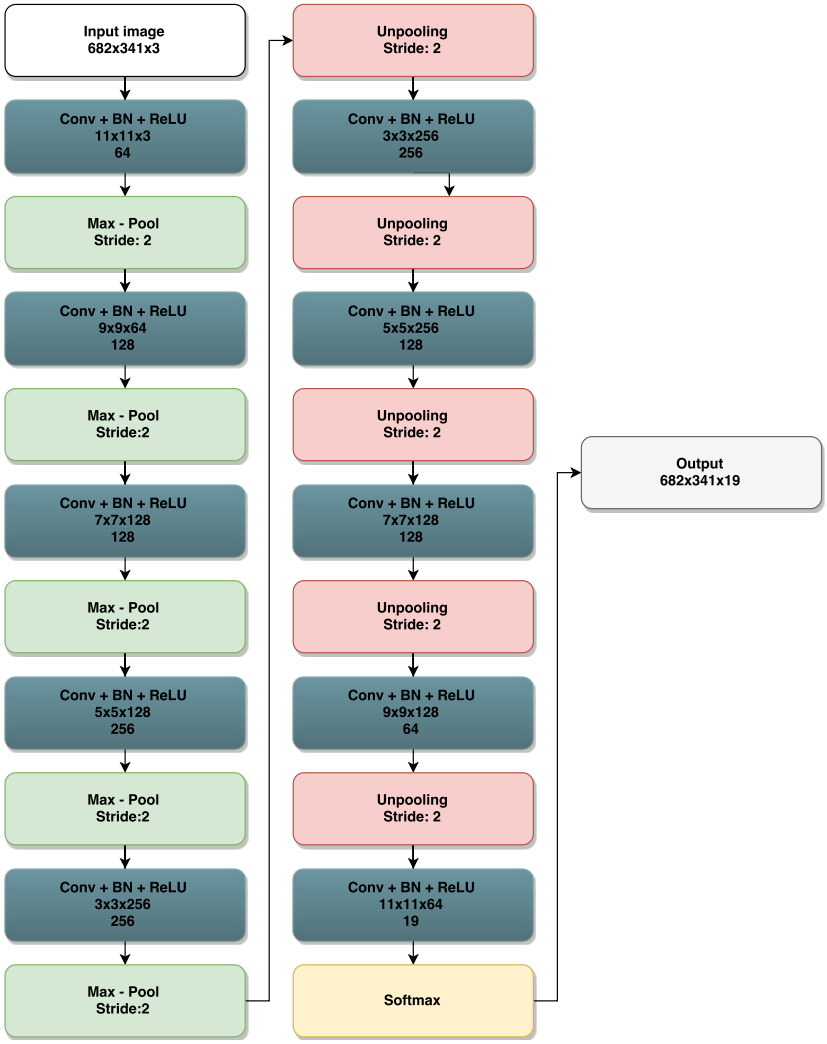


Figure 4.3: *Layer structure of Wide-Model. Each blue blob contains three steps. Convolution, Batch normalization and ReLU activation. The indices of the Max-pooling are saved and put back in the unpooling step. The output contains a prediction for each pixel, which is passed to the lossfunction mentioned in equation 4.1.*

5

Experiments

This chapter describes the details of each experiment. First of all the baseline method is described and then the details of each method. Table 5.1 declares the common parameters used in all experiments. The tables in each section describes the parameters specific for the experimental case. All experiments will be referred to as 1.x, 2.x and 3.x. Where 1.x denotes Baseline, 2.x Deep-model and 3.x the Wide-Model respectively.

Batch size	3
Momentum	0.9
Weight decay	0.0005
γ	1
β	0.001

Table 5.1: *The top three specifies training parameters are common parameters used in all experiments. β and γ are the parameters used to initialize the BN layer.*

The pretraining was always made on the decoding part, where only those weights were updated. The second part of the training was made on the whole network. Figure 5.1 clarifies this. The comparison between the different experiments will reveal information from which the conclusions are drawn. The following three comparisons are the most relevant, but other notable results are declared in chapter 7.

- 1.1 - 1.3 - 2.1 How the 3 different initializations affects the result. (Using 22975 images and Deep-Model)
- 1.2 - 1.4 - 2.2 how the 3 different initializations affects the result. (Using 2975 images and Deep-Model)

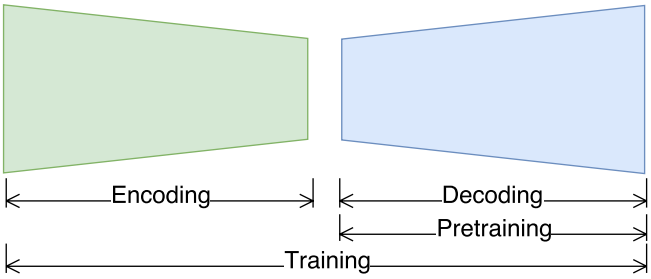


Figure 5.1: *Only the weights in the right part were updated during pretraining. During the main training phase all weights were updated.*

- 3.1 - 3.2 How the 2 different initializations affects the result (Using 2975 images and Wide-Model)

The notations in the following sections and tables are: **CAI** = coarse annotated images, **FAI** = fine annotated images, **LR** = learning rate.

5.1 Baseline

Baselines to compare the results to was required for this thesis. Four different baselines were produced, all using the SegNet architecture (same as Deep-model) but initialized and trained in different ways.

1.1 **Rdm init + FAI**

	# Samples	LR	Weight init	Iterations	Dataset
Pretraining	2975	0.001	MSRA	60000	FAI
Training	2975	0.001	From pretrain	60000	FAI

1.2 **Rdm init + CAI + FAI**

	# Samples	LR	Weight init	Iterations	Dataset
Pretraining	20000	0.01	MSRA	160000	CAI
Training	2975	0.001	From pretrain	60000	FAI

1.3 **VGG16 init + FAI**

	# Samples	LR	Weight init	Iterations	Dataset
Pretraining	2975	0.001	VGG16 + MSRA	60000	FAI
Training	2975	0.001	From pretrain	60000	FAI

1.4 **VGG16 + CAI + FAI**

	# Samples	LR	Weight init	Iterations	Dataset
Pretraining	20000	0.01	VGG16 + MSRA	160000	CAI
Training	2975	0.001	From pretrain	60000	FAI

5.2 Deep-Model

Two experiments were made on this network using sparse dictionaries. First using only the FAI images and secondly using both CAI and FAI for training.

2.1 **Dict. init + FAI**

	# Samples	LR	Weight init	Iterations	Dataset
Pretraining	2975	0.001	Dict + MSRA	60000	FAI
Training	2975	0.001	From pretrain	60000	FAI

2.2 **Dict. init + CAI + FAI**

	# Samples	LR	Weight init	Iterations	Dataset
Pretraining	20000	0.01	Dict + MSRA	160000	CAI
Training	2975	0.001	From pretrain	60000	FAI

5.3 Wide-Model

These results are compared internally and used in addition to the other experiments to see if the behavior is the same in a less deep model. Their performance can be compared to the Deep-model to analyze the effect of a parameter reduction. Final performance and convergence speed are of interest.

3.1 **Rdm init + FAI**

	# Samples	LR	Weight init	Iterations	Dataset
Pretraining	2975	0.001	MSRA	60000	FAI
Training	2975	0.001	From pretrain	60000	FAI

3.2 **Dict. init + FAI**

	# Samples	LR	Weight init	Iterations	Dataset
Pretraining	2975	0.001	Dict + MSRA	60000	FAI
Training	2975	0.001	From pretrain	60000	FAI

3.3 **Dict. init + FAI (high learningrate)**

	# Samples	LR	Weight init	Iterations	Dataset
Pretraining	2975	0.01	Dict + MSRA	60000	FAI
Training	2975	0.01	From pretrain	60000	FAI

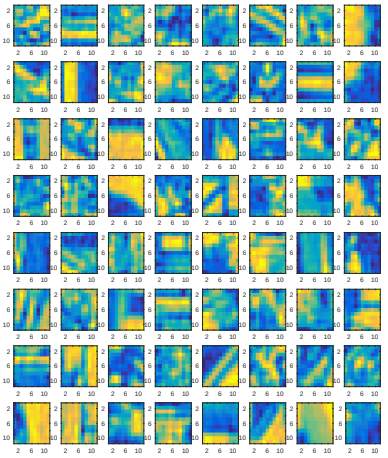
6

Results

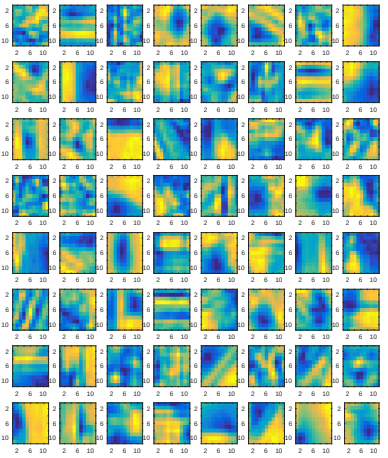
There are five sections in this chapter. The first is about the K-SVD filters the second is the class weights and the last three contains the results from the corresponding experiment.

6.1 K-SVD filters

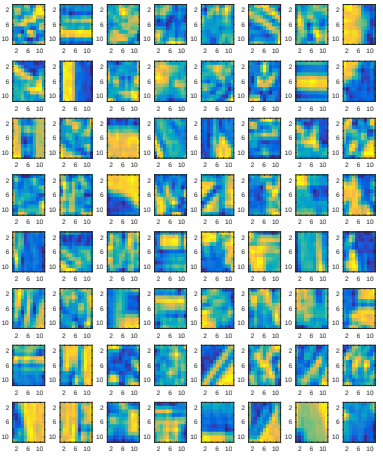
The sparse dictionaries was calculated using the K-SVD method. A comparison of the filters before and after training can be seen in figure 6.1. They were produced using the method presented in section 4.2 and inserted in the networks. It can be obtained that the trained filters is a smooth version of the initial filters. There are a few exceptions however where the difference is bigger. Some of the atoms has general structures such as gradients in different directions and some has more specific feature extracting structures. Another comparison between random - and dictionary initialized filters are shown in figure 6.2. The random initialized filters clearly start to form some specific feature extracting structures but less smooth than the dictionary filters.



(a) Before training.

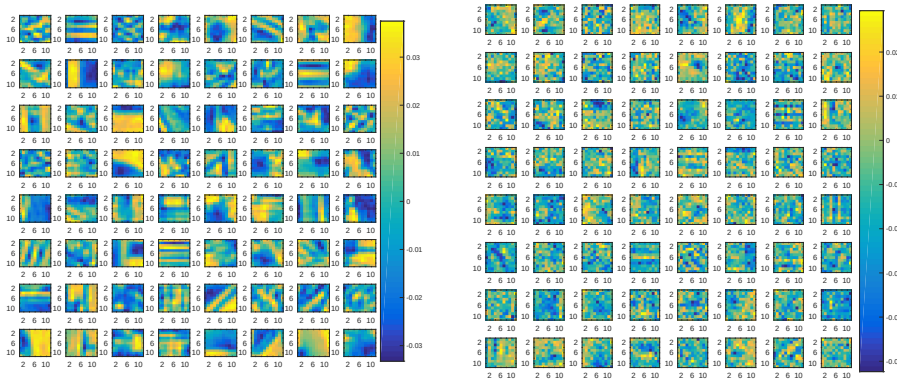


(b) After training.



(c) Difference.

Figure 6.1: *Comparison of K-SVD filters of size 11x11x3 before and after training. Only first color dimension is shown. It is clear that the filters extracts special features from the images. The difference is minor but still distinct. Each weight decay in proportion to its size and sign. This causes the difference map to be have the same appearance but with another scale.*



(a) Dictionary initialized. *(b)* Random initialized.

Figure 6.2: *The difference between dictionary initialized and random initialized filters after complete training.*

6.2 Class weights

The class weights are given in table 6.1. It is clear that some objects are much less common than others. Roads are the most frequent object and traffic lights the least. The class weights were used int the loss-function as described in equation 4.1.

Table 6.1: *Class weights used during training.*

Class	Weight $\hat{c_i}$	Class	Weight $\hat{c_i}$
Void	0	Terrain	1.0
Road	0.0557	Fence	1.0338
Building	0.0901	Wall	1.0349
Vegetation	0.1272	Person	1.3485
Car	0.2834	Pole	1.6829
Sidewalk	0.3235	Bicycle	2.7865
Train	0.4269	Traffic Sign	3.5577
Sky	0.4689	Motorcycle	3.6379
Bus	0.8156	Rider	5.3157
Truck	0.9398	Traffic light	5.5727

6.3 Baseline

The results corresponds to the experiments in section 5.1 and the first column referrers to the corresponding number in the same section. The tables show that

initializing with VGG-weights is better than using Random weight which in turn is better than using the sparse dictionaries. **MSE** denotes *mean square error*

Table 6.2: *Baseline results. IoU and accuracies, used to evaluate the results. The original SegNet architecture is used and initialised in several ways. Either with VGG16-weights, Random weights, CAI trained weights or VGG16+CAI trained weights.*

	IoU [%]	**Test acc.** [%]	**Train acc.** [%]	**MSE** [10^-2]
1.1 Rdm	32.2	69.7	77.4	0.089
1.2 Rdm+CAI	44.6	77.6	85.6	0.088
1.3 VGG	43.5	75.7	81.8	0.00074
1.4 VGG+CAI	56.2	79.7	85.6	0.00070

6.4 Deep-Model

The number in the first column referrers to the corresponding number in section 5.2 These results are mainly compared to 2.x. The first result is significantly lower.

Table 6.3: *Deep-Model results. The first is only trained with FAI and the second with both. Notice the great difference in IoU.*

	IoU [%]	**Test acc.** [%]	**Train acc.** [%]	**MSE** [10^-2]
2.1 Dict	19.3	56.3	66.9	0.0038
2.2 Dict+CAI	35.7	73.3	84.5	0.0038

6.5 Wide-Model

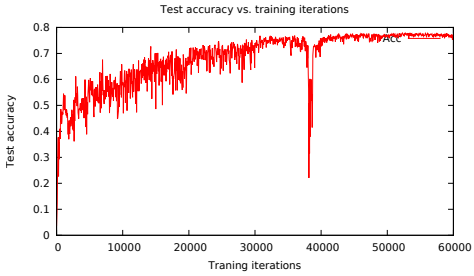
The number in the first column refers to the corresponding number in section 5.3. The results are to be compared to each other and to the Deep-Model results.

Table 6.4: *Wide-Model Results. Notice the difference in IoU using low lr and high lr. It seems like none of the networks has converged completely. Although The Rdm initialized seem to converge faster. There will be a plot for this soon enough.*

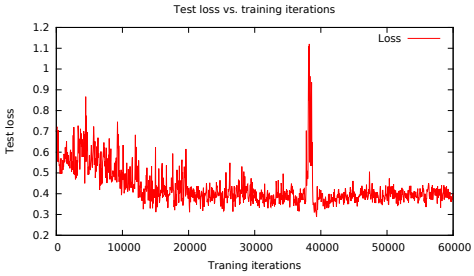
	IoU [%]	**Test acc.** [%]	**Train acc.** [%]	**MSE** [10^-2]
3.1 Rdm	28.9	69.0	77.2	0.09
3.3 Dict + low lr	25.2	64.8	72.1	0.0042
3.2 Dict + high lr	32.8	70.2	76.9	0.047

6.6 Convergence

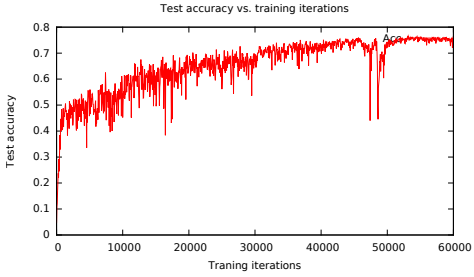
The plots show the accuracy and loss of a random and dictionary initialized network. There is no clear difference in convergence speed in neither of them. There is also a spike in both graphs.



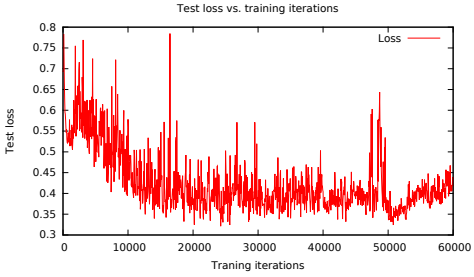
(a) Accuracy plot, random initialization.



(b) Loss plot, random initialization.



(c) Accuracy plot, dictionary initialization.



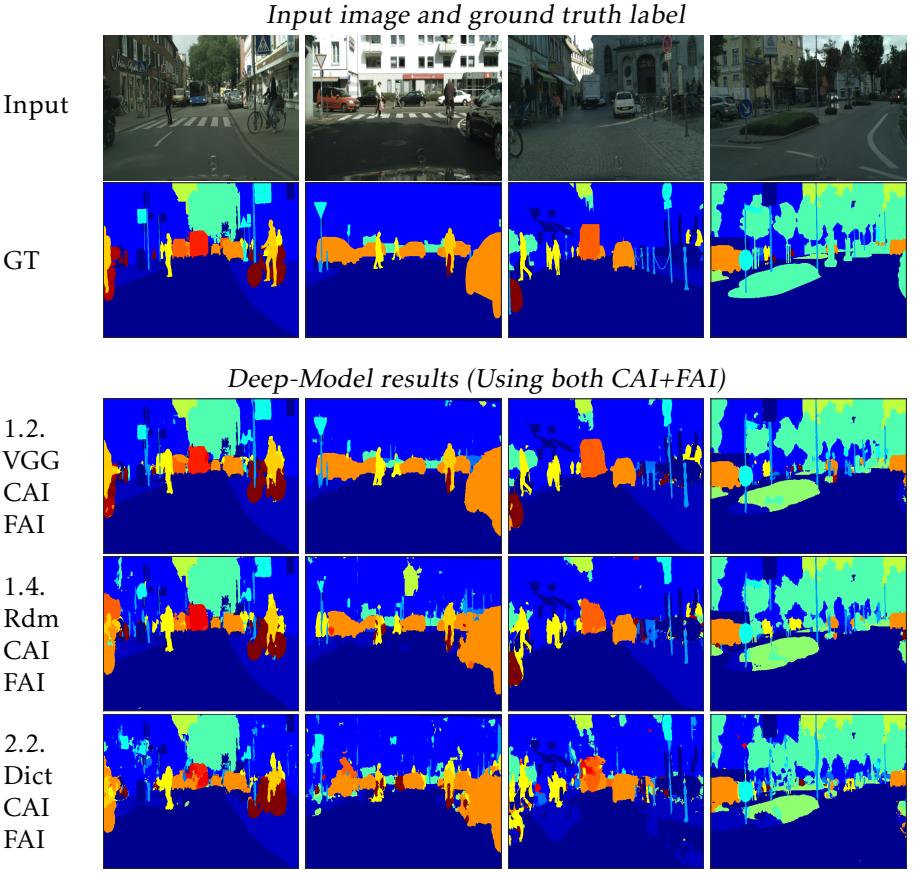
(d) Loss plot, dictionary initialization.

Figure 6.3: *Accuracy and loss plots of using random - vs dictionary initialization.*

6.7 Output images

Figure 6.4 shows the output from each architecture and training scheme. Note that they are put in such an order that they are easy to compare to each other.

The notations are: **CAI** = coarse annotated images, **FAI** = fine annotated images, **VGG** = VGG initialization, **Rdm** = Random weight initialization, **Dict** = Sparse dictionary weight initialization.


Input image and ground truth label
Input
GT
Deep-Model results (Using both CAI+FAI)
1.2.
VGG
CAI
FAI
1.4.
Rdm
CAI
FAI
2.2.
Dict
CAI
FAI

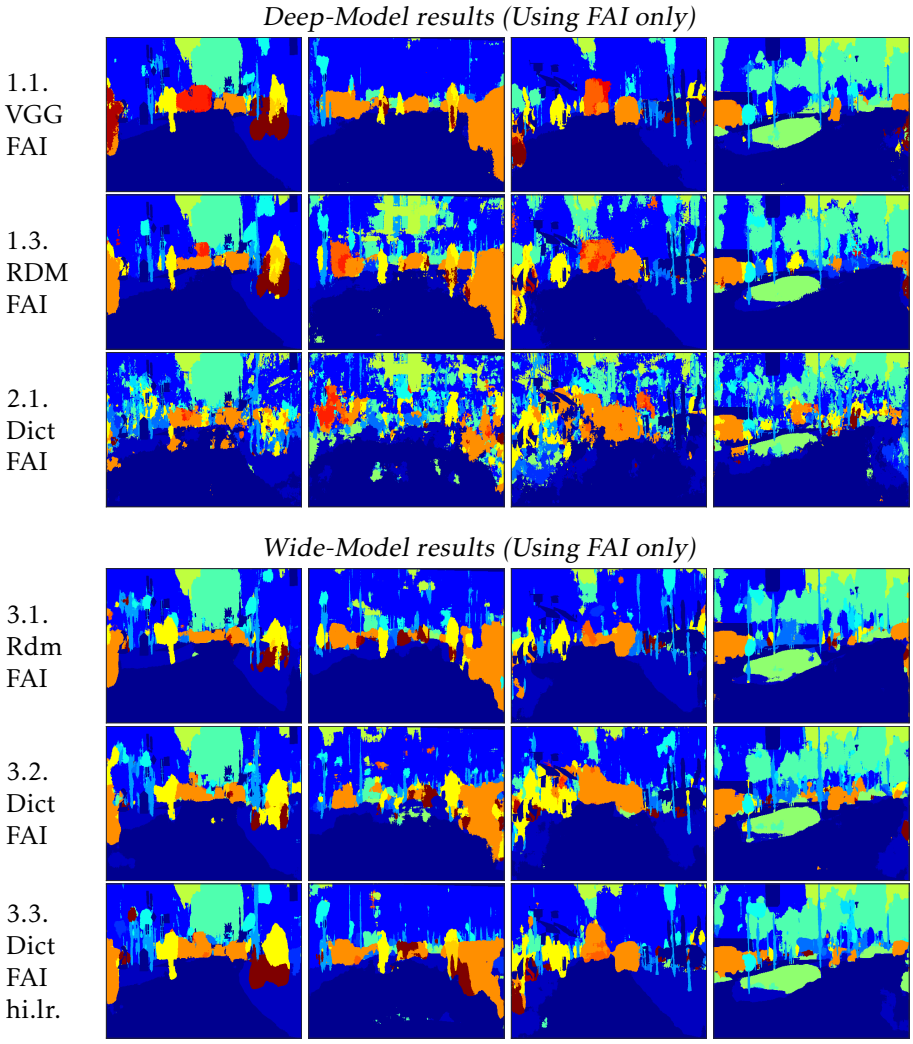


Figure 6.4: *The two images in the top are input and ground truth. The following three shows baseline and result of the Deep-model using all available data. The next three is the output using the same setup as before but less data. The last three shows basline and results using the Wide-model*

7

Analysis of results

There are quite a few conclusions that can be drawn from this thesis. And we will discuss how the usage of sparse dictionaries affects the performance of a convolutional neural network.

Performance

To answer this question I first analyze the final IoU of the different networks. Table 7.1- 7.3 summarizes the results. It is obvious that the network that uses VGG

Table 7.1: *Final output using SegNet architecture and a big dataset of 22975 images with different initializations.*

Name	**Init.**	**IoU [%]**
1.2	Rdm	44.6
1.4	VGG	56.2
2.2	Dict	35.7

initialization gives the best results. This is not surprising since those weights are pretrained on the huge ImageNet dataset. Random initialization performs significantly better than using the sparse dictionary. Table 7.1 and 7.2 both implies that conclusion. One can notice a major difference between 2.1 and 2.2. As [14] argues and apply on Xavier-filters, a bad initialization may cause the training to converge slower or to get stuck in local minimums. This phenomenon is probably what appeared here, where 2.1 is stuck in a local minimum which results in an IoU-value as low as 19.5. The results from the Wide-model in table 7.3 are similar, implying that the usage of sparse dictionaries reduces the final IoU. The plots presented in figure 6.3 shows no enhancement of the convergence. The spikes that occur in both plots may be caused by an abnormal mini-batch not represent-

Table 7.2: Final output using SegNet architecture and a small dataset of 2975 images with different initializations.

Name	Init.	IoU [%]
1.1	Rdm	32.2
1.3	VGG	43.5
2.1	Dict	19.3

Table 7.3: Final output using Wide-Model and a small dataset of 2975 images.

Name	Init.	IoU [%]
3.1	Rdm	28.8
3.2	Dict.	25.2
3.3	Dict. (high lr.)	32.8

ing the global gradient very well. This causes the gradient to point in an other direction than the global one which in turn decreases the generalization and test accuracy.

Output images

The numbers in the tables indicates that some of the networks perform better than others, but how big is the difference when looking at the actual output? The images in figure 6.4 show the differences. One observation is that the network initialized with either random or VGG are less cluttered than dictionary initialized.

The differences between the Wide-model and Deep-model are also interesting to view. The Wide-model seem to understand the big picture but fails on the details. In some cases that is crucial but in other cases not as much. If the purpose is to stay on the road and avoid any other object this could be enough.

Filters

The filters in figure 6.1 a and 6.1 b are similar. This indicates that the dominating parameter is the weight decay parameter denoted as λ in equation 2.5. The influence of this parameter could be discussed but there seem to be room for parameter tuning. In this thesis the parameters are copied from SegNet and the effect of the weight decay parameter are not analyzed further.

The filters in figure 6.2 a and b seem to form specific feature extracting filters of the same type. The random initialized filters are not as smooth as the others though. Why this happens is not an easy question to answer. Let's assume that the difference between the random filters and the sparse dictionary are their complexity. The sparse dictionary is less complex since the filters are smoother and has less structure. The random filters are more complex and could therefor find more complex structures in the image. However the sparse dictionary still constitutes a good feature extracting library. When they are inserted in the neural

network they put the network in a state close to a local minimum. When the training starts the network falls into the local minimum hole. On the other hand when the network is random initialized the probability of falling into the local minimum decreases and the probability that the filters form complex structures and steps towards a global optima increases.

Summary

The goal of the thesis was examine the effect of using sparse dictionaries in a CNN. It was found that the filters definitely affects the performance of the network. The results show that sparse dictionaries provide below-expected results. Never the less, the experiments highlight the importance of filter initialization for training deep networks and provides a motivation for deep future analysis into the issue of filter initialization.

8 Discussion

8.1 Reflections of the method

A major question for this thesis have been how to train the different networks to be able to evaluate the results in a robust way. The intuitive thought was to train each net until it converged. This method was rejected since this adds another uncertainty parameter (training time). Instead the training was kept constant for all networks. A major drawback using this approach is the uncertainty of the performance if the training would have been fully completed. To answer the main question of this thesis, that is however not necessary to know. Instead the performance at a specific point was evaluated and the converging curves analysed. All results were clearly pointing in the same direction why this seem to be enough evidence to draw the conclusions.

Another thought was to optimize the filters even more before the final training. This could be done as a step in the method where the filters are computed, described in section 4.2. For each added convolutional layer, a training session could be performed to optimize the added dictionary. This approach would ensure that each added convolutional layer got enough training and make those layers less sensitive to the vanishing/exploding gradient problem at the final training.

8.2 Extensions and future work

This thesis clearly shows that the weight initialization is an important aspect to consider before training. It would be interesting to further investigate this field. There might be ways to train an additional NN to optimize initial filters to put in the first network. Or there could be other mathematical solutions than sparse dictionaries that could be used for initialization.

Bibliography

[1] Michal Aharon, Michael Elad, and Alfred Bruckstein. An algorithm for designing overcomplete dictionaries for sparse representation. *IEEE Transactions on signal processing*, 54(11):4311–4322, 2006. Cited on pages 3, 16, 17, and 19.

[2] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *arXiv preprint arXiv:1511.00561*, 2015. Cited on pages 2, 9, 10, and 11.

[3] Hilton Bristow, Anders Eriksson, and Simon Lucey. Fast convolutional sparse coding. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 391–398, 2013. Cited on pages 3 and 17.

[4] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Semantic image segmentation with deep convolutional nets and fully connected crfs. *arXiv preprint arXiv:1412.7062*, 2014. Cited on page 9.

[5] Dan Ciregan, Ueli Meier, and Jürgen Schmidhuber. Multi-column deep neural networks for image classification. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 3642–3649. IEEE, 2012. Cited on page 2.

[6] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. *arXiv preprint arXiv:1604.01685*, 2016. Cited on pages 3, 9, 13, and 14.

[7] David Eigen and Rob Fergus. Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2650–2658, 2015. Cited on pages 13 and 23.

[8] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes challenge: A retrospective. *International Journal of Computer Vision*, 111(1):98–136, January 2015. Cited on pages 3, 9, 13, and 14.

[9] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012. Cited on pages 3, 9, and 13.

[10] Golnaz Ghiasi and Charless C Fowlkes. Laplacian pyramid reconstruction and refinement for semantic segmentation. *arXiv preprint arXiv:1605.02264*, 2016. Cited on page 9.

[11] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Aistats*, volume 9, pages 249–256, 2010. Cited on pages 1 and 15.

[12] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015. Cited on pages 11 and 19.

[13] Simon Haykin and Neural Network. A comprehensive foundation. *Neural Networks*, 2(2004):41, 2004. Cited on page 7.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1026–1034, 2015. Cited on pages 1, 3, 15, 23, and 39.

[15] Felix Heide, Wolfgang Heidrich, and Gordon Wetzstein. Fast and flexible convolutional sparse coding. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5135–5143. IEEE, 2015. Cited on pages 3 and 17.

[16] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012. Cited on page 9.

[17] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012. Cited on page 11.

[18] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016. Cited on pages 11 and 19.

[19] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015. Cited on pages 11, 12, 13, and 20.

[20] Vladlen Koltun. Efficient inference in fully connected crfs with gaussian edge potentials. *Adv. Neural Inf. Process. Syst*, 2011. Cited on page 9.

[21] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. Cited on page 11.

[22] Yann LeCun, Corinna Cortes, and Christopher JC Burges. The mnist database of handwritten digits, 1998. Cited on page 9.

[23] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3431–3440, 2015. Cited on pages 2 and 9.

[24] Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. Learning deconvolution network for semantic segmentation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1520–1528, 2015. Cited on pages 2, 9, and 10.

[25] Adam Paszke, Abhishek Chaurasia, Sangpil Kim, and Eugenio Culurciello. Enet: A deep neural network architecture for real-time semantic segmentation. *arXiv preprint arXiv:1606.02147*, 2016. Cited on pages 2 and 9.

[26] Ron Rubinstein, Michael Zibulevsky, and Michael Elad. Efficient implementation of the k-svd algorithm using batch orthogonal matching pursuit. *CS Technion*, 40(8):1–15, 2008. Cited on pages 16 and 17.

[27] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3): 211–252, 2015. doi: 10.1007/s11263-015-0816-y. Cited on pages 3, 9, and 13.

[28] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3): 211–252, 2015. doi: 10.1007/s11263-015-0816-y. Cited on page 9.

[29] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. Cited on pages 2, 9, and 15.

[30] Zifeng Wu, Chunhua Shen, and Anton van den Hengel. Wider or deeper: Revisiting the resnet model for visual recognition. *arXiv preprint arXiv:1611.10080*, 2016. Cited on page 19.

[31] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. *arXiv preprint arXiv:1612.01105*, 2016. Cited on page 19.