Politecnico di Milano
Corso di Laurea Magistrale in
Computer Science and Engineering
Dipartimento di Elettronica e Informazione

Pixelwise Semantic Segmentation of urban scenes using Recurrent Deep Neural Networks

AI & R Lab
Laboratorio di Intelligenza Artificiale
e Robotica del Politecnico di Milano

Advisor: Prof. Matteo Matteucci
Co-advisor: Dr. Francesco Visin

Master graduation thesis of
Marco Ciccone
Student ID: 818260

Academic Year 2014-2015

To my beloved one

"Statistics are like bikinis.
What they reveal is suggestive,
but what they conceal is vital."

Aaron Levenstein

"If you're going to try,
go all the way.
Otherwise don't even start."

Charles Bukowski
Roll the dice

Abstract

Semantic Segmentation is a low level structured prediction task that aims to correctly infer the semantic label of each pixel in an image. Thanks to the breakthrough of Deep Learning, Semantic Segmentation gained a significant interests in particular applied to *Autonomous Driving*.

Almost the totality of the state-of-the-art Semantic Segmentation architecture uses Convolutional Neural Networks (CNNs). Due to the limited size of the local receptive fields, CNNs can usually fail to capture spatially long-range dependence across different local areas of the image. Although these networks have showed their excellent performance for Image Recognition and Classification tasks [1], to correctly modeling the contextual long-term dependencies between distant pixels is crucial for Semantic Segmentation.

Recurrent Neural Networks (RNNs) have proved their ability to modeling long-term dependences in several sequential prediction tasks such as speech recognition and language understanding. In this work we focused on *ReSeg*, a fully recurrent architecture proposed by Visin et Al. [2] for object segmentation. Each recurrent layer consists of two bidirectional RNNs that scan the image vertically and horizontally extracting both local and global features. The recurrent topology of the network allows to learn long term spatial correlation and dependencies between pixels in the image in the context of the entire image. We performed a greedy procedure for hyper-parameter optimization, then, we extensively tested the network on challenging urban scene parsing datasets such as Camvid [3] and Cityscapes [4] showing comparable results to the state-of-the-art convolutional models. We further extended the model combining both convolutional and recurrent layers, showing that the convolutional-based models for Semenatic Segmentation actually benefit from the use of ReSeg layers.

Sommario

La Segmentazione Semantica (Semantic Segmentation) è un tipo di classificazione di basso livello che ha come obiettivo quello di predire la categoria semantica a cui appartiene ogni pixel all'interno di un'immagine. Negli ultimi anni, grazie ai numerosi sviluppi fatti nell'ambito del Deep Learning, l'interesse nei confronti del problema di Semantic Segmentation ha subito un forte incremento, soprattutto nel contesto della *guida autonoma*.

La maggior parte dei sistemi di Semantic Segmentation è basato su Reti Neurali Convolutive (CNNs). Questo tipo di modelli ha dimostrato di raggiungere performace eccellenti nel problema di riconoscimento di immagini, ma non sono sempre in grado di apprendere correttamente le dipendenze tra aree dell'immagine distanti fra loro. Questo tipo di dipendenze è cruciale al fine di ottenere una buona segmentazione.

Le Reti Neurali Ricorrenti (RNNs) hanno dimostrato di essere in grado di modellare le dipendenze a lungo termine in numerosi problemi di predizione sequenziale, quali riconoscimento vocale, comprensione del linguaggio naturale o traduzione automatica. Il nostro lavoro è basato sul modello ricorrente *ReSeg* proposto da Visin et Al. [2] per la segmentazione di oggetti. Ogni layer ricorrente è formato da due RNNs bidirezionali che scorrono l'immagine verticalmente e orizzontalmente al fine di estrarre features sia locali che globali. Grazie alla sua topologia, la rete è in grado di apprendere le dipendenze spaziali tra i pixel nel contesto dell'intera immagine.

In questo lavoro di tesi abbiamo utilizzato una procedura *greedy* al fine di ottimizzare gli iper-parametri della rete e avere un primo grado di sensibilità sulle capacità del modello. Successivamente abbiamo testato le performance della rete su dataset di segmentazione semantica in ambito stradale come Camvid [3] e Cityscapes [4], ottenendo performance di stato dell'arte confrontabili con quelle dei ben più usati modelli convolutivi.

Abbiamo inoltre esteso il modello combinando insieme livelli convolutivi e ricorrenti, dimostrando che i modelli convolutivi possono beneficiare dell'introduzione di livelli ricorrenti incrementando così le performance di segmentazione.

Acknowledgements

This work is dedicated to all the people that I love and that are always present in my life. The time that the academic studies have stolen to your company will never be returned, but I want you to know that is the most valuable thing that I've lost in these years.

I have to thank my projects and study mates *Federico* (thank you for teaching me how to take perfect notes), *Mattia*, and all the guys who shared with me these two years: *Foja*, *Iacopo*, *Peppe*, *Riccardo*, *Camper*, *Stefano* and *Peter*.

I would like to thank Prof. Matteucci for introducing me to Machine Learning and encouraging me during these months, understanding the passion that I always put in my work. Thank you for letting me pursue a thesis like this after that I insisted so much and despite all the initial worries and hesitations. Of course this work wouldn't be possible without the help of *Francesco* who guided me day after day. Thank you for your countless advices, corrections, and code reviews. Moreover I would like to acknowledge S.C. for the moral support and for being always by my side in the time of need.

A special mention goes to Prof. Tagliasacchi who has been a model for me during these years. Your natural way of explaining complex concepts as if they were the simplest things in the world will be always source of inspiration for me.

The current thesis work has been possible thanks to the NVIDIA Hardware Grant Program that has provided the GeForce GTX TITAN X that we used to train and to evaluate our model.

Vorrei inoltre dedicare una sezione particolare alla mia famiglia. So che questi mesi sono stati difficili anche per voi. Grazie ai miei genitori, per essermi stato vicino, anche quando spesso vi ho allontanato. Grazie per avermi sostenuto e insegnato a resistere nonostante le avversità che la vita ti mette davanti. Grazie a mia nonna *Rosa*, senza di lei non sarei mai potuto partire.

Ringrazio i miei fratelli *Filippo* e *Carlotta*, anche se sono sempre lontano vi penso sempre. Trovate la vostra strada, sperimentate, non abbiate paura di rischiare e fate sempre quello che vi rende felici e vi da soddisfazione senza lasciarvi mai abbattere da niente e da nessuno. Vi auguro tutta il bene e la felicitá del mondo. Vi voglio bene. Marco.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Since the introduction of the first computers, researchers have always been attracted by the idea of creating a sort of Artificial Intelligence (AI) that could help humans improve their lives.

Nowadays computers are still far from reaching the performance of the human brain but AI systems are increasingly present in our day-to-day life and help us making choices and solving many more tasks.

Voice assistants have become highly accurate in understanding human language and interacting with us by synthetic speech. Every time we browse the web looking for something to buy, recommendation systems suggest us products, books or movies that we might like based on our history. When we listen to a song by streaming, the music software can create personalized playlists with songs of the same genre or with similar rhythmic patterns. Facebook and Google released their application to automatically recognize faces and tag people in our photographs. Microsoft integrated in Skype a system that is able to perform speech-to-speech translation from a language to another during a voice call in real time, and recently DeepMind built *AlphaGo* [6], a program that learned how to play Go from experience which even won against the world champion.

All of these intelligent systems share the same general idea of trying to mimic how our brain works. Just like during our life we see a lot of examples and learn how to recognize faces, objects, or situations, the modern AI systems use machine learning algorithms to build mathematical models that try to reproduce the same abilities of our brain by processing a huge amount of data samples and learning the underlined representation.

For instance, Computer Vision and Machine Learning focus together on how to make a machine able to develop a vision system that understands the meaning of an image and solves many different tasks such as Image Classification, Face Recognition, Object Detection and ultimately Semantic Segmentation in order to replicate the abilities of human vision and perception.

The performance of a machine learning system crucially depends on the feature representations of the input data. Until few years ago most of the machine learning methods relied on human-designed representations and inputs features. When machine learning is applied only to these input features it becomes merely a matter of parameter optimization in order to make the best final prediction. However handcrafting features is time-consuming and features are often task specific so it's hard to use them for different application.

In the last years a new track of machine learning called *Deep Learning* focused on providing a way to automatically learn a feature representation. The idea behind Deep Learning is indeed to learn a general feature representation that can be exploited for different tasks instead of building hand-engineered features that rely on fixed heuristics. This can be done for example by stacking several layers of Neural Networks, each layer learns a more complex representation of the data. For example Convolutional Neural Networks trained on images learn similar levels of representations as the human brain where the first layer learns simple edge filters, the second layer captures primitive shapes and higher levels combine these to form objects. Therefore, Deep learning can be seen as putting together *Representation Learning* with Machine Learning, attempting to jointly learn good features, across multiple levels of increasing complexity and abstraction, and the final prediction.

The breakthrough of Deep Learning had a strong impact on Machine Learning and in particular on Computer Vision, drastically advancing the state of the art of Image Understanding algorithms. This opened new scenarios for a lot of computer vision applications: one of most interesting and challenging is *Autonomous Driving*.

Autonomous Driving is a fascinating topic that blends together a lot of tasks such as urban scene understanding, sensor fusion and path planning in order to build a self-driving car that autonomously drives in the traffic. There is a significant hype on the possibility to build a self-driving car. One of the first company that showed his interest in building a self-driving car was Google. Then many car companies such as Volvo, Tesla, BMW, Daimler, Mercedes-Benz and Toyota planned to release their self-driving car by 2020, and also Baidu started its research on Autonomous Driving collecting real traffic data to build its own prototypes. Moreover NVIDIA is currently working with most of these companies [7] in order to integrate its hardware in their vehicles. Indeed, Autonomous Driving was the main topic of the NVIDIA keynote at CES 2016, where it's been presented the DriveNet neural architecture and the new PX-2 GPU-based super-computer.

The most critical component to design in an autonomous vehicle is a perception system that is able to understand all the stimulus coming from the surrounding environment. For instance the perception system must know how to recognize roads, sidewalks, buildings, traffic lights, sign symbols, cars, pedestrians and all the objects or situations in the scene with a degree of accuracy at least comparable with that of a human driver. These systems usually acquire data from several camera and depth sensors such as LIDAR, but most of the information is visual, so much of the inference is done by computer vision algorithms.

The field of computer vision that focuses on this task goes under the name of *Scene understanding* and it groups together different subtasks of growing complexity such as Image or Scene Recognition, Object detection and ultimately *Semantic Segmentation* in order to make inference from the visual informations.

In particular Semantic Segmentation is a low level structured prediction task that aims to correctly infer the semantic label of each pixel in an image in order to classify each object in the scene. This kind of task has acquired great importance in the last years in the context of Autonomous Driving, in fact, semantically classifying each object in a urban scene is a critical task in the design of a reliable autonomous car that is able to safely drive in the traffic.

Thanks to the introduction of large annotated datasets collected on real urban street environments such as KITTI [8], and the recently released Cityscapes dataset [4] combined with the increasing computational resources given by the use of GPUs, feature learning with deep neural networks has become the dominating technique for solving a variety of computer vision tasks including Semantic Segmentation. Almost the totality of these systems is built using Convolutional Neural Networks (CNNs) which efficiently exploits the inherent structure of the images.

CNNs have been adapted to perform semantic segmentation tasks [9, 10] and achieved state-of-the-art results, but they suffers a few limitations in modeling the contextual dependencies in distant regions of the image. Specifically, the receptive field of a neuron in the convolutional layer usually corresponds only to a small fixed-sized *local* area of the input image. However, in semantic segmentation, contextual informations from distant areas of the image are usually crucial for a correct prediction, and the context introduced by the local receptive field is not enough.

Due to the limited size of the local receptive fields, CNNs usually fails to capture such spatially long-range dependence across different local areas of the image. In particular it's hard to extract a good feature representation to labeling pixels just by looking at a small neighboring region. To overcome this issue the size of the receptive field is artificially adjusted to gradually cover the entire image by stacking several convolutional and pooling layers, resulting in an implicit but ineffective way of modeling long-range dependencies.

Although these networks have showed their excellent performance for Image Recognition and Classification task, modeling correctly the contextual long-term dependencies between distant pixels is crucial for Semantic Segmentation. In fact, the semantic category of a pixel may depend on relatively short-range information (e.g., the presence of eyes in a human face generally indicates the presence of a nose nearby), but may also depend on long-range information. For example, identifying a grey pixel as belonging to a road, a sidewalk, a gray car, a concrete building, or a cloudy sky requires a wide contextual window that shows enough of the surroundings to make an informed decision.

Recurrent Neural Networks (RNNs) have proved their ability to modeling long-term dependences in several sequential prediction tasks such as speech recognition and language understanding. Nevertheless, RNNs are relatively new and not fully explored in the context of image understanding. A recent work on this topic is the *ReNet* architecture [11] that has been initially proposed as an alternative to the widely used Convolutional Neural Networks to perform object classification. Each ReNet layer is made of two bidirectional recurrent neural networks that scan the image vertically and horizontally in order to extract both local and global features. The recurrent topology of the network allows to learn long term spatial correlation and dependencies between pixels in the image, not only in a fixed size window such as classical CNNs, but possibly in the context of the entire image. This is critically important to task such semantic segmentation.

In this work we focused on the *ReSeg* architecture proposed by Visin et Al. [2] for object segmentation. Starting from their work we extended the ReSeg architecture to address multi-class semantic segmentation and we performed a greedy procedure for hyper-parameter optimization in order to have a first understanding of the strengths and the weaknesses of the model. We implemented the ReSeg architecture using Lasagne [12], a lightweight library to build and train neural networks in Theano [13]. Then, we extensively tested the network on challenging urban scene parsing datasets such as Camvid [3], Daimler [14] and the recently released Cityscapes [4] showing comparable results to the state-of-the-art of convolutional models.

- In the next chapter we give a theoretical background on the Deep Learning models focusing on the most used artificial neural architecture.
- In Chapter 3 we explain the task of Semantic Segmentation and discuss the issues that are involved. Moreover we give an overview of the datasets and the state of the art methods used for urban scene understanding.
- In Chapter 4 we present the ReSeg model and we describe in details each component of the architecture.
- In Chapter 5 we describe the design of the experiments and we show the experimental results that we obtained.
- Finally, in the last chapter, we resume our work and discuss some possible future developments and improvements.

Chapter 2

Deep Learning Models

This chapter gives a theoretical background on the main deep neural architectures. After introducing classic Feedforward Neural Networks we discuss more complicated models called Convolutional Neural Networks that are widely used in Computer Vision tasks. Then we focus on Recurrent Neural Networks that are the main topic of this thesis.

2.1 Introduction

Machine Learning is a field of research in Artificial Intelligence which studies how to build algorithms that are able to learn from experience how to perform a specific task without being explicitly programmed. Usually this kind of tasks imply some degree of intelligence and are too difficult to solve with fixed programs designed by humans. Machine Learning tries to tackle these problems by building a mathematical model from examples input in order to make *data-driven* predictions or decisions and solve the given task. Let's consider for example the task of recognizing a visual concept, this is relatively trivial for a human being to perform because he learned how to do it observing a huge amount of examples since his early days of life. Machine Learning mimics the human process of *Learning* in order to give to the computer the ability to perform complex tasks by means of experiences and observations.

Depending on the kind of experience that is available during the learning process, Machine Learning algorithms are typically classified into three broad categories:

Supervised learning: each observation is a tuple $< \mathbf{x}, y >$ which contains the input vector of features $\mathbf{x}$ and the associated *target* or *label* y. The goal is to learn a general rule that maps input into output. The term supervised learning originates from the view of the target y as being provided by an "instructor" or "teacher" who shows the machine learning system what to do.

Unsupervised learning: no labels are associated to the input features. Unsupervised learning aims to discover hidden patterns in data dividing the dataset into clusters of similar examples.

Reinforcement learning: no input and/or output data are provided and the reinforcement learning algorithm interacts with the environment so there is a feedback loop between the learning system and its experiences. The goal is to learn which actions to perform in the given state of the environment in order to maximize a numerical reward signal. The learning algorithm is not told which actions are better to take in some situation and it must discover which actions yield the most reward by trying them.

Unsupervised and Reinforcement learning algorithms are out of the scope of this work. In this thesis we will focus only on the supervised learning paradigm, specifically on a family of Machine Learning models called *Artificial Neural Networks.*

Artificial Neural Networks, or more simply *Neural Networks*, are nonlinear models that are used for function approximation. Given a generic function $y = f(x)$ the goal of a neural network is to estimate the best parameters θ which approximate the function minimizing a proper performance index which depends on the task to be performed. The performance function evaluates the *error* between the target value $y = f(x)$ and the approximation given by the model $f^*(x; \theta)$.

Depending on the task to be performed, the algorithm learns a specific function. For example in a **Classification** task the computer program is asked to specify among k categories which one the input belongs to. In this case the learning algorithm has to produce a function $f : \mathbb{R}^n \to \{1, \cdots, k\}$ that maps the input space of features into the discrete output space of the classes. Other tasks usually require to predict a numerical value performing a **Regression**, so the input output mapping in this case will be a vector to scalar function $f : \mathbb{R}^n \to \mathbb{R}$.

2.2 Feedforward Neural Networks

Before talking about more complicated and specialized models such as convolutional and recurrent neural networks, it is useful to introduce and briefly describe the basics of artificial neural networks.

The research area of Neural Networks has originally been inspired by the goal of modeling biological neural systems, however modern Neural Network research is currently guided by mathematics, statistics and engineering disciplines and its goal is not to model the brain anymore. As stated before, it is better to think of neural networks as function approximation tools designed to achieve statistical generalization, sometimes inspired by neuroscience rather than as models of a brain function.

The artificial neuron has been presented in 1957 by Rosenblatt with the name of *Perceptron* [15] as a supervised binary classifier that learns whether an input belongs to one class or another. Although the Perceptron initially seemed promising, it was quickly proved that could't be trained to recognize many classes of patterns. In a famous book entitled *"Perceptrons"* published in 1969, Marvin Minsky and Seymour Papert showed that it is impossible for these kind of models to learn a XOR function and they are only capable of learning linearly separable patterns.

This led to the field of neural network research stagnating for many years, before it was recognised that more layers of perceptron can be stacked together in a *feedforward neural network* (also called a *Multilayer Perceptron*) and trained with the backpropagation algorithm [16]. These models are called *feedforward* because their topology is defined so that information flows from x, through the intermediate computations used to define f, and finally to the output y. There are no *feedback connections* in which function being evaluated from x, through the intermediate computations used to output of the model are fed back into itself.

The name *networks* comes from the fact that their representation is typically given by a the composition of many different functions and the model can be described by a directed acyclic graph (DAG) which represents how the functions are composed together. Neural Network models are organized into distinct *layers* of many *units* that act in parallel representing a vector to scalar function. Each unit resembles a neuron in the sense that it receives input from many other units and

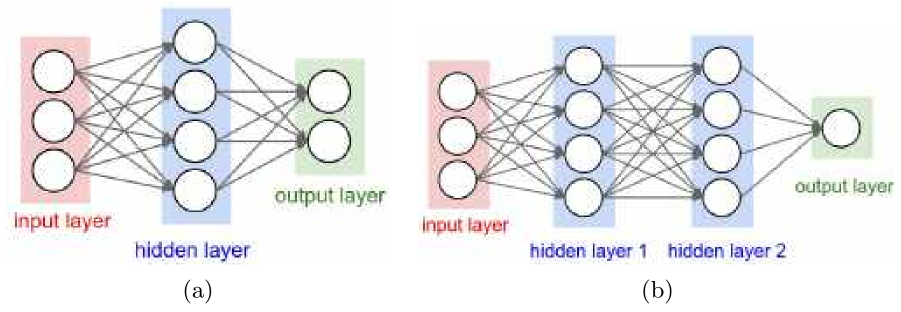


Figure 2.1: An example of a Feedforward fully connected neural network with one and two hidden layers

computes its own activation value. Let's take for example three functions $f^{(1)}, f^{(2)}, f^{(3)}$ connected in a chain form $f(\mathbf{x}) = f^{(3)}(f^{(2)}(f^{(1)}(\mathbf{x})))$. Each function is a *layer* of the model and the overall length of the chain gives the *depth* of the model.

Given a vector of features $\mathbf{x} \in \mathbb{R}^n$ as input, a neural network transforms it through a series of hidden layers into a new representation that is more suitable for the task to be performed by the system. Each example of the training dataset is given by a pair $(\mathbf{x}_i, \mathbf{y}_i)$, specifying the behavior of the output layer associated to each input sample. However the behavior of the other layers is not directly specified by the training data. The learning algorithm must decide how to use the intermediate layers to produce the desired output but the training data does not say anything on what each layer should do.

Since the training data does not show the desired output for each of the intermediate layers, these are called *hidden layers*. The term "*deep learning*" comes from this procedure of stacking many layers together and train them jointly. A key point in designing deep architectures is the choice of the topology to connect these layers to each other.

A typical fully connected neural network layer is described by a linear transformation via a matrix $\mathbf{W}$ and every input unit is connected to every unit of the next layer. However, many specialized networks have fewer connections, so that each unit in the layer is connected to only a small subset of units in the next layer. Reducing the number of connections reduces the number of parameters, the amount of computation required to evaluate the network, and also the risk of incur in overfitting, but these type of architectures are often highly task-dependent. For example, as we will see in the next section, convolutional networks use particular patterns of sparse connections that are very effective for computer vision problems. For regular neural networks, the most common layer type is the fully connected layer (see Figure 2.1) where every neuron of each layer is connected to every neuron of the adjacent layer and there are no connections between units of the same layer. Each neuron returns a nonlinear combination of its input as follows:

$$o_j = \sigma(\sum_i w_{i,j} x_{i,j}). \tag{2.1}$$

In other words, each neuron performs a dot product with the input and its weights, adds the bias and applies the non-linearity (or activation function). The choice of the activation function $\sigma(\cdot)$ is sometimes loosely guided by neuroscientific observations of the biological neurons. This type of structure is called Fully-connected Feedforward Neural Network and became very popular after the introduction of the backpropagation learning algorithm by Rumelhart et. al [16].

Formally, a feedforward neural network with l hidden layers is parametrized by $l+1$ weight matrices $(W_0, \ldots, W_l)$ and $l+1$ bias vectors $(b_0, \ldots, b_l)$ that are trained by means of an optimization algorithm which minimizes a proper error function.

Why Deep Neural networks?

Despite its biological interpretation, an artificial neural networks is just a *universal function approximator*. More rigorously the *universal approximation theorem* states that a feedforward neural network with a single hidden layer can approximate any measurable function to any desired degree of accuracy on a compact set [17].

It means that regardless of what function we are trying to learn, a large enough MLP will be able to represent it. However, it is not guaranteed that the training algorithm will be able to learn that function. Even if the MLP is able to represent the function, the learning procedure can fail because of overfitting or because the optimization algorithm used for training may not be able to find a good value of the parameters to approximate the desired function.

Some bounds on the size of a single-layer network that would be needed to approximate a broad class of functions have been provided by Barron (1993) but unfortunately, in the worse case, an exponential number of hidden units may be required. In summary, a feedforward network with a single layer is sufficient to represent any function, but the layer may have to be infeasibly large and may fail to learn and generalize correctly.

In many cases, using deeper models can reduce the number of units required to represent the desired function and it can reduce the generalization error. Deep learning is based on the evidence that a deep, hierarchical model can be exponentially more efficient at representing some functions than a shallow one [18]. Several recent theoretical results support this hypothesis (see, e.g., [19] Delalleau and Bengio, [20] Pascanu et al., [21]); Indeed, there exist families of functions which can be approximated efficiently by an architecture with depth greater than some value d, but that would require a much larger model when depth is restricted to be less than or equal to d. In many cases, the number of hidden units required by a shallow model is exponential in n. There are several empirical evidences supporting this hypothesis (e.g., Hinton et al., [22, 23]), for instance, it has been shown by Delalleau and Bengio [20] that a deep sum-product network may require exponentially less units to represent the same function compared to a shallow sum-product network.

This suggests that using deep architectures does indeed express a useful prior over the space of functions the model learns. In fact choosing a deep model encodes a very general belief that the function we want to learn should involve composition of several simpler functions. This can be interpreted from a representation learning point of view as saying that we believe the learning problem consists of discovering a set of underlying factors of variation that can in turn be described in terms of other, simpler underlying factors of variation [24].

2.2.1 Training a neural network

Designing a feedforward network requires taking the same design decisions that are necessary for any other machine learning models. In particular we have to choose:

- The optimizer
- The cost function (and any eventual regularization term)
- The kind of hidden units
- The kind of the output units

- The activation functions
- The initialization policy of the parameters
- The topology of the network

In general, we call *hyperparameter* every choice we have to make in the design of the architecture of the network, including how many layers the network should contain (the depth), how many units should be in each layer (the width) and how these units should be connected to each other (the topology), the activation function of each hidden layer.

Gradient based learning

Once we have decided all the hyperparameters, we have to train the model. Neural networks are usually trained by using iterative, gradient-based optimization methods that try to drive the cost function to very low values. Unfortunately nonlinearities and high dimensionality cause the cost function to be highly non convex and make the optimization very hard in practice. For this reasons many variants of the classic gradient descent algorithm have been proposed. Usually these methods are improvements and refinements of the well known *stochastic gradient descent* (SGD) algorithm. The SGD algorithm is an approximation of the classic gradient descent algorithm that uses an estimate of the gradient of the loss function based only on a single example of the training set. A compromise between computing the true gradient and the gradient at a single example, is the "mini-batch" extension which computes the gradient on more than one training example at each step.

Optimization for deep networks is currently a very active area of research. New optimization methods have been proposed [25, 26], to try to ameliorate the performance of the stochastic gradient descent using adaptive learning rates and second-order curvature informations such as Quasi-Newton methods.

Initially it was commonly thought that simple gradient descent would get stucked in poor *local minima* and suboptimal weights configurations, but in practice local minima are rarely a problem with large networks. Regardless of the initial conditions, the system nearly always reaches solutions of very similar quality. Recent theoretical and empirical results [27] strongly suggest that the difficulties of training deep

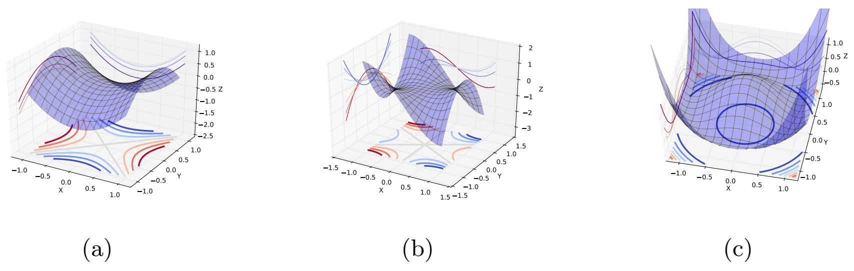


Figure 2.2: Illustrations of two different types of saddle points (a-b) plus a gutter structure (c) where the shape of the function is that of the bottom of a bottle of wine. This means that the minimum is a "ring" instead of a single point. [27]

neural networks originates from the existence of *saddle points* and not local minima. In fact, the cost function is full of saddle points where the gradient is zero, the surface curves up in most dimensions and curves down in the remainder. This dramatically slow down learning, and give the illusory impression of the existence of a local minimum. Different types of saddle points are showed in Figure 2.2.

It is also important to choose a robust initialization method for the parameters of the network in order to help the optimization algorithm to rapidly converge to a good solution. Many approaches have been proposed depending on the activation of the hidden units [28, 29].

Back-Propagation algorithm

The *backpropagation* algorithm provides an efficient and exact way of computing the gradient of the cost function in order to train a neural network.

We can think of training a neural network as two separate phases. In a feedforward neural network the information flows forward from the input layer passing through the hidden units and producing the output prediction $\hat{\mathbf{y}}$. This is called *forward propagation*. After the forward pass the cost function $J(\theta)$ is evaluated and the training algorithm iteratively updates the parameters of the network. In order to minimize the cost we need to compute the gradient of the cost function with respect to the parameters. The *backpropagation* algorithm allows the information given by the cost to *flow backwards* through the network in order to computer the gradients.

The Backpropagation algorithm is a gradient-based learning method that minimizes a proper error function looking at the direction of the gradient. The weights are learned propagating back through all the layers of the network the prediction error. At each step the algorithm updates each weight of a quantity proportional to gradient of the error function with respect to the weight. A general updating rule for the weights of the network can be written as follows:

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \frac{\partial J(\theta)}{\partial \mathbf{w}} \tag{2.2}$$

where $\mathbf{w}$ is the vector of weights, $J(\theta)$ is the error function, and α is the *learning rate.*

Since the output is given by a composition of non linear functions it is sufficient to apply several times the classic *chain rule* to compute the gradients. Let x be a real number , and two functions $f : \mathbb{R} \to \mathbb{R}$ and $g : \mathbb{R} \to \mathbb{R}$. Now consider the composite function $z = f(g(x)) = f(y)$, where $y = g(x)$. Then the derivative of f with respect to x can be computed applying the chain rule as follows:

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx} \tag{2.3}$$

Once the gradient of the cost function is computed, the gradients are used to perform the parameters update.

Cost functions

An important aspect of the design of a deep neural network is the choice of the cost function. In the case of classification task the model defines a distribution $p(\mathbf{y}|\mathbf{x};\theta)$ over a set of classes and we can use the principle of maximum likelihood to estimate the parameters. This means using the *cross-entropy* cost function to measure the error between the training data and the model's predictions.

$$J(\theta) = -\frac{1}{N}\sum_i f(x_i;\theta)\log(y_i). \tag{2.4}$$

If the output of the function are real numbers we are estimating the parameters for a regression task and the usual choice for the cost function is the classic *Mean Squared Error.*

$$J(\theta) = \frac{1}{N}\sum_i (y_i - f(x_i;\theta))^2. \tag{2.5}$$

Output units

Considering a feedforward neural network, the input passes through several hidden layers and is transformed in a new feature representation $\mathbf{h} = f(\mathbf{x}, \theta)$. The output layer provides a last additional transformation to complete the task the network must perform. The choice of the cost function is tightly coupled with the choice of the output unit.

One of the simplest kind of output is based on an affine transformation with no nonlinearity. The are often just called *linear units*. Given features $\mathbf{h}$, a layer of linear output units produces a vector $\mathbf{y} = \mathbf{W}^T\mathbf{h} + \mathbf{b}$. Linear output layers are often used to produce the mean of a conditional Gaussian distribution. In a maximum likelihood framework in fact maximizing the log-likelihood is equivalent to minimizing the mean squared error.

For binary classification tasks the classic approach is based on using *sigmoid* output units combined with maximum likelihood. We can think of the sigmoid output unit as having two components. First, it uses a linear layer to compute $z = \mathbf{w}^T\mathbf{h} + b$. Next, it uses the sigmoid activation function to convert z into a probability.

$$\hat{y} = \sigma(\mathbf{w}^T\mathbf{h} + b) \tag{2.6}$$

This comes from the fact that in the case of binary variable, we wish to produce a single number

$$\hat{y} = P(y = 1|\mathbf{x}) \tag{2.7}$$

Usually the log-likelihood $z = \log \tilde{P}(y = 1|\mathbf{x})$ is used instead, in order to avoid numerical problems due to the fact that the probabilities are in the $[0, 1]$ range. Exponentiating and normalizing we obtain a Bernoulli distribution controlled by the sigmoid function.

Any time we wish to represent a discrete probability distribution over k possible outcome, we may use the *Softmax* function. Softmax functions are used as output of a classifier to normalize in the probability range. This can be seen as a generalization of the sigmoid function which was used to represent a Bernoulli distribution. In this case we need to produce a vector $\hat{\mathbf{y}}$, with $\hat{y}_i = P(y = i|\mathbf{x})$, where each element $\hat{y}_i$ need to be between 0 and 1 and they are constraint to sum to 1 in order to represent a valid probability distribution. The same approach that work for the Bernoulli distribution generalizes to the Multinomial distribution. First, a linear layer is used to predict unnormalized log probabilities

$$\mathbf{z} = \mathbf{W}^T\mathbf{h} + \mathbf{b}, \tag{2.8}$$

where $z_i = \log \tilde{P}(y = 1|\mathbf{x})$. The softmax function can be then exponentiate and normalize $\mathbf{z}$ to obtain the desired output $\hat{\mathbf{y}}$. So formally the softmax function is given by

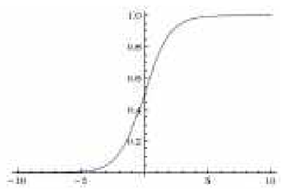
$$softmax(\mathbf{z})_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}, \tag{2.9}$$

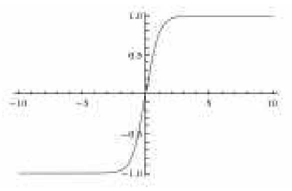
Hidden units

The design of hidden units is a very active area of research. In the last years several nonlinear activation functions have been proposed aiming to improve the performances and trying to make simpler the optimization process.

Currently the most popular activation function for neural networks is the *Rectified Linear Unit* (ReLU), which was first proposed for restricted Boltzmann machines by Nair & Hinton [30] and then successfully used for neural networks [31–33] . The ReLU activation function is linear with slope 1 for positive arguments and zero otherwise, that is, for positive values it is the identity and for negative values the zero function, $g(z) = max\{0, z\}$. Typically a rectified hidden unit is used on top of an affine transformation $\mathbf{h} = g(\mathbf{W}^T\mathbf{x} + \mathbf{b})$. The main advantage of ReLUs is that they avoid the *vanishing gradient problem* [34] since their derivative is 1 for positive values and 0 elsewhere. A detailed description of the vanishing gradient problem is given in section 2.4.1.

Several generalizations of rectified linear units exist, most of these perform comparably to rectified linear units and occasionally perform better. Recently *Leaky ReLUs* (LReLUs) have been proposed, which replace the negative part of the ReLU with a linear function fixing the slope α_i to a small value like 0.01 [35]. Leaky ReLUs have been shown to be superior to ReLUs in terms of learning speed and performance [36]. They have later been generalized to *Parametric Rectified Linear Units* (PReLUs) [29] which treat the slope of the negative part α_i as a learnable parameter. Another version of leaky ReLUs are *Randomized Leaky Rectified Linear Units* (RReLUs), where the slope of the negative part is randomly sampled [36]. Recently also an Exponential Linear Units (ELU) have been presented [37], the hyperparameter *alpha* controls the value to which an ELU saturate for negative net input.


(a) Sigmoid unit


(b) Tanh unit

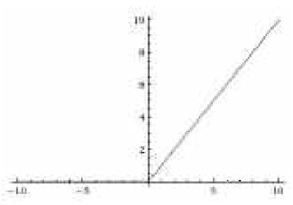

(c) ReLu unit

Figure 2.3: Illustrations of different types of activation functions: Sigmoid non-linearity (a) squashes real numbers to range between $[0, 1]$. *Tanh non-linearity (b) squashes real numbers to range between* $[-1, 1]$. *Rectified Linear Unit (ReLU) activation function is zero when* $x < 0$ *and then linear with slope* 1 *when* $x > 0$

.

Before the introduction of rectified linear units, most neural networks used the *logistic* sigmoid activation function $g(z) = \sigma(z)$ or the *hyperbolic tangent* activation function $g(z) = \tanh(z)$. These two activations are closely related by the fact $tanh(z) = 2\sigma(2z) - 1$. Unlike linear units, sigmoidal units saturate to a high value when their input is very positive and saturate to a low value when the input is very negative. This makes them operative only in a very narrow range around zero making the gradient-based learning very hard. For this reason they have been substituted by other kind of hidden units in feedforward networks. Nowadays they are mostly used as output units or in other settings like in Recurrent Neural Networks or autoencoders.

More specialized types of hidden units which address several issues in Recurrent Neural Networks are discussed in the sections below.

2.3 Convolutional Neural Networks

Fully-connected neural network showed to work well in a lot of fields but they are not really used in practice for image recognition tasks. To understand the motivation behind that, let's take as example an image of size $32 \times 32 \times 3$, a single fully connected neuron in the first layer would have $32 * 32 * 3 = 3072$ weights. Clearly this kind of structure doesn't scale well with larger images.

Using this kind of network for image classification causes the number of parameters to rapidly explode making hard to train the network. Considering a bigger image with a more common size e.g. $640 \times 480 \times 3$, this would lead to neurons that have $640 * 480 * 3 = 921,600$ weights.

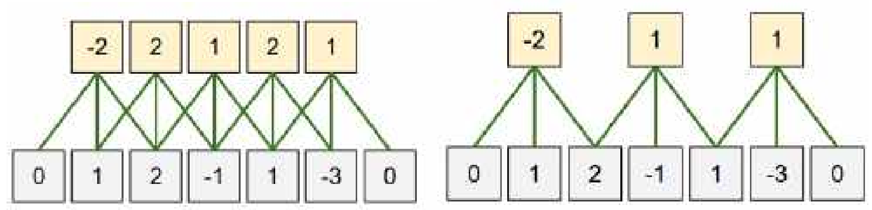
Clearly this kind of full connectivity is wasteful and the huge number of parameters would quickly lead to overfitting.

Besides these important facts there is another reason that motivates the use of a different model for image-based tasks: a fully-connected neural network does not take into account the inherent spatial structure of an image, it treats input pixels which are far apart and close together in exactly the same way.

Convolutional Neural Networks (CNNs or ConvNets) use a special architecture that exploits the spatial structure of the images. More in general these kind of neural networks are specialized in processing data that has a grid-like topology such as time-series (1D grid) or images (2D grid) [24]. In particular, a Convolutional network is a neural network that instead of using a matrix multiplication uses convolution operation to compute the activation of a layer.

The use of convolution is motivated by the fact that it leverages three key ideas: *sparse interaction*, *parameter sharing* and *equivariant representations*. In a fully-connected neural network we have a full interaction between the units of each layer, which means that each input pixel is connected to each hidden unit of the adjacent layer. Convolutional Neural Networks use *sparse interaction* (or *sparse connectivity*), which means that each neuron in a hidden layer is connected only to small localized regions of the previous layer. These regions are called *receptive fields*. The space between each receptive field is called *stride*. Choosing a stride equal to 1 the receptive fields are side by side but we can also have overlapped receptive field with a stride less than 1, or we can skip some input units choosing a stride greater than 1. The latter case correspond to perform a downsample operation. See the Figure 2.4 for a visual representation.

The second important property is *parameter sharing*, it refers to the fact that the network uses the same weights and bias for each location of its input it is applied to. We can also say that the network has *tied weights*, because the value of a weight applied to one location is tied to the value of a weight applied elsewhere. In practice rather than learning a separate set of parameters for every region, we learn only one set. Parameter sharing is used in convolutional layers to control the number of parameters, in fact, it is possible to reduce the number of parameters making the reasonable assumption that one patch feature is useful no matter what is its location in the image. Thanks to parameter sharing



(a) Stride = 1 (b) Stride = 2

Figure 2.4: Examples of two kind of sparse connectivity in 1-dimension: each output unit is affected by a subset of the hidden units. We can also interpret interaction in the opposite sense where each hidden unit affect only a subset of the output units. The neuron weights in this example are [1,0,-1] and the bias is zero. These weights are shared across all yellow neurons using the parameter sharing property.

different neurons share the same weights, this allows the output of a convolutional layer to be implemented as a convolution of the weights with the input. The map from the input layer to the hidden layer is called *feature map*, and the shared weights and bias are often referred as *kernel* or *filter*.

The network structure described so far can detect just a single kind of localized feature, but, in order to use it for tasks such as image recognition, it is necessary to detect more than one feature. Every layer of a ConvNet transforms an input volume into an output volume of neuron activations where each slice of the output volume is a feature map. The structure is depicted in Figure 2.5.

In the case of convolution, parameter sharing causes the layer to have an important property for images that is called *equivariance* to translation. In particular we say that a function $f(x)$ is equivariant to a function $g(\cdot)$ if $f(g(x)) = g(f(x))$. The convolution operator is equivariant to any function $g(\cdot)$ that translates the input. For example, let $\mathbf{I}$ be a function giving the intensity of each pixel of an image, and let $\mathbf{I}' = g(\mathbf{I})$ a function mapping one image function to another image function such that $\mathbf{I}'(x, y) = \mathbf{I}(x - 1, y)$ that shift every pixel of $\mathbf{I}$ one unit to the right. Applying this transformation to $\mathbf{I}$ and then apply the convolution will produce the same result as applying the convolution to $\mathbf{I}$ and then the transformation $g(\cdot)$ to the output of the convolution.

The intuition behind equivariance to translation is that the convolution creates a map of where certain features appear in the input. Moving the object in the input, its representation will move the same

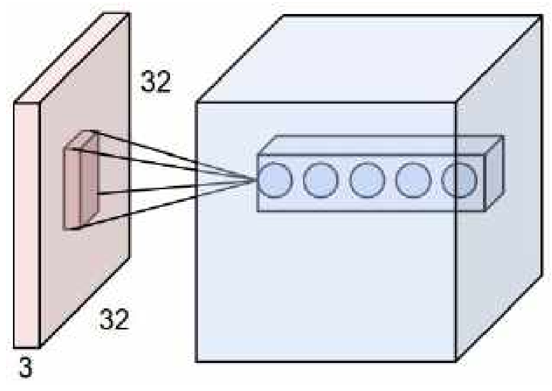


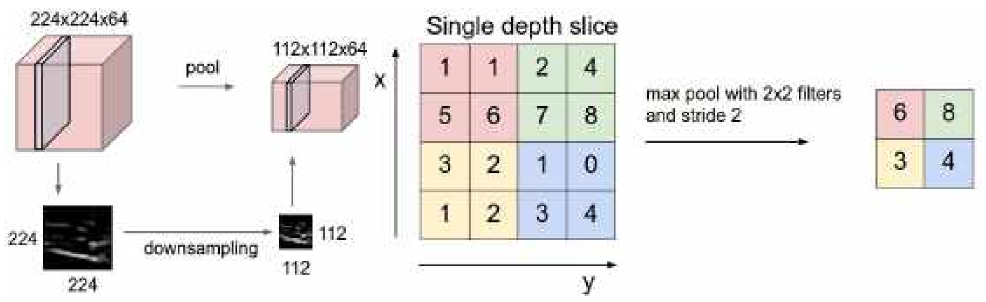
Figure 2.5: In red we can see an example input image 32x32x3, and an example volume of neurons in the first Convolutional layer in blue. Each neuron in the convolutional layer is connected only to a local patch in the input. The neurons of the Neural Network remain unchanged, they still compute a dot product of their weights with the input followed by a non-linearity, but their connectivity is now restricted to be local spatially.

amount in the output. This is important for tasks such as image recognition where the same local feature is useful everywhere in the input. For example, it is useful that the network learns a robust edge detector because the same edges appear more or less everywhere in the image. However, we can have some cases in which we may not wish to share parameters across the entire image. For example, in face recognition tasks we want to extract different features at different locations of the image, and the part of the network processing the top of the face needs to look for the eyes, while the part of the network processing the bottom of the face needs to look for the mouth etc.

Convolution is not naturally equivariant to some other transformations, such as changes in the scale or rotation of an image. Other mechanisms are necessary for handling these kinds of transformations.

Pooling

A typical ConvNet architecture consists in a pipeline of three operations: *convolution*, a *nonlinear transformation* and *pooling*. First of all the ConvNet computes several parallel convolutions to produce a set of feature maps or pre-activations. In the second stage a nonlinear activation function, such as the rectified linear activation (ReLU) is applied to the feature maps. At last it is common to insert a *pooling layer* in-between successive convolutional layers in a ConvNet architecture.



(a) Downsampling (b) Max-pooling

Figure 2.6: A pooling layer downsamples the volume spatially, independently in each depth slice of the input volume. In this example, the input volume of size [224x224x64] is pooled with filter size 2, stride 2 into output volume of size [112x112x64]. Notice that the volume depth is preserved. Figure (b) shows a 2x max pooling operation where each max is taken over 4 numbers on a patch of dimension 2x2.

The important fact behind the use of pooling is that helps to make the representation invariant to small translation of the input. To provide invariance to local translation it also has the effect of progressively reducing the spatial size of the representation to reduce the amount of parameters and computation in the network, and hence to also control overfitting. The pooling layer operates on every slice of the convolution output and resizes it spatially, using an aggregate operation like the Max function as shown in Figure 2.6. In addition to max pooling, the pooling units can also perform other functions, such as average pooling or L2-norm pooling. Invariance to local translation is very useful when we do not need to know the exact position of a feature, but we just need to know that is in a given region. For example, when determining whether an image contains a face, we do not need to know the location of the eyes with pixel-perfect accuracy, we just need to know that there is an eye on the left side of the face and an eye on the right side.

The use of pooling can be also viewed as adding an infinitely strong prior that the function that the layer must learns is invariant to small translations [24]. The correctness of this assumption can greatly improve the statistical efficiency of the network.

A *prior* is a probability distribution over the parameters of a model that encodes our beliefs about the reasonable models before having seen any data. Priors can be weak or strong depending on how concentrated the probability density is. A weak prior is a distribution with high variance, which allows the parameters to vary more or less freely. A density probability with low variance is instead a prior that imposes some strong constraints on the parameters. An infinitely strong prior places zero probability on some parameters and says that these parameters value are completely forbidden, regardless of how much support the data gives to those values.

Implementing a convolutional network as a fully connected network with an infinitely strong prior would be an extreme waste of computation, but this analogy is useful to give us some insights into how convolutional nets work.

2.4 Recurrent Neural Networks

Convolutional Neural Networks proved their ability to learn a general hierarchical feature representation in many contexts and are currently the first choice for many computer vision tasks such as Image Recognition [38, 39], Object Detection or Video Analysis [40]. This is possible thanks to the inherent grid structure of the input data that can be efficiently exploited by these networks through a series of layers of learnable filters.

It often occurs that data examples share a form of sequentiality that have to be considered. A natural choice when we have to handle sequential data is to use *Recurrent Neural networks.* Several sequential machine learning tasks can be taken into account using RNNs: Image Captioning [41], Speech Synthesis and Recognition [42–45], Music Generation and Transcription [46, 47], time-series prediction, and Natural Language Translation [48]. All of these tasks have in common a translation from a signal to another by processing sequential data.

Recurrent neural networks are an extension of feedforward neural networks, augmented with the ability to pass information through recurrent edges that span adjacent samples, introducing a notion of sequentiality to the model. Adding cycles to the network means adding dynamics to the system and enhance the expressive power of the model. The structure of a recurrent neural network is very similar to that of the standard multilayer perceptron, where we add *feedback connections* to the hidden units. Through these connections the model can retain information about the past, enabling it to discover correlations between events that are far away from each other in the data.

These dependencies generate a *context* that cannot be taken into account by classical neural networks because of their simple structure. Feedforward neural networks are in fact memoryless models that rely on the assumption of independence among the data points. Recurrent Neural Networks are feed-back neural networks naturally designed for modeling contextual dependencies. Because of the connections from the previous states to the current ones RNNs have memory. Through such feedback connections, RNNs are able to retain information of the past input, and to discover correlations among the input data that might be far away from each other in the sequence. The concept of context is very important in many fields and it is the reason why Recurrent Neural Networks are preferred when the instances of input data are not independent from each other, but rather they are correlated to the nearby ones.

We can formalize a simple recurrent neural network with one hidden layer exactly how we did for the feedforward neural network. Although we consider the notion of time to define recurrent networks, we have to notice that we can refer to any kind of data sequentiality (e.g., spatial). The general formulation of an RNN at time step t is a function of the state at time $t-1$ and the current input $\mathbf{u}_t$:

$$\mathbf{x}_t = F(\mathbf{x}_{t-1}, \mathbf{u}_t; \theta) \tag{2.10}$$

Specifically we can use the following parametrization:

$$\mathbf{x}_t = \sigma(\mathbf{W}_{rec}\mathbf{x}_{t-1} + \mathbf{W}_{in}\mathbf{u}_t + \mathbf{b}), \tag{2.11}$$

where the parameters of the model are the recurrent weight matrix $\mathbf{W}_{rec}$, the bias $\mathbf{b}$ and the input weight matrix $\mathbf{W}_{in}$, collected in θ for the general case in showed in Equation 2.10. The initial state $\mathbf{x}_0$ can be provided, set to zero or learned, and σ is an element-wise function.

Feedforward neural networks are universal function approximators [17] but RNNs are way more powerful. Siegelman and Sontag [49] proved that a finite sized recurrent neural network with sigmoidal activation functions can simulate a universal Turing machine. Specifically, any function computable by a Turing machine can be computed by a recurrent network of a finite size, in other words an RNN is Turing-complete.

A simple way to visualize the dynamics of a recurrent neural network across time steps is to unfold the network. The model can be

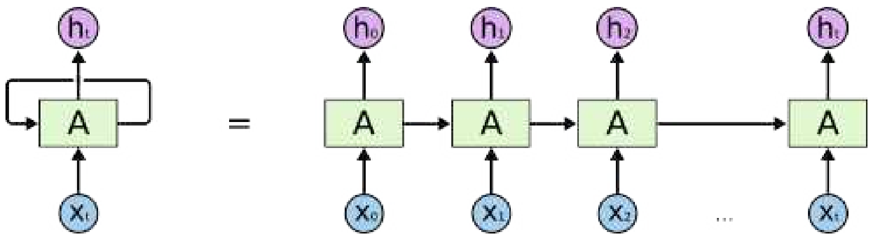


Figure 2.7: An unrolled recurrent neural network.

interpreted not as cyclic, but rather as a deep feedforward network with one layer per time step and tied weights across time steps. The unfolded network can be trained across many time steps using the backpropagation algorithm presented with the name of *backpropagation through time* (BPTT), in 1990 by Werbos [50].

Unfolding RNNs takes advantage of the concept of *parameter sharing*, in fact the weights are shared across different instances of the artificial neurons, each associated with different time steps. This allows to generalize to sequences of different lengths because the same weights are re-used at each time step. The core idea is that it is not important the absolute time step at which an event occurs, but it only makes sense to consider the event in some context that somehow captures what happened before. Once unfolded, recurrent neural networks can be trained end-to-end with backpropagation considering them like deep feedforward neural networks with an unbounded number of layers. However training a deep network is not an easy task because of two problems that occur during the learning process: the *vanishing* and *exploding gradient problems.*

2.4.1 Vanishing and exploding gradient problems

The main reason that makes RNNs so appealing is the fact that they can use their feedback connections to learn dependencies between data and connect input events at different times.

In a classical sequential learning task we often only need to look at recent information to perform the present prediction. For example, consider a language model trying to predict the next word based on the previous ones, RNNs can be train to use the past information to solve this task. When the distance of the relevant information needed for the prediction is small, RNNs can learn to use the past information. But there are also cases where we need more context. Unfortunately, as that gap grows, RNNs become unable to learn how to connect past and present informations. Theoretically, thank to the internal state, the network can remember things for a long time. In principle, the internal state can carry information about a potentially unbounded number of previous input, but in practice training a recurrent network to properly store information that's not needed for a long time can be especially hard.

RNNs are capable of handling such *"long-term dependencies"* but are not able to learn the right weight parameters with conventional Back-Propagation Through Time (BPTT) [50] as explored by Hochreiter in [51] and then extensively studied by Bengio and Frasconi in [34] analyzing the problem of vanishing and exploding gradients. The Back-Propagation Through Time algorithm is applied to a very deep unfolded version of the recurrent network. This means that the error needs to be back propagated trough many layers. Due to the application of the chain rule, the gradient of the error function is given by the multiplication of several terms and this can cause the gradient to rapidly vanish to zero or even to explode making impossible to effectively train the network.

Indeed, let's define a cost function $\mathcal{E} = \sum_{t=1}^{T} \mathcal{E}_t$ that measures the performance of the network on some given task, where $\mathcal{E}_t = loss(\mathbf{x}_t)$. Following [52], we can explicitly compute the gradient of the cost function to highlight the problems that occur during the training:

$$\frac{\partial \mathcal{E}}{\partial \theta} = \sum_{t=1}^{T} \frac{\partial \mathcal{E}_t}{\partial \theta} \tag{2.12}$$

$$\frac{\partial \mathcal{E}_t}{\partial \theta} = \sum_{k=1}^{t} \left(\frac{\partial \mathcal{E}_t}{\partial \mathbf{x}_t} \frac{\partial \mathbf{x}_t}{\partial \mathbf{x}_k} \frac{\partial \mathbf{x}_k^+}{\partial \theta} \right) \tag{2.13}$$

Equation 2.12 consists of unfolding the RNN and summing the gradients of the errors at each time step. Any gradient component $\frac{\partial \mathcal{E}_t}{\partial \theta}$ is also a sum of several *temporal* components. Each of these *temporal* contributions $\frac{\partial \mathcal{E}_t}{\partial \mathbf{x}_t} \frac{\partial \mathbf{x}_t}{\partial \mathbf{x}_k} \frac{\partial \mathbf{x}_k^+}{\partial \theta}$ measures how θ at step k affects the cost at step $t > k$. $\frac{\partial \mathbf{x}_k^+}{\partial \theta}$ refers to the *immediate* partial derivative of the state $\mathbf{x}_k$ with respect to θ where $\mathbf{x}_{k-1}$ is taken as a constant with respect to θ.

The factors $\frac{\partial x_t}{\partial x_k}$ in Equation 2.13 transport the error "in time" from step t back to step k. Considering now the generic parametrization of the recurrent neural network in Equation 2.11, the value of any row i of the matrix $\left(\frac{\partial \mathbf{x}_k^+}{\partial \mathbf{W}_{rec}}\right)$ is just $\sigma(\mathbf{x}_{k-1})$

$$\frac{\partial \mathbf{x}_t}{\partial \mathbf{x}_k} = \prod_{i=k+1}^{t} \frac{\partial \mathbf{x}_i}{\partial \mathbf{x}_{i-1}} = \prod_{i=k+1}^{t} \mathbf{W}_{rec}^T diag(\sigma'(\mathbf{x}_{i-1})) \qquad (2.14)$$

Equation 2.14 provides the form of Jacobian matrix $\frac{\partial \mathbf{x}_i}{\partial i-1}$ for the parametrization in Equation 2.11 where $\sigma'(\cdot)$ computes element-wise the derivative of $\sigma(\cdot)$. From now on we will also distinguish between *long term* and *short term* contributions, where long term refers to components for which $k >> t$ and short term to everything else.

We can interpret the vanishing gradient behavior as the error signal from later time steps that cannot go enough back in time to influence the network at earlier time steps. This makes it difficult to learn long-term dependencies making it impossible for the model to learn correlation between temporally distant events.

In order to understand why this phenomenon occurs during the training we need to look at the form of each temporal component, and in particular at the matrix factors $\frac{\partial \mathbf{x}_t}{\partial \mathbf{x}_k}$ that take the form of a product of $t-k$ Jacobian matrices in Equation 2.14.

Just like a product of $t-k$ real numbers can shrink to zero or explode to infinity, so does a product of matrices (along some direction v). Long term components can both grow exponentially or go fast to norm 0 and so do the gradients, making it impossible for the model to learn correlation between temporally distant events. Now we are going to formalize these intuitions.

Consider at first a linear version of model setting $\sigma(\cdot)$ to the identity function in Equation 2.11. We can use the *power iteration method* to formally analyze the product of Jacobians in Equation 2.14 and obtain conditions for when the gradients explode or vanish.

We define the *spectral radius* ρ of a square matrix as the supremum among the absolute values of the elements in its spectrum, formally, let $\lambda_1, \ldots, \lambda_n$ be the (real or complex) eigenvalues of a matrix $\mathbf{A} \in \mathbb{C}^{n\times n}$, then its spectral radius is defined as

$$\rho(\mathbf{A}) = max\{|\lambda_1|, \ldots, |\lambda_n|\}$$

The spectral radius is closely related to the behavior of the convergence of the power sequence of a matrix and the following theorem holds:

Theorem. Let $\mathbf{A} \in \mathbb{C}^{n\times n}$ with spectral radius $\rho(\mathbf{A})$ then $\rho(\mathbf{A}) < 1$ if and only if

$$\lim_{k\to\infty} \mathbf{A}^k = 0.$$

Moreover, if $\rho(\mathbf{A}) > 1$, $||\mathbf{A}^k||$ is not bounded for increasing values of k. Considering now the *spectral radius* ρ of the recurrent matrix $\mathbf{W}_{rec}$, it is *sufficient* for $\rho < 1$ for long term components to vanish (as $t \to \infty$) and *necessary* for $\rho > 1$ for them to explode.

Pascanu et. Al [52] generalize this result for nonlinear functions σ where $|\sigma'(x)|$ is bounded, $||diag(\sigma'(x_k))|| \leq \gamma \in \mathbb{R}$ by relying on singular values. According to the parametrization of Equation 2.11, let λ_1 the largest singular value of $\mathbf{W}_{rec}$, they prove that it is *sufficient* for $\lambda_1 \leq \frac{1}{\gamma}$ for the *vanishing gradient* problem to occur.
The Jacobian matrix $\frac{\partial \mathbf{x}_{k+1}}{\partial \mathbf{x}_k}$ is given by $\mathbf{W}_{rec}^T diag(\sigma'(\mathbf{x}_k))$. The 2-norm of a product of matrices is bounded by the product of the norm of the matrices.

$$\forall k, \left\| \frac{\partial \mathbf{x}_{k+1}}{\partial \mathbf{x}_k} \right\| \leq \left\| \mathbf{W}_{rec}^T \right\| \|diag(\sigma'(\mathbf{x}_k))\| < \frac{1}{\gamma} < 1 \tag{2.15}$$

Let $\eta \in \mathbb{R}$ be such that $\forall k, \left\| \frac{\partial \mathbf{x}_{k+1}}{\partial \mathbf{x}_k} \right\| \leq \eta < 1$. The existence of η is given by Equation 2.15. By induction over i, we can show that

$$\left\| \frac{\partial \mathcal{E}}{\mathbf{x}_t} \left(\prod_{i=k}^{t-1} \frac{\partial \mathbf{x}_{i+1}}{\partial \mathbf{x}_i} \right) \right\| \leq \eta^{t-k} \left\| \frac{\partial \mathcal{E}_t}{\partial \mathbf{x}_t} \right\| \tag{2.16}$$

According to the previous equation it follows that if $\eta < 1$, long term contributions (for which $t-k$ is large) go to 0 exponentially fast with $t-k$, while the necessary condition for *exploding gradient* problem is given by the inverse, that is $\lambda_1 > \frac{1}{\gamma}$.

Usually the non linear function used in a neural network are the sigmoid or the hyperbolic tangent and their derivatives are respectively in the range of $(-1,-1)$ and $(0,1)$. So computing the gradients with respect to the weights of the k-th layer consists of multiplying k derivatives of values less than 1, meaning that the gradient (error signal) will decrease exponentially with k.

In the following sections we will describe in details two elegant architecture to address the vanishing gradient problem.

2.4.2 Long Short-term memory (LSTM)

Long-Short Term Memory Network (LSTM) [53] has been introduced by Hochreiter and Schmidhuber to overcome the problem of vanishing gradients in recurrent neural networks. This model resembles a standard neural network with a recurrent hidden layer, in which each node in the hidden layer is replaced with a *memory cell*.

The LSTM architecture consist of a memory cell c_t that can maintain its state over time, and some non-linear gating units which regulate the information flow into and out of the cell. The key feature of LSTM nodes is their ability to add or remove information to the cell state by means of these simple structures called gates. The memory cell contains a node with a self-connected recurrent edge of weight 1, ensuring that the gradient can pass across many time steps without vanishing or exploding, while the gates control the amount of changes to and exposure of the memory content. They are composed of a sigmoid neural net layer and a pointwise multiplication operation. The sigmoid layer output are real numbers in the range of zero and one, describing how much of each component should be let through. We can think of the gates as *"taps"* that can be nonlinearly closed or opened to control the flow of information through the memory cell. A value of the gate of zero means no information is let through, while a value of one means that all the information is let through.

During the years many variants of the original LSTM structure have been proposed. For the explanation we refer to the recent state of the art LSTM architecture proposed by Zaremba et al. [54] in 2014.

Now we describe in details each component of the LSTM memory cell depicted in Figure 2.8:

Input Node: this node, labeled c_j^t, behaves as a classical neuron, taking activation from the rest of the hidden layer at the previous time step as well as from the example $\mathbf{x}$. Typically, the linear input is run through an *hyperbolic tangent* activation function.

$$\tilde{\mathbf{c}}_t = \tanh(W_c\mathbf{x}_t + U_c\mathbf{h}_{t-1}), \tag{2.17}$$

Input Gate: it is a sigmoidal unit $\sigma(\cdot)$ that, like the *input node*, takes activation from the hidden layer at the previous time step and also from the example $\mathbf{x}$. The *input gate* $\mathbf{i}_t$ is so named because its output value is multiplied by that of the *input node*. It regulates how

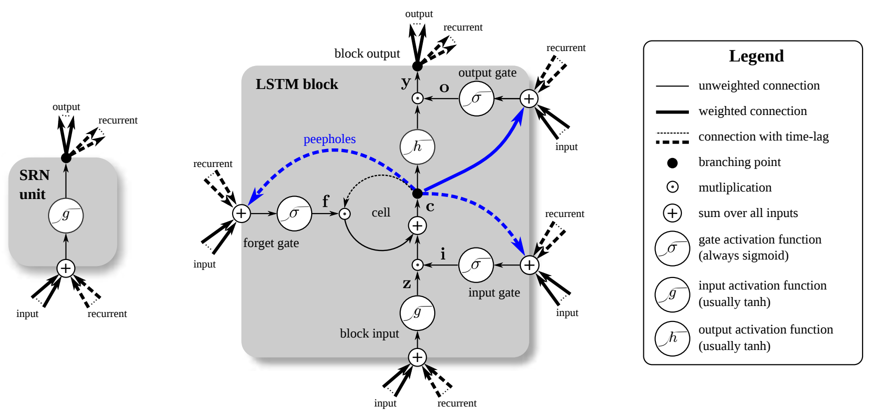


Figure 2.8: Detailed schematic of the Simple Recurrent Network (SRN) unit (left) and a Long Short-Term Memory cell (right).

much of the activation of the input node passes through the gate and can go into the memory cell.

$$\mathbf{i}_t = \sigma(W_i \mathbf{x}_t + U_i \mathbf{h}_{t-1}), \tag{2.18}$$

Internal State: the core of each memory cell is a node c_t with linear activation, which is referred to as *internal state* of the cell. It has a self-connected recurrent edge with weight 1 that is called *constant error carousel* (CEC). This edge spans adjacent time steps with constant weight, assuring that error can flow across time steps without vanishing.

$$\mathbf{c}_t = \tilde{\mathbf{c}}_t \odot \mathbf{i}_t + \mathbf{c}_t, \tag{2.19}$$

where $\odot$ stands for element-wise multiplication.

Forget Gate: with this gate we can feature the network with the ability to learn when it is time to flush the contents of the internal state and clean the memory cell. This is especially useful in continuously running networks.

$$\mathbf{f}_t = \sigma(W_f \mathbf{x}_t + U_f \mathbf{h}_{t-1}). \tag{2.20}$$

With forget gates f_t, the equation to calculate the internal state on the forward pass becomes:

$$\mathbf{c}_t = \tilde{\mathbf{c}}_t \odot \mathbf{i}_t + \mathbf{c}_t \odot \mathbf{f}_t \tag{2.21}$$

Output gate: the output of the internal state is regulated by a last *output gate*, which we denote o_c. As usual the gate is multiplied by the value of the internal state s_c to produce the value v_c of the final output of the memory cell. The internal state can optional pass through a nonlinear unit such as *tanh* before to be multiplied by the gate.

$$\mathbf{o}_t = \sigma(W_o\mathbf{x}_t + U_o\mathbf{h}_{t-1}), \tag{2.22}$$

$$\mathbf{h}_t = \tanh(\mathbf{c}_t) \odot \mathbf{o}_t. \tag{2.23}$$

Forget gates, described above, weren't in the original version of LSTM architecture and were proposed by Gers et al. in 2000 [55]. However, they have proven to be effective and are now standard in most modern implementations.

In traditional LSTM each gate receives connections from the input units and the output of all cells, but there is no direct connection from the Constant Error Carousel it is supposed to control. All it can observe directly is the cell output, which is close to zero as long as the output gate is closed. The same problem occurs for multiple cells in a memory block: when the output gate is closed none of the gates has access to the CECs they control. This results in lack of essential information degrading the performance of the network, especially for tasks that require the accurate measurement or generation of time intervals such as Rhythm detection. To overcome this problem, Gers and Schmidhuber introduced *peephole connections* [56], which are connections from the CEC to the gates of the same memory block. Peephole connections allow all gates to inspect the current cell state even when the output gate is closed. They report that these connections improve performance on timing tasks where the network must learn to measure precise intervals between events.

As mentioned before, there are many variants of LSTM architectures and they use different activation functions for each layer. For example latests state of the art use hyperbolic tangent for the input node, but in the original LSTM paper [53], the activation function for g was the sigmoid σ.

We can see the intuition behind the mechanism of the LSTM analyzing both the forward pass and the backward pass. In terms of the forward pass, the LSTM can learn when to let activation flow into the internal state. As long as the input gate takes value 0, no activation can get in. Similarly, the output gate learns when to let the value out. When both gates are closed, the activation is preserved in the LSTM, neither growing nor shrinking, nor affecting the output at the intermediary time steps. In other words each memory cell learns when to memorize and when to reset from the very same data used to learn the model.

In terms of the backwards pass, the *constant error carousel* enables the gradient to propagate back across many time steps, neither exploding nor vanishing. In this sense, the gates are learning when to let error in, and when to let it out.

Since LSTMs effectively capture long-term temporal dependencies without suffering of the same problems which affect simple recurrent networks they have been used to advance the state of the art for many difficult problems. This includes handwriting recognition [57–59] and generation [60], language modeling [54] and translation [61], speech acoustic modeling [62], speech synthesis [63], protein secondary structure prediction [1], audio and video analysis [64, 65] among others.

2.4.3 Gated Recurrent Unit (GRU)

In a LSTM architecture the input and forget gates control how much of the new information should be memorized in the memory cell and how much of the old information should be forgotten by it. These gates are computed from the previous hidden states and the current input:

$$\mathbf{i}_t = \sigma(\mathbf{W}_i\mathbf{x}_t + \mathbf{U}_i\mathbf{h}_{t-1}), \tag{2.24}$$

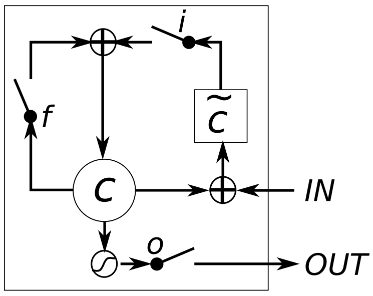
$$\mathbf{f}_t = \sigma(\mathbf{W}_f\mathbf{x}_t + \mathbf{U}_f\mathbf{h}_{t-1}). \tag{2.25}$$

The output gate $\mathbf{o}_t$ controls to which degree the memory content is exposed. In the same way of the other gates, the output gate also depends on the current input and the previous one:

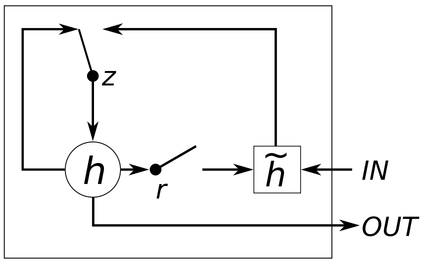
$$\mathbf{o}_t = \sigma(\mathbf{W}_o\mathbf{x}_t + \mathbf{U}_o\mathbf{h}_{t-1}). \tag{2.26}$$

This kind of structure has been the state-of-the-art for learning the long-term dependencies in a recurrent neural network for long time. Recently an interesting modification has been proposed called Gated Recurrent Unit (GRU) that meant to be a simpler and computationally less intensive version of the LSTM.

Introduced by Cho et al. in [48] GRU, like the LSTM, was designed to adaptively reset or update its memory content. Each GRU thus has a reset gate r_{jt} but it couples the input and the forget gate of



(a) Long Short-Term Memory



(b) Gate Recurrent Unit

Figure 2.9: Illustration of (a) LSTM and (b) gated recurrent units. (a) i, f and o are the input, forget and output gates, respectively. c and $\tilde{c}$ denote the memory cell and the new memory cell content. (b) r and z are the reset and update gates, and h and $\tilde{h}$ are the activation and the candidate activation. [66]

the LSTM into a single *update gate* z_{jt}. In such a way GRU balances between the previous memory content and the new one strictly using leaky integration, meaning that the updated value is computed as a weighted average of the candidate and old content of the cell. At timestep t, the state h_{jt} of the j-th GRU is computed by:

$$h_t^j = (1 - z_t^j)h_{t-1}^j + z_t^j \tilde{h}_t^j \tag{2.27}$$

where h_{t-1}^j and $\tilde{h}_t^j$ respectively correspond to the previous memory content and the new candidate memory content. The update gate z_t^j controls how much of the previous memory content has to be forgotten and how much of the new memory content has to be added. The update gate is computed based on the previous hidden states $\mathbf{h}_{t-1}$ and the current input $\mathbf{x}_t$:

$$\mathbf{z}_t = \sigma(\mathbf{W}_z \mathbf{x}_t + \mathbf{U}_z \mathbf{h}_{t-1}) \tag{2.28}$$

Unlike the LSTM, the GRU does not use an output gate, so it fully exposes its memory content each timestep. One major difference from the traditional RNN transition function is that the states of the previous step $\mathbf{h}_{t-1}$ are modulated by the *reset gates* $\mathbf{r}_t$. This behavior allows a GRU to ignore the previous hidden states whenever it is deemed necessary considering the previous hidden states and the current input:

$$\mathbf{r}_t = \sigma(\mathbf{W}_r \mathbf{x}_t + \mathbf{U}_r \mathbf{h}_{t-1}), \tag{2.29}$$

the new memory content is computed by:

$$\tilde{\mathbf{h}}_t = \tanh(\mathbf{W} x_t + r_t \odot U \mathbf{h}_{t-1}). \tag{2.30}$$

The update mechanism helps the GRU to capture long-term dependencies. Whenever a previously detected feature, or the memory content is considered to be important for later use, the update gate will be closed to carry the current memory content across multiple timesteps. The reset mechanism helps the GRU to use the model capacity efficiently by allowing it to reset whenever the detected feature is not necessary anymore. No peephole connections neither output activation functions are used in the GRU architecture.
An initial comparison between GRU and LSTM has been performed by Chung et al. [66] reporting mixed results.

2.4.4 Bidirectional Recurrent Neural Networks

So far we have just considered recurrent networks that have a *"causal"* structure, meaning that the state at time t only captures information from the past. However, in many applications the output prediction may depend on the whole input sequence. For example, in speech recognition, the correct interpretation of the input sequence may depend also on the next future phonemes because of the co-articulation or even on the next few words because of the linguistic context created by word dependencies. Usually in natural language processing tasks like part-of-speech tagging, it happens to have more than one interpretation for the same word, for example it can be a noun or a verb depending on the context. By looking at the overall context, it is possible to solve this ambiguity.

Bidirectional recurrent neural networks (BDRNNs or BRNNs) were invented to address that need by Schuster and Paliwal [67] in 1997. In the BRNN architecture, the information from both the future and the past are used to determine the output at any time t. This is in contrast to the previous systems, in which only the past input can affect the output, and has been used successfully for sequence labeling tasks in natural language processing, among others.

As the name suggests, the idea behind BRNNs is to combine a forward-going RNN and a backward-going RNN. There are two layers of hidden nodes, both are connected to input and output. The two hidden layers are differentiated in that the first has recurrent connections from the past time steps, while in the second the direction of recurrence of the connections is flipped, passing activation backwards in time. Given a fixed length sequence, the BRNN can be trained with ordinary

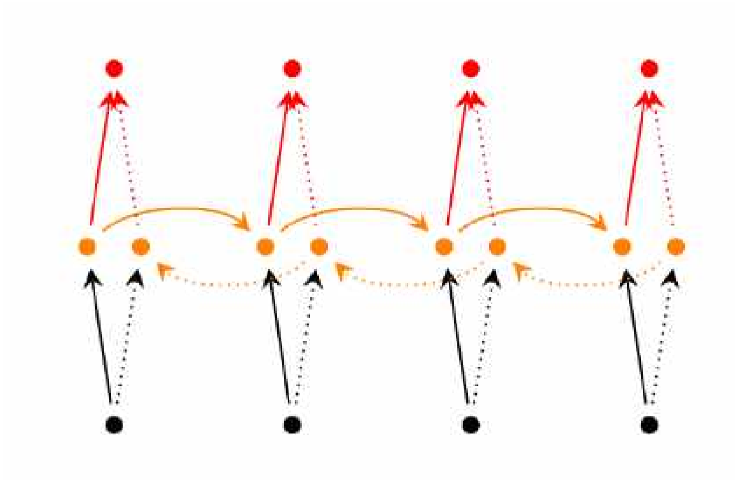


Figure 2.10: A pictorial representation of a bidirectional recurrent network.

backpropagation. The following three equations describe a BRNN:

$$\overrightarrow{\mathbf{h}}_t = \sigma(\mathbf{W}_{\overrightarrow{h}}\mathbf{x}_t + \mathbf{U}_{\overrightarrow{h}}\overrightarrow{\mathbf{h}}_{t-1}) \tag{2.31}$$

$$\overleftarrow{\mathbf{h}}_t = \sigma(\mathbf{W}_{\overleftarrow{h}}\mathbf{x}_t + \mathbf{U}_{\overleftarrow{h}}\overleftarrow{\mathbf{h}}_{t-1}) \tag{2.32}$$

$$\mathbf{h} = [\overrightarrow{\mathbf{h}}_t : \overleftarrow{\mathbf{h}}_t] \tag{2.33}$$

where $\overrightarrow{\mathbf{h}}_t$ and $\overleftarrow{\mathbf{h}}_t$ correspond respectively to the hidden layers in the forwards and backwards directions, and $[\mathbf{x} : \mathbf{y}]$ denotes the concatenation of two vectors.

One limitation of the BRNN is that it cannot run continuously, as it requires a fixed endpoint in both the future and in the past. Actually BRNNs, are not an appropriate machine learning models for the online setting, as it is implausible to receive information from the future, i.e., sequence elements that have not been observed, that would mean that the system is not causal. But for sequence prediction over a sequence of fixed length, it is often sensible to account for both past and future data. Consider the natural language task of part-of-speech tagging. Given a word in a sentence, information about both the words which precede and those which succeed it is useful for predicting that word's POS tag.

Bidirectional RNNs and LSTM have been successfully combined by Graves et al. for phoneme classification [68] and handwriting recognition [57]. Moreover Karpathy et al. use such a network for image caption generation [41].

Chapter 3

Models for Semantic Segmentation

3.1 Introduction

Semantic segmentation is a low-level computer vision problem which involve assigning a label to each pixel in an image. In particular we define this kind of task as a *structured prediction problem* where "structured" means that there is a relation between the output predictions *i.e* the prediction of a pixel is related with the prediction of its neighbours. The feature representation used to classify individual pixels plays an important role in this task, mainly because we need to consider also factors such as image edges, appearance consistency and spatial consistency in order to obtain accurate and precise results. For these reasons designing a strong feature representation is a key challenge in pixel-level labelling problems. Before the arrival of deep networks, the best performing methods mostly relied on hand engineered features classifying pixels independently. Work on this topic includes: TextonBoost [69], TextonForest [70], and Random Forest-based classifiers [71].

Prior works on semantic segmentation include many different approaches, both using RGB data as well as RGB-D. Most of these use local features to classify over-segmented regions, followed by a global consistency optimization such as a CRF.

Typically, a patch is fed into a classifier e.g. Random Forest [3, 70], or Boosting [72, 73] to predict the class probabilities of the center pixel. Features based on appearance [70] or Structure from Motion (SfM) and appearance [3, 72, 73] have been explored for the CamVid road scene understanding test [3]. These per-pixel noisy predictions (often called unary terms) from the classifiers are then smoothed by using a pair-wise or higher order CRF [72, 73] to improve the accuracy. More recent approaches have aimed to produce high quality unaries by trying to predict the labels for all the pixels in a patch as opposed to only the center pixel. This improves the results of Random Forest based unaries [74] but thin structured classes are classified poorly. Dense depth maps computed from the CamVid video have also been used as input for classification using Random Forests [75] . Another approach argues for the use of a combination of popular hand designed features and spatio temporal super-pixelization to obtain higher accuracy [76].

The recent success of supervised deep learning approaches such as large-scale deep Convolutional Neural Networks in many high-level computer vision tasks such as image recognition [38] and object detection [77] motivates exploring the use of CNNs for pixel-level labelling problems.

In parallel to the progress of deep learning techniques, probabilistic graphical models have been developed as effective methods to enhance the accuracy of pixel-level labelling tasks. In particular, Markov Random Fields (MRFs) and its variant Conditional Random Fields (CRFs) have observed widespread success in this area [78, 79] and have become one of the most used graphical models in computer vision. The key idea of CRF inference for semantic labelling is to formulate the label assignment problem as a probabilistic inference problem that incorporates assumptions such as the label agreement between similar pixels. CRF inference is able to refine weak and coarse pixel-level label predictions to produce sharp boundaries and fine-grained segmentations.

In this chapter we give a briefly review the state-of-the-art techniques for semantic segmentation.

3.2 Shallow Models

3.2.1 Conditional Random Field

Most of the state-of-the-art semantic segmentation systems are formulated as the problem of finding the most probable labelling on a Markov Random Field (MRF) or Conditional Random Field (CRF). CRF provides a probabilistic framework to model complex interactions between output variables and observed features. Thanks to the ability to factorize the probability distribution over different labelling of the random variables, CRF allows for compact representations and efficient inference.

A CRF, in the context of pixel-wise label prediction, models pixel labels as random variables that form a Markov Random Field (MRF) when conditioned upon a global observation. The global observation is usually taken to be the image.

Let X_i be the random variable associated to pixel i, which represents the label assigned to the pixel i and can take any value from a predefined set of labels $\mathcal{L} = \{l_1, l_2, \cdots, l_L\}$.

Let $\mathbf{X}$ be the vector formed by the random variables $X_1, X_2, ..., X_N$, where N is the number of pixels (or super-pixels) in the image. Given a graph $G = (V, E)$, where $V = \{X_1, X_2, \cdots, X_N\}$, and the global observation $\mathbf{I}$ (the image) , the pair $(\mathbf{I}, \mathbf{X})$ can be modelled as a CRF characterized by a Gibbs distribution in the form of:

$$P(\mathbf{X} = \mathbf{x}|\mathbf{I}) = \frac{1}{Z(\mathbf{I})} \exp(-E(x|\mathbf{I})) \tag{3.1}$$

Here $E(x)$ is called the energy of the configuration $x \in \mathcal{L}^N$ and $Z(\mathbf{I})$ is the partition function which ensures the distribution is properly normalized and summed to one. Computing the partition function is intractable due to the sum of exponential functions, but such computation is not necessary when the task is to infer the most likely labeling. From now on, we drop the conditioning on $\mathbf{I}$ in the notation for convenience.

Maximizing the posterior probability in 3.1 is equivalent to minimize the energy function. A common model for pixel labeling involves a unary potential $\psi_u(yi; x)$ which is associated with each pixel, and a pairwise potential $\psi_p(x_i, x_j)$ which is associated with a pair of neighborhood pixels:

$$E(x) = \sum_i \psi_u(x_i) + \sum_{i<j} \psi_p(x_i, x_j) \tag{3.2}$$

The unary energy components $\psi_u(x_i)$ measure the inverse likelihood (and therefore, the cost) of the pixel i taking the label x_i, and pairwise energy components $\psi_p(x_i, x_j)$ measure the cost of assigning labels x_i, x_j to pixels i, j simultaneously.

The unary energies predict labels for pixels without considering the smoothness and the consistency of the label assignments. The pairwise energies provide an image data-dependent smoothing term that encourages assigning similar labels to pixels with similar properties.

Given the energy function, semantic segmentation usually follows the following pipelines:

1. Extract features from a patch centered on each pixel
2. With the extracted features and the ground truth labels, an appearance model is trained to produce a compatible score for each training sample
3. The trained classifier is applied on the test image's pixel-wise features, and the output is used as the unary term
4. The pairwise term of the CRF is defined over a 4 or 8-connected neighborhood for each pixel
5. Perform maximum a posterior (MAP) inference on the graph.

To optimize the energy function, various techniques can be applied, such as GraphCut, Belief-Propagation or Primal-Dual methods, etc. A complete review of recent inference methods can be found in [80]. Original CRF or MRF models are usually limited to 4-neighbor or 8-neighbor. Recently, a fully connected graphical model which connects all pixels gained popularity thanks to an efficient inference algorithm based on fast image filtering which only requires that the pairwise term should be a mixture of Gaussian kernels [78]. Modeling the pairwise potentials as weighted Gaussians [78] we obtain:

$$\psi_p(x_i, x_j) = \mu(x_i, x_j) \sum_{m=1}^{M} w^{(m)} k_G^{(m)}(\mathbf{f}_i, \mathbf{f}_j) \tag{3.3}$$

where each $k_G^{(m)}$ for $m = 1, \cdots, M$ is a Gaussian kernel applied on feature vectors.

The feature vector of pixel i, denoted by $\mathbf{f}_i$, can be derived from image features such as spatial location and RGB values. The function $\mu(\cdot, \cdot)$, called the label compatibility function, captures the compatibility between different pairs of labels as the name implies. Minimizing the above CRF energy $E(x)$ yields the most probable label assignment x for the given image.

Before of the deep learning success, the commonly used features were bottom-up pixel-level features such as color or texton, but recently features extracted by deep convolutional neural network have been applied to replace the hand-crafted features, becoming the state-of-the art for these kind of tasks.

The context (such as boat in the water, car on the road) has emerged as another important factor beyond the basic smoothness assumption of the CRF model. Basic context model is implicitly captured by the unary potential, e.g. the pixels with green colors are more likely to be grass class. Recently, more sophisticated class co-occurrence information has been incorporated in the model. Rabinovich et al. [81] learned label co-occurence statistic in the training set and then incorporated it into CRF as additional potential. Later the systems using multiple forms of context based on co-occurence, spatial adjacency and appearance have been proposed in [82–84]. Ladicky et al. [85] proposed an efficient method to incorporate global context, which penalizes unlikely pairs of labels to be assigned anywhere in the image by introducing

Superpixel algorithms

Superpixel algorithms aim to find perceptually meaningful atomic regions grouping pixels together in order to achieve a more suitable representation with respect to the rigid structure of the pixel grid. By exploiting superpixels, the complexity of the model is greatly reduces from millions of variables to only hundreds or thousands. They greatly reduce the complexity capturing image redundancy and providing a convenient primitive from which to compute image features. They have become key building blocks of many computer vision algorithms before the outbreak of deep learning methods. There are many approaches to generate superpixels, each with its own advantages and drawbacks that may be better suited to a particular application. According to [86] there are three main properties which are desirable for the superpixels produced by a given algorithm:

1. Superpixels should adhere well to image boundaries.
2. Superpixels should be fast to compute, memory efficient, and simple to use.
3. Superpixels should both increase the speed and improve the quality of of the results.

Algorithms for generating superpixels can be broadly categorized as *graph-based* or *gradient ascent* methods. Graph-based approaches treat each pixel as a node in a graph where the edge weights between two nodes are proportional to the similarity between neighboring pixels. Then superpixels are created by minimizing a cost function defined over the graph. Gradient-ascent-based algorithms start from a rough initial clustering of pixels and iteratively refine the clusters until some convergence criterion is met to form superpixels. A comprehensive review and comparison of superpixel algorithms can be found in [86].

3.3 Deep models for semantic segmentation

In the last few years Deep Learning transformed Computer Vision, drastically advancing the state of the art of Image Understanding algorithms. There are several reason behind the success of Deep Learning and in particular of ConvNets for computer vision tasks. One of the most important is the massive use of General Purpose GPUs which allowed to learn bigger and deeper models efficiently parallelizing the training phase and outperforming the algorithms used so far. A better understanding of the optimization problems in training neural networks [27,87] led to the discovery of faster optimization methods [25,26] and new parameters initializations [28, 29].

Moreover, the introduction of new kinds of hidden units such as ReLU and its variant pReLU [29] or the recently proposed ELU [37] helped to reduce the vanishing gradient problem and allow to stack more layers.

Bigger datasets such as ImageNet [88] and MS COCO [89] have been introduced for tasks like image recognition, segmentation, and captioning. In particular, an important role in the advance of deep visual recognition architectures has been played by the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) [90], which has served as benchmark tool for different generations of large-scale image classification systems, from high-dimensional shallow feature encodings to deep ConvNets.

Convolutional models

Convolutional networks rapidly became the *de facto* standard in recognition surpassing the performance of shallow models used so far. Convnets showed to be incredibly successful not only on whole-image classification [38, 39, 91], but also on local tasks with structured output such as bounding box object detection [77, 92, 93] and part and key-point prediction [94].

The natural next step in the progression from coarse to fine inference is to make a prediction at every pixel. Recent semantic segmentation algorithms are often formulated to solve structured pixel-wise labeling problems based on CNN. They convert an existing CNN architecture constructed for classification to a fully convolutional network (FCN). They obtain a coarse label map from the network by classifying every local region in image, and perform a simple upsampling, which is implemented as bilinear interpolation, for pixel-level labeling. ConvNets extension for semantic segmentation have been extensively studied in the last years by [5, 9, 95–101].

This approaches aim to learn a strong feature representation end-to-end instead of hand-crafting features with heuristic parameter tuning such as SIFT or HOG. Recent particularly interesting works based on Fully Convolutional Networks (FCN) [9] and "DeepLab" [99] have shown a significant accuracy improvement by adapting state-of-the-art CNN based image classifiers to the semantic segmentation problem.

Gupta et al. [96, 102] create semantic segmentations first by generating contours, then classifying regions using either hand-generated features and SVM , or a convolutional network for object detection. Pinheiro et al. [100] use a recurrent convolutional network in which each application predicts labels at the center location of an input region, given predicted labels from the previous scale and a rescaled input patch. Farabet et al. [103] and Couprie et al. [104] both use a convolutional network applied at multiple scales to find local predictions, then aggregate the predictions using superpixels.

Eiden and Fergus [10] proposed a complete architecture that can be applied both to pixel-wise classification as well to spatially-varying outputs such as depth/normals estimation. They uses an inverted approach with respect to the other methods making a consistent global prediction first and applying iterative local refinements. In so doing, the local networks are made aware of their place within the global scene, and can can use this information in their refined predictions. No superpixels or any post-process smoothing are used in this work.

In their concurrent work, Long et al. [9] adapt the recent VGG ImageNet model to semantic segmentation by applying 1×1 convolutional label classifiers at feature maps from different layers, corresponding to different scales, and averaging the outputs. By contrast, Eiden and Fergus [10] apply networks for different scales in series, which allows them to make more complex edits and refinements, starting from a full image field of view. This architecture easily adapts to many tasks, whereas by using fields of view always centered on the output and summing predictions [9] is specific for semantic labeling.

Hariharan et al. [97] and Gupta et al. [96] likewise adapt deep classification nets to semantic segmentation, but do so in hybrid proposal-classifier models. These approaches fine-tune a Region Convolutional Network system (R-CNN) [77] by sampling bounding boxes and/or region proposals for detection, semantic segmentation, and instance segmentation. Neither method is learned end-to-end since they need a further step to individuate the region proposals.

Besides the success and the wide adoption, there are two significant technical challenges in adapting CNNs designed for high level computer vision tasks such as object recognition to pixel-level labelling tasks: signal downsampling and spatial invariance. Firstly, traditional CNNs have convolutional filters with large receptive fields and hence produce coarse outputs when restructured to produce pixel-level labels. Moreover the presence of strided convolutions (downsampling) and max-pooling layers in CNNs further reduces the chance of getting a fine segmentation output. This, for instance, can result in non-sharp boundaries and blob-like shapes in semantic segmentation tasks.

The second problem relates to the fact that obtaining object-centric decisions from a classifier requires invariance to spatial transformations, inherently limiting the spatial accuracy of the Deep CNN model. Moreover CNNs do not impose any smoothness constraints that encourage label agreement between similar pixels, and spatial and appearance consistency of the labelling output resulting in poor object delineation and small spurious regions in the segmentation output.

To overcome the drawbacks of CNNs in pixel-level labelling tasks, they are usually combined with Conditional Random Field (CRF) in order to improve the ability of the system to capture fine details. CRFs can be applied as post-processing step after the CNNs training [99] but requires two different training steps. This kind of approach is widely used in the literature but it does not fully exploit the strength of CRFs

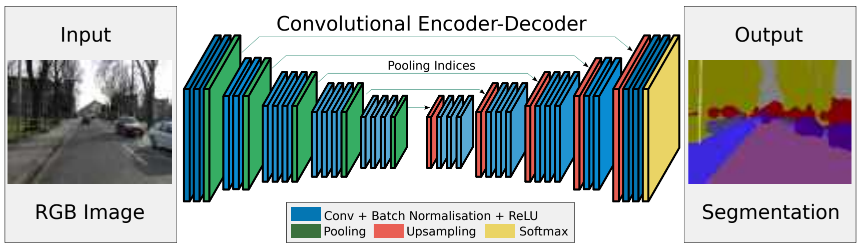


Figure 3.1: An example of deep convolutional architecture: SegNet [5]

since it is not integrated with the deep network. In this setup, the deep network, being unaware of the CRF, only learns to predict unstructured labels for pixels during its training.

Zheng et Al. [105] proposed an end-to-end deep learning solution for the pixel-level semantic image segmentation problem which combines the strengths of both CNNs and CRF based graphical models in one unified framework. More specifically, they showed that a dense CRF inference can be formulated as a Recurrent Neural Network which can refine coarse outputs from a traditional CNN in the forward pass, while passing error differentials back to the CNN during training. Using this formulation the whole deep network can be trained end-to-end utilizing the usual back-propagation algorithm.

Badrinarayanan et Al. [106] proposed a deep fully convolutional neural network architecture for semantic pixel-wise segmentation called *SegNet*. The model consists of an encoder network and a corresponding decoder network followed by a pixel-wise classification layer. The encoder network share the same topology with the convolutional layers in the VGG-16 architecture [91]. In order to adapt the architecture to pixel-wise classification they remove the fully connected layers of VGG-16 making the encoder network significantly smaller in terms of parameters and easier to train. The key component of SegNet is the decoder network which consists of a hierarchy of upsample layers one corresponding to each encoder. The appropriate decoders use the max-pooling indices received from the corresponding encoder to perform non-linear upsampling of their input feature maps. The decoder network map the low resolution encoder feature maps to full input resolution feature maps for pixel-wise classification. Specifically, the SegNet decoder uses pooling indices computed in the max-pooling step of the corresponding encoder to perform non-linear upsampling. This reduce the complexity of the upsample layer needing only to learn how to densify the sparse feature maps.

Similar proposed architectures are Deconvolutional Network [107] and its semi-supervised variant the Decoupled network [108] which both use the max locations of the encoder feature maps (pooling indices) to perform non-linear upsampling in the decoder network. The authors proposed these architectures, independently of SegNet.

However DeconvNet [107] encoder network includes also the fully connected layers from the VGG-16 architecture which consists of about 90% of the parameters of their entire network. This makes training of their network very difficult and thus require additional aids such as the use of region proposals to enable training. Moreover, during inference these proposals are used and this increases inference time significantly.

3.4 Datasets for urban street scene parsing

3.4.1 The Cambridge-driving Labeled Video Database

The Cambridge-driving Labeled Video Database (CamVid) [3, 109] is the first collection of videos with object class semantic labels which has been proposed for urban street scene semantic segmentation. The video footage was recorded from driving around various urban settings and is particulary challenging given various lighting and weather settings such as dusk, dawn or sunny, rainy weather. The database provides ground truth labels that associate each pixel with one of 32 semantic classes, but the standard experimental setting is given by a subset of 11 classes which group together in broader categories.

The dataset contains ten minutes video footage recorded at 30 Hz and 960×720 resolution. The corresponding semantically labeled ground-truth images at 1Hz and in part, 15Hz. The total annotated images are 701, split in 367 images for training , 101 for validation and 233 for testing.

3.4.2 Daimler Urban Segmentation Dataset

The Daimler Urban Segmentation dataset [14] contains 500 stereo grayscale image pairs with pixel-wise semantic class annotations for the left images. While the image size in the dataset is 1024×440, only the middle is fully labeled. Hence, the effective image size is 976×360. The dataset is composed for evaluating only the semantic labeling using

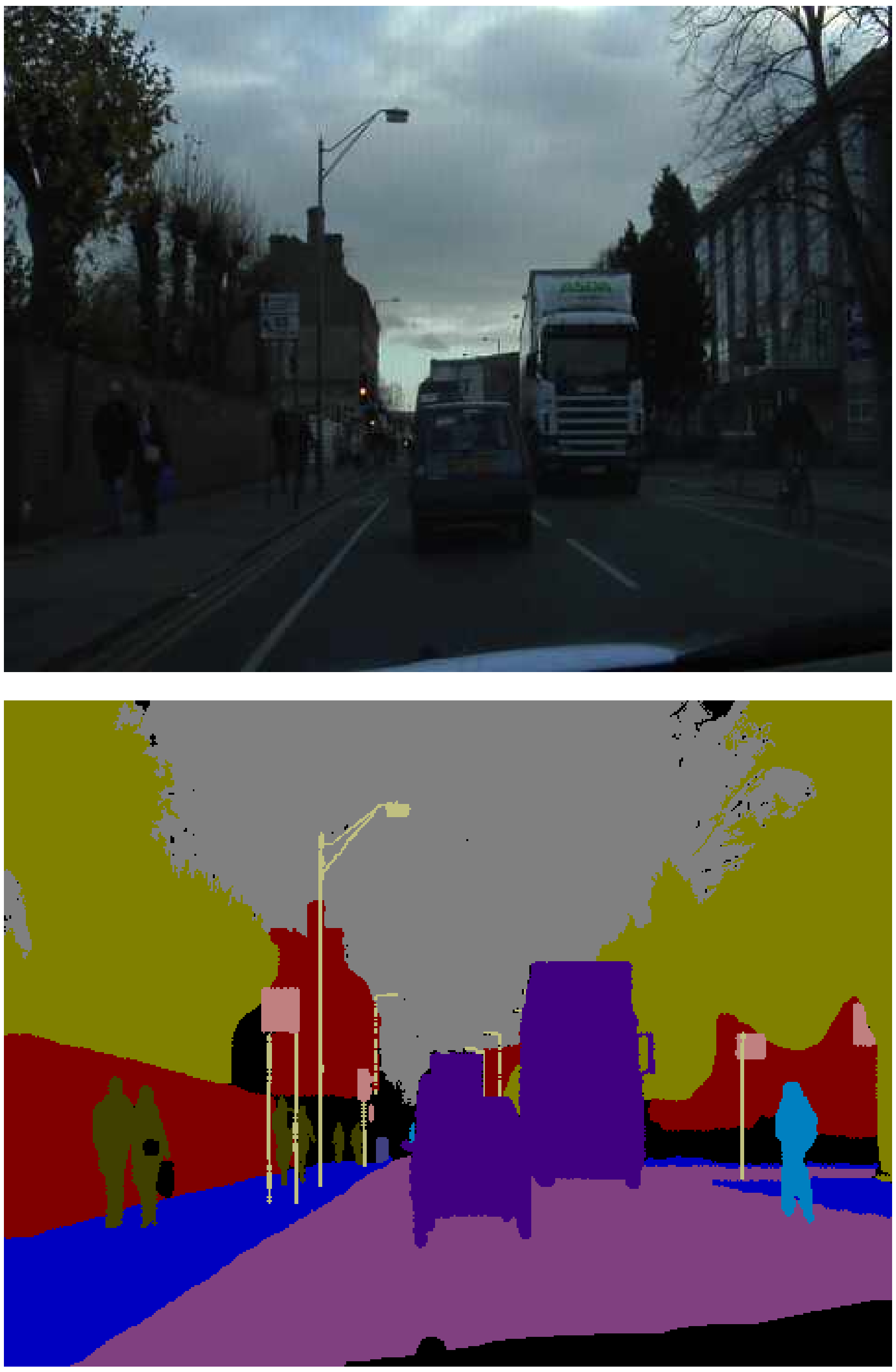


Figure 3.2: An example from the CamVid dataset [3].

stereo image pairs. Disparity maps computed with SfM are provided. The semantic labels in the annotations include ground, sky, building, pedestrian, vehicle, curbs, bicyclist, motorcyclist, and background clutters. However, only the ground, sky, building, pedestrian, and vehicle are considered in the evaluation protocol.

3.4.3 KITTI Road Detection

The KITTI Vision Benchmark Suite [8] supports and assesses vision algorithms in the context of autonomous driving. This benchmark includes datasets for stereo vision, optical flow, visual odometry, 3D object recognition, and tracking. The KITTI dataset does not officially support a Semantic Segmentation benchmark, however several researchers have annotated some KITTI images. The policies behind the annotations are different for every subset of images, so it's hard to use them jointly for a proper training procedure. KITTI have also provides specific benchmark for Road Detection that is simpler binary classification task. The images are all manually annotated and contains binary labels of whether each pixel is drivable road or not. The dataset consist of 289 images for training and 290 images for testing.

3.4.4 The Cityscapes Dataset

Cityscapes dataset [4] is a new large-scale dataset that contains a diverse set of stereo video sequences recorded in street scenes from 50 different cities, with high quality pixel-level annotations of 5000 frames in addition to a larger set of 20000 weakly annotated frames. The dataset is thus an order of magnitude larger than similar previous attempts and the images are provided in high resolution 2048×1024.

The high variability of outdoor urban streets makes accurate scene understanding very challenging. Unfortunately, most of scene parsing datasets fall behind in capturing this high variance, mostly due to scarcity of available samples.

For instance, the KITTI dataset contains 6 hours of video material, all recorded in the Karlsruhe, Germany, metropolitan area. Out of these recordings, only 25% are publicly available and semantic segmentation task is not officially supported, so roughly 430 images have pixel-level semantic annotations, provided by different independent research groups.

Cityscape dataset takes into account this issue by focusing on a high diversity between annotated images in order to capture a wider range of street scenes than any previous dataset. Video have been recorded in approximately 50 different cities to reduce city-specific overfitting, collected during several months, covering spring, summer, and autumn. Moreover recordings are restricted only to good weather conditions, which already pose a significant challenge for computer vision. Approximately half of the annotated images are extracted from long video sequences, while the remaining are the 20th images from 30 frame video snippets (1.8s). The surrounding frames are provided as context for methods exploiting Optical Flow, Tracking, or Structure-from-Motion. In addition to video, they provide corresponding right stereo views, precomputed depth maps, GPS coordinates, and ego-motion data from the vehicle odometry.

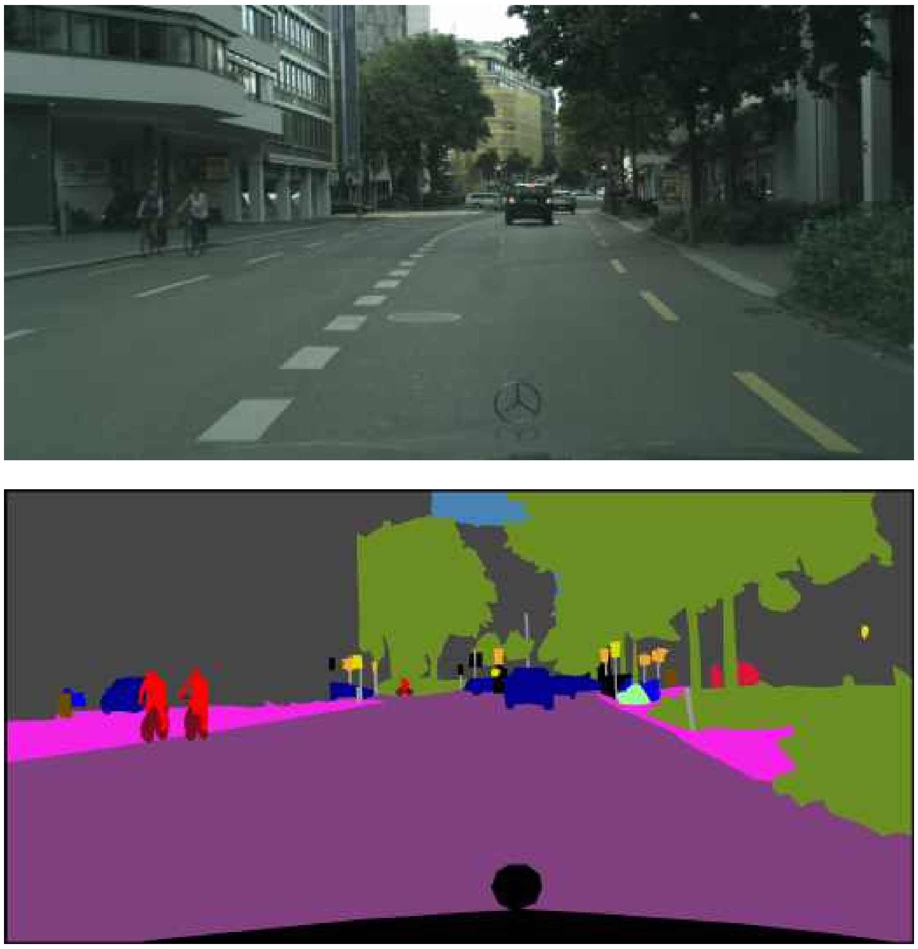


Figure 3.3: An example from the Cityscapes dataset [4].

Chapter 4

ReSeg architecture

In this chapter we describe the original ReSeg model presented by Visin et. Al [2] detailing the recurrent and the upsampling layers that compose the architecture.

4.1 ReNet and ReSeg models

As extensively discussed in the previous chapter, Convolutional Neural Networks have become the *de facto* standard in many computer vision tasks. Object Recognition, Image Classification and recently Semantic Segmentation problems are increasingly often solved with a deep convolutional network leveraging its architecture properties and its capacity of extracting task specific, yet at the same time generic image representations.

On the other hand, recurrent neural networks have become the first choice for modeling sequential data, especially in the field of natural language processing. RNNs have become one of the most widely used methods for natural language related tasks such as language modeling and machine translation [48]. The addition of recurrent connections allows RNNs to exploit the previous context, as well as makes them more robust to warping along the time axis than non-recurrent models. Access to contextual information and robustness to warping are also important when dealing with multi-dimensional data. For example, a face recognition algorithm should use the entire face as a context, and should be robust to changes in perspective, distance etc. It therefore seems desirable to apply RNNs to such tasks.

In the recent years in fact recurrent neural networks have begun to be employed in a few computation vision tasks (Kalchbrenner et al. [110]; Graves & Schmidhuber [57]).

The architecture proposed in Visin et al. [2, 11] is related and inspired by this earlier work, but relies on multiple uni-dimensional RNNs coupled in a novel way to address the problem of Object Classification and Semantic Segmentation. In this work the authors show how Recurrent Neural Networks can be used as an alternative to the classical convolutional architecture to process images for computer vision tasks, specifically for object classification and semantic segmentation.

As we have seen before Bidirectional Recurrent Networks can process a time sequence forward and backward through time so that the output units compute a new representation which depends on both the past and the future samples of the time instant t. This idea can be easily extended to 2-dimensional input, such as images, by having **four** RNNs, each one scanning the four directions: up, down, left, right. At each position (i, j) of the 2-D grid, an output $O_{i,j}$ can encode a representation that would capture mostly local information but also long-range dependencies.

Compared to a convolutional network applying RNNs to images is typically more expensive but allows for long-range lateral interactions between neurons of the same layer.

Convolutional Neural Networks rely on fixed-sized kernels to introduce context then it can be quite hard to extract a good feature representation to labeling pixels just by looking at a small region around. Indeed the category of a pixel may depend on relatively short-range information (e.g. the presence of eyes in a human face generally indicates the presence of a nose nearby), but may also depend on long-range information. For example, identifying a grey pixel as belonging to a road, a sidewalk, a gray car, a concrete building, or a cloudy sky requires a wide contextual window that shows enough of the surroundings to make an informed decision.

In this work we focused on the *ReNet* architecture proposed by Visin et Al. [11] for object recognition. We extensively tested its extension *ReSeg* [2] for semantic segmentation showing that it can reach state-of-the art performance on challenging urban street scene parsing datasets.

In the proposed model, the image passes through several layers. It is first swept by two horizontal RNNs in both directions (left to right and right to left) and the concatenation of their activations is then swept by a second couple of RNNs vertically (top to bottom and bottom to top). The output activation of the ReSeg layer is the concatenation of the hidden states of these RNNs, which encodes the local features of the image in each position *with respect to the whole input image*, in contrast to the usual convolution+pooling layer which only has a local context window. The result of the application of these 4 RNNs is a feature map which summarizes the informations of the image from all the directions encoding them in a context feature map that is both global and local. In the next sections we are going to analyze each component of the architecture in detail.

4.1.1 Recurrent Layer

In the ReSeg architecture [2] each recurrent layer is composed by 4 RNNs coupled together in such a way to capture the local and global spatial structure of the input data. Specifically, we take as input an image (or the feature map of the previous layer) $\mathbf{X}$ of elements $x \in \mathbb{R}^{H \times W \times C}$, where H, W and C are respectively the height, width and number of channels (or features) and we split it into $I \times J$ patches $p_{i,j} \in \mathbb{R}^{H_p \times W_p \times C_W}$.
The input is then swept vertically a first time with two RNNs $f^{\uparrow}$ and $f^{\downarrow}$, with U recurrent units each, that move top-down and bottom-up respectively.

At every time step each RNN reads the next patch $p_{i,j}$ and, based on its previous state, emits a projection $o^{\star}_{i,j}$ and updates its state $z^{\star}_{i,j}$:

$$o^{\downarrow}_{i,j} = f^{\downarrow}(z^{\downarrow}_{i-1,j}, p_{i,j}), \text{ for } i = 1, \cdots, I \tag{4.1}$$

$$o^{\uparrow}_{i,j} = f^{\uparrow}(z^{\uparrow}_{i+1,j}, p_{i,j}), \text{ for } i = I, \cdots, 1 \tag{4.2}$$

Once the first two vertical RNNs have processed the whole input $\mathbf{X}$, their projections $o^{\downarrow}_{i,j}$ and $o^{\uparrow}_{i,j}$ are aggregated together to obtain a composite feature map $\mathbf{O}^{\updownarrow}$ whose elements $o^{\updownarrow}_{i,j} \in \mathbb{R}^{2U}$ can be seen as the activation of a feature detector at the location (i, j) with respect to all the patches in the j-th column of the input. We denote what we described so far as the *vertical recurrent sublayer*.

After obtaining the concatenated feature map $\mathbf{O}^{\updownarrow}$, this is swept over each of its rows with another pair of RNNs, $f^{\leftarrow}$ and $f^{\rightarrow}$. In this case $\mathbf{O}^{\updownarrow}$ is not splitted into patches so that the second recurrent sublayer has the same granularity as the first one, but this is not a constraint

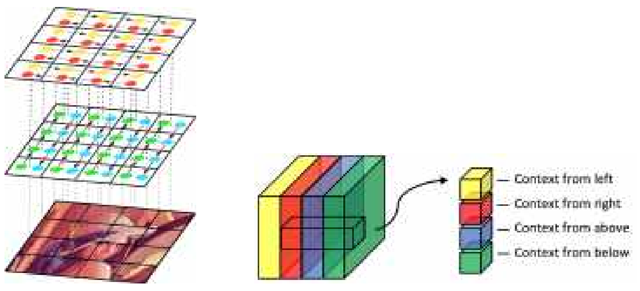


Figure 4.1: In (a) we can see a pictorial representation of a ReNet layer that consists of a vertical and horizontal scanning of the image; in (b) we give an interpretation of the feature map extracted by the layer.

of the model and different architectures can be explored.
With a similar but specular procedure as the one described before, we proceed reading one element $o^{\updownarrow}_{i,j}$ at each step

$$o^{\rightarrow}_{i,j} = f^{\rightarrow}(z^{\rightarrow}_{i,j-1}, o^{\updownarrow}_{i,j}), \text{ for } j = 1, \cdots, J \tag{4.3}$$

$$o^{\leftarrow}_{i,j} = f^{\leftarrow}(z^{\leftarrow}_{i,j+1}, o^{\updownarrow}_{i,j}), \text{ for } j = J, \cdots, 1 \tag{4.4}$$

We obtain a concatenated feature map $\mathbf{O}^{\leftrightarrow} = \{o^{\leftrightarrow}_{i,j}\}^{j=1,\cdots J}_{i=1,\cdots I}$ once again with $o^{\leftrightarrow}_{i,j} \in \mathbb{R}^{2U}$. Each element $o^{\leftrightarrow}_{i,j}$ of this *horizontal recurrent sublayer* represents the features of one of the input image patches $p_{i,j}$ with *contextual information from the whole image.*

It is also possible to apply all the 4 RNNs on the original input layer $\mathbf{X}$ rather than on the feature map $\mathbf{O}^{\updownarrow}$ extracted by the first recurrent sublayer, in order to concatenate $\mathbf{O}^{\updownarrow}$ and $\mathbf{O}^{\leftrightarrow}$ into a bigger feature map where $o_{(i,j)} \in \mathbb{R}^{4U}$. We have studied these two different architectural choices in our experiments, please refer to the chapter 5 for a detailed comparison.

As in any convolutional network, it is possible to concatenate many recurrent layers $\mathbf{O}^{(1,\cdots,L)}$ one after the other to make ReNet deeper and to capture increasingly complex features of the input image. The deep ReSeg is a smooth, continuous function since it is a composition of several nonlinear function, and the parameters can be estimated by any optimization algorithm which perform gradient descent on the cost function, and the gradient is computed by the backpropagation algorithm [16].

Since each ReSeg layer projects its input in a new complex representation, the architecture presented so far can be seen as a complex

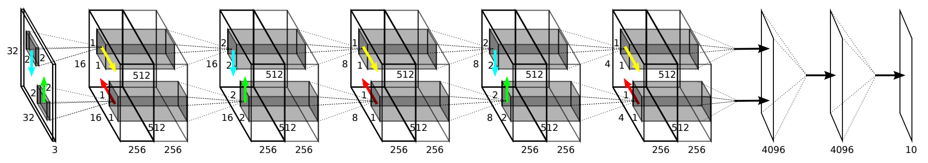


Figure 4.2: The ReNet architecture for image recognition

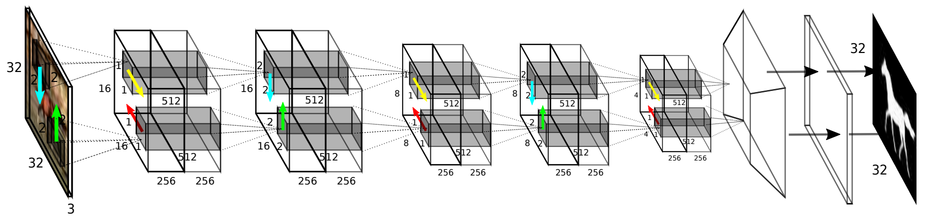


Figure 4.3: The ReSeg architecture for pixel-wise semantic segmentation

feature learning model which can be embedded in any computer vision system. Depending on the task, the last layers of the network that follow the ReSeg layers will vary. In an image classification scenario, after any number of recurrent layers are applied to an input image, the activation of the last recurrent layer may be flattened and fed into a differentiable classifier, for instance it can pass through several fully-connected layers followed by a softmax non-linearity.

Since by design each recurrent layer processes patches, the size of the last composite feature map will be smaller than the size of the initial input $\mathbf{X}$. To perform Semantic Segmentation we will therefore need to add one or more layers to expand the feature map back to the size of the image before passing it into a softmax non-linearity to compute the corresponding segmentation mask. There are several different architectures to accomplish this, we will discuss them in detail in the next section.

4.1.2 Upsampling layer

Linear fully-connected upsampling

The easiest way to enlarge a composite ReSeg feature map $\mathbf{O}^{(l)}$ before feeding it into a softmax non-linearity is a two-step operation: first the $2U$ features of the last feature map are extended by an upsampling factor $U_{up} = U_{up}^W \cdot U_{up}^H$ with a linear fully-connected layer to obtain a feature map $\mathbf{E}$ of $2U \cdot U_{up}$ features:

$$\mathbf{E} = \mathbf{O}^{(l)} \cdot \mathbf{W} + \mathbf{b} \tag{4.5}$$

where $\mathbf{W} \in \mathbb{R}^{2U \times 2U \cdot U_{up}}$ and the upsampling factor components are computed as:

$$U_{up}^{W} = \prod_{l \in layers} W_p^{(l)}, \qquad U_{up}^{H} = \prod_{l \in layers} H_p^{(l)} \tag{4.6}$$

The resulting extended feature map $\mathbf{E}$ can then be re-arranged so that each of its entries $e_{i,j} \in \mathbb{R}^{1 \times 2U \cdot U_{up}}$ is mapped to a patch $f_{i,j} \in \mathbb{R}^{U_{up}^{H} \times U_{up}^{H} \times 2U}$ in the output feature map $\mathbf{F}$. A softmax classifier can then be applied on this upscaled feature map to get the per-pixel class prediction probabilities.

Deconvolutional networks: gradient-based upsampling

Deconvolutional Networks (DeconvNets) were presented for the first time as a way of extracting features by performing unsupervised learning [111]. Successively, DeconvNet have been used to address the problem of ConvNet visualisation by Zeiler et Al. [32]. They proposed an architecture which aims to approximately reconstruct the input of each layer from its output.

Simonyan et Al. [112] gave a more rigorous explanation of the mathematics behind DeconvNets showing that a gradient-based ConvNet visualisation method generalizes deconvolutional networks. DeconvNet-based reconstruction of of the n-th layer input X_n is either equivalent or similar to computing the gradient of the visualised neuron activity f with respect to X_n, so DeconvNet effectively corresponds to the *gradient back-propagation* through a ConvNet.
For the convolutional layer $X_{n+1} = X_n * K_n$, the gradient is computed as:

$$\frac{\partial f}{\partial X_n} = \frac{\partial f}{\partial X_{n+1}} * K_n^T \tag{4.7}$$

where K_n and K_n^T are the convolution kernel and its flipped version, respectively. The convolution with the flipped kernel exactly corresponds to the filtering operation in a DeconvNet which compute the n-th layer reconstruction R_n :

$$R_n = R_{n+1} * K_n^T \tag{4.8}$$

Computing the approximate feature map reconstruction R_n using a DeconvNet is equivalent to computing the derivative $\partial f / \partial X_n$ using back-propagation, indeed, this can be seen as the generalization of DeconvNet [32]. In particular the gradient-based techniques can be applied to the visualization of activities in any layer, not just a convolutional one. In fact, the procedure described so far was initially proposed to visualize the feature maps learned by a ConvNet but it can be used in a more general framework for semantic segmentation to restore the original dimension of the input after that is passed through several layers which reduced its dimension.

Upsamplings of factor s can be considered convolutions with a stride of integer fraction $1/s$, therefore using the backward pass of a convolution of stride s will result in the correct output shape. This kind of upsampling is known as *fractional convolution* [9], *backward convolution* or more commonly *deconvolution* [32]. Upsampling is performed for end-to-end learning by backpropagation from the pixelwise loss. A stack of deconvolution layers and activation functions can be used to learn a nonlinear upsampling.

In our case, it is possible to think of the last ReSeg feature map H as the result of a convolution of a kernel K with the desired pre-sigmoid feature map O:

$$H = K * O \tag{4.9}$$

It is then possible to use the gradient of this convolution to upsample the feature map $O_{k,l}$ and compute the contribution of each element of the composite feature map $H_{i,j}$ to each element of the pre-sigmoid feature map.

In the literature the upsampling operation is termed in many different ways: “In-network Upsampling”, “Fractionally-strided convolution”, “Backwards Convolution” [9], “Deconvolution” [32], while the operation is implemented in Theano with the name of “Transposed convolution” [113].

It is important to stress that the term *Deconvolution* has been abused by the Deep Learning community to refer to the upsampling operation. The original Deconvolution term comes from the Signal Processing field and it has nothing to do with trying to invert a convolution + pooling layer or upsampling but it is the name given to the inverse operator of a linear convolution. The operation described in this section does not try to invert the convolution as that would require actually solving a linear inverse. Also, unless the convolution weights are constraint to be orthonormal, taking the transpose of a convolution is not the same as inverting it. Nothing about this layer is an inverse in the strict sense or an actual deconvolution but this term became of common use in the Neural Network community to address this techniques.

Bilinear Interpolation

In addition to the above variants, it is possible to adopt an upsampling method widely used in image processing called *Bilinear Interpolation*. This form of upsampling provides a way to enlarge the dimension of the feature maps in output at the last layer of the ReSeg network to restore the original dimension of the input image and compute the final prediction mask. The key idea is to perform linear interpolation first in one direction (rows), and then again in the other direction (columns). The operation can be also expressed in form of a convolution where the kernel depends on the upssampling factor. For example the kernel of a 2x upscaling operation with bilinear interpolation is given by:

$$\begin{bmatrix} \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \\ \frac{1}{2} & 1 & \frac{1}{2} \\ \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \end{bmatrix} \tag{4.10}$$

In this case the kernel coefficients are fixed and don't require to be learned by the training procedure. This method reduces the number of parameters and the complexity of the model making the training simpler, and limiting the risk of incurring in overfitting when the size of the training set is small. Moreover Bilinear Interpolation proved to be sufficiently effective with respect to other parametric alternative such as Gradient-based Upsampling [9, 106], so it's worth taking it in consideration.

4.1.3 Comparing ReSeg and ConvNets

The ReSeg model was initially presented as an innovative alternative to the classical convolutional neural network models for image recognition. There are many similarities and differences between ReSeg and a convolutional neural network. In this section we highlight a few points of comparison between the two models.

The main difference between the two architectures is that ReSeg exploit short-range and long-range dependencies in the input learning both local and global features at the same time, while ConvNets use only local information. At each layer, both kind of networks apply the same set of filters to patches of the input image or of the feature map from the layer below. ReSeg, however, propagates information through *lateral connections* that span across the whole image, while ConvNets exploit local information only. The lateral connections should help extract a more compact feature representation of the input image at each layer, which can be accomplished by the lateral connections removing/resolving redundant features at different locations of the image. This should allow ReSeg resolve small displacements of features across multiple consecutive patches.

Max-pooling layers are usually applied after each convolutional layer over a small region in order to achieve *local translation invariance*. In contrast, ReSeg does not use any pooling due to the existence of learned lateral connections. The lateral connection in ReSeg can emulate the local competition among features induced by the max-pooling. This does not mean that it is not possible to use max-pooling in ReSeg. The use of max-pooling in ReSeg could be helpful in reducing the dimensionality of the feature map, resulting in lower computational cost but it might be hard to be inverted.

This is important in architectures such as ReSeg where the dimension of the input is repeatedly reduced by the encoder layers and it has to be restored by decoder upsample layers. ReSeg is end-to-end smooth and differentiable, making it suitable to be used as a decoder also in auto-encoders or any of its probabilistic variants [114].

In some sense, each layer of the ReSeg can be considered as a variant of a usual convolution+pooling layer, where pooling is replaced with lateral connections, and convolution is done without any overlap.

4.1.4 Comparing ReSeg and Multi-Dimensional RNNs

Recurrent Neural Networks were originally developed as a way of extending neural networks to unidimensional sequential data. However Graves et Al. [115] introduced more complex multi-dimensional recurrent neural networks, extending the applicability of RNNs to n-dimensional data for computer vision, video processing, medical imaging and many other areas. This kind of architecture, combined with multi-directional hidden units and LSTM memory cells, showed its success for offline Arabic handwriting recognition [57].

A multidimensional RNN, on the other hand, requires an exponential number of RNNs at each layer (2^d). In ReSeg each patch of the image is flattened over the feature dimension so the input sequence remains uni-dimensional. The main important consequence of this approach is that the number of RNNs at each layer scales linearly with respect to the number of dimensions d of the input image ($2d$).

Furthermore, the proposed variant is more easily parallelizable, as each RNN is dependent only along a horizontal or vertical sequence of patches. This architectural distinction results in our model being much more amenable to distributed computing than that of [57]. Kalchbrenner et Al. [110] further extended many of the concepts from the multidimensional RNN paper of [57], and bears some similarity to the ReSeg approach. Grid LSTM inherently uses three dimensional blocks, and modulates information passed over depth, while ReSeg simply stacks hidden layers and requires less recurrent passes over the data. Kalchbrenner et Al. [110] show promising results over a number of tasks, including MNIST recognition, but do not have results for image segmentation or larger image datasets as of this work.

4.1.5 Classification layer

Once the dimension of the image has been reconstructed by the upsampling layers, the feature maps go in input to a pixel-wise Softmax classifier in order to predict the class label for each pixel and compute the segmentation mask. The final output then has as many channels as there are classes. We use a pixel-wise cross entropy loss function where the loss is summed up over all the pixels in a mini-batch.

$$\mathcal{L}(C, C^*) = -\frac{1}{n}\sum_i C_i^* \log(C_i) \tag{4.11}$$

where $C_i = e^{z_i} / \sum_c e^{z_{i,c}}$ is the class prediction at pixel i given the output z of the final ReSeg layer, and C^* is the ground truth of the segmentation mask.

In the next chapter we are going to show the implementation of the system described so far and present the experimental results that we obtained.

Chapter 5

Model implementation and evaluation

In this chapter we give some insights on the implementation of the ReSeg architecture and we explain the choices that we made in the design of the experiments that we perform. In particular we tested the performance of the model on different dataset for semantic segmentation of urban street scenes.

5.1 Libraries

The ResSeg architecture has been written in Python using two main libraries that support the implementation of neural networks models: *Theano* [13, 116] and *Lasagne* [12].

5.1.1 Theano

Theano is a Python library developed at the LISA laboratory of the University of Montreal to support rapid implementation of efficient machine learning algorithms. It allows to define, optimize and efficiently evaluate symbolic mathematical expressions involving multi-dimensional arrays, also known as *tensor*. When we are creating a model with Theano first we have to define the symbolic expression that we want to compute, then Theano builds the computational graph of all the variables and operations that need to be performed to reach the output values. This graph can be applied on specific inputs to obtain the values of the outputs.

Some of the most appealing features that Theano provides are:

- ***Automatic differentiation***: Theano is able to automatically compute the gradients of a symbolic expression. This allows to implement only the forward pass of the model (the prediction) and then to train it performing gradient descent on the gradient computed by the library.
- ***Transparent use of the GPU***: the same code can be run either on CPU or GPU. More specifically, Theano will figure out which parts of the computation should be moved to the GPU automatically generating CUDA code.
- ***Speed and stability optimizations***: Theano internally reorganizes and optimizes the operations of the symbolic graph, in order to make their computation run faster and be more numerically stable. It also can compile some operations into C code, in order to speed up the computation.

Technically, Theano is not a machine learning library, as it does not provide pre-built models ready to be trained. Instead, it is a mathematical library that provides tools to build machine learning models. For this reason we decided to use a library such as Lasagne, which provides an implementation for most of the current neural network components.

5.1.2 Lasagne

Lasagne is a machine learning library built on top of Theano which provides the implementation of useful blocks making easy to build and train neural networks models. The main reasons why we choose to use Lasagne for the implementation of the ReSeg architecture are the following:

- It is built on top of Theano, so it generally follows Theano's conventions and the methods typically accept and return Theano expressions. This makes constructing commonly used network structures easy but also allows to build more complicated models such as ReSeg.
- It is mainly *Object-Oriented*: it provides an abstract class *Layer* that can be extended to implement the components for any kind of model. The class is a container for the parameters of the layer,

the implementation method, and the method that returns the output shape. An object layer is the basic block to build a neural network, it takes as input a layer and it return another layer as output making possible to stack several layers together and build deep models.

- It provides methods to automatically infer the shape of the result of a stack of layers.
- It provides reference implementations for many kind of models and useful components for the training procedure such as optimization and initialization methods which are highly optimized.
- It is supported by a growing community of researchers and developers.

5.2 Server configuration

We run all the experiments on a Ubuntu 14.04 server that mounted an Intel Core i7-920 CPU (8M Cache, 2.66 GHz), 24G RAM memory, and a NVIDIA GeForce GTX TITAN X graphics card with 12G GDDR5 memory.

5.3 Hyper-parameters

When we design an experiment, to verify the performance of a model on a specific task, we need to make several decisions to choose the correct value for all the hyper-parameters. Basically we can divide the hyper-parameters in two types. The first ones are the hyper-parameters that are specific for the model, which in the case of ReSeg are the following:

ReSeg hyper-parameters

- ***Dimension of the patches*** is the portion of the input that is processed at each step by the recurrent networks and represents the local context that is considered around each pixel. It also determines the downscale factor after each Reseg layer.
- ***Number of layers*** is the *depth* of the network, and it specifies how many ReSeg layers are stacked together.

- ***Number of neurons per hidden layer*** specifies the number of hidden units of each recurrent layer of the model.
- ***Kind of the recurrent unit*** specifies if we are using a GRU or LSTM memory unit.
- ***Stacking the recurrent sub-layers***: the input of each ReSeg layer can be scanned by 4 RNNs in parallel or can be processed by 2 sub-layers stacked together. In the latter case the first sub-layer scans the input in the vertical direction, then the second recurrent sub-layer scans the output of the first recurrent sub-layer in the horizontal direction.
- ***Upsampling layer*** specifies if the we use a the *Linear fully-connected* layer, a *Transposed Convolutional* layer, or simply *Bilinear Interpolation* to reconstruct the initial dimension of the input after that has been reduced by the ReSeg layers.
- ***Activation functions*** are the nonlinearities that are applied after each *Transposed Convolutional* layer.

Training hyper-parameters

These second type of parameters are more general and are involved in many kind of models:

- ***Number of epochs*** is the number of times the entire training set is scanned by the training algorithm. There are many techniques to decide when to stop training the model before it overfits [1]. For instance, in our experiments we use *early stopping*: it requires to split the examples in the dataset in 3 subsets: training set, validation set and testing set. The training stops after that the error on the validation set begins to increase significantly even though the error on the training set continues to decrease. This happens when the network starts to overfit the training set and stopping the training with early stopping avoids this issue.

[1] Overfitting is the opposite of the concept of generalization and happens when the model is learning to classify the examples of the training set, but is not able to correctly classify other examples never seen before. This means that the model is not learning the representation behind the data but also the noise present in the training dataset.

- ***Weight decay*** is a regularization technique that adds a penalty to the cost function based on the $l2$-norm of the weights. This penalty is applied in order not to let the weights increase too much which could be a cause of overfitting.

- ***Optimization method*** is the algorithm that optimize the loss function. Usually it is the Stochastic Gradient Descent, but it can be any variant of it.

- ***Learning rate*** is the step-size that is applied in the weights update equation.

- ***Mini-batch size*** determines the number of examples B to be contained in each iteration of the training procedure. When $B = 1$ this is the original stochastic gradient descent setting, when B is equal to the size of the training set then this is the standard (also called *"batch"*) gradient descent. We define as mini-batch size any intermediate value of B where the update rule is based on an average of the gradients inside each block of B.

- ***Weights initialization*** is the way the weights of our models are initialized. There are many ways to initialize the parameters of a model. In this work, we initialize the weights following the initialization procedure described by Glorot & Bengio [28] and by He et Al. [29]. A good initialization is important to help the optimization method to rapidly converge to a solution.

5.4 Designing the experiments

In their preliminary work Visin et Al. [2] tested the performance of the ReSeg model on simple datasets such as Weizmann Horse [117], Fashionista [118] and Oxford Flower [119] performing binary classification to assess the ability of the model to separate *background* from *foreground.*

In this work we first implemented the ReSeg architecture in *Lasagne* to perform multi-class semantic segmentation, then we tested its performance on more complicated datasets specific for urban street scene understanding. To accomplish this task we needed to accurately design the experiments in order to understand which is the architecture that achieves the best performance on the Semantic Segmentation task.

As we have seen in the chapters 2 and 3 Convolutional Neural networks have been extensively studied for computer vision tasks. On the other hand there are only few works that use fully recurrent models for image understanding, this makes the analysis of the ReSeg architecture hard and full of uncertainties on which are the correct value to pick for the hyper-parameters.

Deep Neural Networks typically require a long time to train, so performing hyper-parameter search can take many days or even weeks. It is important to stress on this fact since it influences the design of the experiments. To make an example of the complexity of our model, training a basic ReSeg configuration on the CamVid dataset [3] requires from 10 to 14 hours on a NVIDIA GeForce GTX TITAN X, which is one of the top GPU available on the market. It is clear that the exploration of the entire space of the hyper-parameters of the model would be infeasible since it would require too much time and computational power.

Usually a *random search optimization* is performed having a *worker* that continuously samples random hyper-parameters and performs the optimization. During the training, the worker keeps track of the validation performance after every epoch, and writes a model checkpoint taking notes of training statistics such as the loss over time, the norm of the gradients and all the informations that can be useful to assess the quality of the training. Then there is a second program, *the master*, which launches or kills workers across a computing cluster, and may additionally inspect the checkpoints written by workers and plot their training statistics, etc.

We do not claim this work to be an exhaustive exploration of the space of the hyper-parameters of the ReSeg model, but a first attempt to have a good degree of sensitivity on the main parameters and be able to design more complex architectures in the future. The first obstacle to overcome is the fact that the number of hyper-parameters of the ReSeg architecture is quite high and the possible number of experiments quickly explodes. A study like this requires running thousand of experiments and so a computational power that is not commonly available in every research laboratory. For this reasons we need to trade-off the quality of the solution and the exploration of the space of the hyper-parameters so we decided to use a *greedy* approach for the design of the experiments. This of course implies reaching sub-optimal solutions but it allows to explore the space of the parameters focusing only on the most promising configurations of hyper-parameters and performing a reasonable number of experiments. Indeed, ReSeg has been recently proposed and its potentialities has never been fully explored. Our aim is to have a first understanding of the strengths and the weaknesses of the model, and figure out how to solve the issues involved in the semantic segmentation task in order to obtain acceptable results. Our work wants also to be an analysis of the performance of the model compared to the other state-of-the-art models.

The "greedy optimization procedure" can be summarized in the following steps:

1. First we fix most of the general hyper-parameters such as the optimization method and the weights initialization approach. This reduces the number of hyper-parameters to be explored, focusing only on the ones that are specific of the model and that give us an understanding of the expressiveness and the power of ReSeg.

2. In our first experiments we also fix parameters that are specific of the model, such as the kind of recurrent unit and the number of layers to further reduce the space of the hyper-parameters.

3. We modify the architecture one parameter at the time according to the results obtained from the experiments in order to improve the performance and to solve the issues that we discover during the analysis.

Now we are going to explain the motivations behind the choices of the hyper-parameters.

5.4.1 Optimization method

One of the most critical hyper-parameter when we train a model with a gradient based learning algorithm involves the choice of the optimization method. In the literature the stochastic gradient descent (SGD) or its variant *with momentum* are widely used in training deep neural networks but they require to correctly tune the learning rate (and the momentum parameter) so that the training does not get stuck after few iterations. There are several issues to take into account when we choose the optimization method:

- Choosing a proper learning rate can be difficult. A learning rate that is too small leads to painfully slow convergence, while a learning rate that is too large can hinder convergence and cause the loss function to fluctuate around the minimum or even to diverge.

- It is a good practice to choose a *learning rate schedule* to try to adjust the learning rate during the training by e.g. *annealing*, i.e. reducing the learning rate according to a pre-defined policy or when the change in objective between epochs falls below a threshold. These schedules and thresholds, however, have to be defined in advance so they do not adapt to the dataset's characteristics during the training procedure.

- We might not want that the same learning rate applies to all the parameter updates. Instead of updating all the parameters to the same extent, we prefer a *per-parameter learning rate*.

- Depending on the shape of the loss function, the optimization problem can be very hard. A key challenge in minimizing highly non-convex error functions is the presence of saddle points [87], i.e. points where one dimension slopes up and another slopes down. These saddle points are usually surrounded by a plateau of the same error, which makes it notoriously hard for SGD to escape, as the gradient is close to zero in all dimensions.

Among the many variants of the SGD algorithm that have been proposed to take into account these issues we decided to use the Adadelta algorithm [26] for our experiments. This method does not require to manually tune the learning rate so it allows us to avoid a lot of experiments to choose the correct value. Moreover Adadelta uses a different learning rate for every parameter of the model θ_i (i.e. the weights of each layer) so that each dimension has its own dynamic rate. The basic idea behind Adadelta is to adjust the learning rate *per-parameter* according to a smoothed sum of the previous gradients computed on a fixed window. Intuitively this means that frequently occurring features get a smaller learning rate (because the sum of their gradients is larger), and rare features get a larger learning rate. Adadelta showed to perform very well on the training of the ReSeg model rapidly converging to a good solution. A detailed explanation of the algorithm can be found in the original work of Matthew Zeiler [26].

Shuffling the dataset

One final but important detail on the optimization of the loss function regards the order in which we present the training data. Generally, we want to avoid providing the training examples in a meaningful order to our model as this may bias the optimization algorithm and lead to poor convergence. This is crucial in our case since the datasets are sampled from video sequences and the network might simply memorize the order in which we feed the samples. A good and efficient method to avoid this issue is to *randomly shuffle* the data before each epoch of training.

5.4.2 Choice of the recurrent unit

To address the task of semantic segmentation with recurrent networks the ReSeg model processes recursively patches of the input scanning the entire image in the four directions. These patches are processed sequentially by each recurrent layer by means of a learning procedure that involves many steps depending on the dimension of the patches and the dimension of the image in order to capture short and long range dependencies between the pixels of the image and learn both local and global features. As we described in the chapter 2 Recurrent Neural Networks suffer from two main issues called *vanishing and exploding gradient* that can make the training procedure hard or even impossible. In the case of semantic segmentation this is a main issue, in fact it is important that the model is able to model dependencies between pixels that are close together but also pixels that are far apart. To solve this problems LSTM and GRU architecture have been proposed showing their ability to model long-range dependencies.

GRUs have been proposed by Cho et Al. [48] and the capabilities of this kind of model have not been fully explored yet. According to the empirical evaluations made by Chung et. Al [66] there is not a clear winner between the GRU and the LSTM architectures. In many tasks both architectures yield comparable performance so tuning hyper-parameters like the hidden layer size is probably more important than picking the ideal architecture.

For our experiments we decided to use Gated Recurrent Units because they are simpler than LSTMs and have less parameters. Moreover the initial experiments performed by Visin et. Al [2] showed that this type of recurrent unit works well for the task of semantic segmentation and so we started out analysis from their results.

We have to acknowledge that the choice of GRU over LSTM unit can also be motivated by the fact that the datasets that we are using for testing the ReSeg model have a limited number of examples available for the training, so reducing the complexity of the model and restricting the number of parameters can reduce the risk of incurring in overfitting.

5.4.3 Stacked Recurrent layers

When we introduced the model in chapter 4 we described two different kind of layers that are present in our architecture: the *recurrent horizontal sublayer* and the *recurrent vertical sublayer*. These two kind of sublayers can be either stacked or concatenated together forming the recurrent layer or *ReSeg layer* that is in charge to learn both global and local features to be used for the segmentation task. For the experiments we decided to stack the sublayers in order to have more compact feature maps representation O i.e., the dimension of each element $o_{(i,j)} \in \mathbb{R}^{2U}$ instead of $o_{(i,j)} \in \mathbb{R}^{4U}$. Stacking the sublayers makes the network deeper and allows the model to capture higher-level pixel interactions and learn more complex representations.

5.4.4 Number of Recurrent layers

For our first analysis we use just 2 ReSeg layers. Of course stacking more layers is possible but this would require a bigger amount of training data in order to prevent the model from overfitting. At the time we started this work the only available datasets specific for semantic segmentation of a urban street scene were CamVid [3] and Daimler [14] but the number of available examples was very low, respectively 367 and 300 . Moreover the Daimler dataset has only 5 classes with highly unbalanced distribution, so for the first experiments we used only the CamVid dataset. At a later time, the Cityscapes dataset [4] has been released providing 5000 fine-annotated examples, allowing to test deeper and more complicated models, as presented in the Section 5.7 of this chapter.

5.4.5 Preprocessing

Before to be fed as input to the network, the input data can be preprocessed in order to prepare the data for the training. Common data preprocessing techniques include mean subtraction and normalization. In particular for our experiments we decided to test a particular type of preprocessing called *Local Contrast Normalization*.

Local contrast normalization (LCN)

Local contrast normalization (LCN) [120] is a preprocessing method that normalizes the contrast of an image in a non-linear way. Instead of performing a global normalization based on the range of values of the entire image, LCN operates on local patches of the image on a per pixel basis. The local normalization normalizes the mean and variance of an image around a local neighborhood. This is done by removing the mean of a neighborhood from a particular pixel and dividing by the variance of the pixel values where the mean and the variance are smoothed by a Gaussian weighting window.

$$g(x,y) = \frac{f(x,y) - m_f(x,y)}{\sigma_f(x,y)} \tag{5.1}$$

where $f(x,y)$ is a pixel of the image, and $m_f(x,y)$, $\sigma_f(x,y)$ are respectively the local mean and the local variance computed on a $9x9$ patch around the pixel. Local Contrast Normalization can be used to:

- correct non-uniform scene illumination reducing the dynamic range (e.g it increases the contrast in shadowed parts). In the urban street context, or more in general when the images are acquired in an outdoor environment, this can be of great help.
- highlight edges, ***which leads the network to learn better category shapes***.

5.4.6 Performance indexes

To compare the quantitative performance of the different ReSeg variants, we rely on three commonly performance measures used for semantic segmentation:

- The ***global accuracy*** (G) measures the percentage of pixels correctly classified in the dataset:

$$G = \frac{correctly\,predicted\,pixels}{total\,number\,of\,pixels} = \frac{TP}{TP + FN}, \tag{5.2}$$

  where TP, FP are the numbers of true positive, false positive pixels, respectively, determined over the whole test set.

- The ***class average accuracy*** (C) is the mean of the accuracy computed over all the classes:

$$C = \frac{1}{n} \sum_{c_i \in classes} G(c_i), \tag{5.3}$$

  where $G(c_i)$ is the global accuracy computed for each individual class.

- The ***mean Intersection over Union*** (IoU) is the hardest metric since it penalizes false positive predictions unlike class average accuracy. The IoU metric is computed for each class individually and then is averaged.

$$IoU = \frac{TP}{TP + FN + FP}. \tag{5.4}$$

5.5 First experiments on CamVid Dataset

The dataset that we choose for the initial experiments is the Cambridge-driving Labeled Video Database (CamVid) [3]. We used the same subset of 11 class categories as SegNet [5] for experimental analysis in. A small number of pixels were labelled as void, which do not belong to one of these classes and are ignored for the evaluation. The dataset is split into 367 training, 101 validation and 233 test images. To make our experimental setup the same as [5], we scaled all the images by a factor of 2 at 480×360 resolution.

5.5.1 Number of hidden units

The first experiments were designed in order to find the correct size of the hidden layers. These tests were fundamental in order to assess the capacity of the network for the task of semantic segmentation.

exp	Patch size	Hidden units	Global avg. (%)	Class avg. (%)	**Mean IoU (%)**	Sky	Building	Column-Pole	Road	Side-walk	Tree	Sign-Symbol	Fence	Car	Pedestrian	Bicyclist
1	$[2 \times 2] - [1 \times 1]$	[50, 50]	85.4	**64.6**	52.2	**96.3**	82.6	**41.4**	**97.8**	69.4	77.3	39.8	**19.9**	**86.7**	**58.6**	40.5
2	$[2 \times 2] - [1 \times 1]$	[50, 100]	85.6	62.2	51.5	95.0	83.8	32.3	98.3	68.8	**81.7**	36.7	13.8	82.7	50.7	40.1
3*	$[2 \times 2] - [1 \times 1]$	[100, 100]	**86.6**	63.5	**52.9**	94.2	**86.5**	**41.4**	97.4	**76.1**	80.5	34.4	10.7	81.8	51.0	**44.4**
4	$[2 \times 2] - [1 \times 1]$	[100, 200]	85.2	62.2	50.9	95.9	82.8	36.4	97.9	75.5	73.0	**40.9**	7.3	85.2	50.1	39.1
5	$[2 \times 2] - [1 \times 1]$	[200, 200]	79.9	52.5	41.6	93.1	73.5	20.2	97.2	61.0	70.0	21.5	6.5	87.2	33.8	13.2
6	$[2 \times 2] - [2 \times 2]$	[100, 100]	85.0	59.4	47.9	**94.2**	78.9	22.3	**97.3**	78.0	**83.3**	32.1	12.8	**87.3**	**48.7**	18.5
7	$[2 \times 2] - [2 \times 2]$	[100, 200]	**85.3**	**61.2**	**50.4**	93.6	**85.4**	26.9	96.5	**82.6**	70.4	**43.5**	**19.0**	81.1	40.8	33.3
8	$[2 \times 2] - [2 \times 2]$	[200, 200]	83.6	57.5	46.8	93.2	84.2	**27.3**	97.0	76.1	65.1	26.8	10.7	84.3	29.5	**38.5**
9	$[2 \times 2] - [2 \times 2]$	[100, 400]	82.2	51.4	42.9	91.0	78.0	17.6	97.1	67.3	83.5	13.3	3.6	78.0	19.7	16.3
10	$[3 \times 3] - [1 \times 1]$	[100, 100]	82.1	57.4	45.7	92.7	73.7	22.2	97.0	76.6	73.0	36.6	14.6	89.8	32.0	23.6

Table 5.1: Comparison of the quantitative results of the ReSeg basic *model* **with 2 recurrent layers and using different number of hidden units and patch size** *on CamVid dataset [3]. (*) indicates the best candidate configuration.*

In the table 5.1 we can see the comparison of the performance of the ReSeg model with different hidden layer sizes maintaining the same basic configuration for each experiment. We call ***ReSeg basic*** the model that uses 2 recurrent layers and the Linear fully-connected upsampling layer. In this set of initial experiments, in order to have a better understanding of the capacity of the model, we did not use any kind of data preprocessing or data augmentation.

All the model variants have been trained using a mini-batch size of 1 example, corresponding to the classic stochastic gradient descent setting.

We tested several configuration for the number of neurons of the hidden layers and the dimension of the patches.

First we fixed the patch size of the first ReNet layer to be 2×2: this has the role to down-sample the input reducing the computational cost and the training time without discarding any information, since each patch is squashed on the channel dimension and processed by the ReNet layer. The second hidden layer scans the input pixel by pixel preserving its original size. Then we designed different experiments varying the size of the hidden layers in order to find the correct dimension.

Increasing the dimension of the patch of the second ReNet layer (Experiments 6-9) seems not to add any benefit. On the contrary further reducing the dimension of the feature maps makes difficult to reconstruct the original input size and degrades the quality of the final segmentation mask.

We decided to choose as best candidate for the dimension of the

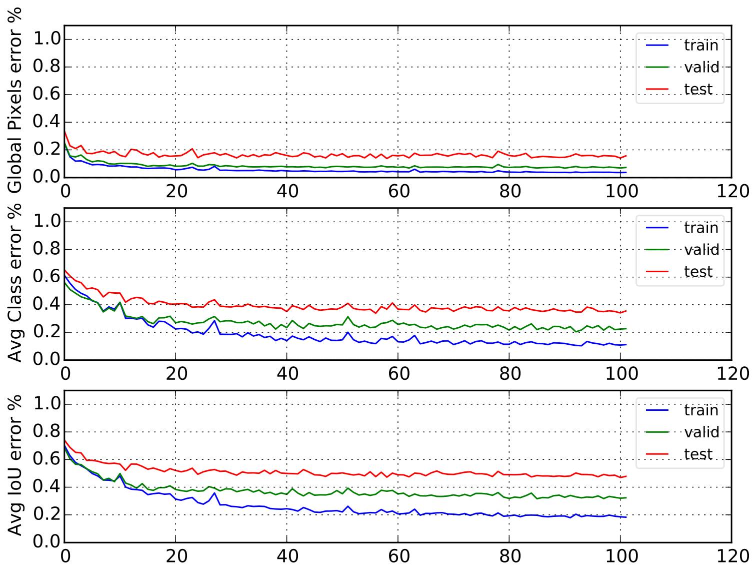


Figure 5.1: Plot of the training, validation and test error for all the performance indexes in **Experiment 1**.

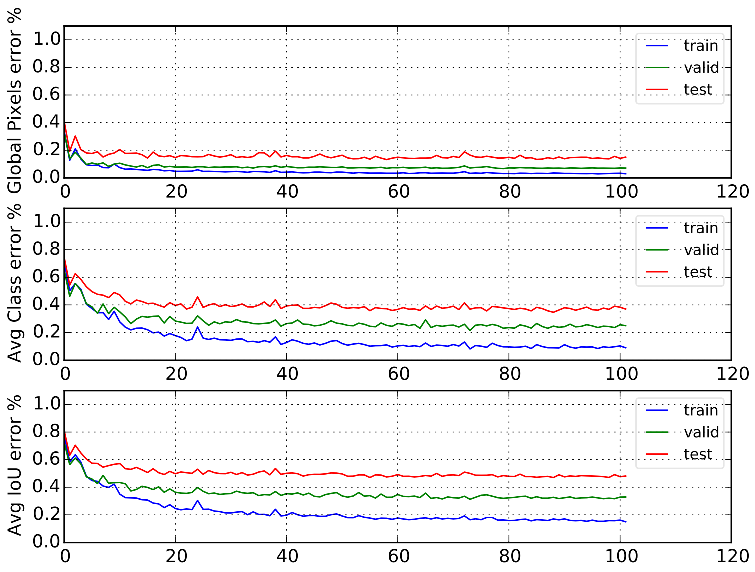


Figure 5.2: Plot of the training, validation and test error for all the performance indexes in **Experiment 2**.

hidden layer the experiment configuration number 3, marked with (*) since it achieves the highest mean IoU index that is the reference performance for a good quality segmentation. Moreover looking at the plots in Figures 5.1 and 5.2 we can see that the experiment number 1 might underfit the dataset due to the size of the hidden layers being too low. A good initial compromise seems to be the (100, 100) configuration of experiment 3.

5.5.2 Size of the mini-batch

Choosing the mini-batch size B involves a trade-off between the convergence of the optimization procedure and the training time. When B increases we can take advantage of the parallelism and the efficient matrix-matrix multiplications provided by the GPU instead of computing separate matrix-vector multiplications. This often allows to gain a good improvement in the overall training time. On the other hand, as B increases, the number of updates per computation done decreases, which slows down convergence since less updates can be performed in the same computing time.

We tested a mini-batch size of 1, 5 and 9 using the best candidate architecture defined in the previous experiments. We did not explore bigger batch size because of the memory limit of the GPU.

Looking at the Table 5.2, it results that $B = 5$ in combination with the Adadelta algorithm is a good value for the mini-batch size improving the class average accuracy and the mean IoU index at the expense of the global accuracy. A larger mini-batch does not seems to improve the performance, probably requiring extra epoch iterations in order to achieve the same results of smaller mini-batches.

In particular we have to note that the dimension of the training set and the variability of the distribution of the CamVid dataset examples is very limited, for this reason the optimization may benefits by a noisier estimation of the gradient. The better test results observed with smaller B may be explained with a better exploration of the parameter space and a form of regularization both due to the “noise injected” by the gradient estimator [121].

Setup	Mini-batch	Layers	Patch size	Hidden units	Global avg. (%)	Class avg. (%)	**Mean IoU (%)**	Sky	Building	Column-Pole	Road	Side-walk	Tree	Sign-Symbol	Fence	Car	Pedestrian	Bicyclist
ReSeg basic	1	2	$[2 \times 2] - [1 \times 1]$	[100, 100]	86.6	63.5	52.9	94.2	**86.5**	41.4	97.4	**76.1**	80.5	34.4	10.7	81.8	51.0	44.4
ReSeg basic	**5***	2	$[2 \times 2] - [1 \times 1]$	[100, 100]	**85.6**	**65.5**	**54.0**	**95.6**	83.9	**43.9**	**98.3**	64.5	81.2	**41.9**	**13.6**	81.8	**65.9**	**49.5**
ReSeg basic	9	2	$[2 \times 2] - [1 \times 1]$	[100, 100]	84.3	60.7	48.7	94.0	83.0	31.6	**98.2**	53.0	**85.0**	31.6	12.1	**87.1**	58.8	32.7

Table 5.2: Comparison of the quantitative results of the ReSeg basic *model on CamVid dataset [3]* **with different mini-batch sizes**.

5.5.3 Class Balancing

The first thing that can be noticed by the results of the experiments in Tables 5.1 and 5.2 is that the average class accuracies of the classes *pole column*, *sign signal* and *fence* are quite low with respect to the other classes. This is due to the fact the frequencies of the classes over the pixels are highly *imbalanced*. In the semantic segmentation task this is a known issue indeed. Taking as example the urban street context, objects like *buildings* or *cars* are much more frequent in the scene with respect to *poles* and *sign signals*. Moreover the quantity of pixels annotated as building is much more than the pixels annotated as poles or sign signal due to the dimension in the image. *Road*, *sky*, *building* pixels are approximately 40-50 times more than *pedestrian*, *poles*, *sign symbols*, *cars*, *bicyclists* in the dataset making it very challenging to label smaller categories. This pushes the network to prefer to minimize the loss by learning to classify the high-occurrence classes really well and to ignore the low-occurrence ones since they have less effect on the loss function.

If we have to perform a classification on an unbalanced dataset, a typical solution would be to augment the dataset replicating the samples for the low-occurrence class. In a semantic segmentation task this is not possible since each image does not correspond to a single prediction like it happens in image classification, but we have a prediction for each pixel. In this case replicating low-occurrence samples would require to replicate some patches of the images, but this is not feasible since there is a structure among pixels that can't be ignored, so augmenting the dataset should be done by hand and this would require the same or more time that would require collecting more data.

An alternative solution to this issue is to work directly on the loss function and train a more balanced version of our model by re-weighting each class in the cross-entropy loss. The loss function takes the follow-

Setup	Mini-batch	Global avg. (%)	Class avg. (%)	**Mean IoU (%)**	Sky	Building	Column-Pole	Road	Side-walk	Tree	Sign-Symbol	Fence	Car	Pedestrian	Bicyclist
ReSeg basic	1	86.6	63.5	52.9	94.2	86.5	41.4	97.4	76.1	80.5	34.4	10.7	81.8	51.0	44.4
ReSeg basic + Class Balancing	1	84.5	**71.9**	50.2	92.1	75.5	**53.8**	96.8	**81.9**	79.0	**58.1**	**30.2**	83.7	**78.3**	**61.7**
ReSeg basic	5	85.6	65.5	**54.0**	**95.6**	83.9	43.9	**98.3**	64.5	81.2	41.9	13.6	81.8	65.9	49.5
ReSeg basic + Class Balancing *	5	**87.4**	**72.7**	**54.8**	93.4	**84.5**	**55.4**	96.0	**82.7**	**83.0**	**53.9**	**23.3**	**86.6**	**83.1**	**57.4**

Table 5.3: Comparison of the quantitative results of the ReSeg basic *model* **with class balancing** *on CamVid dataset [3]. All the experiments share the same basic setup: 2 ReSeg layers with [100, 100] hidden units, and patch size* $[2 \times 2] - [1 \times 1]$. *(*) indicates the best configuration.*

ing form:

$$\mathcal{L}(C, C^*) = -\frac{1}{n} \sum_i \alpha(C_i^*) C_i^* \log(C_i) \tag{5.5}$$

where $C_i = e^{z_i} / \sum_c e^{z_{i,c}}$ is the class prediction at pixel i given the output z of the Softmax layer at the end of the ReSeg network, and C^* is the ground truth of the segmentation mask.

In particular we use a *median frequency balancing* [10] where the weight assigned to each pixel in the loss function is the ratio of the median of the class frequencies computed on the entire training set divided by the class frequency:

$$\alpha(c) = median\_frequency / f(c) \tag{5.6}$$

where $f(c)$ is the number of pixels of class c divided by the total number of pixels in the images where c is present, and $median\_frequency$ is the median of these frequencies.

The class balancing implies that larger and/or more frequent classes in the training set have a weight smaller than 1 and the weights of the smallest classes are the highest trying to give more importance to the classes that are less represented in the dataset and penalizing the learning for the larger classes. We found class weighting to be important for semantic segmentation as it shown by the comparison of the experiments in Table 5.3.

On the same track of the previous experiments, we took the configurations on which we obtained the best results and we trained the same model with the class-balanced cross entropy obtaining the results in Table 5.3.

As we can see from Table 5.3 the performance of the class-balanced experiments have a boost in the average class accuracy of **more than**

8%. We improved the accuracy of the low-occurrence classes, in particular of the sign-symbol, the column pole and the pedestrian classes, **almost 10%**.

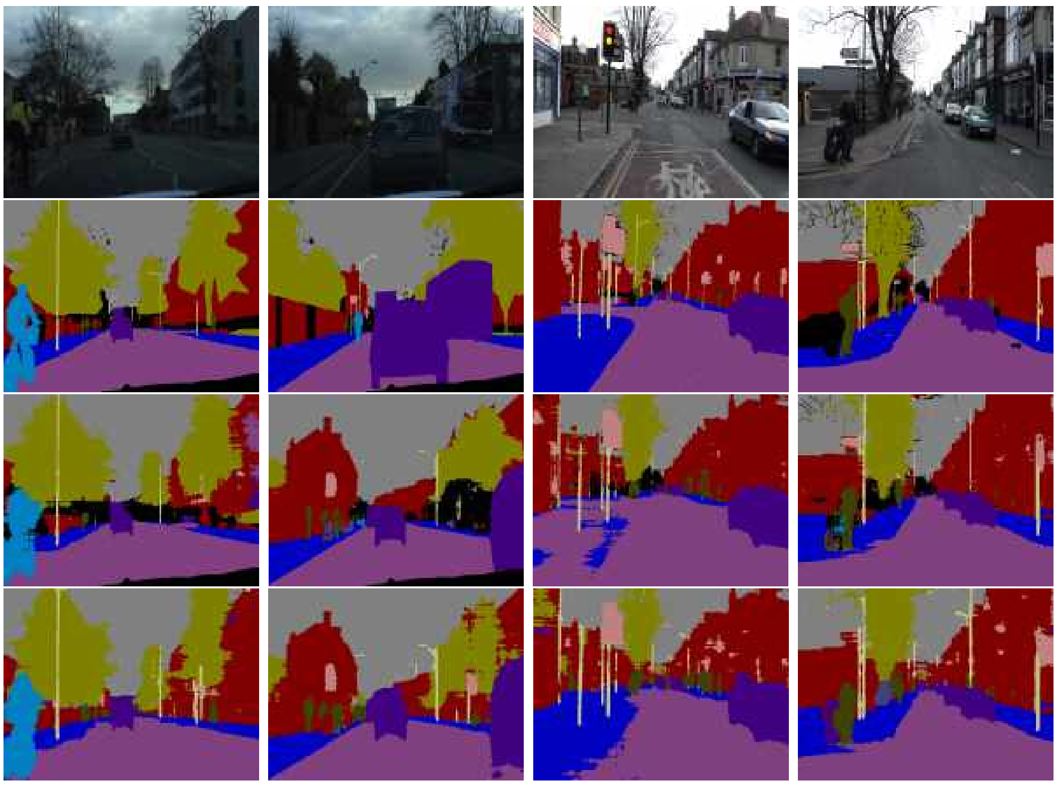


Figure 5.3: **ReSeg qualitative results on CamVid road scene understanding dataset [3].** *The top row is the input image, with the ground truth shown in the second row. The third row shows ReSeg's segmentation prediction, while the fourth row shows the class balanced predictions. In general, we observe high quality segmentation, especially on more difficult classes such as poles, people and cyclists. In the last example we see that the model fails to classify the sidewalk and many sign symbols, but the class balanced version is more accurate.*

On the other hand the mean IoU index does not benefit of the introduction of the class balancing. Indeed, the mean IoU is even 0.3% lower in the case of mini-batch size 1, and improves only by 0.8% using a mini-batch size equals to 5. The predicted segmentation masks in Figure 5.3 show that the low occurrence classes are actually more present in the output mask, but a lot of pixels are misclassified because of the aggressive policy of re-weighting. We can see that the class-balanced segmentation presents more *sign symbols* and *column poles* that are

Setup	Preprocessing	Global avg. (%)	Class avg. (%)	**Mean IoU (%)**	Sky	Building	Column-Pole	Road	Side-walk	Tree	Sign-Symbol	Fence	Car	Pedestrian	Bicyclist
ReSeg basic	-	86.6	63.5	52.9	94.2	86.5	41.4	97.4	76.1	80.5	34.4	10.7	81.8	51.0	44.4
ReSeg basic	LCN (mean sub)	84.8	65.1	52.6	94.7	81.5	36.8	97.4	68.9	80.3	42.8	34.3	78.1	53.5	47.9
ReSeg basic	LCN (mean sub + div std)	85.4	63.8	51.9	91.9	82.3	39.2	97.2	76.6	84.2	32.5	25.1	75.9	56.2	40.5
ReSeg basic + Class Balancing	-	84.5	71.9	50.2	92.1	75.5	53.8	96.8	81.9	79.0	58.1	30.2	83.7	78.3	61.7
ReSeg basic + Class Balancing	LCN (mean sub)	83.9	71.8	49.8	91.1	73.6	56.9	94.8	89.3	81.3	61.5	41.7	69.2	80.3	49.7

Table 5.4: Comparison of the quantitative results of the ReSeg basic *model* **with and without LCN preprocessing** *on CamVid dataset [3]. All the experiments share the same basic setup: 1 minibatch, 2 ReSeg layers with [100, 100] hidden units, and patch size* $[2 \times 2] - [1 \times 1]$.

correctly classified with respect to the unbalanced prediction. Also the *sidewalk* class benefits from the class weighting, that gives less importance to the *road* class. However the predicted masks are noisier since the probability of the underrepresented classes is overestimated and leads to an increase in misclassified pixels in the output segmentation mask.

We need to stress that the IoU metric is not optimized directly through the class balanced cross-entropy loss. The role of the class balancing is to bias the classification towards the low-occurrence class incrementing the probability that a given pixel is classified as one of those classes.

5.5.4 Adding Local Contrast Normalization

We decided to exclude the data preprocessing step from the first experiments in order to fully understand the ability of the model to assign the correct label simply by processing the raw data without any additional step. However it is useful to test the use of the preprocessing since it could improve the final results and speed the learning. In particular we want to understand if a data preprocessing step can help the network to understand the correct label category in unclear situations. In an outdoor environment such as the urban street context, it is frequent to have different illumination and weather conditions that can make hard the prediction task. We tested Local Contrast Normalization because it has the remarkable property to normalize local patches of the image instead of the entire image.

We experimented two version of the LCN preprocessing, the first one just subtracts the local (gaussian smoothed) mean of the local patch,

Method	Global avg. (%)	Class avg. (%)	**Mean IoU (%)**	Building	Tree	Sky	Car	Sign-Symbol	Road	Pedestrian	Fence	Column-Pole	Side-walk	Bicyclist
CRF models														
Super Parsing [76]	83.3	51.2	n/a	**87.0**	67.1	**96.9**	62.7	30.1	95.9	14.7	17.9	1.7	70.0	19.4
Boosting+Higher order [72]	83.8	59.2	n/a	84.5	72.6	**97.5**	72.7	34.1	95.3	34.2	45.7	8.1	77.6	28.5
Boosting+Detectors+CRF [73]	83.8	62.5	n/a	81.5	76.6	96.2	78.7	40.2	93.9	43.0	47.6	14.3	81.5	33.9
Neural Network based models														
SegNet-Basic (layer-wise training [106])	84.3	62.9	n/a	75.0	84.6	91.2	82.7	36.9	93.3	55.0	37.5	44.8	74.1	16.0
SegNet-Basic [5]	82.8	62.3	46.3	80.6	72.0	93.0	78.5	21.0	94.0	62.5	31.4	36.6	74.0	42.5
SegNet [5]	88.6	65.9	50.2	**88.0**	**87.3**	92.3	80.0	29.5	**97.6**	57.2	**49.4**	27.8	**84.8**	30.7
ReSeg basic + Class Balance	87.4	72.7	**54.8**	84.5	83.0	93.4	**86.6**	**53.9**	96.0	**83.1**	23.3	**55.4**	82.7	**57.4**
Sub-model averaging														
Bayesian SegNet-Basic [98]	81.6	70.5	55.8	75.1	68.8	91.4	77.7	52.0	92.5	71.5	44.9	52.9	79.1	69.6
Bayesian SegNet [98]	86.9	76.3	63.1	80.4	85.5	90.1	86.4	67.9	93.8	73.8	64.5	50.8	91.7	54.6

Table 5.5: CamVid. The table reports the per-class accuracy, the average per-class accuracy, the global accuracy and the average intersection over union. The best values and the values within 1 *point from the best are highlighted in bold for each column.*

while the second also normalizes by the local standard deviation. For all the experiments we use a patch size of 9×9 as suggested in the original paper [120].

From the results in Table 5.4 it seems that the model does not require any particular data preprocessing. This can be explained by the fact that ReSeg network exploits the correlation between the pixels while LCN does exactly the opposite, decorrelating the pixels in the same patch. Indeed, the combination of a subtractive and divisive LCN can be considered as a kind of approximate whitening.

The application of a preprocessing step combined with the ReSeg model require further investigation that is not the scope of this work. For this reason we no longer consider preprocessing in the next experiments.

5.6 Results on CamVid dataset

We can summarize our results so far and compare them with the state of the art models.

As reported in Table 5.5, our model exhibits state-of-the-art performance in terms of IoU when compared to both standard segmentation methods and neural network based methods, showing an increase of **4.6%** w.r.t. the recent SegNet model. It is worth highlighting that incorporating sub-model averaging to SegNet model, as in [98], boosts the original model performance, as expected. Therefore, introducing sub-model averaging to ReSeg would also presumably result in a signif-

Method	Global avg. (%)	Class avg. (%)	**Mean IoU (%)**
CRF models			
Super Parsing [76]	83.3	51.2	n/a
Boosting+Higher order [72]	83.8	59.2	n/a
Boosting+Detectors+CRF [73]	83.8	62.5	n/a
Neural Network based models			
SegNet-Basic (layer-wise training [106])	84.3	62.9	n/a
SegNet-Basic [5]	82.8	62.3	46.3
SegNet [5]	88.6	65.9	50.2
ReSeg basic + Class Balance	87.4	72.7	54.8
VGG-16 + ReSeg + Class Balance	88.1	72.5	**60.3**
Sub-model averaging			
Bayesian SegNet-Basic [98]	81.6	70.5	55.8
Bayesian SegNet [98]	86.9	76.3	63.1

Table 5.6: CamVid. The table reports the per-class accuracy, the average per-class accuracy, the global accuracy and the average intersection over union. The best values and the values within 1 *point from the best are highlighted in bold for each column.*

icant increase of the performance. However, this remains to be tested since it is not the core of this thesis.

We see that the proposed model can be used for semantic segmentation reaching state of the art performance compared to fully convolutional based models. What we can understand from the output prediction masks is that the network is able to make a correct inference in most of the situations but there are some cases in which the model fails.

Observing the CamVid dataset we see that most of the images have the same structure, i.e., the buildings, the sidewalks, the vegetation are usually on the left and right sides of the camera while the vehicles are in front of it. There are only few examples in which the structure of the scene changes and it happens when the vehicle on which it is

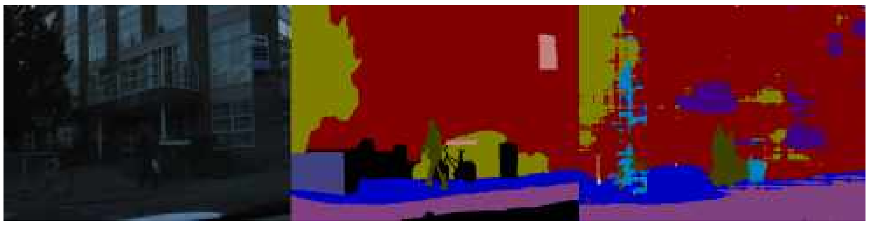


Figure 5.4: An example of test image that the network is not able to correctly predict

mounted the camera turns left or right (see Figure 5.4). In this cases it happens that the position of the buildings, the trees and the other classes in the scene are not anymore on the left or right, but in front of the camera. These cases are very hard to be predicted by the network since the amount of training examples with the classes in this kind of spatial configuration are very few.

This is important to notice because it seems that the network is able to learn a spatial prior on the distribution of the classes. This depends on the low variability of the training examples. The network is trained with most example having the objects in the scene with the same structure, so it learns to predict that spatial configuration since it expects the classes to be distributed in specific locations of the image.

Another example of misclassification is pointed out in Figure 5.5. Here the ReSeg network fails to label the car, however, it fills this part with buildings, trees and sidewalks that are very reasonable predictions due to the position of the bus in the scene.

We want to stress that these kind of artifacts and misclassification are not a limitation of our model, but they are mostly due to the poor quality of the dataset. After several experiments it seems evident that CamVid dataset is not sufficiently rich of examples to properly train deep neural networks models. The potentiality of deep models like ReSeg can be better exploited by using a dataset that fully represents the real world distribution of the classes. Using deep models with small datasets like CamVid can be useful to have a first understanding of the ability of the model to perform the task, but in order to achieve good generalization performance we need to train it with more examples.

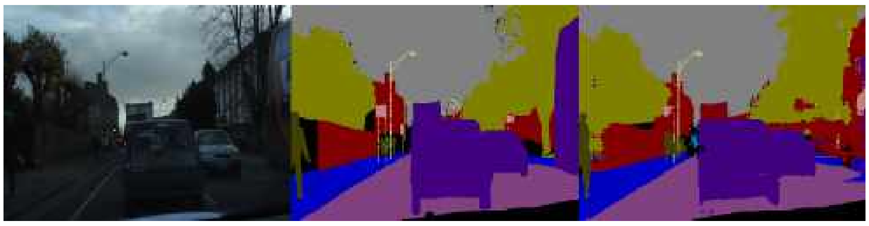


Figure 5.5: An example of misclassification filled with reasonable predictions.

5.7 Combining ReSeg with Convolutional Models

The first set of experiments on the CamVid dataset showed that a fully recurrent architecture such as ReSeg achieves comparable state of the art performance with respect to the convolutional models by exploring the spatial correlation between pixels in the images.

In this second batch of experiments we extend the ReSeg architecture by adding several convolutional layers before the ReNet layers. Convolutional networks proved their ability to extract general feature representations for many computer vision tasks, and they have been successfully applied to semantic segmentation, so it is natural to see if they can combine with recurrent neural networks in order to exploit the strenghts of both the architectures.

The ConvNet is in charge to extract a hierarchical feature representation. Then the recurrent layers combine the features extracted by the convolutional layers and model the long-term spatial dependencies. The last part of the architecture consists of the upsampling layers that restore the original size of the input. We tested both transposed convolutional and bilinear interpolation layers for this task showing comparable performances.

5.7.1 VGG-16

Several convolutional architectures have been presented in the last years, mostly for Image recognition and classification. Some of the most famous are known with the name of LeNet [122], AlexNet [38], GoogLeNet [39] and VGGNet [91].

Many semantic segmentation architectures such as SegNet [5], the works of Long et. Al [9] and Chen et. Al [99] are based on VGGNet or one of its derivations. In fact, although it was initially presented for image classification tasks, it was later found the VGG ConvNet features outperform those of GoogLeNet in multiple transfer learning tasks. For instance, Long et. Al [9] compared AlexNet, GoogLeNet and VGG-16 finding that VGG-16 achieves the best performance for the semantic segmentation. For this reason the VGG-16 network is currently one of the most preferred CNN-based choice to extract features from images, hence we decide to adopt it in our work as well.

The VGG-16 network is an homogeneous architecture that only performs 3×3 convolutions and 2×2 pooling from the beginning to the end. The input image passes through 5 groups of convolutional layers, *conv1* - *conv5*, and 5 max-pooling layers *pool1* - *pool5*, where the filters have a very small receptive field of 3×3, that is the smallest size in order to capture the notion of left/right, up/down, center. The Max-pooling is performed after each convolutional group, over a 2×2 pixel window, with stride 2.

The convolution stride of each convolutional layer is fixed to 1 pixel, as well as the spatial padding in order to preserve the spatial resolution after the 3×3 convolution operation. A stack of two 3×3 convolutional layers (without spatial pooling) has an effective receptive field of 5×5, while three such layers have a 7×7 effective receptive field.

Using a stack of three 3×3 convolutional layers instead of a single 7×7 convolutional layer has two important advantages. First, it incorporates three ReLU nonlinearities instead of a single one, second, it decreases the number of parameters from $C \times (7 \times 7 \times C) = 49C^2$ to $3 \times (C \times (3 \times 3 \times C)) = 27C^2$ where C is the number of channels.

Intuitively, stacking convolutional layers with small filters as opposed to having one convolutional layer with big filters allows to express more powerful features of the input, and with fewer parameters, at the expense of the memory required to hold all the intermediate results of the convolutional layers to compute backpropagation.

5.7.2 Adapting the VGG-16 network

We removed the fully connected layers from the original architecture, and we used the remaining layers which include 5 groups of convolutional layers { *conv1*, *conv2*, *conv3*, *conv4*, *conv5* } and 5 max-pooling layers { *pool1*, *pool2*, *pool3*, *pool4*, *pool5* }.

One of the biggest issues in adapting a convolutional architecture for image recognition to perform semantic segmentation is given by the fact that each max-pooling layer reduces the size of the feature map by a half compromising the quality of the prediction and leading to coarse segmentations. In the case of VGG-16, after all the max pooling layers, the size of the feature maps is reduced by a factor of 32.

Decreasing the stride of the pooling layers is the most straightforward way to obtain finer predictions, therefore we decrease the strides of the layers *pool4* and *pool5* from 2 to 1 in order to have a final downsampling factor of 8.

Then we stacked a pixelwise recurrent ReNet layer [1 × 1] that scans horizontally and vertically the features maps extracted by the convolutional layers and models long-term dependencies. The final architecture is depicted in Figure 5.6.

Batch Normalization

Batch Normalization (*BN* or *BatchNorm*) is a recently developed technique by Ioffe and Szegedy [123] for accelerating deep neural network learning. The authors suggest that the change in the distribution of the network activations due to the parameter updates slows the learning. This phenomenon is called *internal covariate shift* and can be addressed by explicitly forcing the activations throughout the network to take a unit gaussian distribution as the training progresses.

The batch normalization procedure $y = BN_{\gamma,\beta}(x)$ can be applied to any activation and consists in the normalization step followed by the scaling and shift step:

$$\hat{x}_i = \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \tag{5.7}$$

$$y_i = BN_{\gamma,\beta}(x_i) = \gamma \hat{x}_i + \beta \tag{5.8}$$

where γ and β are the scale and shift parameter to be learned, $\mu_{\mathcal{B}}$ and $\sigma_{\mathcal{B}}^2$ are respectively the mean and the variance estimated over the samples of the current mini-batch $\mathcal{B} = \{x_1, x_2, \dots\}$, and ϵ is a small constant to avoid numerical problems.

The core observation is that this is possible because normalization is a simple differentiable operation. It has been shown that in practice networks that use Batch Normalization are significantly more robust to bad initialization, moreover BatchNorm allows the use of higher learning rates. Batch Normalization can be interpreted as doing preprocessing at every layer of the network, but integrated into the backpropagation procedure.

In the implementation, applying this technique consists in inserting a BatchNorm layer immediately after each convolutional/upsampling layer and before the ReLU nonlinearities. We do not use Batch Normalization for the recurrent layers since it does not seem to give any improvement in the proposed implementation and its usefulness in the recurrent setting is still object of study [124].

Feature concatenation

Motivated by the prior works on object and semantic segmentation of Long et Al. [9], Bell et Al. [125] and Hariharan et Al. [126] we combined features extracted by different layers in order to achieve more discriminative features for semantic segmentation.

Decreasing the stride of the *pool4* and *pool5* layers from 2 to 1 plays the double role of reducing the overall downsampling factor of the network, and allow us to concatenate the feature maps from *pool3*, *pool4* and *pool5* layers. Indeed the feature maps from the pool layers now have the same size and can be concatenated before being fed as input to the recurrent layer(s). This specific kind of connections are called *skip-layer connections* and can be used to route directly a lower layer output to a higher layer input by-passing the intermediate layers.

It is important to notice that the scale of the feature values from different pooling layers may vary substantially, making it hard to directly combine them for the prediction. Then we need to normalize the feature maps before concatenation to avoid the final combination to be dominated by the strongest feature map. This has been firstly noted by Liu et Al. [127]. They found that L2 normalizing the features for each layer and combining them using a scaling factor learned through backpropagation works well. Since we adopt Batch-Normalization after each convolutional layer we decided to add a further BatchNorm layer after each pooling layer in order to normalize the features before combining them together.

ImageNet pretrained weights

In order to speed up the training we initialize the weights of each layer of the network with a pre-trained model on the ImageNet dataset [88] freely available for plug and play use. Specifically, we fixed the weights of the convolutional layers of the first 2 groups (*conv1* and *conv2*)

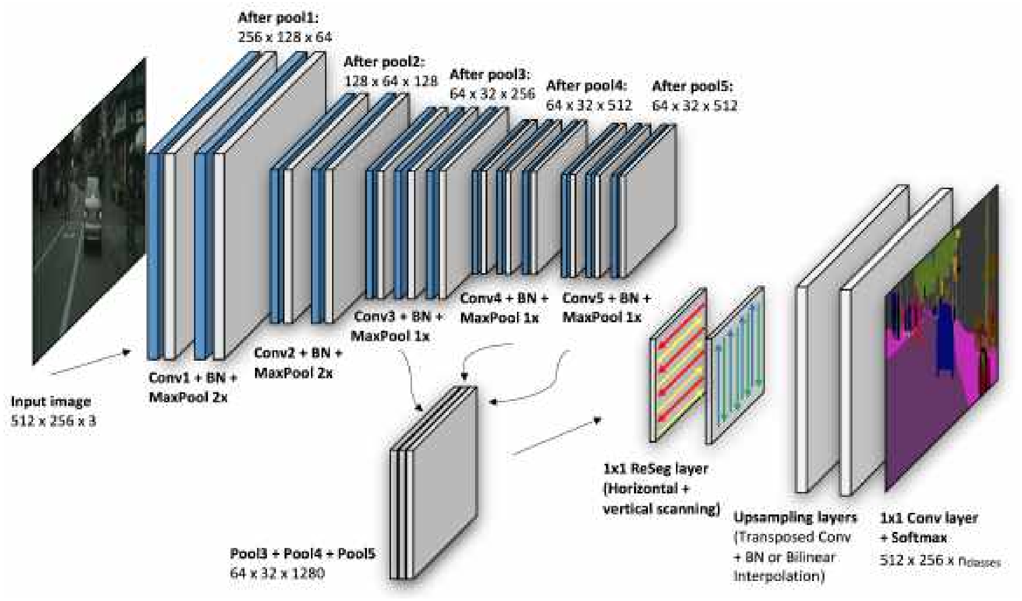


Figure 5.6: The final VGG-16 + ReSeg architecture

excluding them from the training procedure, and learning only the weights of the remaining layers. This is possible because the VGG-16 network extracts hierarchical feature representations that are more complex in the deeper layers. The first layers are in charge to extract very general features i.e, *edges*, so we can assume that they can be transferred from a dataset to another without applying any fine-tuning procedure. Excluding the first layers from the backward pass allows to reduce the time required for the training without affecting too much the performance of classification.

5.8 Experiments on Cityscapes Dataset

Cityscapes [4] is a large-scale dataset that contains a diverse set of stereo video sequences recorded in street scenes from 50 different cities, with high quality pixel-level annotations of 5 000 frames. The dataset is an order of magnitude larger than similar previous attempts such as CamVid [3] or Daimler [14], and it is specifically intended for supporting research on algorithm that aims to exploit large volumes of annotated data such as deep neural networks.

The images are provided at the high resolution of 2048×1024. In order to test the model in a reasonable amount of time we downsampled the images of a factor of 4 to a resolution of 512×256. We have to note that decreasing the size of the images reduces the overall quality of the segmentation. However the choice of using a downsampled version of the images is in accord with our initial goal of showing that the ReSeg model can be used in combination with VGG-16 increasing the performance of the convolutional model, and not to stress on achieving the best performance on the dataset. The Cityscapes dataset contains 19 classes, there are also a few pixels that are not annotated and are non considered in the evaluation.

The dataset provides 5000 examples split into 2975 training, 500 validation and 1525 test images. The ground truth is not available for the testing set; it is possible to assess the performance on the testing set submitting the predicted segmentation to the Cityscapes evaluation server. For this reason we evaluate our experiments only on the validation set and then we submitted to the evaluation server only the best results that we obtained on the validation set.

5.8.1 Design of the experiments and results

Thanks to Batch Normalization we are able to use higher learning rates and pay less attention to the initialization of the parameters. For this reason it has been possible to train the VGG-16+ ReSeg model with Adam [25] optimization algorithm, learning rate 10^{-3} and mini-batch size of 10 samples for all the experiments. For the evaluation we considered only the mean Intersection over Union (IoU) index since it is the reference metric of the benchmark suite proposed by the authors of Cityscapes.

First, we tested the plain VGG-16 model that consists only of convolutional and max-pooling layers, as described in Section 5.7.2.

Since the VGG-16 model downsamples the input image of an overall $\times 8$ factor, we added 3 Transposed Convolutional upsampling layers to restore the original resolution by gradually doubling the size of the feature maps. Then we stacked a last 1×1 convolutional layer in order to reduce the dimensionality of the feature maps to match the number of semantic categories before feeding it as input to the Softmax classifier.

By adding a recurrent 1×1 ReSeg layer after the VGG-16 network and before the upsampling layers, we improved the overall performance by ***2.8%*** on the mean IoU with respect to the plain VGG-16 model,

Setup	Concat-Features	**Mean IoU (%)**	Road	Sidewalk	Building	Wall	Fence	Pole	Traffic light	Traffic sign	Vegetation	Terrain	Sky	Person	Rider	Car	Truck	Bus	Train	Motorcycle	Bicycle
VGG-16	✓	52.1	91.9	64.5	81.4	27.2	27.8	35.8	32.8	45.9	85.3	47.8	86.9	**56.3**	33.0	85.2	31.3	50.8	31.2	22.7	**51.3**
VGG-16 + 1 ReSeg layer	✓	**54.9**	**92.7**	64.8	**82.9**	**36.8**	**33.8**	**38.1**	**31.5**	**44.8**	**86.3**	**49.0**	**89.8**	**56.3**	30.0	**86.7**	45.2	**59.4**	40.6	23.8	49.9
VGG-16 + 1 ReSeg layer	-	52.4	92.3	65.8	81.7	30.0	26.9	34.5	27.2	43.2	84.9	47.6	87.1	53.5	31.6	85.1	43.0	55.9	36.2	20.7	48.3
VGG-16 + 2 ReSeg layers	✓	**55.1**	92.3	**66.2**	82.8	26.9	**35.4**	36.3	**31.8**	**44.6**	85.3	45.7	87.6	**56.0**	**34.8**	**86.4**	**56.6**	61.6	**42.2**	**25.8**	49.2

Table 5.7: Comparison of the quantitative results of the VGG-16 + ReSeg model on the validation set of the Cityscapes dataset [4]

showing that the convolutional model benefits from the introduction of a recurrent layer that better models the long-term dependencies in different local area of the images.

We also compared the use of concatenated features from different max-pool layers with the use the output from the last max-pool layer (*pool5*) showing that the model actually benefits from the use of a combination of features with hierarchical complexity.

By adding a second recurrent 1×1 ReSeg layer we only improved the performance by 0.2% on the mean IoU. This shows the importance of finding the correct number of hidden layers, and the number of hidden units of each layer of the network, requiring a further investigation and an accurate analysis.

5.9 Results on Cityscapes dataset

We evaluated the best configuration on the testing set submitting our segmentations to the evaluation server. Although this is a preliminary work, our results are in line with the other state-of-the-art models that use a downsampling factor of $\times 4$. As discussed with the authors of Cityscapes, downsampling the input images severely reduces the quality of the segmentation especially of the smaller objects present in the scene. We expect that using the input images at their original size could improve the performance of the network at the expense of the training and inference time.

Method	**Mean IoU (%)**
Dilation10 [128]	67.1
Adelaide [129]	67.1
FCN 8s [9]	65.3
downsampling $\times 2$	
DeepLab LargeFOV StrongWeak [130]	64.8
VGG-16 + ReSeg	**64.5**
DeepLab LargeFOV Strong [99]	63.1
CRFasRNN [105]	62.5
downsampling $\times 3$	
DPN [131]	59.1
downsampling $\times 4$	
Segnet basic [5]	57.0
VGG-16 + ReSeg	**56.6**
Segnet extended [5]	56.1

Table 5.8: Comparison of the quantitative results of the best VGG-16 + ReSeg model on the test set of the Cityscapes dataset [4] with respect to the other state-of-the-art methods.

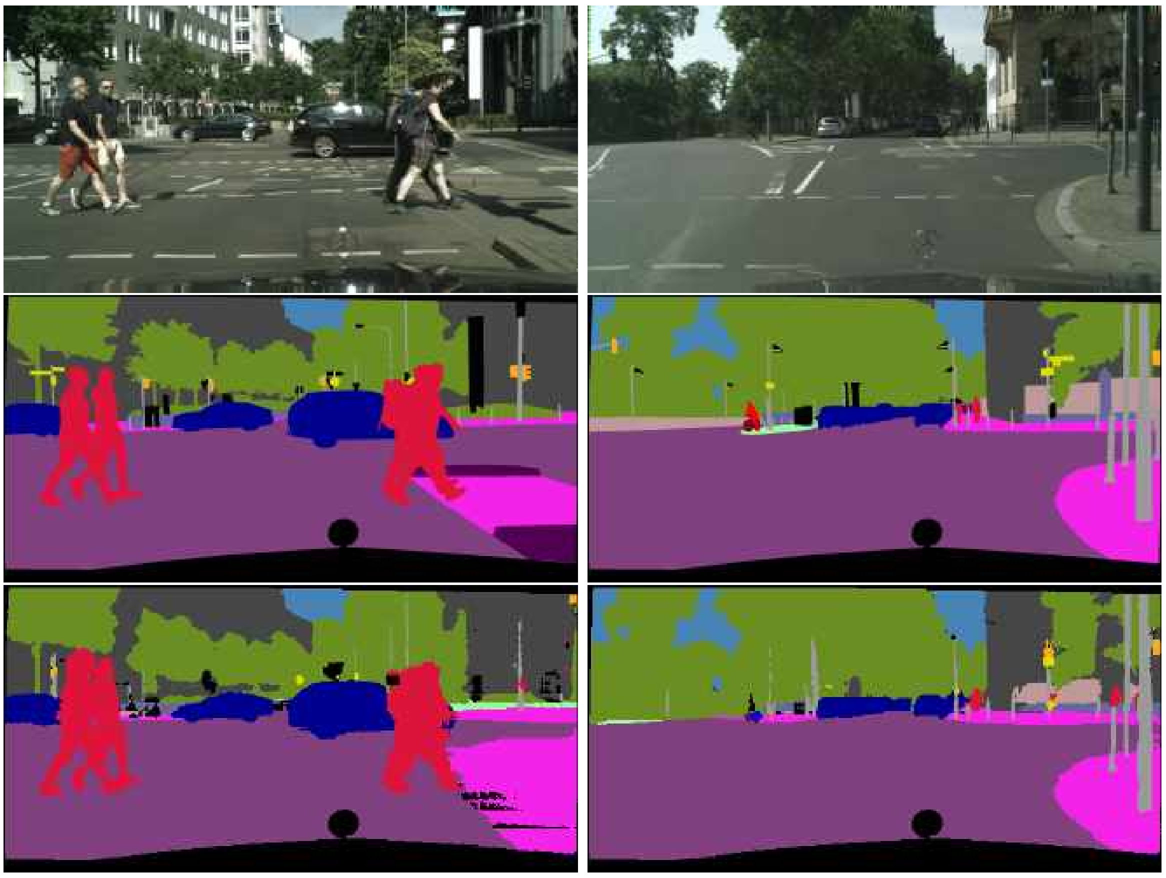


Figure 5.7: **VGG-16 + ReSeg qualitative results on Cityscapes dataset [4].** *The top row is the input image, with the ground truth shown in the second row. The third row shows the segmentation prediction.*

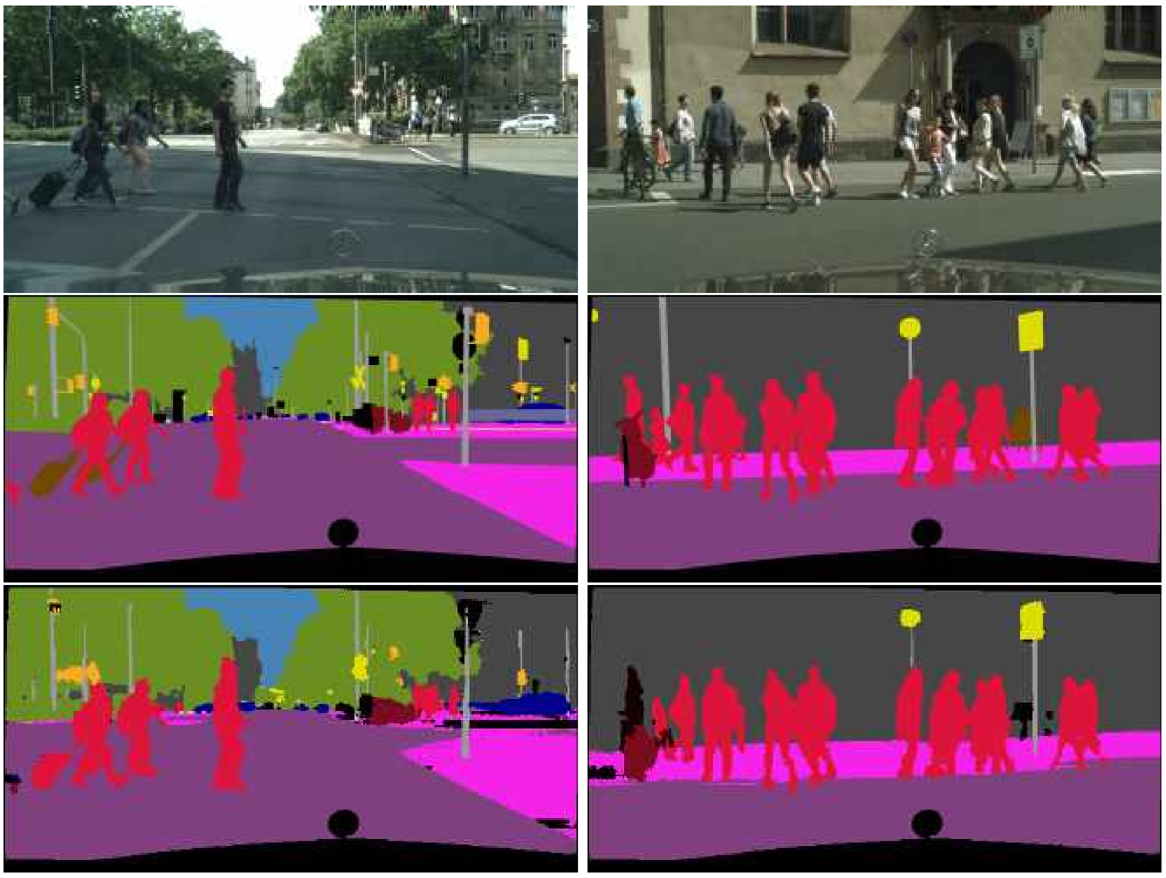


Figure 5.8: **VGG-16 + ReSeg qualitative results on Cityscapes dataset [4].** *The top row is the input image, with the ground truth shown in the second row. The third row shows the segmentation prediction.*

Chapter 6

Conclusion and future developments

In this work we focused on the *ReSeg* model proposed by Visin et Al. [2] for object segmentation. We implemented the ReSeg architecture using Lasagne [12], a lightweight library to build and train neural networks in Theano [13]. Then, we performed a greedy procedure for hyper-parameter optimization in order to have a first understanding of the strengths and the weaknesses of the model, and we extensively tested the network on challenging urban scene parsing datasets, achieving state-of-the-art performance on Camvid [3] and comparable to the state-of-the-art performance on Cityscapes [4] with respect to convolutional models.

We showed that recurrent neural networks are capable of capturing global and local contexts in the images exploiting the spatial correlation between pixels, and to learn a good feature representation that can be used for semantic segmentation. Finding the best architecture for semantic segmentation requires a more complete exploration of the hyper-parameter space, but our work traces a first path in the study of recurrent architectures for semantic image segmentation, showing that they can be a valid alternative to fully convolutional models.

Then we also combined the ReSeg model with several convolutional layers taken from the VGG-16 architecture [91] in order to exploit the best from both models. We proved that the two kind of models can be used together, and that the convolutional models can benefit from the introduction of recurrent layers such as ReSeg to improve the accuracy of the segmentation.

In our work we did not stress on reaching the best possible performance, but we rather focused on showing that the ReSeg model is a valid alternative to the convolutional architectures, or can be used in combination with them by exploiting the spatial correlation among the pixels to extract a better feature representation for semantic segmentation.

6.1 Future developments

6.1.1 Complete exploration of the hyper-parameter space

We presented a greedy procedure for the exploration of the hyper-parameter space, however, this is likely to lead to sub-optimal solutions. Many other combinations of hyper-parameter that probably achieve better performance have not been explored because of the time and computational resources that all the experiments would have required. For this reason we believe that our work can be considered a good starting point for further explorations of the hyper-parameter space using methods such as *bayesian optimization* in order to improve the results that we obtained. Moreover it is usually a good practice to apply *cross-validation* to each hyper-parameter in order to find their correct values and to achieve better generalization performance.

In all our experiments we tested the architecture fixing a random seed in order to reproduce the results. A better training procedure would involve the exploration of many different random seeds and to provide the average and the variance of the performance for each experiment configuration. This has not been done basically for computationally reasons.

Adadelta algorithm showed to be very effective thanks to its remarkable properties, nevertheless the choice of the optimization algorithm is crucial in order to reach a good minimum of the loss function. We showed that the use of Batch Normalization layers allows us to use other optimization methods such as Adam, and to pay less attention to the optimization parameters. However the exploration of other methods and an accurate tuning of the learning rate could lead to better solutions and performance.

6.1.2 Improve class balancing

We saw that Semantic Segmentation is inherently an unbalanced classification problem due to the difference of size among the objects present in the scene and also to the rarity of the instances of many classes. We showed that this problem can be partially solved by reweighting the loss function by proper weights that are estimated by taking into account the frequencies of the classes in the training set. This naive procedure showed good results, but the weights are not learned. A possible improvement would be to embed the class balancing into the training procedure and learn the weights that balance the class predictions.

A better approach could be to directly minimize the mean Intersection over Union index adding a penalization term in the loss function that takes into account the false positive classifications.

6.1.3 Adding inter-frame correlation and depth information

We showed how it is possible to exploit the spatial correlation between pixels to build a semantic segmentation system for urban scene parsing. The final goal of this kind of system is to be actually used in a real world environment, having a continuous stream of images coming from a camera that feeds the network in real time. For this reason it makes sense to exploit also the *inter-frame correlation* between subsequent frames of the video stream. Integrating the information coming from time correlation between frames can help to have a smooth and consistent segmentation mask in the temporal axis. This can be done, for instance, by conditioning the current output prediction of the network on the previous frame prediction.

Moreover, fusing the camera stream with the depth information coming from the LIDAR could surely improve the overall accuracy of the system adding robustness to the prediction in unclear situations.

A complete architecture that fuses together spatial, time and depth informations would improve the quality of the segmentation achieving accurate results that could be used for a real-world implementation on board of a self-driving car.

6.1.4 Integrating Fully Connected CRFs for finer segmentation

We showed that the use of ReSeg layers improve the performance of fully convolutional networks. However the segmentation masks that the model produces for the smallest objects in the scene are still coarse and blurry, due to the downsampled resolution of feature maps. To refine the boundaries and obtain finer segmentations we could integrate in the architecture a post-processing step that uses a fully-connected CRF layer. In particular, a fully connected CRF is a graphical model that connects all the pixels of the image, and that can be efficiently trained with an efficient inference algorithm as proposed by Krahenbuhl & Vladlen [78].

CRFs are usually adopted as post-processing step to improve the quality of the segmentations produced by other systems. A recent approach by Zheng et Al [105] shows that one iteration of the mean-field inference algorithm can be formulated as a stack of common CNN layers. Then the iterative mean-field inference can be seen as a Recurrent Neural Network (RNN) and the parameters can be learnt using the standard back-propagation-through-time algorithm [50]. In this setting, CRFs can be used in combination with our VGG-16 + ReSeg architecture and trained end-to-end using back-propagation. This allows to jointly train the network and the CRF instead of using a separate post processing procedure, hopefully achieving better performance.

6.1.5 Visualizing ReNet

It has been shown that Recurrent Neural Networks in the LSTM and GRU variants are very good at learning long-term dependencies in sequence learning tasks but it is not always easy to visualize what kind of features they actually learn. A very deep and interesting analysis and a comparison with other models has been done by Karpathy et Al [132] attempting to give an explanation on what the neurons and the gates are learning. Understanding what are the features that the network is learning is important in order to design an effective architecture and solve hard learning problems. The work of Karpathy et Al. relates to another famous work by Zeiler and Fergus [32] in which the authors show how to visualize the filters learned by a Convolutional Network and how each layer of the network learns more complex features going deep in the network.

It is reasonable to think that a similar work can be done for the ReNet layers applied to image recognition or semantic segmentation. Showing the visual features that the recurrent layers are learning would be crucial to understand the behavior of the architecture and could give a real perception on the power and the expressiveness of the network.

A first attempt to understand and visualize the features extracted by the ReNet layers has shown, as expected, that the first layer of the network extracts low level feature maps such as edge detectors in order to determine the contour of the objects, while the second layer extracts more complex and higher level feature maps presumably related to the particular semantic category of the objects.

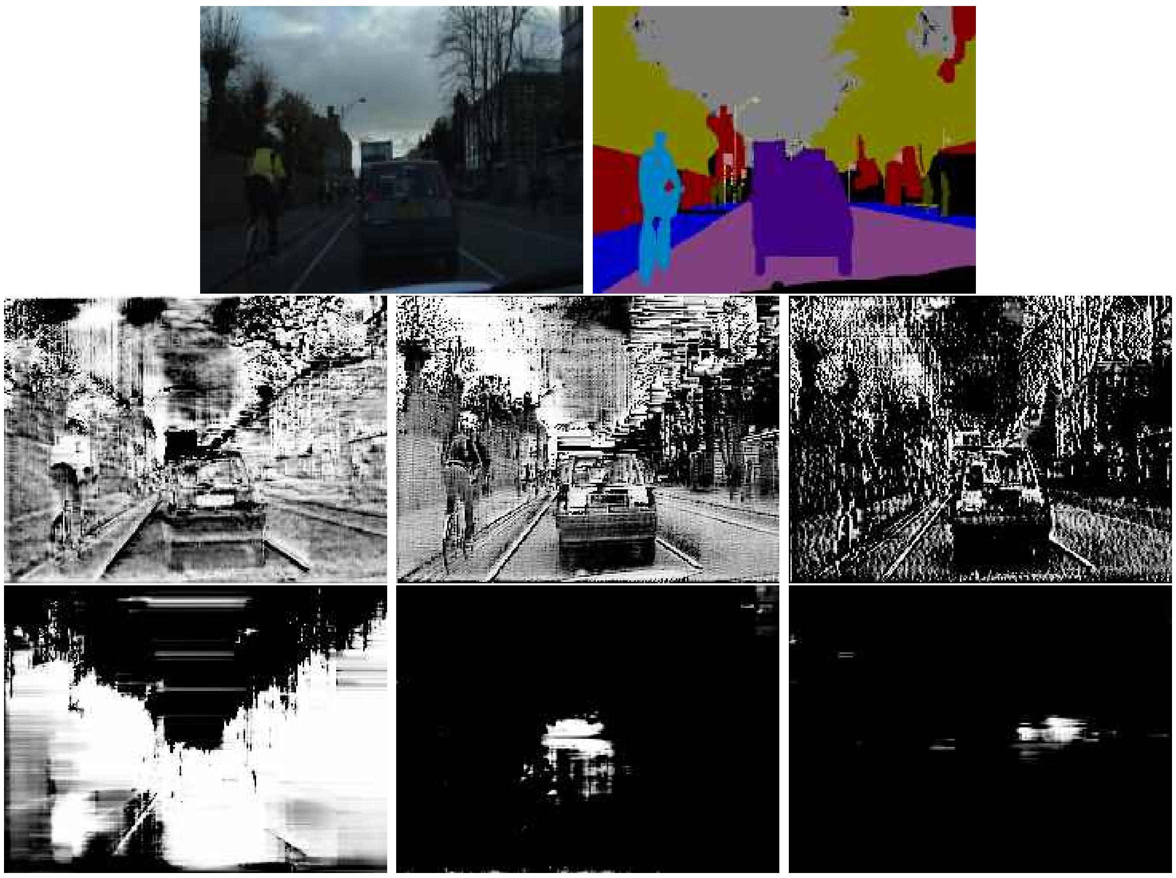


Figure 6.1: In the first row, the input image and the ground truth of an image of the Camvid dataset. The first and the second rows are respectively some examples of feature maps extracted by the first and the second layer of the ReSeg basic configuration.

Bibliography

[1] S. K. Sønderby and O. Winther, "Protein Secondary Structure Prediction with Long Short Term Memory Networks," 2014.

[2] F. Visin, K. Kastner, A. Courville, Y. Bengio, M. Matteucci, and K. Cho, "ReSeg: A Recurrent Neural Network for Object Segmentation," pp. 1–12, 2015.

[3] G. J. Brostow, J. Fauqueur, and R. Cipolla, "Semantic object classes in video: A high-definition ground truth database," *Pattern Recognition Letters*, vol. 30, no. 2, pp. 88–97, 2009.

[4] M. Cordts, M. Enzweiler, R. Benenson, S. Ramos, T. Scharw, U. Franke, S. Roth, D. A. G. R, T. U. Darmstadt, M. P. I. Informatics, and T. U. Dresden, "The Cityscapes Dataset," *Cvpr*, 2015.

[5] V. Badrinarayanan, A. Handa, and R. Cipolla, "SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation," p. 5, 2015.

[6] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484–503, 2016.

[7] NVIDIA, *NVIDIA Automotive Partners*, 2016. [`http://www.nvidia.it/object/automotive-partner-innovation-it.html`; Online; accessed 6-April-2016].

[8] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite,"

IEEE Conference on Computer Vision and Pattern Recognition, pp. 3354–3361, 2012.

[9] J. Long, E. Shelhamer, and T. Darrell, "Fully Convolutional Networks for Semantic Segmentation,"

[10] D. Eigen and R. Fergus, "Predicting Depth, Surface Normals and Semantic Labels with a Common Multi-Scale Convolutional Architecture," 2014.

[11] F. Visin, K. Kastner, K. Cho, M. Matteucci, A. Courville, and Y. Bengio, "ReNet: A Recurrent Neural Network Based Alternative to Convolutional Networks," pp. 1–9, 2015.

[12] S. Dieleman, J. Schluter, C. Raffel, E. Olson, S. K. Sonderby, D. Nouri, D. Maturana, M. Thoma, E. Battenberg, J. Kelly, J. D. Fauw, M. Heilman, diogo149, B. McFee, H. Weideman, takacsg84, peterderivaz, Jon, instagibbs, D. K. Rasul, CongLiu, Britefury, and J. Degrave, "Lasagne: First release.," Aug. 2015.

[13] F. Bastien, P. Lamblin, R. Pascanu, J. Bergstra, I. J. Goodfellow, A. Bergeron, N. Bouchard, and Y. Bengio, "Theano: new features and speed improvements." Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop, 2012.

[14] T. Scharwächter, M. Enzweiler, U. Franke, and S. Roth, "Stixmantics: a medium-level model for real-time semantic scene understanding," *Eccv*, pp. 533–548, 2014.

[15] F. Rosenblatt, "The perceptron: a perceiving and recognizing automaton," Tech. Rep. 85-460-1, Cornell Aeronautical Laboratory, 1957.

[16] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Neurocomputing: Foundations of research," ch. Learning Representations by Back-propagating Errors, pp. 696–699, Cambridge, MA, USA: MIT Press, 1988.

[17] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Netw.*, vol. 2, pp. 359–366, July 1989.

[18] Y. Bengio, "Learning Deep Architectures for AI," *Foundations and Trends in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009.

[19] N. Le Roux and Y. Bengio, "Deep belief networks are compact universal approximators.," *Neural computation*, vol. 22, no. 8, pp. 2192–2207, 2010.

[20] O. Delalleau and Y. Bengio, "Shallow vs. Deep Sum-Product Networks," *Neural Information Processing Systems (NIPS)*, pp. 666–674, 2011.

[21] R. Pascanu, G. Montufar, and Y. Bengio, "On the number of response regions of deep feed forward networks with piece-wise linear activations," *Nips*, pp. 1–17, 2013.

[22] G. Hinton, L. Deng, D. Yu, G. Dahl, A.-r. Mohamed, N. Jaitly, V. Vanhoucke, P. Nguyen, T. Sainath, and B. Kingsbury, "Deep Neural Networks for Acoustic Modeling in Speech Recognition," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[23] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv: 1207.0580*, pp. 1–18, 2012.

[24] Y. Bengio, I. J. Goodfellow, and A. Courville, "Deep learning." Book in preparation for MIT Press, 2015.

[25] D. P. Kingma and J. L. Ba, "Adam: a Method for Stochastic Optimization," *International Conference on Learning Representations*, pp. 1–13, 2015.

[26] M. D. Zeiler, "ADADELTA: An Adaptive Learning Rate Method," 2012.

[27] R. Pascanu, Y. N. Dauphin, S. Ganguli, and Y. Bengio, "On the saddle point problem for non-convex optimization," pp. 1–12, 2014.

[28] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," *Aistats*, vol. 9, pp. 249–256, 2010.

[29] K. He, X. Zhang, S. Ren, and J. Sun, "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification," *arXiv preprint*, pp. 1–11, 2015.

[30] V. Nair and G. E. Hinton, "Rectified Linear Units Improve Restricted Boltzmann Machines," *Proceedings of the 27th International Conference on Machine Learning*, no. 3, pp. 807–814, 2010.

[31] X. Glorot, A. Bordes, and Y. Bengio, "Deep Sparse Rectifier Neural Networks," *Aistats*, vol. 15, pp. 315–323, 2011.

[32] M. D. Zeiler and R. Fergus, "Visualizing and Understanding Convolutional Networks," *arXiv preprint arXiv:1311.2901*, 2013.

[33] G. E. Dahl, T. N. Sainath, and G. E. Hinton, "Improving deep neural networks for LVCSR using rectified linear units and dropout," *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, pp. 8609–8613, 2013.

[34] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, 1994.

[35] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," *Icml-2013*, vol. 28, 2013.

[36] B. Xu, N. Wang, T. Chen, and M. Li, "Empirical Evaluation of Rectified Activations in Convolutional Network," may 2015.

[37] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)," no. 1997, pp. 1–14, 2015.

[38] A. Krizhevsky, I. Sulskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," *Advances in Neural Information and Processing Systems (NIPS)*, pp. 1–9, 2012.

[39] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going Deeper with Convolutions," *arXiv preprint arXiv:1409.4842*, pp. 1–12, 2014.

[40] A. Karpathy and G. Toderici, "Large-scale video classification with convolutional neural networks," *. . . on Computer Vision . . .*, pp. 1725–1732, 2014.

[41] A. Karpathy and L. Fei-Fei, “Deep visual-semantic alignments for generating image descriptions,” *Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on*, pp. 3128–3137, 2015.

[42] D. Amodei, R. Anubhai, E. Battenberg, C. Carl, J. Casper, B. Catanzaro, J. Chen, M. Chrzanowski, A. Coates, G. Diamos, E. Elsen, J. Engel, L. Fan, C. Fougner, T. Han, A. Hannun, B. Jun, P. LeGresley, L. Lin, S. Narang, A. Ng, S. Ozair, R. Prenger, J. Raiman, S. Satheesh, D. Seetapun, S. Sengupta, Y. Wang, Z. Wang, C. Wang, B. Xiao, D. Yogatama, J. Zhan, and Z. Zhu, “Deep-speech 2: End-to-end speech recognition in English and Mandarin,” *arXiv*, p. 28, 2015.

[43] H. Sak, A. Senior, K. Rao, and F. Beaufays, “Fast and Accurate Recurrent Neural Network Acoustic Models for Speech Recognition,” *arXiv*, pp. 1468–1472, 2015.

[44] A. Graves, A.-r. Mohamed, and G. Hinton, “Speech Recognition With Deep Recurrent Neural Networks,” *Icassp*, no. 3, pp. 6645–6649, 2013.

[45] F. Beaufays, H. Sak, and A. Senior, “Long Short-Term Memory Recurrent Neural Network Architectures for Large Scale Acoustic Modeling Has,” *Interspeech*, no. September, pp. 338–342, 2014.

[46] N. Boulanger-Lewandowski, P. Vincent, and Y. Bengio, “Modeling Temporal Dependencies in High-Dimensional Sequences: Application to Polyphonic Music Generation and Transcription,” *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, no. Cd, pp. 1159–1166, 2012.

[47] D. Eck and J. Schmidhuber, “A First Look at Music Composition using LSTM Recurrent Neural Networks,” *Idsia*, 2002.

[48] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, “Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation,” 2014.

[49] H. T. Siegelmann and E. D. Sontag, “Turing computability with neural nets,” *Applied Mathematics Letters*, vol. 4, no. 6, pp. 77–80, 1991.

[50] P. J. Werbos, "Backpropagation through time: What it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.

[51] S. Hochreiter, "Untersuchungen zu dynamischen neuronalen Netzen," *Master's thesis, Institut fur Informatik, Technische Universitat, Munchen*, 1991.

[52] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," *International Conference on Machine Learning*, no. 2, pp. 1310–1318, 2013.

[53] S. Hochreiter and J. Schmidhuber, "Long short-term memory.," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[54] W. Zaremba, I. Sutskever, and O. Vinyals, "Recurrent Neural Network Regularization," *Icrl*, no. 2013, pp. 1–8, 2014.

[55] F. a. Gers and F. Cummins, "Learning To Forget," pp. 1–19, 1999.

[56] F. A. Gers and J. Schmidhuber, "Recurrent nets that time and count," in *Proceedings of the IJCNN'2000, Int. Joint Conf. on Neural Networks*, (Como, Italy), 2000.

[57] A. Graves and J. Schmidhuber, "Offline handwriting recognition with multidimensional recurrent neural networks," pp. 545–552, 2009.

[58] V. Pham, T. Bluche, C. Kermorvant, and J. Louradour, "Dropout improves Recurrent Neural Networks for Handwriting Recognition," 2013.

[59] P. Doetsch, M. Kozielski, and H. Ney, "Fast and robust training of recurrent neural networks for offline handwriting recognition," in *14th International Conference on Frontiers in Handwriting Recognition, ICFHR 2014, Crete, Greece, September 1-4, 2014*, pp. 279–284, 2014.

[60] A. Graves, "Generating sequences with recurrent neural networks," *arXiv preprint arXiv:1308.0850*, pp. 1–43, 2013.

[61] M.-T. Luong, I. Sutskever, Q. V. Le, O. Vinyals, and W. Zaremba, “Addressing the Rare Word Problem in Neural Machine Translation,” *Arxiv*, pp. 1–11, 2014.

[62] H. Sak, A. Senior, and F. Beaufays, “Long Short-Term Memory Recurrent Neural Network Architectures for Large Scale Acoustic Modeling,” *Interspeech 2014*, no. September, pp. 338–342, 2014.

[63] Y. Fan, Y. Qian, F. Xie, and F. K. Soong, “TTS synthesis with bidirectional LSTM based recurrent neural networks,” in *INTERSPEECH 2014, 15th Annual Conference of the International Speech Communication Association, Singapore, September 14-18, 2014*, pp. 1964–1968, 2014.

[64] E. Marchi, G. Ferroni, F. Eyben, L. Gabrielli, S. Squartini, and B. Schuller, “Multi-resolution linear prediction based features for audio onset detection with bidirectional LSTM neural networks,” *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing* - *Proceedings*, pp. 2164–2168, 2014.

[65] J. Donahue, L. A. Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, T. Darrell, U. T. Austin, U. Lowell, and U. C. Berkeley, “Long-term Recurrent Convolutional Networks for Visual Recognition and Description,” *Cvpr*, 2015.

[66] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, “Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling,” *arXiv*, pp. 1–9, 2014.

[67] M. Schuster and K. K. Paliwal, “Bidirectional recurrent neural networks,” *IEEE Transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, 1997.

[68] A. Graves and J. Schmidhuber, “Framewise phoneme classification with bidirectional lstm and other neural network architectures,” *Neural Networks*, pp. 5–6, 2005.

[69] J. Shotton, J. Winn, C. Rother, and A. Criminisi, “TextonBoost for image understanding: Multi-class object recognition and segmentation by jointly modeling texture, layout, and context,” *International Journal of Computer Vision*, vol. 81, no. 1, pp. 2–23, 2009.

[70] J. Shotton, M. Johnson, and R. Cipolla, “Semantic Texton Forest for Image Categorization and Segmentation,” *Proceedings of the conference on Computer Vision and Pattern Recognition*, pp. 1–8, 2008.

[71] J. Shotton, A. W. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake, “Real-time human pose recognition in parts from single depth images,” *Cvpr*, pp. 1297–1304, 2011.

[72] P. Sturgess, K. Alahari, L. Ladicky, and P. H. S. Torr, “Combining Appearance and Structure from Motion Features for Road Scene Understanding,” *Procedings of the British Machine Vision Conference 2009*, pp. 62.1–62.11, 2009.

[73] L. Ladický, P. Sturgess, K. Alahari, C. Russell, and P. H. S. Torr, “What, where and how many? Combining object detectors and CRFs,” *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 6314 LNCS, no. PART 4, pp. 424–437, 2010.

[74] P. Kontschieder, S. R. Bulò, H. Bischof, and M. Pelillo, “Structured class-labels in random forests for semantic image labelling,” *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2190–2197, 2011.

[75] C. Zhang, L. Wang, and R. Yang, “Semantic Segmentation of Urban Scenes Using Dense Depth Maps,” pp. 708–721, 2010.

[76] J. Tighe and S. Lazebnik, “Superparsing: Scalable nonparametric image parsing with superpixels,” *International Journal of Computer Vision*, vol. 101, no. 2, pp. 329–349, 2013.

[77] R. Girshick, J. Donahue, T. Darrell, U. C. Berkeley, and J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation,” *Cvpr'14*, pp. 2–9, 2014.

[78] P. Krähenbühl and V. Koltun, “Efficient inference in fully connected crfs with gaussian edge potentials,” *arXiv preprint arXiv:1210.5644*, pp. 1–9, 2012.

[79] L. Ladický, C. Russell, P. Kohli, and P. H. S. Torr, "Associative hierarchical random fields," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, no. 6, pp. 1056–1077, 2014.

[80] J. H. Kappes, B. Andres, F. A. Hamprecht, C. Schnörr, S. Nowozin, D. Batra, S. Kim, B. X. Kausler, T. Kröger, J. Lellmann, N. Komodakis, B. Savchynskyy, and C. Rother, "A Comparative Study of Modern Inference Techniques for Structured Discrete Energy Minimization Problems," *International Journal of Computer Vision*, vol. 115, no. 2, pp. 155–184, 2015.

[81] A. Rabinovich, A. Vedaldi, C. Galleguillos, E. Wiewiora, and S. Belongie, "Objects in context," *Proceedings of the IEEE International Conference on Computer Vision*, 2007.

[82] C. Galleguillos, a. Rabinovich, and S. Belongie, "Object categorization using co-occurence, location and appeearance," *Proc. CVPR*, 2008.

[83] C. Galleguillos, B. McFee, S. Belongie, and G. Lanckriet, "Multiclass object localization by combining local contextual interactions," in *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pp. 113 –120, June 2010.

[84] L. Ladicky, C. Russell, P. Kohli, and P. H. S. Torr, "Associative hierarchical CRFs for object class image segmentation," *Proceedings of the IEEE International Conference on Computer Vision*, pp. 739–746, 2009.

[85] L. Ladicky, C. Russell, P. Kohli, and P. H. S. Torr, "Graph cut based inference with co-occurrence statistics," in *Proceedings of the 11th European Conference on Computer Vision: Part V*, ECCV'10, (Berlin, Heidelberg), pp. 239–253, Springer-Verlag, 2010.

[86] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Süsstrunk, "SLIC superpixels compared to state-of-the-art superpixel methods," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 11, pp. 2274–2281, 2012.

[87] Y. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio, "Identifying and attacking the saddle point problem

in high-dimensional non-convex optimization," *arXiv*, pp. 1–14, 2014.

[88] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pp. 248–255, June 2009.

[89] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: Common Objects in Context," *arXiv preprint arXiv . . .*, vol. cs.CV, pp. 1–15, 2014.

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

[91] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *Iclr*, pp. 1–14, 2015.

[92] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition," *arXiv preprint arXiv . . .*, vol. cs.CV, pp. 346–361, 2014.

[93] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, "OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks," pp. 1–16, 2013.

[94] N. Zhang, J. Donahue, R. Girshick, and T. Darrell, "Part-based R-CNNs for for Fine-Grained Category Detection," pp. 834–849, 2014.

[95] C. Couprie, L. Najman, and Y. Lecun, "Learning Hierarchical Features for Scene Labeling," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 35, no. 8, pp. 1915–1929, 2013.

[96] S. Gupta, R. Girshick, P. Arbeláez, and J. Malik, "Learning Rich Features from RGB-D Images for Object Detection and Segmentation," *arXiv preprint arXiv:1407.5736*, pp. 1–16, 2014.

[97] B. Hariharan, P. Arbeláez, R. Girshick, and J. Malik, "Simultaneous Detection and Segmentation," *arXiv preprint arXiv ...*, vol. cs.CV, pp. 1–16, 2014.

[98] A. Kendall, V. Badrinarayanan, and R. Cipolla, "Bayesian SegNet: Model Uncertainty in Deep Convolutional Encoder-Decoder Architectures for Scene Understanding," 2015.

[99] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Semantic Image Segmentation with Deep Convolutional Nets and Fully Connected CRFs," *Iclr*, pp. 1–14, 2015.

[100] P. O. Pinheiro and R. Collobert, "Recurrent Convolutional Neural Networks for Scene Parsing," *arXiv preprint arXiv ...*, vol. cs.CV, no. June, 2013.

[101] P. O. Pinheiro and R. Collobert, "From Image-level to Pixel-level Labeling with Convolutional Networks," 2014.

[102] S. Gupta, P. Arbelaez, and J. Malik, "Perceptual organization and recognition of indoor scenes from RGB-D images," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 564–571, 2013.

[103] C. Farabet, C. Couprie, L. Najman, and Y. LeCun, "Scene Parsing with Multiscale Feature Learning, Purity Trees, and Optimal Covers," *arXiv preprint arXiv: ...*, p. 9, 2012.

[104] C. Couprie, "Indoor Semantic Segmentation using depth information," *Iclr*, pp. 1–8, 2013.

[105] S. Zheng, S. Jayasumana, B. Romera-Paredes, V. Vineet, Z. Su, D. Du, C. Huang, and P. H. S. Torr, "Conditional Random Fields as Recurrent Neural Networks," *International Conference on Computer Vision (ICCV)*, p. 16, 2015.

[106] V. Badrinarayanan, A. Handa, and R. Cipolla, "SegNet: A Deep Convolutional Encoder-Decoder Architecture for Robust Semantic Pixel-Wise Labelling,"

[107] H. Noh, S. Hong, and B. Han, "Learning Deconvolution Network for Semantic Segmentation," vol. 1.

[108] S. Hong, H. Noh, and B. Han, “Decoupled Deep Neural Network for Semi-supervised Semantic Segmentation,” *Nips*, pp. 1–9, 2015.

[109] G. J. Brostow, J. Shotton, J. Fauqueur, and R. Cipolla, “Segmentation and Recognition using Structure from Motion Point Clouds,” *Eccv*, vol. 5302, pp. 1–14, 2008.

[110] N. Kalchbrenner, I. Danihelka, and A. Graves, “Grid Long Short-Term Memory,” p. 14, 2015.

[111] M. D. Zeiler, G. W. Taylor, and R. Fergus, “Adaptive deconvolutional networks for mid and high level feature learning,” *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2018–2025, 2011.

[112] K. Simonyan, A. Vedaldi, and A. Zisserman, “Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps,” pp. 1–8, 2013.

[113] V. Dumoulin and F. Visin, “A guide to convolution arithmetic for deep learning,” pp. 1–28, 2016.

[114] M. Welling and D. P. Kingma, “Auto-Encoding Variational Bayes,” no. Ml, pp. 1–14, 2014.

[115] A. Graves, S. Fernández, and J. Schmidhuber, “Multi-dimensional recurrent neural networks,” *Artificial Neural Networks–ICANN 2007*, no. 1, pp. 549—-558, 2007.

[116] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio, “Theano: a CPU and GPU math expression compiler,” in *Proceedings of the Python for Scientific Computing Conference (SciPy)*, June 2010. Oral Presentation.

[117] E. Borenstein, E. Sharon, and S. Ullman, “Combining Top-Down and Bottom-Up Segmentation,” *Cvprw*, p. 46, 2004.

[118] K. Yamaguchi, M. H. Kiapour, L. E. Ortiz, and T. L. Berg, “Parsing clothing in fashion photographs,” *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, no. Fig 1, pp. 3570–3577, 2012.

[119] M. E. Nilsback and A. Zisserman, “A visual vocabulary for flower classification,” *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2, pp. 1447–1454, 2006.

[120] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun, “What is the best multi-stage architecture for object recognition?,” *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2146–2153, 2009.

[121] Y. Bengio, “Practical recommendations for gradient-based training of deep architectures,” *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 7700 LECTU, pp. 437–478, 2012.

[122] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2323, 1998.

[123] S. Ioffe and C. Szegedy, “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift,” *Arxiv*, 2015.

[124] C. Laurent, G. Pereyra, P. Brakel, Y. Zhang, and Y. Bengio, “Batch Normalized Recurrent Neural Networks,” pp. 1–9, 2015.

[125] S. Bell, C. L. Zitnick, K. Bala, and R. Girshick, “Inside-Outside Net: Detecting Objects in Context with Skip Pooling and Recurrent Neural Networks,” *Arxiv*, pp. 1–24, 2015.

[126] B. Hariharan, P. Arbel, and R. Girshick, “Hypercolumns for Object Segmentation and Fine-grained Localization,” *Cvpr*, pp. 447–456, 2015.

[127] W. Liu, A. Rabinovich, and A. C. Berg, “ParseNet: Looking Wider to See Better,” *arXiv preprint: arXiv:1506.04579*, pp. 1–11, 2015.

[128] F. Yu and V. Koltun, “Multi-Scale Context Aggregation by Dilated Convolutions,” pp. 1–9, 2015.

[129] G. Lin, C. Shen, I. Reid, and A. van dan Hengel, “Efficient piecewise training of deep structured models for semantic segmentation,” *Arxiv 2015*, pp. 1–13, 2015.

[130] G. Papandreou, L.-C. Chen, K. Murphy, and A. L. Yuille, “Weakly- and Semi-Supervised Learning of a DCNN for Semantic Image Segmentation,” *arXiv preprint*, p. 10, 2015.

[131] Z. Liu, X. Li, P. Luo, C. Change, and L. X. Tang, “Semantic Image Segmentation via Deep Parsing Network,” *Iccv*, pp. 1377–1385, 2015.

[132] A. Karpathy, J. Johnson, and F.-F. Li, “Visualizing and Understanding Recurrent Networks,” pp. 1–13, 2015.

Appendices

Appendix A

Submission to CVPR 2016 DeepVision Workshop

We include the paper that summarizes our results and we that submitted to the CVPR 2016 DeepVision Workshop with the title *"ReSeg: A Recurrent Neural Network-based Model for Semantic Segmentation"*.

ReSeg: A Recurrent Neural Network-based Model for Semantic Segmentation

Anonymous CVPR submission

Paper ID 22

Abstract

We propose a structured prediction architecture, which exploits the local generic features extracted by Convolutional Neural Networks and the capacity of Recurrent Neural Networks (RNN) to retrieve distant dependencies. The proposed architecture, called ReSeg, is based on the recently introduced ReNet model for image classification. We modify and extend it to perform the more challenging task of semantic segmentation. Each ReNet layer is composed of four RNN that sweep the image horizontally and vertically in both directions, encoding patches or activations, and providing relevant global information. Moreover, ReNet layers are stacked on top of pre-trained convolutional layers, benefiting from generic local features. Upsampling layers follow ReNet layers to recover the original image resolution in the final predictions. The proposed ReSeg architecture is efficient, flexible and suitable for a variety of semantic segmentation tasks. We evaluate ReSeg on several widely-used semantic segmentation datasets: Weizmann Horse, Oxford Flower, and CamVid; achieving state-of-the-art performance. Results show that ReSeg can act as a suitable architecture for semantic segmentation tasks, and may have further applications in other structured prediction problems.

1. Introduction

In recent years, Convolutional Neural Networks (CNN) have become the *de facto* standard in many computer vision tasks, such as image classification and object detection [22, 14]. Top performing image classification architectures usually involve *very* deep CNN trained in a supervised fashion on a large datasets [29, 40, 44] and have been shown to produce generic hierarchical visual representations that perform well on a wide variety of vision tasks. However, these deep CNNs heavily reduce the input resolution through successive applications of pooling or subsampling layers. While these layers seem to contribute significantly to the desirable invariance properties of deep CNNs, they also make it challenging to use these pre-trained CNNs for tasks such as semantic segmentation, where a per pixel prediction is required.

Recent advances in semantic segmentation tend to convert the standard deep CNN classifier into Fully Convolutional Networks (FCN) [31, 34, 2, 37] to obtain coarse image representations, which are subsequently upsampled to recover the lost resolution. However, these methods are not designed to take into account and preserve both *local* and *global* contextual dependencies, which has shown to be useful for semantic segmentation tasks [41, 16]. These models often employ Conditional Random Fields (CRFs) as a post-processing step to locally smooth the model predictions, however the long-range contextual dependencies remain relatively unexploited.

Recurrent Neural Networks (RNN) have been introduced in the literature to retrieve global spatial dependencies and further improve semantic segmentation [35, 16, 9, 8]. However, training spatially recurrent neural networks tends to be computationally intensive.

In this paper, we aim at the *efficient* application of Recurrent Neural Networks RNN to retrieve contextual information from images. We propose to extend the ReNet architecture [46], originally designed for image classification, to deal with the more ambitious task of semantic segmentation. ReNet layers can efficiently capture contextual dependencies from images by first sweeping the image horizontally, and then sweeping the output of hidden states vertically. The output of a ReNet layer is therefore implicitly encoding the local features at each pixel position with respect to the whole input image, providing relevant global information. Moreover, in order to *fully* exploit local and global pixel dependencies, we stack the ReNet layers on top of the output of a FCN, i.e. the intermediate convolutional output of VGG-16 [40], to benefit from generic local features. We validate our method on Weizmann Horse and Oxford Flower foreground/background segmentation datasets as a proof of concept for the proposed architecture. Then, we evaluate the performance in the standard benchmark of urban scenes CamVid; achieving state-of-the-art in all three datasets.

2. Related Work

Methods based on FCN tackle the information recovery (upsampling) problem in a large variety of ways. For instance, Eigen et al. [13] introduce a multi-scale architecture, which extracts coarse predictions, which are then refined using finer scales. Farabet et al. [15] introduce a multi-scale CNN architecture; Hariharan et al. [18] combine the information distributed over all layers to make accurate predictions. Other methods such as [31, 2] use simple bilinear interpolation to upsample the feature maps of increasingly abstract layers. More sophisticated upsampling methods, such as unpooling [2, 34] or deconvolution [31], are introduced in the literature. Finally, [37] concatenate the feature maps of the downsampling layers with the feature maps of the upsampling layers to help recover finer information.

RNN and RNN-like models have become increasingly popular in the semantic segmentation literature to capture long distance pixel dependencies [35, 16, 8, 42]. For instance, in [35, 16], CNN are unrolled through different time steps to include semantic feedback connections. In [8], 2-dimensional Long Short Term Memory (LSTM), which consist of 4 LSTM blocks scanning all directions of an image (left-bottom, left-top, right-top, right-bottom), are introduced to learn long range spatial dependencies. Following a similar direction, in [42], multi-dimensional LSTM are swept along different image directions; however, in this case, computations are re-arranged in a pyramidal fashion for efficiency reasons. Finally, in [46], ReNet is proposed to model pixel dependencies in the context of image classification. It is worth noting that one important consequence of the adoption of the ReNet spatial sequences is that they are even more easily parallelizable, as each RNN is dependent only along a horizontal or vertical sequence of pixels; i.e., all rows/columns of pixels can be processed at the same time.

3. Model Description

The proposed ReSeg model builds on top of ReNet [46] and extends it to address the task of semantic segmentation. The model pipeline involves multiple stages.

First, the input image is processed with the first layers of VGG-16 [40] network, pre-trained on ImageNet [11], where the number of VGG layers used to process the image varies from dataset to dataset, and is set such that the image resolution does not become too small. The resulting feature maps are then fed into one or more *ReNet layers* that sweep over the image. Finally, one or more *upsampling layers* are employed to resize the last feature maps to the same resolution as the input and a softmax non-linearity is applied to predict the probability distribution over the classes for each pixel.

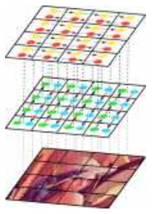


Figure 1. A ReNet layer

The recurrent layer is the core of our architecture and is composed by multiple RNN that can be implemented as a vanilla $\tanh$ RNN layer, a Gated Recurrent Unit (GRU) layer [10] or a LSTM layer [19]. Previous work has shown that the ReNet model can perform well with little concern for the specific recurrent unit used, therefore, we have chosen to use GRU units as they strike a good balance between memory usage and computational power.

In the following section we will define the recurrent and the upsampling layers in more detail.

3.1. Recurrent layer

As depicted in Figure 1, each recurrent layer is composed by 4 RNNs coupled together in such a way to capture the local and global spacial structure of the input data.

Specifically, we take as an input an image (or the feature map of the previous layer) $\mathbf{X}$ of elements $x \in \mathbb{R}^{H \times W \times C}$, where H, W and C are respectively the height, width and number of channels (or features) and we split it into $I \times J$ patches $p_{i,j} \in \mathbb{R}^{H_p \times W_p \times C}$. We then sweep vertically a first time with two RNNs $f^{\downarrow}$ and $f^{\uparrow}$, with U recurrent units each, that move top-down and bottom-up respectively. Note that the processing of each column is independent and can be done in parallel.

At every time step each RNN reads the next non-overlapping patch $p_{i,j}$ and, based on its previous state, emits a projection $o^{\star}_{i,j}$ and updates its state $z^{\star}_{i,j}$:

$$o^{\downarrow}_{i,j} = f^{\downarrow}(z^{\downarrow}_{i-1,j}, p_{i,j}), \text{ for } i = 1, \cdots, I \quad (1)$$

$$o^{\uparrow}_{i,j} = f^{\uparrow}(z^{\uparrow}_{i+1,j}, p_{i,j}), \text{ for } i = I, \cdots, 1 \quad (2)$$

We stress that the decision to read non-overlapping patches is a modeling choice to increase the image scan speed and lower the memory usage, but is not a limitation of the architecture.

Once the first two vertical RNNs have processed the whole input X, we concatenate their projections $o^{\downarrow}_{i,j}$ and $o^{\uparrow}_{i,j}$ to obtain a composite feature map $\mathbf{O}^{\updownarrow}$ whose elements $o^{\updownarrow}_{i,j} \in \mathbb{R}^{2U}$ can be seen as the activation of a feature detector at the location (i, j) with respect to all the patches in the

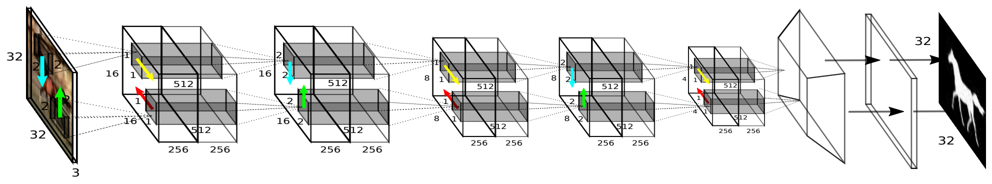


Figure 2. The ReSeg network. For space reasons we do not represent the pretrained VGG-16 convolutional layers that we use to preprocess the input to ReSeg.

j-th column of the input. We denote what we described so far as the *vertical recurrent sublayer*.

After obtaining the concatenated feature map $\mathbf{O}^{\updownarrow}$, we sweep over each of its rows with a pair of new RNNs, $f^{\rightarrow}$ and $f^{\leftarrow}$. We chose not to split $\mathbf{O}^{\updownarrow}$ into patches so that the second recurrent sublayer has the same granularity as the first one, but this is not a constraint of the model and different architectures can be explored. With a similar but specular procedure as the one described before, we proceed reading one element $o^{\updownarrow}_{i,j}$ at each step, to obtain a concatenated feature map $\mathbf{O}^{\leftrightarrow} = \left\{h^{\leftrightarrow}_{i,j}\right\}^{j=1\ldots J}_{i=1\ldots I}$, once again with $o^{\leftrightarrow}_{i,j} \in \mathbb{R}^{2U}$. Each element $o^{\leftrightarrow}_{i,j}$ of this *horizontal recurrent sublayer* represents the features of one of the input image patches $p_{i,j}$ *with contextual information from the whole image*.

It is trivial to note that it is possible to concatenate many recurrent layers $\mathbf{O}^{(\mathbf{1}\cdots\mathbf{L})}$ one after the other and train them with any optimization algorithm that performs gradient descent, as the composite model is a smooth, continuous function.

3.2. Upsampling layer

Since by design each recurrent layer processes non-overlapping patches, the size of the last composite feature map will be smaller than the size of the initial input $\mathbf{X}$, whenever the patch size is greater than one. To be able to compute a segmentation mask at the same resolution as the ground truth, the prediction should be expanded back before applying the softmax non-linearity.

Several different methods can be used to this end, e.g., fully connected layers, full convolutions and transposed convolutions. The first is not a good candidate in this domain as it does not take into account the topology of the input, which is essential for this task; the second is not optimal either, as it would require large kernels and stride sizes to upsample by the required factor. Transposed convolutions are both memory and computation efficient, and are the ideal method to tackle this problem.

Transposed convolutions – also known as *fractionally strided convolutions* – have been employed in many works in recent literature [49, 51, 32, 36, 20]. This method is based on the observation that direct convolutions can be expressed as a dot product between the flattened input and a sparse matrix, whose non-zero elements are elements of the convolutional kernel. The equivalence with the convolution is granted by the connectivity pattern defined by the matrix.

Transposed convolutions apply the transpose of this transformation matrix to the input, resulting in an operation whose input and output shapes are inverted with respect to the original direct convolution. A very efficient implementation of this operation can be obtained exploiting the gradient operation of the convolution – whose optimized implementation can be found in many of the most popular libraries for neural networks. For an in-depth and comprehensive analysis of each alternative, we refer the interested reader to [12].

4. Experiments

4.1. Datasets

We evaluated the proposed ReSeg architecture on several benchmark datasets. We proceeded by first assessing the performances of the model on the Weizmann Horse and the Oxford Flowers datasets and then focused on the more challenging Camvid dataset. We will describe each dataset in detail in this section.

4.1.1 Weizmann Horse

The Weizmann Horse dataset, introduced in [6], is an image segmentation dataset consisting of 329 variable size images in both RGB and gray scale format, matched with an equal number of groundtruth segmentation images, of the same size as the corresponding image. The groundtruth segmentations contain a foreground/background mask of the focused horse, encoded as a real-value between 0 and 255. To convert this into a boolean mask, we threshold in the center of the range setting all smaller values to 0, and all greater values to 1.

4.1.2 Oxford Flowers 17

The Oxford Flowers 17 class dataset from [33] contains 1363 variable size RGB images, with 848 image segmentations maps associated with a subset of the RGB images.

There are 8 unique segmentation classes defined over all maps, including flower, sky, and grass. To build a foreground/background mask, we take the original segmentation maps, and set any pixel not belonging to class 38 (flower class) to 0, and setting the flower class pixels to 1. This binary segmentation task for Oxford Flowers 17 is further described in [47].

4.1.3 CamVid Dataset

The Cambridge-driving Labeled Video Database (CamVid) [7] is a real-world dataset which consists of images recorded from a car with an internally mounted camera, capturing frames of 960×720 RGB pixels per frame, with a recording frame rate of 30 frames per second. A total of ten minutes of video was recorded, and approximately one frame per second has been manually annotated with per pixel class labels, from one of 32 possible classes. A small number of pixels were labelled as void in the original dataset. These do not belong to any of the 32 classes prescribed in the original data, and are ignored during evaluation. We used the same subset of 11 class categories as [2] for experimental analysis. The CamVid dataset itself is split into 367 training, 101 validation and 233 test images, and in order to make our experimental setup fully comparable to [2], we downsampled all the images by a factor of 2 resulting in a final 480×360 resolution.

4.2. Experimental settings

To gain confidence with the sensitivity of the model to the different hyperparameters, we decided to evaluate it first on the Weissman Horse and Oxford Flowers datasets on a binary segmentation task; we then focused the most of our efforts on the more challenging semantic segmentation task on the CamVid dataset.

The number of hyperparameters of this model is potentially very high, as for each ReNet layer different implementations are possible (namely vanilla RNN, GRU or LSTM), each one with its specific parameters. Furthermore, the number of features, the size of the patches and the initialization scheme have to be defined for each ReNet layer as well as for each transposed convolutional layer. To make it feasible to explore the hyperparameter space, some of the hyperparameters have been fixed by design and the remaining have been finetuned. In the rest of this section, the architectural choices for both sets of parameters will be detailed.

All the transposed convolution upsampling layers were followed by a ReLU [23] non-linearity and initialized with the fan-in plus fan-out initialization scheme described in [17]. The recurrent weight matrices were instead initialized to be orthonormal, following the procedure defined in [39]. We also constrained the stride of the upsampling transposed convolutional layers to be tied to their filter size.

In the segmentation task, each training image carries classification information for all of its pixels. Differently from the image classification task, small batch sizes provide the model with a good amount of information with sufficient variance to learn and generalize well. We experimented with various batch sizes going as low as processing a single image at the time, obtaining comparable results in terms of performance. In our experiments we kept a fixed batch size of 5, as a compromise between train speed and memory usage. In all our experiments, we used L2 regularization [24], also known as weight decay, set to 0.001 to avoid instability at the end of training. We trained all our models with the Adadelta [50] optimization algorithm, for its desired property of not requiring a specific hyperparameter tuning. The effect of Batch Normalization in RNNs has been a focus of attention [27], but it does not seem to provide a reliable improvement in performance, so we decided not to adopt it.

In the experiments, we varied the number of ReNet layers and the number of upsampling transposed convolutional layers, each of them defined respectively by the number of features $d_{\mathrm{RE}}(l)$ and $d_{\mathrm{UP}}(l)$, the size of the input patches (or equivalently of the filters) $ps_{\mathrm{RE}}(l)$ and $fs_{\mathrm{UP}}(l)$.

4.3. Results

In Table 1, we report the results on the Weizmann Horse dataset. On this dataset, we verified the assumption that processing the input image with some pre-trained convolutional layers from VGG-16 could ease the learning. Specifically, we restricted ourselves to only using up to 4 convolutional layers from VGG, as we only intended to extract some low-level generic features and learn the task-specific high-level features with the ReNet layers. The results indeed show an increase in terms of average Intersection over Union (IoU) when these layers are being used, confirming our hypothesis.

Table 2 shows the results for Oxford Flowers dataset, when using the full ReSeg architecture (i.e., including VGG convolutional layers). As shown in the table, our method clearly outperforms the state-of-the-art both in terms of global accuracy and average Intersection over Union (IoU).

Table 3 presents the results on CamVid dataset using the full ReSeg architecture. Our model exhibits state-of-the-art performance in terms of IoU when compared to both standard segmentation methods and neural network based methods, showing an increase of 4.6% w.r.t. to the recent SegNet model. It is worth highlighting that incorporating sub-model averaging to SegNet model, as in [21], boosts the original model performance, as expected. Therefore, introducing sub-model averaging to ReSeg would also presumably result in significant performance increase. However, this remains to be tested.

Method	Acc.	**Avg IoU**
All background baseline	74.7	0.0
All foreground baseline	25.4	79.9
ReSeg (no VGG)	94.9	79.9
Kernelized structural SVM [5]	94.6	80.1
ReSeg	93.3	**83.0**
CRF learning [30]	95.7	84.0
PatchCut [48]	95.8	84.0

Table 1. Weizmann Horses

Method	Acc.	**Avg IoU**
All background baseline	71.0	0.0
All foreground baseline	29.0	29.2
GrabCut [38]	95.9	89.3
Tri-map [47]	96.7	91.7
ReSeg	98.0	**93.7**

Table 2. Oxford Flowers

Method	Building	Tree	Sky	Car	Sign-Symbol	Road	Pedestrian	Fence	Column-Pole	Side-walk	Bicyclist	Avg class acc	Global acc	**Avg IoU**
Segmentation models														
Super Parsing [45]	**87.0**	67.1	**96.9**	62.7	30.1	95.9	14.7	17.9	1.7	70.0	19.4	51.2	83.3	n/a
Boosting+Higher order [43]	84.5	72.6	**97.5**	72.7	34.1	95.3	34.2	45.7	8.1	77.6	28.5	59.2	83.8	n/a
Boosting+Detectors+CRF [26]	81.5	76.6	96.2	78.7	40.2	93.9	43.0	47.6	14.3	81.5	33.9	62.5	83.8	n/a
Neural Network based segmentation models														
SegNet-Basic (layer-wise training [1])	75.0	84.6	91.2	82.7	36.9	93.3	55.0	37.5	44.8	74.1	16.0	62.9	84.3	n/a
SegNet-Basic [2]	80.6	72.0	93.0	78.5	21.0	94.0	62.5	31.4	36.6	74.0	42.5	62.3	82.8	46.3
SegNet [2]	**88.0**	**87.3**	92.3	80.0	29.5	**97.6**	57.2	**49.4**	27.8	**84.8**	30.7	65.9	88.6	50.2
ReSeg + Class Balance	84.5	83.0	93.4	**86.6**	**53.9**	96.0	**83.1**	23.3	**55.4**	82.7	**57.4**	72.7	87.4	**54.8**
Sub-model averaging														
Bayesian SegNet-Basic [21]	75.1	68.8	91.4	77.7	52.0	92.5	71.5	44.9	52.9	79.1	69.6	70.5	81.6	55.8
Bayesian SegNet [21]	80.4	85.5	90.1	86.4	67.9	93.8	73.8	64.5	50.8	91.7	54.6	76.3	86.9	63.1

Table 3. CamVid. The table reports the per-class accuracy, the average per-class accuracy, the global accuracy and the average intersection over union. The best values and the values within 1 point from the best are highlighted in bold for each column.

Model	ps_{RE}	d_{RE}	fs_{UP}	d_{UP}	Building	Tree	Sky	Car	Sign-Symbol	Road	Pedestrian	Fence	Column-Pole	Side-walk	Bicyclist	Class avg.	Global avg.	**Mean IoU**
ReSeg + LCN	$(2 \times 2), (1 \times 1)$	(100, 100)	(2×2)	(100, 100)	81.5	80.3	94.7	78.1	42.8	**97.4**	53.5	**34.3**	36.8	68.9	47.9	65.1	84.8	52.6
ReSeg	$(2 \times 2), (1 \times 1)$	(100, 100)	(2×2)	(100, 100)	83.9	81.2	**95.6**	81.8	41.9	**98.3**	65.9	13.6	43.9	64.5	49.5	65.5	85.6	**54.0**
ReSeg + Class Balancing	$(2 \times 2), (1 \times 1)$	(100, 100)	(2×2)	(100, 100)	**84.5**	**83.0**	93.4	**86.6**	**53.9**	96.0	**83.1**	23.3	**55.4**	**82.7**	**57.4**	**72.7**	**87.4**	**54.8**

Table 4. Comparison of the performance of different hyperparameter on CamVid.

5. Discussion

As reported in the previous section, our experiments on the Weizmann Horse dataset show that processing the input images with some layers of VGG-16 pre-trained network improves the results. In this setting, pre-processing the input with Local Contrast Normalization (LCN) does not seem to give any advantage (see Table 4). We did not use any other kind of pre-processing.

While on both the Weizmann Horse and the Oxford Flowers datasets we trained on a binary background/foreground segmentation task, on CamVid we addressed the full semantic segmentation task. In this setting, when the dataset is highly imbalanced, the segmentation performance of some classes can drop significantly as the network tries to maximize the score on the high-occurrence classes, *de facto* ignoring the low-occurrence ones. To overcome this behaviour, we added a term to the cross-entropy loss to bias the prediction towards the low-occurrence classes. We use *median frequency balancing* [13], which re-weights the class predictions by the ratio between the median of the frequencies of the classes (computed on the training set) and the frequency of each class. This increases the score of the low frequency classes (see Table 4) at the price of a more noisy segmentation mask, as the probability of the underrepresented classes is overestimated and can lead to an increase in misclassified pixels in the output segmentation mask, as shown in Figure 3 and Figure 4.

6. Conclusion

We introduced the ReSeg model, an extension of the ReNet model for image semantic segmentation. The proposed architecture shows state-of-the-art performances on CamVid, a widely used dataset for urban scene semantic

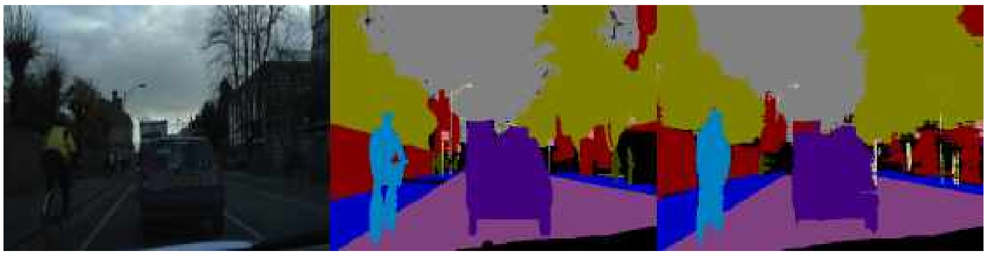


Figure 3. Camvid segmentation example without class balancing. From the left: input image, ground truth segmentation and ReSeg segmentation.

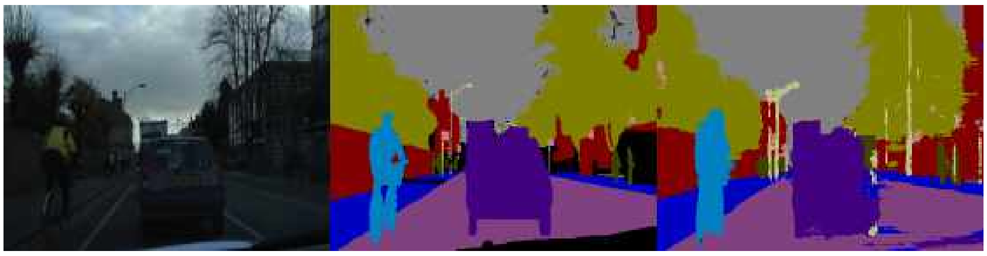


Figure 4. Camvid segmentation example with class balancing. From the left: input image, ground truth segmentation and ReSeg segmentation.

segmentation, as well as on the much smaller Oxford Flowers dataset. We also report near state-of-the-art performance on the Weizmann Horses.

In our analysis, we discuss the effects of applying some layers of VGG-16 to process the input data, as well as those of introducing a class balancing term in the cross-entropy loss function to help the learning of under-represented classes. Notably, it is sufficient to process the input images with just a few layers of VGG-16 for the ReSeg model to gracefully handle the semantic segmentation task, confirming its ability to encode contextual information and long term dependencies.

References

[1] V. Badrinarayanan, A. Handa, and R. Cipolla. SegNet: A Deep Convolutional Encoder-Decoder Architecture for Robust Semantic Pixel-Wise Labelling. 5

[2] V. Badrinarayanan, A. Handa, and R. Cipolla. SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation. page 5, 2015. 1, 2, 4, 5

[3] F. Bastien, P. Lamblin, R. Pascanu, J. Bergstra, I. Goodfellow, A. Bergeron, N. Bouchard, D. Warde-Farley, and Y. Bengio. Theano: new features and speed improvements. Submited to the Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop, 2012.

[4] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio. Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*, 2010.

[5] L. Bertelli, T. Yu, D. Vu, and B. Gokturk. Kernelized structural svm learning for supervised object segmentation. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 2153–2160. IEEE, 2011. 5

[6] E. Borenstein. Combining top-down and bottom-up segmentation. In *In Proceedings IEEE workshop on Perceptual Organization in Computer Vision, CVPR*, page 46, 2004. 3

[7] G. J. Brostow, J. Fauqueur, and R. Cipolla. Semantic object classes in video: A high-definition ground truth database. *Pattern Recognition Letters*, 30(2):88–97, 2009. 4

[8] W. Byeon, T. M. Breuel, F. Raue, and M. Liwicki. Scene labeling with lstm recurrent neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3547–3555, 2015. 1, 2

[9] L.-C. Chen, J. T. Barron, G. Papandreou, K. Murphy, and A. L. Yuille. Semantic image segmentation with task-specific edge detection using cnns and a discriminatively trained domain transform. *arXiv preprint arXiv:1511.03328*, 2015. 1

[10] K. Cho, B. van Merrienboer, C. Gulcehre, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Proceedings of the Empiricial Methods in Natural Language Processing (EMNLP 2014)*, Oct. 2014. to appear. 2

[11] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009. 2

[12] V. Dumoulin and F. Visin. A guide to convolution arithmetic for deep learning, 2016. cite arxiv:1603.07285. 3

[13] D. Eigen and R. Fergus. Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. *CoRR*, abs/1411.4734, 2014. 2, 5

[14] D. Erhan, C. Szegedy, A. Toshev, and D. Anguelov. Scalable object detection using deep neural networks. In *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '14, pages 2155–2162, Washington, DC, USA, 2014. IEEE Computer Society. 1

[15] C. Farabet, C. Couprie, L. Najman, and Y. LeCun. Learning hierarchical features for scene labeling. *IEEE TPAMI*, 35(8):1915–1929, 2013. 2

[16] C. Gatta, A. Romero, and J. van de Weijer. Unrolling loopy top-down semantic feedback in convolutional deep networks. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR Workshops 2014, Columbus, OH, USA, June 23-28, 2014*, pages 504–511, 2014. 1, 2

[17] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *International conference on artificial intelligence and statistics*, pages 249–256, 2010. 4

[18] B. Hariharan, P. Arbeláez, R. Girshick, and J. Malik. Hypercolumns for object segmentation and fine-grained localization. In *Computer Vision and Pattern Recognition (CVPR)*, 2015. 2

[19] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997. 2

[20] D. J. Im, C. D. Kim, H. Jiang, and R. Memisevic. Generating images with recurrent adversarial networks. *arXiv preprint arXiv:1602.05110*, 2016. 3

[21] A. Kendall, V. Badrinarayanan, and R. Cipolla. Bayesian SegNet: Model Uncertainty in Deep Convolutional Encoder-Decoder Architectures for Scene Understanding. 2015. 4, 5

[22] A. Krizhevsky, I. Sutskever, and G. Hinton. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25 (NIPS'2012)*. 2012. 1

[23] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012. 4

[24] A. Krogh and J. A. Hertz. A simple weight decay can improve generalization. In *ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS 4*, pages 950–957. Morgan Kaufmann, 1992. 4

[25] D. Kuettel and V. Ferrari. Figure-ground segmentation by transferring window masks. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 558–565. IEEE, 2012.

[26] L. Ladický, P. Sturgess, K. Alahari, C. Russell, and P. H. S. Torr. What, where and how many? Combining object detectors and CRFs. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 6314 LNCS(PART 4):424–437, 2010. 5

[27] C. Laurent, G. Pereyra, P. Brakel, Y. Zhang, and Y. Bengio. Batch normalized recurrent neural networks. *CoRR*, abs/1510.01378, 2015. 4

[28] A. Levin and Y. Weiss. Learning to combine bottom-up and top-down segmentation. *International Journal of Computer Vision*, 81(1):105–118, 2009.

[29] M. Lin, Q. Chen, and S. Yan. Network in network. In *Proceedings of the Second International Conference on Learning Representations (ICLR 2014)*, Apr. 2014. 1

[30] F. Liu, G. Lin, and C. Shen. Crf learning with cnn features for image segmentation. *Pattern Recognition*, 2015. 5

[31] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. *CVPR (to appear)*, Nov. 2015. 1, 2

[32] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3431–3440, 2015. 3

[33] M.-E. Nilsback and A. Zisserman. A visual vocabulary for flower classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 1447–1454, 2006. 3

[34] H. Noh, S. Hong, and B. Han. Learning deconvolution network for semantic segmentation. *arXiv preprint arXiv:1505.04366*, 2015. 1, 2

[35] P. Pinheiro and R. Collobert. Recurrent convolutional neural networks for scene labeling. *JMLR*, 1(32):82–90, 2014. 1, 2

[36] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015. 3

[37] O. Ronneberger, P.Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, volume 9351 of *LNCS*, pages 234–241. Springer, 2015. (available on arXiv:1505.04597 [cs.CV]). 1, 2

[38] C. Rother, V. Kolmogorov, and A. Blake. Grabcut: Interactive foreground extraction using iterated graph cuts. *ACM Transactions on Graphics (TOG)*, 23(3):309–314, 2004. 5

[39] A. M. Saxe, J. L. McClelland, and S. Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. In *Proceedings of the Second International Conference on Learning Representations (ICLR 2014)*, Apr. 2014. 4

[40] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015. 1, 2

[41] G. Singh and J. Kosecka. Nonparametric scene parsing with adaptive feature relevance and semantic context. In *2013 IEEE Conference on Computer Vision and Pattern Recognition, Portland, OR, USA, June 23-28, 2013*, pages 3151–3157, 2013. 1

[42] M. F. Stollenga, W. Byeon, M. Liwicki, and J. Schmidhuber. Parallel multi-dimensional lstm, with application to fast biomedical volumetric image segmentation. In *Advances in*

Neural Information Processing Systems, pages 2980–2988, 2015. 2

[43] P. Sturgess, K. Alahari, L. Ladicky, and P. H. S. Torr. Combining Appearance and Structure from Motion Features for Road Scene Understanding. *Procedings of the British Machine Vision Conference 2009*, pages 62.1–62.11, 2009. 5

[44] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *arXiv preprint arXiv:1409.4842*, 2014. 1

[45] J. Tighe and S. Lazebnik. Superparsing: Scalable nonparametric image parsing with superpixels. *International Journal of Computer Vision*, 101(2):329–349, 2013. 5

[46] F. Visin, K. Kastner, K. Cho, M. Matteucci, A. Courville, and Y. Bengio. Renet: A recurrent neural network based alternative to convolutional networks. *arXiv preprint arXiv:1505.00393*, 2015. 1, 2

[47] X. Wu and K. Kashino. Tri-map self-validation based on least gibbs energy for foreground segmentation. In *Proceedings of the British Machine Vision Conference*. BMVA Press, 2014. 4, 5

[48] J. Yang, B. Price, S. Cohen, Z. Lin, and M.-H. Yang. Patchcut: Data-driven object segmentation via local shape transfer. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1770–1778, 2015. 5

[49] M. Zeiler, G. Taylor, and R. Fergus. Adaptive deconvolutional networks for mid and high level feature learning. In *Proc. International Conference on Computer Vision (ICCV'11)*, pages 2146–2153. IEEE, 2011. 3

[50] M. D. Zeiler. ADADELTA: an adaptive learning rate method. Technical report, arXiv 1212.5701, 2012. 4

[51] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *ECCV'14*, 2014. 3