FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Detecting garment and its landmarks

Daniel Fernandes Gomes

WORKING VERSION

Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Luís Filipe Teixeira

August 31, 2017

Detecting garment and its landmarks

Daniel Fernandes Gomes

Mestrado Integrado em Engenharia Informática e Computação

August 31, 2017

Abstract

Garment folding is a task happening daily at our homes, retail and industry. When put into numbers, over a lifetime people spend on average 375 days performing this chore, while employees at a store may fold the same shirt 119 times per day.

Despite the associated repetitive characteristics of this task, its automation is still far from being achieved mainly due to the large number of possible configurations that a crumpled piece of clothing may assume. In general, highly deformable objects still present large challenges for both fields of Robotics and Computer Vision.

We attempt to offer a contribution to the garment folding automation by addressing the recognition of clothing pieces without much constraints on their pose or wrinkling, mimicking a most realistic scenario as possible. Such capability would enable a folding robot to choose and adapt its execution plan to the current clothing category.

Because the considered problem revolves around clothe recognition, this work may also be of the interest of many other clothe related software applications such as recommendation systems existing on e.g., online e-commerce platforms, or intelligent surveillance setups that require tracking of people by their clothing description.

Some work has been produced using Machine Learning techniques that, in general, consist on extracting a set of engineered features from the source image and then applying classification algorithms (e.g., Support Vector Machines) to find the associated clothing category and or pose. With the recent success of Convolutional Neural Networks, where features extraction is incorporated in the learning process, on the object classification problem, these have been preferred in favor of the previous pipelines.

We apply Deep Learning techniques on images containing a single piece of clothing in a flat, wrinkled and semi-folded pose, existing on a clean background with the goal of classify and localize each piece. Furthermore, its relevant landmarks (shoulders, legs, crotch, etc) are equally treated. We train and evaluate our solution using the datasets produced by CTU at the CloPeMa project.

Resumo

A dobragem de vestuário é uma atividade que acontece diariamente nas nossas casas, na indústria e no retalho. Quando expressa em números, cada pessoa passa 375 dias da sua vida a realizar esta tarefa, enquanto funcionários de uma loja poderão dobrar a mesma t-shirt até 119 vezes por dia.

Apesar das repetitivas características associadas a esta tarefa, a sua automação está ainda longe de ser alcançada dado o grande número de possíveis configurações que uma peça de roupa pode assumir. Em geral, objetos altamente deformáveis ainda apresentam grandes desafios para ambos as áreas de estudo Robótica e Visão por Computador.

Na tentativa de oferecer uma contribuição à automação da dobragem de vestuário, o reconhecimento de peças de roupa será aborado, sem que muitas restrições sejam na sua posse ou enrugamento, com o objetivo de simular um cenário o mais realista quanto possível. Esta capacidade permitirá a um robô escolher e adaptar o seu plano de execução à peça de roupa de então.

Porque o problema considerado gira em torno de reconhecimento de peças de roupa, este trabalho poderá ainda ser do interesse de outras aplicações tais como: sistemas de recomendação existentes, por exemplo, em plataformas comércio online, ou instalações de vigilância inteligentes que necessitem de detetar pessoas com base na descrição do seu vestuário.

Algum trabalho tem já sido produzido utilizando técnicas de Machine Learning que, no geral, consistem na extração de conjuntos de características da imagem seguida da aplicação de algoritmos de classificação (e.g., Support Vector Machines) para encontrar a posse ou categoria associada. Com o recente sucesso das Redes Neuronais Convolucionais, onde a extração de características é incorporada no processo de aprendizagem, no problema de reconhecimento de objetos, estas têm vindo a ser preferidas em detrimento das anteriormente utilizadas cadeias.

Técnicas de Deep Learning serão aplicadas em imagens que contêm uma peça de roupa em poses plana, enrugada e semi dobrada, e cujo o seu fundo é sólido, com o objetivo de classificar e localizar cada peça. Os pontos de interesse de cada peça (ombros, pernas, entre-pernas, etc) são igualmente tratados. Ambas as fases treino e a avaliação da solução apresentada são realizados utilizando os conjuntos de dados produzidos pela CTU no decorrer do projeto CloPeMa.

Acknowledgements

In this new digital era, information is more accessible then ever, being openly distributed through these worldwide connected pages that we shortly call web. Such library is a powerful tool that helps to put into motion ideas and dreams of many.

I would like, therefore, to start by thanking to all authors, that through articles, tutorials, lessons or documentaries share their own work, or interpretations of works produced by others. Such documents were a big influence to reach this dissertation, and continued to be during its development. I thank particularly the CS231n for Visual Recognition professors from Stanford who share their lectures on the course website.

I then thank professor Luis Filipe Teixeira that accepted my initial proposal and gave me the necessary guidance to transform it into the dissertation that is exposed in this document. I also thank the Graphics, Interaction and Gaming group members who opened their doors for me and included me in their fortnightly reunions that much helped me on the assessment of dissertation good progress.

I finally thank to my family and friends who through these years supported me in the pursue of my own goals and dreams; and, in one way or another, helped to shape my curious and problem solving oriented personality.

Daniel Fernandes Gomes

"If you just have a single problem to solve, then fine, go ahead and use a neural network. But if you want to do science and understand how to choose architectures, or how to go to a new problem, you have to understand what different architectures can and cannot do."

Marvin Minsky

Contents

List of Figures

List of Tables

Abbreviations

CNN	Convolutional Neural Networks
FCNN	Fully Convolutional Neural Network
CV	Computer Vision
SVM	Support Vector Machine
DL	Deep Learning
ANN	Artificial Neural Network
ML	Machine Learning
MLP	Multilayer Perceptron
RNN	Recursive Neural Networks
SVM	Support Vector Machine
mAP	mean Average Precision
IoU	Intersection over Union
VOC	PASCAL VOC, PASCAL Visual Object Classes Challange
ILSVRC	ImageNet Large Scale Visual Recognition Challange
NAG	Nesterov accelerated gradient

Chapter 1

Introduction

1.1 Context

Since always the automation of repetitive and time consuming tasks has been the focus of many areas as its accomplishment allows to free people to perform others, develop them quicker and with reduced costs. Beginning with mechanical solutions, then electro-mechanic and later using sensors and clever algorithms a set of simple ones were, and are being, solved. These days with the latest advances in Machine Learning and Computer Vision a broader set, where the input domain tends to be much larger, are being taken into consideration, e.g, autonomous people and goods transportation, or general purpose product manufacturing.

In this context, many of our home tasks are also included, in particular the laundry chores. Although, washing and drying subtasks can be considered already solved with the existing electro-mechanic solutions, folding (and ironing) is still a time consuming task to be solved. Furthermore, this task does not only takes place in our homes but in retail stores and clothing manufacturing facilities as well, where its cost directly affects the goods we purchase price. When put into numbers, over a lifetime people spend on average 375 days performing this chore [Ell16], while employees at a store may fold the same shirt 119 times per day [Ron14].

It is clear that the design and development of a general purpose laundry machine imposes a large variety of challenges of not trivial answer, that range from psychical aspects — should it hold one general purpose actuator or multiple task specific ones?; to cognitive — What level of perception it is required to be capable of folding garment?; to its introduction in the market — Should it be business or people oriented? Nevertheless, it necessary to start from somewhere, explore, test and validate possibilities. We assume that an autistic machine, without any external world information should not be capable folding all, not even a large portion of garment and, as a consequence, perception capabilities are required. Such information would enable the machine, for example, to choose and adapt its execution plan to the current clothing category and pose.

Perception is a good starting point as it is probably the one that presents higher levels of uncertainty and technical challenges; can be studied almost without considering the remaining; and its

unlocking could influence the answer of multiple remaining questions. Adding to all of these positives aspects, perception is a subject being actively studied in the academy and recently have seen promising advances, with garment recognition being of interest of many other applications as well e.g., online e-commerce platforms that want to make suggestions based on image information, or intelligent surveillance systems that want to track people based on the clothing description.

In 2012, [KSH12] using a convolutional Neural Network was able to reduce the error rate in roughly 10%, when comparing to previous existing solutions, in the ILSVRC-2010 and ILSVRC-2012 competitions datasets. This was mainly possible due to the great increase in computing power — mainly provided by GPUs; the large amount of data available — today, ImageNet, the image database associated to the ILSVRC competition, provides 14M images, organized into 21k synsets; and a few clever configurations on the network architecture. This rapid error decrease motivated other Neural Net based solutions in the following years on the same and other image competitions with continuous improvement reported. In 2015, [HZRS15] surpassed human level on image classification. With this, the world wide major companies also showed their interest in this technology, its research and development, having open sourced their deep learning frameworks: Google Tensorflow, Microsoft Cognitive Toolkit and Facebook Caffe2.

Thence, in this dissertation, we address the recognition of garment, having in mind the laundry folding task automation, using convolutional Neural Networks.

1.2 Problem

Object visual recognition is one fundamental problem in Computer Vision to be solved. Being able to identify objects existing in a scene is one essential capability not only for clothing related applications but for almost all systems that need to interpret and or interact with the real world.

This might seem a trivial task to us Humans as in our heads most of it happens beyond of our conscientious aware yet, when considering that an image is just a grid of numbers, we quickly realize that finding a set of rules that would enable the recognition of all kinds, or even one kind, of objects is an almost impossible task. This large distance between the raw data information and the interpretations attributed to it, commonly is referred to as the semantic gap problem.

Recognition of pieces of clothing or in general, highly deformable objects, takes this problem to its highest level of difficulty as one object may assume multiple poses and deformations and thus the intra-class variability may be much larger than the inter-class. Furthermore, although recent works have been able to achieve great results on the image classification problem, even outperforming Human level [HZRS15], this might not be enough for many practical applications, in particular robotic systems and the folding of garment. In fact, the object recognition is a vague term that appears associated with a group of more specific problems:

Image classification from a predefined set of classes, one characteristic class, is assigned to each image. Because only one label is generated, it might be argued that this is a subjective and Human biased problem. Because of its simplicity this is a good problem to benchmark the algorithm generalization capacity.

Object location to each image N class and bounding boxes, surrounding the corresponding objects, are assigned to the N objects of interest in each image. Where N is known and constant. Typically, the bounding boxes are often defined in terms of x, y, coordinates with and height.

Object detection is a more general version of object location: the number of objects in each image is unknown.This characteristic introduces some additional challenges, both in terms of how to build a module that give variable number of answers and how to evaluate it, since do not exist a one-to-one relation between the predictions and ground truths. Because of that, the proposals must be also ranked, i.e., ahead from the label and bounding-box, a confidence score is required.

Semantic segmentation to each image pixel one class, from a set of predefined classes, is assigned. It should be noticed that in semantic segmentation does not care about regions, only pixels.

Instance segmentation also known as simultaneous detection and segmentation consists on attributing one segmentation contour and class to each instance of interest on the image.

One other problem that does do not appear so much in general image perception literature, and it is of interest when studying garment perception, in particular for the development of a folding machine, is the recognition of grasping locations. As it might be expected, this is a problem mostly studied in the context of robotics, for both rigid and non rigid objects, and consists on determining points of interest that would allow the correspondent object handling.

Although some rule based primitive solutions have been developed e.g. template matching, more recently recognition problems have been addressed using ML approaches. Yet, using the raw image i.e., each pixel is a feature, resulting in millions of features, as the input of traditional ML algorithms is not possible, due to the known dimensionality curse problem. To solve this, one approach is to create a smaller set of representative features e.g., SIFT, HoG that describe these points of interest with vectors of representative values. Clustering these descriptors into a small set of groups is a technique known as Bag of Visual Words. Finally, in case of classification, these *words* are fed into a classifier e.g., Support Vector Machines [LCA14]. More recently, with the success of [KSH12], this pipeline has been replaced a CNN where a set of convolutional layers play the role of feature extractor and fully connected layers perform the classification (or regression, in the case of a location problem).

When solving using the ML, training and learning consist in practice on adapting a model's (learnable) parameters such that one certain objective function is minimized/maximized (when the goal is to minimize we refer to this as a cost function). Depending on the type of correct information that the system has access to when learning these can be grouped into three subcategories: (a) **Supervised learning** for each prediction, one correct annotation is provided. (b) **Reinforcement learning** positive or negative reinforcements are provided to the system. (c) **Unsupervised**

learning no information about the correct answer exists. Although some works have been conducted using (b) and (c), when solving object recognition problems mostly (a) is used. This raises the necessity of data quantity and quality (of annotations) for training these systems. In particular, for problems where annotating each image is not a trivial process, which is the case of instance segmentation.

Consequently, we address the scenario where an image containing a single piece of clothing, flat, wrinkled and semi-folded exists on a clean background and our goal is to localize and classify the present garment piece and its landmarks. A landmark, e.g., neckline-left, right-armpit, right-sleeve-inner, is a two dimensional point. Each garment class has a different amount and types of landmarks e.g., towels have four, t-shirts and long t-shirts have both 12. Because of that, when framing this garment+landmark detection problem into the previously described object recognition ones, it could be formulated in two different ways: (a) garment finding as object localization, followed by conditional, class specific, landmarks finding, also as object localization. (b) Finding all landmarks existing in the image independently of the garment class as object detection, and the garment piece as object location.

Although (a) might seem a simpler solution to conceive and get results, this is a more inefficient solution.1. It requires one different sub-model for each n garment categories, being that many landmarks are shared between garment categories e.g., a sleeve of a hoody is similar to a sleeve in jacket. 2. On the other hand, ANNs work by building more and more complex representations as the depth progresses, so it is probable that the landmarks representations are inferred by the global classifier+localizer model; or at least, its inclusion in the global model should help the learning process, as reported in [LLQ+16]. Adding to this, when considering more general cases e.g., a top view of a laundry bin, the garment classification might not be possible with good accuracy and detecting with good confidence certain landmarks, that would allow a machine to grasp a piece of clothing for further recognition analysis, might be just as valuable.

Therefore, we consider the detection of landmarks and classification+localization of garment independently. We further combine both into the same network, with the expectation of achieve a more efficient and and better performing model.

1.3 Contributions

During the development of this MSc thesis, multiple visual recognition techniques were explored, however our focus stays on the detection of garment and its landmarks existing in an image and it landmarks. The main novel components of this work can be summarized into:

1. Deep Learning for Visual Recognition offers a summary on the main concepts, at the field current level of understanding, and with particular focus on convolutional Networks for visual recognition tasks.

2. A compilation of the existing related works on both general purpose recognition and garment specific, is performed in Related Work. Similarly, the publicly available sources of garment data and DL frameworks for Python are described in Available resources.

3. We augment the a dataset produced by CTU at the EU-FP7 CloPeMa project, to be possible to use it for garment localization and with it train and test a convolutional Neural Network, that achieves 100% accuracy on garment classification, 82% in garment classification+localization, and 36.1% mAP on landmark detection (when comparing with the ground truth anchor boxes). We describe it in Localizing garment and detecting landmarks.

4. Currently there is no standard for graphically represent ANNs architectures, resulting in self made diagrams that are often used to explain only the model novel aspects. We use the UML components diagram together with textual descriptions, to fully describe our architecture, at multiple abstraction levels.

5. Our final conclusions regarding the DL field environment, its application on the visual recognition of garment and possible future work are listed in Conclusions and Future Work.

Chapter 2

Deep Learning for Visual Recognition

"It's often the case that young fields start in a very ad-hoc manner. Later, the mature field is understood very differently than it was understood by its early practitioners."

Christopher Olah

2.1 Origins

Around 1900, Santiago Ramón y Cajal, performs the first studies on biological neurons and neural networks. Alan Turing, in 1948, after had proposed the now named Turing Machine model, proposes the Unorganized Machine model. In 1949 Donald Hebb publishes the Organization of Behavior: A Neuropsychological Theory. David Hubel and Torsten Wiesel, in early 1950s record spike sounds from neurons in the visual cortex of a cat. In 1957, Frank Rosenblatt builds the Perceptron. In 1969, Marvin Minsky and Seymour Papert, show the neural networks limitations of that time. Kunihiko Fukushima proposes, in 1980, Neocognitron ANN for visual recognition. Hopfield networks are popularized in 1982 by John Hopfield. In 1988, Yann LeCun applies gradient descent on these models optimization.

2.2 Artificial Neural Networks

In its most general definition, an Artificial Neural Network is a directional graph where each node (neuron) produces a combinative output in respect to its inputs. ANNs are inspired after the biological brain and the nets of neurons that it is composed of, hence its name, yet the current differences between the two are noticeable and that motivates the study of ANN as more abstract concept. The Google's Tensorflow refers to this concept as Computational Graphs while in [YES+14] Microsoft's team goes with Computational Networks.

Because the first architectures would contain a small number of neurons (at the order of the dozens) and the historical roots, a representation at the neuron level, were each of these units operate over real numbers and edges represent weights (parameters), is frequently used. Yet, more

Table 2.1: Comparison levels of abstraction when interpreting ANNs.

	Neuron level	**Module level**
Unit of study / graph's node	Neuron	Module
Operation domain	Reals	Tensors
Layer	Group of nodes	One node
Weights	Exist in edges	Exist in nodes
Activation	Exist in the node	A one-to-one layer
Usage	To study a particular layer	To study the network

recently, with the popularization of convolutional and other kind of layers, and the rapid growth in the number of nodes in each architecture, were often multiple equivalent groups of neurons serve a specific purpose, makes representation cumbersome and not much helpful.

Therefore, these days, it is also useful to reason of a network in terms of its **modules** (or blocks), wherein each operates over tensors (multi dimensional arrays) rather than reals. One module can represent one or more layers, or even an entire sub-network. Edges in this graph simply represent the flow of tensors. The produced tensors and correspondent format is referred to as the module API [dF16].

This concept model enables the reasoning about these networks in multiple abstraction levels, and because do no defines the implementation details, removes some existing embarrassment when fitting into the definition layers such as convolutional, were multiple neurons share weights. Besides that, it is condescending with any new layer that follows the tensor API.

With this, currently research in DL ranges from the study of better optimization algorithms and techniques to the effective and efficient combination of multiple modules in order to tackle more complex problems. Hence, ANNs should not be seen as another algorithm but rather a framework, a programming paradigm, for the development of data driven, capable to learn, programs. Some authors compare ANNs to functional [Chr15] while others associate it to the data-flow paradigm.

2.3 Learning parameters

Besides the intuitive definitions presented above, ANNs can be mathematically defined as a function that given an X and θ output Y_{pred}

$$s : \mathbf{X}, \theta \rightarrow \mathbf{Y_{pred}} \tag{2.1}$$

in which X is the set of examples under scrutiny, θ the net's learnable parameters and Y the labels or values associated to each element in X. This function, s, is commonly named the score function and consists on the values collected at the output layer. Because the output layer depends on the previous and so forth, we get

$$s = O_n(\theta_n, \quad \dots \quad O_2(\theta_2, O_1(\theta_1, X))) \tag{2.2}$$

to each $y \in \mathbf{Y_{pred}}$ it is possible to associate a loss function, L, that expresses the error between the model prediction and the correct one. The learning process consists, on finding the parameters that minimize the losses over the training dataset, i.e., minimize the cost function $\mathbf{J}(\theta)$.

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{n} L(s(x_i, \theta), y_i) \tag{2.3}$$

Because each O in 2.2 is a differentiable function, and by extension s also is, it is possible to compute, node by node, the gradient of s with respect to θ using the derivative chain rule

$$\frac{\partial J}{\partial \theta_m} = \frac{\partial J}{\partial s} \cdot \frac{\partial s}{\partial O_n} \quad \dots \quad \cdot \frac{\partial O_{m+1}}{\partial \theta_m} \tag{2.4}$$

and update θ accordingly in a process referred to as the gradient descent back-propagation. This is an important characteristic of ANNs as it allows the design of layers independently, yet because each node gradient computation depends on its successors, if one node gradient is zero, then all its predecessors are too, resulting in no parameters updates to all those nodes. This is referred to as the vanishing gradient problem. Since the updates are made using local information, it is possible that the optimization process halts at a local minimum or skips over it. Also, because J must be differentiable, and the previously described problems, often L is a surrogate function with better mathematical properties that allow a more efficient optimization. e.g., in classification, cross entropy loss is commonly used, instead of a 0-1 function.

By minimizing J over the training set we hope to minimize it also over the test set and indirectly improve some other desired performance metric e.g., precision, accuracy. In practice only small portions of the dataset are considered at each time, these are named mini-batch algorithms. [GBC16].

2.3.1 Gradient Descent algorithms

Gradient Descent (vanilla) The most basic form of ANNs optimization consists on, iteratively, feed forwarding subsets of the training data trough the network, computing the losses and gradients, and updating each parameter.

$$\Delta\theta = -\eta \nabla_\theta J(\theta) \tag{2.5}$$

$$\theta_t = \theta_{t-1} + \Delta\theta \tag{2.6}$$

The learning rate hyper-parameter, r, allows to regulate the parameter update speed and mitigate the problems described in 2.3. When the only one example is used per update then it should be referred to as **Stochastic Gradient Descent**, when the total dataset is used, **Batch Gradient Descent** else **Mini-batch Gradient Descent**. [Kar]

Momentum When in the presence of ravines, e.g, when the curvature along one dimension is much more steeply in than others, SGD causes oscillations in the traversal path. Momentum attempts to mitigate this problem by increase the dimensions that keep the trajectory constant, which is translated into faster convergence. [Rud16]

$$\begin{aligned} v_t &= \gamma v_{t-1} - \eta \nabla_\theta J(\theta) \\ \Delta\theta_i &= -v_t \end{aligned} \tag{2.7}$$

Nesterov accelerated gradient With the inertia accumulated by momentum, it is probable that the optimization process overshoots the local minima. NAG addresses this by estimating the position after the momentum application and calculating the gradient based on that. [Rud16]

$$v_t = \gamma v_{t-1} - \eta \nabla_\theta J(\theta - \gamma v_{t-1}) \tag{2.8}$$

Adagrad In opposition to the previous strategies, Adagrad (Adaptive gradient) applies a different learning rate to each parameter, penalizing parameters that are more frequently updated.

$$\Delta\theta_i = -\frac{\eta}{\sqrt{S_i + \varepsilon}} \nabla_\theta J(\theta_i) \tag{2.9}$$

where S_i is the sum of squares of the gradients w.r.t θ_i until that time and ε is a smoothing term that avoids division by zero (on the order of $1e-8$). As a consequence, because S_i keeps growing, eventually the net stops updating its parameters (learning).

Adadelta To mitigate the Adagrad continuously decreasing learning rate problem, Adadelta [Zei12] imposes a window of fixed size w. Yet, because storing all the previous w gradients is highly inefficient, this is implemented as an exponential decaying average $E[g^2]$ of all the past squared gradients.

$$\begin{aligned} g_i &= \nabla_\theta J(\theta_i) \\ E[x] &= \gamma E[x] + (1-\gamma)x \\ RMS[x] &= \sqrt{E[x] + \varepsilon} \\ \Delta\theta_i &= -\frac{\eta}{RMS[g^2]} g_i \end{aligned} \tag{2.10}$$

where RMS stands for Root Mean Square.

Also, the authors also notice that the units of the increment do not update with the units of θ, and the same happens with Adagrad, SGD and Momentum but in second order methods, such as the Newton's method this do not happen. With this, the authors rearrange the previous expression to

$$\Delta\theta_i = -\frac{RMS[\Delta\theta]}{RMS[g]} g_i \tag{2.11}$$

and as consequence, Adadelta discards the need of a configurable learning rate. Having ε in the numerator serves to start off the first iteration and ensure that $\Delta\theta$ do not converges to zero.

2.3.2 Initialization techniques

When initializing the parameters of a network two important characteristics to have into consideration are: 1. If all of parameters in a layer are equal, then all will behave and evolve equally, which is not desirable. 2. The output of the correspondent layer should not be too high or too low, resulting in a low, or even none, gradient.

2.3.3 Loss functions

As stated previously in 2.3, in DL all the learning process consists on minimizing a cost function that consists on the average of each example loss, over the training batch. For each task multiple losses exist. One loss is typically appropriated for one type of task.

Three possible approaches are: constants, for biases; random values drawn from an uniform or normal distributions, for learnable parameters.

Loss 1 (L1) and Loss 2 (L2) The simplest loss consists on the summing the absolute difference between the predicted and expected values (L1). To give greater significance to large errors over small ones, these differences can be squared (L2).

$$L_{L1} = \sum_{i=1}^{n} |y_i - s_i| \tag{2.12}$$

$$L_{L2} = \sum_{i=1}^{n} (y_i - s_i)^2 \tag{2.13}$$

where s_i and y_i are the correspondent predicted and expected values of each neuron on the output layer. This loss is commonly used in regression tasks, such as object localization.

Cross-entropy This loss is applicable on classification and assumes that the score function produces a class probability distribution. Therefore, when used, the associated output layer is most often a *softmax* activation.

$$L_{ce} = -\sum_{i} y_i \log(s_i) \tag{2.14}$$

where the s is the predicted probability distribution, and y the ground truth. Because the ground truth is commonly one-hot encoded, the above expression can be, simplified to $-log(s_t)$ where s_t is the predicted value for the true class.

2.3.4 Parameter regularization

Ahead from the basic options described above it is possible to add a regularization term to the loss function. This term is calculated in terms of the parameters values and associated to the loss function favors parameter combinations that have lower sums i.e., if the the predicted value is one, than the loss is zero ($log(1) = 0$).

$$L_{w/reg} = L + R \tag{2.15}$$

The two popular most commonly used regularization terms are L1 and L2, that consist on the sum of the absolute and square sum of the existing parameters values.

$$R_{L1} = \sum_{i=1}^{k} |\theta_k| \tag{2.16}$$

$$R_{L2} = \sum_{i=1}^{k} (\theta_k)^2 \tag{2.17}$$

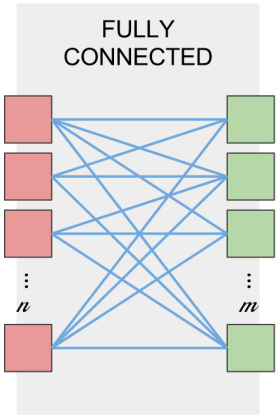

Figure 2.1: Representation of a FC layer as a modular operation. The layer input and output are represented in red and green respectively. In blue the associations between the input and output vector positions.

2.4 Layers

Layers are the smallest independently functional unit in a network. Each serves a specific purpose and may or may not be learn oriented i.e., contain parameters. One layer may be configured using its hyper-parameters.

Although each layer is explained in terms of its input and output nth dimensional tensors, in practice, because we use mini-batch optimization algorithms, layers operate over multiple tensors in a single execution, that is, one nth+1 dimensional tensors.

2.4.1 Fully Connected

A Fully Connected typically operates over vectors, 1D tensors. It produces an output vector such that each of its positions is an weighted sum of all input tensor positions. The weights are all-to-all associations between the input and output vectors positions.

$$O_j = \sum_{j=1}^{n} W_{(i,j)} \cdot I_i \tag{2.18}$$

where I_i is i-th position of the input vector, O the output and $W_{(i,j)}$ the wight associating the positions i an j of the input and output vectors respectively. The hyper-parameters of a FC layer are the input and output vector sizes. The input vector size can be inferred taking into account the output size of the previous layer.

A network that consists on one Fully Connected layer is named Perceptron. When it is entirely composed of more than one FC layers is referred to as Multi Layer Perceptron. These architectures are the most basic in the sense that do not explore the correlation between features in the input

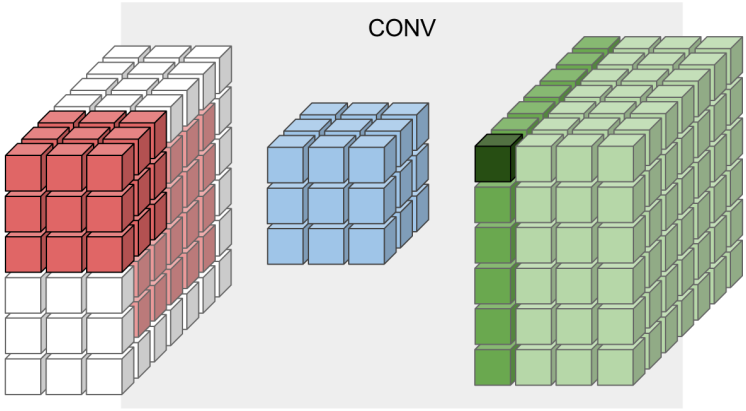

Figure 2.2: Internal representation of a convolution operation. From left to right: the input, (one) filter and output tensors, of a convolutional layer. In white it is represented a size one zero padding around the original light red input tensor. The darker red and green illustrate the covered portion of input and produced result, when applying one filter in one position. In vivid green, the first slice of the output tensor, is the result of applying one filter to the entire input volume.

and, as a consequence, these do not scale well as the number of weights is n^2. Because of that, these are often used for classification or regression tasks when the number of features is small.

2.4.2 Convolutional

The convolutional layer implements a 2D convolution operation over a 3D volume that is arranged into two spacial and one depth dimensions and typically is used with images. A 3D tensor of weights (named kernel or filter) is slided across the input spatial dimensions being, at each position, element wise multiplied with the covered portion and the result sum reduced, producing a 2D activation map. Convolving multiple filters over the input results in a volume containing multiple activation maps.

Often it is convenient to pad the input volume with zeros to e.g., to preserve it size after the CONV operation, this is achieved by padding with zeros (zero-padding). Other times it is useful to slide the filter tensor in strides rather than contiguously e.g., to consume less computational steps and memory. All these scenarios can be configured using the CONV layer hyper-parameters: the number of activation maps (and consecutively filters) s, filter spatial dimensions size f, stride s and padding p. Because the filter at each position must be inside the input volume these are self constrained by

$$(n - f + 2p) \mod s \equiv 0 \tag{2.19}$$

the operation is considered undefined when the constraint isn't valid. The output tensor is then $m \times m \times w$, where m is given by

$$m = \frac{n - f + 2p}{s} + 1 \tag{2.20}$$

Convolutional layers take advantage of the fact that the input features are highly correlated spatially and if a transformation is relevant locally then it should be useful at the whole spacial dimensions. These characteristics make it, in opposition to FC, require a small number of parameters, $f \times f \times w$.

Implementation as matrix multiplication When implementing, the convolution operation is transformed into a matrix-matrix multiplication which allows to take advantage of efficient implementations. The input volume is mapped into a two dimensions matrix $\mathbf{I}$, in an operation commonly named as **im2col**: each column contains the linear stretched portion of the input volume that is covered by one any filter at one position (because, receptive fields may overlap, input values may appear repeated on the new generated matrix). The convolutional layer filters are equally stretch into rows and grouped in a matrix $\mathbf{W}$. Therefore performing $W \times I$ results in a new matrix where each row contains the result of convolving one filter trough one spatial dimension. Reshaping is performed to obtain the three dimensional output volume. [Kar]

Fully connected as convolutional In many architectures it is common to chain FC after a set of convolutional layers. Remembering that in a FC each output position is associated to all input positions, it is possible to convert the FCs into CONVs by using a filter of the same size has the input volume. Thus, the first FC implements a filter of the same size as the last CONV activation map and the following ones with filter sizes of $1 \times 1 \times n$, where n is the number of features of the previous layer.

On other architectures, used in e.g. object detection and localization, it also often useful to apply a small MLP classification head is applied on multiple smaller portions of a CONV activation map. This can efficiently be achieved by using a CONV with the appropriated filter size an and stride.

2.4.3 Pooling

A pooling layer is used to quickly reduce the number of features existing in a, typically, three dimensional tensor by subdividing each of its depth slices into equally sized cells and pooling one value for each. Although other forms can be considered, such as average, most often it is used maximum pooling. It is also possible to consider other shapes of tensors.

Pooling layers can be configured regarding its pooling size and similarly to convolutional, can be configured regarding its stride and padding, its output tensor size is given by the same expression as in 2.20 and its hyper-parameters must respect 2.19 (replacing filter size by pool size). The stride size is commonly set equally to the pooling size, forming a pooling grid, opposed to convolutional where the stride value often 1.

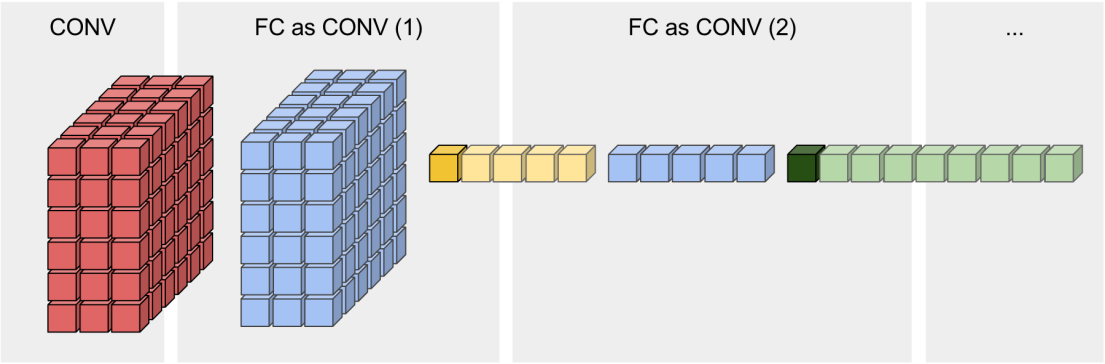

Figure 2.3: Usage of convolutional layer with the same significance as fully connected layers. Each filter has the same shape as the input tensor, and as result it is only applied once. To produce an N-dimensional tensor, N filters are required.

2.4.4 Dropout

The Dropout is a regularizing layer used to address the overfitting problem. It zeros out each feature of its input in its output tensor, with a probability p. This is performed by sampling a tensor of independent Bernoulli random variables, and multiplying it element wise with the input tensor. This vector is re-sampled periodically, e.g, for each mini-batch. At test time, dropout is disabled, behaving as an identity operation. [SHK+14]

2.4.5 Batch Normalization

While training, to avoid the vanishing of gradients, it is important to keep layer's weights stable between a certain range, depending on the associated activation function. Batch Normalization, proposed in [IS15], is placed before each activation layer and enforces that each input tensor feature (neuron activation) to be a Gaussian distribution across the batch dimension. Batch Normalization is also able to lean to parameters that scale and shift the normal curve. As result, it improves the gradient flow (by diminishing the vanishing gradient problem), allows higher learning rates, reduces the strong dependence on weight initialization and reduces the need of dropout. [Kar]

2.4.6 Activations

Sigmoid The sigmoid function squashes the input values into a range between zero and one. Sigmoids have been highly used in the past as it closely matches the firing rate of neurons. Because of its S shape, when its input values that deviate from zero, converge to zero, i.e., the gradient vanishes. Also its outputs are non zero centered, which is an undesirable property that can contribute to unnecessary zig-zagging when optimizing. In CNNs, this activation has been replaced by others, mainly ReLU.

$$O = \frac{1}{1+e^{-I}} \tag{2.21}$$

ReLU The Rectified Linear Unit it was popularized in [KSH12] where it was showed to improve convergence by a factor of 6. Compared to other activations, such as sigmoid, is less computational expensive. Yet it can also kill gradients (when $I \leq 0$).

$$O = \max(0, I) \tag{2.22}$$

Softmax It assumes that the input values are transform unnormalized log probabilities and transforms them into probabilities. It is mostly used on the last layer in conjunction with the multi class cross-entropy loss function.

$$O_i = \frac{e^{I_i}}{\sum_{j=1}^{n} e^{I_j}} \tag{2.23}$$

2.5 Development and evaluation

2.5.1 Methodology

DL as a subfield, follows the same methodology as ML. Given one specific problem and one dataset, provided with the necessary annotations, a model, in this case a ANN, is trained (optimized) and then tested. The training process consists on running an optimization algorithm that over time updates the net parameters, such that the loss function is minimized, as described previously. The optimized model is then tested and evaluated using a problem specific metric.

Dataset splits Each dataset is composed of pairs of examples (images, in the case of visual perception) and annotations (classes, in the case of classification). Because the goal is to design models that achieve accurate predictions even for previously unseen examples, the dataset under scrutiny is split into two distinct parts train and test. The test split is only used in the test phase, and the network has no access to the annotations i.e., only feed forward operations are performed. The annotations are yet used to compute the desired performance metrics. In public competitions, it is common that the test split annotations aren't provided and the performance metrics are computed by the organization given the candidates predictions.

Hyper parameter setting Ahead from the learnable parameters, each net contains a set of hyper-parameters that must be set à priori. This is performed by picking values from a range of chosen values. In [BB12], a random search layout is recommended. To be able to evaluate the multiple

combinations of hyper-parameters, the training set is re-split into (effective) training and validation. Other techniques such as k-fold cross-validation are not much used in DL. For the final training, with the chosen hyper-parameters,

2.5.2 Transfer learning

In ML, transfer learning is associated with the capability of applying a model trained on a certain problem to a different one, and this performing better results then if no pre training existed. With ConvNets this explored in mainly two ways: [Kar]

Feature Extractor To a ConvNet trained on a general purpose dataset, e.g. ImageNet, the last(s) fully connected layers are removed, and the remaining used, without any training, for feature extraction. The extracted features are used as the input of other model e.g., Softmax classifier, and applied to the new dataset and problem.

Fine-tuning The network classification or regression module is removed and replaced with a new, adapted for the classes and problem in question and the network is re optimized. Depending on how similar the datasets domain are, multiple net early layers can be kept fixed, as early features are more abstract, and thus should be shared among the two.

2.5.3 Data analysis and preparation

Data visualization t-Distributed Stochastic Neighbor Embedding was proposed in [vdMH08], and transforms each n-dimensional data point into a two or three-dimensional space, by an optimization process. t-SNE improves on a previous technique, SNE, and with the introduced changes it is able to better handle high dimensional spaces. This makes it a popular technique for represent images datasets.

Data augmentation Because ANNs require large datasets, at least when training from scratch, it is common to use augmentation techniques to enlarge the dataset size. The most used transformations include horizontal flips, translations, crops or scales and color jitter.

Mean subtraction The most common form of preprocessing used in DL for image inputs, is the mean subtraction of every input feature. This operation as the effect of center the data cloud on the origin along every dimension. For convenience, in practice, the mean is calculated and subtracted w.r.t. each color, instead of pixel. [Kar]

2.5.4 Performance metrics

Although in DL the learning process is consists on minimizing a loss function, performance metrics are important to evaluate and compare solutions. Depending on the problem, different metrics are used.

Top-5 and top-1 error As the names suggest, these measure the percentage of errors over the total of predictions, wherein the model makes N predictions for each example, and it is considered error if this set does not contain the correct one.

For the multi-label classification problem, the error of each test example is given as,

$$e = \frac{1}{n}\sum_{i=1}^{n} \min_{j} d(c_{j,k}, C_i) \tag{2.24}$$

$$d(x,y) = \begin{cases} 0, & \text{x = y} \\ 1, & \text{otherwise} \end{cases} \tag{2.25}$$

where C_i is the i-th ground-truth label of the image, and c_j the j-th prediction for the same image. d represents the distance/error between the prediction and expected value.

For the classification+localization problem it is often considered that it is a truth positive if the predicted class it is correct and the predicted bounding-box has an interception over the union (IoU) with the ground truth greater than 50%. Therefore the error expression becomes

$$e = \frac{1}{n}\sum_{j=1}^{n} \min_{k} \min_{m} \max[d(c_{i,k}, C_{i,j}), f(b_{i,k}, B_{i,j,m})] \tag{2.26}$$

$$f(x,y) = \begin{cases} 0, & \text{IoU(x,y) > 0.5} \\ 1, & \text{otherwise} \end{cases} \tag{2.27}$$

The final error is the average over the test dataset.

Mean Average Precision When considering more general problems, e.g., object detection, where the number of entities to find is unknown, and the proportion of positive to negative examples is accentuated, metrics such as error or accuracy are not appropriate. As an example, considering accuracy, an image containing one object, a system A that covers all the image with positive bounding-boxes predictions and a system B that predicts one single bounding-box in the wrong position; A would be the best rated.

The AP metric mitigates theses problems by combining two complementary metrics: precision and recall. More precisely, at each level of the predictions rank, the precision and recall pair is calculated, forming together the precision-recall curve. AP is the area under this curve. In practice the curve is approximated by interpolation and the integral by a discrete sum:

$$AP(c) = \frac{1}{11} \sum_{\{r \in 0, 0.1, \ldots, 1\}} p_{interp}(c, r) \tag{2.28}$$

$$p_{interp}(r) = \max_{\tilde{r}:\tilde{r}>=r} p(c, \tilde{r}) \tag{2.29}$$

The approximations described in 2.28 are the ones followed at the PASCAL VOC challenge [EGW+10]. Averaging over all the classes AP gives the **mean Average Precision** (mAP). When areas are involved, it is important to underline the positive/negative criteria, commonly being defined as: correct class and IoU greater that 50%.

2.6 Summary

As general, and reasonable, guidelines the following rules can be followed.

1. Convolutional layers are applied for feature extraction and fully connected for classification and/or regression. Therefore, CNNs consist of convolutional followed by FC layers.

2. ReLU activations are used in hidden layers. In the last layer, softmax together with cross-entropy is used to perform classification; for regression, activation and L1 or L2 for can be used. Weight regularization, L1 or L2, can are used to mitigate over-fitting.

3. BatchNormalization is used before non linearities to keep the previous layers activations between desirable ranges during training.

4. An adaptive learning rate algorithm, such as Adadelta can be used to avoid the necessity of adjusting it during the optimization process.

5. Use a dataset split for train and another for hyper-parameters validation. Data augmentation can also be used to mitigate over-fitting problems. Finally apply your trained model on a test split for benchmark.

Chapter 3

Related Work

3.1 Image classification

3.1.1 Plain architectures

AlexNet In 2012, AlexNet [KSH12] achieved 15.3% and 16.4% top-5 error rates on the ILSVRC2012 image classification competition, which translates to a difference of 11%, when comparing against the next best solution (The best of the two classifications was achieved by training the net using data from the ILSVRC2011). This rapid percentage decrease was one of the motivations of the recent wave of interest on DL. AlexNet is a ConvNet composed of five convolutional and three fully-connected layers. It used and popularized the usage of ReLU activation functions; max-pooling to progressively reduce the amount of features and computation; dropout layers to reduce overfitting; and the usage of GPUs.

Very Deep Convolutional Networks In 2014, [SZ14] explored the effect of growing ConvNets depth wise, varying the number of layers between 11 and 19 maintaining similar architectures, wherein all the convolutional layers are configured with 3×3 filters and are interspersed with max pooling. With that, the authors show significant improvement of increasing the number of layers to 16 – 19, reporting 24.8% top-1 and 7.5% error rates, for the classification problem using the ILSVRC-2014 dataset, for the 16 and 19 layer architectures (commonly referred to as VGG-16 and VGG-19) versus 28.2% and 9.6% for the 13 layer one.

3.1.2 Complex architectures

GoogleNet Increasing the number of layers and filters at each layer is a safe way of improving a network performance however, with that, so does the number of net's parameters, and consequently the memory and processing power required, which is not ideal. Notice that, the depth of the network is related to the size of each convolutional layer kernel, as larger kernels more quickly reduce the number of features available. In [SLJ+14], the authors explore the idea of extracting, in parallel, different sized kernels that are further concatenated into a single tensor. Because

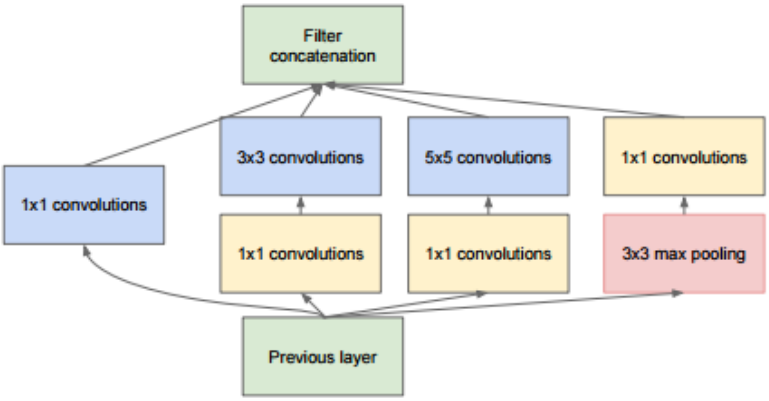

Figure 3.1: Inception module, retrieved from [SLJ+14].

this naïve solution would still result in a large number of weights the authors refine the idea with 1×1. GoogLeNet is a 22 layers network built out of inception blocks, which was used in the ILSVRC2014 competition, achieving the best top-5 classification error rate 6.67% on the image classification competition.

ResNet In [HZRS15], the authors explore the fact that when networks with a large number of layers are starting to converge, a *degradation* of the training accuracy happens and the optimization reaches a plateau at a higher error than shallower networks. To solve this, the authors introduce the residual block that, adds a identity connection that skips over one or more connections. With the skip connection, shorter paths are created between the output and the first layers. The first three positions of the ILSVRC2015 competition are ensembles of 152-layer ResNets with the best achieving 3.5% top-5 error rate. Although the humongous number of layers, this architectures have lower complexity (less parameters and activations), requiring less memory and processing power, than VGG nets, which is achieved by applying a larger filter in the first layers (7×7 vs 3×3) and more quickly reducing the input size and less filters per layer, when comparing layers at the roughly same input size.

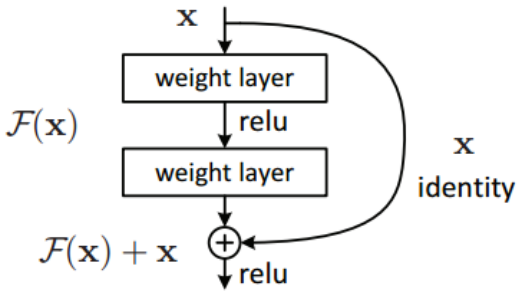

Figure 3.2: Residual block. Retrieved from [HZRS15].

3.2 Object detection

3.2.1 Region-based Convolutional Networks

R-CNN In [GDDM13], region proposal techniques are combined with CNNs. The result solution named R-CNN, is a pipeline with four steps: 1. A class agnostic method, e.g. selective search was used by the authors, to propose regions of interest. 2. Each proposed region is fed into a FCNN, e.g. VGG-16 without the fully connected layers, that outputs a feature vector. 3. Multiple SVM score each feature vector regarding the likelihood of belonging to that class, in a one-vs-all strategy. 4. A class specif linear model is used to fine tune the bounding-box location and size. 5. An greedy non-maximum suppression is applied to reject regions that have an IoU larger than a certain threshold e.g., the authors select 0.5 after testing multiple values over the validation dataset. With this technique the authors achieve 53.7% mAP on PASCAL VOC 2010 and 31.4% on ILSVRC2013 dataset an improvement of 7% when comparing against the until then best result 24.3%, achieved by OverFeat. In fact, the ILSVRC2014 winner, submitted by the GoogLeNet team, achieves 42% mAP with this technique, updating the FCNN module to an Inception model and, combining selective search with multi-box predictions to achieve higher proposals recall. No bounding-box regressor is used.

Fast R-CNN Because R-CNN is a multi-stage pipeline, it contains various inefficiencies that result in a time and space consuming solution, mainly due to the fact that for each proposal, a feed forward pass over the FCNN is performed and, features extracted from each object proposals need to be stored into disk. In [Gir15], the authors introduce multiple innovations, in particular the RoI pooling layer, that result in Fast R-CNN being $9\times$ quicker to train and $213\times$ faster then R-CNN at test-time. The **RoI pooling** layer, is similar to the spatial pooling, in the sense that pools the maximum element from each cell of a hyper parameterizable $H \times W$ grid, yet this grid exists hover the suggested region, rather than the whole input spatial area. The inputs of this layer are activation map output and a set of bounding boxes to pool. The output a $N \times H \times W$ tensor, where N is the number of proposals. Ahead from RoI pooling the authors also replace the SVM classifier with a softmax classifier, resulting in a two headed CNN trained with a multi-task loss.

Faster R-CNN Although Fast R-CNN achieves great processing efficiency improvements, it is still dependent on region proposal methods that are responsible for the majority of processing time required. Selective search, for example, takes 2 seconds to process an image, an order of magnitude slower than a detection network. Re-implement such methods to run on GPUs, would be a solution yet, with that we would be missing the opportunity to share computation with the FCNN. In [RHGS15], the authors introduce **Region Proposal Networks** (RPNs), a module that extracts class agnostic bounding boxes from a convolutional layer activation map. A RPN is a small three layer sliding fully connected network, implemented with convolutional layers, that at each input's spatial position, extracts $s \times r$ bounding boxes, where s, scale, and r, aspect ratio, are hyper-parameters that are associated with the bounding-box size. The authors refer to this pair

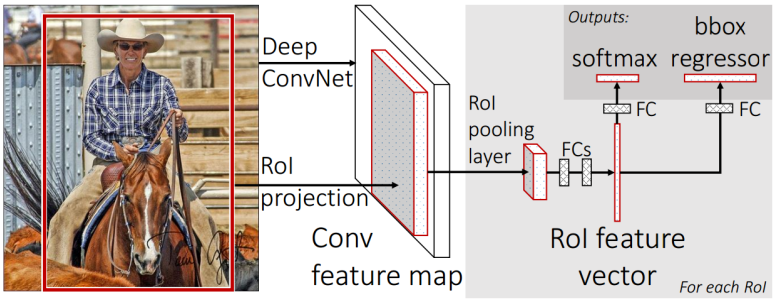

Figure 3.3: Fast R-CNN model. To notice, the RoI pooling layer that extracts a proposed region from a convolutional activation map and reshapes it to fit the following layers. Retrieved from [Gir15].

position-scale-ratio triplet as anchor points and use three values for each: 128^2, 245^2 and 512^2 for scale and $1:1$, $1:2$ and $2:1$ for aspect ratio; a total of 9 anchor boxes per sliding position.

3.2.2 You only look once

YOLO Although many efficiency improvements are achieved with the successive iterations on the R-CNN architecture, one still persists: the classification in two steps. In [RDGF15] overpasses this by simply placing two heads: one for regression and another classification, on the top of a regular fully CNN. The input image is divided into an $S \times S$ grid and, for each cell, the regression head outputs n bounding-boxes, referent to objects that have its center in that cell, and a confidence value. If a cell should predict no bounding-boxes, than the confidence value should be zero. The classification value head predicts a classification conditional probability grid map, that are conditioned to the cell containing an object. YOLO is able to operate at 45 frames per second and achieves 63.4% mAP on PASCAL VOC 2007.

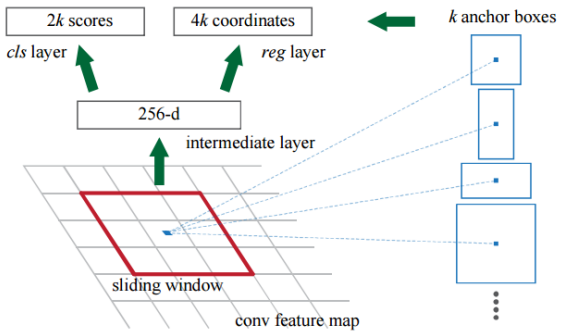

Figure 3.4: Faster R-CNN model. Retrieved from [LLQ+16].

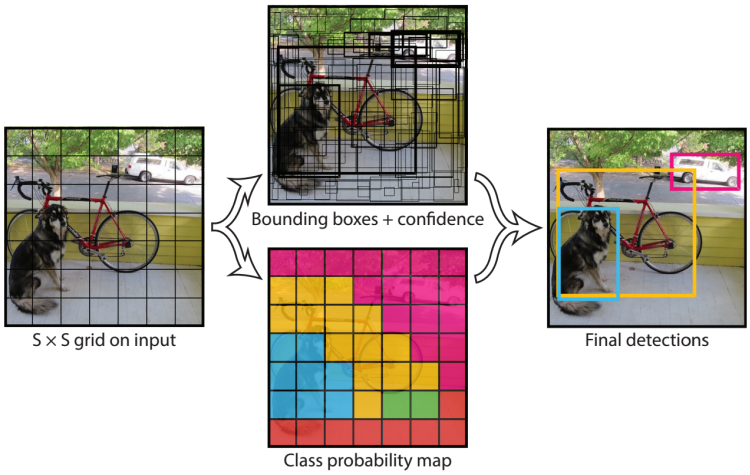

Figure 3.5: YOLO model. At the left, the grid placed on top of the input image. On top, example of the predictions performed the regression head. In the bottom, the prediction grid map predicted by the classification head. Retrieved from [RDGF15].

YOLO9000 In [RF16], the second version of the YOLO model, the authors introduce various improvements, proposed by the authors after an error analysis of YOLO against Fast R-CNN. By experimenting wiht multi sized input images, multiple versions of the model are produced, achieving 76.8 mAP at 67FPS, with 544×544 images, and 78.6% at 40 FPS with 416×416, on the VOC 2007 dataset.

3.3 Semantic segmentation

Deconvolution Network In semantic segmentation, the goal is to classify each pixel present in the input image. It can therefore be understood the necessity of having the last layer outputting a tensor with the same spatial dimensions as the input image. Yet when examining convolutional architectures it can be concluded that as the depth of the network increases, the output tensor spatial size decreases, unless padding is used. If Max pooling is used, then padding isn't an option. In [NHH15], the authors introduce a solution to solve this by applying Deconvolution (or Transposed convolution) and Max Unpooling, inverse operations of the convolutional and Max pooling operations, to construct a Deconvolution network (DNN). Combed a CNN followed by a DNN the authors are able to achieve 72.5% accuracy on the PASCAL VOC 2012 dataset.

3.4 Garment recognition

Early works The first attempts to handle clothes happen in Robotics with the folding task, and the problem domain is constrained enough to avoid the necessity of classification i.e., only one type of clothing is considered. In, [MSCTLA10] towels are considered and depth discontinuities

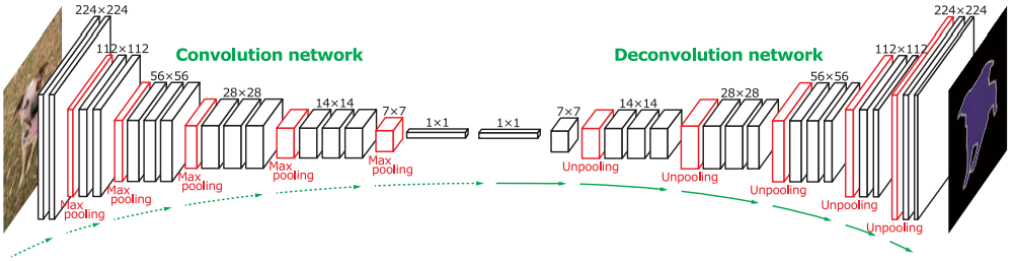

Figure 3.6: Deconvolution network model. Retrieved from [NHH15].

are explored to detect its border line and corners. With that information, a PR2 robot is able pick them from a random dropped position and, following a predefined sequence of steps, fold them.

Classification with ML Clothing classification is latter addressed using ML approaches, not only in Robotic tasks but other software applications as well e.g., in [YY11] real-time classification and segmentation of people's clothing appearing in a camera video feed are addressed. Yet, using the raw image i.e., each pixel is a feature, would result in a low performance results due to the curse of dimensionality that many ML suffer from. Therefore, one common approach is the Bag of Visual Words (BoW) model wherein each image is represented by an histogram of words and a word is a cluster of features extracted and described using one of many existing techniques, e.g. SIFT or HoG. This histogram of words is then the input of a classifier algorithm that outputs the correspondent class. In [LCA14] the authors apply this technique in a two a layer hierarchical classifier to determine the category and grasping point (in this work, equivalent to pose) of a piece of clothing. The input of this pipeline are rendered and real depth images of hung sweater, jeans and shorts. At the first layer an SVM with a Radial Basis Kernel function is used to determine the garment class, and in the second a Linear Kernel function was used to determine the grasping point index. Other works focus on the study of better, domain specific forms of describe a piece an image in order to enhance certain cloth characteristics that can later improve the overall classification performance. In [Yam16], a set of Gabor filters are used to filter edges magnitude and orientation that are representative of wrinkling and overlaps and with that information, the authors propose three types of features: Position and orientation distribution density, cloth fabric and wrinkle density; and existence of cloth-overlaps.

Classification with DL With the recent wave of success with CNNs, some works that address on garment recognition also explore their potential. One case of that is [MPKM15], that addresses the same problem as [LCA14]: pose and category recognition. The solution is also similar, a two layer hierarchical classifier. Here, instead of BoW and SVMs, one CNN is used to determine the garment class, followed by a class specific CNN that determines the garment pose index. The CNNs used are composed of three blocks of convolutional and Pooling, followed by three fully connected layers, that differ in size depending on the associated classification task. The authors compare the model against other and report gains of at least 3% in accuracy.

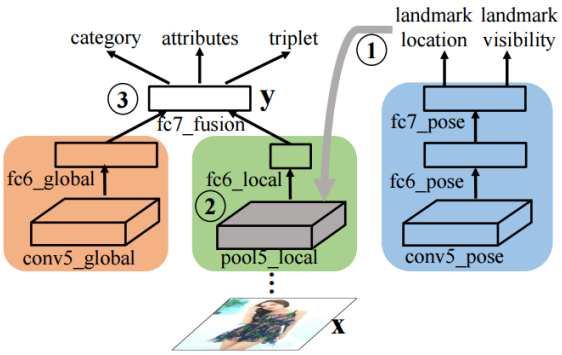

Figure 3.7: FashionNet branches illustration. Retrieved from [LLQ+16].

FashionNet To demonstrate the advantages of the large dataset DeepFashion, described in the next chapter 4, FashionNet is a CNN that simultaneously predicts the garment category, attributes, landmarks and pairs. This net implements a similar strategy to Faster R-CNN by after a set of convolutional layers, a branch extracts binary classified regions (in these landmarks, classified regarding its visibility) that are then used to pool the associated set of feature from the feature extractor activation map. FashionNet also includes a third parallel branch that processes global features of the image. Finally the attributes the outputs of the local and global features are concatenated, yielding the garment category, attributes and model clothes pairs present in a batch. In this architecture, the Landmark pooling layer has a different role then the RoI pooling introduced in [Gir15], as there isn't the necessity of classify each landmark. All the pooled features are, because of that, concatenated and used to enhance the information available in the final fusion layer.

Chapter 4

Available resources

4.1 Garment datasets

4.1.1 CloPeMa

The Clothes Perception and Manipulation, was an open-source EU-FP7 research project conducted, between 2012 and 2015, by a consortium of five european institutions: Centre For Research and Technology Hellas (CERTH), Greece; Neovision SRO and Ceske Vysoke Uceni Technicke v Praze (CTU), Czech Republic; University of Glasgow, United Kingdom; and Universita Degli Studi Di Genova, Italy. With a self made robot equipped with stereo vision and two arms, this project aimed to advance the state of art in garments and textile perception and manipulation.

From this project resulted twenty-eight publications and multiple datasets made available by the institutions: CERTH, CTU and Glasgow. Currently only CERTH and CTU datasets are publicly accessible.

1. **CTU Color and Depth Image Dataset of Spread Garments**, is dived into two groups: "Flat and wrinkled", with 2050, and "Folded", with 1280 examples. Each example contains one image of a piece of clothing placed on a wooden floor and is annotated with the stereo disparity; garment category; interest points coordinates and category; and other meta-information.

Figure 4.1: Image examples retrieved from the CloPeMa project resulting datasets. From left to write: CTU "Folded"; CTU "Flat and wrinkled"; and Glasgow.

Figure 4.2: Adapted from [dee]. At top row, examples from the category and attribute prediction dataset; at the bottom, fashion landmark detection.

2. **Glasgow**, contains examples of 16 pieces of garment in 5 different poses each, ranging from flat to completely crumpled. Each example encompasses: two views (left and right cameras), stereo disparity information and one binary segmentation for each view, as well camera calibration information.

4.1.2 DeepFashion

The Large-scale Fashion (DeepFashion) Database was launched in 2016, by the Multimedia Laboratory of the Chinese University of Hong Kong. It provides a large-scale garment dataset containing over 800,000 images annotated with category, multiple attributes and landmarks. The dataset images were retrieved from Google search engine and the online e-commerce platforms *Forever21* and *Mogujie*, then were curated by automatic processes and human annotators. DeepFashion provides one datasets, with the respective train, validation and test partitions for benchmarking for different kinds of problems, of which:

1. **Category and Attribute Prediction** The dataset contains 289,222 images distributed along 50 types of garment, that grouped into 3 categories (upper, lower or full-body) each annotated with one category and bounding box; and the applicability of each one of the 1000 possible attributes.

2. **Fashion Landmark Detection** Contains 123,016 examples, each annotated with the garment category and bounding box and 8 landmarks, that depend on the clothing category, its visibility and location.

4.1.3 Cloth Co-Parsing

In the context of [YLL15], where the authors address the co-segmentation problem, a compilation of 2,098 high-resolution photos was produced. Each containing a single fashion model posing

on a outdoors scenario. From the total amount of images, 1000 contain super pixel-level annotations distributed through 58 categories,(e.g., bag, coat, hair, shoes, skin, t-shirt or sunglasses) and background. The remaining are annotated image-level tags.

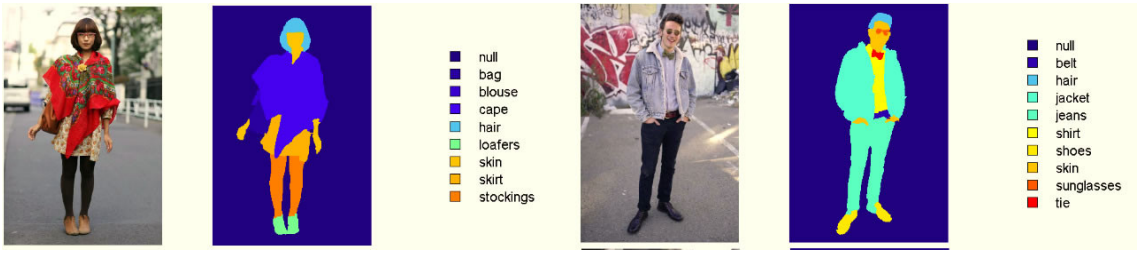

Figure 4.3: Adapted from [ccp].

4.2 Python libraries

Python is a widely known programing language and one of its advantages is the large number of available libraries for the development of scientific works. With that in mind we limited our range of options to the Python ecosystem.

Theano Released in 2007 by the LISA group, Theano allows the definition, optimization and evaluation of mathematical expressions, in particular ones with multi-dimensional arrays. It offers efficient symbolic differentiation capabilities, that are required for the implementation of gradient descent optimization. Theano also offers a transparent to the developer support for GPU processing.

TensorFlow Similarly to Theano, TensorFlow is a library developed by the Google Brain Team designed for the calculation of mathematical operations over multi dimensional arrays i.e., tensors, where an operation is considered a node of a (computational) graph. Beyond the similarities with Theano, TensorFlow offers better support for running its models in production; TensorBoard, a graphical interface of the visualization of the ANN graph, loss curves and dataset visualization with t-SNE, etc. Tensorflow also includes ML models, such as SVMs or K-means.

Keras With the goal of helping reduce the amount of boilerplate code, required to implement an ANN, Keras is an high-level library that runs on top of TensorFlow or Theano back-ends. [C+15]. It offers support for both CNNs and RNNs. Keras provides out of the box many configurable layers as well possibility of the development of new ones. Recently, Keras was included in the Tensorflow library packages.

Caffe Developed by the Berkeley AI Research and community contributers, Caffe is a complete DL library. In Caffe, using the Protocol Buffers serialization format developed by Google, it is

possible to define an ANN architecture and solver avoiding the necessity of writing code. It also offers Model Zoo, an archive of models implemented with Caffe, including e.g., Faster R-CNN.

Caffe2 After the success of Caffe, Facebook creates Caffe2, offers support for large-scale training and new hardware support, with particular focus on mobile devices. Caffe2 also offers support for *protobuf* files and a Model Zoo.

PyTorch In contrast with the remaining libraries, in PyTorch the computational graph is dynamic i.e., can change over time. Although it is GPU capable and efficiently implemented, because it written in Python it might not be the best library to use in production.

Chapter 5

Localizing garment and detecting landmarks

5.1 Approach description

Our network, that we fondly name **GarmNet**, is inspired on Faster R-CNN [RHGS15], that we simplify after noticing its inefficiencies and key differences between object and garment landmark detection. In the end, our approach most resembles YOLO [RDGF15] then Faster R-CNN [RHGS15], in the sense that classification is performed in a single step; but similarities with other architectures may also be found. With FashionNet [LLQ+16], in the way that the landmark detector output features are combined with the garment localizer branch; or other models used in semantic segmentation e.g., [NHH15], and the way that information is encoded in the last layer output tensor.

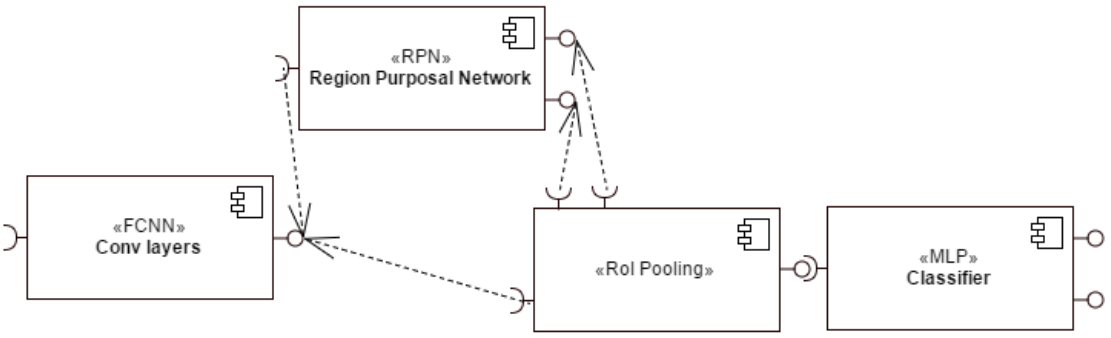

Figure 5.1: UML representation of the Faster R-CNN model.

At a macro level, Faster-RCNN can be decomposed into four modules: Conv layers (Feature extractor), Region Proposal Network, RoI Pooling and Classifier (and localizer); being that the latter performs the classification and bounding-box fine tunning of the proposals emitted by the

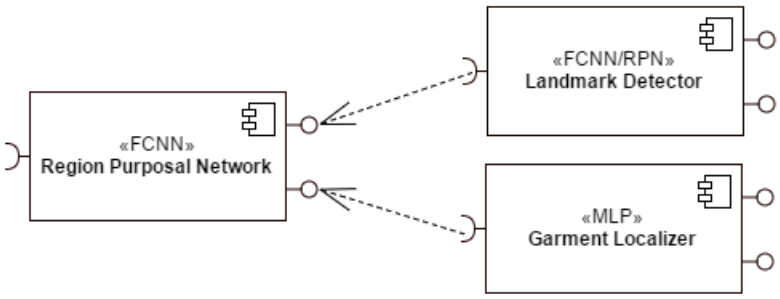

Figure 5.2: GarmNet macro view (first version)

RPN, see 5.2 [1]. This is, in great part, possible due to the ROI pooling layer, that extracts and resizes variable sized regions from the Feature extractor activation map to a fixed size tensor. This is a necessary characteristic as convolutional layers, used on the Object detector module, operate over batches of images that must constitute a constant sized, along all dimensions, tensor. On the other hand, because RoI pooling operates image by image and proposals sometimes regions overlap, this is a less efficient operation. We dispense the two step classification process in favor of a direct multi class classification, preforming the necessary changes to the RPN classifier head and dropping the Object Detector and RoI pooling layers, achieving effective training and test speed gains. Ahead from that, because landmarks are a point, we also simplify the multi size and ratio anchor-boxes to a single one per activation map position.

Beyond that, we add a new branch to the network, after the feature extractor, that ends in two heads: classifier and regressor; and performs the garment piece localization. We latter, experiment with using the output of the landmark detection together with feature extractor activations in the input of this branch. Our main ANN architecture can, therefore, be summarized into three modules: **Feature extractor**, **Landmark detector** and **Garment localizer**.

Feature extractor We implement the feature extraction module with a 50 layer ResNet, pre trained on ImageNet, to which we remove the last FC layers. Because this is a deep network the resulting output tensor has a spacial dimension of 7×7. Since in some cases we have multiple landmarks close to each other, we preferred a larger output size, that would result in a higher number of anchors in the landmark detector. We achieve this by probing the ResNet at the end of the *conv4_x* block, which has an output size of 14×14. We use the larger endpoint to connect the landmark detector, and the smaller and deeper to connect the garment localizer.

[1] In the current literature there's no standard for the graphical representation of ANNs architecture, being that each author creates its own diagrams with different graphical languages, making it cumbersome to understand each. We'll use UML components diagram combined with a textual description of each interface's tensor shape and meaning, each component hyper-parameters and values. Besides offering a standard graphical representation, UML components diagram enables an hierarchical representation and reasoning, compatible with the definitions presented in 2.2. It is important to note though, that arrows in this diagram do not represent the flow of tensors but the functional dependencies between components, which in practice makes them appear in the opposite direction relative to the flow of tensors.

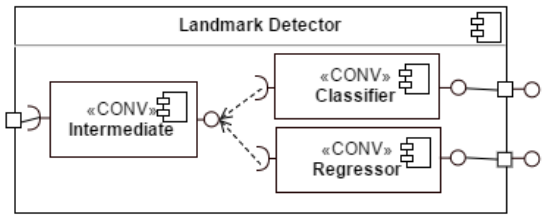

Figure 5.3: Landmark detector internal view

Landmark detector Responsible for classifying and localizing all the landmarks present in the image, this module is a small, 3×3, sliding fully connected network, similar to the RPN introduced in [RHGS15], and it is implemented with convolutional layers. The network is composed by an intermediate 256-d layer, with a ReLU activation, followed by two heads: one for localization and other for classification.

The localizer head, a convolutional layer with two filters, outputs the relative coordinates of the predicted landmark such that

$$A_{(i,j)} = (s * j, s * i) \tag{5.1}$$

$$L_{(i,j)} = A_{(i,j)} + O_{(i,j)} \tag{5.2}$$

where L is a predicted landmark location, A stands for the sliding network position O the localization head output, and s a stride value that we define to spread the base referential (or anchor points, we detail it in 5.2.2) evenly across the input image (we set it as 18, that together with a 26 landmark area matches the 224 image input size).

The classifier head, a convolutional layer with $n+1$ filters, where n is the number of landmark classes and the plus one answers for background, non positive, landmarks. We apply a *softmax* activation along the depth dimension and, therefore the output of this layer can be interpreted, at each position, as the probability of the associated landmark $L_{(i,j)}$ being of a certain class, or background.

Garment localizer To perform the localization of the piece of clothing present in the image, we use a two head three layer fully connected network, similar to the sliding window used for landmark detection. The intermediate layer is a 512-d FC with ReLU activation, followed by the regression and classification heads. The regression head outputs four values: x, y, with and height; and the classification head outputs n values, where n is the number of garment classes, that we remap to probabilities with a *softmax* activation. We retrieve the predicted landmarks by computing the *argmax* over the classifier output tensor depth dimension, and associating it with the point predicted in same spacial position on the regression head. We further discard all the landmarks that have a confidence value i.e., the value that motivated the argument to be the maximum, lower

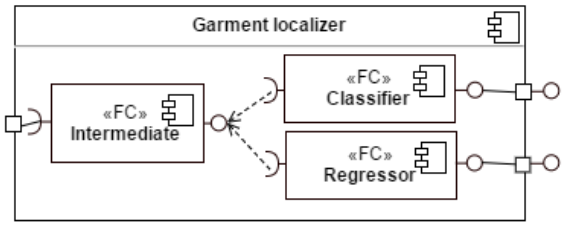

Figure 5.4: Garment localizer internal view (first version)

that 0.5. We initialize kernels with random values drawn from a normal distribution, and bias with ones.

5.2 Train

5.2.1 Dataset preparation

The first step of the development phase consisted on the selection and preparation of the dataset. We choose the CTU Color and Depth Image Dataset of Spread Garments 4.1.1. Since this dataset only contains information regarding each landmark position, we extend its annotation with the garment bounding box as follows:

$$P_1 = (\min_{\forall l \in L} l_i, \min_{\forall l \in L} l_j) \tag{5.3}$$

$$P_2 = (\max_{\forall l \in L} l_i, \max_{\forall l \in L} l_j) \tag{5.4}$$

where L is the set of Landmarks, P_1 and P_2 are the top-left and bottom-right corners of the bounding box. There's a total of 27 landmark categories, distributed among 9 types of garment. As mentioned in 1.2, problem description, some landmark categories are shared among classes, See B for in detail description of the landmarks and garment categories distributions. We then create two splits, with 300 randomly chosen images for validation and the remaining the ones used for training. The remaining 2318, make the training split.

5.2.2 Landmark detection anchors

For training the landmark detector heads we transform the landmark locations into small squared areas and follow a strategy similar to the anchor boxes described in [RHGS15]. To all the anchor boxes that intercept a landmark box with IoU > 0.7, we consider it a positive for the respective class. If it does not intercept any with IoU >= 0.3 we set it to background. Because the ratio between positive and negative anchor boxes is high, we create a binary mask that is used to filter the anchors that effectively are considered in the loss function. This mask selects all the class positive and 10 randomly chosen background anchors. Although we have used the area based criteria, after [RHGS15], using distance metrics might be a better and more elegant solution.

5.2.3 Loss functions

Landmark detector For the localization head, we apply the robust loss defined in [RHGS15] to all the active anchors that are selectable by the mask, m, described in 5.2.2.

$$L_{reg} = m \odot L_{robust}(P,T) \tag{5.5}$$

$$L_{robust}(P,T) = \begin{cases} (P-T)^2, & P-T < 0.5 \\ |P-T|, & \text{otherwise} \end{cases} \tag{5.6}$$

For the classification head, we apply the cross entropy loss function to all active anchors, being that at each anchor the landmark class is one-hot encoded.

$$L_{cls} = m \odot L_{ce}(T,P) \tag{5.7}$$

Garment localizer With the garment classes represented in a one hot encoded vector, we use the cross entropy loss on the classification head and, for the regression head, we use the mean squared error.

5.3 Experiments

Our implementation was performed using Keras library with the TensorFlow back-end. All experiments were carried out on a laptop, with GPU support enabled, equiped with an Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz, 16GB DDR4 RAM, and a Nvidia Geforce GTX 1060 6GB GDDR5 GPU. We make our source code available at https://github.com/danfergo/garment.

We perform all trainings using Adadelta with 1.0 learning, a batch size of 30, and 40 epochs per experiment. At test time our model runs at roughly 30 FPS.

5.3.1 One landmark per class, constraint

One important peculiarity of landmark detection to consider is the fact that, per image, only one class of landmarks exist. Therefore, we can introduce this constraint into the loss function and promote parameter combinations that tend to predict only one landmark per class. We implement this constraint by also applying cross-entropy over the spacial dimension, resulting in 5.8. However, because cross-entropy expects its input to be a probability distribution, we must firstly apply *softmax* accordingly. We therefore, place two *softmax* after the last convolutional layer activation: the first, (regular) depth wise, the second, spacial wise; and pass each output to to the correspondent cross-entropy. At test time, we average the two *softmax*, similarly to 5.8. Because for each garment category, only a few landmarks are active, we further create a second mask, that is the ground-truth max spacial value, and we use it to ignore the loss spacial component for the landmark classes that are not applicable.

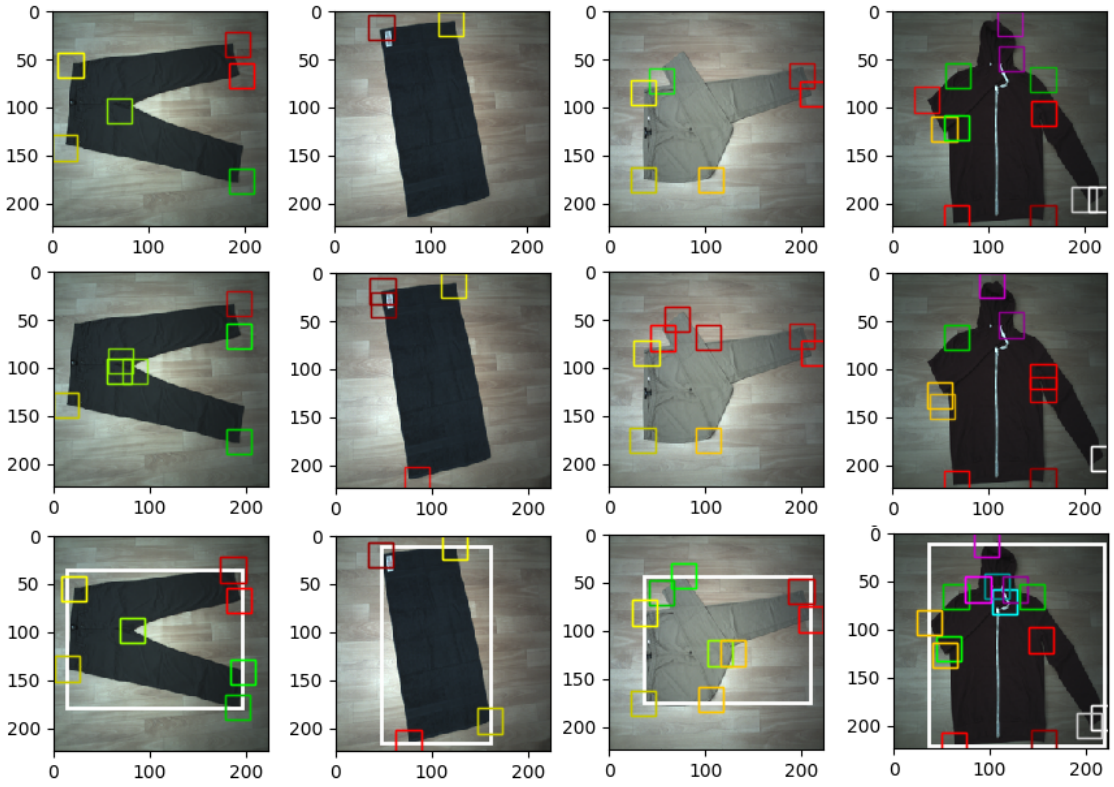

Figure 5.5: Representative cases of the result of applying the spacial constraint loss. At the top row, predictions with composed loss, at the middle, without, and the bottom, the ground truth.

$$L_{cls} = \frac{1}{2}\left[m \odot L_{ce}(T,P) + \max_{w \times h} T \odot L_{ce(spatially)}(T,P)\right] \tag{5.8}$$

With the spacial constraint loss addition we are able to achieve lower duplicated predictions, illustrated in 5.5. Although we could simply threshold duplicated predictions at test time, by integrating this constraint into the loss function, we reduce the search space.

5.3.2 Considering landmarks for garment localization

We investigate the possible gains in feeding the landmark detector classification head output features into the landmark localizer intermediate layer, with the expectancy that these help better frame the garment bounding box and, the gradients that are back-propagated trough this connection influence the landmark branch to do not predict invalid landmarks classes for a given garment.

We load the garment detector branch learned parameters from the previous trainings (we do not load the garment localizer, as with the bridge connection, and the changes to its final intermediate FC layer, this is not possible). The best combination of options achieves 68.0% classification+localization a 24.7% improvement, when comparing with the individual garment detector

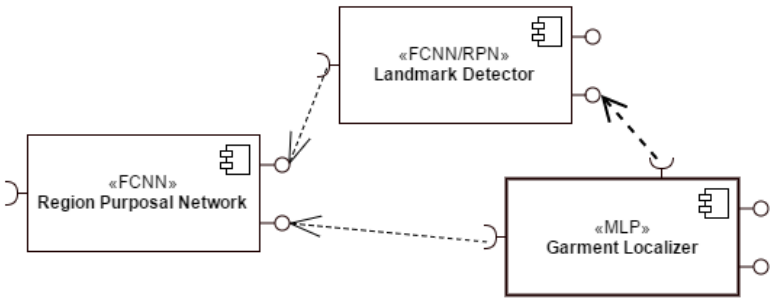

Figure 5.6: GarmNet macro view final version (after 5.3.2)

training. The overall decrease in landmark detection mAP, might be justified with the slight overfitting noticeable when analyzing the loss functions, in C.

5.3.3 Final optimization

We perform one last training, without loading any previous learned parameters (with the exception of the feature extractor ImageNet parameters) and using augmented data. The data augmentation is achieved by repeating examples of less numerous classes and adding Gaussian and hue noise, see B. The obtained results are: 100% classification, 82.0% classification+localization and 36.2% mAP.

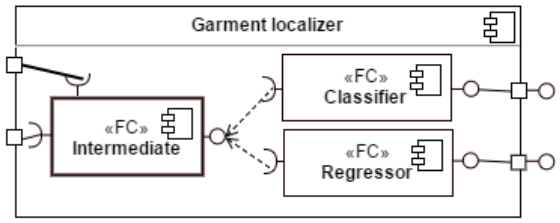

Figure 5.7: GarmNet garment localizer (after 5.3.2)

Table 5.1: Summary of classification and classification+localization results.

	Separated training	Combined w/o bridge and w/ spacial constraint	Combined w/ bridge and spacial constraint	Combined w/ bridge and w/o spacial constraint	Combined w/bridge, w/o spacial constraint and augmented data
Classification	100	93.3	100	100	100
Class.+loca.	43.3	54.0	10	68.0	82.0

Table 5.2: Summary of landmark detection results. The landmark mAP values are measured against the ground truth anchor boxes.

	Separated training w/o spacial constraint	Separated training w/ spacial constraint	Combined w/o bridge and w/ spacial constraint	Combined w/ bridge and spacial constraint	Combined w/ bridge and w/o spacial constraint	Combined w/bridge, w/o spacial constraint and augmented data
mAP	37.8	35.7	37.5	36.8	34.5	36.1
Left leg outer	38.5	33.7	44.0	41.6	33.0	38.2
Left leg inner	47.6	39.5	44.6	47.6	0	44.1
Crotch	46.4	42.5	47.2	42.2	43.7	34.3
Right leg inner	46.0	42.6	41.4	45.8	50.5	39.9
Left leg inner	45.3	41.4	49.1	45.5	43.7	40.9
Top right	62.1	54.7	58.8	58.0 -	50.5	55.7
Top left	57.3	58.4	56.3	51.3	47.8	50
Right sleave inner	44.3	38.7	40.8	35.7	60.2	37.3
Right sleave outer	38.2	30.4	48.3	39.4	53.6	42.2
Left sleave inner	43.9	41.5	41.0	40.7	46.5	48.3
Left selave outer	34.5	30.7	31.3	32.4	46.0	35.7
Hood right	0	54.5	0	0.51	46.4	0
Hood top	0	0	53.4	0	42.5	40.9
Hood left	56.4	30.9	0	0	0	0
Bottom left	38.8	38.2	53.4	37.6	34.5	40.7
Bottom middle	0	0	0	40.5	0	43.9
Bottom right	38.6	37.1	36.4	41.7	40	39.9
Right armpit	46.1	41.7	46.3	46.0	44.7	50.7
Right shoulder	44.0	39.7	36.2	38.6	47.1	60.2
Neckline right	44.5	32.2	40.7	37.6	45.0	36.1
Collar right	44.1	38.7	41.0	43.2	47.2	48.7
Collar left	55.3	48.4	40.2	54.2	51.9	60.2
Neckline left	34.3	33.8	41.6	29.7	35.6	36.1
Left shoulder	44.4	39.1	40.6	32.3	46.7	48.7
Left armpit	43.3	38.8	34.2	41.8	39.0	37.4
Fold 1	27.5	26.4	18.7	23.3	27.5	25.2
Fold 2	0	0	0	0	0	0

Chapter 6

Conclusions and Future Work

6.1 Results discussion

With this dissertation we explored the architectures that form the basis of object detection and, with that, we were able to compile a simple and elegant model that performs garment classification+localization and landmark detection reasonably. Although the quantitative results may be some percentage points off from the state of the art in object localization and detection, we believe that there are still some opportunities to improve them, one of them being the exploration of Transposed convolutional and Up-sampling layers to increase the spacial resolution of the feature extractor module, and with that, being able to use deeper networks maintaining the same number of anchor points/boxes. The dataset used, is also somewhat limited for DL purposes as the number of images is relatively small, specially when looking into the garments that have special landmarks e.g., the 82 hoodies and 80 blouses, and considering the intra class in terms of variance in poses and wrinkling. We implemented the single landmark class per image trough the usage of the second *softmax* and cross-entropy functions but in the end this showed to be not much, and in terms of implementation, a bit cumbersome, due to the *softmax* disabling at training time and enabling at test time. Other approaches could also have been considered based on regularization like terms e.g., based the difference between the maximum and the second maximum $1 - \max_1 - \max_2$, but due to time constraints were not tested. Furthermore, we assumed that both the class loss and spacial constraint loss terms contribute equally to the loss function but, it would be interesting to evaluate different weights on these terms.

6.2 About Deep Learning

The field In opposition to other ML methods, ANNs aren't a static model, an algorithm, to perform classification or regression. Instead, ANNs define a structure, a framework, for the development of adaptive, in the sense that are capable to change its internal parameterization to adapt to

a certain goal, software. More than just apply and optimize its hyper-parameters to specific problems, or datasets, its study should be centered on the fundamental concepts i.e., the type of layers and modules, layers, loss functions and its applicability; and the gradient descent dynamics. With such understandings its is possible to construct more and more complex architectures to tackle broader problems e.g., generating the source code for a given interface mock up [Bel17].

Tensor reasoning When working with DL learning models, in particular implementing layers, it is important to do it in the most efficient way as possible, as this directly impacts the model execution speed and dictates the number of train trials, epochs and amount of data possible to be considered in a given time interval. Because of the intrinsically characteristics of ANNs, it is possible to take advantage of the data parallelism and, for example, process all the activations of a layer simultaneously (hence the usage and success of GPUs). Besides, as these operations are identical, writing them using the classic imperative programming would translate in the excessive usage of loops performing the same operation over multiple positions of the input tensor. This motivates the usage of array programming paradigm that is present in libraries such as NumPy or TensorFlow. Broadcast, slice and reshape operations and other quirks should be considered to take advantage of the lower level efficiencies e.g. a conditional sum over a tensor is better implemented by generating a binary mask with a vectorial boolean operator, followed by the element-wise multiplication of the mask with the original tensor and finally sum reduced, than using an accumulator, with an if case inside a loop.

Tools and libraries With the recent growth in popularity of the DL field, many are the libraries for the development of ANN models, yet a lot of space still exists for their improvement as these focus mostly on the code implementation and not the development as whole. Because of that a clear line between the development and the model itself does not exist. An IDE adjusted for this type of work would be ideal. Capabilities such as assembling the model using visual programming paradigms, present in tools such as RapidMiner; being able to write custom scripts, in this case, layers or loss functions, as in game engines such as Unity3d; monitoring of the training process and display of graphical information (metrics and loss functions); debugging of the learned parameters and active activations; would help not only speed up the development process but also would enable a better understanding of the model.

6.3 Future Work

We demonstrated, in this dissertation, a way to extract useful information from a given image existing in a dataset, but that does not necessarily means that this architecture, or even the considered problem is useful on the automation of the laundry folding task, our underlying goal. For instance, if we intend to use the output of our module, the garment category and its landmarks, as the information source of a controller that moves a robotic arm to pick up one piece, a question consequently emerges: what landmark should be chosen? And when considering the usage of DL

another should be also considered: Why do not let the model learn what's the best point? i.e., given an image, the model predicts one single, the best, point. Beyond that, if we consider that such machine might follow a pipeline as the described in [LCA14], where a series of grasps are performed until a certain state (pose) is achieved, why do not consider two adversarial networks, one that predicts a point, and another that, after the grasping of the suggested point is performed, evaluates the new pose and helps train the first? Such investigations would require an online, broader, training pipeline that could be performed using a real robot and clothes or virtual ones in a simulated environment and would diminish the gap the desired machine. We consider the Robotic Operating System (ROS) in conjunction with the Gazebo simulator, a good candidate framework for the development of such work, that among other features offers the environment (simulated or real) independent development.

References

[BB12] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, 13:281–305, February 2012.

[Bel17] Tony Beltramelli. pix2code: Generating code from a graphical user interface screenshot. *CoRR*, abs/1705.07962, 2017.

[C+15] François Chollet et al. Keras. https://github.com/fchollet/keras, 2015.

[ccp] Clothing co-parsing (ccp) dataset. https://github.com/bearpaw/clothing-co-parsing. Accessed: 2017-06-26.

[Chr15] Christopher Olah. Neural Networks, Types, and Functional Programming – colah's blog, 2015.

[dee] Large-scale fashion (deepfashion) database. http://mmlab.ie.cuhk.edu.hk/projects/DeepFashion.html. Accessed: 2017-06-26.

[dF16] Nando de Freitas. Learning to learn and compositionality in deep learning. Knouwledge Discovery and Data Mining, 2016.

[EGW+10] Mark Everingham, Luc Gool, Christopher K. Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *Int. J. Comput. Vision*, 88(2):303–338, June 2010.

[Ell16] Byron Ellen. The Most Frustrating Chore. *The Wall Street Journal*, 2016.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[GDDM13] Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, abs/1311.2524, 2013.

[Gir15] Ross B. Girshick. Fast R-CNN. *CoRR*, abs/1504.08083, 2015.

[HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[IS15] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

[Kar] Andrej Karpathy. Stanford university CS231n: Convolutional neural networks for visual recognition.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, 2012.

[LCA14] Yinxiao Li, Chih-Fan Chen, and Peter K. Allen. Recognition of deformable object category and pose. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2014.

[LLQ+16] Ziwei Liu, Ping Luo, Shi Qiu, Xiaogang Wang, and Xiaoou Tang. Deepfashion: Powering robust clothes recognition and retrieval with rich annotations. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[MPKM15] Ioannis Mariolis, Georgia Peleka, Andreas Kargakos, and Sotiris Malassiotis. Pose and category recognition of highly deformable objects using deep learning. In *2015 International Conference on Advanced Robotics (ICAR)*, pages 655–662. IEEE, jul 2015.

[MSCTLA10] J. Maitin-Shepard, M. Cusumano-Towner, J. Lei, and P. Abbeel. Cloth grasp point detection based on multiple-view geometric cues with application to robotic towel folding. In *2010 IEEE International Conference on Robotics and Automation*, pages 2308–2315, May 2010.

[NHH15] Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. Learning deconvolution network for semantic segmentation. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, ICCV '15, pages 1520–1528, Washington, DC, USA, 2015. IEEE Computer Society.

[RDGF15] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015.

[RF16] Joseph Redmon and Ali Farhadi. YOLO9000: better, faster, stronger. *CoRR*, abs/1612.08242, 2016.

[RHGS15] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015.

[Ron14] Max Ronny. Behavior Analytics Case Study Behavior Analytics for Smart Carts and Fitting Rooms, 2014.

[Rud16] Sebastian Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016.

[SHK+14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, January 2014.

[SLJ+14] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.

[SZ14] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

REFERENCES

[vdMH08] Laurens van der Maaten and Geoffrey E. Hinton. Visualizing high-dimensional data using t-sne. *Journal of Machine Learning Research*, 9:2579–2605, 2008.

[Yam16] Kimitoshi Yamazaki. Instance recognition of clumped clothing using image features focusing on clothing fabrics and wrinkles. *2015 IEEE International Conference on Robotics and Biomimetics, IEEE-ROBIO 2015*, pages 1102–1108, 2016.

[YES+14] Dong Yu, Adam Eversole, Mike Seltzer, Kaisheng Yao, Oleksii Kuchaiev, Yu Zhang, Frank Seide, Zhiheng Huang, Brian Guenter, Huaming Wang, Jasha Droppo, Geoffrey Zweig, Chris Rossbach, Jie Gao, Andreas Stolcke, Jon Currey, Malcolm Slaney, Guoguo Chen, Amit Agarwal, Chris Basoglu, Marko Padmilac, Alexey Kamenev, Vladimir Ivanov, Scott Cypher, Hari Parthasarathi, Bhaskar Mitra, Baolin Peng, and Xuedong Huang. An introduction to computational networks and the computational network toolkit. Technical report, October 2014.

[YLL15] Wei Yang, Ping Luo, and Liang Lin. Clothing co-parsing by joint image segmentation and labeling. *CoRR*, abs/1502.00739, 2015.

[YY11] Ming Yang and Kai Yu. Real-time clothing recognition in surveillance videos. In Benoît Macq and Peter Schelkens, editors, *ICIP*, pages 2937–2940. IEEE, 2011.

[Zei12] Matthew D. Zeiler. ADADELTA: an adaptive learning rate method. *CoRR*, abs/1212.5701, 2012.

Appendix A

Other experiments

During the development of this dissertation many other experiments that aren't described in the main portion of this document. Of all of them, we pick two that we found the results to be interesting, and describe them quickly here.

A.1 Semantic segmentation with Deconvolution Network

We build a eight layer netowrk, following the model described in [NHH15]: a first Convolutional with three filters, followed by three Convolutional with sixty four filters each, and then three Convolutional Transpose with also sixty four filters each. Intercalated with these layers there are Batch Normalization and ReLU activations. We also use L2 regularization on the kernel parameters. Regarding the last layer and loss function, we experiment with two options:

1. The first, wherein the last layer has three filters, representing RGB components, and the loss function is the mean absolute error.

2. The last layer has fifty eight filters encoding the fifty eight available classes (fifty seven and background), and is followed by a depth wise *softmax* activation function. The loss function is the cross entropy.

The distinct types of output predictions are shown in A.2.

A.2 Pants landmarks localization

When experimenting with the detection of garments we also attempted to useFC layers in the top of the output detection and simply regress the coordinates of the existing landmarks, pre trained on ImageNet, to wich we remove the top FC layers and attach and intermediate 512-d FC (with Batch Normalization and ReLU), followed by a $2 \times n$ FC layer, that regresses the x,y landmark coordinates for each of the n landmarks. Cowardly the model learned to predict all the landmarks in the middle of the image.

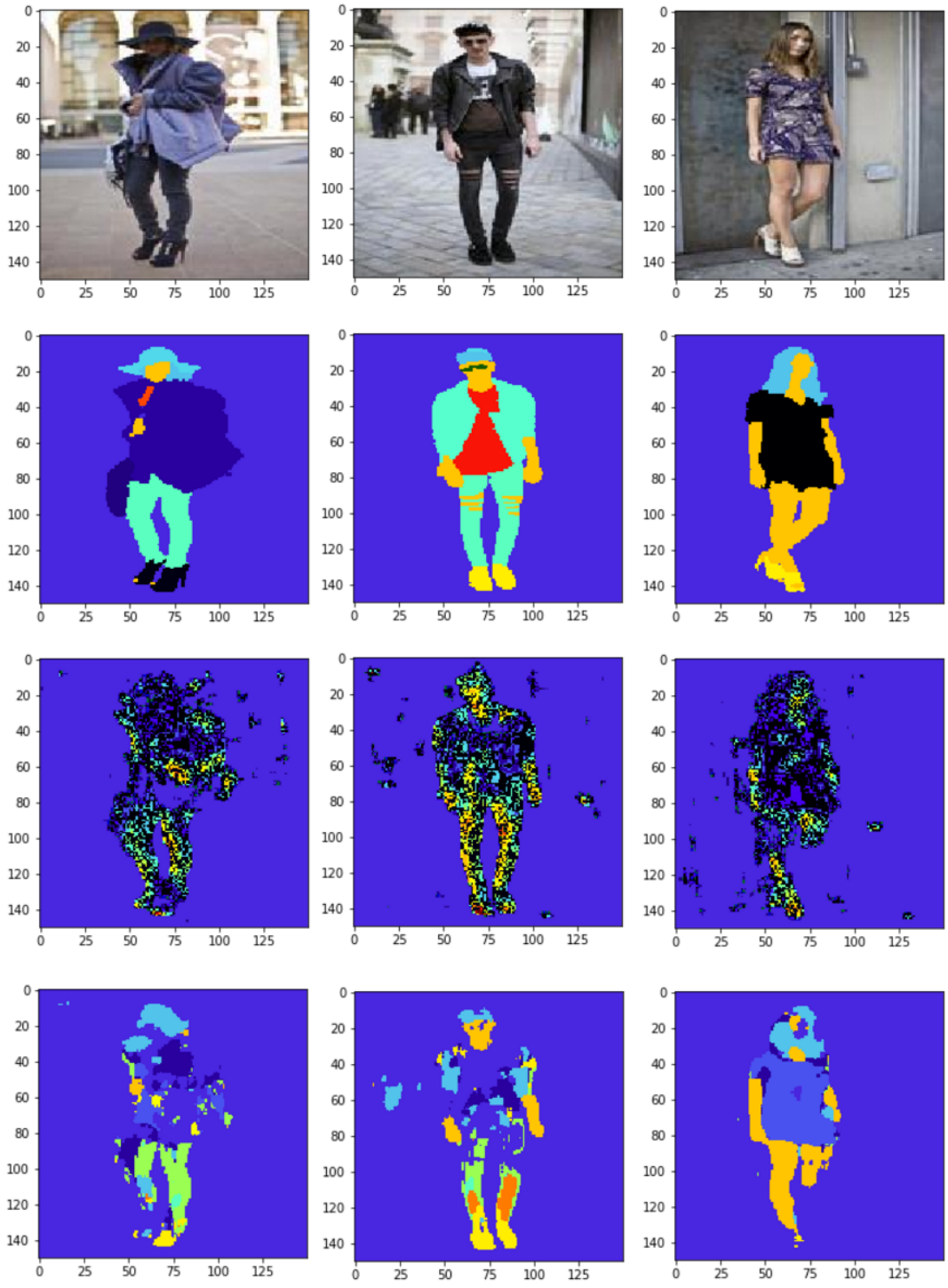

Figure A.1: Image segmentation with Deconvolution Network. On the first second rows, the input and ground truth. On the third and forth rows, the first and second approaches predictions.

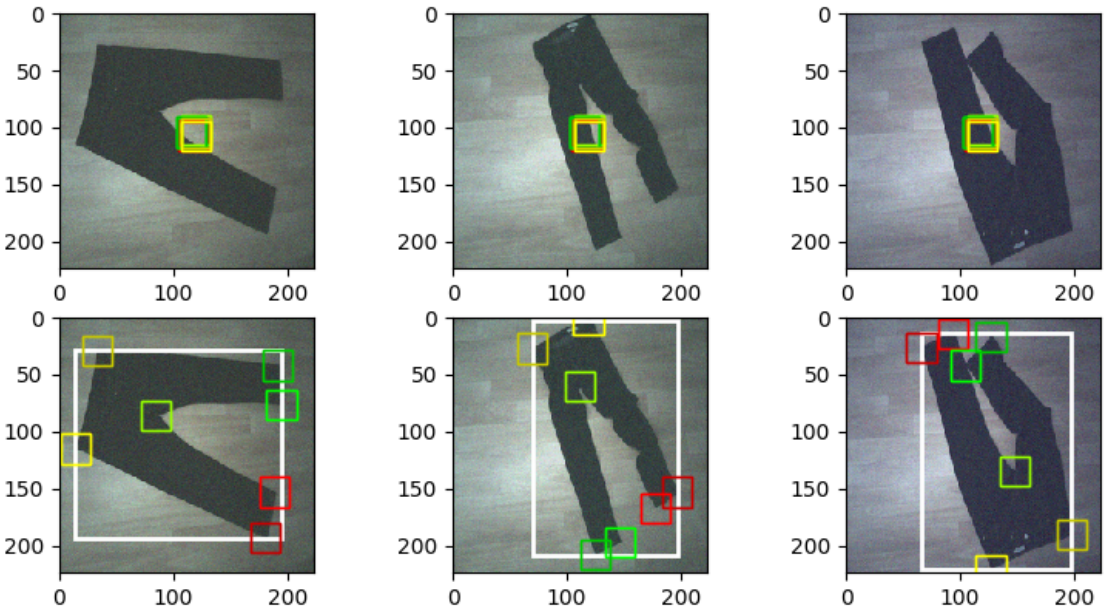

Figure A.2: Pants landmark localization. On on the top row, the predictions, on the bottom, the ground truths. (The garment bounding box, appearing in the ground truths, is not being predicted by the model)

Appendix B

Dataset detailed description

B.1 Garment classes

Table B.1: Counting of garment classes, per class and dataset split.

	Train	Train (balanced)	Validation
Pants	1054	1054	141
Polo	266	532	34
Hoody	82	410	10
Tshirt	291	873	33
Tshirt long	222	666	31
Polo long	163	815	22
Towel	95	570	11
Skirt	65	520	6
Bluse	80	640	12

B.2 Landmark classes per garment categories

Ahead from the landmark classes described in the table B.2 two other classes exist, *fold1* and *fold2*, that appear in examples retrieved from the Flat and Folded portion of the CTU dataset , wherein one fold has been performed on the garment piece.

Table B.2: Distribution of landmarks across garment categories

	Pants	Polo	Hoody	Tshirt	Tshirt long	Polo long	Towel	Skirt	Bluse
Left leg outer	✓								
Left leg inner	✓								
Crotch	✓								
Right leg inner	✓								
Left leg inner	✓								
Top right	✓						✓	✓	
Top left	✓						✓		
Right sleave inner		✓	✓	✓	✓	✓			
Right sleave outer		✓	✓	✓	✓	✓			
Left sleave inner		✓	✓	✓	✓	✓			
Left selave outer		✓	✓	✓	✓	✓			
Hood right			✓						
Hood top			✓						
Hood left			✓						
Bottom left		✓	✓	✓	✓	✓	✓	✓	✓
Bottom middle									✓
Bottom right		✓	✓	✓	✓	✓	✓	✓	✓
Right armpit		✓	✓	✓	✓	✓			✓
Right shoulder		✓	✓	✓	✓	✓			✓
Neckline right		✓		✓	✓	✓			✓
Collar right						✓			✓
Collar left						✓			✓
Neckline left		✓		✓	✓	✓			✓
Left shoulder		✓	✓	✓	✓	✓			✓
Left armpit		✓	✓	✓	✓	✓			✓
N.E. Train	1054	532	410	873	666	815	570	520	640
N.E. Val	141	34	10	33	31	22	11	6	12

B.3 Data augmentation

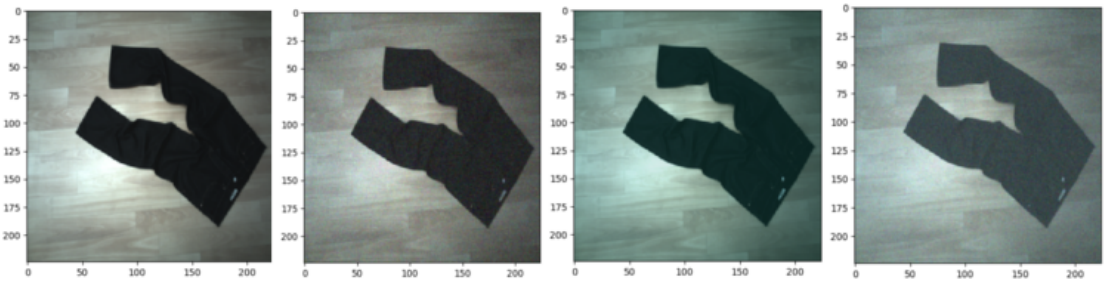

Figure B.1: Augmentation operations. From the left to the right: original, Gaussian noise, hue noise, Gaussian followed by hue noise.

Appendix C

Loss functions history

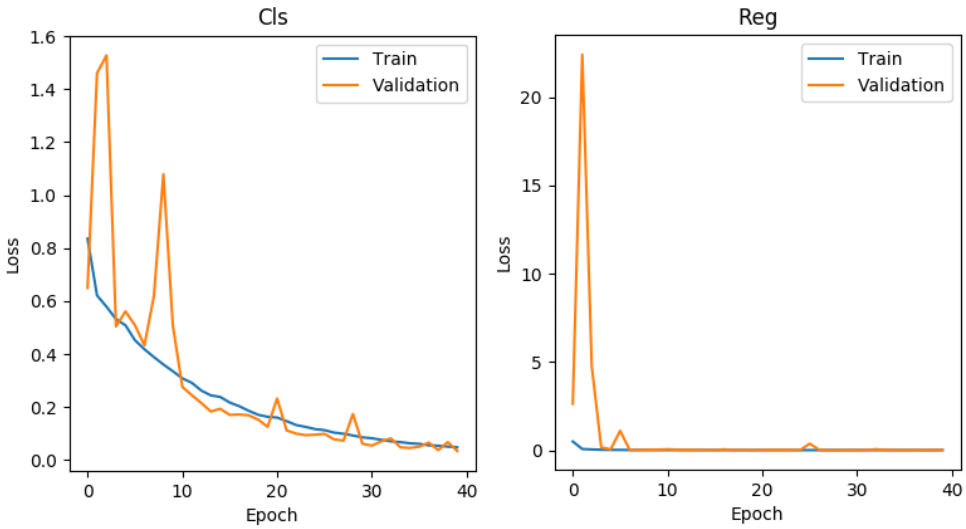

Figure C.1: Garment localizer branch isolated training.

Loss functions history

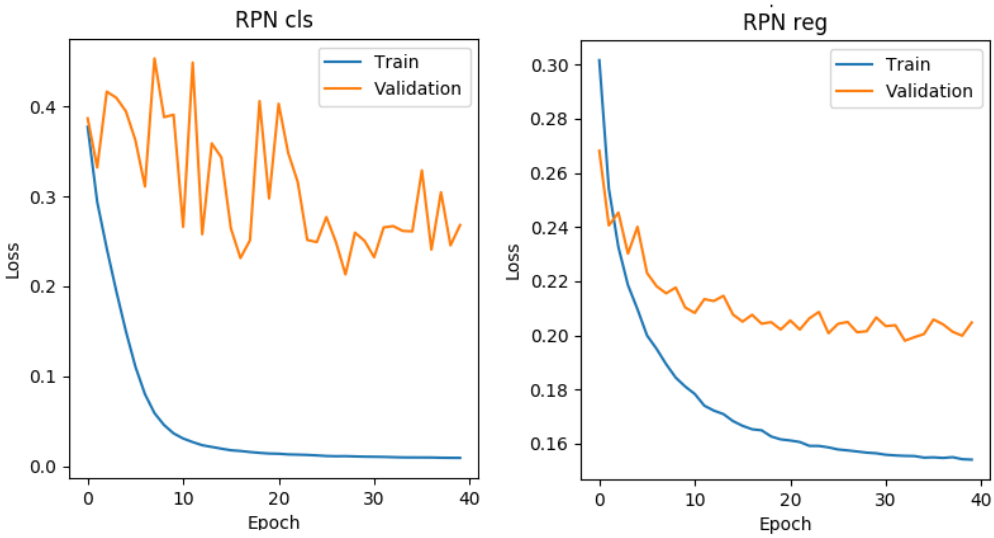

Figure C.2: Landmark detector branch isolated training, without spacial constraint.

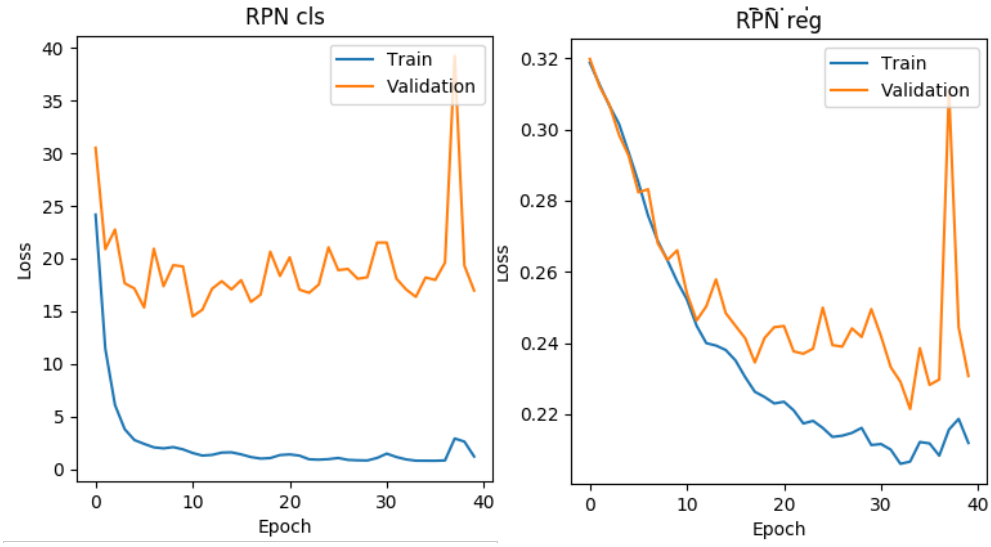

Figure C.3: Landmark detector branch isolated training, with spacial constraint.

Loss functions history

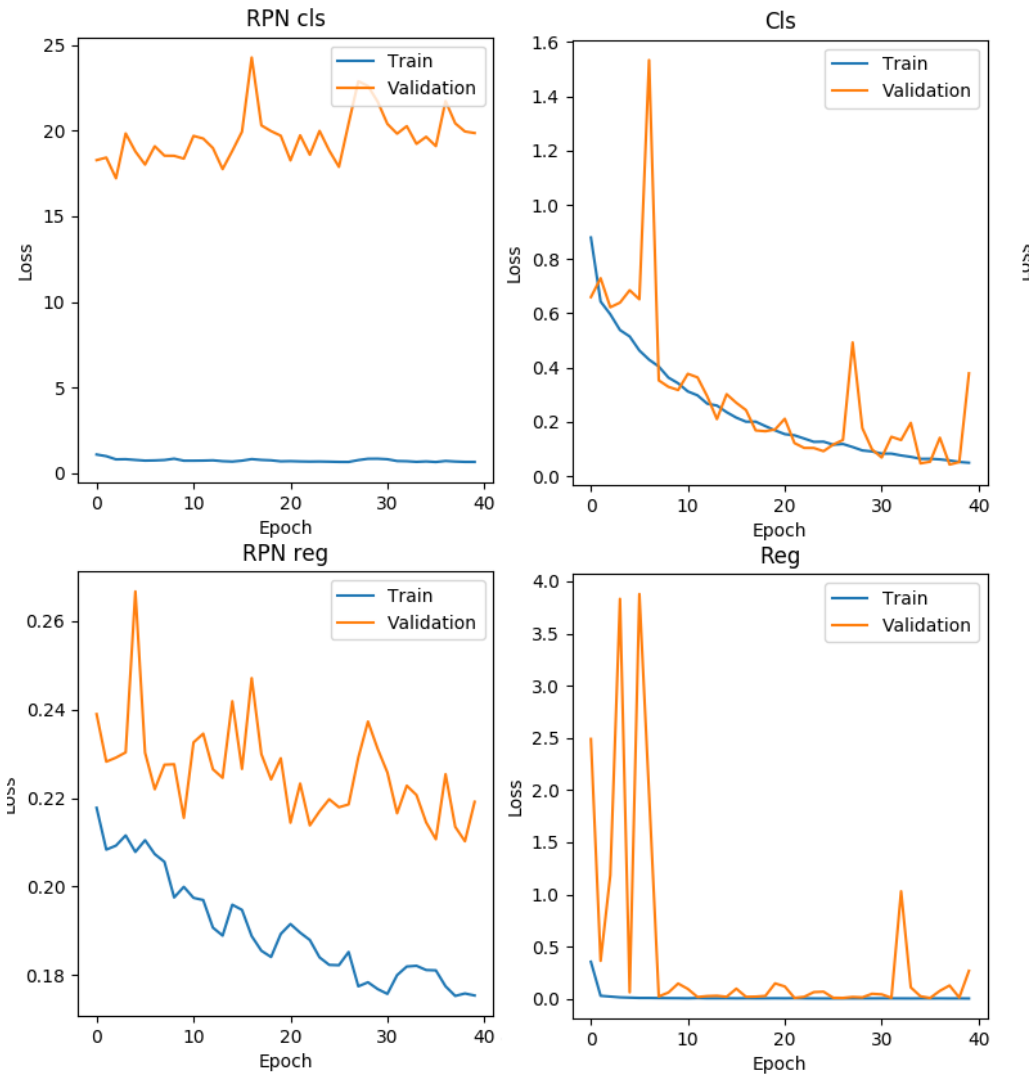

Figure C.4: Full model, with double softmax and without bridge connection.

Loss functions history

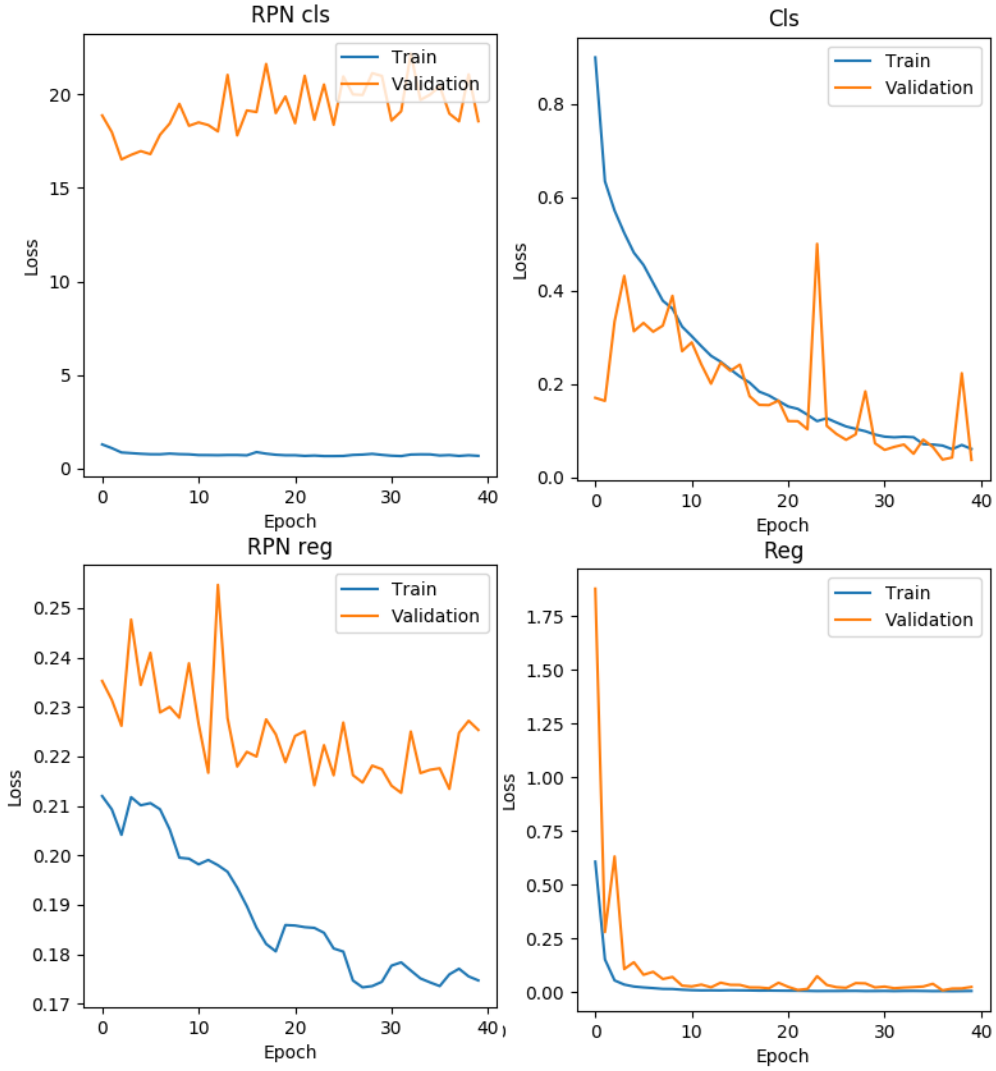

Figure C.5: Full model, with double softmax and with bridge connection.

Loss functions history

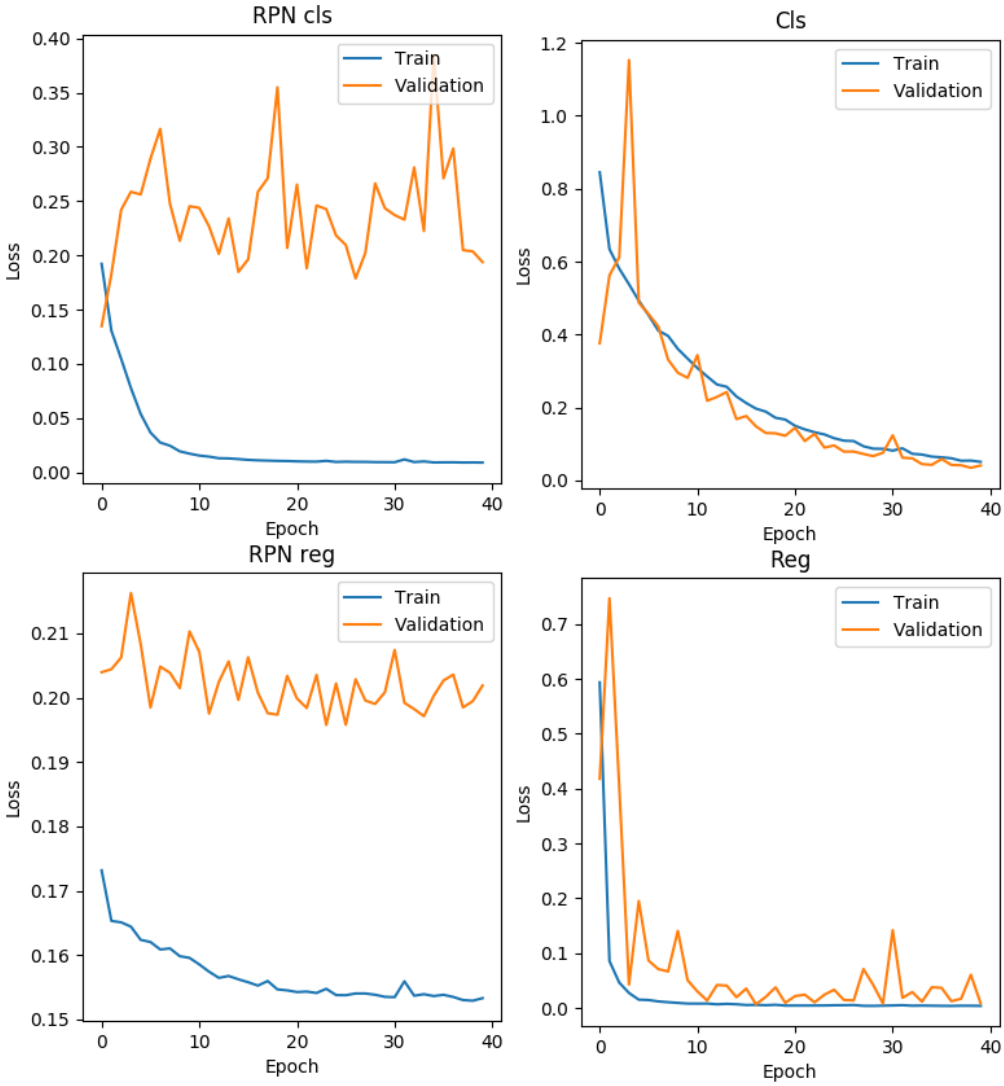

Figure C.6: Full model, without double softmax and with bridge connection.