



ANOMALY DETECTION IN SURVEILLANCE VIDEOS USING DEEP RESIDUAL NETWORKS

Lucas Pinheiro Cinelli

Projeto de Graduação apresentado ao Curso de Engenharia Eletrônica e de Computação da Escola Politécnica, Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Engenheiro.

Orientadores: Eduardo A. B. da Silva

Lucas Arrabal Thomaz

Rio de Janeiro
Fevereiro de 2017

ANOMALY DETECTION IN SURVEILLANCE VIDEOS USING
DEEP RESIDUAL NETWORKS

Lucas Pinheiro Cinelli

PROJETO DE GRADUAÇÃO SUBMETIDO AO CORPO DOCENTE DO CURSO
DE ENGENHARIA ELETRÔNICA E DE COMPUTAÇÃO DA ESCOLA PO-
LITÉCNICA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO
PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU
DE ENGENHEIRO ELETRÔNICO E DE COMPUTAÇÃO

Autor:



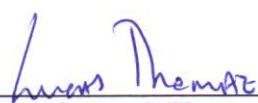
Lucas Pinheiro Cinelli

Orientador:



Prof. Eduardo A. B. da Silva, Ph. D.

Orientador:



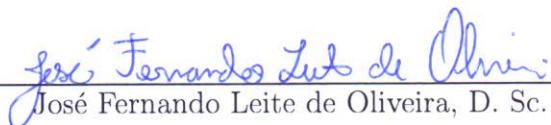
Lucas Arrabal Thomaz, M. Sc.

Examinador:



Prof. Sergio Lima Netto, Ph. D.

Examinador:



José Fernando Leite de Oliveira, D. Sc.

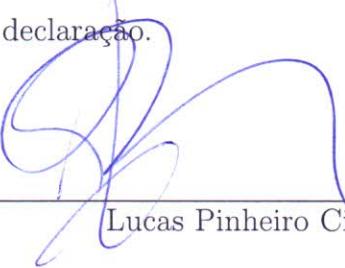
Rio de Janeiro

Fevereiro de 2017

Declaração de Autoria e de Direitos

Eu, Lucas Pinheiro Cinelli CPF 145.784.957-79, autor da monografia *Anomaly detection in surveillance videos using deep residual networks*, subscrevo para os devidos fins, as seguintes informações:

1. O autor declara que o trabalho apresentado na disciplina de Projeto de Graduação da Escola Politécnica da UFRJ é de sua autoria, sendo original em forma e conteúdo.
2. Excetuam-se do item 1. eventuais transcrições de texto, figuras, tabelas, conceitos e idéias, que identifiquem claramente a fonte original, explicitando as autorizações obtidas dos respectivos proprietários, quando necessárias.
3. O autor permite que a UFRJ, por um prazo indeterminado, efetue em qualquer mídia de divulgação, a publicação do trabalho acadêmico em sua totalidade, ou em parte. Essa autorização não envolve ônus de qualquer natureza à UFRJ, ou aos seus representantes.
4. O autor pode, excepcionalmente, encaminhar à Comissão de Projeto de Graduação, a não divulgação do material, por um prazo máximo de 01 (um) ano, improrrogável, a contar da data de defesa, desde que o pedido seja justificado, e solicitado antecipadamente, por escrito, à Congregação da Escola Politécnica.
5. O autor declara, ainda, ter a capacidade jurídica para a prática do presente ato, assim como ter conhecimento do teor da presente Declaração, estando ciente das sanções e punições legais, no que tange a cópia parcial, ou total, de obra intelectual, o que se configura como violação do direito autoral previsto no Código Penal Brasileiro no art.184 e art.299, bem como na Lei 9.610.
6. O autor é o único responsável pelo conteúdo apresentado nos trabalhos acadêmicos publicados, não cabendo à UFRJ, aos seus representantes, ou ao(s) orientador(es), qualquer responsabilização/ indenização nesse sentido.
7. Por ser verdade, firmo a presente declaração.



Lucas Pinheiro Cinelli

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
Escola Politécnica - Departamento de Eletrônica e de Computação
Centro de Tecnologia, bloco H, sala H-217, Cidade Universitária
Rio de Janeiro - RJ CEP 21949-900

Este exemplar é de propriedade da Universidade Federal do Rio de Janeiro, que poderá incluí-lo em base de dados, armazenar em computador, microfilmar ou adotar qualquer forma de arquivamento.

É permitida a menção, reprodução parcial ou integral e a transmissão entre bibliotecas deste trabalho, sem modificação de seu texto, em qualquer meio que esteja ou venha a ser fixado, para pesquisa acadêmica, comentários e citações, desde que sem finalidade comercial e que seja feita a referência bibliográfica completa.

Os conceitos expressos neste trabalho são de responsabilidade do(s) autor(es).

DEDICATION

To my parents

To my friends

To my future self

ACKNOWLEDGMENTS

First and foremost, I am grateful to my parents and family, who have always supported me no matter what and whom I can count on.

Furthermore, I would like to thank both my advisers Eduardo da Silva and Lucas Thomaz, as well as their patience and willingness in helping me; Professor Paulo Diniz, who introduced me to them and without whom I would consequently be lost; Allan for his ideas and tips throughout this project; and the members of the committee, Sérgio Lima Netto and José Fernando de Oliveira, whom I know and appreciate.

I could not go without deeply thanking Igor Quintanilha and Roberto Estevão, with their knowledge, insightful suggestions and jokes about our desperateness; and my dear friends Antonio Lobato, Carlos Oliveira, Eduardo Barretto, Hugo Sadok, Thiago Cosenza and Ulisses Figueiredo, who embarked in this journey with me half a decade ago.

At last, to all professors of our department, DEL/UFRJ, that I had the pleasure to exchange ideas with, discuss with and learn from, I am eternally thankful.

RESUMO

A detecção eficaz de anomalias em vídeos de vigilância em diversos cenários é um grande desafio em Visão Computacional. Esse trabalho propõe uma abordagem de subtração de *background* baseada em redes neurais residuais, uma recente técnica de *Deep Learning*, capaz de detectar múltiplos objetos de tamanhos diferentes através da segmentação individual e simultânea dos pixels. O algoritmo recebe como entrada uma imagem de referência (sem anomalia) e uma de alvo, que devem estar temporalmente alinhadas, e computa o mapa de segmentação com a mesma resolução da imagem de entrada. Experimentos mostram desempenho competitivo na base de dados analisada, assim como capacidade de processamento em tempo real.

Palavras-Chave: aprendizado profundo, redes residuais, subtração de fundo, segmentação, detecção de anomalia, vigilância, tempo real.

ABSTRACT

Efficient anomaly detection in surveillance videos across diverse environments represents a major challenge in Computer Vision. This work proposes a background subtraction approach based on the recent deep learning technique of residual neural networks capable of detecting multiple objects of different sizes by pixel-wise foreground segmentation. The proposed algorithm takes as input a reference (anomaly-free) and a target frame, which should be temporally aligned, and outputs a segmentation map of same spatial resolution. Experiments show competitive performance in the studied dataset, as well as real-time capability.

Keywords: deep learning, ResNet, residual networks, background subtraction, segmentation, anomaly detection, surveillance, real-time.

Acronyms

AI - Artificial Intelligence

ANN - Artificial Neural Network

API - Application Programming Interface

BG - Background

BN - Batch Normalization

CDNET - Change Detection .net

CNN - Convolutional Neural Network

CPU - Central Processing Unit

FC - Fully-connected

FG - Foreground

FOV - Field-Of-View

GPGPU - General-purpose computing on graphics processing units

ILSVRC - ImageNet Large Scale Visual Recognition Challenge

ML - Machine Learning

MLP - Multilayer Perceptron

MoG - Mixture of Gaussian

NN - Nearest Neighbor

PTZ - Pan-Tilt-Zoom

RGB - Red Green Blue

ROI - Region Of Interest

SFO - Static Foreground Object

SGD - Stochastic Gradient Descent

UFRJ - Universidade Federal do Rio de Janeiro

VDAO - Video Database of Abandoned Objects

Contents

1	Introduction	1
1.1	Theme	1
1.2	Scope	1
1.3	Proposal and objectives	2
1.4	Methodology	2
1.5	Text structure	3
2	Theoretical foundations	5
2.1	Machine learning	5
2.2	Deep learning	6
2.2.1	Motivation	6
2.2.2	History	7
2.2.3	The neuron model and the classical MLP	8
2.2.4	Convolutional neural networks	11
2.2.5	Layers	12
2.2.6	The learning process	23
2.3	Residual networks	28
3	Related work	33
3.1	Video surveillance and Anomaly detection	33
3.2	Deep learning image segmentation	36
4	Databases	38
4.1	ImageNet	38
4.2	CDNET	39
4.3	VDAO	41

5 Proposed convolutional network	46
5.1 The Torch framework	46
5.2 The network architecture	46
5.2.1 Proposed changes	46
5.3 Pre-processing of input video	48
5.4 Training phase	51
5.4.1 Full-training	55
5.4.2 Fine-tuning	55
5.5 Post-processing of network output	57
6 Results	58
6.1 Original setting	58
6.2 Modifications to the LeNet5 architecture	59
6.3 Modifications to the ResNet architecture	67
6.4 Use of pre-trained models	72
6.5 Summary	73
7 Future work and conclusion	78
7.1 Future work	78
7.2 Conclusion	79
Bibliography	81

List of Figures

2.1	Biological and artificial neuron models	9
2.2	Arrangement of neurons in Multilayer Perceptron	10
2.3	The LeNet-5 network architecture	12
2.4	2D convolution operation	14
2.5	Pooling operation examples	16
2.6	Sigmoid and hyperbolic tangent functions	17
2.7	The rectified linear unit function	17
2.8	Dilated operation examples	22
2.9	Error curves comparison between regular plain and residual nets . . .	28
2.10	The residual block of a ResNet	29
2.11	Diagram of a regular plain and a residual architecture	32
3.1	Fully convolutional networks for image segmentation	36
3.2	Deep background subtraction	37
4.1	Example of ImageNet's hierarchical structure	39
4.2	video frame samples from CDNet database	43
4.3	Objects used in VDAO single-object videos	44
4.4	Objects used in VDAO multiple-objects videos	45
5.1	Different possible adaptation schemes to reuse pre-trained networks .	49
5.2	Learning rate decay schedules employed	54
5.3	Different possible adaptation schemes to reuse pre-trained networks .	56
6.1	Performance curves of original architecture for different RMSProp hyper-parameter values	60
6.3	F1-score and loss curves for the bilinear up-sampling version of [1] .	63

6.4	F1-score and loss curves for the deconvolutional versions of [1]	64
6.5	F1-score and loss curves for 100 epochs of the best deconvolutional model	65
6.6	Checkerboard patters in deconvolutional models	66
6.7	F1-score and loss curves for the dilated versions of [1]	66
6.8	F1-score and loss curves for the bilinear up-sampling residual networks based on ILSVRC	68
6.9	F1-score and loss curves for 100 epochs of the best up-sampling ResNet model	69
6.10	F1-score and loss curves for the deconvolutional residual networks . .	70
6.11	F1-score and loss curves for the dilated residual networks	71
6.12	F1-score and loss curves for fine-tuned up-sampling residual networks	73
6.13	F1-score and loss curves for the fine-tuned dilated residual networks .	74
6.14	Examples of segmentation results for the CDNET database using the LeNet-based models	76
6.15	Examples of segmentation results for the CDNET database using the ResNet-based models	77

List of Tables

2.1	Size in number of parameters of 2 different networks	18
2.2	Performance comparison of different deep networks	31
4.1	Statistics of high level categories of the ImageNet dataset	40
4.2	CDNet database video categories	42
5.1	Videos used during training and validation	52
6.1	Summary of hyper-parameter values used for RMSProp	59
6.2	Summary of metrics for the best single epoch of each trained model .	75

Chapter 1

Introduction

1.1 Theme

The present work deals with the foreground segmentation and anomalous object detection in surveillance videos. It focuses on machine learning, a branch of artificial intelligence, to perform automatically this task. More precisely, it investigates the use of deep learning methods to perform pixel-wise foreground segmentation in target frames by comparing them to reference images.

1.2 Scope

The present work is mainly concerned with analyzing existing feed-forward neural network architectures and how they may apply to the foreground segmentation problem in a known scenario. Hence, it does not bother to classify the objects nor the scenarios according to preset labels (e.g dog, bottle, backpack, park, factory). Furthermore, instance segmentation¹ is not discussed here either, only pixel-wise segmentation² is considered.

¹It extends image segmentation to include the notion of different instances of the same class, that is, it is aware of each object of each class individually.

²The task of assigning a label to each pixel of an image, thus dividing the image into separate categories.

1.3 Proposal and objectives

Background subtraction is embedded in a wide range of image processing applications, such as surveillance and tracking, since it is frequently employed as one of the first steps in computer vision frameworks [2]. However, designing a system that can handle diverse environmental conditions still is a major challenge in this area. Furthermore, those background subtraction methods have been almost solely developed for static cameras, consequently, they do not perform satisfactorily in videos where there is translational or rotational motion.

On the other hand, deep learning presents itself as a powerful tool to automatically learn and extract abstract representations of data by building a hierarchy of features along the network, from simple low-level representations to task-specific high-level ones. The deep learning approach has led to breakthroughs in image classification, segmentation, and object detection, among others tasks, thus raising the question of how far it can be pushed. Hence, this project takes upon itself the mission of being one of the first to combine both deep learning and background subtraction to achieve proper pixel-wise foreground segmentation, aspiring to carry the excellent results shown in other domains over to the one at hand.

The main objective is to adapt famous and well-performing deep learning network architectures to predict the pixel-wise foreground segmentation of a given input image. Specifically, the current work aims to:

1. Propose and discuss different adaptations to existing architectures that were primarily designed for solving other problems;
2. Evaluate the performance of such networks and compare to other existing foreground segmentation methods.

1.4 Methodology

The work will port several deep learning networks acclaimed within their application fields to the numeric computing framework Torch7 [3]. After transforming architectures originally designed for image classification and multi-label semantic segmen-

tation into models capable of outputting a dense pixel-wise probability, the models will be trained in a set of video frames taken from the change detection database [4], a publicly available database specifically designed for background/foreground segmentation.

Several modification schemes shall be further experimented with according to inspirations of other well-performing techniques in deep learning, aiming to maximize the specificity of the network. All these different architectures will then be tested against a validation set, that is, another set of video frames with no overlap whatsoever with the training one.

The approach to be adopted throughout the project is similar to that adhered by Brahmans *et al* [1], where they use as input to a deep neural network a two-channel image with one channel being a simple background model and the other the grayscale version of the image to be analyzed. Although some have opted for different frameworks [5], we feel that these deviate from the beauty of deep learning in that they break from the end-to-end concept.

The success of this work will be assessed by verifying that the proposed network is at least competitive with [1] while being faster and more efficient. Also, an investigation of whether it is also effective in more challenging videos [6], i.e. an industrial plant footage from a moving camera in a closed circuit, is possible.

1.5 Text structure

Chapter 2 briefly mentions the role of machine learning in computer science and provides a historical overview of neural networks. Then it introduces the basic notions of deep learning so that the reader can comprehend what has been developed in this work.

Chapter 3 reviews the related work in video surveillance and image segmentation found in literature, indicating the differences between the developed method and other approaches and how they motivated this project.

The video databases employed for training, validating and evaluating the constructed algorithms are presented in Chapter 4.

Chapter 5 discusses the deep learning architectures used along with minute explanations of changes made and their motivation. It also describes how video data is divided and employed for training and validation.

Next, Chapter 6 shows the results obtained through the different architectures examined, reasoning about the attained performances and the disparity among them.

Finally, Chapter 7 summarizes the work done and debates about its benefits, ending with considerations about possible future works and axes of research that can be undertaken in order to improve results and foreground segmentation with deep learning in general.

Chapter 2

Theoretical foundations

This chapter starts with a brief description of the computer science subfield devoted to solve real-world problems by making data-driven decisions, the *machine learning* domain.

Secondly, it presents a historical overview of artificial neural networks (ANN) followed by explanation of concepts. Then, it treats the more recent subject of *deep learning* along with the fundamental bricks employed in such approach.

Finally, it discusses *Residual Networks* at length, the convolutional neural network (CNN) architecture family of choice for the present system, highlighting their advantages.

2.1 Machine learning

Artificial Intelligence (AI) is the Computer Science field that aims to produce a machine capable of perceiving its environment and interacting with it purposefully to attain a goal. Experience suggests that AI obtains better results by constructing their own knowledge based on data. The AI branch devoted to the study and construction of algorithms that can learn from and make predictions on data is called *Machine Learning* (ML). According to [7], ML is the only viable approach to building AI systems that can operate in complicated, real-world environments.

The objective of an ML algorithm is to, given input data X , predict its output by employing a mapping function which is assumed to be parameterized. This mapping is found by going through training samples that belong to the training set and latter evaluated on new unseen examples which forms the testing set. Hence, the performance of those algorithms will depend upon the representation of the data they are given. Each of those representations is called a feature and ML must correctly correlate them to the possible outcomes. A very basic ML algorithm but yet strikingly present on daily life is the *naive Bayes* approach, which is a key method in filtering spam e-mail [8].

Machine learning can be divided into three main categories according to the learning process, i.e. how the training occurs:

- **Supervised learning:** Both input samples and their correct outputs are available. The goal is to learn a general function that maps inputs to outputs.
- **Unsupervised learning:** There is no labeled output, the algorithm should unveil the underlying structure of the input by itself.
- **Reinforcement learning:** There are not input/output pairs either, but rather rewards from the dynamic environment with which the algorithm interacts in order to achieve a goal.

2.2 Deep learning

2.2.1 Motivation

As mentioned previously in Section 2.1, a great number of AI tasks can be easily solved if the correct set of features is given. Nevertheless, for most tasks knowing which features to extract is not a trivial matter, in fact it requires a great deal of domain knowledge and insight.

One approach to overcome this barrier is to grasp through machine learning the representation of data. This is known as representation learning and often results in much better performance than hand-crafted representations.

However, even then, extracting abstract features may be challenging since profound comprehension of the data is frequently needed. This is where deep learning excels at: it introduces representations that are built upon simpler ones, going from bottom up. By learning to represent the world as a nested hierarchy of concepts, it achieves unprecedented flexibility and performance.

Naturally, many deep learning applications are highly profitable and not surprisingly many top technology companies have begun using it, e.g. Google, Microsoft, Facebook, IBM, Baidu and NVIDIA.

As a side note, there is no threshold to a model to be considered *deep*. Instead, deep learning should be considered the machine learning domain that employs a much greater amount of explicitly learned concepts than traditional machine learning models.

2.2.2 History

Artificial neural network history begins in 1943 with McCulloch-Pitts neuroscience-based linear neuron model [9] whose weights had to be manually set. Only in the late 1950s did Rosenblatt's perceptron neuron arise, capable of learning the weights for each output category by input samples. The goal was to obtain a structured way to solve general learning problems.

However, all those models were linear and suffered from numerous drawbacks. Quite remarkable was the fact that no model could learn the simple XOR function which heavily contributed to the NN's first disappearance.

It was not until the 1980s that ANNs were rediscovered under the central idea that learning may be handled by combining several small simple units together whether they are neurons in human brain or units in computational models. While in this second wave, the back-propagation algorithm was successfully transported from the control theory domain to ML [10] and it still is by far the most used algorithm for training (see Section 2.2.6 for detailed explanation).

Nonetheless, neural networks had several problems: a large number of parameters to optimize, absence of large datasets to train with, low computational power at the time. All of this culminated in overfitting during training, poor performance during testing and in really slow training even for a small net. Thus, neural networks dove back into oblivion in the mid-1990s.

The third and present wave has been triggered mainly by Geoffrey Hinton [11] who demonstrated how to separately pre-train each layer of a specific type of ANN and was able to achieve state-of-the-art results in 2006.

Systems, whether animals or computers, become more intelligent as the number of neurons working together increases. A small set of neuron units is not particularly useful. Nowadays, the computational resources available¹ allow running much larger models [12] in a feasible amount of time.

Moreover, the age of *Big Data* has given enormously more data to construct datasets with and feed to learning algorithms [7]. As a consequence, the performance attainable by machine learning has soared.

Only in 2012, deep learning started to be noticed by the computer vision community after [13] proposed the deep learning architecture Alexnet, which won one of the most important challenges in computer vision, the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) and outperformed the runner-up by an error rate more than 10% lower.

2.2.3 The neuron model and the classical MLP

In neural networks, a single computation unit is called a neuron. Similar to the biological neuron, an artificial one takes in multiple input signals from its dendrites and injects output signals onto its axon which will further branch and connect to

¹As follows Moore's law, much faster CPUs are available, but beyond that General Purpose Graphics Processing Units (GPGPU) allow highly parallel computing and are the leading technology for the purpose of training deep models. Furthermore, software libraries such as Torch, Theano and Caffe made fast prototyping, training and testing possible.

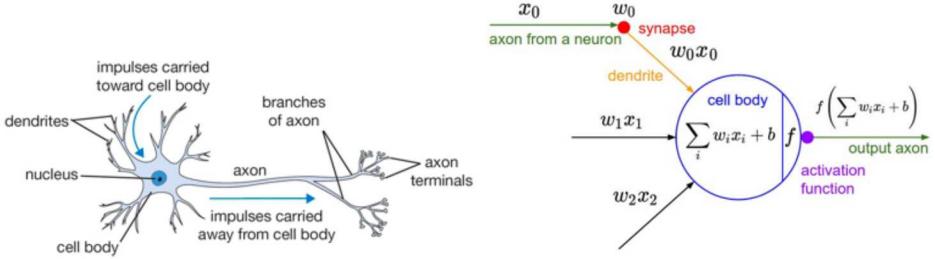


Figure 2.1: On the left, a simple biological model for a neuron. On the right, the artificial model for the same neuron: its inputs come into the cell body, where they are linearly combined, from the dendrites, which can have different interaction strengths, and if strong enough the neuron triggers an output signal.

other neurons' dendrites forming a directed graph. However, whereas input signal and dendrites can interact in complex non-linear fashion in biological neurons, in the computational model they may only interact multiplicatively (Fig. 2.1). The idea is to encode in a learnable manner how much a signal excites or inhibits the neuron. In the basic model, the input signals are all summed in the cell body and if the result is greater than the threshold level, the neuron activates, that is, it triggers a spike down the axon. If input and output signals are restricted to binary values, it becomes the Rosenblatt's perceptron [14].

The multilayer perceptron (MLP), in spite of its name, does not simply implies stacking several perceptron units together, but rather the multilayered association of multiple neurons whose input/output values may assume any real number in the closed set $[0, 1]$ and whose activation functions instead of being the Heaviside step function (equation 2.1) can be any desired function, though sigmoids (equation 2.2) are typically chosen for its probabilistic interpretation and resemblance with the firing rate of an active neuron.

$$H(x) := \frac{d}{dx} \max\{x, 0\} \quad (2.1)$$

$$\sigma(x) := \frac{1}{1 + e^{-x}} \quad (2.2)$$

Neurons in adjacent layers are fully pairwise connected (the habitual shorthand term is fully-connected), each connection having its strength defined by weight be-

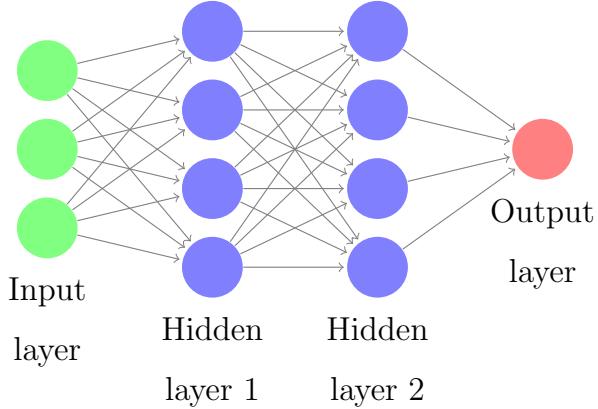


Figure 2.2: Neurons in adjacent layers are fully pairwise connected. Neurons pertaining to the same layer do not share any connection whatsoever.

tween neurons. Neurons pertaining to the same layer do not share any connection whatsoever, as is shown in Fig. 2.2.

Evaluation of samples consists in inserting input data and propagating the result through each layer of the MLP continuously until last one, whose result will be the net output. This procedure is called *feed-forward* operation. A second operation is the so-called *backward-propagation* in which the gradient of the error function calculated on the correct output and the predicted one is propagated back to the beginning of the network in order to update its parameters and teach the network. Those two procedures are interleaved leading the net to learn the proper parameters and correctly predict the output.

The universal approximation theorem [15] states that given any desired continuous function $f(x)$ and some finite error $\epsilon > 0$, there exists a neural network $g(x)$ with one hidden layer capable of representing such function with error upper bounded by ϵ . That means that $\forall x, |f(x) - g(x)| < \epsilon$. Simply put, a single layer network is sufficient to represent any function with any desired degree of accuracy given that the net is large enough.

Although seemingly powerful, in practice this statement is meaningless since there is no guarantee that the training algorithm will learn that function, furthermore the layer may become infeasibly large and may fail to generalize correctly. Generally, the opposite approach is the best: deeper models reduce the number of required

units and reduce the generalization error as shown in [16]. Even though it has been known for long that, in general, depth, regardless of width, does increase prediction accuracy, not until very recently such approach was impractical. Only with the computational capacity made available by cutting-edge GPUs, has the community begun to speculate with such models.

2.2.4 Convolutional neural networks

The design of convolutional neural networks are based on the visual mechanism of the brain, the visual cortex. Neuroscience has long been an active research field, especially on the visual cortex, which corresponds to approximately 25% of the human cortex.

The first model of circuits in the visual cortex comes from the research done in cats' cortex by Hubbel and Wiesel in [17]. They demonstrated that the visual cortex is arranged retinotopically, that is, nearby cells in the cortex process nearby regions of the visual field, which implies that information locality is preserved in the processing stage. Moreover, cells are structured in a hierarchical manner so that their activations progressively build up to more complex patterns.

One of the first attempts to imitate the visual cortex through code was Fukushima's Neocognitron [18]. It consisted of a layered architecture with several layers of local receptive cells that processed small regions of the input followed by pooling operation². This systematical structure was in accordance with what was known about the visual cortex, just like Hubbel and Wiesel's research had shown, locality was preserved and hierarchically arranged.

After a long break, Yann LeCun takes upon the Neurocognitron's design of 1980 and learns a similar architecture through back-propagation whose goal was to classify hand-written mail digits (Fig. 2.3). The LeNet-5 network [19] though still small, employed convolutional layers (explained in *convolutional* in Section 2.2.5)

²The pooling operation basically performs a spatial down-sampling (*Pooling* in Section 2.2.5 further explains the pooling operation).

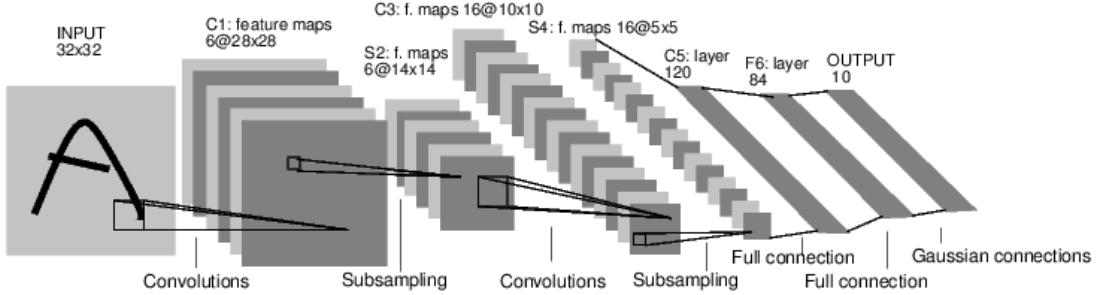


Figure 2.3: The LeNet-5 network consists of 5 stacked layers. The first layers are composed of alternating convolution and pooling layers. The three last layers are fully-connected and are similar to the traditional MLP model. Between each convolution-pooling stage, there are several feature maps which are mappings of the original image. Source: [19]

and the modular structure of Fukushima's work, just as modern convolutional neural networks. CNN is the most widely used network in computer vision applications.

Instead of employing matrix multiplication in all of its layers as the mapping function, convolutional networks use, in at least one of the layers, convolutions to obtain the mapping.

The output of each stage and, consequently, input of the next one is a set of feature maps. That is, the features extracted from all the locations of the input by that layer. Naturally, the feature maps for the first layer would be the image itself. Generally, counting the number of layers is not done by counting the number of layers *per se*, but instead by counting the number of stages of the net, i.e. convolutional layer, non-linearity layer, and pooling layer, for the early layers and fully-connected for the later ones (the Section 2.2.5 explains all above mentioned layers).

2.2.5 Layers

There are several different types of layer one can use in a network architecture, some of them have even been mentioned previously. In what follows, the most important ones either in general or for this project will be succinctly explained.

Convolutional. Convolution is a mathematical concept widely used in the Signal Processing domain. It can be described mathematically for one dimension as

$$y[n] = (x * h)[n] = \sum_k x[n - k]h[k] . \quad (2.3)$$

While the two-dimensional version including the convolution of 2D images with 2D kernels will follow

$$y[n, m] = (x * h)[n, m] = \sum_k \sum_l x[n - k, m - l]h[k, l] . \quad (2.4)$$

In practice, a convolutional layer consists on sliding a kernel (or filter, both will be used interchangeably) on all possible positions of the input and computing the value of each of those positions by calculating the Hadamard product between the kernel and the corresponding region of the input (Fig. 2.4). An important observation is that besides the two spatial dimensions, there are N feature maps and so the kernel actually is a three-dimensional block with depth equal to the number N of maps.

As stated in [7], “convolution leverages three important ideas that can help improve a machine learning system: sparse interactions, parameter sharing and equivariant representations”. Those traits are inspired by biological systems as it was discussed in Section 2.2.4.

Sparse connections means that each output neuron is only connected to a limited number of input positions instead of the whole input data as in the MLP. Whilst parameter sharing entails that the same parameters are used more than once in the model, in the CNN context that amounts to employing the same kernel to each position of the input. Consequently, a visual pattern which activates the filter may appear at different locations of the input, hence this structure also leverages translation invariance. Those concepts result in considerably smaller filters, fewer parameters, less memory usage and faster models.

The kernel size which determines the receptive field of the neuron is a hyper-parameter as well as the number of filters to use and their stride³. The number of

³Stride is the pixel-wise distance between two adjacent positions where the kernel is applied to.

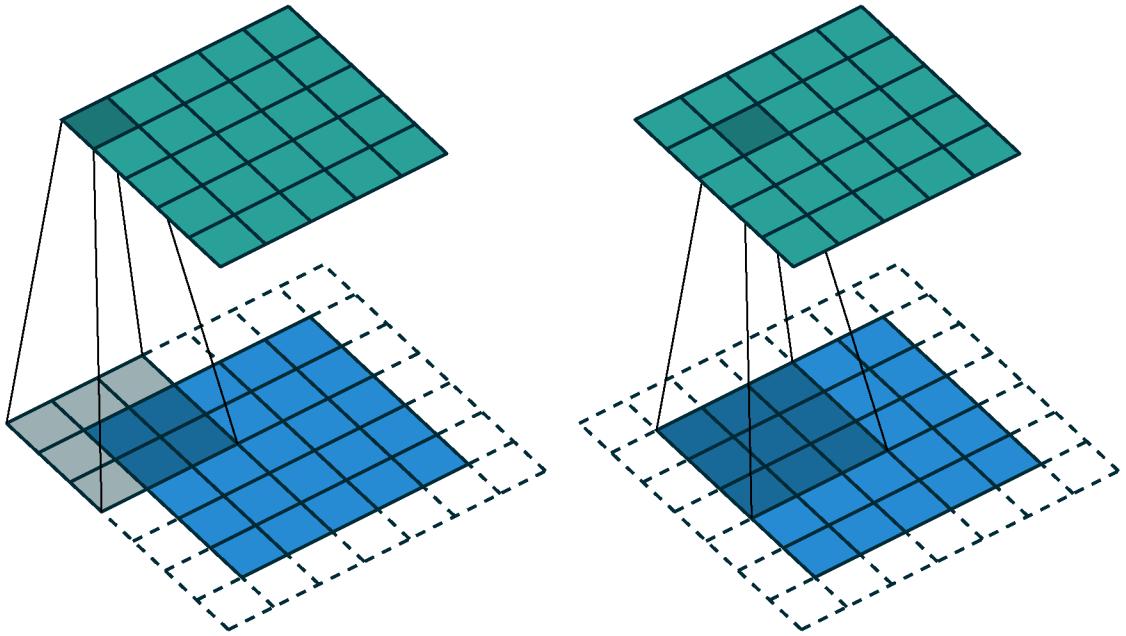


Figure 2.4: In blue and green the input and output feature maps respectively. The shadowed region in the input shows the the receptive field for the shadowed output neuron. Note that without the addition of the dashed region (zero-padding) the output would have smaller spatial dimensions.

kernels determines the depth size of the output with each filter learning a different feature while the stride determines the spatial size of the output, larger strides produce smaller maps.

Generally, several small convolutional filters are preferred over a single large one. Larger receptive fields have much more parameters and represent a linear function of the input whereas the stack of small filters have both less parameters and more non-linearities which is a positive token as the features learned shall be more expressive.

When a valid convolution between a kernel and an input image (or feature map) is computed, the kernel cannot be made to fit in all input positions and the convolution is not computed on the image borders. Specifically, if K is the size of the kernel, then $(K - 1)/2$ pixels on each border are left out. Let the input be of spatial size W , the output F and the stride equal to S , one may compute the feature map's size at the output by $F = (W - K)/S + 1$.

Although it may seem irrelevant, many convolutional layers stacked together can reduce the image to the point of uselessness or at least poor final system performance due to the great loss in information. Padding the input image prior to the convolution leads to undesirable boundary effect, but can guarantee that the spatial size will remain the same. Sometimes it will be convenient to pad the input volume with zeros around the border. If on each input border P pixels are padded ($P = 1$ for the padding shown in Fig.2.4), then the spatial size of the output is determined by:

$$F = \frac{W - K + 2P}{S} + 1 . \quad (2.5)$$

Among the different types of padding, zero-padding which consists of attributing zero to the padded input pixels is the most common one in the deep learning community.

Pooling. Pooling replaces the value of a small region with a statistic over that region. As a result, it reduces the resolution of feature maps, increasing invariance to small local translations and reducing sensitivity to distortions, both typically within the local receptive field of the neuron. Therefore, it progressively decreases the amount of parameters and compute in the net.

This approach is actually imposing a prior to the system by forcing the layer to learn this invariance. If correct, it can significantly increase the efficiency of the network. Anyhow, in most computer vision tasks that is the case.

The most common forms of pooling are max-pooling and average-pooling, the former outputs the average of a rectangular region while the latter its maximum value (Fig. 2.5). Normally, those regions do not overlap, the pooling filter and its stride are forced to be equal. Moreover, the pooling operation acts independently on each feature map.

Activation. Each neuron can be interpreted as a detector for a certain visual pattern that will fire proportionally to its activation function output. This is a point-wise nonlinear function applied to all neurons of the input volume. This

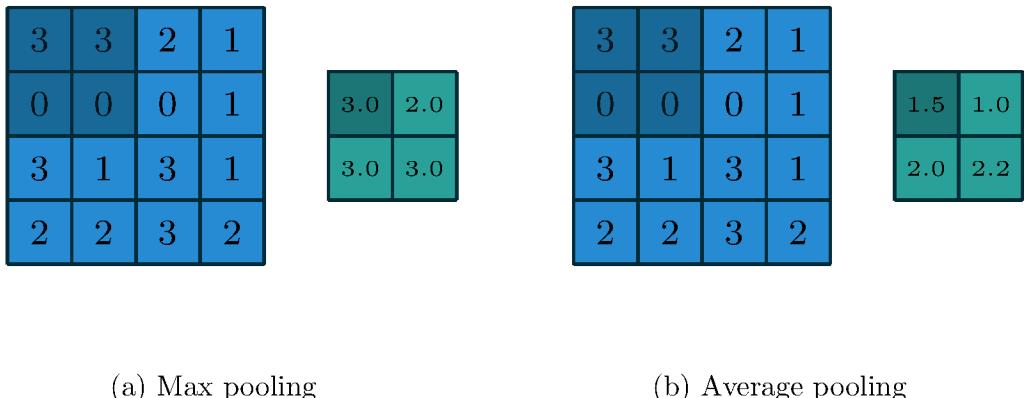


Figure 2.5: The two most commonly used poolings. In blue and green the input and output feature maps respectively. The shadowed region in the input shows the the receptive field for the shadowed output neuron. The max-pooling extracts the maximal value of the region. The average pooling computes the mean of the region. In the above examples, kernel size and stride are equal to 2.

nonlinearity is critical to the network since it allows the system to learn complex nonlinear mappings, otherwise the whole network would be an overly-complicated linear function.

There are multiple choices for the nonlinear function. Historically, sigmoids (equation 2.2) had been chosen for several reasons: it is bounded to the range $[0, 1]$ which has both probabilistic and biological (firing rate) interpretations, it is smooth and infinitably derivable everywhere (Fig. 2.6). Nonetheless, it has major flaws when used in the optimization context of deeper models: it is not zero-centered and even worse due to its flat tails the neuron may saturate and the gradients forced to zero which prevents learning; this is known as the vanishing gradient problem.

The hyperbolic tangent $\tanh(x)$ (Fig. 2.6) is another common activation function that has been recently discouraged by the community. It shares very similar properties with the sigmoid function, which is natural since $\tanh(x)$ is a scaled (and dislocated) version of $\sigma(x)$, indeed: $\tanh(x) = 2\sigma(2x) - 1$. Hence, it has the same pitfalls as the sigmoid, but with one additional advantage of being zero-centered. Thus, sigmoid-like functions have been almost completely deprecated.

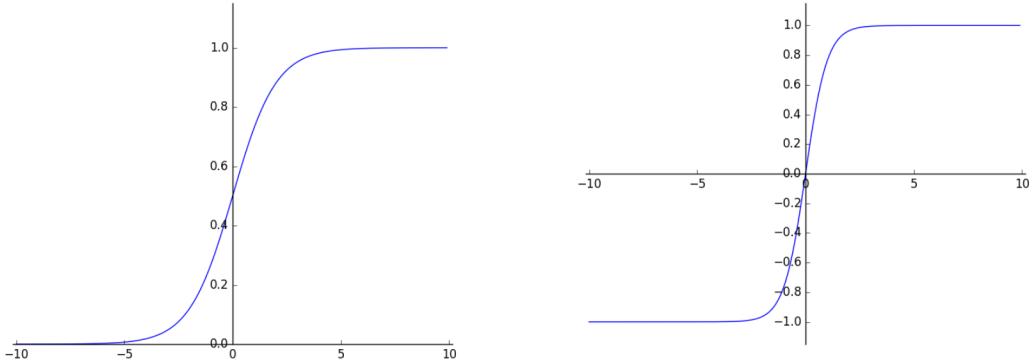


Figure 2.6: The $\sigma(x)$ function on the left and the $\tanh(x)$ function on the right. They share very similar properties, $\tanh(x)$ is only a scaled $\sigma(x)$, indeed: $\tanh(x) = 2\sigma(2x) - 1$. Their flat tails are the cause for the vanishing gradient problem during learning.

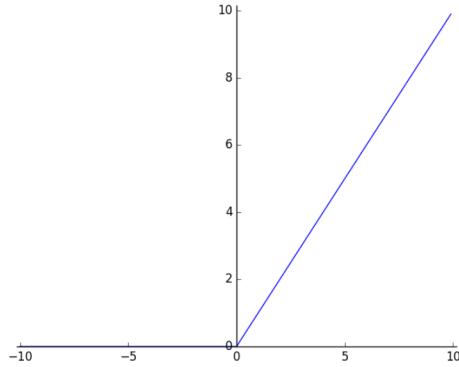


Figure 2.7: The ReLU unit computes the function $f(x) = \max(0, x)$

Currently, the most widely used function is the Rectified Linear Unit (ReLU), shown in Fig. 2.7. It simply thresholds the input at zero according to the function $f(x) = \max(0, x)$, hence it computationally cheaper than the sigmoid function and accelerates the convergence of the optimization algorithm (e.g. in [13] it converged $6\times$ faster than for sigmoid activation). Its only shortcoming is that, due to the unitary slope, even for great value data might end up being thrown out of the manifold and cause the neuron to die, that is, never activate again.

Fully-connected. Similarly to the MLP model, this type of layer has full pairwise connection with its input and no connections within its neuron (refer to Section 2.2.3 for further information). Their output feature maps can be computed by a full matrix multiplication.

Table 2.1: Per layer breakdown of the VGG-16 network at the right and of the ResNet-20 (Section 2.3 explains the ResNet architecture). The fully-connected layers concentrate the majority of the parameters of the net with astonishing 90% of the whole size. In effect, removing the FC layers in favor of an approach such as the average-pooling layer which does not introduce any parameters whatsoever can have a powerful effect in reducing the network size.

layer	number of parameters	layer	number of parameters
conv3-64 x 2	38,720	conv7-64 x 1	9,408
conv3-128 x 2	221,440	conv3-64 x 2	147,456
conv3-256 x 3	1,475,328	conv3-128 x 2	516,096
conv3-512 x 3	5,899,776	conv3-256 x 2	2,064,384
conv3-512 x 3	7,079,424	conv3-512 x 2	8,257,536
fc1	102,764,544	(avg-pool-7)	0
fc2	16,781,312	fc	512,000
fc3	4,097,000	TOTAL	10,994,880
TOTAL	138,357,544		

Let W be the weight matrix and x the input feature maps volume, then:

$$y = Wx . \quad (2.6)$$

There is no sparse connection nor parameter sharing as in the convolution operation. Hence, it concentrates the majority of the whole network's parameters as shown in Table 2.1. For that reason, the recent approach of using fully-connected (FC) layers at the end of the net has been frequently replaced with other methods such as average-pooling. [12].

Despite the difference between FC and CONV layers both compute dot products and can as a consequence be converted from one form to the other. It suffices to set the kernel size to be exactly equal to the size of the input volume which will have a $1 \times 1 \times n$ output size, producing the same outcome as the equivalent FC layer. The real advantage of this approach is being able to have larger inputs without any further modifications to the net, giving the same result as a sliding window at the

input but with much lower computational cost since there is no recomputation of overlapping pixels.

Batch normalization. Deep learning suffers from severe covariance shift, that is, the distribution of the function’s domain changes along training, and this problem gets worse as the number of layers increases. Since the architecture’s design is hierarchical, during training the input distribution of the layer L varies due to parameter actualization of all previous $L - 1$ layers even if the training samples are taken at random, thus forcing the layer to continuously adapt to the new distribution. This phenomenon, coined internal covariance shift, slows training and makes parameter initialization critical to convergence.

Ioffe & Szegedy [20] argue that whitening the input to each layer is a first step in fighting those problems, and, so, propose a new layer whose function is to normalize independently each mini-batch⁴ feature by both its mean and variance, as full whitening of each layer’s input is costly and not everywhere differentiable, fully integrating it to the architecture. For a layer L with d -dimensional input $x = (x^{(1)} \dots x^{(d)})$, the normalization is

$$\hat{x}^{(k)} = \frac{x^{(k)} - E[x^{(k)}]}{\sqrt{Var[x^{(k)}]}}, \quad (2.7)$$

where $E[\cdot]$ is the expected value operator and $Var[\cdot]$ the variance operator.

However, this transformation limits the representational capacity of the layer. Therefore, to avoid this issue the authors introduce a set of parameters $\gamma^{(k)}$ and $\beta^{(k)}$ which scale and shift the normalized feature $x^{(k)}$ by the mapping

$$y^{(k)} = \gamma^{(k)} \hat{x}^{(k)} + \beta^{(k)} \quad (2.8)$$

that the model learns iteratively with all other parameters.

Whereas the normalization uses mini-batch statistics for efficient training; for inference, it uses the population statistics instead, since the output should be predicted deterministically. Algorithms 1 and 2 respectively present the Batch Normalization

⁴A mini-batch is a randomly sampled subset of the whole training set.

(BN) transform of an activation x over a mini-batch, and the procedure for training and later testing batch-normalized networks. In both algorithms, ϵ is a small constant used to avoid division by zero and increase numerical stability.

Algorithm 1 Batch Normalizing Transform (BN)

Input: Values of x over a mini-batch $\mathcal{B} = x_{1\dots m}$;

Parameters to be learned γ, β

Output: $y_i = BN_{\gamma, \beta}(x_i)$

$$\begin{aligned} \mu_{\mathcal{B}} &\leftarrow \frac{1}{m} \sum_{i=1}^m x_i && \text{(mini-batch mean)} \\ \sigma_{\mathcal{B}}^2 &\leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 && \text{(mini-batch mean)} \\ \hat{x}_i &\leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} && \text{(normalize)} \\ y_i &\leftarrow \gamma \hat{x}_i + \beta \equiv BN_{\gamma, \beta}(x_i) && \text{(scale and shift)} \end{aligned}$$

Batch normalization may be applied to whatever set of activations in the network one wishes, but the authors in [20] argue it is more effective right after affine transformations, which covers both convolutional and fully-connected layers in feedforward CNNs. Therefore, the operation

$$x = g(Wu + b) \quad (2.9)$$

with learnable parameters W and b , nonlinearity g and input u , becomes

$$x = g(BN(Wu)) , \quad (2.10)$$

where BN stands for the Batch Normalization transform and the bias b has been dropped because its effect is canceled by the mean subtraction and its role taken by the shift parameter β in the transformation. In the specific case of convolutional layers, normalization is further jointly applied over all spatial locations, and the parameters γ and β are, consequently, learned per feature map instead of per activation.

Batch Normalization allows the use of saturating nonlinearities and higher learning rates, diminishes the number of iterations needed for training ($\sim 10\times$), and (partially) regularizes the model, since training samples in the same mini-batch are jointly seen.

Algorithm 2 Training a Batch-Normalized Network

Input: Network N with trainable parameters Θ ;

subset of activations

Output: Batch-normalized network for inference N_{BN}^{inf}

1: $N_{BN}^{tr} \leftarrow N$ (Training BN network)

2: **for** $k = 1 \dots K$ **do**

3: Add transformation $y^{(k)} = BN_{\gamma^{(k)}, \beta^{(k)}}(x^{(k)})$ to N_{BN}^{tr} (Alg.1))

4: Modify each layer in N_{BN}^{tr} with input $x^{(k)}$ to take $y^{(k)}$ instead

5: **end for**

6: Train N_{BN}^{tr} to optimize the parameters $\Theta \cup \{\gamma^{(k)}, \beta^{(k)}\}_{k=1}^K$

7: $N_{BN}^{inf} \leftarrow N_{BN}^{tr}$

8: **for** $k = 1 \dots K$ **do**

For clarity, $x \equiv x^{(k)}$, $\gamma \equiv \gamma^{(k)}$, $\mu_B \equiv \mu_B^{(k)}$

9: Process multiple training mini-batches \mathcal{B} , each of size m ,

and average over them:

$$E[x] \leftarrow E_{\mathcal{B}}[\mu_{\mathcal{B}}]$$

$$Var[x] \leftarrow \frac{m}{m-1} E_{\mathcal{B}}[\sigma_{\mathcal{B}}^2]$$

10: In N_{BN}^{inf} , replace the transform $y = BN_{\gamma, \beta}(x)$ with

$$y = \frac{\gamma}{\sqrt{Var[x]+\epsilon}} x + \left(\beta - \frac{\gamma E[x]}{\sqrt{Var[x]+\epsilon}}\right)$$

11: **end for**

Deconvolutional. This operation in spite of its name is not the inverse of the convolution in the mathematical sense. However, the scientific community has not yet agreed on its name, e.g. deconvolution, fractional-stride convolution, transposed convolutional layer.

Actually, this operation still is a convolution but greatly differs in its motivation and use, therefore a separate section is dedicated to it. Those filters are used to implement learnable up-sampling within the network in an end-to-end fashion and can even learn nonlinear up-sampling when conjugated with activation functions. The $1/f$ -stride convolution is equivalent to an f factor up-sampling, so it may be implementing by a *reversed* convolution, that is, by reversing the forward and

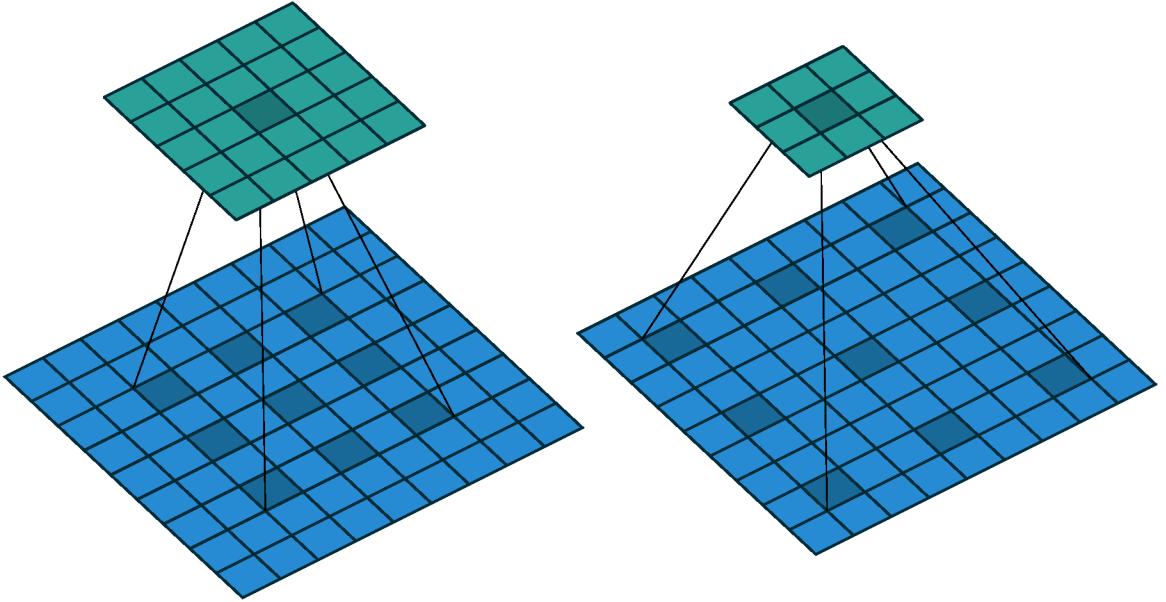


Figure 2.8: In blue and green the input and output feature maps respectively. The shadowed region in the input shows the the receptive field for the shadowed output neuron. On the left a 2-dilated operation ($*_2$) and on the right a 3-dilated one ($*_3$). The dilation factor is the distance from one filter tap to another

backward passes of the *regular* convolution.

This type of operation was first introduced in [21] for dense scene segmentation. The authors claim that in-network up-sampling is fast and effective for learning dense pixel-wise prediction.

Dilated operation. We may generalize convolution and pooling by dilating their kernels, meaning that we can artificially expand their sizes by spacing the taps and filling the empty positions with zero. The distance between each filter value is the dilation factor l (Fig. 2.8). In what follows, we will only discuss the convolution, although similar reasoning is valid for pooling. The definition of the corresponding convolution is

$$y[n] = (x *_l h)[n] = \sum_k x[n - lk]h[k] , \quad (2.11)$$

where the $*_l$ operator represents an l -dilated convolution. According to this, the previously presented discrete convolution $*$ is simply the 1-dilated convolution ($*_1$).

Strictly speaking, there is no expanded filter since that is not efficient. Instead, we pair the kernel elements with non-adjacent positions of the input, which means, that the modification relies solely on the operator itself. Hence, the same filter can be applied with varying dilations.

As discussed in [22], the main application for this kind of operation is dense prediction, since the larger receptive field allows to better integrate context while the spatial resolution remains intact, as long as the stride is 1, which generally is the case. In order to better understand the resolution preservation, consider two stacked convolutional layers: the first with stride $S > 1$; and the second with stride 1 and filter values f_{ij} given by the position indexes i and j . Then, change the stride of the first layer to one so its output is enlarged by S . Consequently, the filter of the second layer sees a smaller region of the image and the result can no longer be the same. Nonetheless, by dilating the kernel of the second convolution by this same factor S

$$f'_{ij} = \begin{cases} f_{i/S, j/S} & S \text{ divides both } i \text{ and } j; \\ 0 & \text{otherwise,} \end{cases} \quad (2.12)$$

the correspondence can be recovered for the filter once again sees the same pixels as before (after the up-sampling, the distance between previously neighboring pixels became S , so setting the spacing between each tap to S necessarily makes the results the same as before). Repeating this procedure for all down-sampling operations along the network (exponentially increasing the dilation at each replacement), ensures the resolution stays the same throughout the net, whilst results are still equivalent. However, maintaining the original size during the whole process might become inviable in terms of memory requirements as model size increases, therefore a compromise must be reached between down-sampling and later up-sampling, and dilation, which will vary for each architecture and will be specified individually for each case.

2.2.6 The learning process

The training phase of a neural network is the search for the optimal set of parameters that minimizes a cost function $C(D, Z)$ defined on the desired output D

and the predicted one Z . The most common method for training is still the back-propagation first proposed in the AI domain back in 1985. This way of computing the gradient is explained below.

Let $C(D, f(I, \theta))$ define the cost function between the desired output D for input I and the output of the system for a given set of parameters θ . The learning problem consists in finding the values of θ that minimize this cost function. In practice the performance of the system on the training set is of little interest. Much more relevant is its performance in the field, meaning the test set, to know whether or not it generalizes to new data.

Let the function f_k , which implements $X_k = f_k(\theta_k, X_{k-1})$, be the mapping of the K^{th} layer that takes as input X_{k-1} and outputs X_k , and θ_k the set of tunable parameters for that function. If the partial derivatives of the cost function C_s with respect to X_k is known, than the partial derivatives of C_s with respect to θ_k and X_{k-1} can be computed by backward recurrence using the chain rule of differentiation:

$$\begin{cases} \frac{\partial C_s}{\partial \theta_k} = \frac{\partial f}{\partial \theta}(\theta_k, X_{k-1}) \frac{\partial C_s}{\partial X_k} \\ \frac{\partial C_s}{\partial X_{k-1}} = \frac{\partial f}{\partial X}(\theta_k, X_{k-1}) \frac{\partial C_s}{\partial X_k} \end{cases} \quad (2.13)$$

Where $\frac{\partial f}{\partial \theta}(\theta_k, X_{k-1})$ is the Jacobian of f with respect to θ evaluated at the point (θ_k, X_{k-1}) and $\frac{\partial f}{\partial X}(\theta_k, X_{k-1})$ is the Jacobian of f with respect to X . The learnable parameters θ in a deep learning architecture are, for the most part, the weights W of the convolutional and FC layers, hence, from now on, we will use W and θ interchangeably, following the common practice in literature.

When the above functions are applied in reverse order, from the last layer down to the first (whose input is the sample I_s), all the partial derivatives of the cost function with respect to all the parameters can be computed. Furthermore, the value for the local gradients can be calculated independently for each operation gate without it being aware of the rest of the network.

Two main assumptions must be made in order to use the back-propagation algorithm:

- The cost function C can be written as the average over the cost of each individual training sample since the algorithm computes the partial derivatives separately for each sample;
- The cost function C can be written as a function of the NN's output.

Generally, back-propagation is used with gradient descent or other first-order iterative optimization method to iteratively update the parameters values:

$$W_{k+1} = W_k - \eta \nabla C(W_k) \quad (2.14)$$

Where η is an extremely important and sensitive hyper-parameter called learning rate and $\nabla C(W)$ is the gradient of the objective function with respect to the weights W at the k_{th} iteration.

The surface of the cost function is typically highly complex with many local minima. Therefore, there is no guarantee that the global minimum can be achieved by the optimization since a first-order method is used. The local minimum found depends on the (hyper-)parameter initialization.

The most common variations of the gradient descent algorithm are the stochastic gradient descent (*SGD*), *momentum SGD*, adaptive gradient (*Adagrad*), root means square propagation *RMSProp*, and adaptive moment estimation (*Adam*). They all are succinctly commented below.

SGD. It is an alternative to regular gradient descent that evaluates the gradient of the cost function for all the training samples in the set which can have up to hundreds of thousands of samples. Stochastic gradient descent instead simply samples a random subset of the whole training sample and analyzes one such *mini-batch* at a time taking it as an estimate of the whole set. This approach leads to a much faster algorithm with often better results due to the noise introduced. This algorithm is

extremely sensible to the learning rate value and should be set and monitored with care.

Momentum SGD. This method is designed to accelerate convergence. It introduces a new variable ν that takes into account the previous steps by accumulating an exponentially decaying moving average of past gradients. The new variable, as the name of the algorithm implies, can be seen as the velocity (or momentum assuming unity mass in the physics equation $momentum = mass \times velocity$) of the particle in the parameter space and the negative gradient of the cost function the force applied to it. In this context, the hyper-parameter μ of the moving average would be the viscous drag of the medium and the responsible for decelerating the particle, otherwise it might never come to rest. This method is most useful when dealing with high curvature parameter space, small but consistent gradients, or noisy gradients.

$$\begin{aligned} \nu &= \mu\nu - \eta \nabla C(W_k) \\ W_{k+1} &= W_k + \nu \end{aligned} \tag{2.15}$$

Nesterov momentum. A variant of standard momentum SGD which essentially attempts to add a correction factor by evaluating the gradient after the current velocity is applied and not the converse. This version of momentum is the most applied one though theoretically it does not improve the rate of convergence [7].

$$\begin{aligned} \tilde{W}_k &= W_k + \nu \\ \nu &= \mu\nu - \eta \nabla C(\tilde{W}_k) \\ W_{k+1} &= W_k + \nu \end{aligned} \tag{2.16}$$

The previous learning algorithms manipulated the learning rate equally for all parameters. The cost function may have different curvatures in different dimensions causing sensitivity to vary through different directions. Therefore, it proves beneficial to use distinct learning rates to update each parameter. The following approaches do so by adaptively tuning the rates. Moreover, the extra hyper-parameters they add are well-behaved for a broader range of values than the raw learning rate.

Adagrad. This method scales the learning rate of each parameter individually by a factor inversely proportional to the root mean square of all its past gradient

values. Consequently, parameters that have high gradients will have their update rate reduced, whilst the opposite will happen for parameters with small gradients. Nevertheless, Adagrad leads to monotonic learning rates too aggressive for deep learning. Let r be the variable that caches the previous squared gradients, σ a smoothing term to avoid division by zero and \odot the element-wise Hadamard product:

$$\begin{aligned} r &= r + \nabla C(W_k) \odot \nabla C(W_k) \\ W_{k+1} &= W_k - \frac{\eta}{\delta + \sqrt{r}} \odot \nabla C(W_k) \end{aligned} \tag{2.17}$$

RMSProp. This algorithm was first introduced in 2012 by Geoffrey Hinton in a slide of his deep learning online course. It slightly modifies Adagrad through the incorporation of an exponentially weighted moving average of the past squared gradient values. As a consequence, the method avoids the monotonic decrease in learning rates that causes learning to stop earlier and as a result performs better than Adagrad in non-convex spaces. However, as the moving average filter is initialized at zero, it has high variance at the first training iterations.

$$\begin{aligned} r &= \rho r + (1 - \rho)(\nabla C(W_k) \odot \nabla C(W_k)) \\ W_{k+1} &= W_k - \frac{\eta}{\delta + \sqrt{r}} \odot \nabla C(W_k) \end{aligned} \tag{2.18}$$

Adam. It adds yet some minor modifications on top of RMSProp, namely, it uses the exponentially weighted moving average of the gradient instead of the raw gradient values and adds correction factors $1/(1 - \rho_i^k)$, where ρ_i^k is the coefficient ρ_i of the i^{th} averaging filter raised to the k^{th} power at iteration k , to the moving averages which may have high bias at the beginning since they are initialized with zero. This method is fairly robust to hyper-parameter setting and is currently the preferred optimization algorithm in deep learning for image processing.

$$\begin{aligned} \nu &= \frac{\rho_1 \nu + (1 - \rho_1) \nabla C(W_k)}{1 - \rho_1^k} \\ r &= \frac{\rho_2 r + (1 - \rho_2)(\nabla C(W_k) \odot \nabla C(W_k))}{1 - \rho_2^k} \\ W_{k+1} &= W_k - \eta \frac{\nu}{\delta + \sqrt{r}} \end{aligned} \tag{2.19}$$

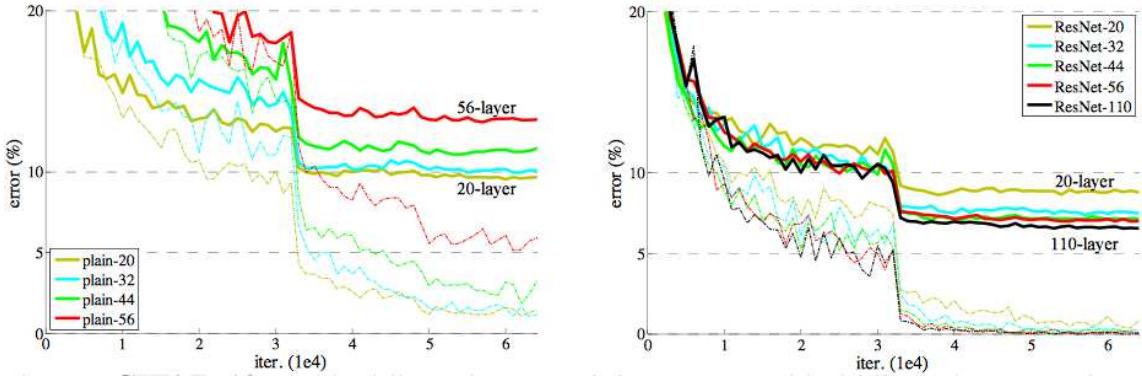


Figure 2.9: Comparison between regular plain nets (at the left) and its residual counterparts (at the right) for multiples depths according to training error (dashed lines) and testing error (bold lines). The error of plain-110 is higher than 60% and not displayed.
Source: [12]

2.3 Residual networks

Feed-forward convolutional nets suffer degradation beyond a certain depth, i.e. the insertion of extra layers in the network causes the accuracy prediction to decrease (Fig. 2.9). This deterioration, however, is not caused by overfitting the training data since both the training and validation errors get greater as the number of layers added increases.

In [12], the authors argue that this underfitting is unlikely to be caused by vanishing gradients either, given that this problem is present even when correct normalization techniques are applied to the layers. They reason that given two identical architectures, where one of them is made deeper by stacking more layers on top of it, this deeper version should be at least as good as the other since the extra layers would learn the identity mapping in the worst case scenario. However, experiments show that current optimizers cannot achieve this constructed solution. They speculate that “the solvers might have difficulties in approximating identity mappings by multiple nonlinear layers” [12].

The paper [12] then proposes adding skip connections to bypass a small number of convolutional layers at a time. Thus, instead of learning the whole mapping, the block learns a residual that is at the end summed with the block’s input by this skip connection. These shortcuts have virtually no cost: they do not add neither extra

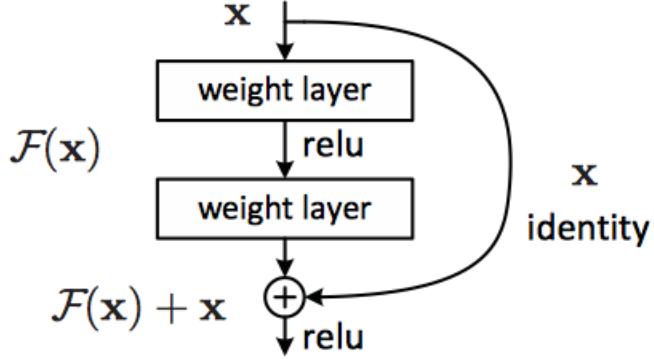


Figure 2.10: A residual block is formed by adding a skip connection that shortcuts a few convolution layers. The convolution layers predict a residual that is added to the block’s input volume. Source: [12]

parameters nor computational complexity.

More precisely, let x be the input to the first layer of residual block and the desired underlying mapping to be learned $\mathcal{H}(x)$. Adding the skip connection forces the stacked layers to fit the mapping $\mathcal{F}(x) := \mathcal{H}(x) - x$ which is the residual function. The authors hypothesize that it is easier to optimize the residual mapping \mathcal{F} than to optimize the original \mathcal{H} .

When the depth of the feature maps of the input and output are different, the additional dimensions must be matched before adding them together. The first and simpler solution is to pad zeros to the shallower volume in order to match the dimensions. The second involves doing a projection from one space to another using 1×1 convolutions whose weights can also be learned⁵, this approach increases the number of parameters by $\sum_{n=1}^N F_n \cdot K_n$ where F_n and K_n are the number of feature maps at the input and output of the n^{th} shortcut projection, and N is the total number of shortcut projections in the net which is very low, generally inferior to a dozen units.

⁵ 1×1 convolutions may seem odd at first but one must remember that in CNNs there are F stacked feature maps and a 1×1 convolution actually has a $1 \times 1 \times F$ kernel and so this is equivalent to associating the corresponding pixels of each of those F maps. Consequently, spatial information is unaltered.

The output of a generic residual block can then be represented by the following equation

$$y = F(x, \{W_i\}) + W_s x . \quad (2.20)$$

where W_i are the input weights and W_s the shortcut projection weights.

The residual network (ResNet) model won first place in many categories of the Imagenet 2015 challenge with a 152-layer net. Table 2.2 presents the results for this architecture as well as some others. The ResNet studied in [12] has the VGG spirit, it is a modular stack of blocks, each applying small 3×3 kernels in the convolutional layers, except for the very first one whose larger size aims quickly reducing the image size. Fig. 2.11 shows the end-to-end structure of a 34-layer residual network, that preserves the same structure of the 152-layer winner, alongside its plain counterpart and the VGG-19. However, it substitutes the fully-connected layers by one simple average pooling, considerably reducing the number of parameters of the net. Furthermore, apart from one pooling layer at the very beginning to aid in rapidly reducing the input image size, the model has no other pooling layer, all the down-sampling is done through convolutions of stride 2.

Residual Networks, or ResNets, are the default choice for convolutional networks. Not only they are currently state-of-the-art, but also converge for a wider range of hyper-parameters.

Table 2.2: Classification error rates (%, **10-crop** testing) on ImageNet validation set. Configuration A means zero-padding for increasing dimensions, B means that the model only uses shortcut projections for increasing dimensions, and C that it uses projections for all shortcuts. Top-1 err. measures the number of times the correct class was not the most probable prediction while top-5 err the number of times the correct class was not one of the 5 most probable classes. The use of projection for matching dimensions reduces the error with almost no cost, however using projection in all skip connections increases cost and entails almost no reduction. The configuration C is not worth its additional cost.

Source: [12]

model	top-1 err.	top-5 err.
VGG-16	28.07	9.33
GoogLeNet	-	9.07
plain-34	28.54	10.02
ResNet-34 A	25.03	7.76
ResNet-34 B	24.52	7.46
ResNet-34 C	24.19	7.40
ResNet-50 B	22.85	6.71
ResNet-101 B	21.75	6.05
ResNet-152 B	21.43	5.71

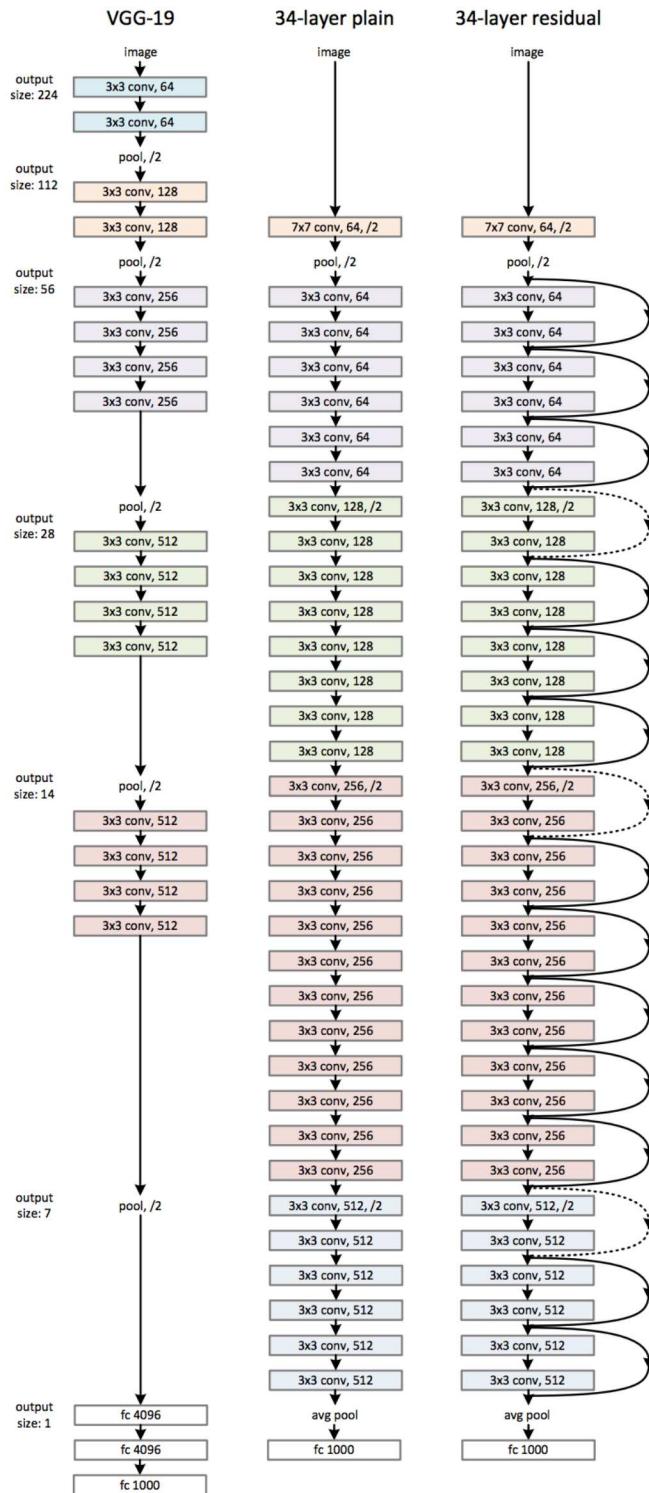


Figure 2.11: Example of network architectures. Left: the VGG-19 model (19.6 billion FLOPs) as a reference. Middle: a plain network with 34 parameter layers (3.6 billion FLOPs). Right: a residual network with 34 parameter layers (3.6 billion FLOPs). The dotted shortcuts increase dimensions. Source: [12]

Chapter 3

Related work

This chapter reviews the main works in literature that relates to the present one either in nature or practice, namely, in anomaly detection in video or in deep learning based image segmentation.

3.1 Video surveillance and Anomaly detection

Background subtraction is embedded in a wide range of image processing applications since it is frequently employed as one of the first steps in computer vision frameworks. It is the process through which one distinguishes among the foreground (FG) and background (BG) of a scene, that is, one identifies the background of a scene and segment the anomalous objects from it without necessarily having prior knowledge about those objects.

Video surveillance systems generally use static cameras. The kind of footage thus obtained causes many challenges to properly model the background and, consequently, detect the foreground. Some examples of these challenges are: illumination changes, dynamic background and FG object occlusion. Hence, countless methods sprouted since the 1990's in the attempt to address those challenges.

Gaussian models are often used to model BG due to its robustness to noise, gradual illumination changes and slowly moving background. The authors in [23] proposed modeling each pixel with a Gaussian distribution based on its N last values in a scheme called Running Gaussian Average. More recent real-time object tracking

applications, such as [24], use this framework to construct a background model and determine the normality mode of a pixel as the region enclosed by a threshold K , attributing the FG label to outliers.

However, when the scene contains many non-static object, the single Gaussian assumption for the pixel intensity distribution will not hold. Then, one may model the background as a Mixture of Gaussians (MoG), instead of only one, as proposed by Stauffer and Grimson in [25]. The work presented in their paper compares every new pixel intensity value to the existing Gaussians modeling it in order to find a match and if none is found, the pixel model is recalculated. MoGs typically allow dealing with illumination changes, low contrast, dynamic background, and camouflage, among other others.

Mixture of Gaussians, nonetheless, has its shortcomings. It is still not very robust to sudden global illumination changes, and may incorporate foreground objects into the background model if the scene remains stationary for a long time causing misdetections and also ghost regions after the objects starts moving once again. Besides, MoG's parameters require careful tuning. Despite the drawbacks of the Gaussian modeling approaches, they are widely deployed in research and practice, accounting for nearly 50% of all background subtractions used for Stationary Foreground Object (SFO) detection [2].

Some methods do not model the BG through raw pixel values, but rather rely on features extracted from the input image. Li *et al.* [26] characterize the background by the most significant and frequent spectral, spatial, and temporal features, and use the Bayes Theorem to determine the *a posteriori* probability of the pixel belonging to the foreground.

The body of literature is rather scarce when dealing with non-stationary cameras. Liu *et al* [27], for example, decomposes the scale range and wide scene of a PTZ camera into different layers of overlapping partial scenes and performs frame registration by feature descriptor matching of the input frame with the partial key-frames of the hierarchical ensemble. Then, foreground objects are detected using

the corresponding key-frame local background models.

Moo Yi *et al.* [28] propose a system that runs in real-time on mobile devices. They use dual-mode single variate Running Gaussian Average model with variable learning rates to model each background pixel vicinity and apply sparse optical-flow tracker along with homography to estimate camera motion between adjacent frames and mix the neighboring BG models to reduce the errors arising from motion compensation.

Notwithstanding, the well established techniques for fixed cameras such as Gaussian-mixtures, and statistical approaches do not transfer properly to situations like those present in the Video Database for Abandoned Objects (VDAO) introduced in [6] (Section 4.3), i.e. heavily cluttered background in moving camera surveillance systems. Recent works, such as [29], handle those issues exploring the low-rank similarities between the reference and target videos to detect abandoned or removed objects. Specifically, the authors propose a framework where the reference video is described by matched-filters spanning optimal sub-space representation and the anomaly is detected by computing the normalized cross-correlation between the filter output when applied to the target frame and the expected value for such filter. This approach, though only requiring very rough synchronization, demands finding the optimal sub-space for each different reference video beforehand. The present work, on the other hand, defines a framework that can be directly employed to new reference/target pairs.

In order to construct the BG models, the previous works (with exception of [29] that uses sub-space representation) rely either on raw pixel values, which are not as robust as features, or hand-crafted features which are problematic as it may take decades of effort from a whole body of researchers to reach a reasonable representation and performance gain (Section 2.2.1). This work sets itself apart from them as it employs deep learning to directly learn those features.

Nevertheless, this is not without precedent. Indeed, in 2013, Maddalena and Petrosino [30] proposed a 3D neural model based on a Self Organizing Map (SOM)

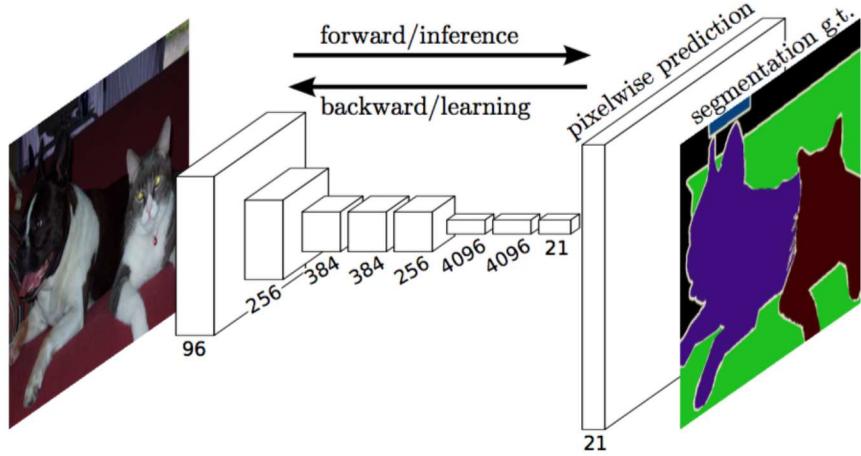


Figure 3.1: Fully convolutional networks for image segmentation. Source [21]

to continuously adapt to changes in the scene without need to supervise learning. The idea was to produce a 3-D grid topographic map that learns image sequence variations seen as trajectories of pixels in time, preserving the input spatial locality.

In a similar fashion, DeepAnomaly [5] pipelines a deep learning network, which acts as a robust feature extractor, with conventional methods for modeling the background to detect anomalies in moving cameras in real-time. The authors verify that the best performing algorithms set on top of the learned feature maps are Gaussian model variants (single and multivariate, and mixture). However, the system targets a very specific scenario, i.e. anomalies in agricultural fields, which are by nature uncluttered and have periodic background motion (oscillation due to the wind).

3.2 Deep learning image segmentation

Deep convolutional neural networks have been pushing state-of-art performance in classification tasks as well as in other visual tasks. It is rather natural, then, to port these solutions to image segmentation. Long *et al.* [21] extend famous deep classification architectures to semantic segmentation by shaping them into fully convolutional networks which are trained end-to-end to predict dense pixelwise semantic segmentation map of same size as the input independently of what it might be. This is achieved by replacing the fully-connected layers by convolutional layers (explained in **Fully-connected** in Section 2.2.5) and performing a fractional-

stride convolution (explained in **Deconvolution** in Section 2.2.5) at the end to up-sample the feature map and recover original image size (Fig. 3.1). Hence, this architecture dispenses with the need for the sliding window technique to construct dense maps and the common but inefficient patch-wise training.

Whilst [5] employs a pre-trained CNN simply as a feature extractor for later processing, Brahams and Van Droogenbroeck [1] use deep learning end-to-end to propose a background subtraction method. They use the Lenet5 [19] to predict the probability of a pixel being part of the FG given its vicinity in the image and the correspondent BG model region as the inputs (Fig. 3.2). Individually sampling all input pixels and their respective vicinity, therefore capturing the local context of the pixel, allows pixel-wise classification of the input, but considerably increases the computational cost of the network. Consider the input image to be of width W and height H and that the net processes patches of size T one at a time, then $W \times H$ evaluations are needed to construct the output map with each input pixel being computed $T \times T$ times, that is, as many times as the size of the patch, instead of only one. The present work is very similar in nature to [5], however, it explores the advantages of fully convolutional networks to diminish the computational requirements of a deep learning foreground detection.

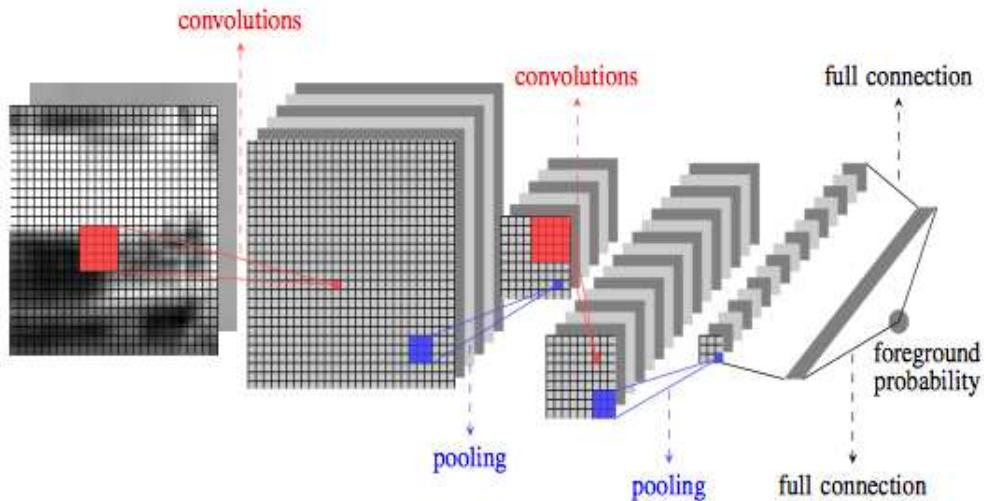


Figure 3.2: Deep background subtraction performed by a variant of the Lenet5 [19] learned by patch-wise training to predict pixel-wise FG probability. Source: [1]

Chapter 4

Databases

This chapter presents the datasets used throughout the project, either directly (CDNet and VDAO) or indirectly (ImageNet). While the first two are video-surveillance specific, the latter is a widely employed dataset for classification, detection and localization, and segmentation using Deep Convolutional Networks.

4.1 ImageNet

The colossal amount of data available in the present due to the arrival of digital era demands rapidly improving methods to harness them. Motivated by this scenario, scientists created the ImageNet [31] database to allow research in the computer vision field to be done in a proper manner and also to allow a fair comparison between similar methods.

ImageNet is one of the most comprehensive and diverse coverage of the image world with 14,197,122 images (as of Jan/2017), providing classes of images in a densely populated semantic hierarchy (Fig. 4.1 exemplifies the structure), where the subcategories (*synsets*) are interlinked by different types of relation. The final objective is to gather tens of millions of annotated images, around 1000 images per synset.

The database offers images with different levels of complexity with variable appearances, positions, view points, poses, and also background clutter and occlusions, each one of them quality-controlled and human-annotated [31]. The plethora of cat-

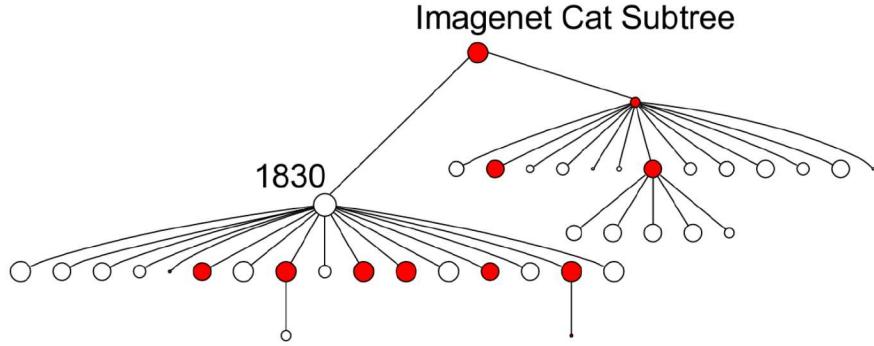


Figure 4.1: The ‘cat’ subtree on ImageNet. Within each tree, the size of a node is proportional to the number of images it contains. The number of images for the largest node is shown for each tree. Source: [31]

egories (Table 4.1 shows a summary of synsets’ size) calls for distinguishing even between very similar classes (i.e. 147 different dog breeds).

The database is so large that spans the most important challenge in computer vision, the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), a benchmark in object category classification and detection on a subset of ImageNet with over fifty different institutions participating in. Furthermore, it is so common to reuse the features learned in CNNs trained on ImageNet on other tasks since there is so much data and the features learned throughout the different layers are quite general, that one may say it is the driving force of deep learning. Following the standard practice, e.g. [21, 32, 22, 33] fine-tune their segmentation models on weights pre-trained on the ImageNet classification task either from the VGG-net [34] or ResNet [12] architecture, this work does not train on the ImageNet dataset for it takes too long, rather it does also import the publicly available pre-trained ResNet’ weights.

4.2 CDNET

The ChangeDetection.NET (CDNET) [4] database is a compilation of 53 videos ($\sim 160,000$ frames) from very different sources covering a wide range of detection challenges. It targets change and motion detection applications and provides a rigorous and comprehensive academic benchmark of fixed surveillance camera videos.

Table 4.1: Statistics of high level categories of the ImageNet dataset. Source:
<http://www.image-net.org/about-stats>

High level category	# synsets	Avg # images per synset	Total # images
amphibian	94	591	56K
animal	3822	732	2799K
appliance	51	1164	59K
bird	856	949	812K
covering	946	819	774K
device	2385	675	1610K
fabric	262	690	181K
fish	566	494	280K
flower	462	735	339K
food	1495	670	1001K
fruit	309	607	188K
fungus	303	453	137K
furniture	187	1043	195K
geological formation	151	838	127K
invertebrate	728	573	417K
mammal	1138	821	934K
musical instrument	157	891	140K
plant	1666	600	999K
reptile	268	707	190K
sport	166	1207	200K
structure	1239	763	946K
tool	316	551	174K
tree	993	568	564K
utensil	86	912	78K
vegetable	176	764	135K
vehicle	481	778	374K
person	2035	468	952K

The dataset is divided in 11 different categories (Table 4.2), each with 4 to 6 video sequences and representing a specific kind of challenge. Hence, CDNET enables identification of the most suitable algorithms for a specific problem as well as the most competent overall.

The videos are camera-captured from typical indoor and outdoor scenes with spatial resolution varying from 320×240 to 720×576 pixels and frame length 1000 to 8000 frames. Besides, they have accurate pixel-level groundtruth segmentation for either all or half the frames with 5 distinct classes:

- **Static** background pixels;
- **Shadow** pixels denoting hard and well-defined moving shadows (Fig.4.2(b));
- **Non-ROI** pixels prevents analyzing activities unrelated to the category considered;
- **Unknown** pixels that are half-occluded or corrupted by motion blur;
- **Moving** pixels from the region of interest.

The CDNET database plays a major role in this work as it is the main source of task-specific data, allowing to learn foreground-specific features and validate the efficacy of network architectures and other ideas, while the ImageNet serves the purpose of laying out basic generic features that, otherwise, would need massive amounts of data to learn from. The CDNET dataset is divided according to [1], discarding *PTZ* and *Intermittent background motion* videos during training and validation, so both results may be compared.

4.3 VDAO

The Video Database for Abandoned-Object Detection [6] radically differs from all other publicly available databases as it presents a set of videos recorded from a moving platform in a heavily cluttered industrial background, whilst all others (such as CDNET) consist of static cameras and, hardly ever, PTZ cameras. The nonlinear movement of the platform and the strong jitter render traditional approaches such

Table 4.2: CDNet database video categories with a brief description of each one.

Video category	Description
Baseline	mixture of mild typical challenges
Dynamic background	scenes with strong (parasitic) background motion
Camera jitter	videos captured by unstable cameras
Intermittent background motion	scenarios known for causing “ghosting” artifacts
Shadows	videos with varying degrees of shadow
Thermal	videos captured by far-infrared cameras
Challenging weather	outdoor videos in challenging winter conditions
Low frame-rate	varying frame-rates between 0.17 fps and 1 fps
Night	videos captured at night (difficult light conditions)
PTZ	footage from pan-tilt-zoom camera
Air turbulence	air turbulence caused by rising heat

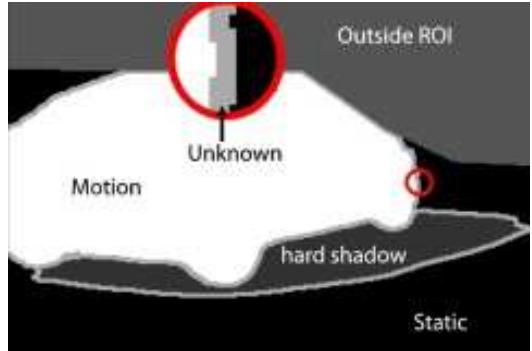
as homography noneffective. However, correct time-space alignment is paramount if one intends performing comparisons between target and reference frame. Furthermore, according to Cuevas *et al.* [2], is one of the only two databases specifically designed for the evaluation of abandoned object detection algorithms.

Among its 8.2 hours and 66 videos, VDAO has 56 single object, 6 multiple objects of different colors, textures, shapes and sizes, as well as 4 clean reference videos filmed from two different cameras with distinct light and color characteristics, but same frame rate and resolution (examples of objects in Fig. 4.3). Moreover, multiple scenes exist where objects are partially or fully occluded and the scene casts unwanted mirrored images due to shadows and reflexive surfaces. In order to assess results, manually annotated bounding boxes are defined for each object in each video of database.

Therefore, VDAO is one of the most difficult and realistic database publicly available, as far as our knowledge goes, for training change detection and motion segmentation. Thus, its purpose in the present work is to challenge the proposed deep learning architectures and test their robustness, inferring how well they may gener-



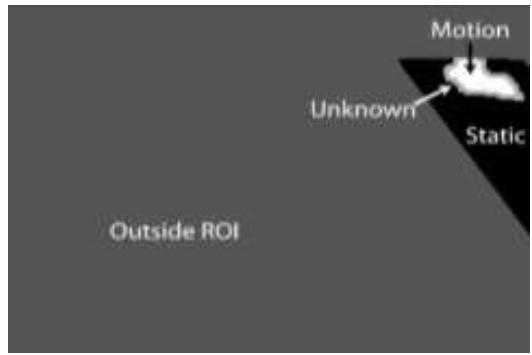
(a) Input frame



(b) Groundtruth frame



(c) Input frame



(d) Groundtruth frame

Figure 4.2: Two video frame samples (a,c) with associated manually-labeled groundtruth (b,d), respectively. There are five different values for the groundtruth mask, they are: static, shadow, non-ROI, unknown (pixels half-occluded or corrupted by motion blur), and moving. However, this work, as [1], does not use all the 5 classes. Instead the static, shadow, non-ROI, and unknown are grouped into non-moving-object category, simply referred as background. Source: [35]

alize.

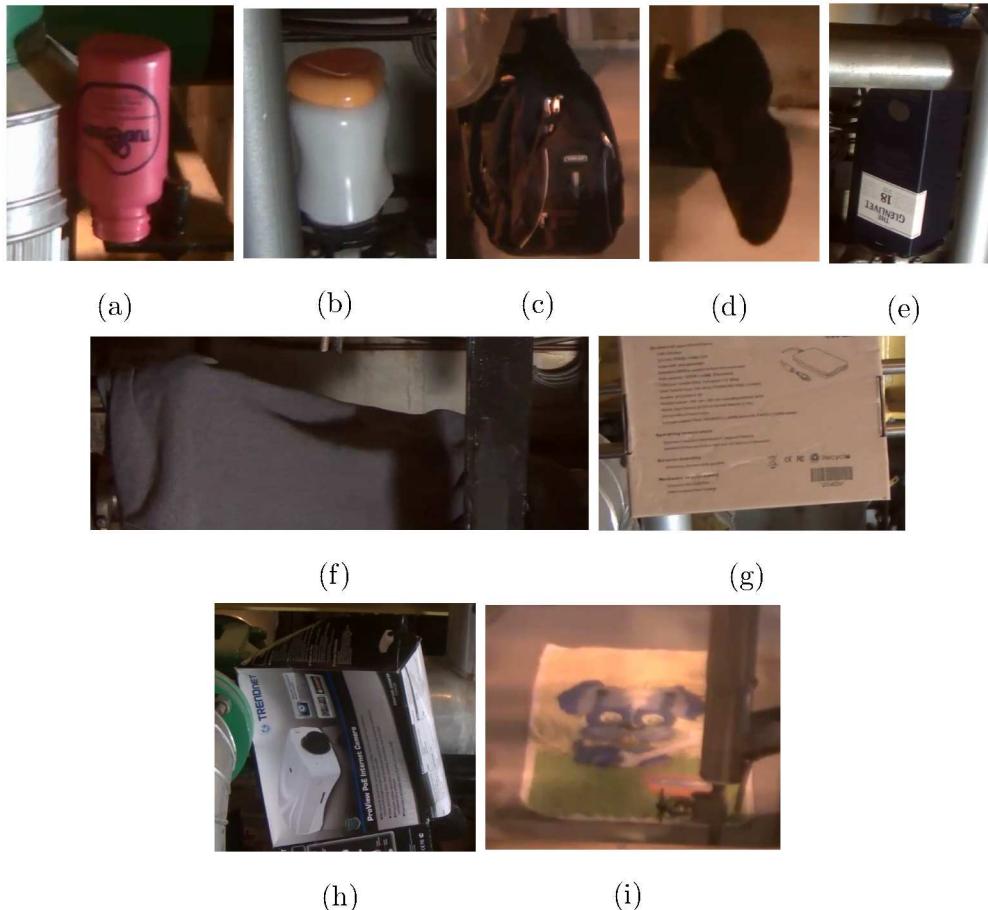


Figure 4.3: Objects used in VDAO single-object videos: (a) Pink bottle; (b) White jar; (c) Black backpack; (d) Shoe; (e) Dark blue box; (f) Black coat; (g) Brown box; (h) Camera box; (i) Towel. Source: [6]

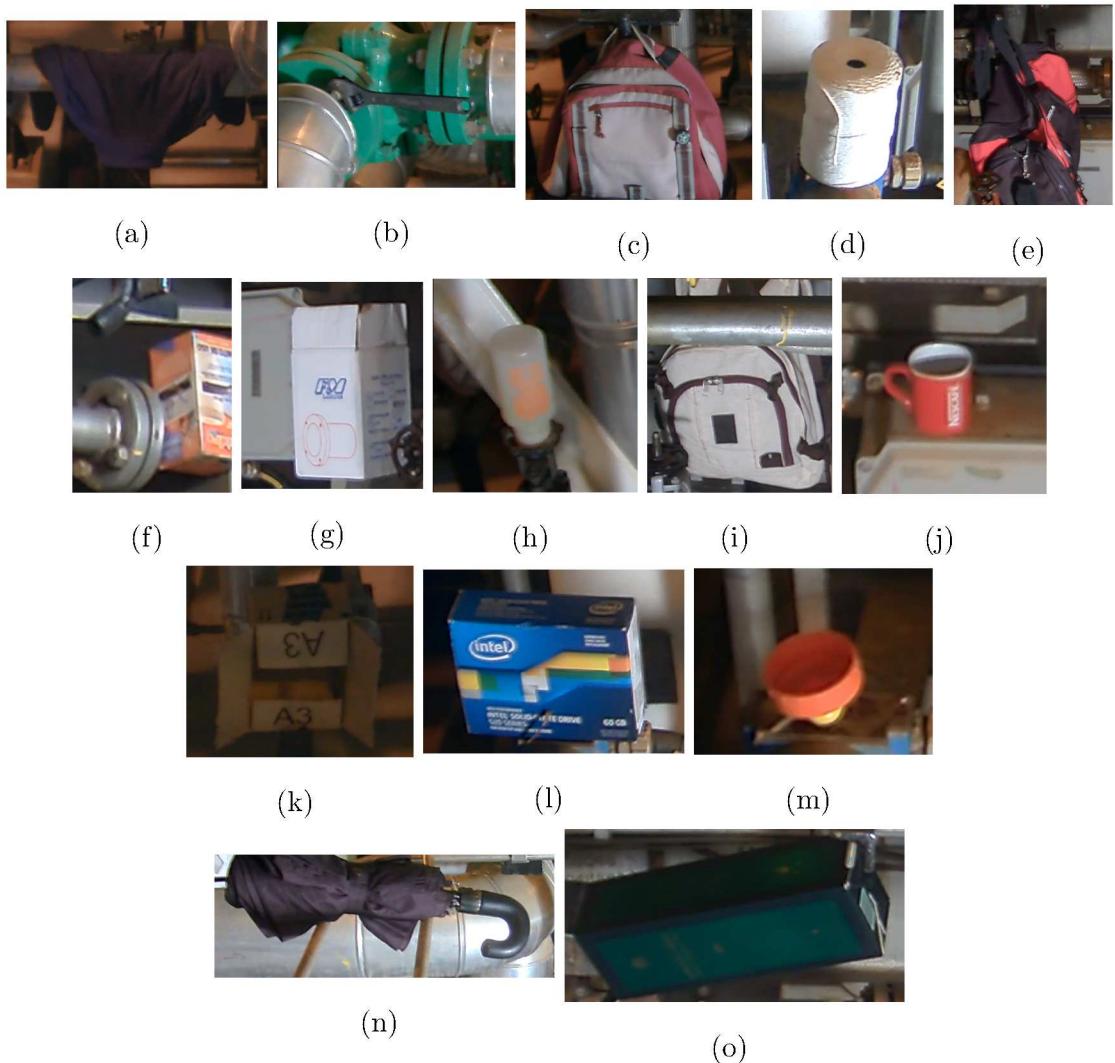


Figure 4.4: Objects used in VDAO multiple-objects videos: (a) Blue coat; (b) Wrench; (c) Pink backpack; (d) String roll; (e) Bag; (f) Lamp-bulb box; (g) Spotlight box; (h) Bottle; (i) Backpack; (j) Mug; (k) White box; (l) Blue box; (m) Bottle cap; (n) Umbrella (o) Green box. Source: [6]

Chapter 5

Proposed convolutional network

5.1 The Torch framework

Torch7 [3] is the numeric computing framework used through the whole project to develop and test the network models studied. Endorsed by major technology companies such as Google and Facebook, it extends the light-weight scripting language Lua at the same time that offers compiled and optimized backend APIs (Application Programming Interface) (e.g. C, C++, CUDA, OpenMP), which renders the process of development easier and more understandable while still being fast and efficient.

5.2 The network architecture

5.2.1 Proposed changes

The studied networks are derivations either from Braham's [1] implementation of LeNet5 [19] or from ResNet [12] architectures made available by the Facebook AI team [36].

The architecture proposed in [1] (Fig. 3.2) performs a binary classification of a single pixel given a 27×27 image patch as input by passing it through two convolutional stages (5×5 convolution kernels followed by non-overlapping 3×3 max-pooling) and two fully-connected layers. The classification step itself is done by the activation of the last and only neuron in the second FC layer, which is a

sigmoid function that acts as a logistic regression and outputs the probability of the pixel at the center of the patch being foreground. Hence, for the construction of the full output map of the same size as the input, the system requires as many forward passes as there are pixels in the input image. This design re-computes the input data many times over, since at a stride of 1, each analyzed patch has a considerable overlap with the previous one. In order to circumvent this issue and gain efficiency, the present work uses the whole image at once as input. To achieve that, the first modification is to replace the fully-connected layers by convolutional equivalents so the network can process images of any size. However, this does not guarantee that the output will have the same size as the original image, thus further adaptations are required. Three main options were studied:

- Simple bilinear up-sampling of the output with factor equal to the number of down-sampling operations done along the network, namely poolings and convolutions with stride greater than 1 (shown in Fig. 5.1a);
- Replacement of common operations by their dilated counter-parts (explained in *Dilated operation* of Section 2.2.5) to preserve resolution (depicted in Fig. 5.1b);
- Deconvolution (definition in item Deconvolutional of Section 2.2.5) operations appended to the end of the network to recover the input size (represented in Fig. 5.1c).

Reutilization of the ResNet architecture faces a similar problem. Besides, being a model originally designed for image classification, prior to the last fully-connected layer, an average pooling operation over the already spatially small feature map transforms the $N \times M \times kFeatures$ tensor in a $1 \times 1 \times kFeatures$ vector, that is, it completely removes all remaining spatial information. Although, such procedure may work well for image classification as location does not matter, in our case of image segmentation it is paramount. Were we trying to detect whether or not there was an anomalous object present in the scene, instead of trying to localize and segment it, then maybe such procedure would be of worth. Hence, we remove the average pooling at the end and repeat the steps done in the previous architecture,

namely replace the fully-connected layer by its convolutional counterpart and adapt the network to output a dense probability map by doing one of the options presented before and adding a sigmoid at the end.

The reason why dilation may work is not as straightforward as why the other two methods might. Consider two stacked convolutional layers: the first with stride $S > 1$; and the second with stride 1 and filter values f_{ij} given by the position indexes i and j . Then, change the stride of the first layer to one so its output is enlarged by S . Consequently, the filter of the second layer sees a smaller region of the image and the result can no longer be the same. Nonetheless, by dilating the kernel of the second convolution by this same factor S

$$f'_{ij} = \begin{cases} f_{i/S, j/S} & S \text{ divides both } i \text{ and } j; \\ 0 & \text{otherwise,} \end{cases} \quad (5.1)$$

the correspondence with the original setting can be recovered because once again the filter combines the same pixels as before (after the up-sampling, the distance between previously neighboring pixels became S , so setting the spacing between each filter tap to S necessarily makes the results the same as before). Repeating this procedure for all down-sampling operations along the network (exponentially increasing the dilation at each replacement), ensures the resolution stays the same throughout the net, whilst results are still equivalent. However, maintaining the original size during the whole process might become inviable in terms of memory requirements as model size increases, therefore a compromise must be reached between down-sampling and later up-sampling, and dilation, which will vary for each architecture and will be specified individually for each case.

5.3 Pre-processing of input video

As explained in Section 4.2, a background model must be extracted from each video in the dataset. Therefore, similarly to [1], to create such model the first 150 frames of each video are passed through a median filter.

The database has a file that defines the ROI (Region Of Interest) of the video, which often is a small subset of the image area. So all regions outside the ROI of

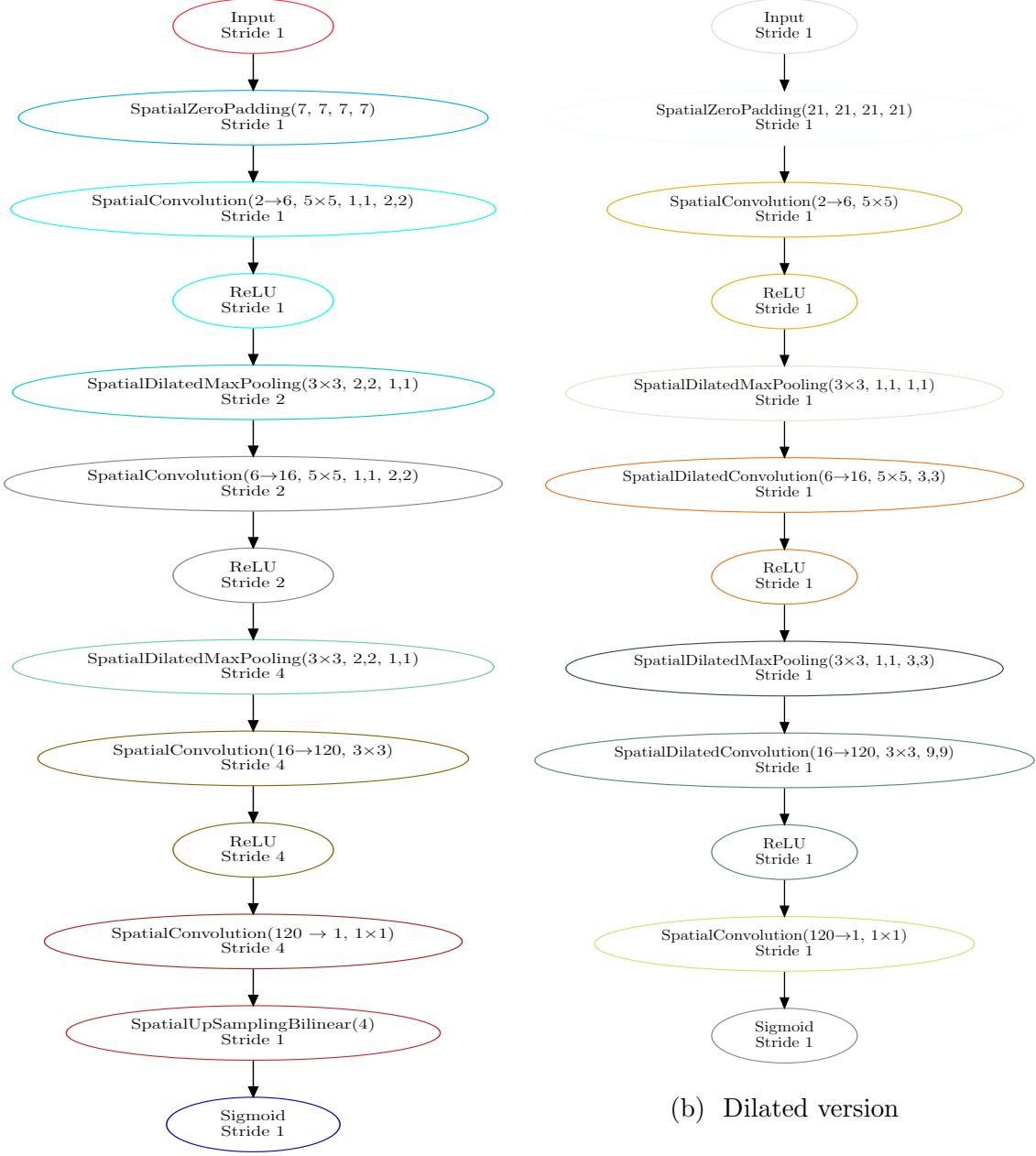
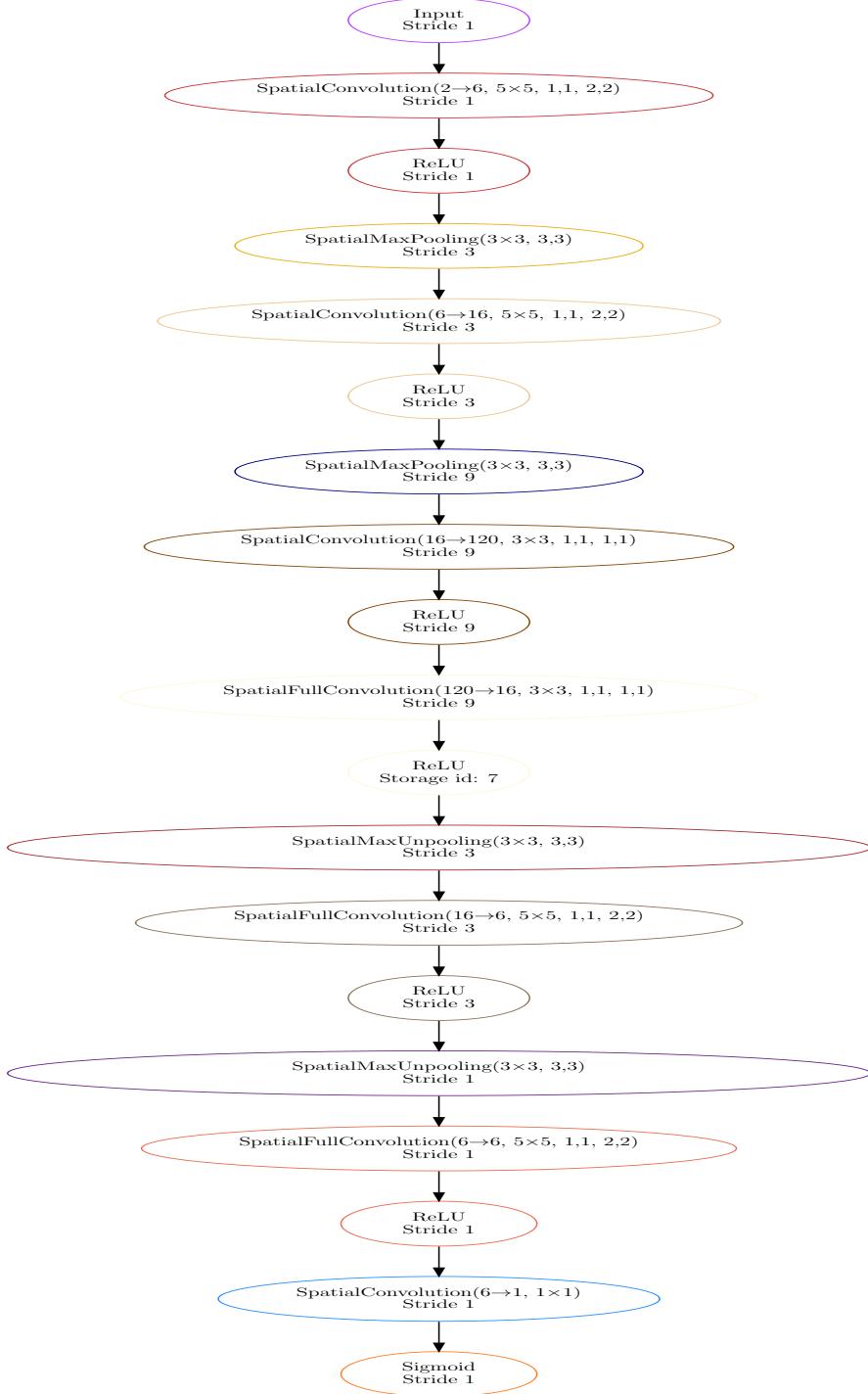


Figure 5.1: Different possible adaptation schemes in order to render possible the reuse of pre-trained networks. Each node of the directed graphs represent an operation in the network. The \rightarrow symbol indicates a mapping from a feature map depth to another, $n \times n$ the size of the operation kernel and the occasional remaining pairs of parameters are the stride and padding (both vertical and horizontal). On the left, a simple bilinear interpolation at the end to up-sample the down-sized image back to its original resolution. On the right, the down-sampling operations (i.e. SpatialMaxPooling) are replaced by their dense version, so instead of having stride of 3, they have stride of 1, this modification requires the adaption of all subsequent spatial operations so that they remain equivalent.



(c) Deconvolutional version

Figure 5.1: (Cont.) The deconvolutional version adds, at the end of the original network [1], a sequence of unpooling (inverse of pooling, that is, up-sampling of the same factor and with no interpolation), deconvolution (named SpatialFullConvolution in the *Torch* framework) and activation function operations, all stacked forming a *deconvolutional stage*. At the very end, a last convolutional layer acts as a fully-connected taking from 6 features maps, instead of 120 as the other structures, to only 1.

both the newly created background model and the input image are set to zero. Furthermore, this considers only two classes (among five of the CDNET), FG and BG, thus all other intermediate classes are also set to zero, that is, they are incorporated into the background.

Next, input and background model are converted to their respective grayscale representation and latter both merged together as two distinct channels of one sole image. That is the input of the network.

As frames may vary from 320×240 to 720×576 pixels, images are scaled down to 256×192 pixels to ease training time.

Note that there is no color-normalization and the sole data augmentation technique is the horizontal flip with probability 0.5 of the input image.

5.4 Training phase

Just as it was explained in Section 2.2.6 of the theoretical foundations chapter, the learning process is divided in two distinct phases: learning (or training), where parameter actualization occurs; and validation, where the model is tested (or validated) in a separate disjoint set of data. The division of those sets is the same as in [1]. Particularly, not all videos are considered, only those shown in Table 5.1. Initially, we divided videos in two halves, the first for training and the second for validation, exactly as Braham *et al.*. Later, however, the same videos were divided in a more usual 70%/30% scheme in order to gather a more representative training set, since there is not much data available. Further modifications include suppressing all training frames that do not have any foreground pixel in an attempt to diminish the class imbalance problem, which reduces it from 21,773 frames to 10,441, whilst keeping the validation intact so as to keep it representative of reality. At the end, there are 10,441 images for training and 9,330 for validation, that corresponds to approximately 50% for each just as the original setting.

The cost function to be optimized is the binary cross-entropy between the estimated class probability q_i and the correct distribution p_i , which would be 1 on the

Table 5.1: CDNET videos used during training and validation.

Category	Considered videos		
Baseline	Highway	Pedestrians	
Camera jitter	Boulevard	Traffic	
Dynamic background	Boats	Fall	
Shadow	Bungalows	People in shade	
Thermal	Park		
Bad weather	Blizzard	Skating	Snow fall
Low framerate	Tram crossroad	Turnpike	
Night videos	All 6 videos		
Turbulence	Turbulence 3		

correct class and 0 on the other. The mathematical equation is as follows:

$$H(q, p) = -\frac{1}{n} \sum_i (p_i * \log(q_i) + (1 - p_i) * \log(1 - q_i)) . \quad (5.2)$$

The cross-entropy can be written in terms of the entropy $H(p)$ and the Kullback-Leibler divergence $D_{KL}(p||q)$ as $H(p, q) = H(p) + D_{KL}(p||q)$. Since the entropy of the Dirac delta function p is null, this is also equivalent to minimizing the KL divergence between the two distributions (a measure of distance), that is, minimize the distance between them.

The objective is the average over the output of all training images in the mini-batch and over all pixels of each image, n values in total. This is a major point of distinction between the training done in [1] and in here, the former samples a few dozens patches and calculates the average of the cost over them, whereas here the averaging is over whole images, which equates to more than 100,000 patches at the same time. This difference is inevitable from how we approach the problem: to densely solve the foreground segmentation task. In order to address class imbalance in this setting, different weights for each class could have been chosen, but it was decided not to, keeping the simplicity of the formula.

The final cost function is achieved by combining the cross-entropy term of the previous equation with a regularization penalty $R(\theta)$. To this purpose, we use the L_2 norm, that is, the element-wise quadratic sum of all parameters, which discourages large values. Let s be the training sample and λ a constant scalar, then the final form of the objective is

$$C(s, p) = H(q, p) + \lambda R(\theta) . \quad (5.3)$$

However, to assess the real-world quality of the model, the F1-score of the detection is used as a proxy for the cost function for it measures the efficiency of the network's predictions in a more palpable manner and is what really matters for a anomaly detection system. The F1-score is defined as the harmonic mean of the precision and the recall:

$$F1 = \frac{precision \times recall}{precision + recall} . \quad (5.4)$$

Which are defined as:

$$Precision = \frac{2 \times true\ positives}{true\ positives + false\ positives} , \quad (5.5)$$

$$Recall = \frac{true\ positives}{true\ positives + false\ negatives} . \quad (5.6)$$

Or, equivalently:

$$F1 = \frac{2 \times true\ positives}{2 \times true\ positives + false\ positives + false\ negatives} . \quad (5.7)$$

Additionally, to minimize the cost function, the optimizer of choice is the Adam algorithm (Section 2.2.6 explains this as well as other solvers) with its default values, which are $\rho_1 = 0.9$ and $\rho_2 = 0.999$. Besides, all models run for 60 epochs with a mini-batch size of 16 with initial learning rate and weight decay hyper-parameters of, respectively, 0.01 and 2e-4, where the latter value is within the customary range of $\sim 1e-4$, but is applied to all parameters (i.e. weights and biases), which is not so common. The learning rate decay schedule, given by

$$LR_t = LR_{init}(1 - decay)^{\lfloor \frac{t-1}{interval} \rfloor} , \quad (5.8)$$

where LR_t is the learning rate at the t^{th} epoch, LR_{init} the initial learning rate (at $t = 1$), *interval* is the amount of epochs t necessary to produce a reduction in LR_t , and *decay* the percentage of the reduction, changed throughout the experiments. Initially, a step decay of 90% at each 30 epochs was used (blue curve in Fig. 5.2), however the cost function seemed to remain more or less stable, in most cases, after 10 epochs following each decay. Therefore, a step decay of 50% at each 10 epochs, which gives 12.5% at the 30th epoch, was adopted instead (green curve in Fig. 5.2). Other schemes could have been employed, but the decay step is the simpler and also most typical one, all others need more hyper-parameter setting. Except for the modification in the learning rate decay configuration, there was no kind of hyper-parameter optimization whatsoever. Unless otherwise stated, these are the settings of all experiments performed.

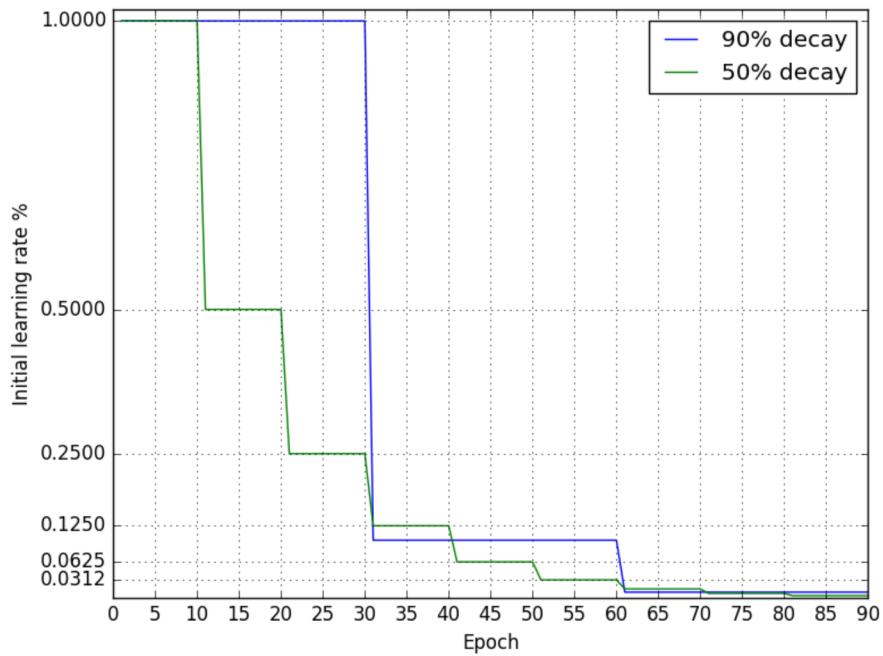


Figure 5.2: Both learning rate decay schedules employed during training. Initially a step decay of 90% at each 30 epochs was used, but initial experiments seemed to remain more or less stable 10 epochs after each decay step (in most cases). This behavior motivated the use of a more aggressive reduction, so a 50% decay each 10 epochs was used instead.

5.4.1 Full-training

Fully training a model consists of building a model from scratch as opposed to fine tuning, that is, parameters are randomly initialized. Generally, initialization follows either the Xavier [37] or the He initialization [38] (both zero-centered gaussian distributions with variance inversely proportional to kernel size \times number of output feature maps).

Brahams *et al.* [1] trained their model in this fashion. However, since it is a Lenet5 and they use 100 27×27 patches per iteration for 10,000 iteration, training is extremely fast taking only a few dozen minutes to run. The situation is quite different for other cases and requiring up to a full day of training is considered to be fast.

In the present work, evaluated architectures are trained from ground and compared. This type of training is given more emphasis, as it is better explained in the next subsection, because the input for our models is completely different from the commonly used RGB image. Therefore, we do not rely on previous networks to initialize ours.

5.4.2 Fine-tuning

In fine-tuning, the starting point of the training phase, instead of being random parameter values, are the values achieved by a network previously trained, that is, the parameters of the new model are initialized to those pre-trained values. This is a very frequent practice in the deep learning community as parameters learned are generic (up to a certain point): the lower the layer, the more generic its parameters, conversely, the higher the layer, the more task-specific it is [39]; and training models from scratch requires lots of data and lots of time (in the order of weeks). Generally, models are first trained on the large ILSVRC dataset (Section 4.1) and then trained on the task-specific set [21, 32, 33].

The main problem with this approach for the case at hand is that all publicly available pre-trained model take as input an RGB (Red Green Blue) image, but

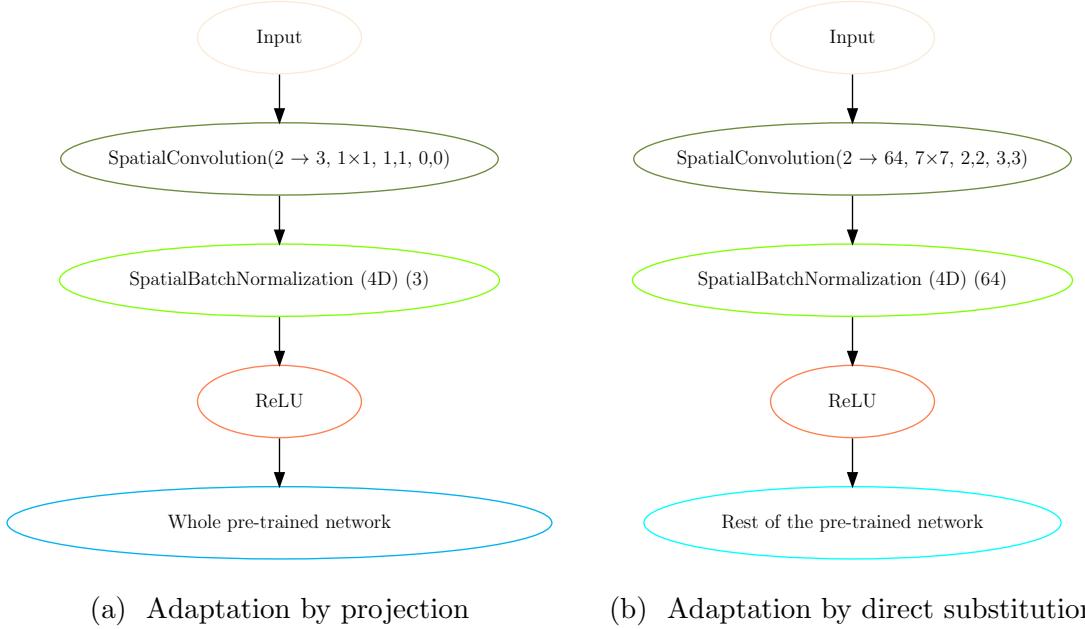


Figure 5.3: Originally, the pre-trained networks take 3-channel (RGB) images as input, but since the developed architectures take only 2-channel images, some kind of adaptation is necessary to interface the new type of entry with the old network model. On the left, the adaptation is achieved by removing the first stage (convolutional layer, batch normalization and activation function) and replacing it by a new one, whose parameters are randomly initialized and take as input 2-channel images and outputs the same number of features maps as the original. On the right, instead of removing, we add a new stage at the beginning, also randomly initialized, responsible for projecting the 2-channel input to the 3-channel space expected by the pre-trained model, thus there is no need to delete anything and the previous network can be completely reused.

our input is a 2-channel image, where each channel is the grayscale of another image. Therefore, both inputs are essentially very different and what has been formerly learned may not transfer properly. At any rate, an adjustment must be made since the first pre-trained convolutional layer expects a 3-channel image, not

2. We consider two options:

- Add a 1×1 convolution (and a batch normalization) before all else so as to project the 2-channel image into a 3-channel space, integrally preserving all pre-trained layers (Fig. 5.3a);
 - Remove the first convolutional layer and the corresponding batch normalization layer and replace them by new randomly initialized ones (Fig. 5.3b).

Nevertheless, it may not be as simple as that for all subsequent layers are built upon the first and changing it implies altering the parameters of the whole model, possibly canceling the positive effects of fine-tuning itself, throwing the process back to full-training again.

5.5 Post-processing of network output

The last layer of the network being a sigmoid, its output is something like a probability map for the pixels of the input. Nonetheless, the detection system should deliver a binary decision, whether there is or there is not a foreground object at that position. Thus, a simple global threshold at the end binarizes the image so it can be compared with the corresponding groundtruth and metrics such as the F1-score calculated. The threshold value during training is set to 0.5 for the sake of simplicity, though the optimum value may very well be lower as the class balance is skewed towards negative examples and almost nothing is done to revert this issue (selection of foreground-present only frames for the training set as discussed at the beginning of Section 5.4), what leads to the classifier naturally being biased towards the negative class. Consequently, imposing a low threshold value would translate to being more acceptant of what should be considered FG, counter-weighting the original problem. Although we fix this value during training, it can be tuned after the network converges so that the F1-score achieves maximal result.

Besides global thresholding, other more sophisticated methods may be employed to adaptively threshold the output. However, that is not the primary focus of this work.

Chapter 6

Results

This chapter presents and examines the results obtained throughout this work. First of all, we present the results obtained by our implementation of the original architecture proposed in [1]. Next, the chapter investigates the outcomes of the modifications suggested in Chapter 5.

6.1 Original setting

Firstly, we tried to reproduced the results of [1], which, originally, motivated the present research. Therefore, we set all hyper-parameters to be exactly as described in [1]: not only the learning rate (0.001), mini-batch size (100) and number of iterations (10000, roughly 65 epochs) are the same, but also the database organization (50% for training and 50% for validation, using all frames), the parameters initialization (bias equal to 0.1 and weight values randomly drawn from normal distribution $\mathcal{N}(0, 0.01)$ truncated at 0.2) and the optimizer algorithm (RMSProp). Furthermore, we recover the original patch-wise architecture by randomly sorting a 27×27 patch out of each image in the mini-batch. However, as Braham & Droogenbroeck did not specify the values employed neither for the the solver's hyper-parameter ρ (brief explanation in item *RMSProp* of Section 2.2.6), nor for the regularization penalty λ (refer to equation 5.3 in Section 5.4), we experiment with $\lambda = 2\text{e-}4$, the same value of all other networks, and $\lambda = 0$ to assess the impact of regularization, as well as we vary ρ among some common values as summarized in Table 6.1.

Table 6.1: Summary of hyper-parameter ρ values used for the RMSProp optimizer algorithm. Not all configurations curves are depicted in Fig. 6.1 and Fig. 6.2, only the more relevant ones (meaning, the ones that achieved better results up until epoch 65).

RMSProp ρ	0.6	0.7	0.8	0.9	0.99	0.999	0.9999	0.99999
----------------------------------	-----	-----	-----	-----	------	-------	--------	---------

Unfortunately, we could not achieve the same F1-score average of 0.9046 as reported in [1], instead, our best model could only attain 0.793 with the configuration: $\rho = 0.9$ and $\lambda = 0$ (Fig. 6.1). We attribute this 10% gap to the fine-tuning of both unknown hyper-parameters λ and ρ , as well as the down-sampling of all dataset images to 256 pixels \times 192 pixels we perform in the pre-processing stage. Motivated by this discrepancy and by the observation that the 70%/30% database division with the filtering of empty groundtruth frames in the training set gives slightly better results, we repeated the tests for all previous settings with this other dataset repartition and the results are shown in Fig. 6.2. Nevertheless the results remain far below the original paper [1]. Moreover, it led to even more deceiving performances with lower F1-score average across the values of ρ and higher validation losses.

Additionally, it is strikingly remarkable that, in virtually all cases, the models' F1-scores oscillate considerably between epochs and the loss stays over $\lambda = 2e-2$ throughout the epochs (which is not the case for the other architectures we investigate and discuss in the remaining sections of this chapter). This phenomenon is rather disturbing because it indicates that the network may not have really learned to segment the foreground, but rather the algorithm of optimization might have stumbled upon a poor local minima and cannot quite settle in this region of the hyper-space as there is no learning rate decay, only the smoothening effect of the per-weight exponential moving average filter of RMSProp acts towards this purpose. Hence, the model may possibly not generalize well for other images.

6.2 Modifications to the LeNet5 architecture

Secondly, we studied the adaptations of Chapter 5 in the patch-wise segmentation network of Section 6.1 in order to transform it into an image-wise system. Extending

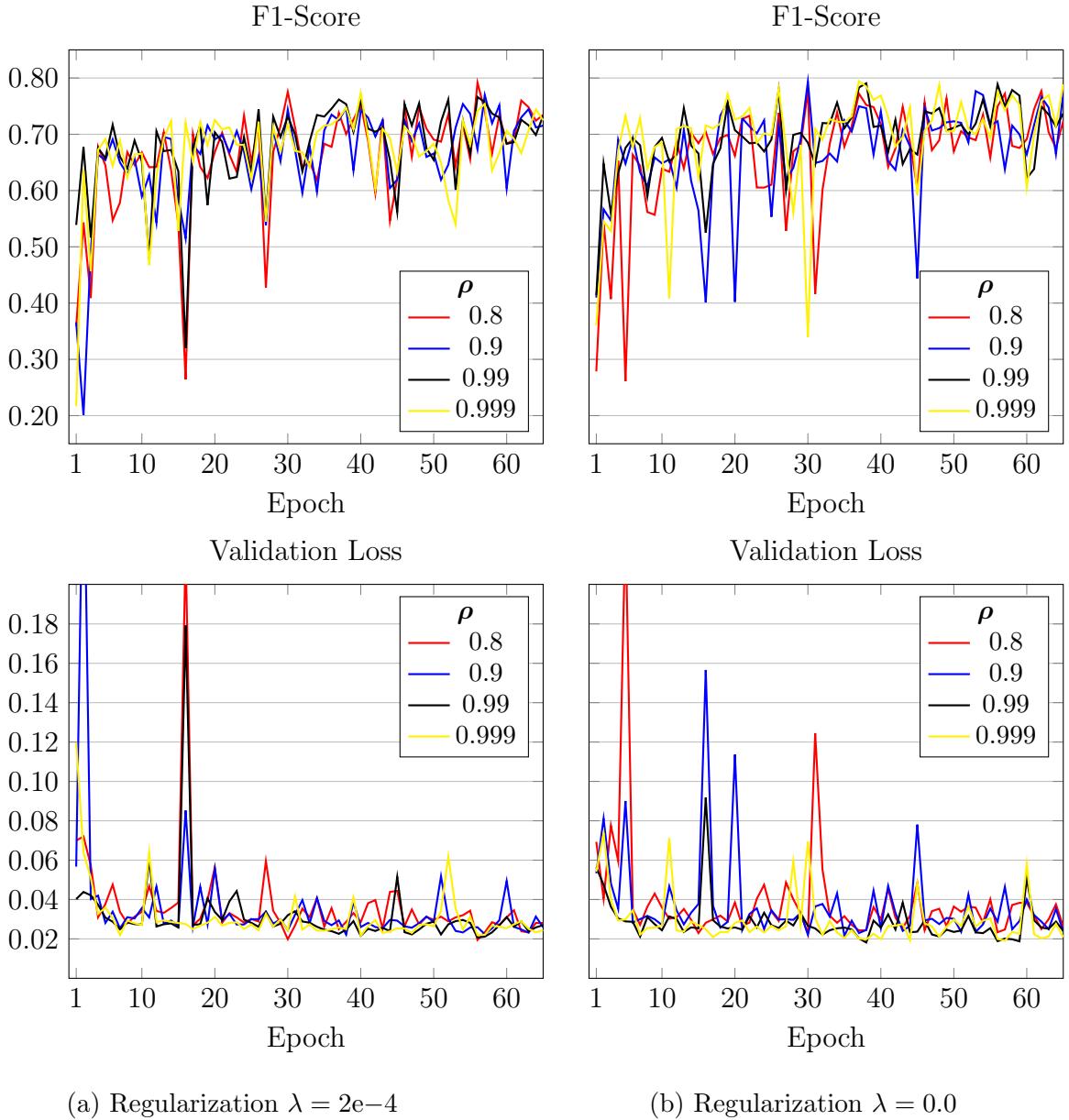


Figure 6.1: Performance curves of the original deep background subtraction architecture [1] for different RMSProp hyper-parameter ρ values and two distinct regularization λ . Train/Val split followed the 50%/50% approach. Employing regularization has a stabilizing effect on the performance curves, though both cases are noisy with F1-Score variations of up to 10% even on the last epochs. The highest achieved F1-score value 0.793, using $\rho = 0.9$ and $\lambda = 0$, is still far from the 0.9046 reported by [1]. We attribute this discrepancy to fine-tuning of both λ and ρ , along with the evaluation on full-resolution images of the validation set, instead of their down-sampled versions ($\sim 2\times$ on each dimension).

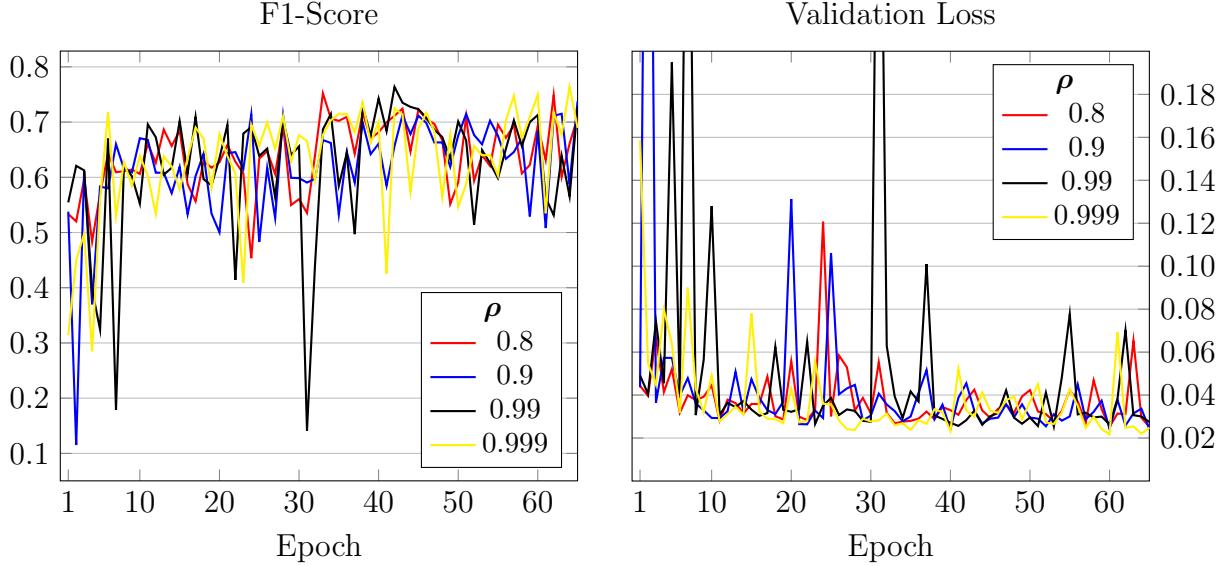


Figure 6.2: The same curves as Fig. 6.1, but the dataset partition follows the 70%/30% distribution with the training filtering scheme, and we only analyze the $\lambda = 2e-4$ case as it was more stable. These configurations are inferior to the previous ones, which rely on the 50%/50% split: the F1-score is lower on average and the validation loss remains higher.

[1] is the most straightforward approach, the net is small and simple, and training is fast. We train all models until the 60th epoch with a mini-batch size of 16 (hyper-parameter configurations in Section 5.4) using three different modifications:

- Bilinear up-sampling of the output back to the original image resolution (Fig. 5.1a);
- Deconvolution operations at the end of the network to recover the input size (Fig. 5.1c);
- Dilated equivalents of the operations done along the network to preserve resolution (depicted in Fig. 5.1b).

We start with bilinear up-sampling of the final feature map (Fig. 5.1a). The architecture performs two pooling operations, each one down-sampling every spatial dimension of the image by a factor of 2, rather than of 3 as in the original network; the basic idea of the overlapping pooling regions is to avoid loss of valuable spatial information. Hence, the 4× interpolation at the end recovers the original input size.

Besides those modifications, we also inspect the effectiveness of batch normalization (homonym item in Section 2.2.5) in this scenario. Thus we train both with and without BN as depicted in Fig. 6.3; we observe that the BN version is consistently better throughout training, though validation values fluctuate more, specially on the first half. Albeit the 0.795 result in Fig. 5.1a is similar to the single best point in the original architecture’s training (0.792), its evolution is greatly steadier, what reinforces confidence in effective learning. Moreover, the curves’ trend indicates that we could further improve performance, even if slightly, by training longer. Interestingly, the validation loss is higher than the training loss, which is highly uncommon; however this behavior is expected and is product of the unusual database pre-processing, more precisely, the removal of training frames with no FG pixel. This causes the training set to be harder than the validation set as the FG pixels (true positives) are harder to correctly predict than the BG ones (true negatives). This is not apparent from the F1-score because this metric takes into consideration both true positive and true negative rates, whereas the loss only takes into account the true positives.

Next, we proceed to deconvolutional architectures (Fig. 5.1c), either with linear or nonlinear deconvolution operations (e.g. either with nonlinear ReLU activation function in-between deconvolutional layers or not), each also further examined with and wthout batch normalization. Deconvolutions are powerful as they give the model the chance to learn the proper up-sampling algorithm, specially the nonlinear version. Nonetheless, they are complicated functions to learn, specially the nonlinear version; indeed, Noh *et al.* [32] state that “although batch normalization helps escape local optima, the space of semantic segmentation is still very large compared to the number of training examples and the benefit to use a deconvolution network for instance-wise segmentation would be canceled”, and evade this obstacle by employing a two-stage training method. In these experiments, we verify that the networks without batch normalization cannot learn the desired mapping: losses remain high and F1-scores low for both sets, besides overtraining is also noticeable. Curiously, the presence of nonlinear activation functions doesn’t seem to alter performance significantly, while the linear version with BN achieved an F1-score of 0.848, the nonlinear-with-BN one attained 0.830; our hypothesis is that this difference arises

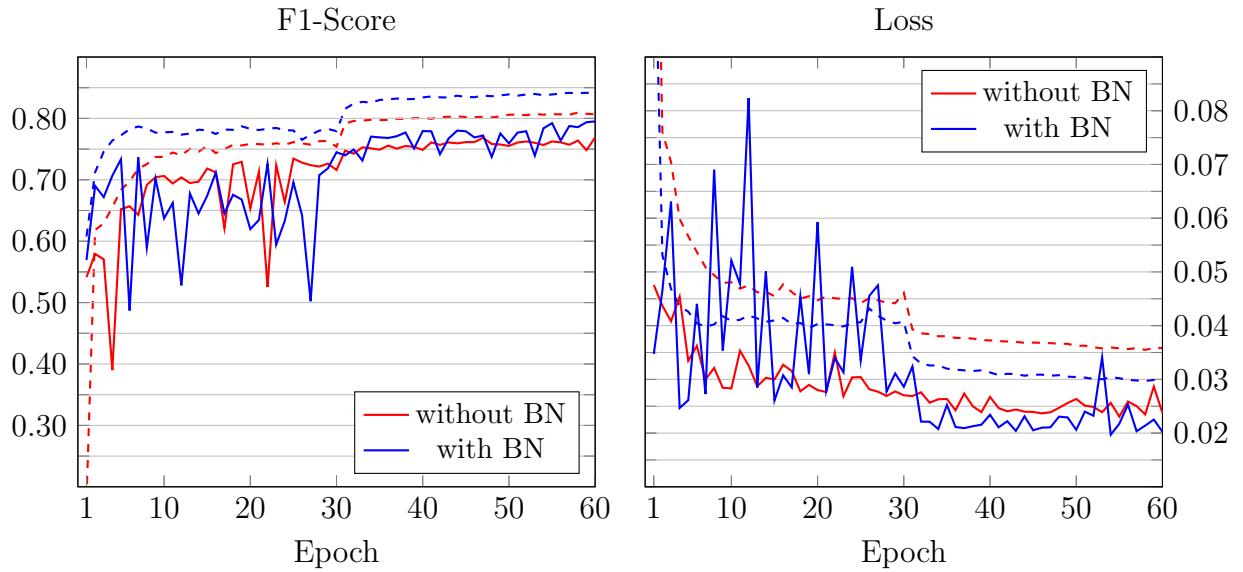


Figure 6.3: F1-score and loss curves for the bilinear up-sampling version of [1] with and without batch normalization (in blue and in red, respectively). Values for both training (dashed lines) and validation (solid lines) are shown. The BN version is consistently better, though it does not reduce the train/val gap. It is also worth noting that the odd behavior of lower-than-training validation loss comes from the unusual dataset split: training set has higher loss because all of its images contain FG pixels, which are harder for the model to correctly classify (true positives), whilst for validation all frames are kept, therefore containing many more BG pixels, which have a higher classification rate (true negatives).

from arguments in [32] mentioned above.

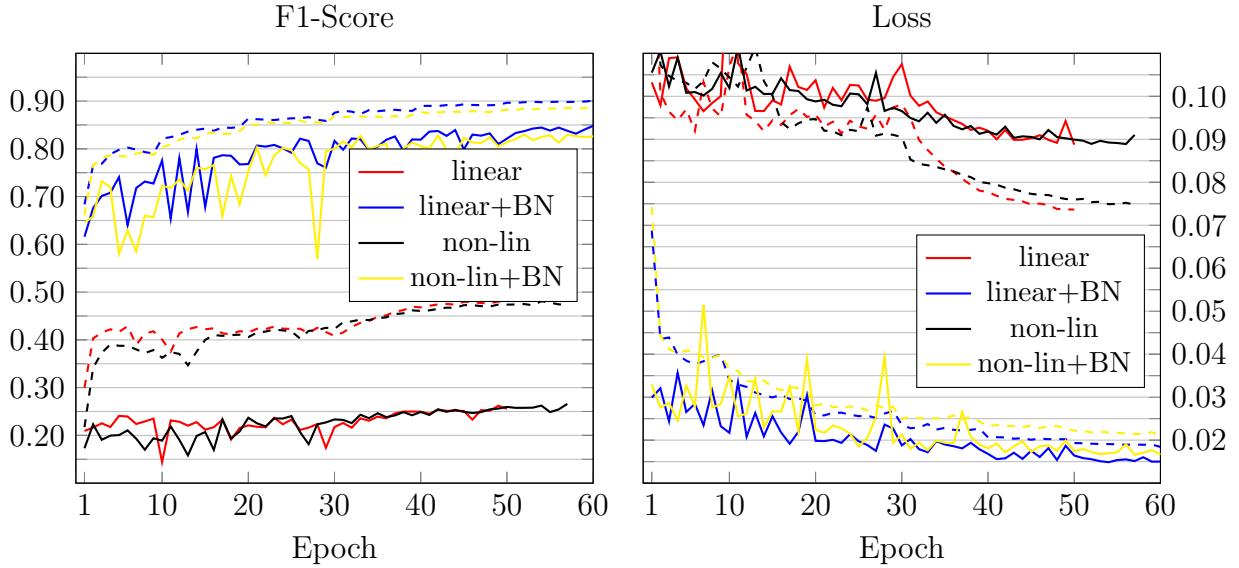


Figure 6.4: F1-score and loss curves for the deconvolutional linear and nonlinear versions of [1] with and without batch normalization. Values for both training (dashed lines) and validation (solid lines) are shown. The versions without batch normalization completely fail to learn the segmentation task, while their counterparts succeed and achieve similar results: 0.848 F1-score for the linear one and 0.830 for the nonlinear. We hypothesize nonlinear deconvolutions are harder to learn and need more complex training schemes to benefit of their full potential [32].

The exciting result of the batch-normalized nonlinear version motivated us to further train this model 40 more epochs, that is up to 100, so as to probe how farther it could go (Fig. 6.5). Performance is indeed better (0.849), but only marginally (0.001), since the curve basically saturates at 60 epochs. In short, the extended training brings no practical benefit for the model at hand.

Yet another problem with deconvolutions is the checkerboard effect they cause when kernel size is not multiple of the stride, which cause uneven overlaps. Although the network theoretically could learn to compensate it and output a balanced map, in reality not only it struggles to avoid the checkerboard patterns, but also models with even overlap (kernel size divisible by stride) frequently cause similar artifacts. Aiming to minimize this issue we only use 1-strided deconvolutions, which are guaranteed to generate even overlaps only, however this effect is still visible even for our

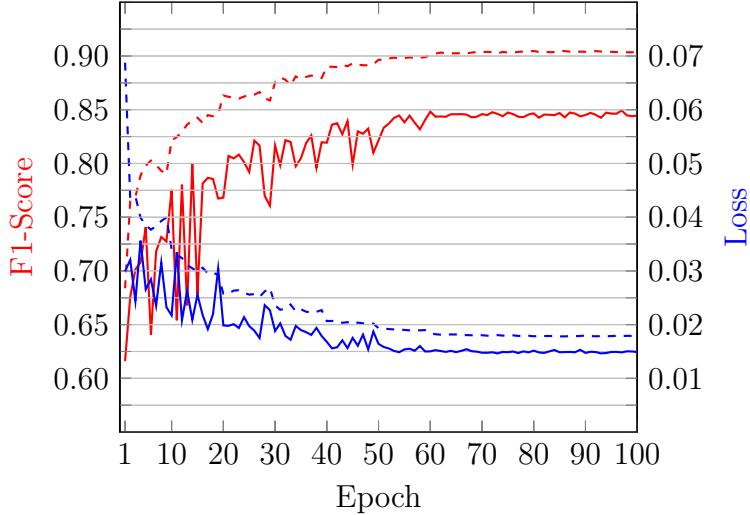


Figure 6.5: F1-score and loss curves for the best deconvolutional model, linear deconvolution with batch normalization. Values for both training (dashed lines) and validation (solid lines) are shown. Further training does not bring any practical benefit for the model since the curve basically saturates at 60 epochs.

best architecture, the non-linear deconvolutional net with batch normalization, as exemplifies Fig. 6.6.

At last, we try the dilated variation (Fig. 5.1b), where all poolings have stride modified to 1 and after each all subsequent operations are dilated by 3 (i.e. after two poolings, operations are dilated by a factor of 9) so as to preserve equivalence with the original model. As all other models, the BN version attains a score approximately 5% higher, though both F1-score and loss fluctuates considerably more on the first half of training, scoring an F1-measure of 0.823 (Fig. 6.7). This value, however, does not surpass the 0.848 of the deconvolutional linear BN architecture. We speculate that the small size of the architecture contributes for the superior performance of the deconvolutional network and as the model deepens the deconvolutional approach deals with an increasingly harder task. Fig. 6.14 shows examples of segmentation results from four different CDNET videos using the best LeNet variation of each adaption scheme.



Figure 6.6: Visible checkerboard patterns in the output image of the best deconvolutional model, the linear deconvolutional net with batch normalization. The deconvolution fails to learn a proper function, which generates this effect even when stride and kernel size are properly chosen.

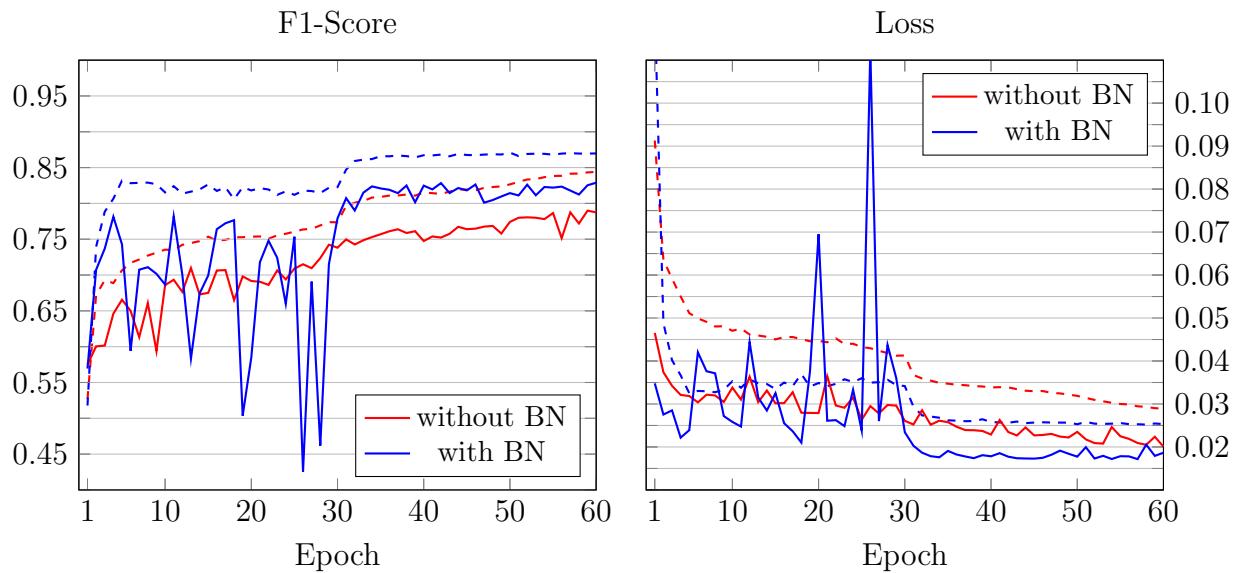


Figure 6.7: F1-score and loss curves for the dilated version of [1] both with and without batch normalization. Values for both training (dashed lines) and validation (solid lines) are shown. The batch normalized model is much noisier, but achieves better performance from the second half of training, ending with a 0.848 validation F1-score.

6.3 Modifications to the ResNet architecture

Residual networks exhibit interesting characteristics [40]: they often perform better than shallower models and at the same time have a clear and simple structure. These components motivated us to experiment with residual networks instead of the more traditional LeNet5-based models described so far. We repeat all previous steps, that is, we implement and evaluate the same structures of up-sampling, deconvolution, and dilation, to convert the architecture into one proper for FG segmentation. The ResNets in [12] presents a greater degree of hyper-parameter setting, namely the size of the model and even the organization of layers, when compared to the LeNet5. Basing ourselves on the FAIR implementation [36], we use the networks designed for the ILSVRC challenge, which deal with 224×224 pixel-images, and those for the CIFAR-10 and CIFAR-100 datasets, which have 32×32 pixel-images as input, since our task is on one side akin to that of the ILSVRC challenge, large high-resolution entry images, and on the other similar to the CIFAR databases, image patches and local context as backbone. Besides the two different base designs, we experiment with three different model sizes for each: 34, 50, 101 (ILSVRC) and 32, 56, 110 (CIFAR).

The curves in Fig. 6.8 show that all ILSVRC-based models have similar results, though we note that the smaller the model, the less its metrics fluctuate. Moreover, no matter the depth, they all perform worse than the LeNet5 with up-sampling of Section 6.2. We suppose the reason for this phenomenon is the substantial loss of spatial information due to the many pooling operations that down-sample the image by a factor of 32, whereas the LeNet5 variation reduces the input by only four times. The fact that the sole CIFAR-derived model with 34 layers achieves the best score so far (0.858) reinforces the hypothesis of too strong down-sampling and consequent loss of local context of the ILSVRC-derived networks. Analogously to the action taken with the best deconvolutional net, we proceed to extend the training of the outstanding model up to 100 epochs and verify a 0.88% gain in performance (Fig. 6.9), attaining a final F1-score of 0.867. Once again the model converges shortly after the 60 epochs mark.

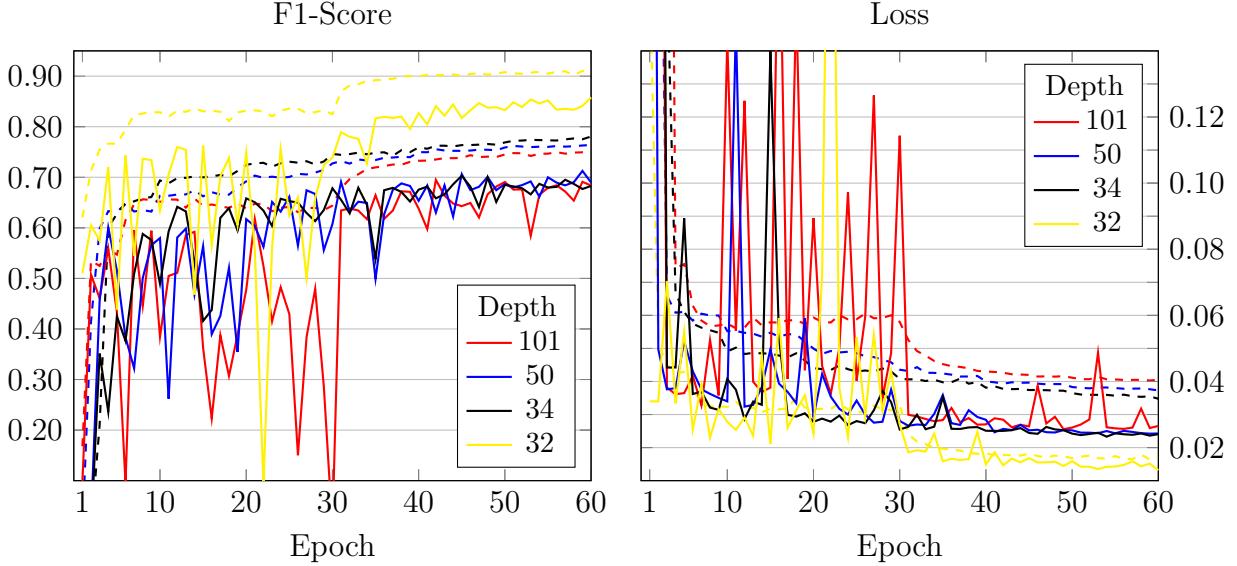


Figure 6.8: F1-score and loss curves for the bilinear up-sampling residual networks. Values for both training (dashed lines) and validation (solid lines) are shown. All ILSVRC-derived nets (depths 34, 50, and 101) perform similarly and significantly worse than their LeNet5 equivalent, we hypothesize that the 32 times spatial down-sampling causes too much spatial information to be lost. Besides, depth does not aid in getting better models, the deeper the model, the noisier its metrics. Clearly, the best performing architecture is the 32-layer CIFAR-based, which achieves a F1-score of 0.858 and only down-samples its input by a factor of 4, like the up-sampling LeNet5 version.

Next, we evaluate segmentation by deconvolutional reconstruction with the ILSVRC-based models (Fig. 6.10a) and (Fig. 6.10b). Just as the upsampling-case, results are globally similar and lean favorably towards the shallower versions (F1-score 2~3% higher). Unexpectedly, however, the nonlinear model (F1-score = 0.842) has a 1% edge over the linear deconvolutional case (F1-score = 0.832) for the ILSVRC networks. As stated before, the CIFAR networks, having higher resolution, achieve better performance on average, albeit only slightly: 0.849 for the linear deconvolutional case and 0.841 for the nonlinear one. Thus, we may deduce that the deconvolution operations accomplish reasonable decoding of the feature representations in spite of great spatial information loss (32 times in the ILSVRC-derived models).

At last, we employ the dilation technique. The CIFAR networks have already little down-sampling (four times), hence we only dilate the second (out of two) max-pooling, consequently obtaining an output map two times smaller than the input.

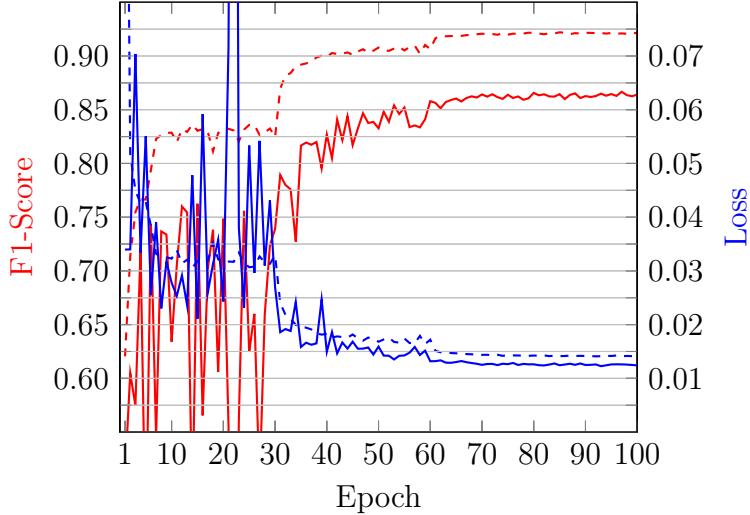


Figure 6.9: F1-score and loss curves for the best up-sampling ResNet model, the 32-layer CIFAR-derived. Values for both training (dashed lines) and validation (solid lines) are shown. The network achieves an F1-score of 0.867. Once again the model converges shortly after the 60 epochs mark.

On the other hand, the ILSVRC nets operate five sub-sampling operations and have, at their end, segmentation results reduced by 32 times, thus, in this case, we dilate the networks twice, obtaining instead a resolution eight times smaller than the input. The 56-layer and the 110-layer models of Fig. 6.11b were trained with mini-batch sizes of 8 and 4, respectively, instead of the default 16 because those models were too large to fit the 8GB GPU memory available. which implies they were trained for many more iterations.

The ILSVRC-derived nets converge to similar values, even though the 34-layer one net attains an F1-score of 0.857 at the 55th epoch, which lowers afterward, whilst the 50-layer one achieves 0.825 at the the last epoch. The 32-layer CIFAR achieves 0.872 of F1, the highest score of all models tested in the present work, while the deeper 56-layer remains at 0.828, comparable with the 50-layer ILSVRC (0.826). We further train the 34-layer ILSVRC-based model and the 32-layer CIFAR-derived one until the 100th epoch, however none of them is improved by this action, even worse, the latter has a poorer F1-score (0.853 instead of 0.857). Fig.6.15 shows examples of segmentation results from four different CDNET videos using the best ResNet variation of each adaption scheme.

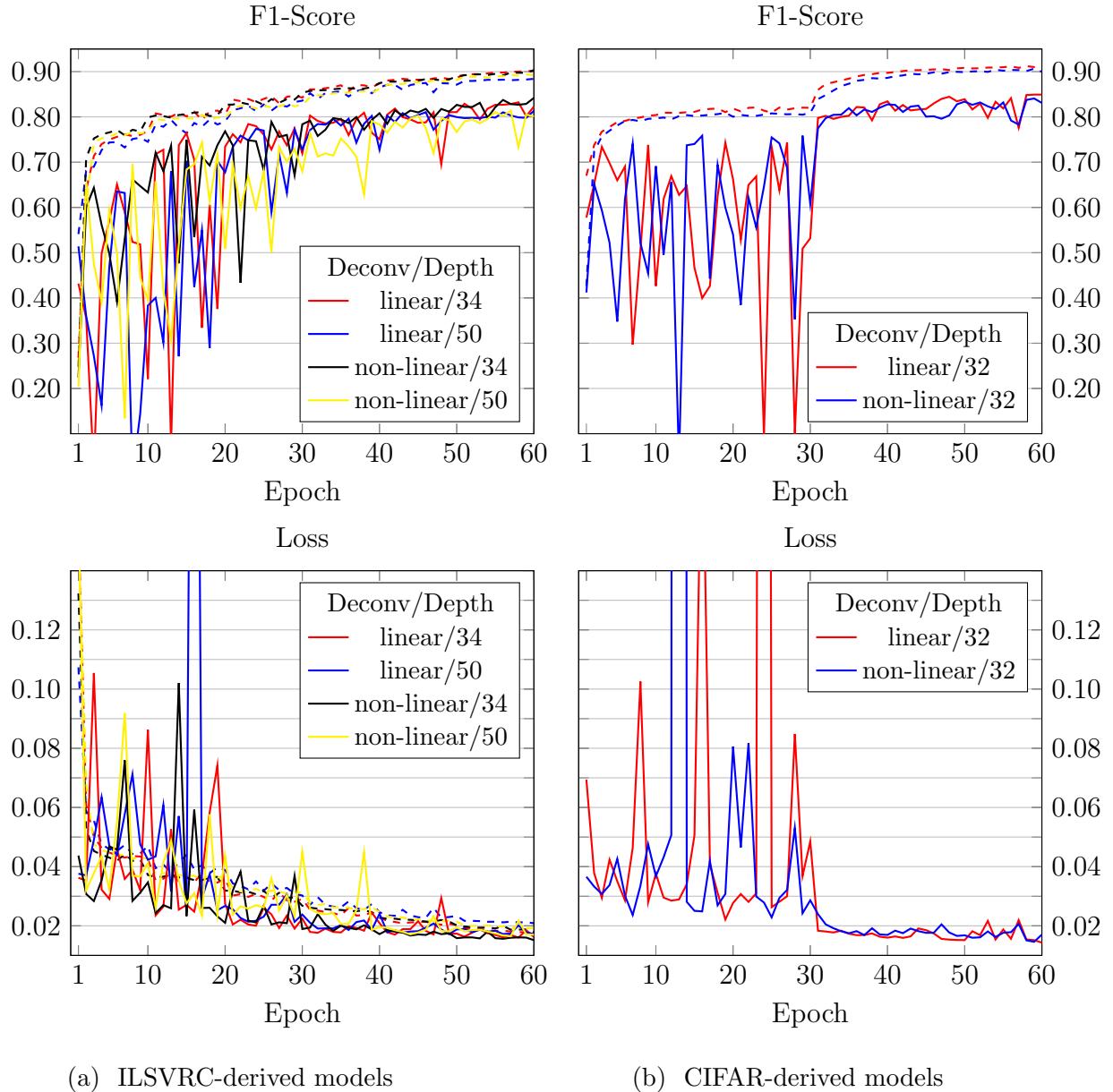


Figure 6.10: F1-score and loss curves for the deconvolutional residual networks. Values for both training (dashed lines) and validation (solid lines) are shown. Once again, the smaller models (34-layer in Fig. 6.10a) have better performance: 0.842 (nonlinear) and 0.832 (linear) for the ILSVRC; and the CIFAR-based nets achieve higher F1-scores: 0.849 (linear) and 0.841 (nonlinear).

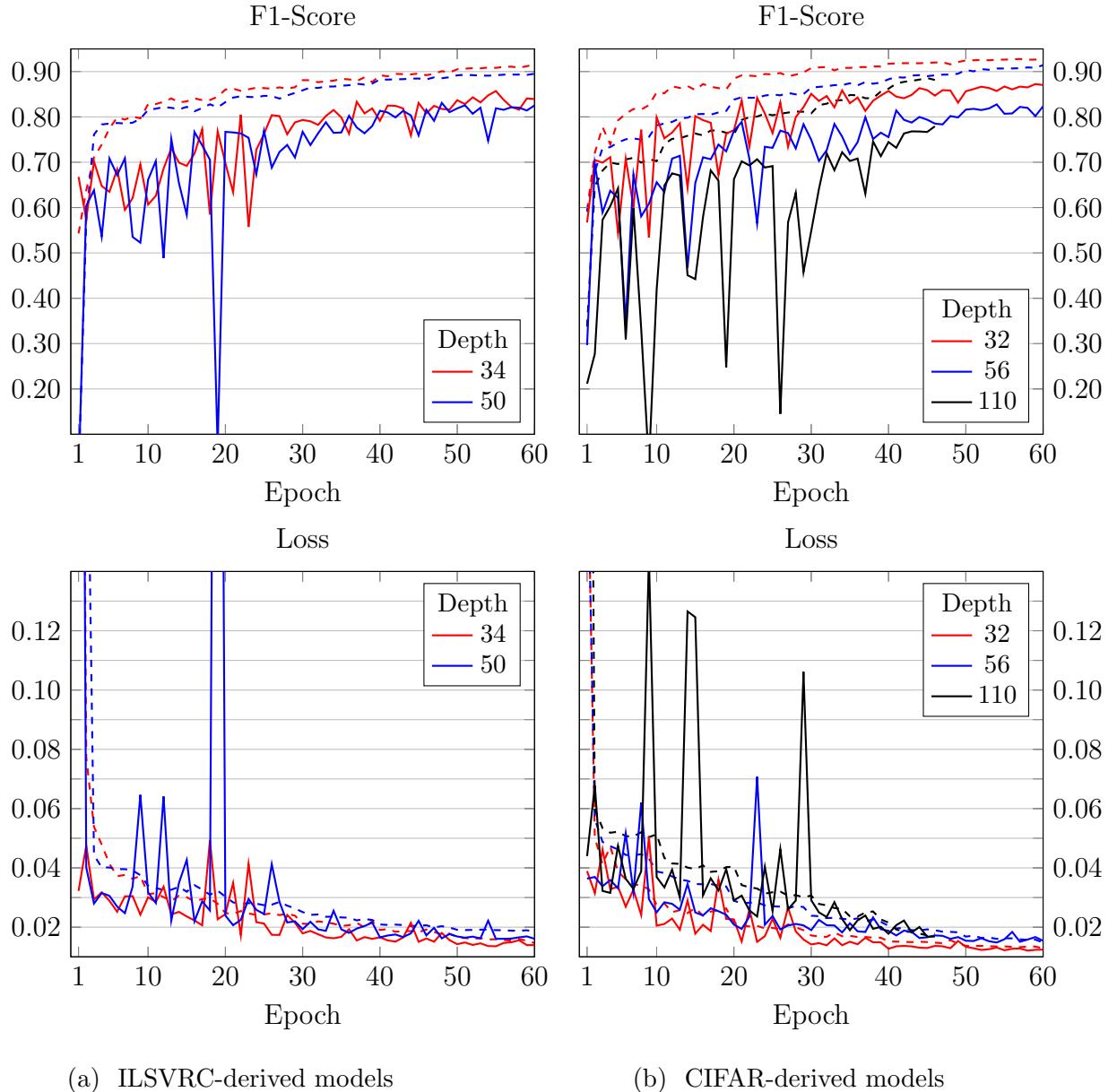


Figure 6.11: F1-score and loss curves for the dilated residual networks. Values for both training (dashed lines) and validation (solid lines) are shown. The 110-layer model of Fig. 6.11(b) had its training halt at 46 epochs, that is the reason why its curve stops shortly after the 40 mark, however it was trained for more iterations since it had 4 frames per mini-batch, as opposed to the 16 standard. Both ILSVRC models appear to converge to similar values, though the 34-layer net attains an F1-score of 0.857 at the 55th epoch. The 32-layer CIFAR achieves the highest score of all models with 0.872 of F1-score, while the deeper 56-layer remains at 0.828, comparable with the 50-layer ILSVRC (0.825).

6.4 Use of pre-trained models

Next, we assess how pre-trained models influence training and if results are indeed better. Furthermore, we analyze whether projection or direct substitution (Section 5.4.2) is more adequate to adapt the 3-channel entry of the original networks to our 2-channel input. There are several models available to be used in this case, but since we had a constraint of time in this work we only tested the ILSVRC networks, since those are more widely available and have Torch implementations that could be easily modified, more precisely we evaluated 34 and 101-layer variations by virtue of the smaller being consistently the best across all experiments, and the larger to measure if more complex networks also benefit from this approach. The sole training configuration we altered for the specific case of pre-trained models was the initial learning rate, which was reduced by a factor of 10 (becoming equal to 0.001), because parameters are not random anymore and fine-tuning requires smaller updates.

Initially, we fine-tuned the 101-layer up-sampling architecture using both types of adaption (Fig. 6.12) and obtained gains in performance for both, while the randomly initialized model had a maximum F1-score of 0.691 within 60 epochs, the pre-trained model with direct substitution adaptation achieved 0.736 and the other 0.728, that is respectively, 4.5% and 3.7% of difference.

Secondly, we proceed to the dilated networks and trained the 34-layer version. Despite the short training (23 epochs), the net whose first convolution was replaced had its F1-score increased by 1.5% and performed exceedingly well, reaching 0.872, on pair with the best randomly initialized model (CIFAR-based dilated residual network) and even lower loss. However, the architecture adapted by the projection scheme (as well as the nonlinear deconvolution variations) failed to learn, that is, its score remained at zero. Those behaviors indicate that indeed the use of pre-trained parameters have the potential to greatly improve the models if hyper-parameters are set correctly, moreover, the direct substitution design are superior to projection.

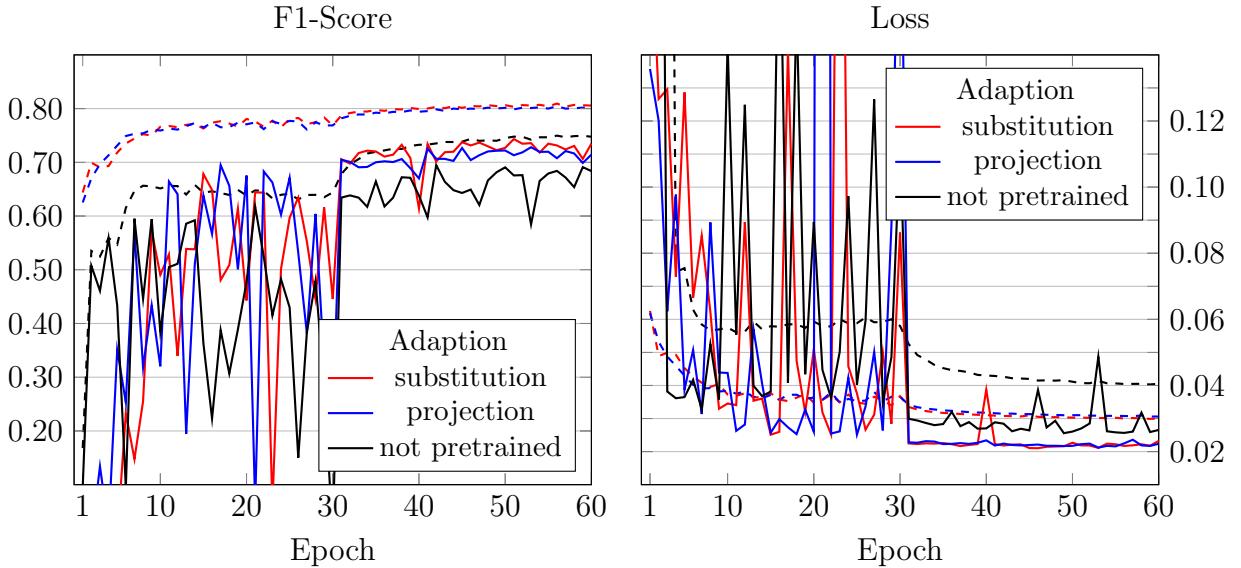


Figure 6.12: F1-score and loss curves for fine-tuned up-sampling residual networks. Values for both training (dashed lines) and validation (solid lines) are shown. Both types of adaption succeed and attain higher scores than their randomly initialized counterpart, the one with projection has an F1-score of 0.728 and the one adapted by direct substitution of 0.736, while the model trained from scratch achieves 0.691.

6.5 Summary

At last, we compare all trained models with regards not only to their F1-score and loss, but also the precision and recall measures individually (Table 6.2), so it is possible to better comprehend how each model has learned and whether they have a bias towards the most predominant class (background). We can instantly see that the two best models are the 32-layer CIFAR-derived dilated network and the pre-trained 34-layer ILSVRC-based dilated one adapted by direct substitution, both with 87.2% of F1-score, which is competitive with the 90.5% reported in [1] while requiring significantly less computation and being much faster, since our models calculate all output pixels simultaneously and the original, one at a time.

Besides, it is clear that most models have similar precision and recall, which means that they were capable of learning to predict the foreground and not to constantly guess the same class. The ILSVRC-based up-sampling networks are the models whose metrics are uneven the most, this phenomenon may be explained by the strong down-sampling factor (32 times) they perform that cannot be later recovered

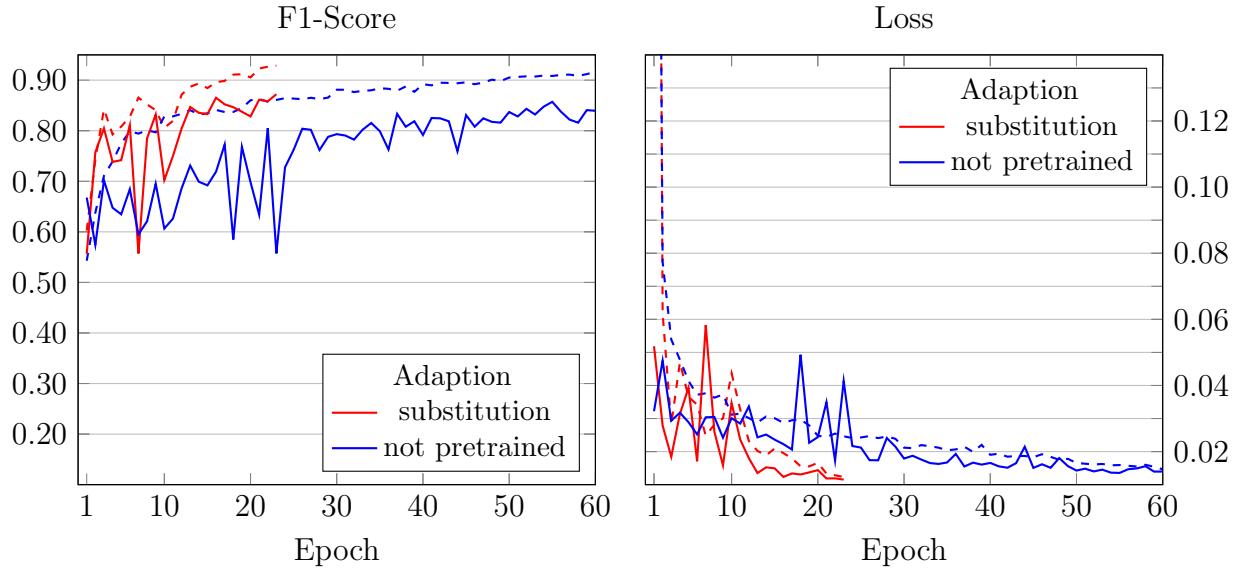


Figure 6.13: F1-score and loss curves for the fine-tuned dilated residual networks. Values for both training (dashed lines) and validation (solid lines) are shown. The pre-trained dilated version with direct adaptation had its training halt at the 23th epoch, however it is obvious the gain in performance over the model randomly initialized; even with such short training it attains 0.872 of F1-score and can still further improve. The network with adaption by projection failed to learn and is not plotted in the figure.

by the simple bilinear interpolation, which amplifies the small initial bias.

Table 6.2: Summary of metrics for the best single epoch of each trained model. The F1-score (**F1**), precision (**Prec**), recall (**Rec**), and loss (**Loss**) measured in the validation set for the best epoch (**Epoch**, outside parenthesis) within the trained interval (**Epoch**, inside parenthesis). Model whose metrics have — instead of values failed to converge. The best models were the 32-layer CIFAR-derived dilated network and the pre-trained 34-layer ILSVRC-based dilated model adapted by direct substitution.

Model			F1	Prec	Rec	Loss	Epoch	
Base	Adaption	Depth	Pretrained					
LeNet5	original [1]	4	no	79.3	79.6	84.0	2.22E-2	30 (65)
	lin deconv	4	no	26.2	16.1	87.1	9.42E-2	49 (50)
	lin deconv w/ BN	4	no	84.9	84.4	85.9	1.48E-2	97 (100)
	non-lin deconv	4	no	26.6	16.4	87.3	9.10E-2	57 (57)
	non-lin deconv w/ BN	4	no	83.0	82.6	83.9	1.72E-2	52 (60)
	up-sample	4	no	76.9	78.3	76.3	2.38E-2	60 (60)
	up-sample w/ BN	4	no	79.5	81.7	78.3	2.02E-2	60 (60)
	dilation	4	no	78.9	77.6	81.5	2.24E-2	59 (60)
	dilation w/ BN	4	no	82.9	81.4	85.3	1.87E-2	60 (60)
CIFAR ResNet	lin deconv	32	no	84.9	86.6	83.8	1.43E-2	60 (60)
	non-lin deconv	32	no	84.1	86.9	81.9	1.46E-2	59 (60)
	up-sample	32	no	86.7	85.9	87.7	1.26E-2	97 (100)
		32	no	87.2	86.8	88.1	1.23E-2	59 (60)
	dilation	56	no	82.8	85.1	81.8	1.48E-2	54 (60)
		110	no	77.9	81.9	76.9	1.68E-2	46 (46)
	lin deconv	34	no	83.2	82.4	84.6	1.62E-2	58 (60)
		50	no	81.3	81.8	81.4	1.72E-2	60 (60)
	non-lin deconv	34	no	84.2	83.4	85.4	1.52E-2	60 (60)
ILSVRC ResNet	non-lin deconv	34	substitution	—	—	—	—	—
		50	no	81.4	86.2	77.7	1.65E-2	57 (60)
	lin deconv	34	no	70.3	76.0	66.2	2.44E-2	45 (60)
		50	no	71.3	75.6	68.2	2.42E-2	59 (60)
	up-sample	101	no	69.5	68.3	71.7	2.84E-2	42 (60)
		101	substitution	73.6	82.0	67.6	2.11E-2	53 (60)
	up-sample	101	projection	72.8	80.6	67.1	2.12E-2	53 (60)
	dilation	34	no	85.7	84.9	87.0	1.35E-2	55 (60)
		34	substitution	87.2	88.8	86.0	1.15E-2	23 (23)
	dilation	34	projection	—	—	—	—	—
		50	no	82.6	82.8	82.9	1.64E-2	51 (60)

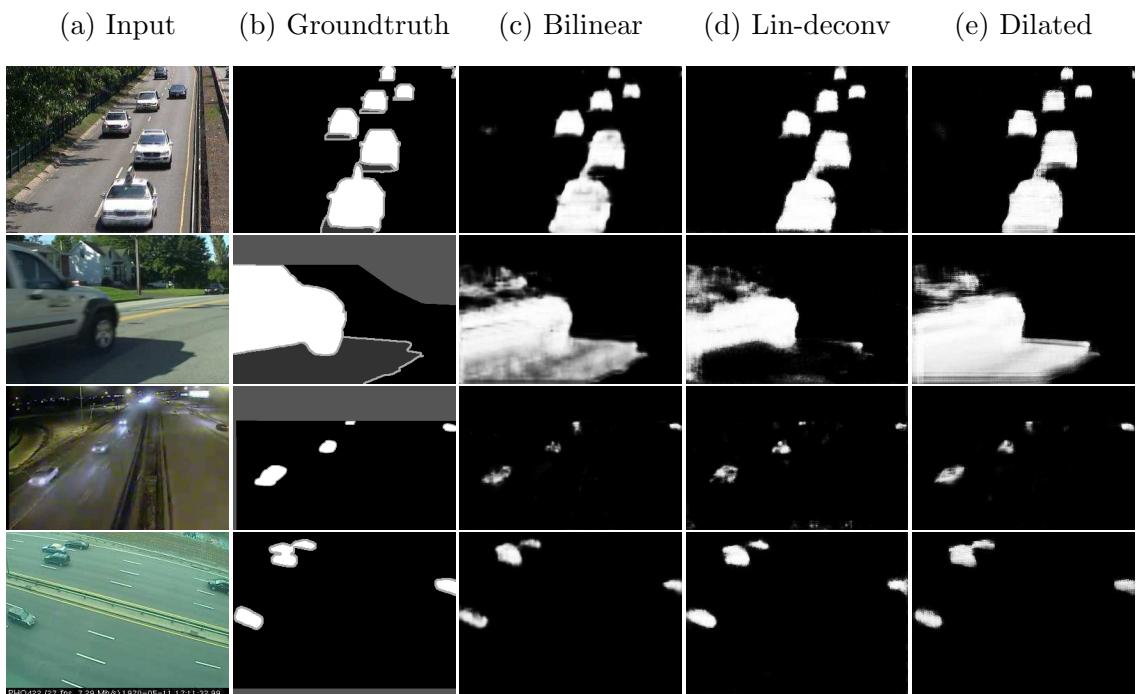


Figure 6.14: Examples of segmentation results for the CDNET database using either one of the three adaptations on the LeNet-based model. All three models use batch normalization. Note that the linear deconvolution model is the only one among the three capable of dealing with shadows, while de dilated model is the worst one.

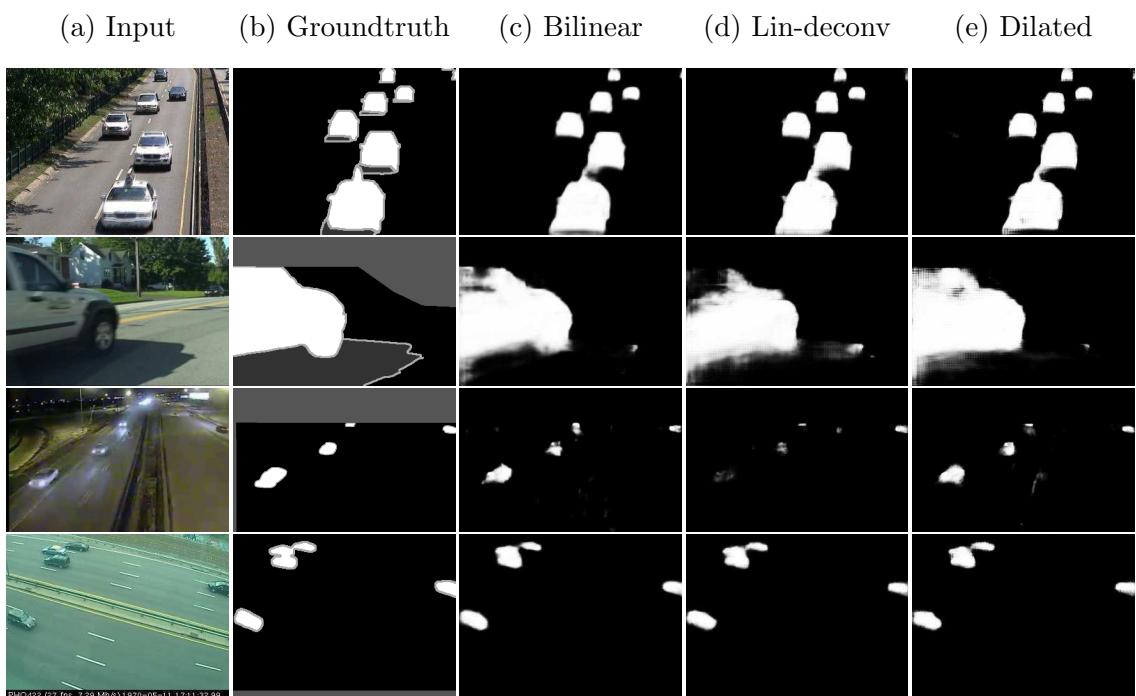


Figure 6.15: Examples of segmentation results for the CDNET database using either one of the three adaptations on the ResNet-based model. Note that the residual networks were capable of learning to ignore shadows.

Chapter 7

Future work and conclusion

Throughout this work we studied different deep learning networks to deal with the foreground segmentation task in videos from surveillance cameras. We proposed fast and efficient models that compute the pixel-wise segmentation map in real-time taking as input temporally aligned reference and a target frames. The proposed techniques are competitive with current stat-of-the-art methods for the CDNET database [4]. Also, we analyzed the effectiveness of transfer learning when both task domain and input representations are different.

7.1 Future work

This work focused on training networks through optimization of the binary cross entropy as the cost function to detect the presence of foreground objects in surveillance videos. However, we verified that this is not ideal for binary segmentation, since it does not take into account the class imbalance natural to the FG segmentation task. We could try setting different weights to each class in order to compensate for this issue, but directly employing the F1-score (Dice coefficient) as the cost function is the more reasonable alternative because it inherently accounts for the false positive and negative rates, and that is the function we ultimately want to maximize. Thus replacing the the cross entropy by the Dice loss should contribute in training better models. Another straightforward modification is fine-tuning of the detection threshold at the end, which may boost the F1-score a few percentiles by equalizing precision and recall, or more radically substituting the fixed pre-defined

threshold by an adaptive one.

Furthermore, we were able to demonstrate that using dilation to adapt a classification network into a FG segmentation one indeed works, thus we intended to further experiment with this approach by increasing the dilation factor until the image resolution is kept intact throughout the whole model in order to determine where the computational cost surpasses the benefit of increased performance.

In this work, we did not investigated any pre-trained version of CIFAR-based models, even though such architecture achieved the best performances. Hence, we should train these models since that seems to be a simple and effective manner of enhancing the F1-score. Moreover, we assumed that bilinear interpolation was the best up-sampling method with non-learnable parameters. Nevertheless, we could use Nearest Neighbor (NN) up-sampling that also fulfills the no learning requirement while being even simpler. We should then inspect whether bilinear or NN is better.

At last, we ought to test the best performing models in the challenging VDAO dataset and assess if the proposed networks are robust even when dealing with moving cameras in extremely cluttered industrial environments.

7.2 Conclusion

This work verified that classification models can be successfully transposed to the FG segmentation domain and that transfer learning is effective even when models' input is essentially different. Furthermore, we demonstrated that, for the task at hand, dilation is a better approach than direct up-sampling and reconstruction by deconvolution, and that preserving local spatial context contributes to the network's classification power. Besides, we corroborate the recent trend that deeper models are not necessarily always better as our shallower networks consistently outperform their deeper counterparts.

Finally, we obtain a real-time foreground segmentation algorithm that is competitive with the state-of-the-art for the CDNET dataset (at least within the considered

video categories) and which may be readily enhanced and possibly surpass the currently best methods.

Bibliography

- [1] BRAHAM, M., VAN DROOGENBROECK, M., “Deep background subtraction with scene-specific convolutional neural networks”. In: *International Conference on Systems, Signals and Image Processing (IWSSIP)*, pp. 1–4, Bratislava, Slovakia, May 2016.
- [2] CUEVAS, C., MARTÍNEZ, R., GARCÍA, N., “Detection of stationary foreground objects: a survey”, *Computer Vision and Image Understanding*, v. 152, pp. 41–57, November 2016.
- [3] COLLOBERT, R., KAVUKCUOGLU, K., FARABET, C., “Torch7: a matlab-like environment for machine learning”. In: *BigLearn, NIPS Workshop*, n. EPFL-CONF-192376, 2011.
- [4] WANG, Y., JODOIN, P.-M., PORIKLI, F., *et al.*, “CDnet 2014: an expanded change detection benchmark dataset”, pp. 387–394, June 2014.
- [5] CHRISTIANSEN, P., NIELSEN, L. N., STEEN, K. A., *et al.*, “DeepAnomaly: combining background subtraction and deep learning for detecting obstacles and anomalies in an agricultural field”, *Sensors*, v. 16, n. 11, pp. 1904, November 2016.
- [6] DA SILVA, A. F., THOMAZ, L. A., CARVALHO, G., *et al.*, “An annotated video database for abandoned-object detection in a cluttered environment”. In: *International Telecommunications Symposium (ITS)*, pp. 1–5, August 2014.
- [7] GOODFELLOW, I., BENGIO, Y., COURVILLE, A., *Deep learning*. MIT Press, 2016. Available at: <http://www.deeplearningbook.org>.

- [8] SAHAMI, M., DUMAIS, S., HECKERMAN, D., *et al.*, “A Bayesian approach to filtering junk e-mail”. In: *Learning for Text Categorization: Papers from the 1998 workshop*, v. 62, pp. 98–105, July 1998.
- [9] MCCULLOCH, W. S., PITTS, W., “A logical calculus of the ideas immanent in nervous activity”, *The bulletin of mathematical biophysics*, v. 5, n. 4, pp. 115–133, December 1943.
- [10] RUMELHART, D. E., HINTON, G. E., WILLIAMS, R. J., *Learning internal representations by error propagation*, Report, DTIC Document, January 1986.
- [11] HINTON, G. E., OSINDERO, S., TEH, Y.-W., “A fast learning algorithm for deep belief nets”, *Neural computation*, v. 18, n. 7, pp. 1527–1554, July 2006.
- [12] HE, K., ZHANG, X., REN, S., *et al.*, “Deep residual learning for image recognition”. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.
- [13] KRIZHEVSKY, A., SUTSKEVER, I., HINTON, G. E., “ImageNet classification with deep convolutional neural networks”. In: *Advances in neural information processing systems*, pp. 1097–1105, January 2012.
- [14] ROSENBLATT, F., “The perceptron: a probabilistic model for information storage and organization in the brain.”, *Psychological review*, v. 65, n. 6, pp. 386, November 1958.
- [15] HORNIK, K., “Approximation capabilities of multilayer feedforward networks”, *Neural networks*, v. 4, n. 2, pp. 251–257, 1991.
- [16] ROMERO, A., BALLAS, N., KAHOU, S. E., *et al.*, “FitNets: hints for thin deep nets”, *CoRR*, v. abs/1412.6550, January 2014.
- [17] HUBEL, D. H., WIESEL, T. N., “Receptive fields, binocular interaction and functional architecture in the cat’s visual cortex”, *The Journal of physiology*, v. 160, n. 1, pp. 106–154, January 1962.
- [18] FUKUSHIMA, K., “Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position”, *Biological cybernetics*, v. 36, n. 4, pp. 193–202, April 1980.

- [19] LECUN, Y., BOTTOU, L., BENGIO, Y., *et al.*, “Gradient-based learning applied to document recognition”, *Proceedings of the IEEE*, v. 86, n. 11, pp. 2278–2324, November 1998.
- [20] IOFFE, S., SZEGEDY, C., “Batch normalization: accelerating deep network training by reducing internal covariate shift”. In: *Proceedings of The 32nd International Conference on Machine Learning*, Lille, France, July 2015.
- [21] LONG, J., SHELHAMER, E., DARRELL, T., “Fully convolutional networks for semantic segmentation”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3431–3440, June 2015.
- [22] YU, F., KOLTUN, V., “Multi-scale context aggregation by dilated convolutions”, May 2016.
- [23] WREN, C. R., AZARBAYEJANI, A., DARRELL, T., *et al.*, “Pfinder: real-time tracking of the human body”, *IEEE Transactions on pattern analysis and machine intelligence*, v. 19, n. 7, pp. 780–785, July 1997.
- [24] GALLEGOS, J., PARDAS, M., LANDABASO, J.-L., “Segmentation and tracking of static and moving objects in video surveillance scenarios”. In: *2008 15th IEEE International Conference on Image Processing*, pp. 2716–2719, October 2008.
- [25] STAUFFER, C., GRIMSON, W. E. L., “Adaptive background mixture models for real-time tracking”. In: *Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on.*, v. 2, June 1999.
- [26] LI, L., LUO, R., MA, R., *et al.*, “Evaluation of an IVS system for abandoned object detection on pets 2006 datasets”. In: *Workshop on Performance Evaluation of Tracking and Surveillance*, pp. 91–98, December 2006.
- [27] LIU, N., WU, H., LIN, L., “Hierarchical ensemble of background models for PTZ-based video surveillance”, *IEEE transactions on cybernetics*, v. 45, n. 1, pp. 89–102, January 2015.

- [28] MOO YI, K., YUN, K., WAN KIM, S., *et al.*, “Detection of moving objects with non-stationary cameras in 5.8 ms: bringing motion detection to your mobile device”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 27–34, June 2013.
- [29] THOMAZ, L. A., DA SILVA, A. F., DA SILVA, E. A., *et al.*, “Abandoned object detection using operator-space pursuit”. In: *Image Processing (ICIP), 2015 IEEE International Conference on*, pp. 1980–1984, September 2015.
- [30] MADDALENA, L., PETROSINO, A., “Stopped object detection by learning foreground model in videos”, *IEEE transactions on neural networks and learning systems*, v. 24, n. 5, pp. 723–735, January 2013.
- [31] DENG, J., DONG, W., SOCHER, R., *et al.*, “ImageNet: a large-scale hierarchical image database”. In: *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pp. 248–255, IEEE, June 2009.
- [32] NOH, H., HONG, S., HAN, B., “Learning deconvolution network for semantic segmentation”. In: *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1520–1528, 2015.
- [33] CHEN, L.-C., PAPANDREOU, G., KOKKINOS, I., *et al.*, “Deeplab: semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected CRFs”, , June 2016.
- [34] SIMONYAN, K., ZISSERMAN, A., “Very deep convolutional networks for large-scale image recognition”. In: *International Conference on Learning Representations (ICLR)*, May 2015.
- [35] GOYETTE, N., JODOIN, P.-M., PORIKLI, F., *et al.*, “Changedetection. net: a new change detection benchmark dataset”. In: *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pp. 1–8, June 2012.
- [36] RESEARCHERS, F. A. I., “Torch implementation of ResNet and training scripts”, available at: <https://github.com/facebook/fb.resnet.torch>.

- [37] GLOROT, X., BENGIO, Y., “Understanding the difficulty of training deep feedforward neural networks”. In: *Aistats*, v. 9, pp. 249–256, March 2010.
- [38] HE, K., ZHANG, X., REN, S., *et al.*, “Delving deep into rectifiers: surpassing human-Level performance on ImageNet classification”. In: *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.
- [39] YOSINSKI, J., CLUNE, J., BENGIO, Y., *et al.*, “How transferable are features in deep neural networks?” In: *Advances in neural information processing systems*, pp. 3320–3328, December 2014.
- [40] HE, K., ZHANG, X., REN, S., *et al.*, “Identity mappings in deep residual networks”, *arXiv preprint arXiv:1603.05027*, , October 2016.