Adapting pre-trained convolutional neural network models for traffic scene labeling

Master Thesis

submitted to

Institute of Control Theory and Systems Engineering

Faculty of Electrical Engineering and Information Technology

Technische Universität Dortmund

by

Carlos Miguel Trevino Campa

Dortmund, Germany

Date of Submission: October 12, 2015

Responsible Professor:
Univ.-Prof. Dr.-Ing. Prof. h.c. Dr. h.c. Torsten Bertram

Academic Supervisors:
apl. Prof. Dr. rer. nat. F. Hoffmann
Malte Oeljeklaus, M.Sc.

"*Das ist die Widmung (optional)*"

Acknowledgement

I would like to take this opportunity to express my gratitude to M.Sc. Malte Oeljeklaus and apl. Prof. Dr. rer. nat. Frank Hoffmann for their guidance and help provided during the work.

I also express my gratitude to my family and close friends for their constant support during the program.

Abstract

Modern driver assistance systems aim to automate the task of driving a vehi-cle in increasingly complex situations. In order to handle these situations, a profound understanding of the scene surrounding the vehicle is essential. The rich level of detail found in camera images makes them a natural choice to approach this problem. In computer vision, scene labeling is known as la-beling each pixel in an image with the category of the object or region it belongs to. Existing methods rely on learning databased models from recorded and hand-labeled datasets using machine learning techniques. One method are so called convolutional neural network (CNN) models, which are variants of regular neural networks consisting of a number of convolutional and subsam-pling layers optionally followed by fully connected layers. CNN models de-pict the state of the art in several computer vision problems. However, they typically require large datasets and long computation times during the training phase. One way to overcome this drawback is to use pre-trained, readily para-metrized models as an initialization and adapt (fine-tune) them to solve a specific problem. For this, the thesis task is to analyze, how existing pre-trained CNN models can be adapted towards the problem of traffic scene la-beling. Therefor the method of fully convolutional networks is to be exam-ined, which does not require complex pre- or post-processing steps and al-lows for adapting CNN models trained for whole-image classification towards scene labeling tasks. The task includes preparing an appropriate dataset for traffic scenes using typical category labels including road, car and pedes-trian.

Contents

Nomenclature

$\mathbf{x}(t)$

t

$u(t)$

x, y, z

Abbreviations and acronyms

Abrev.

KF

Greek symbols

α

1 Introduction

1.1. Motivation

Computer vision systems and their capabilities have advanced in the recent years. They make use of the more powerful computer processing capabilities and new learning methods/algorithms to improve the overall performance. Autonomous driving is one of several application that benefits from these advances.

In order to obtain a deeper understanding of the surrounding environment, autonomous driving applications can take advantage of camera images. Complex situations can take place in traffic environments, therefore an efficient analysis of the provided images is required. In order to extract the important information contained in the images, different technologies are used (e.g object detection, semantic segmentation, etc). *Semantic segmentation* (pixel classification) simplifies an image by associating a predefined class labels to each pixel, while *Object detection* localizes objects which belong to determined classes.

The purpose of this work is to yield a Semantic segmentation of traffic scenes with an outstanding performance and able to be implemented in real time applications. The selected approach exploits the recent developments in machine learning and more specifically makes use of *Convolutional Neural Networks* (CNNs) using **Caffe** (Jia et al. (2014)) as framework.

Convolutional Neural Networks (CNNs) have become the mainstream of high-level vision research. They are powerful visual models that yield hierarchies of features, and were successfully used for the first time to solve document recognition (LeCun et al. (1998)). CNNs currently show state of the art performance in high level vision tasks (e.g. image classification, object recognition,object detection, etc.). More information about the success of CNNs is presented in Section 2.1.

The spatial relation yield in the image semantic segmentation is also benefited from a temporal relation within the continuous stream of images. The previous frames provide information which allows to enhance the current image segmentation. The main idea of this addition is to get a more accurate segmentation while producing a smoother stream of segmented images. Causal segmentation with state of the art results is covered in Section 2.2.

1.2. Objective

1.3. Experimental Setup with Caffe

Caffe (Jia et al. (2014)) is a deep learning framework developed by the Berkeley Vision and Learning Center (BVLC). It is a C++ library with Python bindings for state of the art deep learning algorithms. The code is written in a clean and efficient way with CUDA used for GPU computation. All of these features allow to speed up the research progress and facilitate to share results. There are similar frameworks (e.g. Theano, Torch, etc.) that provide similar functionalities.

Caffe allows to create, train, test, fine-tune and deploy deep learning architectures in a simple way. It is mostly focused on Convolutional Neural Networks for large scale visual recognition. It has been mostly used for object classification, semantic features learning and object detection. However it has also been used for different purposes (e.g. speech recognition, robotics, astronomy, etc.).

The great advantages of this framework are the availability of reference models and a clean separation of the network definition from the implementation among others. It is under constant development and maintenance by the efforts of BVLC students and open-source contributions.

The work presented in this thesis profits from the available pre-trained models with state of the art results (e.g. "AlexNet", "GoogLeNet", etc.)and, n order to yield pixelwise classification, the changes presented in Long, Shelhamer, and Darrell (2014) (i.e. Crop Layer) are applied to the current Caffe distribution. Caffe implementation details are described in Chapter 4.

1.4. CamVid Dataset

CamVid (Cambridge-driving Labeled Video Dataset) is a freely available dataset consisting of videos with object class semantic labels (Brostow, Fauqueur, and Cipolla" ("2008")). All the data contained within was captured from the perspective of a driving automobile. Therefore this dataset is suitable for traffic scene understanding.

The dataset includes four HD video sequences with a 960x720 resoltion at 30 fps. This dataset provides 701 images with their respective ground truth labels (manually labeled) that associate each pixel with one of 32 semantic labels. It consists of a mix of environments with both urban an residential scenarios. It was also captured at daylight and at dusk. This capture settings make the CamVid dataset ideal to develop robust and diverse learning methods.

The training and testing data was taken from previous works Brostow et al. (2008) to simplify comparison with different segmentation techniques. In order to reduce computational processing efforts and due a big number of labels with relative scarce presence only 11 semantic classes will be considered in this thesis. The high definition and the length of the captured videos allow to cope with small objects (i.e. pedestrians, traffic signs, road marking, etc.) and to get multiple testing images under the same illumination and camera conditions. CamVid dataset processing and handling is covered in Chapter 4

1.5. Overview

In Chapter 2 covers the related work. Several approaches for semantic segmentation and temporary consistent segmentation with state of the art results are presented with their respective description. For semantic segmentation, just work related to CNNs is considered. Explanations about complementary parts of this work is beyond the scope of this thesis. For temporary consistent segmentation, only methods that rely on current and past data. This is done to avoid frame-by-frame and omnipresent approaches that cannot be implemented on real time (online) applications.

Chapter 3 contains the theoretical framework. The first section explains what Convolutional Neural Networks are, their characteristics (e.g. architecture, deep learning, layers, etc.) and describes how they are used for high-level vision tasks. The second section covers the temporary consistent segmentation. It mainly focuses on segmentation algorithms and image matching techniques which are explained in order to have a clear understanding of causal segmentation. Temporal smoothing and segmentation accuracy are also covered.

Chapter 4 presents the experiments that were driven. The experimental framework is divided into four sections: CamVid dataset, CNNs, , temporary consistent segmentation and their respective results. The first section gives details of how the image processing, data augmentation, and training/testing data partition take place. The CNN section describes the adaptation process of pre-trained networks, different architectures, initialization methods, and learning techniques used. The third section covers the integration between independent segmentations and temporary consistent segmentations (i.e. how causal segmentation is yield). The last part contains the results of the driven experiments.

Chapter 5 finishes by covering various conclusions as well as the outlook of pre-trained convolutional neural networks for traffic scene labeling. Finally it presents the possibilities for future work.

2 Related Work

The work of this thesis is divided into two main tasks:

- **Pixelwise Semantic Segmentation**
- **Temporary Consistent Segmentation**

The following sections contain related work with state of the art results.

2.1. Semantic Segmentation

Semantic segmentation task is approached using a deep learning method, the so-called *Convolutional Neural Networks*. This decision is based on the recent success of CNN models for visual tasks (e.g. image classification (Krizhevsky, Sutskever, and Hinton (2012) Szegedy et al. (2014), Simonyan and Zisserman (2014)), object detection (Sermanet et al. (2013),Girshick et al. (2014)), local correspondence (Fischer, Dosovitskiy, and Brox (2014), Long, Zhang, and Darrell (2014))) and more recently in pixelwise labeling task (Clement Farabet and LeCun (2013), Lin, Shen, Reid, et al. (2015), Long, Shelhamer, and Darrell (2014), Chen et al. (2015), Hariharan et al. (2014)).

Current state of the art segmentation methods are observable in the Pascal VOC 2012 website. Most of the presented methods possess a CNN stage or are completely based on Convolutional Neural Networks. Lin, Shen, Reid, et al. (2015), Chen et al. (2015) have reached outstanding performance and when compared with similar approaches, both show a combination of CNNs and *Conditional Random Fields* (CRFs).

All these methods are similar to Clement Farabet and LeCun (2013), which successfully implemented the strengths of CNNs and CRFs together. The CNNs perform especially well for feature representations, while the CRFs capture contextual relation modeling. In order to obtain a more robust feature representation, a multi-scale pyramid was added at the beginning. The multi-scale pyramid exploits different feature representations that are only seen at a certain scale. Figure 2.1 presents a diagram of the scene parsing system proposed by Clement Farabet and LeCun (2013). It consists of two parallel branches where the first one is composed by a Laplacian pyramid and a CNN. The latter branch is a superpixel segmentation or a segmentation tree followed by a CRF model. Both branches are merged to produce a final pixelwise classification.

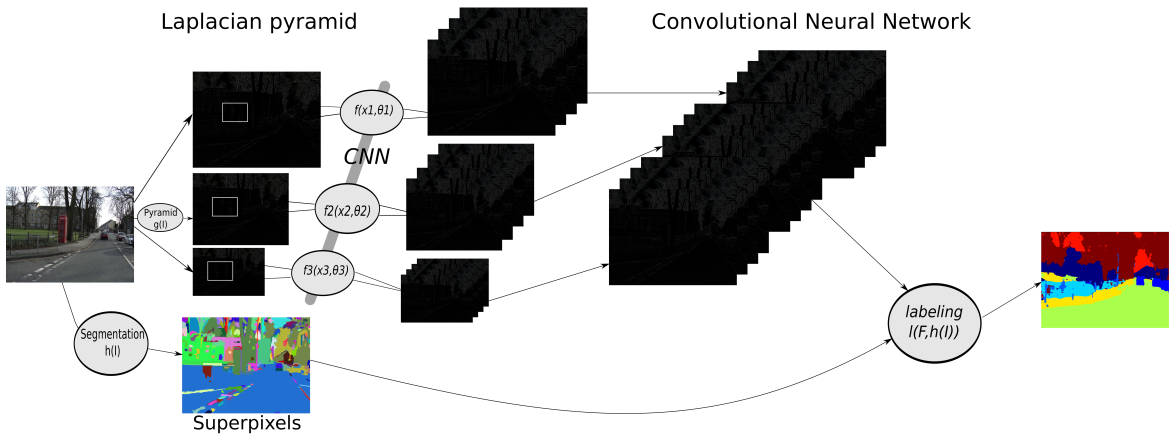

Figure 2.1.: Scene parsing System from Clement Farabet and LeCun (2013)

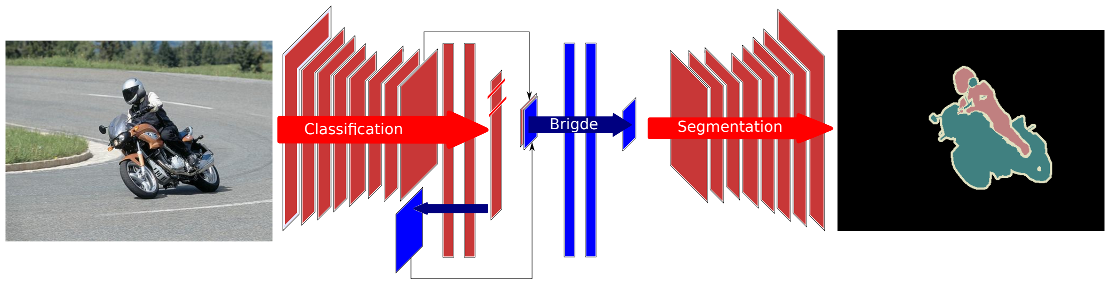

Figure 2.2.: Architecture of the networks proposed in Hong, Noh, and Han (2015). Classification takes places through several convolution layers. Semantic segmentation uses deconvolutional layers to rebuild the original image.

Hong, Noh, and Han (2015) and Noh, Hong, and Han (2015) approaches differ significantly from the typical CNN models. Both propose a two CNN model, where the classification and segmentation tasks are decoupled. Each CNN deals with an independent task. This decoupled architecture offers the possibility to train CNN separately. These approaches mitigate the heavy requirement of a large number of segmentation ground truth data by using weakly-supervised learning methods. Object labels associated with an input image are identified by a classification network, and object figure ground segmentation by the segmentation network. There is a third element in this algorithm: the bridging layers. Their function is to tight the otherwise loose connections between classification and segmentation. This algorithm solves two drawbacks of traditional segmentation methods: smoothed or lost detailed structures due a coarse label map (present in end-to-end systems using only CNNs) and a fixed-sized receptive field (no multi-scale representation), which leads to fragmented or mislabeled segmentations. Figure 2.2 gives an overview of this architecture.

Hariharan et al. (2014) introduce the *"Hypercolumn"* taken from neuroscience to also produce state of the art results. In order to overcome a problem with information from the last layers being too coarse, information from early layers must be also considered. The hypercolumn allows to have information from several layers inside the CNN. It benefits from the trade-off between different layers by combining them. The last layers capture semantics in a more efficient way (category-level information), while the early layers help to obtain a more precise location (e.g. pose, illumination, articulation,

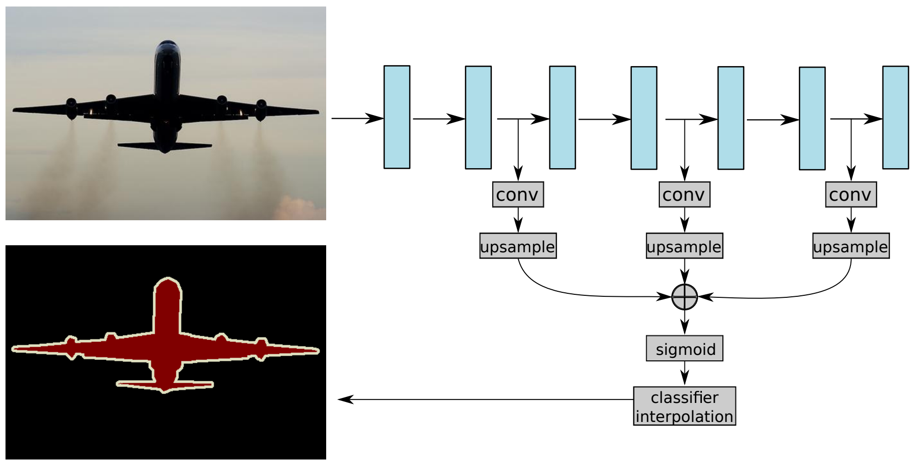

Figure 2.3.: Original classification CNN is shown in blue, and added layers for the hypercolumn classifier are in gray (Hariharan et al. (2014)).

location, etc.). The *Hypercolumns* work as pixel descriptors, and were successfully used for diverse visual tasks (e.g. detection, segmentation, keypoint localization and part labeling). Needless to say, it also uses multiple levels of abstraction and scale are necessary. This can be comparable to Clement Farabet and LeCun (2013) mutli-scale pyramid.However the invariance in is similar, since the features come from the same level of the CNN. Figure 2.3 shows the hypercolumn idea implemented in a CNN.

Notably, Long, Shelhamer, and Darrell (2014) present an end-to-end system for semantic segmentation using only *fully convolutional* networks. The models presented here are characterized for an architecture which combines deep, coarse, semantic informantion and shallow, fine, appearance information in a multi-layer framework. The fully convolutional models are obtained from pre-trained CNN that perform image classification (Krizhevsky, Sutskever, and Hinton (2012), Szegedy et al. (2014) and Simonyan and Zisserman (2014)). In order to make them suitable for pixelwise classification, the inner product layers are replaced by convolutional layers, while a transfer of parameters occurs based on Donahue et al. (2013). A fine-tuning process is driven to adapt the fully convolutional networks for new semantic segmentation tasks. In order to produce finer segmentations, different layers are upsamled and fused. The main advantages of this approaches lay in the capacity of reusing learned values, reducing the need of a big training dataset (which is costly effort) and the use of Caffe. Figure 2.4 explains how semantic segmentation is possible with a fully convolutional net.

The approach of this thesis is based on Long, Shelhamer, and Darrell (2014). Fully convolutional networks are possible to train using Caffe (Jia et al. (2014)) as framework. Caffe allows to use pre-trained models, which are publicly available and well documented. It is also handy to choose this framework due the big communities of users and developers. In order to make fine-tuning using a small dataset (Brostow, Fauqueur, and Cipolla" ("2008")) feasible, this thesis also makes use of transfer of parameters exposed in Donahue et al. (2013).

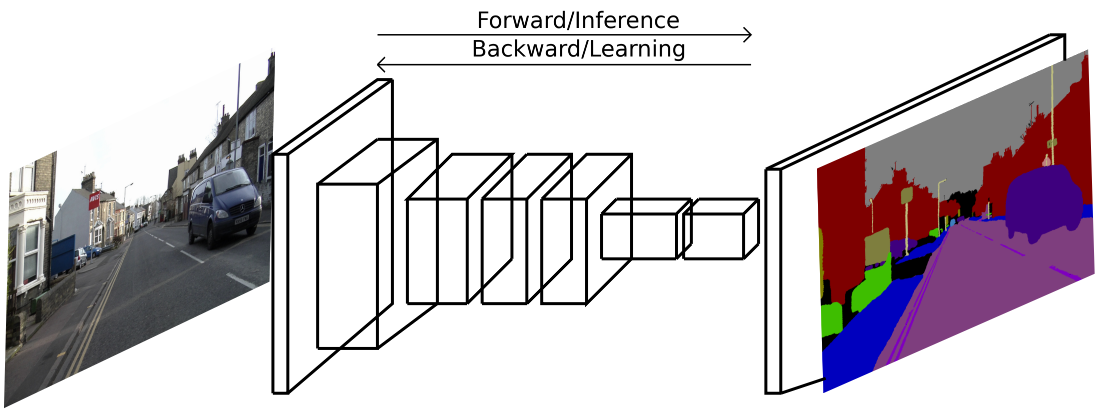

Figure 2.4.: Fully convolutional networks make dense predictions for pixel classification.The final layer makes use of a bilinear interpolation to produce the semantic segmentation

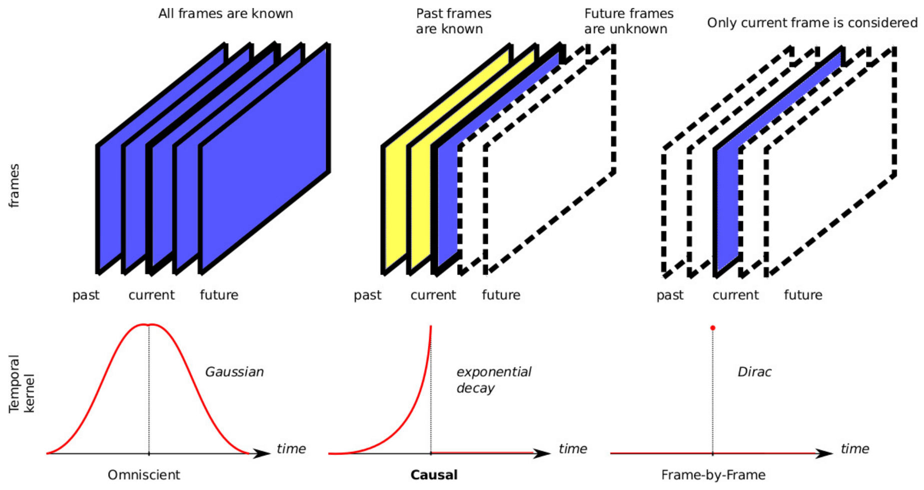

Figure 2.5.: There are three types of video segmentation. In order to improve the segmentation performance, this thesis focuses on causal segmentation.

2.2. Temporary Semantic Segmentation

This section presents the approaches with state of the art results for temporary semantic segmentation. Since it is desired to yield a causal segmentation for online applications, omniscient methods are not covered due their inherent need of future frames. The purpose of this implementation is to enhance the traffic scene labeling using a temporary relation (besides the spatial relation from the CNN). The final result is expected to have avoid flickering segmentations and to increase the accuracy based on previous knowledge. Figure 2.5 shows the desired kind of approach.

Ess et al. (2009) is a work whose purposes are very similar to the ones from this thesis. A two-stages approach is introduced where the first stage generates a superpixel segmentation based on Felzenszwalb and Huttenlocher (2004), which is later used in a second stage to construct a feature set for a classifier. The proposed image-based system is able to perform road recognition as well as a set of objects (e.g. cars, pedestrians, etc.). In order to allow online applications, the temporal smoothing is obtained using the forward pass output of a *Hidden Markov Model* (HMM) Rabiner (1989).

Paris (2008) profit from the spatial-temporal dimensions mix from video datasets.

This approach relies on a bilateral filtering and a mean-shift segmentation from Paris and Durand (2007). This is an online approach, therefore it only uses current and past data. The previously acquired data helps to ensure temporal coherence which otherwise cannot be accomplished by using frame-by-frame approaches.

Grundmann et al. (2010) use a hierarchical graph-based algorithm in order to yield a temporary consistent segmentation. This algorithm addresses three problems: Temporal coherence, automatic tracking and scalability. Like similar approaches, it is initialized with an oversegmentation using the graph-based image segmentation from Felzenszwalb and Huttenlocher (2004). Afterwards, the algorithm makes use of Optical Flow (Horn and Schunck (1981)) to modify the graph structure with respect to time. A hierarchical segmentation yields a region grpah from a previous segmentation level. This approach exhibits long-term temporal coherence and overcomes memory and runtime

Miksik et al. (2013) perform a scene analysis generating predictions of semantic categories over time. These predictions are characterized for being spatially and temporally consistent. It is an online algorithm which uses a temporal filter based on exponential smoothing over past predictions. This method can be applied in combination with any frame-by-frame analysis technique, as long as it produces a pixelwise labeling. This temporally smoothing technique works using Optical Flow (Horn and Schunck (1981)) to estimate correspondances with the previous frame.

Couprie et al. (2014) propose an approach whose objectives are similar to the ones of this thesis. It yields real-time segmentation of indoor scenes. This method benefits from the semantic segmentation of Clement Farabet and LeCun (2013). In order to improve the performance of the segmentation, the temporal consistency of the video sequences is exploited. This causal segmentation is taken from Couprie et al. (2013) and uses the superpixel segmentation method from Felzenszwalb and Huttenlocher (2004). A graph-matching technique is developed in order to match previous segmentations and to enrich the current frame segmentation. The main advantages of this approach are the computational efficiency and the possibility of future implementations working together with another frame-by-frame technique.

The method developed in this thesis, makes use of the causal segmentation proposed in Couprie et al. (2013), due its capabilities for real-time implementations and the availability of the programming code.

3
Theoretical Framework

This chapter presents a theoretical definition of all the main concepts used in the implementation of a traffic scene understanding system.

The section 3.1 covers the theoretical principles of Convolutional Neural Network and how they yield a semantic labeling process. Applications not related to computer vision tasks are beyond the scope of this thesis and omitted.

Section 3.5 explains several segmentation methods used in the temporary consistent segmentation. Efficient graph-based segmentation from Felzenszwalb and Huttenlocher (2004) and Watershed cuts as described in Cousty et al. (2009). These segmentation methods are characterized for requiring a small computationally effort and being fast enough to allow their use in real-time applications.

To end this chapter, section 3.7 describes how causal segmentation is yield (e.g. previous frames matching, Optical Flow and combined used with frame-by-frame approach).

3.1. Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are Neural Networks which perform a **convolution instead of a matrix multiplication in at least one of their layers** (Bengio, Goodfellow, and Courville (2015)). In order to process data inside a CNN, this must have a grid-like topology. CNNs are loosely based on neuroscientific principles and inspired on the mammalian vision system.

CNNs were firstly introduced in Fukushima (1980), but it was until Le Cun et al. (1990) (introduction of backpropagation) and LeCun et al. (1998) where they were successfully implemented to solve visual tasks. After the first CNN implementation with GPU in Ciresan et al. (2011), this deep learning method started to achieve a state of the art performance and consequently better results than traditional approaches.

The many arguments in CNNs are: the inputs, kernels and the outputs, which are also known as *Feature Map*. The inputs are multidimensional array of data, while the kernels are a multidimensional array of parameter that can be learned.

Due the inherent nature of a CNN model, the memory requirements diminish, the statistical efficiency is improved and fewer parameters are used compared to traditional Neural Network models. This is due the sparse interactions product of a kernel several times smaller than the input. The mathematical concept is defined in subsection 3.1.1

3.1.1. Convolution Operation

The convolution operation used in CNNs differs from the definition used in some other fields (e.g. Engineering, Mathematics). *Convolution* is an operation on two functions of a real-valued argument. It is represented in equation 3.1.1:

$$s(t) = \int x(a)w(t-a)da \tag{3.1.1}$$

However the convolution operations is often denoted with an asterisk, like in equation 3.1.2:

$$s(t) = (x * w)(t) \tag{3.1.2}$$

In convolutional network terminology, the argument x is known as the *input* and the other argument w as the *kernel*. The output *s(t)* is referred as *feature map*. In machine learning applications , the input and the kernel are multidimensional arrays. Since convolution is used for two-dimensional images:

$$s[i,j] = (I * K)[i,j] = \sum_m \sum_n I[m,n]K[i-m, j-n] \tag{3.1.3}$$

Another view is more straightforward for a machine learning library implementation. Due the commutative property of the convolution operation, (3.1.3) is equivalent to:

$$s[i,j] = (I * K)[i,j] = \sum_m \sum_n I[i-m, j-n]K[m,n] \tag{3.1.4}$$

Where I is the two-dimensional image and K is the two-dimensional Kernel. The commutative property is not fundamental for a neural network implementation. Neural network libraries have an operation called *cross-correlation*, which is the same as a convolution with no kernel flip as can be seen in equation 3.1.5. Many machine learning applications implement cross-correlation, but address it as convolution.

$$s[i,j] = (I * K)[i,j] = \sum_m \sum_n I[i+m, j+n]K[m,n] \tag{3.1.5}$$

Figure 3.1 show how a 2-D convolution takes place with no kernel-flipping. It can also be noticed that due the smaller kernel size, the convolution operation correspond to a sparse matrix. Most Neural Network models allow to easily replace a matrix multiplication with a convolution (as long as the model does not depend on specific matrix structures). Convolution can be seen as a multiplication by a matrix, nevertheless the matrix has its entries constrained . For unidimensional convolution this is known as *Toeplitz matrix*.For a 2-D convolution operation the *doubly block circular matrix* is the analogue. Convolution can be seen as multiplication by a matrix with several constrained entries (e.g. Univariate discrete convolution is constrained to be a *Toeplitz matrix*). A convolution in two dimensions corresponds to a *doubly block circular matrix*.

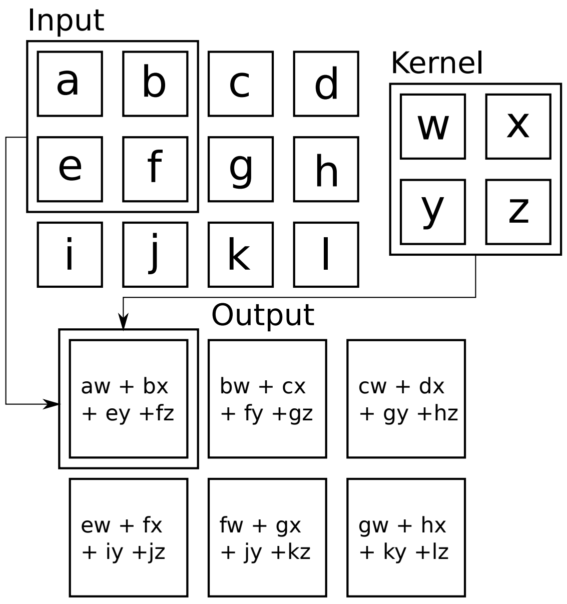

Figure 3.1.: 2-D convolution.

Typical CNN models make use of more specialized stages in order to deal with large data inputs. These stages perform different operations and together form a *Convolutional layer*. Convolutional layers are explained in the following subsection 3.1.2.

Variants of the Convolution function

As explained in the past section 3.1.1, the functions used in CNNs differ slightly from the convolution operation in the mathematical literature. First of all, there are several convolution operations which run in parallel, this is due the kernel properties. A kernel can only extract one kind of features, therefore multiple convolutions in a layer help to obtain many kind of features. Also the inputs are generally a grid of vector-based observations rather than real values. When working with images, the inputs and outputs are represented as 3-D tensors, with one index for the different channels (i.e. red, green and blue) and the other two indices for the spatial coordinates. In equation 3.1.6, $K_{i,j,k,l}$ is the kernel, which gives the connection strength between a unit in channel i of the output and a channel j of the input, with an offset of k rows and l between the output and the unit. The input data is formed by the elements $V_{i,j,k}$, where i indicates the channel and j and k indicate rows and columns respectively. The convolution is assumed without kernel flipping.

$$Z_{i,j,k} = \sum_{l,m,n} V_{l,j+m,k+n} K_{i,l,m,n} \tag{3.1.6}$$

In order to reduce computational costs, it is possible to skip over some positions of the kernel (the feature extraction becomes coarse). It can be seen as a downsampling in the convolution and appears in equation 3.1.7. The s is referred as the stride and allows to sample only every s pixels in each direction of the output.

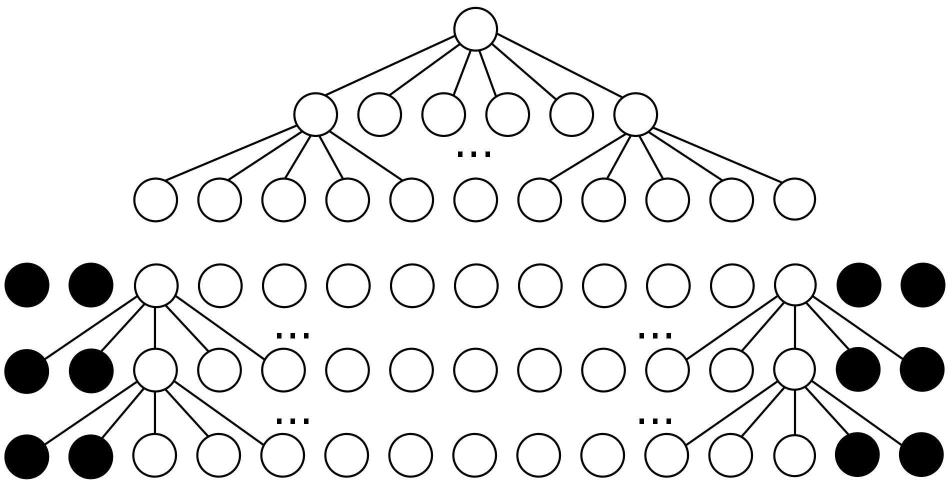
Figure 3.2.: *Zero Padding effect:* **TOP:** Zero padding is not implemented,, therefore the network shrinks its size. **BOTTOM:** Zero padding prevents the network from shrinking with depth.

$$Z_{i,j,k} = \sum_{l,m,n} [V_{i,j\times s+m,k\times s+n} K_{i,l,m,n}] \tag{3.1.7}$$

Zero-pad is one essential feature of the convolution that should be implement at the input in order to make it wider. The width of the representation is reduced by the kernel width - 1 at each layer if zero padding is not implemented. Therefore zero padding avoids the necessity to choose between shrinking the spatial representation and using small kernels.

In order to perform learning in a convolutional neural network, other operations are required. A Computation of gradients with respect to the kernel, given the gradients with respect to the output must be yield.

3.1.2. Convolutional Layer

CNN models are composed of several layers, where at least one of them is a *convolutional layer*. Convolutional layers have these stages:

- Convolution stage
- Detector stage
- Pooling stage
- (Local Response Normalization stage)

The norm stage appears in brackets because it is not always present in a convolution layer. However, it is present in several architectures. Figure 3.3 shows how all the stages inside a convolutional layer are interconnected. The convolution stage is already explained in 3.1.1 and the rest of the stages are explained below.

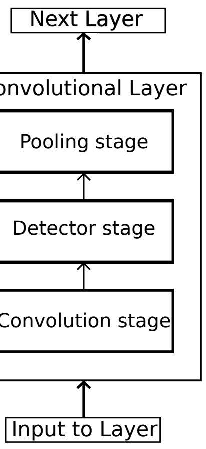

Figure 3.3.: Each convolution layer is divided into three stages.

Detector stage

In the detector stage an *activation function* is performed. An activation function defines an output for a given input or set of inputs. There are three main kinds of activation functions:

- *Hyperbolic tangent function (Tanh)*:

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{3.1.8}$$

- *Logistic sigmoid function*:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{3.1.9}$$

- *Rectified Linear Unit function (ReLU)*:

$$f(x) = max(0, x) \tag{3.1.10}$$

- *Parametric Rectifier Linear Unit function (PReLU)*:

$$f(x) = \begin{cases} ax & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases} \tag{3.1.11}$$

In equations 3.1.8, 3.1.9, 3.1.10 and 3.1.11; x represents the input and *f(x)* is the respective output. The behavior of each activation function can be observed in Figure 3.4. All of these functions are widely used for deep learning, but only the last two (ReLU and PReLU) are nowadays used for Convolutional Neural Networks. Therefore *tanh* and *sigmoid* functions are not covered. In order to coincide with Bengio, Goodfellow, and Courville (2015) the rectifier functions will be referred as *Detector stage*.

Zeiler et al. (2013) mention and prove empirically how CNNs benefit from ReLU functions. The nature of ReLU functions allows to easily train deep networks (even from random initialization). Equation 3.1.10 transform the system into a linear system since it only focuses on non-zero units. ReLU also yield a more regularized internal

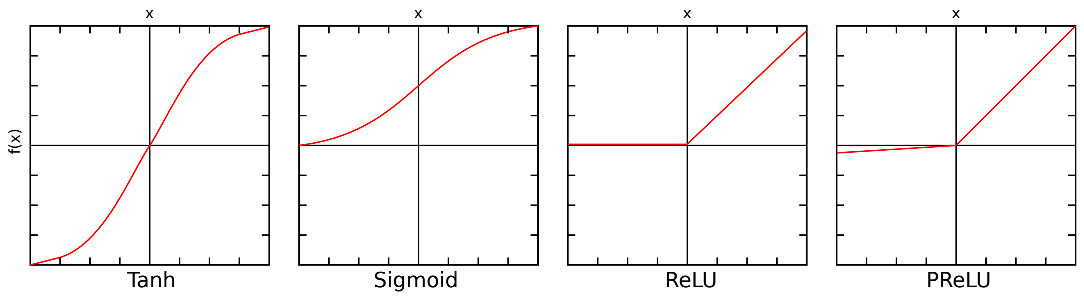

Figure 3.4.: The different types of activation functions used in Convnets.

representation (zero-outputs are often produced, if there is misalignment between inputs and internal weights). state that there are two reasons for ReLU to be beneficial for efficient learning of deep neural networks. The first reason is the system reduction to a linear convex system (it only focuses on units that are non-zero). The other reason is a more regularized internal representation. ReLU often produce zero-outputs, if the input is not aligned with the internal weights.

According to Glorot, Bordes, and Bengio (2011) ReLU units are suitable for CNNs due the sparsity of data, an inherent property of the convolution operation. It is worth to mention that not only vision tasks obtain a performance improvement, but also acoustic models (Maas, Hannun, and Ng (2013)) and speech processing (Zeiler et al. (2013)). It becomes obvious that Rectified Linear units are a must in order to yield state of the art results while using CNNs.

The Parametric Rectified Linear Unit (PReLU) is shown in figure 3.4 and can be seen as modified version of the ReLU function. It succesfully surpassed the performance achieved by ReLU based CNN models (He et al. (2015)). Its implementation improves model fitting with almost no extra computational effor and reducing the overfitting risk. PReLU Robustness lies in the a parameter from equation 3.1.11 which is learnable. This is similar to the Leaky Rectified Linear Unit (LReLU) proposed in **maas2013**, which uses a small fixed value ($a = 0.01$) to avoid zero gradients; but when comparing to ReLU only PReLU performance has a significant impact. In PReLU, only by adding few parameters with a negligible computation cost, a CNN performance is improved due the adaptative learning of activation function shapes.

Pooling stage

As shown in figure 3.3, the convolution layer is formed of convolution, detector and pooling stage. The pooling stage modifies the output of the layer further. It replaces the output in a certain location with a statistical summary of the nearby outputs. The Pooling stage comes after the Detector stage. It replaces the output in a certain location with a statisctial summary of the nearby outputs. Its purpose is to reduce the invariance caused by small translations inside the inputs. The most common pooling functions are:

- *Max function*: Maximum output within a rectangular neighborhood.
- *L2 norm function*: Squared root of the absolute values squared.
- *Average function*: Average output of a rectangular neighborhood.

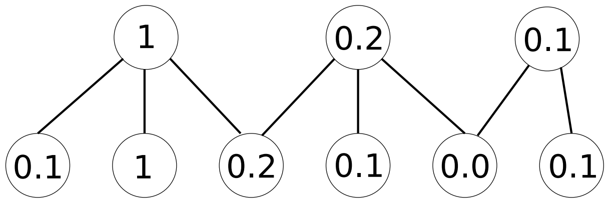

Figure 3.5.: Pooling with downsampling

- *Weighted average function*: Output based on the distance from the central pixel.

Invariance to local translation has been proved to be an important property if the vision task focuses more on the presence of a feature rather than in its location. However depending on the task to preserve the location of a feature is desired. Pooling summarizes over an output neighborhood allow to use fewer pooling stages than detector stages . Figure 3.5 shows how this summarization generates an increased computer efficiency and reduced memory requirements because the next layer will have fewer parameters to process. The figure depicts a max-pooling operation with a pool width of three and a stride between pools of two.

Convolution and Pooling can cause underfitting, therefore state of the art models (e.g Krizhevsky, Sutskever, and Hinton (2012) and Szegedy et al. (2014)) are designed to use pooling on some channels. This yields highly invariant features and features which are non prone to underfitting

Local Response Normalization (LRN)

Response normalization is implemented as a form of lateral inhibition, and is inspired on the real neurons. Hinton et al. (2012) introduced this type of layer in order to encourage competition for large activations amongst neuron outputs which are computed using different kernels. In order to yield a better generalization, the local normalization is performed as in equation 3.1.12.$b^i_{x,y}$ represents the response-normalized activity, $a^i_{x,y}$ denotes the activity of a computed neuron, where i is the kernel and (x, y) the position coordinates. The sum runs over n kernel maps at the same spatial position and N stands for the number of kernels in the layer.

$$b^i_{x,y} = a^i_{x,y} / \left(k + \alpha \sum_{j=max(0,i-n/2)}^{min(N-1,i+n/2)} (a^j_{x,y})^2 \right)^{\beta} \tag{3.1.12}$$

The constanst k,n,α and β are hyperparameters which are calculated using a validation set. A LRN layer follows a ReLU layer, though ReLU units do not require input normalization, but as Krizhevsky, Sutskever, and Hinton (2012) implemented it prevents saturating inputs.

3.1.3. Deconvolutional Layer

Zeiler, Taylor, and Fergus (2011) proposes this novel architecture which uses the same components of a convolutional layer (See section 3.1.2) but in reverse. Originally used to perform unsupervised learning, the deconvolutional layers are later used as a

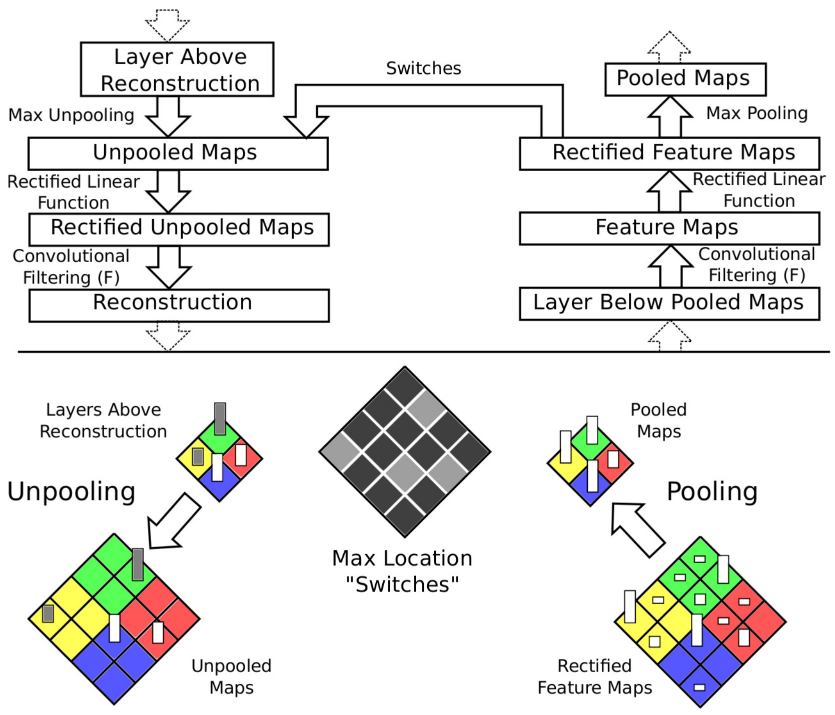

Figure 3.6.: **LEFT**: deconvolutional layer. **RIGHT**: convolutional layer. The deconvolution layer yields an approximate reconstruction of the layer beneath convolution layer. **BOTTOM**: This diagram depicts the unpooling operation in the deconvolutional layer. The black/gray squares are negative/positive activations in the feature map (Zeiler and Fergus (2014)).

visualization technique in CNNs. This approach appears in Zeiler and Fergus (2014). It allows to to find the performance contribution of a layer, and have also been used in image segmentation tasks (e.g. Long, Shelhamer, and Darrell (2014), Noh, Hong, and Han (2015))

Equation 3.1.13 shows how the reconstruction $\hat{y_1}^c$, where c represents the color channels, is performed by the sum of convolution operations of the feature maps $z_{k,1}$ with filters $f^c_{k,1}$.

$$\hat{y_1}^c = \sum_{k=1}^{K_1} z_{k,1} * f^c_{k,1} \tag{3.1.13}$$

The deconvolution tries to minimize the reconstruction error under a sparsity constraint and a set of feature maps. This is explained in equation 3.1.14. $C_l(y)$ stands for the cost function and comprises two terms weighted by λ_l a likelihood which keeps the $\hat{y}_t$ reconstruction close to y input, and a regularization term of the feature maps $z_{k,l}$.

$$C_l(y) = \frac{\lambda_l}{2} ||\hat{y}_l - y||_2^2 + \sum_{k=1}^{K_t} |z_{,l}|_1 \tag{3.1.14}$$

Figure 3.6 explains how a deconvolutional layer works. The activities performed in a convolutional layer are mapped back to the pixel space. The stages are performed successively in this order: unpooling, rectifying and filter.

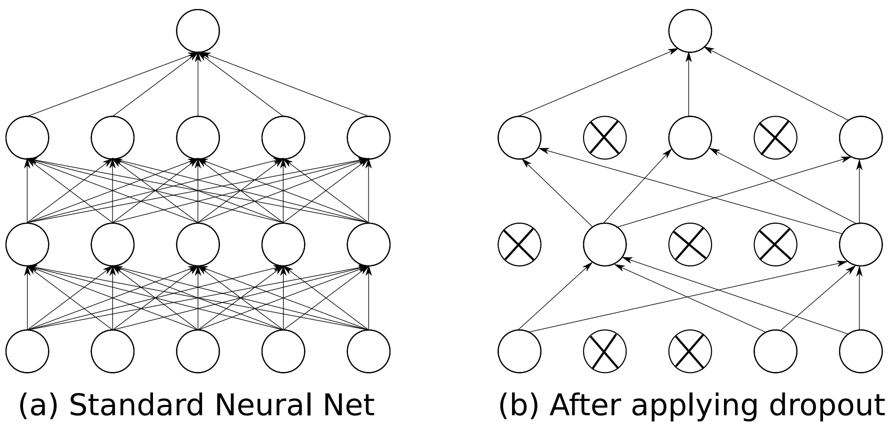

Figure 3.7.: **Left**: A standard neural net with 2 hidden layers. **Right**: An example of a thinned net, producht of the dropout technique.

3.2. Convolution parameters

Dropout

Another addition to the convolutional neural networks is the so called *dropout layer*. Deep neural nets are characterized for having a large number of parameters, therefore the nets are prone to overfitting. In order to address the overfitting problem, there are several techniques that can be implemented (e.g. soft weight sharing, penalties like L1 and L2 regularization, stop the training if the performance in the validation data is reduced).

It is possible to "regularize" a model by averaging the predictions of all possible setting of the parameters. This works well for small models, however the use of this technique on a several number of learned models that share parameters (i.e. Model combination) is computationally expensive.

Srivastava et al. (2014) proposed the *Dropout* technique in order to address the two cited problems (i.e. overfitting and high computational effort). It also allows to combine many different neural networks architectures in an efficient way. Figure 3.7 depicts the dropout technique, which can be define as the drop out of units in a neural network. Normally each unit is retained with a fixed probability p (usually set at 0.5) in order to be optimal for many kinds of networks and tasks. A Neural Network trained with the dropout technique can be seen as a collection of 2^n thinned networks, each of which shares its weights.

A model where dropout is performed can be described as follows: Given L hidden layers, let $l \in \{1, ..., L\}$ index the hidden layers. z^l and y^l are vectors of inputs into layer l and outputs from layer l respectively. w^l and b^l represent the weights and biases at layer l. Feed forward operation of a standard neural network is then described in equations 3.2.1 and 3.2.2.

$$z_i{}^{(l+1)} = w_i{}^{(l+1)} y^l + b_i{}^{(l+i)} \tag{3.2.1}$$

$$y_i{}^{(l+1)} = f(z_i{}^{(l+1)}) \tag{3.2.2}$$

where f is an activation function, like the ones seen in the detector section 3.1.2. The feed-forward operation in a network where dropout looks like this:

$$r_j^l \sim Bernoulli(p) \tag{3.2.3}$$

$$\tilde{y}^l = r^l * y^l \tag{3.2.4}$$

$$z_i^{(l+1)} = w_i^{(l+1)}\tilde{y}^l + b_i^{(l+1)} \tag{3.2.5}$$

$$y_i^{(l+1)} = f(z_i^{(l+1)}) \tag{3.2.6}$$

Equation 3.2.3 shows a r^l, which is a vector of random variables that have a probability p of being 1. In equation 3.2.4, $*$ denotes an element-wise product. The vector r^l is sampled and multiplied with the outputs y^l of that layer, to create a thinned output $\tilde{y}^l$. This process is applied at each layer and the outputs are used as inputs for the next layer.

The training of the dropout units is similar to the Stochastic Gradient Descent method used in standard neural networks. Forward and backpropagation are done on the thinned networks, while the gradients for each parameter are averaged over the training cases. Momentum, annealing learning rates and L2 weight decay are also useful to omprove the stochastic gradient descent. The noise generated by the dropout allows the optimization to be most robust by exploring different regions of the weight space.

It must be noticed that dropout introduces noise in the gradients if compared to the standard stochastic gradient descent (i.e. some grandients cancel each other). In order to overcome this issue, the learning rate should be increased 10-100 times and a high momentum must be chosen. This increase the speed of the learning process.

Nevertheless, the dropout technique in neural network has its drawbacks. The training process is 2-3 longer due a very noisy parameter update. Each training case focuses on a specific random architecture, therefore the calculated gradients are not the same gradients used in the final architecture. There is a trade-off between training time and overfitting that should be analyzed depending the characteristics of a given neural network (i.e. dataset, architecture).

3.2.1. Learning

The forward pass and backward propagation are the main operations being yield in a neural network.

The forward pass computes the output given the input for inference.

$$fw(x) = h(g(x)) \tag{3.2.7}$$

The backward pass *(back-propagation)* computes the gradients given the loss for learning

$$\frac{\partial fw}{\partial h}\frac{\partial fw}{\partial W_{ip}}\frac{\partial fw}{\partial g} \tag{3.2.8}$$

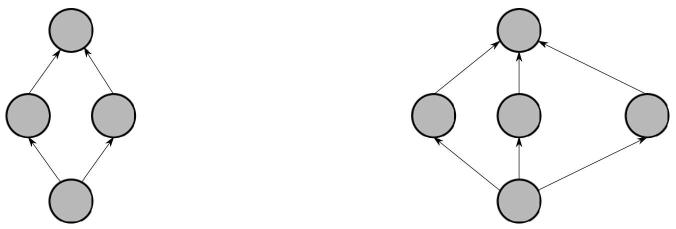

Figure 3.8.: backpropagation

Loss functions

A loss function evaluates the quality of the learning by mapping parameter settings to a scalar value, to test the *accuracy* of the learned parameters. The goal of the learning process is to find a set of parameters which minimizes the loss function.

When Neural Network models started to become popular, the most used loss function was the Least Squared Error (LSE) $L(f_w(x), y) = ||f_w(x) - y||^2$. Nevertheless this is not effective if the values of y are discrete (i.e. classification tasks), and therefore other loss functions should be employed.

Softmax

$$p = softmax(a) \Longleftrightarrow p_i = \frac{e^{a_i}}{\sum_j e^{a_j}} \tag{3.2.9}$$

Equation 3.2.9 is the *softmax* (**bridle1990probabilistic**) and its purpose is to specify multiple Bernoulli (*multinoulli*) distributions. Where a is a set of activations

The softmax loss layer is conceptually identical to the softmax layer applied before a multinomial logistic layer. It is the loss function used in Caffe

3.2.2. Optimization

The backward pass defines how to compute the gradient of the loss with respect to the model parameters. However, in order to update the weights of a Neural Network, gradient-based learning algorithms need to be implemented. Though several learning algorithms for deep neural networks exist, this thesis only covers some of them due their popularity and therefore further implementation in Caffe

Stochastic Gradient Descent (SGD)

Stochastic Gradient Descent is a widely used optimization algorithm for machine learning. It uses a stochastic estimator of the gradient in order to perform an update.

empirical risk $E_n(f)$ measures the training set performance, while the *expected risk* $E(f)$ calculates the expected performance on future examples. Both appear in equation 3.2.10

$$E(f) = \int l(f(x), y) dP(z) \qquad E_n(f) = \frac{1}{n}\sum_{i=1}^{n} l(f(x_i), y_i) \tag{3.2.10}$$

Gradient Descent (GD) minimizes the empirical risk $E_n(f_w)$. In each iteration the weights w on $E_n(f_w)$ are updated as in equation 3.2.11:

$$w_{t+1} = w_t - \gamma \frac{1}{n} \sum_{i=1}^{n} \nabla_w Q(z_i, w_t), \tag{3.2.11}$$

γ represents the learning rate, which achieves *linear convergence* in case of being sufficiently small and in conjunction with a w_o close to the optimum. If the scalar learning rate γ is replaced by a positive matrix Γ_t a better optimization can be designed. Equation 3.2.12 shows a *second order gradient descent*.

$$w_{t+1} = w_t - \Gamma_t \frac{1}{n} \sum_{i=1}^{n} \nabla_w Q(z_i, w_t). \tag{3.2.12}$$

Stochastic Gradient Descent is a widely used optimization algorithm for machine learning. It uses a stochastic estimator of the gradient in order to perform an update. It is also known as *online gradient descent* . It is a drastic simplification since each iterations estimates the gradient based on single random z_t:

$$w_{t+1} = w_t - \gamma_t \nabla_w Q(z_i, w_t) \tag{3.2.13}$$

The SGD optimizes the expected risk, due the fact that the examples are randomly drawn from a ground truth distribution. The randomly drawn examples introduces noise, therefore the SGD gradient does not become 0 even when a minimum is reached.

The learning rate γ_t is a crucial hyperparameter, which sometimes must be allowed to decrease its size in order to converge to a minimum.

Since the data sizes grow faster than the processing speed, Stochastic Gradient Descent have become popular. This is due its performance, which is good enough if the training time is the bottleneck.

$$w_{t+1} = w_t - \gamma_t \Gamma_t \nabla_w Q(z_t, w_t) \tag{3.2.14}$$

```
Stochastic Gradient Descent (SGD)
Require: Learning rate $\gamma_t$
Require: Initial parameter $w_t$
    while Stopping criterion not met do
        Sample a minibatch of $t$ examples from the training set $\mathbf{z}_{(1)}, ..., \mathbf{z}_{(t)}$
        Set $f(t) = 0$
        for i = 1 to $t$ do
            Compute gradient estimate: $f(t+1) = f(t) + \nabla_w Q(z_i, w_t)$
        end for
        Apply update: $w_{t+1} = w_t - \gamma_t f(t+1)$
    end while
```

Table with SGD Algorithm Bottou (2012) Bengio, Goodfellow, and Courville (2015) Krizhevsky, Sutskever, and Hinton (2012)

Momentum

Momentum helps to overcome inherent weaknesses of the stochastic gradient descent strategy. One of the main drawbacks of SGD is the slow training time, which is particularly slow when the gradient is small.

Momentum function is to speed up the learning process

```
Stochastic Gradient Descent (SGD) with momentum
Require: Learning rate $\gamma_t$, momentum parameter $\alpha$
Require: Initial parameter $w$, inivital velocity $v$
    while Stopping criterion not met do
        Sample a minibatch of $t$ examples from the training set $\mathbf{z}_{(1)}, ..., \mathbf{z}_{(t)}$
        Set f(t) = 0
        for $i = 1$ to $t$ do
            Compute gradient estimate: $f(t+1) = f(t) + \nabla_w Q(z_i, w_t)$
        end for
        Compute velocity update: $v = \alpha v - \gamma_t f(t+1)$
        Apply update: $w_{t+1} = w_t + v$
    end while
```

Adaptative Gradient (AdaGrad)

The AdaGrad algorithm scales the learning rates of all parameters inversely proportional to a sum of squared partial derivates over the iterations in order to adapt them individually. A rapid decrease in the learning rate corresponds to the parameters with large partial derivative of the loss. This yields a greater advance in the more sloped directions of parameter space.

The use of AdaGrad has a drawback if it is applied to deep neural networks. Due an accumulation of squared gradients, results in a premature and excessive decrease in the learning rate. The AdaGrad algorithm is explained in 3.2.2.

```
AdaGrad algorithm
Require: Global learning rate $\gamma$,
Require: Initial parameter $w_t$
    Initialize gradient accumulation variable $r = 0$
    while Stopping criterion not met do
        Sample a minibatch of $t$ examples from the training set $\mathbf{z}_{(1)}, ..., \mathbf{z}_{(t)}$
        Set $f(t) = 0$
        for $i = 1$ to $t$ do
            Compute gradient: $f(t+1) = f(t) + \nabla_w Q(z_i, w_t)$
        end for            Accumulate gradient: $r = r^2 + f^2(t+1)$
        Compute update: $\Delta w_t = \frac{\gamma_t}{\sqrt{r}} f(t)$
        Apply update: $w_{t+1} = w_t + \Delta w_t$
    end while
```

Bengio, Goodfellow, and Courville (2015) Duchi, Hazan, and Singer (2011)

Nesterov's Accelerated Gradient (NAG)

```
Stochastic Gradient Descent (SGD) with Nesterov Momentum
Require: Learning rate $\gamma$, momentum parameter $\alpha$.
Require: Initial parameter $w_t$, initial velocity $v$.
    while Stopping criterion not met do
        Sample a minibatch of $t$ examples from the training set $\mathbf{z}_{(1)}, ..., \mathbf{z}_{(t)}$
        Apply interim update: $w_{t+1} = w_t + \alpha v$
        Set $f(t) = 0$
        for $i = 1$ to $t$ do
            Compute gradient (at interim point): $f(t+1) = f(t) + \nabla_w Q(z_i, w_t)$
        end for
        Compute velocity update: $v = \alpha v - \gamma_t f(t+1)$
        Apply update: $w_{t+1} = w_t + v$
    end while
```

Nesterov (1983) Sutskever et al. (2013)

3.3. CNN outputs

CNN models have structured outputs. To achieve classification on a pixel level, in order to perform an image segmentation like in Clement Farabet and LeCun (2013). There is a trade-off due the pooling layers, since they subsample the feature maps and therefore they are skipped in some levels.

3.4. Classifiers

3.4.1. AlexNet

AlexNet is a classifier that won ImageNet Large-Scale Visual Recognition Challenge 2012 (ILSVRC12). (Cite ImageNet paper) The AlexNet architecture is shown in Figure . It has five Convolution Layers and three Fully Connected Layers. In order to train the AlexNet architecture, a dataset consisting in 1.2 million images representing 1000 categories is used. The output of its neurons is modelled by Rectified Linear Units (ReLU), which satisfy the non-saturating nonlinearity function $f(x) = max(0, x)$.

CNNs that use ReLU instead of other equivalents (e.g. tanh and sigmoid functions) train several times faster. It would not be possible to train large networks with the traditional saturation models such as the previously mentioned functions. Local Response Normalization (LRN) is present, because it aids generalization. Nevertheless ReLU prevent saturation for themselves. Pooling layers are contained to summarize the output of neighboring neurons in a statistical representation. In order to have an overlapping pooling, Stride value is set to two. This have shown to avoid overfitting during training. This network has 1000 non-spatial class labels. The output of the last fully connected layer is fed to a softmax layer which produce a distribution over the class labels.

The AlexNet includes the Dropout techinque (cite dropout paper), in which the output of each hidden neuron is given the value zero with a probability of 0.5. These neurons are "dropped out" and therefore do not take part in backpropagation and do not participate in the forward pass. This technique makes neurons unable to rely on the presence of particular neurons. Dropout encourages learning of robust features whoch are useful with different subsets of neurons in conjunction. There is a tradeoff within the Dropout technique: By decreasing the overfitting, the number of iterations needed to converge is doubled.

The Details of learning. This model is produced using Stochastic Gradient Descent (SGD) algorithm with a batch size of 128 images, the momentum of 0.9 and the weight decay of 0.0005. The weight decay is important for the network architecture to learn, since it reduces the network's training error. Zero-mean Gaussian distributions with a standard deviation of 0.01 are used to initialize the weights. The biases for the hidden layers are initialized with the constant 1 in the second, fourth and fifth Convolutional Layers, and in all fully connected layers; the remaining layers are initialized with a bias equal to the constant 0. The initial learnng rate is 0.01 and it gets divided by 10 when the validation error stops improving.

3.4.2. GoogLeNet

GoogLeNet participated and performed well in the ILSVRC14. Its results for classification and detection are the state of the art.It consist of 22 layers. Remarkable is that this network uses several times fewer parameters in comparison with other CNNs. In order to accomplish this a 22 layers architecture was designed with an intention to increase the network depth. This new concept is denominated "Inception". The main idea of the Inception architecture is to find an optimal local sparse structure using readily available components.

Traditional methods tend to increase the size of deep neural networks to improve their performance. However such an increase in size also has its drawbacks: The network has more parameters which makes it more prone to overfitting if the training data is not sufficient; the increase in size implies an increased use of computation resources. The GoogLeNet overcomes this problem in a more efficient way with its "Inception" technique which avoids the inherent problems in size increase. The network is divided into "Inception Modules" which are stacked in top of each other. They are shown in Figure.

This Inception approach was first proposed by (cite Lin et al). It basically adds 1x1 convolutional layers. 1x1 convolutional layers help to remove computational bottlenecks by dimension reduction. Figure b shows how this dimension reduction allows to deal with the prohibitively expensive 5x5 convolutions inside the "Inception Modules" while keeping the computational requirements expensive. In ordert to save computational budget and to increase the quality of the results, the GoogLeNet architecture is sparsely connected.

The GoogLeNet consists if these "Inception Modules" stacked upon each other. Max-pooling layers appear occasionally to halve the grid resolution. It must be noticed that the "inception modules" only appear in the higher layers, while the lower layers present a traditional convolutional approach. ReLU is the activation function for all the convolutions. In order to enable fine-tuning and adaptation of this network an extra linear layer was added. Dropout for GoogLeNet has a 70 percent ratio. The training method relies on Stochastic Gradient Descent (SGD), a 0.9 momentum and a fixed learning rate (it decreases 4% every 8 epochs).

3.5. Semantic Segmentation

The invariance to local translation is important for vision tasks where the presence of a feature rather than exact location is required.

Classifiers

Y. Lecunn (cite) created the modern CNNs in 1989

3.6. Efficient based graph segmentation

This algorithm yields a segmentation based on a graph-based approach and is contained in Felzenszwalb and Huttenlocher (2004). The *Efficient based graph segmentation* algorithm considers pixels as vertices (v) and the dissimilarity between two neighboring pixels (v_i, v_j) as edges. The image is considered as an undirected graph $G = (V, E)$ where V is the set of vertices ($v \in V$ and E is the set of edges ($v_i, v_j \in E$). Each edge (v_i, v_j) has a corresponding weight ($w(v_i, v_j)$ which measures the dissimilarity (e.g color, intensity, motion, etc.).

A segmentations is induced by a subset of edges. This means that a segmentation S partitions the set V into components which correspond to $G' = (V, E')$ such that $C \in S$ and $E' \subseteq E$. The main criterion to define a segmentation is to have similar elements in a component C. According to this criterion, edges between two elements in a component have low weights, while two edges between elements in different components must have higher weights.

A predicate D should be defined to evaluate if a boundary between two components is present (two regions in an image). The resulting predicate makes a comparison between the inter-component difference and the internal difference within the components. The predicate is therefore adaptive with respect to the local characteristics of data.

The *Internal difference* of a component $C \subseteq V$ is defined as the largest weight in the minimum spanning tree of the component itself ($MST(C, E)$). It is defined in the equation 3.6.1 :

$$Int(C) = \max_{e \in MST(C,E)} w(e) \tag{3.6.1}$$

The *difference between two components* is the minimum weight edge connecting these components. It is defined as follows:

$$Dif(C_1, C_2) = \min_{v_i \in C_1, v_j \in C_2, (v_i, v_j) \in E} w(v_i, v_j) \tag{3.6.2}$$

In case that no edge is connecting C_1 and C_2, the difference is considered to be infinite ($Dif(C_1, C_2) = \infty$). This criterion could be changed but it would modify the difficulty of the problem. In order for the region comparison to work, a threshold function must be used to check if evidence of a boundary between a pair of components exists. The pairwise comparison is shown in 3.6.3

$$D(C_1, C_2) = \begin{cases} \text{true} & \text{if } Dif(C_1, C_2) > MInt(C_1, C_2) \\ \text{false} & \text{otherwise} \end{cases} \tag{3.6.3}$$

However the size of a component is irrelevant to this definition. A threshold function τ must be defined to control the internal difference taking into account the component size. By adding this threshold to **??**, the next equation is obtained:

$$MInt(C_1, C_2) = \min(Int(C_1) + \tau(C_1), Int(C_2) + \tau(C_2)) \tag{3.6.4}$$

where:

$$\tau(C) = k/|C| \tag{3.6.5}$$

In equation 3.6.5, $|C|$ is the size of C and k is a constant parameter. A larger k value causes a preference for larger components.

A Gaussian filter is used to remove small artefacts product of the digitization. The Gaussian filter is set with a $\sigma = 0.8$, it smooths the image before computing the edge weights.

An extra parameter t can be set, in order to remove small and undesired components. It defines a minimum size of the components and enforces those components which are too small to be joined with their nearest neighbor.

This algorithm can yield a segmentation that is neither too coarse nor too fine. This is done by changing the k, σ and t parameters.

The *Efficient graph-based segmentation* is a fundamental part of the Causal Segmentation algorithm found in Couprie et al. (2013) which is going to be described later in this chapter.

3.6.1. Watershed cuts

The idea of watershed comes from topography: a drop of water which falls on a surface reaches the minimum by following a descending path. It can be definded as separating lines of the domain of attraction of drops of water. This notion is applied in an edge weighted-grap.

The regions of a watershed (*Catchment basins*) are the regional minima of the map. Each catchment basin contains a unique regional minimum and each regional minimum is included in a unique catchment basin. Edge-weighted graphs The concept of edge-weighted graphs is explained in 3.6 F is a set of all maps from E to Z and Watershed-cuts by the drop of a water principle: The *Minimum Spanning Forest* algorithm is explain in Table 3.6.1

```
Minimum Spanning Forest algorithm
Data: A weighted graph $G(V,E)$ and a set of labeled nodes markers $L$. Nodes of $V \backslash L$ have unknown labels initially.
Result: A labeling $x$ associating a label to each vertex.
Sort the edges of $E$ by increasing order of weight.
while any node has an unknown label do
    Find the edge $e_{ij}$ in $E$of minimal weight;
    if $v_i or v_j have unknown label values$ then
        Merge $v_i$ or $v_j$ into a single node, such that when the value for this merged node becomes known, all merged nodes are assigned the same value of $x$ and considered known.
```

3.6.2. Optical Flow

Optical Flow infers information about motion within a scene over time. Horn and Schunck (1981) defines the Optical Flow as the distribution of apparent velocities of movement of brightness patterns in an image. The discontinuities in the optical flow are useful to segment images into regions that correspond to different objects.

Motion is perceived when a changing picture is projected onto a stationary screen

$$\frac{dE}{dt} = 0 \tag{3.6.6}$$

$$\frac{\partial E}{\partial x}\frac{dx}{dt} + \frac{\partial E}{\partial y}\frac{dy}{dt} + \frac{\partial E}{\partial t} = 0 \tag{3.6.7}$$

$$u = \frac{dx}{dt} \text{ and } v = \frac{dy}{dt},$$

$$E_x u + E_y v + E_t = 0, \tag{3.6.8}$$

$$E_x, E_y \cdot (u, v) = -E_t.$$

A smoothness constraint is added since opaque objects of finite size are observed while they are subject of rigid motion or deformation. This constraint implies that neighboring points on the objects have similar velocities and the changes in the velocity field are smooth. However when an objects occludes another a discontinuity in flow occurs. The smoothness constraint can be expressed limiting the difference between the flow velocity at a point and the average volocity over a small neighborhood containing the point. The Laplacians of u and v are defined in equation 3.6.9

$$\bigtriangledown^2 u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} \text{ and } \bigtriangledown^2 v = \frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \tag{3.6.9}$$

3.7. Causal graph-based segmentation

Causal segmentation for this thesis purposes is based on Couprie et al. (2013). In order to define markers, independent segmentations are perfomed and the produced super-pixels are matched. A final segmentation is produced using the obtained markers by minimizing a global criterion defined on the image.

A segmentation S_{t+1} at a time $t+1$ which is consistent with a given segmentation S_t of an image at time t.

In order to produce super-pixels, independent segmentations are done using the *efficient graph-based method* explained in 3.6. These independent segmentations will be referred as $S'_1 ..., S'_t$.

After an independent segmentation S'_{t+1} is produced , the propagation of the temporal consistency between the non overlapping contours of S'_t and S'_{t+1} is required. This will be addressed as *Graph matching* procedure.

$$w_{ij} = \frac{(|r_i| + |r_j| d(c_i, c_j)}{|r_i \cap r_i|} + a_{ij}, \tag{3.7.1}$$

This causal segmentation algorithm makes use of watershed cuts found in Cousty et al. (2009). The watershed algorithm allows to reuse the sorting of edges done in 3.6 and represents the main computational effort. A detailed explanation of watershed cut algorithm is present in **??**

$$E(x) = \sum_{e_{ij} \in E} w_{ij}^p |x_j - x_i|^q + \sum_{v_i \in V} w_i^p |l_i - x_i|^q, \tag{3.7.2}$$

l represents a given configuration and x represents a target configuration

4 Experimental Framework

In order to yield semantic segmentation results product of *Convolutional Neural Networks*, this thesis uses and modifies two pre-trained *Convnets* (e. g. Alexnet and GoogLeNet). The AlexNet Krizhevsky, Sutskever, and Hinton (2012) won the ILSVRC12. The GoogLeNet Szegedy et al. (2014) performed especially well in ILSVRC14 and proposed an *inception* architecture. The adaptation of these classifier *convnets* is based on Long, Shelhamer, and Darrell (2014). Afterwards a temporary consistent segmentation is developed using the efficient graph-based segmentation Felzenszwalb and Huttenlocher (2004) as implemented in Couprie et al. (2013) The following sections explain in detail the experiments realised in each stage.

4.1. Adapting classifiers

The main change done inside the selected networks w Replacing the Inner Classifiers yield a non-spatial outputs. The fully connected layers (convolution layers) take fixed size inputs and produce spatial coordinates as outputs. If the convolution are considered to cover entire input regions. By modifying the last layers within the Classifiers, a coarse pixelwise prediction is produced. The network becomes a fully convolutional layer takes inputs and yields a classification map as an output.

The key factor are these outputs maps and they are a natural choice for semantic segmentation. A drawback of this approach are the output dimensions, which are reduced by subsampling. This yields a coarse output, reducing it from the original input size by a factor of the stride within each pooling layer.

4.2. F

Hariharan et al. **\^a\IeC {\texteuro }\IeC {\textcent }** introduce the concept of a hypercolumn. It basically starts from the idea that recognition algorithms only use the information of the last layer of the CNN. This makes sense for tasks like classification, due the invariance product of the pooling layers. For tasks as segmentation, pose estimation, detection, et cetera. this last layer is not enough for an optimal representation. Intermediate layers possess information less prone to be sensitive to semantics. Multiple levels of abstraction and scale are recommended to consider information within inner layers. Farabet et al. **\^a\IeC {\texteuro }\IeC {\textcent }** consider this fact by creating a multiscale feature learning method. This thesis approach is inspired in Long et al. **\^a\IeC {\texteuro }\IeC {\textcent }** combining the last prediction layer with inner layers finer strides. The combination of shallow and deep layers information yields an output for a global structure.

Part of the experiments was to select where to start the inner layer branching. Inside each layer exist Pooling, Convolution, Normalization and Detector stages. After each pooling stage the input size is downsampled (stride).

Early branching stages and Late branching stages are presented. The early branching happens right after the respective Pooling stage having less processed data. The latter on the other hand occurs just before the next Pooling stage, this has

4.3. AlexNet

Krizhevsky et al. **\^a\IeC {\texteuro }\IeC {\textcent }**

4.4. Dropout

4.5. Weight Filler

When a layer is created from scratch it has no values. In order to modify the learning process the bias and the weight initializers can be modified. This thesis present the obtained results with 3 types of weight fillers: Gaussian, Xavier **\^a\IeC {\texteuro }\IeC {\textcen**
and no weight filler. For the bias, 0 and 1 values were chosen. In the case of the Xavier and Gaussian fillers, stride

4.6. Learning Rate

- Weight Filler:
  - Xavier Filler
  - Gaussian Filler
  - No Filler
- Learning Rate
- Momentum
- Learning Policy

4.7. Net Surgery

Net Surgery is . It allows to transplant previous learned weights from one layer into another. For CNNs this is an advantage since it ports the already available information from the inner product layers inherent in a classifier. These transplanted weights are later used to initialize the new convolutional layers.

4.8. Fine-tuning

Fine-tuning is the act of modifying an already available layer by changing its number of outputs. The modified layer re-learns its weights and use previous data to do so. In the chosen network architectures. Fine-tuning allows to use pre-trained networks and cope with the need of enormous amount of data for training. The main advantage of fine-tuning is

4.9. CamVid Dataset

The CamVid dataset Brostow, Fauqueur, and Cipolla" ("2008") contains 701 images from traffic scenes. The dataset is provided with High Resolution images (960x720) taken at 1Hz and 15 Hz respectively. It has 32 semantic labels that were manually added. In order to accomplish an appropriate fine-tuning for the AlexNet and GoogLeNet, these images were used as follow:

- Images are downsampled by a factor of two. This changes the resolution from 960x720 to 480x360. The main reason to do this downsampling is due hardware limitations that make training more difficult and time consuming. The raw pixels images are downsampled using the *Antialiasing* algorithm, while the ground truth images use the *Nearest Neighbor* algorithm, which avoid color mixing that would otherwise corrupt our ground truth.
- Training and testing datasets are gotten from Brostow et al. (2008). This allows our results to be compared with other segmentation algorithms.
- Because the dataset could be to small to train efficiently a CNN, *Data augmentation* is applied. In order to augment the available training data, the images are mirrored, and the center of the images with original resolution (960x720) is cropped.
- A mask is applied to the 32 semantic classes, just leaving the 11 more frequent classes (See Table xx). This reduces computation requirements and allow to compare results with already available segmentation algorithms results on CamVid data set (cite papers with CamVid accuracy).

4.10. Temporary Consistent segmentation

CamVid provides the video sequences from where the image dataset was obtained. All the images are obtained from these video sequences using the **FFMPEG** software. These images belong to the video sequences Seq05VD and 0001TP. After a frame by frame segmentation using the fine-tuned CNN, these images are processed using the code from The parameters to be modified belong to the efficient graph based segmentation The result images are later merged in a video back into a video. The images that are available in the original dataset will be taken with their respective segmentation to be compared with the frame by frame approach.

4.11. Results

5 Conclusions and Outlook

5.1. Outlook

With the advances in autonomous driving

In this thesis an approach to understand traffic scenes using deep learning methods

5.2. Future Work

The results of this thesis could be expanded in the near future in several ways. Development in Graphic Processor Units, (GPU). Computer Vision challenges (i.e. Pascal VOC, Imagenet, et cera.) keep on showing outperforming results every year. New techniques and methods yield more accurate predictions in vision tasks. GPUs manufacturers are not only developing more powerful hardware, but also improving (CUDNN) These perks will allow to train bigger, more complex models in a faster and more efficient way. To have a real time semantic segmentation with an good understanding of the surroundings... Information from Pedestrians, Autos, Biciclyst and more important classes could be used for security. Traffic signs, and (pinturas en el asfalto) could be interpreted and used to simplify autonomous driving. Some works (cite paper) already show similar approaches with acceptable resutls. Nevertheless as time passes, technologies are developed and more informationd and more capable devices

5.3. Results and Conclusions

The work in this thesis shows that with . I would like to remark that some already available models have a better performance, but due hardware limitations it was not feasible to train certain models. Some image preprocessing was also mandatory to cope with traffic scene understanding. All the images belonging to the CamVid dataset were downsampled. This probably means that by having input images with greater quality, the learning process inside the CNN would perform better. The following tables show the obtained results on the CamVid dataset by using independent segmentations obtained from a CNN and temporary consistent segmentations. We have moved from the HOG and SIFT era to the convolutional networks features era.

Bibliography

Bengio, Y., I. J. Goodfellow, and A. Courville (2015). “Deep Learning”. Book in preparation for MIT Press. URL: http://www.iro.umontreal.ca/~bengioy/dlbook.

Bottou, L. (2012). “Stochastic gradient descent tricks”. In: *Neural Networks: Tricks of the Trade*. Springer, pp. 421–436.

Brostow, G. J., J. Fauqueur, and R. Cipolla" ("2008"). “"Semantic Object Classes in Video: A High-Definition Ground Truth Database"”. In: *"Pattern Recognition Letters"* "xx"."x", "xx–xx".

Brostow, G. J., J. Shotton, J. Fauqueur, and R. Cipolla (2008). “Segmentation and Recognition Using Structure from Motion Point Clouds”. In: *ECCV (1)*, pp. 44–57.

Chen, L.-C., G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille (2015). “Semantic Image Segmentation with Deep Convolutional Nets and Fully Connected CRFs”. In: *ICLR*. URL: http://arxiv.org/abs/1412.7062.

Ciresan, D. C., U. Meier, J. Masci, L. Maria Gambardella, and J. Schmidhuber (2011). “Flexible, high performance convolutional neural networks for image classification”. In: *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*. Vol. 22. 1, p. 1237.

Clement Farabet Camille Couprie, L. N. and Y. LeCun (2013). “Learning Hierarchical Features for Scene Labeling”. In: *IEEE Trans. on Pat. Anal. and Mach. Int.* in press.

Couprie, C., C. Farabet, Y. LeCun, and L. Najman (2013). “Causal graph-based video segmentation”. In: *Image Processing (ICIP), 2013 20th IEEE International Conference on*. IEEE, pp. 4249–4253.

Couprie, C., C. Farabet, L. Najman, and Y. Lecun (2014). “Convolutional nets and watershed cuts for real-time semantic labeling of rgbd videos”. In: *The Journal of Machine Learning Research* 15.1, pp. 3489–3511.

Cousty, J., G. Bertrand, L. Najman, and M. Couprie (2009). “Watershed cuts: Minimum spanning forests and the drop of water principle”. In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 31.8, pp. 1362–1374.

Donahue, J., Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell (2013). "Decaf: A deep convolutional activation feature for generic visual recognition". In: *arXiv preprint arXiv:1310.1531*.

Duchi, J., E. Hazan, and Y. Singer (2011). "Adaptive subgradient methods for online learning and stochastic optimization". In: *The Journal of Machine Learning Research* 12, pp. 2121–2159.

Ess, A., T. Mueller, H. Grabner, and L. J. Van Gool (2009). "Segmentation-Based Urban Traffic Scene Understanding." In: *BMVC*. Vol. 1. Citeseer, p. 2.

Felzenszwalb, P. F. and D. P. Huttenlocher (2004). "Efficient graph-based image segmentation". In: *International Journal of Computer Vision* 59.2, pp. 167–181.

Fischer, P., A. Dosovitskiy, and T. Brox (2014). "Descriptor matching with convolutional neural networks: a comparison to sift". In: *arXiv preprint arXiv:1405.5769*.

Fukushima, K. (1980). "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position". In: *Biological cybernetics* 36.4, pp. 193–202.

Girshick, R., J. Donahue, T. Darrell, and J. Malik (2014). "Rich feature hierarchies for accurate object detection and semantic segmentation". In: *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*. IEEE, pp. 580–587.

Glorot, X., A. Bordes, and Y. Bengio (2011). "Deep sparse rectifier neural networks". In: *International Conference on Artificial Intelligence and Statistics*, pp. 315–323.

Grundmann, M., V. Kwatra, M. Han, and I. Essa (2010). "Efficient hierarchical graph-based video segmentation". In: *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*. IEEE, pp. 2141–2148.

Hariharan, B., P. Arbeláez, R. Girshick, and J. Malik (2014). "Hypercolumns for object segmentation and fine-grained localization". In: *arXiv preprint arXiv:1411.5752*.

He, K., X. Zhang, S. Ren, and J. Sun (2015). "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification". In: *arXiv preprint arXiv:1502.01852*.

Hinton, G. E., N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov (2012). "Improving neural networks by preventing co-adaptation of feature detectors". In: *arXiv preprint arXiv:1207.0580*.

Hong, S., H. Noh, and B. Han (2015). "Decoupled Deep Neural Network for Semi-supervised Semantic Segmentation". In: *arXiv preprint arXiv:1506.04924*.

Horn, B. K. and B. G. Schunck (1981). "Determining optical flow". In: *1981 Technical symposium east*. International Society for Optics and Photonics, pp. 319–331.

Jia, Y., E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell (2014). "Caffe: Convolutional Architecture for Fast Feature Embedding". In: *arXiv preprint arXiv:1408.5093*.

Krizhevsky, A., I. Sutskever, and G. E. Hinton (2012). "Imagenet classification with deep convolutional neural networks". In:

Le Cun, B. B., J. S. Denker, D Henderson, R. E. Howard, W Hubbard, and L. D. Jackel (1990). "Handwritten digit recognition with a back-propagation network". In: *Advances in neural information processing systems*. Citeseer.

LeCun, Y., L. Bottou, Y. Bengio, and P. Haffner (1998). "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11, pp. 2278–2324.

Lin, G., C. Shen, I. Reid, et al. (2015). "Efficient piecewise training of deep structured models for semantic segmentation". In: *arXiv preprint arXiv:1504.01013*.

Long, J., E. Shelhamer, and T. Darrell (2014). "Fully convolutional networks for semantic segmentation". In: *arXiv preprint arXiv:1411.4038*.

Long, J. L., N. Zhang, and T. Darrell (2014). "Do Convnets Learn Correspondence?" In: *Advances in Neural Information Processing Systems 27*. Ed. by Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger. Curran Associates, Inc., pp. 1601–1609. URL: `http://papers.nips.cc/paper/5420-do-convnets-learn-correspondence.pdf`.

Maas, A. L., A. Y. Hannun, and A. Y. Ng (2013). "Rectifier nonlinearities improve neural network acoustic models". In: *Proc. ICML*. Vol. 30.

Miksik, O., D. Munoz, J. A. Bagnell, and M. Hebert (2013). "Efficient temporal consistency for streaming video scene analysis". In: *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. IEEE, pp. 133–139.

Nesterov, Y. (1983). "A method of solving a convex programming problem with convergence rate O (1/k2)". In: *Soviet Mathematics Doklady*. Vol. 27. 2, pp. 372–376.

Noh, H., S. Hong, and B. Han (2015). "Learning Deconvolution Network for Semantic Segmentation". In: *arXiv preprint arXiv:1505.04366*.

Paris, S. (2008). "Edge-preserving smoothing and mean-shift segmentation of video streams". In: *Computer Vision–ECCV 2008*. Springer, pp. 460–473.

Paris, S. and F. Durand (2007). "A topological approach to hierarchical segmentation using mean shift". In: *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*. IEEE, pp. 1–8.

Rabiner, L. R. (1989). "A tutorial on hidden Markov models and selected applications in speech recognition". In: *Proceedings of the IEEE* 77.2, pp. 257–286.

Sermanet, P., D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun (2013). "Overfeat: Integrated recognition, localization and detection using convolutional networks". In: *arXiv preprint arXiv:1312.6229*.

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

Srivastava, N., G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov (2014). "Dropout: A simple way to prevent neural networks from overfitting". In: *The Journal of Machine Learning Research* 15.1, pp. 1929–1958.

Sutskever, I., J. Martens, G. Dahl, and G. Hinton (2013). “On the importance of initialization and momentum in deep learning”. In: *Proceedings of the 30th international conference on machine learning (ICML-13)*, pp. 1139–1147.

Szegedy, C., W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich (2014). “Going deeper with convolutions”. In: *arXiv preprint arXiv:1409.4842*.

Zeiler, M. D. and R. Fergus (2014). “Visualizing and understanding convolutional networks”. In: *Computer Vision–ECCV 2014*. Springer, pp. 818–833.

Zeiler, M. D., G. W. Taylor, and R. Fergus (2011). “Adaptive deconvolutional networks for mid and high level feature learning”. In: *Computer Vision (ICCV), 2011 IEEE International Conference on*. IEEE, pp. 2018–2025.

Zeiler, M. D., M. Ranzato, R. Monga, M. Mao, K. Yang, Q. V. Le, P. Nguyen, A. Senior, V. Vanhoucke, J. Dean, et al. (2013). “On rectified linear units for speech processing”. In: *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*. IEEE, pp. 3517–3521.

A
Appendix

Das ist der Anhang (optional).