UNIVERSITÀ
DEGLI STUDI
FIRENZE

PHD PROGRAM IN SMART COMPUTING
DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE (DINFO)

Deep Learning Methods for Document Image Understanding

Samuele Capobianco

Dissertation presented in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Smart Computing

PhD Program in Smart Computing
University of Florence, University of Pisa, University of Siena

Deep Learning Methods for Document Image Understanding

Samuele Capobianco

Advisors:

Prof. Simone Marinai
Prof. Kimbal Marriott

Head of the PhD Program:

Prof. Paolo Frasconi

Evaluation Committee:
Prof. Stefano Ferilli, *Dipartimento di Informatica, Universitá di Bari*
Prof. Josep Lladós, *Computer Vision Center, Universitat Autònoma de Barcelona*

XXXII ciclo — October 2020

What we think, we become.
– Siddharta Gautama

Acknowledgments

The journey towards getting a Ph.D. is full of up and downs, failure and successes. Very special gratitude goes out to all down at Regione Toscana for helping and providing the funding for Pegaso fellowship.

I would like to express my deep and sincere gratitude to my first research advisor Prof. Simone Marinai for giving me the opportunity to do research and providing invaluable guidance throughout this research. In particular, I would like to thank my second research advisor, Prof. Kim Marriott, for the hospitality received during my research period at Monash University and for his participation in the survey who helped me get results of better quality.

I am also grateful to Prof. Andrew David Bagdanov and Prof. Jean-Marc Ogier, the members of my supervisor committee, for their patience and support in overcoming numerous obstacles I have been facing through my research. I would also like to thank the experts who were involved in this research: Prof. Paolo Frasconi and Prof. Luca Facheris.

With a special mention to Daniele Baracchi, Alessandro Lazzeri, Stefano Martina, Francesco Orsini, La Ode Husein ZT, Amin Zadenoori, Zahra Ziran, and AI Team in general. I would also like to acknowledge my colleague Anuradha Madugalla, and undergraduates Giulio Bazzanti, Niccolo Biondi, Andrea Gemelli, Andrea Marino, Leonardo Scommegna for their contributions to my research. It was fantastic to have the opportunity to work the majority of my research in your company. What a cracking place to work!

Last but not least, I would like to thank my mother for supporting me spiritually throughout writing this thesis and my life in general. Furthermore, this research path would not have been possible without my darling Martina, thanks for everything.

And finally, last but by no means least, also to everyone in the impact hub, it was a great sharing laboratory with all of you during the last three years.

Abstract

Document image understanding involves several tasks including, among others, the layout analysis of historical handwritten and the symbol recognition in graphical documents. The understanding of document images implies two processes, the analysis and the recognition, which are complex tasks. Moreover, each application domain has a specific information structure which increases the complexity of the understanding process.

In the last years, many machine learning approaches have been presented to address document image understanding. In this research, we present a series of deep learning methods to address different application domains: historical handwritten and graphical documents understanding. We show the difficulties encountered when applying these techniques and the proposed solutions for each application domain. We cope with the problem of working with supervised deep networks that require to have a large dataset for training. We address the over-fitting related to the scarcity of labeled data showing several solutions to prevent this issue in these application domains.

First, we show our contributions to historical handwritten layout analysis. We propose a toolkit to generate structured synthetic documents emulating the actual document production process. Synthetic documents can be used to train systems to perform layout analysis. Then, we study the use of deep networks for counting the number of records in each page of a historical handwritten document. Furthermore, we present a novel approach for the extraction of text lines in handwritten documents using another deep network to label document image patches as text lines or separators. Related to the page segmentation, we propose a fully convolutional network trained by a domain-specific loss for classifying pixels to segment semantic regions on handwritten pages.

Second, we propose a novel interactive annotation system to help users to label symbols at the pixel level for the graphical symbol understanding problem. Using the proposed interactive system we can improve the annotation results and reduce the time-consuming process of labeling data. Using this system, we built a novel floor plan image dataset for object detection. We show preliminary results by using state-of-the-art deep networks to detect symbols on this dataset.

In the end, we provide an extensive discussion for each task addressed showing the obtained results and proposing future works.

Contents

Contents **1**

1 Introduction **3**
1.1 Goals of this Thesis 5
1.2 Contributions of the Thesis 6
1.3 Outline 7

2 Deep Learning for Document Analysis **9**
2.1 Document analysis and recognition applications 9
2.2 Deep Learning methods 13
2.3 Representation Learning for object recognition 14
2.4 Object detection 24
2.5 Semantic segmentation 26
2.6 Variational Autoencoder 31
2.7 Labeling systems 33
2.8 Summary 34

3 Validating Deep Learning models **35**
3.1 Layout analysis datasets 35
3.2 Floor plan datasets 38
3.3 Evaluation metrics 41
3.4 Summary 43

4 Generating training data **45**
4.1 Labeling graphical documents for symbol detection 46
4.2 Clustering visually similar structured documents 54
4.3 A Toolkit to generate structured documents 56
4.4 Modeling datasets 62
4.5 Generating graphical symbols 65
4.6 Summary 74

5 Record counting in historical handwritten documents **75**

5.1 Record counting system 76
5.2 Convolutional models 78
5.3 Evaluating system 79
5.4 Experiments 80
5.5 Summary 86

6 Historical handwritten page analysis **89**
6.1 Patch-based system for text line extraction 90
6.2 Evaluating patch-based system 94
6.3 Pixel-wise page segmentation 99
6.4 Evaluating pixel-wise page segmentation 103
6.5 Summary 105

7 Conclusions **107**
7.1 Future works 108
7.2 Contributions 109

A Publications **111**

Bibliography **113**

Chapter 1

Introduction

In this thesis, we investigate document image understanding by using deep learning methods addressing an assortment of challenging applications. Document image understanding includes the logical comprehension of arbitrary documents. This process transforms the informative content into a logical content representation. This transformation involves high-level deduction which is difficult to be computed by an automatic system.

For example, a picture from an identity card contains information that a human can easily read and understand, assigning a meaning to each informative region. Anyway, the understanding document image is not just a matter of translating graphical symbols into text form but assigning a semantic structure to the extracted knowledge.

Understanding the document images is a difficult task because each application domain has a specific information structure. For example, an image from a business letter or one historical document image need two different understanding approaches to transform the informative content into a well-organized format.

Document image understanding is related also to graphics. Graphical symbols are considered as visual design or icon which represent the informative content related to the document context. In particular, the visual elements are special drawings of specific object categories that differ from real image representations.

In this research, we address the document image understanding process in two different application domains: historical handwritten and graphical documents. We see the peculiarities of these topics and how we can solve the related issues by using deep learning methods.

A historical handwritten collection can be produced by various writers through the time showing different character shapes or writing styles. Moreover, the aging of documents caused degradations which make harder the layout analysis and text recognition tasks.

The historical value of these documents requires digitization for better preserva-

tion in Digital Libraries. One of the primary aims of digital libraries is to collect and organize information that can be accessed all over the world. Clearly, digital information needs less space than paper-based information reducing the library costs. Another advantage coming from the digitization, is the reduction of the access time to the information. Digitization is also a way to preserve paper information from the damage and wearing effect of time. Having digitized documents it is possible to use automatic techniques to analyze and extract structured information from the original document. To extract the content from different handwritten collections, we require suitable methods.

In the last years, many machine learning approaches have been presented to recognize text or analyze layout in historical documents. In this research, we present a series of machine learning methods to understand historical handwritten collection showing the difficulties encountered when applying the proposed methods.

Graphical documents contain symbols which are related each others, these relations have different meaning depending on the domain. These symbols are composed of line drawings as lines, arcs, etc. Graphical symbol recognition is one of the most active research area in document image understanding field. Diagrams are composed of visual elements used to describe the domain content. Some of the most relevant application domains which use graphical recognition techniques belong to engineering fields: electrical and logic circuit diagrams, engineering diagrams, architectural drawings, and maps.

Considering how wide is the variety of documents for some of these two proposed tasks the main purpose of this thesis is to devise efficient methods for document image understanding both in handwritten and graphical documents. We investigate how recent deep learning methods can advance layout analysis in historical documents or detect symbols in floor plan images.

Recently, deep learning methods have solved many complex problems achieving state of the art performances. These techniques are mainly focused on training neural network architectures able to solve a wide variety of pattern recognition tasks. The most important applications in computer vision are based on deep learning architectures, but the great success of these techniques is also related to the amount of training data available for learning in these application domains.

One of the first ingredients for the initial progress in deep learning has been the advent of fast graphics processing units (GPUs) which allowed the researchers to train networks 10 or 20 times faster than before[70]. The second ingredient to have better results with deep learning models is related to the training data that should represent the complexity of real data in a given application.

A large number of parameters have to be learned when we train a deep learning architecture meaning a huge labeled dataset is necessary. Labeling data is a hard time-consuming job, it is not easy to find public labeled datasets for every task and

we often need to label data when we address novel problems.

In this thesis, we propose different deep learning architectures to address historical handwritten and graphical document understanding. We focus on supervised learning methods showing different training methodologies to solve over-fitting caused by the scarce quantity of labeled data.

1.1 Goals of this Thesis

In this thesis, we want to investigate the use of new techniques to improve the capability for document image understanding to address various tasks. We are interested to explore the use of deep neural network architectures recently proposed in other domains, such as in computer vision. In particular, we want to address the layout analysis on historical handwritten documents and graphical symbol understanding on floor plan images.

The layout analysis on historical handwritten documents is complex because each application domain has different document structures. This problem is a good test case for deep learning architectures. Unfortunately, another related problem is the scarcity of labeled data for training models which makes the learning more complex. Over-fitting occurs when models achieve good performance on the training data, while they do not generalize well on unseen data. We study and extend deep learning methods to apply on this topic to prevent over-fitting. For this topic we ask the following questions:

- Is it possible to generate synthetic handwritten documents useful to extend a small labeled collection?
- Can we use synthetic documents to train deep networks on the layout analysis?
- Can we segment historical handwritten documents into regions training a deep network with soft-constraints based on layout information?

Then, the graphical symbol understanding is addressed in this thesis, similar problems related to those pointed out before need to be treated in this domain. Also for this topic, the main aspect to solve is how to manage data for learning. Starting from scratch, annotating data is a very labor-intensive task and we want to reduce this time-consuming process. For this topic we have other questions to address:

- Is it possible to reduce the time for annotating data in the graphical symbol domain?
- Can we learn to generate graphical symbols directly from few labeled drawings?

- Is it possible to use deep learning methods to detect objects on graphical documents?

Our research is focused therefore on learning from data, especially when we want to train deep architectures with a scarce quantity of labeled data. We aim to contribute to the understanding and development of deep network models to produce systems able to learn in these conditions and defining well-suited solutions for the proposed topics.

We use deep architectures because differently to other machine learning approaches, deep networks can learn useful representations from raw data making this approach more suitable to different tasks. Our research also shows as layout analysis and graphical symbol understanding can be solved by using the representation learning capability of deep networks. On the other hand, deep learning methodologies give a set of general-purpose solutions to use in different contexts and it is possible to solve specific challenges in different domains adding ad-hoc operations in the same computational flow.

1.2 Contributions of the Thesis

This research is focused on solving different document image understanding tasks with deep learning methods. We made a valid step forward historical handwritten layout analysis. We explored deep learning solutions on graphical symbol understanding proposing a novel dataset. We summarize the major contributions of the thesis as follows:

- We have proposed a novel document generator to produce structured historical handwritten pages to efficiently extend the training data improving performances on layout analysis. Combining the generator ability with deep learning architectures we can obtain good results with less labeled data.

- We have developed deep architectures based on different learning schemas to address page segmentation and text line extraction in historical documents. The first is a patch-based convolutional network to extract contiguous text lines. The second is a fully convolutional network trained by a domain-specific loss for classifying pixels to segment semantic regions on the page.

- Labeling data is a labor-intensive task, especially when we need to label small and sparse objects. We have developed a toolkit to help the user to label symbols at the pixel level. Using the proposed interactive toolkit we can improve annotation results and reduce the time-consuming process of labeling data.

- We have investigated how to use deep learning methods for generating and detecting symbols in floor plan images to improve results having scarce data. A generative model can produce new symbol instances similar to the originals.

The code and dataset related to this research are accessible online and, we hope these contributions will be extended by other researchers improving the work done till now.

1.3 Outline

This thesis is organized in other six chapters as follows:

- Chapter 2: A brief introduction on document analysis and recognition (DAR) tasks. Followed by an introduction to deep learning methods for computer vision and DAR systems.
- Chapter 3: The effective performance evaluation of deep networks models require datasets and metrics. We present datasets used in this research and also metrics to validate the obtained results.
- Chapter 4: Learning deep models in a supervised manner needs a lot of labeled data. We show several techniques to label, select and generate data useful for training models.
- Chapter 5: The layout analysis for handwritten documents is a complex task. We show a deep learning solution to understand structured documents by counting page records.
- Chapter 6: On historical handwritten document we want segment pages into different semantic regions. We propose two solutions for text line extraction and page segmentation based on different deep network architectures.
- Chapter 7: We close this final dissertation highlighting on the obtained results and how to improve this research with possible future works.

Chapter 2

Deep Learning for Document Analysis

The information extraction from documents, either scanned or digital-born, requires the integration of several techniques in Document Analysis and Recognition (DAR). DAR applications are built to compute a suitable symbolic representation from input documents by computer systems.

Several applications in DAR field are traditionally related to the processing of office documents. Recently, the community research aims to use DAR techniques also on ancient/historical documents in digital libraries and natural images containing textual information.

This is important also nowadays since new application scenarios require the digitization of large collections of historical documents and an increasing number of documents are digitally produced.

Several competencies in computer science are needed to develop DAR systems, in particular, image processing, pattern recognition, and artificial intelligence are common skills useful to build these kinds of applications.

A very good introduction to DAR systems has been proposed by [86]. The authors propose an overview of DAR systems. It is important to also give a brief introduction to deep learning methods applied to computer vision and document image understanding tasks.

2.1 Document analysis and recognition applications

Common DAR techniques have been used to develop commercial and research-driven systems. It is possible to define two broad categories related to DAR application which are business-oriented and user-centered ones.

For business-oriented applications, we can include automatic check processing (amount reading, signature verification), invoice reading and page classification systems. Instead, for a user-centered application, we can mention software for general purpose as OCR which is useful to extract information from a paper form. Moreover,

other tools have been developed to improve access to documents in digital libraries and the processing of historical documents.

Large collections of digitized documents are available on the Internet, these digital libraries are a source of knowledge to extract by DAR techniques.

We can describe the document content with two different aspects, *physical* and *logical* structures. The *physical* structure describes the visual object and their relative positions on the document. The *logical* structure assigns to each object a suitable meaning.

Processing steps

DAR applications, following other Pattern Recognition Systems, include four principal components as pre-processing, object segmentation (or detection), object recognition, and post-processing. The pre-processing aims at improving the quality of images before other computation phases. The object segmentation allows identifying (detect) basic objects (or sub-parts) in the document. The object recognition, or classification, deals with assigning a category label to a document. The post-processing manages the results from the previous phases into a final symbolic representation which represents the computed result.

DAR applications can be categorized into two principal approaches: top-down or bottom-up. The major differences between these two approaches are related to the structure of the document layout. When document structure is prior knowledge of the analysis, it is possible to use a model-driven approach. Instead, when the document structure is not known in advance it is possible to use a data-driven approach.

Preprocessing

The document acquisition process used to digitize the original paper is the first step in document analysis. Using the flatbed scanner is possible to create a digitized version of paper-based documents, in digital libraries also book scanners can be used to produce an electronic image from the source.

Pre-processing operations are useful to prepare data for the following analysis phases. The same operation in document image analysis transforms the input into a better image representation.

To reduce the image noise or produce a better representation, one possible solution is to filter the input image whose value in a generic position (i, j) is related to the input values in a neighborhood of the point (i,j). The main classes of filtering operations are binarization, noise reduction, and signal enhancement.

Binarization is the process of converting a greyscale pixel image to a binary image, one common solution for this task is to use a threshold on the input to sep-

arate pixels into two categories (black, white). The Otsu method [94] is a way to find a threshold considering the histogram of pixel values. Then, iterating through all the possible values, it is possible to evaluate the measure of spread related to the foreground and background variance. The best value is the computed threshold. Sauvola method [105] is a local thresholding technique that is useful for images where the background is not uniform.

Many feature enhancement algorithms have been applied as preprocessing in the document analysis field, the difference of gaussian (DoG) is one of them. Having grayscale images, it is possible to compute DoG by subtracting two different blurred versions of an original image (two blurred images are obtained by convolving the original grayscale images with Gaussian kernels having differing standard deviations).

Classification in DAR

Document image classification is an important task to identify the document structure. This task is used in various Document Image Processing Pipelines, such as in document retrieval, information extraction, and text recognition. Document classification is considered overall as a learning of the document layout to classify it in a genre. The basic assumption behind document classification approaches is the observation that we are often able to identify the document function by simply looking to its organization considering, e.g., the space between cells, the font used, the number of columns.

It is possible to summarize the various DAR sub-tasks in which solutions are based on supervised classifiers. Most of these approaches consider specially designed local and global features along with their combinations for classification. Generally, document image classification approaches are divided into two major groups, structure/layout based, and content-based.

We can find several approaches to structure or content based on document classification. Recent research has used various techniques, from region-based analysis to whole image analysis, and at the same time, from handcrafted features to representation learning way. In [67] the authors proposed a method based on patch-codewords over different regions of the image as a "bag-of-words" representation to be used for the retrieval of document images with chosen layout characteristics. It uses spatial relationships between patches partitioning recursively the input image in vertical and horizontal parts and then compute a histogram of patch-codewords. Another work has been proposed by [60] where they want to index linear singularities and curved handwritten shapes in documents images. Using a Curvelet transform they can represent several scales of details for the handwritten shape to detect oriented and curved fragments. The adaptable system described in [8] is capable of

learning the logical structure of documents. This structure induces to cluster and organize a concept hierarchy to classify unknown documents. The first approach based on Convolutional Neural Networks to classify document genres is proposed by [69]. In their work, they use a simple CNN using dropout and ReLU activation function. In [51] the authors proposed a large dataset to train a Convolutional Neural Network to classify 16 document genres.

Layout Analysis

In document analysis research, layout analysis is one important task to segment and recognize a document page into regions having a homogeneous content. Physical layout analysis is used to identify the geometric page structure. The logical layout analysis assigns a logical meaning to each region generating the logical structure of the document.

The two major approaches for layout analysis are bottom-up and top-down methods. Localizing connected components with subsequent aggregation in higher-level structures is a bottom-up approach. Then, the top-down analysis considers the whole page to segment in larger components to smaller sub-components. Top-down methods are faster but need documents with a regular layout.

The logical layout analysis assigns a meaning to previously identified regions by physical analysis method. Often, physical and logical analysis have been computed concurrently assigning meaning to blocks during the prediction phase.

The task to extract homogeneous components from a page image is named page segmentation. It is possible to define a physical layout analysis task as a combination of page segmentation and classification. One of the most important applications is to extract text-line or segment characters from printed document images. Several techniques have been presented to address text line segmentation on historical documents [75]. Instead, one common solution to detect and recognize text in printed documents is an optical character recognition (OCR) named Tesseract [112] which includes line finding, features/classification methods, and an adaptive classifier.

Drawings understanding

Architectural floorplans define indoor spaces and these drawings are designed by software that produces a vector-graphics representation. Often, this representation needs to be rasterized to print or digital media for publication. In this way, we lose representation which we need to reconstruct the lost information from a rasterized floor-plan image.

In particular, the visual elements are special drawings for specific object categories that differ from real image representations. In the case of floor plans, we

have mainly black and white symbols that map the real object shape. One interesting application in this research field is [32] where the authors propose a system for generating synthetic graphics documents (including floor plans) that contain visual symbols in a real context. However, symbols in [32] are nearly identical one to the other and do not show significant and realistic variability between different instances of the same type of objects.

Recently, another work [129] has been proposed to address the object detection task in the floorplan image by using a deep network architecture. In Section 4.5 we propose a solution to generate graphical symbols with a suitable generative network.

2.2 Deep Learning methods

In the early day of the Artificial Intelligence era, machines were able to solve tasks following hard-coded programs. Defining soft or hard rule was possible to solve well-defined, logical problems but not complex tasks as object recognition or document classification tasks. Machine learning and deep learning techniques have been developed to solve this (and more) complex tasks.

Machine learning is a sub-field of Artificial Intelligence research that aims to build data-driven models to solve complex tasks. Machine learning systems usually need a lot of data used to build predictive analytics models.

Building good machine learning methods is heavily dependent on the choice of data representation (or features) on which they are applied. The data representation is the first step to deploy machine learning algorithms designing preprocessing pipelines and data transformations directly on input data. Feature engineering is the most labor-intensive phase where the goal is to describe input data to perform better machine learning solutions[12]. Automatic feature engineering is a current hot topic in machine learning to learn representation directly from input data.

In most of the case, it is difficult to design suitable hand-craft features able to improve the learning algorithm performances. Representation learning is a set of methods to build algorithms able to discover object representation from raw data automatically. Learned representations are obtaining better performance compared to the use of hand-designed features [46].

Deep learning techniques are based on representation-learning methods that can learn multiple levels of representation with linear or non-linear transformation starting at raw data into a representation at a more abstract level. The methods are very good to discover intrinsic patterns in high dimensional data solving very complex tasks in different domains [70]. One common way to train this kind of neural network is based on an algorithm named stochastic gradient descent (SGD) which can update network parameters after computing a gradient vector of the objective function related to the learning process.

We could split machine learning into several sub-fields related to the type of learning [107]. In this thesis, we are interested in two types of learning: supervised and unsupervised. The learning process involves an interaction between learner and environment, this interplay defines the nature of learning.

One of the most common forms of machine learning, deep or not, is supervised learning. In this scenario the learner uses a predefined experience to gain expertise, each training example contains a label (experience) used to train the model. The acquired expertise during the training phase is used to predict the unseen test set examples.

In unsupervised learning, there is any difference between training and test dataset because, for the learner, data are without labels. In this case, the goal is to learn a representation for input data distribution preserving its structure.

In the following sections, we will explain some deep learning techniques for computer vision. In particular, we will show the reason why deep learning methods have gained a lot of interest in academia and business companies.

We will see three different aspects for image understanding which are similar to document image understanding. Object recognition, it takes an image as input and computes a class label of a set of categories. Object detection, it takes an image as input and computes object classification and localization on all the objects in the image detecting multiple bounding boxes. Semantic segmentation, it takes an image as input labeling each pixel with a class of objects.

2.3 Representation Learning for object recognition

Data representation is very important for the performance of machine learning algorithms. Many researchers around the world have worked hard to deploy machine learning algorithms in a more general way, these networks can capture the internal representation of the data to which they are applied. These efforts have been the basis for research named "representation learning". It is based on learning representations of the data that make it easier to extract useful information when building classifiers or other predictors [12].

During the application of deep learning on various tasks, it was empirically observed that layer-wise stacking of feature extraction often yielded better representations, e.g., in terms of classification error [36, 68], and in quality of the invariance properties of the learned features [47]. These approaches are based on the computation of several transformation layers to capture the intrinsic properties of the train data. In several computer vision works, a powerful approach is based on more basic knowledge of merely the topological structure of the input dimensions. Based on such a structure, one can define local receptive fields [57], so that each low-level feature will be computed from only a subset of the input: a neighborhood in the

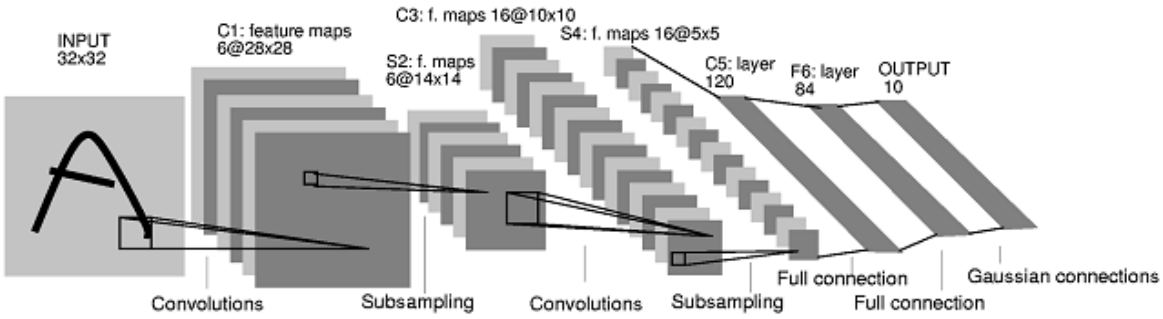


Figure 2.1: LeNet neural network is composed of 2 convolutional layers, 2 subsampling layers, and 3 fully connected layers. It is used for digit classification adapted by [71].

topology (e.g. a sub-image at a given position). This idea comes from the receptive field properties found on the primary visual cortex of vertebrate species. A good way, then, is to discover the topological structure of the input by moving the receptive field like a local feature extractor: this corresponds to a convolution that transforms the input into a similarly shaped feature map.

In the following sections, we will show how deep networks learn a representation directly from raw data during the training phase. We put more emphasis on Convolutional Neural Network (CNN) architectures which are the most used deep architectures in computer vision.

LeNet

A successful neural network model for handwritten digit recognition is the Convolutional Neural Network proposed by [71]. This model combines three architectural ideas to obtain scale or distortion invariance: local receptive fields, shared weights, and spatial or temporal sub-sampling. A typical convolutional neural network for the digit recognition is shown in 2.1. In this model, as in a biological way, the neurons can extract elementary visual features such as edge, end-points, corners. These features are then combined by the subsequent layers to detect high-order features. In the end, the computed features are treated as representation features to be classified with Multilayer Perceptron classifier (MLP). Units in the transformation layers are organized in planes where all the units of the same plane share the same set of weights. These output planes are called feature maps, where every feature map corresponds to a convolution result for a shared filter weight.

To understand better the model[71], we can see that the first convolutional layer is organized by six feature maps computed by six shared weights named receptive fields on the input layer. In this case, the first receptive field dimension is a connected window formed by 5x5 pixels. Each unit has 25 inputs, and therefore 25

trainable coefficients and a trainable bias. The receipt fields are moved on the input with overlapping. The feature maps in a layer use different sets of weights and bias. Each unit in the feature map is applying a non-linear activation function concerning the result of the linear combination between the input part and its receptive field. The operation is equivalent to convolution, followed by an additive bias and a squashing function, which is the motivation for the model named "convolutional network". An interesting property of the convolutional layers is that if the input is shifted, the feature map output will be shifted by the same amount, but it will be left unchanged otherwise. This property is the base of the robustness of convolutional networks having distortions of the input. The second hidden layer is a sub-sampling layer. This layer comprises six feature maps, one for each feature map in the previous layer. The receptive field of each unit is an area input 2×2 in the previous layers corresponding feature map. Each unit computes the average of its four inputs, multiplies it by a trainable coefficient, adds a trainable bias, and passes the result through an activation function (sigmoid). Contiguous units have non-overlapping and contiguous receptive fields. Consequently, a sub-sampling layer reduces the previous layer to half the number of rows and columns. The trainable coefficient and bias control the effect of the sigmoid non-linearity [71]. The "transformation layer" is composed of a convolutional layer followed by a sub-sampling layer. So, the model is formed by two "transformation layer". The second "transformation layer" has a different number of parameters but it is structured in the same way. The approach is the same, so that, the model is scalable for each input data. In the end, the computed features are the input for a fully-connected classifier composed of two hidden layers and one output layer. This model is formed by a lot of parameters, so the challenging problem was to train all of them and find the optimal ones. The solution proposes to use the back-propagation method [102]. It is a solution for the general problem of minimizing a function for a set of parameters. Using gradient-based learning it is generally much easier to minimize a reasonably smooth, continuous function than a discrete (combinatorial) function [71]. Convolutional Neural Networks eliminate the need for hand-crafted feature extractors. This model was innovative, however, its application to more challenging problems was limited by the computation resources available at that time. New advances in computing capabilities due to the wide introduction on Graphical Processor Units (GPUs) contributed to a renewed interest for Convolutional Neural Networks in recent years.

Alexnet on Imagenet

The Alexnet [65] model uses two GPUs in a parallel way, sharing feature maps. That was the first approach and destroyed the competition with other models based on hand-crafted feature extractors.

Figure 2.2: Some example from Imagenet

The entire net is based on the state-of-the-art layer in deep learning techniques, using the Convolutional Neural Network as an "optimized" feature extractor. The architecture of this network is summarized in Figure 2.3. In this model [65], there are several transformation layers based on convolutional and pooling transformation, newer learning mechanisms as activation function, normalization, overlapping pooling and a trick to reduce overfitting.

A novel non-linear activation function, inspired by the human brain, the closest activation function to brain activation, is the Rectifier Linear function [88]. The results of this function on other Neural Networks has proved the great ability to limit the overfitting and improve the learning. One of the best characteristics demonstrated by this function is the sparse representation with true zero as an output value. This representation is sparse and robust to small input changes, with efficient variable-size representation and it is highly linear separable from the computed features.

A Local Response Normalization [65] defines a local normalization scheme. It

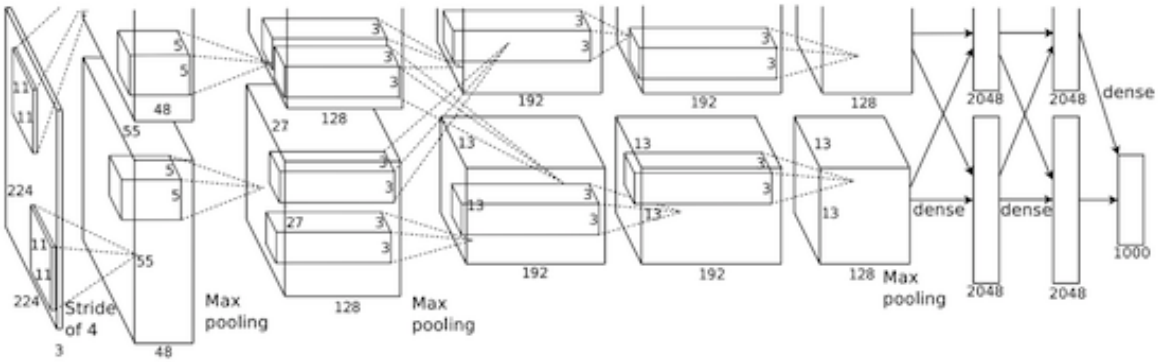


Figure 2.3: The Alexnet architecture adapted from [65].

works on the previous layer output in this way:

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

the response-normalized activity $b^i_{x,y}$ computes the sum over n "adjacent" kernel maps at the same spatial position, and N is the total number of kernels in the layer. The constants k, n, α, and β are hyper-parameters whose values are determined using a validation set ($k = 2, n = 5, \alpha = 10^{-4}, \beta = 0.75$). This normalization is applied after applying the ReLU nonlinearity function.

Overlapping pooling is an operation to summarize the outputs of neighboring groups of neurons in the same kernel map. Traditionally, the neighborhoods summarized by adjacent pooling units that do not overlap (e.g., [58, 71]). In the traditional models, the local pooling has the stride equal to the pooling window size. The size of the pooling kernel is 3 with a step size of 2.

The Alexnet architecture [65] has a deep structure with more layers than the past models. In this way, more parameters used to perform a better regression with respect to the objective function. So, this network maximizes the multinomial logistic regression objective, which is equivalent to maximizing the average across training cases of the log-probability of the correct label under the prediction distribution. This neural network architecture has 60 million parameters. With so many parameters, the overfitting problem will likely destroy this approach. They used two techniques to resolve this issue, data augmentation, and dropout.

- Data augmentation: A simple way to reduce the over-fitting during a training pass on image data is to artificially enlarge the dataset adding new "similar" images. The data augmentation consists of generating image translations and horizontal reflections. The way to increase the dataset is to extract random 224×224 patches (and their horizontal reflections) from the training image

256×256. This increases the size of the training set but maintaining the dataset input variance.

- Droupout: this technique consists of setting to zero the output of each hidden neuron with probability 0.5. The neurons which are "dropped out" in this way do not contribute to the forward pass and do not participate in backpropagation. So every time an input is presented, the neural network samples a different architecture, but all these architectures share weights [56]. During testing, they use all the neurons but they multiply their outputs by 0.5 to approximate the predictive distribution produced by the dropout networks.

This model [65] has been the milestone for Object Recognition task using Convolutional Neural Networks. Several works followed this approach, but with other improvements. The learning is the base of the result, they trained the model using stochastic gradient descent with a batch size of 128 examples, momentum of 0.9, and weight decay of 0.0005. The authors [65] found that a small amount of weight decay is important for learning the model. In this case, the weight decay is not merely a regularizer but it is used to reduce the training error [65]. The update rule for weight w was:

$$v_{i+1} := 0.9 \cdot v_i - 0,0005 \cdot \epsilon \cdot w_i - \epsilon \langle \frac{\partial L}{\partial w} \mid_{w_i} \rangle_{D_i}$$

$$w_{i+1} := w_{i+1} + v_{i+1}$$

where i is the iteration index, v is the momentum variable, ϵ is the learning rate and $\langle \frac{\partial L}{\partial w} \mid_{w_i} \rangle_{D_i}$ is the average over the i_{th} batch D_i of the derivative of the objective with respect to w, evaluated at w_i. The authors [65] initialize the weight in each layer from a zero-mean Gaussian distribution with a standard deviation of 0.01. The authors [65] evaluated the bias initialization with different values for each layer. They used an equal learning rate for all layers, which they adjusted manually throughout the training. The authors [65] trained the network for roughly 90 epochs through the training set of 1.2 million images, which took five to six days on two NVIDIA GTX 580 3GB GPUs.

Network in Network

The Alexnet model [65] consists of alternatively stacked convolutional layers and spatial pooling layers. The convolutional layers generate feature maps by linear convolutional filters followed by nonlinear activation functions (rectifier, sigmoid, tanh, etc.). In this way, we could obtain good discriminant features, but we know that the latent concepts which should to be learned are not always linearly separable. A CNN computes several convolutional transformations followed by a non-linear transformation, this linear convolution is sufficient for abstraction when the instances of the

latent concepts are linearly separable. Working with several filters for each convolutional layer could help us to cover all variations on the latent concepts. However, having too many filters for a single concept, that imposes extra work on the next layer, so it needs to consider all combinations of variations from the previous layer [48]. A good idea to reduce the number of feature map is proposed in the Maxout Network [49] where affine feature maps are the direct results from linear convolution without applying the activation function. The idea proposed by Lin et al. [76] is to develop a general function approximator when the distributions of the latent concepts are more complex. Network in Network [76] (NIN) can compute more discriminant features obtaining more linearly separable concepts only adding an MLP trainable approximator function. MLP works with the backpropagation algorithm becoming the best option for this task. It is possible to see this part as a micro-network integrated into a classic CNN structure to obtain a better abstraction for all levels of features. In this general prospective, the authors [76] propose to add two MLP layers after the classical convolutional transform. The author [76] defines this computational part as "mlpconv" layer, it is depicted in Figure 2.4 where the transformation is computed on the input channels. To obtain this transformation layer

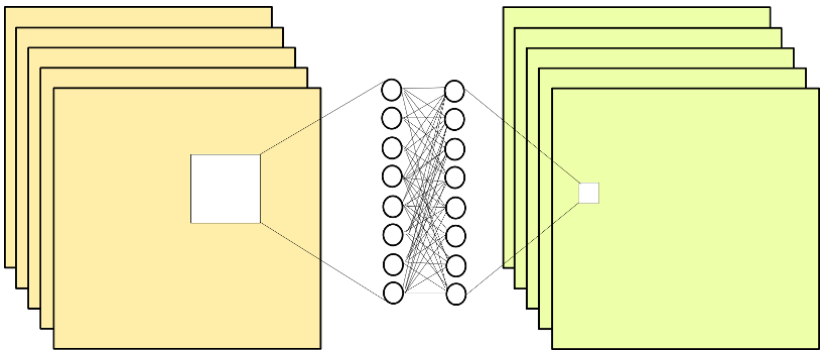


Figure 2.4: Mlpconv transformation layer.

it is possible to use only the classic convolutional layer followed by two convolutional layer with a 1 by 1 kernel. It is a Fully Convolutional Network, it computes only convolutional and pooling transformation. Instead, to have a fully connected layer after the computed features, the author [76] propose a strategy called global average pooling to replace the traditional fully connected layers on CNN. The idea is to generate one feature map for each corresponding concept of the classification task in the last mlpconv layer. Instead of adding fully connected layers on top of the feature maps, they compute the average of each feature map, and the resulting vector is fed directly into the softmax layer. In the end, the structure of NIN [76] is a stack of mlpconv layers, on top of which we fined the global average pooling and the objective cost layer.

VGG Network

After the first real improvement with AlexNet [65], other researchers developed novel models using a similar approach. With the born of several development frameworks, the research has gained more helps to develop new models. The first interesting framework to implement Neural Network models is Theano [10, 14]. After that, the second famous framework is Caffe [59] based on DeCaf [34] milestone. The developed frameworks helped the researcher to create a quickly novel model obtained good results.

An example is a research made by Simonyan et al. [111] following the same principles inspired by Cirasan et al.[26] and Krizhevsky et al. [65]. The authors [111] implemented a very deep Convolutional Neural Network which obtained good results in object recognition. The architecture has an input size of 224×224 pixels. The only preprocessing made on the input is subtracting the mean RGB value, computed on the training set, from each pixel. The classification is computed by a stack of convolutional layers, where the receipt fields are very small: 3×3 and 1×1.

- The receipt field 3×3 is the smallest window to capture the notion of left-/right, up/down and center. The convolution stride is fixed to 1 pixel; the spatial padding used 1 pixel for 3×3 convolutional layers.

- To use 1×1 convolution filters in some configuration, which can be seen as a linear transformation of the input channels (followed by non-linearity). The convolution stride is fixed to 1 pixel without spatial padding. The idea to use 1×1 con layers is the way to increase the non-linearity of the decision function without affecting the receptive fields of the convolutional layers [111]. It works as a transformation into the space of the same dimensionality for the input channels to obtain more discriminant features and an additional non-linearity using the rectification function. This approach is used in the "Network in Network" architecture [76].

Spatial pooling is computed to sub-sample the feature maps reducing the parameter dimension. Max-pooling is performed over a 2×2 window, with stride 2. The number of channels used in the whole net starts with 64 for the first convolutional layer increasing by a factor of 2 after each max-pooling layer until 512 a the last convolutional layer. In effect, this model has a different configuration and it depends about the depth of the net, we can find a small configuration with 11 weight layers to a large configuration with 19 weight layers in the network. A stack of convolutional layers is followed by three Fully-Connected (FC) layers: the first two have 4096 channels each, the third performs 1000-way classification and thus contains 1000 channels (one for each class). The final layer is the soft-max layer. This model [111] is quite different from the previously developed models. Rather than

using relatively large receptive fields in the first convolutional layers (e.g. 11×11 with stride 4 in (Krizhevsky et al., [65]), or 7×7 with stride 2 in (Zeiler & Fergus [127]), the authors [111] use very small 3×3 receptive fields throughout the whole net, which are convolved with the input at every pixel (with stride 1). The reason is that the convolutional layers larger than 3×3 compute a transformation could be obtained with a sequence of 3×3 convolution with no sub-sampling in between. It is possible to obtain a convolutional layer 5×5 with a sequence of two 3×3 convolutional layer, or a convolutional layer with a sequence of three 3×3 convolutional layer [111]. In this way, the transformation could incorporate several non-linear rectification layers instead of a single one, which makes the decision function more discriminative. Another advantage is that the model has less parameter to train, the authors [111] explain that, with a convolutional layer with a receipt field 5 x 5 we have 25 x C parameters (channels) to train with respect to have with a receipt field 3×3 only 9 x C + 9 x C = 18 x C parameters. The training pass is made using the multinomial logistic regression objective with mini-batch gradient descent on back-propagation with momentum proposed by LeCun et al. [71]. In this case, the batch size was set to 256, momentum to 0.9. The training was regularized by weight decay (the L_2 penalty multiplier set to $5 \cdot 10^{-4}$) and dropout regularization for the first two fully-connected layers (dropout ratio set to 0.5). The learning rate was initially set to 10^{-2} , and then decreased by a factor of 10 when the validation set accuracy stopped improving. One problem to solve when we want to train a deep convolutional net is the weights initialization because a bad initialization can stall learning due to the instability of gradient in deep nets [111]. The authors train before a less depth model then using this pre-trained weights to initialize the deepest model. Exist another way to initialize the weights with a random procedure using the prior information from the model structure Glorot & Bengio [45]. The data augmentation is similar to the previous model Alexnet [65], but in this case, scaling two times concerning the shorter dimensions and achieving better results. This model is implemented with a branch from the publicly available C++ Caffe toolbox [59]. The Caffe [59] framework allows to perform training and evaluation on multiple GPUs installed in a single system, as well as train and evaluate on full-size images or multiple scales. Multi-GPU training exploits data parallelism and is carried out by splitting each batch of training images into several GPU batches, processed in parallel on each GPU. They trained the model using a system equipped with four NVIDIA Titan Black GPUs and took 2–3 weeks of learning [111].

Googlenet

The most straightforward way of improving the performance of deep neural networks is by increasing their size. This is the baseline for the novel convolutional neural network named Inception [113]. But not only this feature is only specific

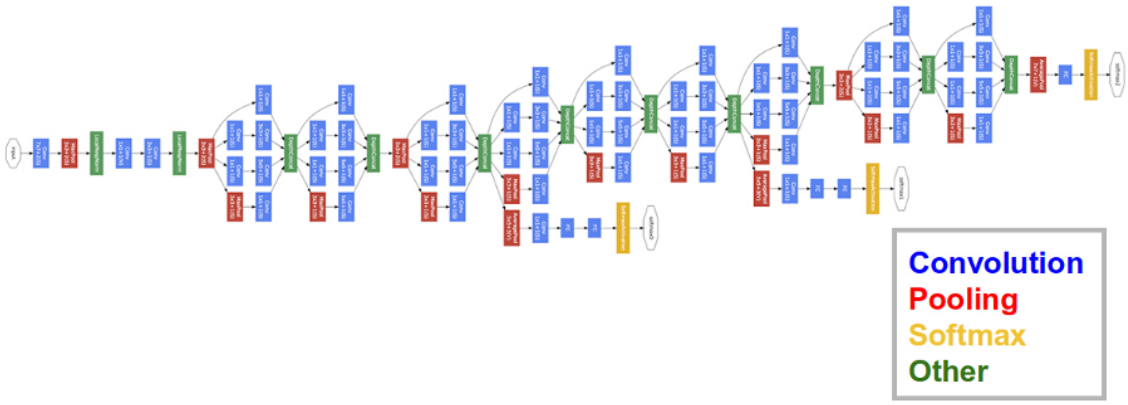


Figure 2.5: The Googlenet architecture.

for this net. During the evolution of the convolutional models, larger size models have a larger number of parameters being more prone to over-fitting, especially if the number of labeled examples in the training set is limited. Another important characteristic is that increasing the network size is dramatically increased the use of computational resources. The Inception [113] architecture is a sophisticated network topology construction algorithm that tries to approximate a sparse structure for vision networks and covering the hypothesized outcome by dense, readily available components [7]. The Inception [113] architecture is the baseline for the novel model named GoogLeNet. It is an homage to Yann LeCuns pioneering LeNet5 [71]. This model is a deeper and wider Inception network where all the convolutions use rectified linear activations. The network is 22 layers deep counting only layers with parameters (or 27 layers counting also pooling layers). The overall number of layers (independent building blocks) used for the construction of the network is about 100. In Figure 2.5 is depicted as the structure of the deep and large model based on Inception idea previously defined.

ResidualNets

Make deeper networks it is hard because when the network depth increase, accuracy gets saturated and then degrades rapidly. The notorious problem of vanishing/exploding gradients is one problem related to deep neural networks [13].

Comparing a shallower network and its deeper network version adding more layers (identity mapping layers) onto it, we see as current SGD solvers are not able to train deeper networks that produce better performance than shallower version [54].

In [54] the authors address the degradation problem by introducing a deep residual learning framework. Considering $H(x)$ as an underlying mapping to be fit by a few stacked layers, it is possible to say that multiple nonlinear layers can asymptot-

ically approximate complicated functions (this hypothesis is still an open question). In this way, assuming that input and output have the same dimensions, also the residual function $H(x) - x$ can be asymptotically approximated with few stacked layers. Defining $F(x) := H(x) - x$ the residual function, it is possible to have the original function as $F(x) + x$ motivated by the counter-intuitive phenomena about the degradation problem. With the residual learning reformulation, if identity mappings are optimal, the solvers may simply drive the weights of the multiple nonlinear layers toward zero to approach identity mappings [54]. Always in [54], the experiments confirm as the identity mapping is reasonable preconditioning to improve results in deeper neural networks.

In [54], the authors propose to define a residual block composed by few stacked layers defined as $y = F(x, W_i) + x$ where x and y are the input and the output vectors of the layers and $F(x, W_i)$ is the residual mapping to be learned. Defining a residual block composed by two convolutional layers as $F = W_2\sigma(W_1 x) + x$ where σ is RELU activation function, the operation $F + x$ is performed by a shortcut connection and element-wise addition. The shortcut does not introduce parameters or complexity for the training phase.

The experiments in [54] prof as the residual block in very deep neural networks improve results in object recognition and detection task in different image datasets.

2.4 Object detection

In these last years, many solutions have been proposed to solve the object detection which is based on Convolutional Neural Networks. In this section, we show some state of the art deep networks from object detection.

Region Based Approaches

In this last years, many state-of-the-art CNN-based deep learning for object detection has been proposed to classify and localize objects on input images which are based on a pipeline composed by region proposal followed by the detection phase.

Region-CNN (R-CNN) proposed by [44] uses selective-search ([116]) to detect 2000 object proposals, for each bounding box the feature extraction is done through pre-trained CNN architecture. In the end, the feature vector is fed into an SVM to classify the presence of the object. To predict the object location inside region proposals, the model can compute four values which are offset values to increase the precision of the bounding box. This solution is very slow and it is dependent on an external algorithm that cannot be trained with the whole system.

Fast R-CNN also proposed by [43] is an improved version of the previous architecture following a similar idea. In Fast R-CNN the whole input image is fed to the

CNN model generating convolutional feature maps and it is possible to identify the region proposed by a predefined search method (selective search, . . .) on computed maps. On squared features, the RoI pooling layer can compute the most discriminant features to fed into classification and regression layers to classify the object category and detect the bounding box area inside the proposed region.

Faster R-CNN proposed by [98] is an improved version which follows some concepts. In previous versions the bottleneck is the region proposal phase which is separated into the whole architecture, it is a very slow and time-consuming process affecting the performance of the network. For Faster R-CNN region proposal phase is substituted by Region Proposal Network (RPN). The whole input image is fed into CNN model (VGG, ResNet, . . .) to compute feature maps, then RPN uses a sliding window approach to compute each location over the feature maps. For each location, nine anchor boxes are used as three scales (128, 256 and 512) and three aspect ratios (1:1, 1:2, 2:1) for generating region proposals. The region proposal network is trained with Faster R-CNN sharing the same CNN backbone feature maps. In this way, RPN generates region proposal which is used by ROI pooling layer to compute the most discriminant features. Then, following Fast R-CNN architecture, the model computes classification score and bounding box area to detect objects. Faster R-CNN is trained with a unique loss function that combines RPN and detection networks. [89] proposes non-maximum suppression (NMS) which is used to reduce the number of detected objects according to classification score and overlapping areas.

Feature pyramid networks

Detecting objects at different scales is a challenging task for an object detection system. One approach could be considered the same input image at different scale computing the result for each scale. Another proposed approach named Feature Pyramid Network (FPN) in [77] which creates a pyramid of feature and use them for object detection. As said by [77], Feature Pyramid Network is a feature extractor designed for such the pyramid concept with accuracy and speed in mind. It replaces the feature extractor of detectors like Faster R-CNN [98] and generates multiple feature map layers (multi-scale feature maps) with better quality information than the regular feature pyramid for object detection.

Single Shot Detector

Differently to region-based networks that need two shots for object detection task, Single Shot Detector (SSD) [80] has been proposed to take one single shot to detect multiple objects within the image. SSD is much faster compared with two-shot RPN-based approaches. In SSD, a pre-trained CNN has been used to compute feature maps which are fed into a 3×3 convolutional layer to compute k bounding box for

each location. These k bounding boxes have different sizes and aspect ratios. Then, for each bounding box, the network computes classification scores and four offsets relative to the original default bounding box shape. A unique loss function is used to train the multi-box detector network.

Symbol detection in floor plan images

Recovering the symbol information from a rasterized floorplan image is a hard task. One solution [79] has been proposed to solve this task where the authors want to transform the rasterized floorplan image into an intermediate floorplan representation to describe the whole image. The authors propose a deep network to extract low-level geometric and semantic information into a set of junctions that are combined by integer programming solution. Moreover, the authors have manually annotated 870 floorplan images for training as well as for quantitative evaluation.

2.5 Semantic segmentation

We have seen several models for classification implemented using Convolutional Neural Networks with various structures. Now we will see how the learned features figure out the concept and how we could use this information to localize the object on the input image.

Deep inside convolutional networks

Almost all the previous ConvNets models are trained on the large-scale ImageNet challenge dataset and it can predict the class of the input image. This kind of architecture computes a transformation from an input image to a class score after several layer transformations. Every layer computes a transformation on the previous output channels. The layers are convolutional, pooling and fully connected units which compute a classification score concerning the learned concepts. Computing the layers transformation we obtain feature maps with several channels for each layer which are spatial representations from the input image. Simonyan et al. [110] present a method to compute a class saliency map, specific to a given image and class. The intuition is to separate the foreground from background pixels using the backpropagation algorithm. Given an image I_0, a class c and a ConvNet model which defines a class score function $S_c(I)$, we could to rank the pixels of I_0 based on their influence on the score $S_c(I_0)$. Using a deep ConvNet, the class score $S_c(I)$ is a highly non-linear function of I, so we can approximate $S_c(I)$ with a linear function in the neighborhood of I_0:

$$S_c(I) \approx w_T I + b$$

Figure 2.6: Saliency map computed over some input image extracted from Imagenet adopted from [110].

where w is the derivative of S_c for the image I at the point (image) I_0:

$$w = \frac{\partial S_c}{\partial I}$$

The more intuitive description about the derivative of the class score with respect to an input image is to consider the area more representative for each layer transformation and combine all these information together. We consider the model as a stack of layer computation, the gradient $\frac{\partial f_i}{\partial x}$ of the model $f(x)$ with respect to input image x could be define:

$$\frac{\partial f_i}{\partial x} = \frac{\partial f_i}{\partial g^{(n-1)}} \frac{\partial g^{(n-1)}}{\partial g^{(n-2)}} \cdots \frac{\partial g^{(2)}}{\partial g^{(1)}} = \frac{\partial g^{(n)}}{\partial x^{(n)}} \frac{\partial g^{(n-1)}}{\partial x^{(n-1)}} \cdots \frac{\partial g^{(1)}}{\partial x}$$

In this way we could obtain the pixel area where the extracted features classify the object in the input. In Figure 2.6 are presented some result about the saliency map computed on winning class with respect to the input image.
Another point of view of this approach is proposed by Simon et al. [109] where they define a "part detector discovery" (PDD) based on analyzing the Gradient Maps of the network outputs.

Fully convolutional networks

We have seen several models that work with receipt fields for convolutional transformation. These models compute several non-linear transformations to obtain a classification score. Using a general point of view, we could say that each layer of data in a ConvNet is a three-dimensional array of size $d \times h \times w$, where h and w are spatial dimensions, and d is the feature or channel dimension. ConvNet models are invariant to the translation. Their basic components (convolution, pooling,

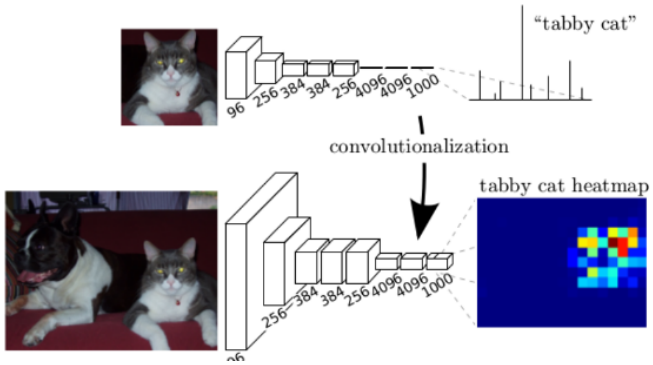


Figure 2.7: From ConvNet to fully ConvNet adopted by [81].

and activation functions) operate on local input regions and depend only on relative spatial coordinates [81]. Every transformation is spatially related so each part of input corresponds a part of the output on feature map In this way, the basic components are independent with respect to the input size but depend only by kernel sizes. While a general deep net computes a general nonlinear function, a net with only spatial layers computes nonlinear filtering over the input. These models using only spatial transformation could be called deep filters or fully convolutional networks. A fully convolutional neural network naturally operates on an input of any size and produces an output of corresponding (possibly resampled) spatial dimensions. In this way, all transformations are spatial transformation during the forward and backward pass. Considering the entire image we could obtain a better result than computing patch-by-patch representation. We have seen typical recognition nets that take fixed-sized inputs and produce non-spatial outputs. We know full convolutional nets are more adaptive to variable input dimensions. Every "classic" model could be transformed in a fully convolutional net, we only need to reshape the fully connected weights layers into a convolutional transformation so it could be spatially dimension independent. We can figure in Figure 2.7 how the model transformation adapts the weights for the classification task into weights for spatial representation. These models compute a spatial output and naturally adapted to dense problems like semantic segmentation. Every classic convolutional network can be transformed in a fully ConvNet which computes a spatial big filter transformation. An example could be Alexnet [65], after the previous transformation it maps an input 224×224 in a final feature map of 7×7 units. In that work [81], they proposed a model to make a semantic segmentation on images using a fully ConvNet model considering the entire pixel ground truth. The fast way to connect the resulted feature map to a dense pixel representation of the ground truth is to reshape computed features with interpolation. For instance, simple bilinear inter-

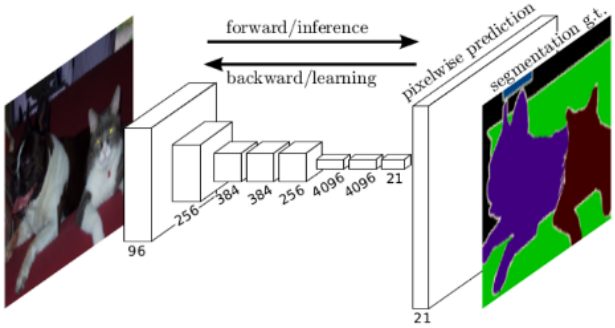


Figure 2.8: Fully convolutional neural network in action adapted by [81].

polation computes each output y_{ij} from the nearest four inputs by a linear map that depends only on the relative positions of the input and output cells. An interpolation is an upsampling with a factor f, it could be seen as a convolutional operator with stride $\frac{1}{f}$ named *backward convolution* (or *deconvolution*) with an output stride of f. This kind of upsampling is performed in-network for end-to-end learning by back-propagation from the pixel-wise loss. This approach could be compared with patch wise training because each batch consists of all the receptive fields of the units below the loss for an image (or collection of images). While this is more efficient than a uniform sampling of patches, it reduces the number of possible batches. We could obtain the random patch selection on the input using DropConnect mask [123] between the output feature map and the loss. In the end, they cast a pre-trained VGG [111] classifiers into FCNs to predict a dense classification with in-network upsampling and a pixel-wise loss. They train for segmentation by fine-tuning. An example of the entire architecture is depicted in the Figure 2.8

U-Net

Several architectures have been proposed to address the semantic segmentation. One model that gained attention in biomedical image segmentation is the *U-net* [101]. In thesis, we propose a neural network that is strongly inspired by the *U-net* model to solve object segmentation in a couple of document image understanding tasks. By inspecting the architecture in Figure 2.9 we can notice the U-shaped model where the first part consists of a contracting path and the second consists of an expansive path.

The contracting path consists of many encoding operations composed by convolutional operators with kernel 3×3, stride 1, and max-pooling operator with kernel 2×2 stride 2, respectively. In this way, the model is able to learn a data representation based on many local transformations computed by sequential convolution and pooling operations. In particular, for each transformation layer, we have two con-

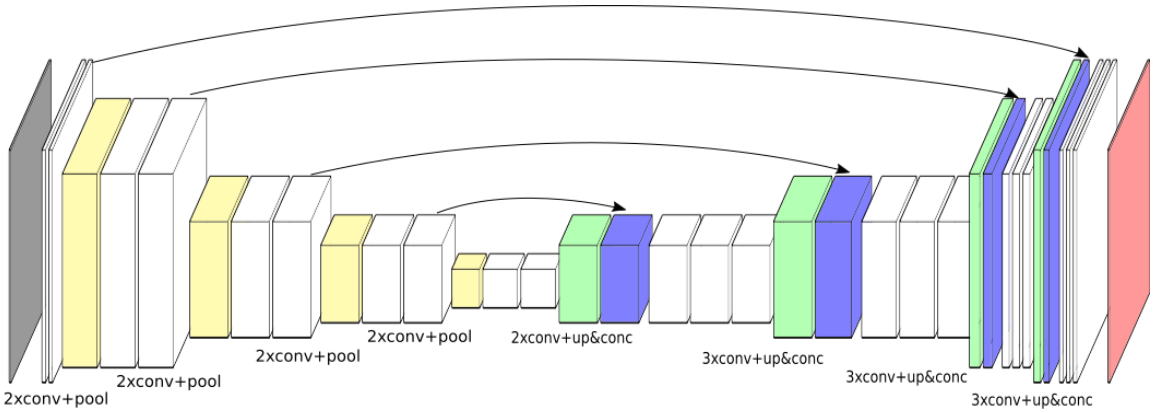


Figure 2.9: The model architecture. Different transformations are depicted in different colors. The Input Layer is identified in gray, Convolutional Layers in white, Max Pooling in yellow, green for Upsampling Layers, blue for Combination Layers and red for the Softmax.

volution operation followed by a pooling operation. The number of filters for each transformation layer is variable and we adapted these values to our problem. In particular, in the first layer, we have 32, in the second 64, in the third 128, in the forth 256, and in the last 512 filters.

The expansive path consists of several decoding operations composed of upsampling and convolutional operators. Having a look at Figure 2.9, for each decoding step the features are concatenated with the computed feature maps from the contracting path (with the same shape). Still, in the same decoding layer, two convolutional operations with kernel 3×3 and stride 1 are applied to the previously computed features. The expansive path proposes the same number of filters for each decoding layer, but in reverse order with respect to the contracting path. All the convolutional operators use Rectified Linear Units (ReLUs) as an activation function. In the final layer one single 1×1 convolutional linear operator is used to map the last features into the number of desired output channels.

We will see in Section 6.3 how we use this network for historical handwritten document segmentation.

Instance segmentation

As we have seen in this brief introduction, the object recognition task is related to classify object category inside a given input image, instead, for object detection, we intend to provide not only the object category but also indicate the position on the image.

Instance segmentation is a task related to the previous which has obtained a lot of interest from the research community. In this task, we want to detect each object

identifying their boundaries at the detailed pixel level.

Mask R-CNN [52] is a region-based deep network architecture for instance segmentation task. It follows a similar approach to Faster R-CNN [98] but locating each pixel of every object in the image instead of the bounding boxes. It is based on Faster R-CNN with Feature Pyramid Network [77], having a multi-scale output features, for each output layer we find a fully convolutional network to produce a mask which represents the detected object silhouette. We use this network for object detection in floorplan image at Section 4.1.

2.6 Variational Autoencoder

In machine learning, we have two families of learning method: discriminative and generative algorithms. Mathematically, a discriminative algorithm map features (x) to predefined labels (y) with the formulation $p(y|x)$. Differently, generative algorithms attempt to predict features (x) given a certain label or category (y) with the formulation $p(x|y)$.

Differently from sparse [97], denoising [118] autoencoders, Variational Autoencoders are probabilistic generative models which are structured as directed graphical models and parameterized as neural networks [62]. Following [62], we assume that observed data are generated by some random process involving unobserved latent variables z. Considering X as set of samples $\{x^{(1)}, x^{(2)}, \ldots, x^{(n)}\}$, we can define a joint probability [16] of the model as $p(x,z) = p(x|z)p(z)$ where for each sample i, $z_i \sim p(z)$ and $x_i \sim p(x|z)$.

We know the true posterior density $p(z|x) = \frac{p(x,z)}{p(x)}$ and the marginal likelihood (evidence) $p(x) = \int p(x|z)p(z)dz$ are in many cases intractable [62]. Variational Bayesian (VB) inference is useful to approximate the true posterior distribution $p(z|x)$ with $q_\lambda(z|x)$ by family λ of densities over the latent variable [16]. In this way we want that our proposal distribution $q_\lambda(z|x)$ to be close to $p(z|x)$ which means minimizing their Kullback-Leibler (KL) divergence [66].

$$\mathbb{KL}(q_\lambda(z|x)||p(z|x)) = \mathbb{E}[\log q_\lambda(z \mid x)] - \mathbb{E}[\log p(x,z)] + \log p(x) \tag{2.1}$$

We see in Equation 2.1 that $\mathbb{KL}(q_\lambda(z|x) \,||\, p(z|x))$ is related to the computation of $\log p(x)$. Computing $p(x)$ is intractable as well as the previous KL-divergence [61]. Considering $q_\lambda(z|x)$, the evidence lower bound $L(\lambda)$ (ELBO) is useful for optimizing $\log p(x)$ [16]:

$$\log p(x) \geq L(\lambda) = \mathbb{E}[\log p(x,z)] - \mathbb{E}[\log q_\lambda(z \mid x)] \tag{2.2}$$

because one property of $L(\lambda)$ is to be the lower bound of the log probability of the observations.

Examining the $L(\lambda)$ gives intuitions about the optimal variational density. We rewrite the $L(\lambda)$ as a sum of the expected log likelihood of the data and the KL divergence between the prior $p(z)$ and previous $q_\lambda(z|x)$ [16]:

$$L(\lambda) = -\mathbb{KL}(q_\lambda(z|x)||p(z)) + \mathbb{E}_{q_\lambda(z|x)}[\log p(x \mid z)] \tag{2.3}$$

The family of proposal distributions $q_\lambda(z|x)$ has been chosen so that the lower bound $L(\lambda)$ is easy to differentiate and optimize. To compute and minimize the KL-divergence between the approximated and exact posteriors, we can maximize the evidence lower bound $L(\lambda)$ which is equivalent [16] (but computationally tractable).

Considering X as the set of samples, each x_i is related to its z latent variable, so it is possible to decompose $L(\lambda)$ into a sum where each term depends on a single point x_i having:

$$L(\lambda)_i = -\mathbb{KL}(q_\lambda(z|x_i)||p(z)) + \mathbb{E}_{q_\lambda(z|x_i)}[\log p(x_i|z)] \tag{2.4}$$

equivalent to previous Equation 2.3 but related to prior and likelihood terms [61].

Neural Network

Using neural networks it is possible to approximate the posterior $q_\lambda(z|x)$ and the likelihood $p(x|z)$ by learning some networks parameters ϕ and θ to maximize the evidence lower bound $L(\lambda)$ (Equation 2.4). In particular this can be done for both the encoder and the decoder as proposed in [61].

Encoder

We define an inference network to approximate posterior $q_\lambda(z|x)$ as $q_\phi(z|x,\lambda)$ (parameters ϕ related to Equation 2.1) which takes x input data and computes outputs parameters to model the chosen λ family distribution.

We will assume that the true posterior takes on approximate Gaussian form with an approximately diagonal covariance, so

$$\log q_\phi(z|x_i) = \log \mathcal{N}(z; \mu_i, \sigma^2 I) \tag{2.5}$$

where the μ mean and σ standard deviation (modeled as λ in the previous definition related to Equation 2.1) of the approximate posterior are outputs of the encoder network.

Let z be a continuous random variable and $z \sim q_\phi(z|x)$. It is possible to express the random variable z as a deterministic variable $g_\phi(\epsilon, x)$ where ϵ is an auxiliary variable with independent marginal $p(\epsilon)$ and $g_\phi(\cdot)$ is a trainable network with ϕ parameters.

Actually, having z as a normally-distributed variable with mean μ and standard deviation σ, we can sample z as $\mu + \sigma \odot \epsilon$, where $\epsilon \sim \mathcal{N}(0, 1)$ is a noise variable. In this way, it is possible to define a trainable network by backpropagation algorithm which can approximate the true posterior distribution.

Decoder

Then, we can define the generative model $p_\theta(z)p_\theta(x|z)$ as a generative network which takes the latent variable z and computes the data distribution $p_\theta(x|z)$ modeled by parameters θ. In the current case the decoding term $\log p_\theta(x_i|z_i)$ is modeling as a Bernoulli distribution related to the observed data.

Loss function

It is possible to train the network by maximizing the evidence lower bound $L(\lambda)$ in Equation (2.4). For each sample x_i, the loss function will be:

$$L_i(\phi, \theta) = -\mathbb{KL}(q_\phi(z|x_i)||p_\theta(z)) + \mathbb{E}_{z\sim q_\phi(z|x_i)}[\log p_\theta(x_i|z)] \tag{2.6}$$

where $z_i = g_\phi(\epsilon_i, x_i)$ and $\epsilon \sim p(\epsilon)$. Considering a mini-batch gradient descent algorithm, the loss function will bus the sum over all the mini-batch samples.

In Equation 2.6 the first term acts as a regularizer, the second term is the expected negative log-likelihood to encourage the decoder to learn data reconstruction.

2.7 Labeling systems

Deep learning architectures need tons of data for the training process. We need toolkits to facilitate annotating data. Following some tools proposed for this task.

Labelme [104] is a tool focused on scene annotation which also has web and mobile versions allowing collaborative work. This tool provides functionalities such as drawing polygons, querying images, and browsing the database. The user may label a new object by clicking control points along the object's boundary and finishes by clicking on the starting control point. One open-source version in Python has been released at the link.

A Graphical Annotation Tool [42] based on region-based has been proposed to build a hierarchical representation of images. Using Partition Trees the system can navigate through the image segments selecting the object components.

SmartAnnotator [125], an interactive system to facilitate annotating raw RGBD image combining incremental learning method to label prediction and 3D structure inference and refinements to help users in the annotation process.

In Section 4.1 we will show our interactive labeling system used to label objects in floor plan images for instance segmentation task.

2.8 Summary

In this chapter we have introduced some DAR application systems which are also addressed by deep learning methods. The major point to recall are as follows:

- Feature learning directly from raw data has obtained very good results on the object recognition. These CNN architectures have gained a lot of interest in many application fields. We will use some of these architectures in this research.
- CNN-based object detectors are complex architectures to classify and localize objects on an input image. We will see an application of one specific deep network in Section 4.1.
- Deep architectures for semantic segmentation task to classify pixels which belong to a semantic category. We will show how to use an FCN-based architecture for layout analysis at Section 6.3
- Variational Autoencoders are generative models that add a probabilistic approach to constraint the low-dimensional representation. We will use this approach to generate floor plan symbols at Section 4.5
- Labeling systems are useful toolkits for annotating data. These systems can help researchers to create labeled data for learning systems. We will show our annotation toolkits in Section 4.1.

Chapter 3

Validating Deep Learning models

Validating deep learning methods requires datasets useful for training and clear evaluation metrics to measure and compare the obtained results. In this chapter, we present some datasets used in this thesis. Then, we also explain the evaluation metrics used in the experiments.

3.1 Layout analysis datasets

In computer visions and also in document understanding there are several datasets for different applications used to validate machine learning solutions. Some of these datasets are public while others are private collections.

Tobacco

ImageNet is an image dataset [33], it contains millions of images according to the WordNet hierarchy totaling 3.2 million images. ImageNet contains general real-world images of categories like car, scooter, tiger, monkey, ship, web category, etc. It has been used to train models for object recognition and detection tasks.

The Tobacco dataset [74] shows several challenges including multiple fonts, poor quality and the presence of relevant information in handwritten annotations. Tobacco dataset consists of 5590 tax-form images and 3482 non-tax-form images organized in 10 classes, including Memo, E-mail, Resume, Letter, Report, Forms, Advertisement, Scientific, Note, Letter. It used to compare different approaches for genre classification. Images in categories such as Advertisement, Resume, Report, News exhibit a high variation in structure, and images of different genres may have a similar structure. This dataset has been used in [2] for genre classification taking into account only the document aspect by training a suitable neural network.

Figure 3.1: Some examples from Saintgall dataset

Saintgall

For page analysis task it is no easy to find public databases. HisDoc [39] is a scientific research project dedicated to the textual heritage, it aims to the development of a complete processing chain for analysis, recognition, indexation, and retrieval of historical documents. It presents some datasets for developing handwriting recognition and layout analysis systems.

One of them is SaintGall dataset presented in [40]. It contains a handwritten historical manuscript with hagiography *Vita sancti Galli* by Walafrid Strabin in the Latin language from the 9th century. The original manuscript is housed at the Abbey Library of Saint Gall, Switzerland. The manuscript has been most likely written by one single hand in Carolingian script with ink on parchment. Carolingian minuscules are predominant, but some upper script letters emphasize the structure of the text and some richly ornamented initials. The Saint Gall database includes 60 pages, 1410 text lines, and 11,597 words. Each page is written in a single column that contains 24 text lines. The database is freely downloadable and it is provided with layout descriptions in XML format. The document images in the original dataset have an average size of 3328 $\times$ 4992 pixels. The ground truth information is gathered from the Hisdoc Divadia site[1]. A couple of examples from the collection are shown in Figure 3.1.

Figure 3.2: Some examples from Esposalles dataset

Esposalles

In Archives of the Cathedral of Barcelona, it is possible to find the Marriage Licenses Books which are composed of 291 books with information of approximately 600,000 unions celebrated in 250 parishes between 1451 and 1905.

Each marriage license contains information about the husband's occupation, husband's and wife's former marital status, socioeconomic position signaled by the fee imposed on them, and in some cases, fathers' occupations, place of residence or geographical origin.

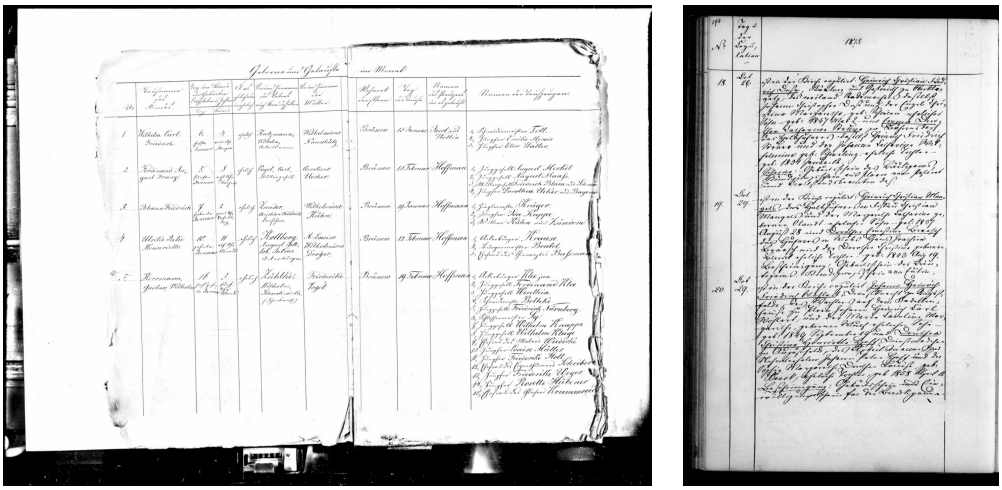
The original documents have been digitized at 300 dpi in true colors ($\approx 2750 \times 3940$ pixels). The set of books includes approximately 550,000 marriage licenses from 250 parishes [100].

From this collection, volume 208 has 593 pages where the first 200 pages have been labeled at pixel level [5]. In [5] the authors also selected 150 pages for training, 10 pages as the validation set and the remaining 40 for testing. A couple of examples from the collection are shown in Figure 3.2.

Brandenburg

Through our research collaboration with Ancestry, we have the chance to work with *Brandenburg* collection containing 78781 images and only the number of records on each page is available. We can see one example page from *Brandenburg* collection in Figure 3.3 (a). However, this collection contains pages with significantly different layouts, we need to group similar pages by using rule-based clustering technique explained in Section 4.2. After the selection, we obtain 4,956 pages with a more

[1]http://diuf.unifr.ch/main/hisdoc/divadia



(a) Example for counting task (b) *Branden-ancestry* example

Figure 3.3: Some examples from *Brandenburg* dataset.

homogeneous layout. This selected collection will be more useful to test our system for record counting tasks proposed at Chapter 7.

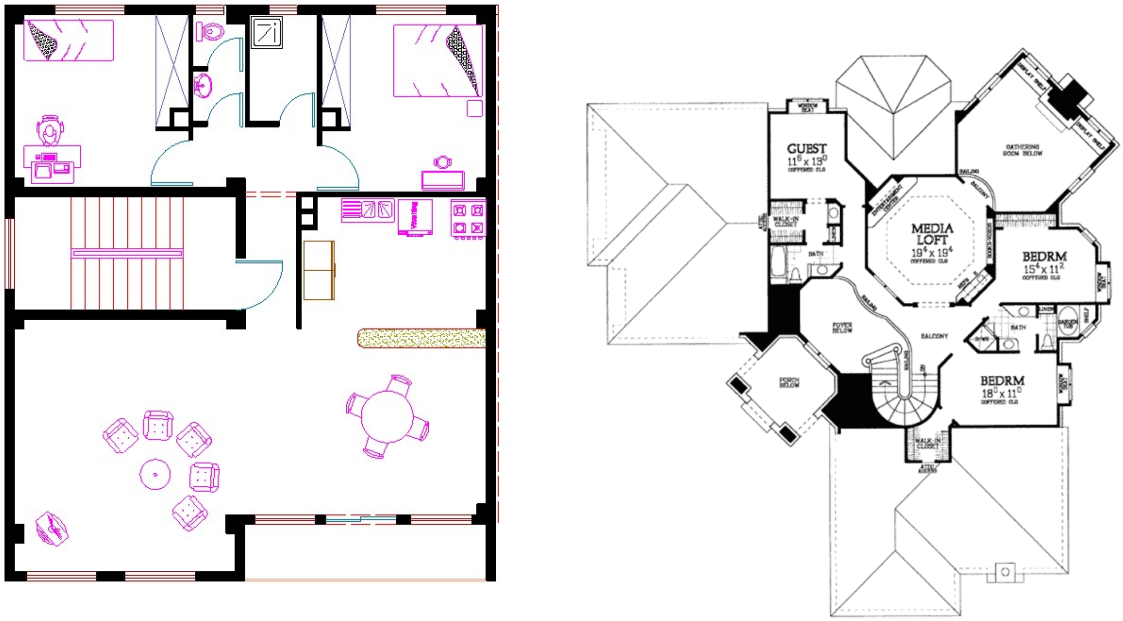
Textline detection

A subset of *Brandenburg* collection consists of 54 structured documents written in English by several writers. This subset, named *Branden-ancestry*, has been labeled at the text line level. The line ground truth has been produced using one tool specifically developed to manually segment the lines. We can see one example of this subcollection in Figure 3.3 (b). In Section 6.1 we will see how we use *Branden-ancestry* to evaluate our proposed solution for text line extraction.

3.2 Floor plan datasets

UAB-CVC

The CVC collection [31] comprises 122 scanned floor plan documents which are divided into four different subsets concerning their origin and style. It contains documents of different qualities, resolutions and modeling styles. In the ground truth, we can find structural symbols: rooms, walls, doors, windows, parking doors, and room separations. We have annotated also non-structural symbols (bed, toilet, bathtub, shower, sink) using the labeling system proposed at Section 4.1



(a) ISTA (b) Flo2Plan

Figure 3.4: Some examples from two floor plan datasets

ISTA

This is a collection of Middle Eastern floor plan images that come from one architecture company. Initially, this collection was named d2 as mentioned in [129]. This dataset is composed of 160 labeled images for the object detection task.

Since these design drawings come from one single architectural firm, the symbol contained in these images are more homogeneous than other collections. This collection is composed of 160 images, with an amount of 7788 labeled objects which are divided into 12 classes. in the next Section 3.2 we will see another floor plan image dataset which contains a larger intra-class variance. We show in Figure 3.4 one example from *ISTA* collection.

Flo2Plan

Collecting floorplan dataset is difficult because these drawings are covered by copyright. We developed a tool to search floor plan images from Google Image Search Engine by content description. Using this proposed tool we downloaded from several websites in different countries (Italy, Usa, France, Germany) a few thousand of images organized by name and content information. Unfortunately, almost half of the downloaded images were not good for our needs. We selected the best floor plan image manually.

Figure 3.5: Pre-processing phase: from input object to cleaned output object.

This collected dataset named *flo2plan* contains 452 heterogeneous floor plan images with different scales and RGB colors. Moreover, downloading from different websites, in this dataset we can find also blurry or watermarked images. Part of this collection comes from d1 dataset in [129]. In Figure 3.4 we show one example from *flo2plan* collection.

After the selection, we labeled the dataset at the pixel level using the interactive labeling system explained in Section 4.1. We created a labeled dataset with an amount of 11517 symbol instances divided into 13 object categories: door, sink, toilet, bathtub, shower, bidet, table, chair, couch, armchair, hotplate, night table, bed. For each object category, we have a large object-size variance.

Differently to ISTA collection (Section 3.2), in this dataset, we can find a large intra-class variance which makes the object detection task more difficult concerning other datasets.

To give more structural information to the system, we developed a tool to group objects by their relationship. We defined 4 macro categories: bathroom, bedroom, table-chair, kitchen. In this way, we can group objects into these categories noting object relationship into ground truth information.

Flo2Plan Symbols

The Flo2Plan and ISTA dataset contain several symbol instances that are very well annotated. Having the ground truth at the pixel level, it is possible to extract symbols automatically. The extracted symbols have different sizes and, often, they are

affected by noise. To normalize the data we perform a suitable pre-processing to reduce the aspect variability helping the model training.

To produce a dataset which less size variability, we rescale the input symbols to a fixed size transforming symbols to a square shape by padding the shortest dimension with background values, then we use bilinear interpolation to reshape the padded image into a fixed size (128×128). To remove the artifact, we use a Gaussian filter followed by a binarization operation (Otsu [94]). The results of these operations are shown in Figure 3.5, we can see one example for each category before and after the pre-processing phase.

Dataset	# floor plans	# beds	# armchairs	# toilet
ISTA	200	559	1366	346
flo2plan	454	413	456	1042

Table 3.1: Number of symbols for each dataset.

Since both datasets are labeled at the object level, it is possible to extract locations of symbols according to their coordinates and produce a collection of heterogeneous symbols.

The total number of objects is shown in Table 3.1, we can notice that the images in the *flo2plan* dataset are more affected by noise than those in ISTA dataset. Moreover, all the images in ISTA dataset come from one architectural firm, in this way all instances from each category are very similar to each other, instead, *flo2plan* has a lot of variations for object size, noise, and distortions.

3.3 Evaluation metrics

In this section, we present the measure used in the following application in document understanding research fields.

Precision, Recall and $F_1 score$

In information retrieval task the two most frequent measures used to evaluate the solution method effectiveness are Precision and Recall.
Precision (P) is the fraction of retrieved documents that are relevant.

$$Precision = \frac{|relevant\_items \cap retrieved\_items|}{|retrieved\_items|}$$

Recall (R) is the fraction of relevant document that are retrieved.

$$Recall = \frac{|relevant\_items \cap retrieved\_items|}{|relevant\_items|}$$

In binary classification, $F_1 score$ is a measure which considers the harmonic mean of precision and recall.

$$F_1 score = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

Accuracy, Mean Accuracy

Informally, accuracy is the fraction of predictions our model got right. Formally, accuracy has the following definition:

$$Accuracy = \frac{|relevant\_items|}{|items|}$$

Having a multiclass prediction task, we can compute the accuracy per class and then compute the average on the classes. This accuracy is named "mean accuracy" measure.

Jaccard Index

The Jaccard coefficient or Intersection over Union is defined to measure the similarity and diversity between two finite sample sets.

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

If A and B are empty set, $J(A,B) = 1$.

This coefficient also defined Intersection over Union measures the overlap between two sets. It is used in the case when we want to compare a detected rectangle area and ground truth rectangle area as $IoU = \frac{overlap\_area}{union\_area}$

Mean Average Precision

In Information Retrieval, Mean Average Precision (mAP) provides the measure of quality across recall levels [85].

$$mAP(Q) = \frac{1}{Q} \sum_{j=1}^{|Q|} \frac{1}{m_j} \sum_{k=1}^{m_i} Precision(R_{jk})$$

where relevant documents $q_j \in Q$ is $\{d_i, \ldots d_{m_j}\}$ and R_{jk} is the set of ranked retrieval results from top result until you get to document d_k.

This measure is also be used in Object Detection task in Pascal VOC competition [37].

Mean absolute error

Considering two continuous variables, Y and $\hat{Y}$, we want to measure the average distance between their values. Assume Y and $\hat{Y}$ are variables of predicted versus observed values from a regression trained model. Mean Absolute Error (MAE) is the average distance between each value respectively. The mean absolute error is given by:

$$MAE(Y, \hat{Y}) = \frac{\sum_i^n |y_i - \hat{y}_i|}{n}$$

3.4 Summary

In this chapter, we presented the datasets and metrics used in our research. The major point to recall are as follows:

- We saw some public datasets used for layout analysis which come from digital libraries.
- We proposed also a *Brandenburg*, a private collection of birth records which we will use in Chapter 5, in Section 4.2 and Section 4.3 for layout analysis tasks. A subset *Branden-ancestry* from this collection will be used in Section 6.1 for text line extraction.
- We produced a floor plan image dataset named *flo2plan* which can be used for object detection (in Section 4.1) for training and testing suitable deep network architectures. Moreover, we will see in Section 4.5 how to use the extracted symbols from *flo2plan* to train a generative model for symbol generation.

Chapter 4

Generating training data

In the last years, extensive use of deep learning techniques brought an improvement of results in a wide range of research areas, especially in computer vision. Deep learning architectures have a large set of free parameters that have to be learned in the training. To achieve the best trained models we need large datasets to learn these parameters. Often, the labeled data are limited, in some case we need to augment the training data in some way. In this chapter, we will show different approaches to produce training data.

As we have seen in Section 2.7, it is very important to have labeling systems able to help users during the annotation process. In particular, we propose one interactive labeling system to annotate graphical documents quickly. Creating labeled data, we will show how to train a deep network for object detection in floor plan images. We will compare our proposed deep learning approach to other classic methods for symbol recognition and symbol spotting.

In the historical document domain it is important to have large collections to address layout analysis task. If we have a collection with a large variance of layout structures becomes very hard to learn a model which could fit the data distribution, especially if the number of examples is not enough to cover all the aspects of the data distribution. We will explore a technique to select similar document images by using a rule-based selection method. This approach can select documents with a homogeneous layout from the *Brandenburg* dataset (Section 3.1). We will show in Chapter 5 a way to use the selected pages to train a system for record counting.

We will also show one toolkit to generate synthetic documents to simulate a structured document production process. Moreover, we will present a system for generative modeling that uses a model trained on a symbol images to generate new samples imitating the initial symbol dataset.

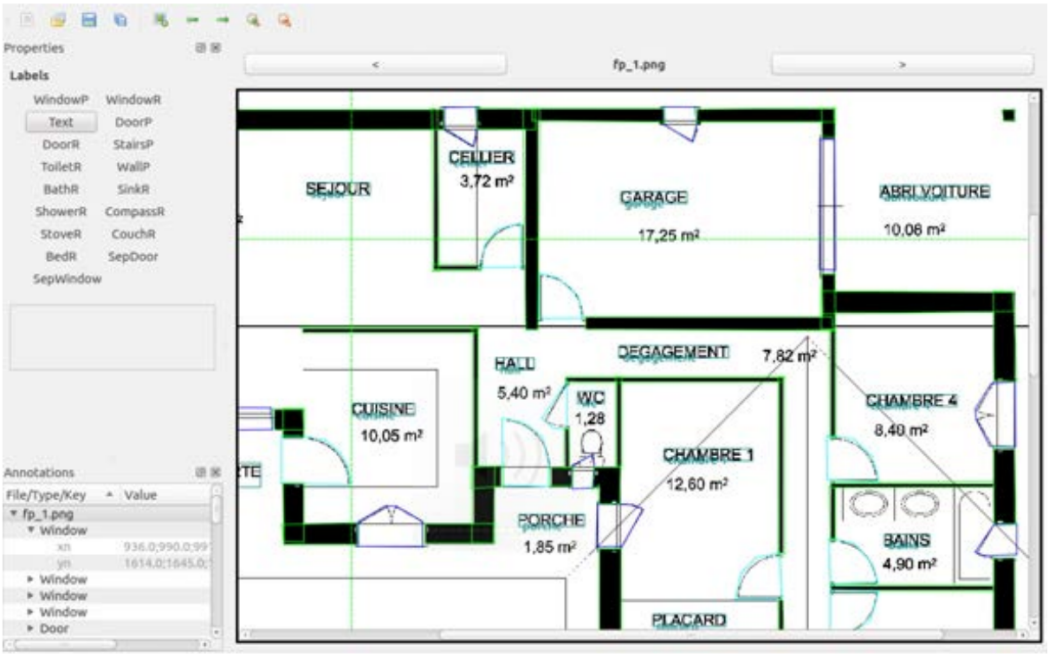


Figure 4.1: The floor plan editing tool

4.1 Labeling graphical documents for symbol detection

In computer vision, instance segmentation (Section 2.5) is one of the most labor-intensive annotation tasks because it requires pixel-level accuracy. Many systems have been created to solve this task, most of them are useful to label real scene images.

In this section we show two labeling systems used for two different purposes: SlothV2, and ActiveLabelMe. With the first tool, we can correct recognition errors made by a floor plan recognition system; the second is used to label a floorplan dataset for instance segmentation.

After a brief introduction of the floor plan recognition system proposed in [84] we show SlothV2 in action; then, the ActiveLabelMe tool can be used to label a floor plan image and to produce instance segmentation ground truth.

SlothV2

We have created a proof-of-concept editor for annotating floor plan images which gives everyone a chance to correct the recognition output from the floor plan recognition system summarized in the following sub-section. We decided to extend a pre-defined labeling system to describe images named Sloth[1]. Sloth is a tool to label

[1]https://github.com/cvhciKIT/sloth

image and video data for computer vision research, it is possible to define polygons or rectangular bounding boxes around the regions in input documents. We defined several semantic categories to label a floor plan image, in the end, the user can save the created items in a structured JSON file. The tool can read JSON files to initialize the annotations. In this use-case, the tool takes a JSON file produced by the proposed recognition system and the related input image. It displays the original image highlighting recognized graphic symbols such as text doors, walls, windows, rooms, etc. The user can delete or add new annotations to correct the recognized symbols updating the information in the initial JSON file. A screenshot of the tool is shown in Figure 4.1.

Floor plan recognition system

The system is a pipe-line that consists of a chain of image processing elements. Each layer defines different transformation functions implemented in Python with OpenCV, an open-source computer vision and machine learning software library.

The pipeline processing consists of nine different steps: *pre-processing*, clean the image removing thin lines, then, find connected components considering their aspect-ratio and covered area. The *text identification and recognition*, optical character recognition (OCR) is performed on the textual connected components to identify and recognize text labels. Followed by *structure object recognition*, the target is to classify the found graphical connected components like walls, doors, windows, and stairs. The connected components corresponding to walls are segmented and categorized as internal or external walls based on their average thickness. Sometimes cause the external walls found in the floor plan image may contain gaps, then, the *close external walls* step, we need to close walls with a gap to properly detect rooms. To *detect closet* (closet is a tall cupboard or wardrobe with a door, used for storage) we need to restore some removed line during the pre-processing step, using the recognized text labels ('closet','cupboard', 'wardrobe') it is possible to identify and to compute the right bounding box. In the *room identification* step, it is used a similar approach to [3] for recognizing rooms. It is often required to detect sub-areas in the open plan regions, it is time to do an open plan partitioning [83] generating candidate partitions for each sub-area in the open plan. *Object/furniture identification* is a parallel step where the target is to recognize furniture objects. It computes an adjacency graph between the found connected components and a symbol dictionary.

The result of the whole pipeline is *high-level description generation* of the text, walls, doors, windows, stairs, single-use rooms and open plan areas which are saved into JSON file. Using the proposed annotation system (SlothV2) we can label data and correct recognition errors made by the recognition system proposed before.

This system exports an accessible floor plan description in a JSON file having three formats: text description, tactile floor plan or GraVVITAS presentation. Based

on the formative study, an entire floor plan view and individual room views are generated in the desired format. The entire floor plan view contains information about doors, windows, walls, stairs, rooms, text labels and unknown elements in the floor plan. More details can be found in [84].

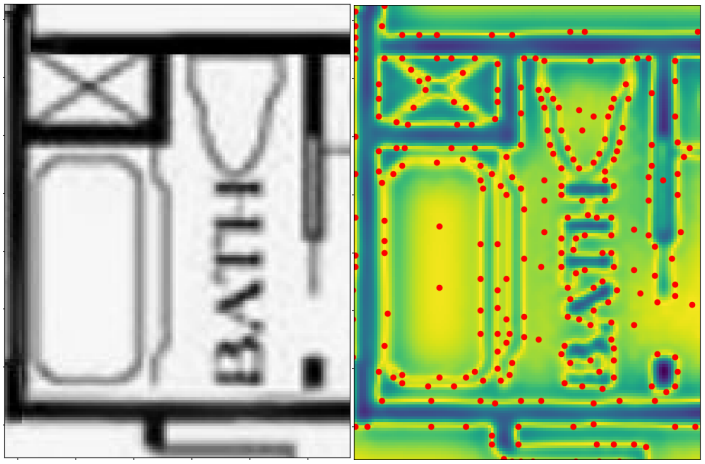
ActiveLabelMe

We propose an interactive labeling system that helps the user to label object instances at the pixel level. To apply this software on floor plan images, we define a set of tools to help the labeling process.

Considering floor plan images, the background is often uniform and the objects are usually small and close with each other. Having this kind of problem, we did not find any existing system able to help users in labeling tasks properly.

Using a pre-existing open-source software named "labelme" [104] it is possible to annotate images with polygons assigning one object category for each defined polygon. We decided to extend [120] by adding some useful features.

Initially, with the software proposed by [120], it was possible to define only polygons around symbols and to assign a label to defined symbols. To make the labeling job easier, we added a function allowing users to annotate objects with rectangle areas. In this way, doing only 2 mouse clicks we can define the left-top and right-bottom coordinates for the bounding box.



(a) Seleted area (b) Best Keypoints

Figure 4.2: How the interactive labeling system computes best key-points to detect object silhouette

Labeling many small objects from scratch is a tedious task, to simplify it we added a function to label objects semi-automatically using the output from a trained graphical symbol detector [129]. In this way, users can check and edit the detected

objects by using this tool. Labeling data in this active mode can reduce the time-required.

In architectural floor plan images, the background has more or less a uniform color, we want to define a technique to localize a symbol inside one selected area. We define this feature as implementing the following mechanism.

We detect the edges by the difference of gaussian (DoG) at different scales, then we define a fixed grid (stride by 3 pixels) on image and for each cell, we compute to maximum DoG value in the neighborhood. We call these values key-point. Selecting a rectangle area, we select the best key points inside the area, then we segment the instance mask computing the convex hull on the selected key points. After this computation, if we need to correct the computed mask, it is possible to add or remove key points to detect a better object silhouette.

In Figure 4.2 we show a selected area from the input image (a) finding several key points computed by a multi-scale difference of gaussian (DoG) technique (b). In Figure 4.2b we show in yellow the DoG output value which corresponds to the most probable area to find edges related to the object contour. On the edges area, we can find the best key points (red points) used to compute the object contour by the convex hull.

In Figure 4.3 we show how it is possible to edit the computed contour after the bounding box selection. In window (a) we can see the selected area from the input image, then it is possible to edit key points in the window (b) and check the obtained contour by the convex hull in the third window (c). Inside window (b) it is possible to see how Delaunay tessellation connects key-points during the editing phase.

We added another tool to the system to group labeled instances in a super-category. This tool allows the user to group objects related to their relationship. It is possible to select objects defining a rectangle area to group selected instances into a super-category.

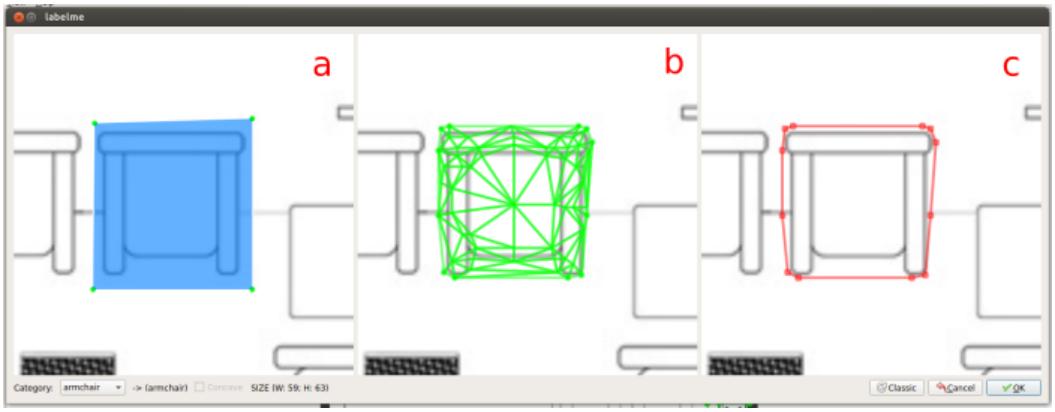


Figure 4.3: Editing tool to correct and improve the computed object contour

Object detection in Floorplan images

One of the most exciting topics for the graphics recognition community is *symbol recognition*. Many different approaches described in the literature [25, 27] work on logical diagrams, engineering drawings, maps, etc. In the last years, many other graphics recognition methods have been proposed.

One of the major problems in symbol recognition is to combine segmentation and recognition. It is important to understand the logical and semantic structures from graphics-rich documents: technical documentation, maps, schemas. The methods developed to understand graphics-rich documents have been proved to be of interest to the problem of indexing the information for efficient retrieval and browsing [114].

In [115] the authors define a way to localize possible symbols reducing the computational complexity without using full recognition methods. In this way, it is possible to define a task named *symbol spotting* as a kind of middle-line technique combining recognition and segmentation. The major difference between symbol recognition and symbol spotting is that in the first method we want to find the location and recognize every symbol in the document; the second method can be viewed as a kind of retrieval system where the user selects a symbol from the document defining a query by example.

There are many difficulties to locate graphical symbols in huge collections of documents, as the presence of distortions, occlusions and geometric transformations in graphical documents make the symbol recognition a challenging task. Although well-known hand-craft document descriptors perform quite good recognition, these techniques are time-consuming and only work for recognizing isolated shapes [103].

In [103] the authors propose a method to represent document regions as a composition of object primitives. Every described document region is associated with a code computed by a hashing function. Computing a connected component analysis, the system can describe all the object regions, subsequently, each component is polygonally approximated to take the chains of adjacent segments. To compute a code for each polyline it is possible to use their proposed hash function. Then, describing the query example with their proposed approach it is possible to detect similar symbols in the input image.

Another work [35] explains how to represent a document with a graph, detecting symbols by using a (sub)graph matching technique. A graph allows catching structural properties of the symbol primitives (especially various kinds of line drawings). They represent a document considering the detected critical points as nodes and joining the lines to nodes as edges. Then, a factorization step is useful to split the graph into a set of all acyclic paths. By using Locality-Sensitive Hashing it is possible to organize similar paths in the same neighborhood in hash tables. A query by example approach is performed by a spatial voting scheme evaluating pre-computed paths from the database.

We have seen several methods to perform symbol spotting in floor plan images which need complex mechanisms to describe documents with suitable data structures. In the following, we explore the use of deep learning techniques for object detection in floor plan images. Using these techniques it is possible to localize and recognize symbols in a single step.

We propose to detect symbols on plans using a deep architecture based on Faster R-CNN [98]. This network has been used on real scene images to recognize and to localize objects. There are many differences in object detection considering floor plan or real scene images. Comparing COCO dataset [78] composed by real scene images and *flo2plan* dataset at Section 3.2 composed by floor plan images. We can see in Figure 4.4 these main differences. In COCO dataset we have RGB image, objects are not very small considering the input size and are quite far from each other. Instead in *flo2plan* we small objects close each other.

(a) COCO example (b) *flo2plan* example

Figure 4.4: Comparing one image from COCO [78] dataset and *flo2plan* dataset

Having this kind of task we need to adapt our data and network model to use Faster R-CNN architecture. We use ResNet [53] backbone starting from pre-trained weights on ImageNet dataset.

Feature Pyramid Network

In *flo2plan* dataset, the object sizes have large inter-class and extra-class variances. To address this problem we need to use a network capable to manage a multi-scale object representation. One solution is to use Feature Pyramid Network [77] which can construct a multi-scale, pyramidal hierarchy during deep convolutional network processing. This network has surpassed all existing heavily-engineered single-model architectures for object detection, and, moreover, it computes different output at different scales.

This multi-scale architecture manages different anchors at different scales learning better representations, also for small objects. It is possible to define anchor size and stride similar to classic Faster R-CNN [98].

Adding perturbation to learn more robust features

The *flo2plan* collection is a very small dataset considering the large number of network parameters which we need to learn during the training phase. One solution to reduce over-fitting is to inject random perturbation inside the network [6, 72]. The backbone network is a composition of residual blocks with different channels.

The residual block formulation in ResNet 50:

$$y = RELU(F(x, w) + x)$$

where

$$F(x, w) = BN_{\gamma_2,\beta_2}(ConvOp(RELU(BN_{\gamma_1,\beta_1}(ConvOp(x, w_1))), w_2))$$

Having convolutional operator (ConvOp), batch normalization (BN) and Relu activation function (RELU). We decide to inject random perturbation inside the residual block as:

$$y = RELU(\epsilon(F(x, w)))$$

where the perturbation ϵ is:

$$\epsilon(x) = x + \mathcal{N}(0, \sigma x)$$

In this way, we inject perturbation with σ value is 0.1 directly inside the residual block during the training phase.

Preliminary Results

We perform several experiments on the *flo2plan* dataset to test different Faster R-CNN model versions. In particular, we check how the object size is important to detect symbols in floorplan images. We use ResNet-50 as a backbone network for the experiments setting different anchor-ratio and stride. The number of output neurons is related to the dataset classes. The evaluation metrics used to compare results is based on Mean Average Precision (mAP) with Intersection over Union > 0.5 as default for Pascal Object Detection Task [37].

We test three Faster R-CNN version: Faster R-CNN C4, single-scale model where the computed features are fed into classification and regression layer to classify and localize object; Faster R-CNN with Feature Pyramid Network, multi-scale model with a classification and localization layer per anchor-scale; Mask R-CNN, similar to

Table 4.1: Floor plan object detection, the obtained results

Backbone	Model	Weights	Inp.Size	Anchors	mAP50
50	Faster/C4	I	1024	(32, 64, 128, 256, 512)	0.4832
18	FPN	I	1024	(32, 64, 128, 256, 512)	0.4871
50	FPN	R	1024	(32, 64, 128, 256, 512)	0.4060
50	FPN	I	1024	(32, 64, 128, 256, 512)	0.6336
50	FPN	I	1024	(16, 32, 64, 128, 256)	0.6492
50	FPN/Mask	I	1024	(16, 32, 64, 128, 256)	0.6405
50	FPN	I	2048	(32, 64, 128, 256, 512)	0.6887
50	FPN	I	2048	(16, 32, 64, 128, 256)	0.6927
50	FPN+Noise	I	2048	(16, 32, 64, 128, 256)	0.6902
50	FPN/Mask	I	2048	(16, 32, 64, 128, 256)	0.7089
50	FPN/Mask	I	1024,1536,2048	(16, 32, 64, 128, 256)	0.7183

Faster R-CNN with Feature Pyramid Network and also segmentation layer to compute instance mask. We consider scale jitter data augmentation technique defining three image scales: 1024, 1542, and 2048 pixels.

We show in Table 4.1 the computed results, we can see different performance related to different architectures. The Faster R-CNN with FPN has better results than a singe-scale model. Multi-scale representation helps the network to detect small objects improving the overall result. Moreover, we tested also ResNet 18 having the worst results. It is also important to use pre-trained weights (I) concerning start training from scratch (R).

Another important aspect in Table 4.1 is the input scale, we can see as larger scale has better results, it is come out because reducing input scale, also the object size is reduced performing worse results. Moreover, also the anchor-ratio for Region Proposal Network is important, having smaller anchor sizes we obtained better results.

We have seen that using Mask R-CNN for instance segmentation we can get better results than other models. We can say that using the ground truth with mask information improves the network capability to learn better object representations. Unfortunately, noise injection does not improve the performance in Faster R-CNN with FPN. We need to investigate more in this direction to understand the reason.

In the end, considering the oldest methods for symbol spotting explained at the beginning of this section, our proposed deep network detects symbols in one step learning more suitable features to describe all patterns in the document. Moreover, we have proposed ad-hoc feature learning tricks (FPN, random perturbations) inside the network computational process making this methodology very competitive with respect to the oldest segmentation-recognition paradigm.

4.2 Clustering visually similar structured documents

In Section 3.1 we have presented a private collection named *Brandenburg* which contains a large variety of document layout. We need a procedure to discover similarities in the dataset considering a defined page fingerprint. In the following section, we explain the procedure to select pages to deal with relatively similar images. This solution has been published in [22].

To build one homogeneous subset one visual scan of all the images is infeasible. In the following section, we show the procedure to select similar pages automatically given a predefined template.

In Section 5.4 we will see the experiments done using the dataset produced by this approach.

Page fingerprint

The selection of the sub-set of pages is based on a page description (like a fingerprint) that is based on the column structure that is defined by the spatial organization of the vertical ruling lines. To represent the pages we therefore first identify the vertical lines in the top half part of the page by using the Hough transform. The vertical lines identified in one page are shown in red in Figure 4.5. The relative line position computed in the center of the top half part of the page (along the blue dotted line) defines the page fingerprint. Along the blue line, we compute the distances between neighboring lines that define the column width. Excluding lines too close to the page borders the fingerprint is made by considering the list of distances between neighboring vertical lines. In the example in Figure 4.5 the fingerprint is: $F = (d(1,2), d(2,3), \ldots, d(12,13))$ where $d(i,j)$ is the distance between the i-th and j-th vertical lines.

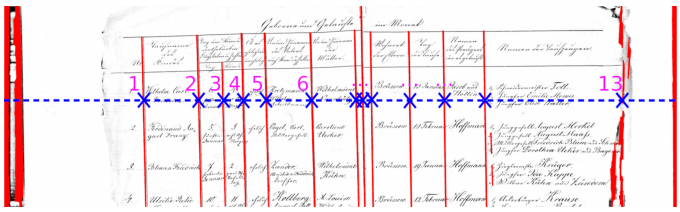


Figure 4.5: Page fingerprint.

Finding similar pages

Using the previous approach we can define one fingerprint for each page. As we can see from Figure 4.5 the scanned images are noisy and in particular, the identification

of vertical lines is not reliable in the center and at the borders of the image. This noise is incorporated in the fingerprint that is not unique for very similar pages.

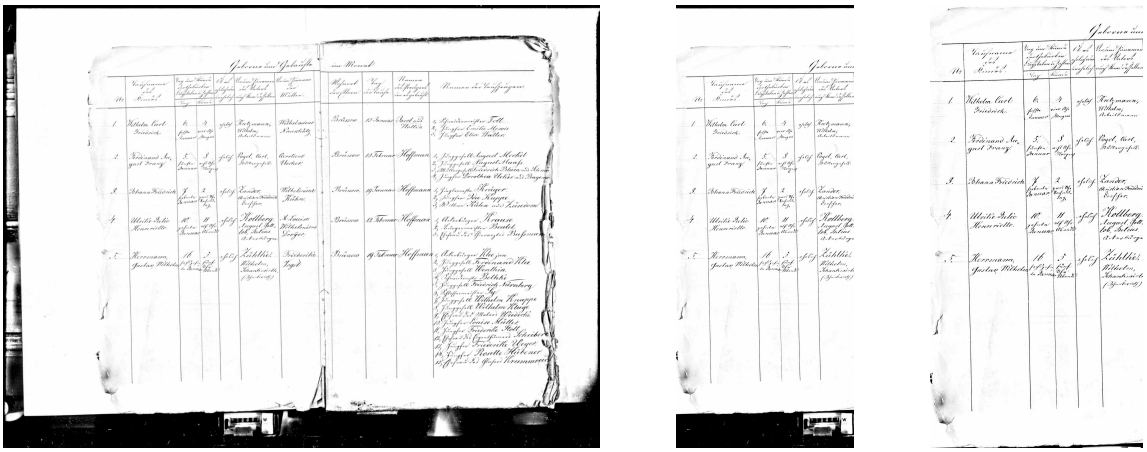
To identify the pages most similar to one prototype page we represent this page with one subset of the page signature, one pattern, which represents only the columns that are correctly identified in most cases. For instance, the pattern that defines the example page is $P = (d(1,2), d(2,3), \ldots, d(5,6))$.

The distance between one page fingerprint F and one pattern descriptor P can be computed (when $|P| < |F|$) by:

$$\mathcal{D}(F,P) = \min_{j \in 0,\ldots,|F|-|P|-1} \left(\sum_{i=0}^{|P|-1} |F_{j+i} - P_i| \right) \tag{4.1}$$

where $|P|$ and $|F|$ denote the length of the vector P and F respectively.

$\mathcal{D}(F,P)$ defines the smallest distance between the pattern P and one subset of the descriptor F. By using this distance it is possible to estimate the similarity between each document and the pattern and therefore select a subset of pages (with distance below a given threshold) from the larger dataset. In this way, we obtained the 4,956 pages used in the experiments in the *Brandenburg* collection.



(a) Input image (b) First segmentation (c) Second segmentation

Figure 4.6: One example page from the *Brandenburg* dataset.

As a side effect of this column alignment, it is also possible to crop the left page in a two-page document and use only this portion for the subsequent steps. In Figure 4.6 we show the whole page and in Figure 4.6*b* the cropped area. As we can see from the image, the cropped part needs to be cleaned by removing the noisy parts on the top and bottom. These parts are removed by first binarizing the page and the identifying the largest blobs in the top and bottom part of the page obtaining the image shown in Figure 4.6*c*.

This last image is used as input to the neural network. Since the records span the whole two-pages image (Figure 4.6*a*) it is possible to count the number of records only considering the smaller image in Figure 4.6*c*. In this way, we can reduce the image size and we can deal with less noisy pages. It is therefore also easier to generate the synthetic pages.

4.3 A Toolkit to generate structured documents

The generation of synthetic datasets is one common procedure when the labeled dataset is not large enough or when the production of ground truth is expensive. Several works use this approach to improve the results in different tasks. For instance, in [50] is proposed the creation of synthetic datasets to perform text localization in natural images. To address the ImageNet challenge, the data augmentation approach proposed in [65] consists of image translations, horizontal reflections, and patch extractions.

To address document image analysis, the generation of synthetic document images has been adopted for many years now. The seminal work by Baird (e.g. [9]) studied the problem of document image degradation and how to model it to improve the document recognition algorithms. Using the model proposed it is possible to generate synthetic data sets used in training classifiers for document image recognition systems. These techniques have been applied particularly to address printed documents and OCR algorithms.

In the area of historical printed documents, [63] proposed one solution for word spotting by generating synthetic image words. After one suitable pre-processing step, performed to segment words on the document images, the word matching involves the comparison of one query word synthetically generated with all the indexed words.

In [55] it is proposed one relevant approach for the automatic generation of ground-truth information that can be used to design a complete system for solving document image analysis and understanding tasks. In particular, it is possible to define the document structure through an XML file and appropriate stylesheets. The system produces synthetic documents with ground truth information that can be used for page segmentation, layout structure analysis, and other tasks. In the area of handwriting recognition, cursive fonts have been used to synthetically generate handwritten documents [30]. More sophisticated models, not needed in our task, have been proposed as well [15].

In this Section we describe one open-source toolkit, called *DocEmul*, that can be used to support the researchers to generate synthetic documents that emulate a real structured handwriting collection. This toolkit has been published in [19]. The main focus of the system is in the generation of structured synthetic Documents i.e.

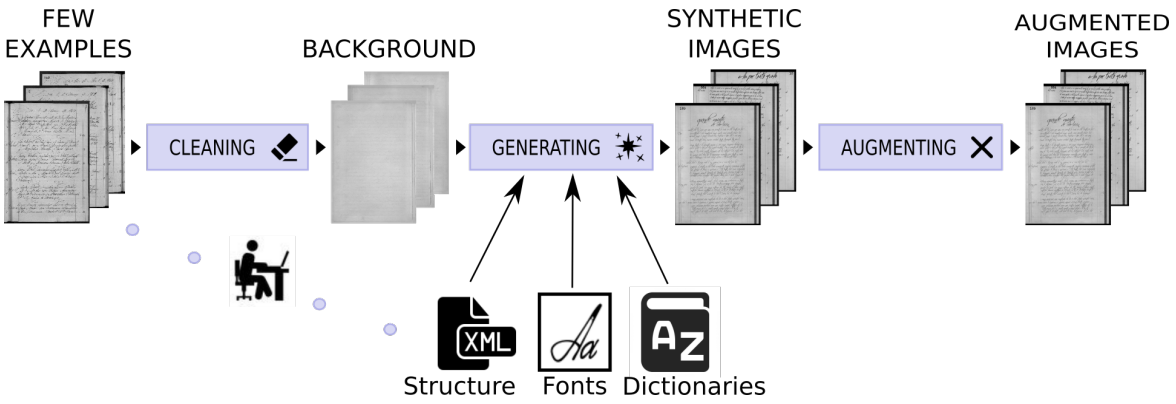


Figure 4.7: Flowchart of the synthetic document production process. It is possible to set the document structure, the fonts and the dictionaries used to generate the pages.

documents with a record-like structure. To the best of our knowledge, this is one important novelty of the proposed tool. By using the *DocEmul* toolkit it is possible to use the synthetic data to address document analysis tasks also when the real labeled datasets are too small for a suitable model training. In particular, we aim to design one generator for structured documents where each page might be composed of one header (usually at the top of the page) and several records that are written in the rest of the page. If we think of a structured document as a sort of handwritten table with different columns, every record can be considered as composed by several text lines and in turn, each line can be formed by a variable number of cells (text boxes).

We depict in Figure 4.7 one overview of all the implemented tools that can be used to generate synthetic handwritten documents. Starting from some example pages selected from the collection which we want to model, it is possible to extract the page background. This background is then used as a substrate where records will be written, each record could have variable structures using different cursive fonts. Moreover, it is possible to extend the synthetic collection by adding random noise, page rotations, and other visual variations.

To indirectly evaluate the DocEmul toolkit we considered the task of record counting in historical handwritten documents by using deep architectures proposed at Chapter 5. The target is, therefore, to count the number of records on each page and not to explicitly segment each record. This task can be seen as the first step towards layout analysis and record segmentation.
We performed some experiments on two collections: one benchmark dataset proposed in [5] for addressing the segmentation of historical handwritten documents and one collection composed by images provided by Ancestry (the global leader in family history and consumer genomics) through our research collaboration. We propose an open-source toolkit for the semi-automatic generation of synthetic handwritten documents containing records following a general structure. The Python

code, together with document structures and some generated images can be freely downloaded[2].

The DocEMul toolkit makes it possible to model the document aspect generating synthetic documents that look similar to actual pages. The toolkit is designed to emulate the document production process imitating the different ways adopted by the writers to create a kind of document collection.

In Figure 4.6(c) we present some examples of real documents that we want to emulate. The collections are composed of handwritten structured documents where the production process created a table-like structure composed of several columns and a variable representation of records.

In these documents, we need to model different aspects of the page. Historical documents could have degradations caused by aging and storage conditions which complicate the task. To properly model these artifacts, it is useful to model also the background. In this toolkit, this is achieved by extracting it directly from real images. Since several writers participated in the collection production during the time, usually we can find different writing styles and also the record structure could change with some variations from page to page. These are some of the possible variations that could be seen in historical handwritten collections which make the task challenging.

The DocEmul toolkit is composed of separate modules which could work together or at different times. In the following, we describe each module in detail.

Background Extraction

To extract the page background from real pages, we need to remove only the ink used to write the text on the paper and therefore no layout analysis is performed on the documents. The first step is to localize background pixels on the page by using binarization algorithms. The use of binarization in DocEmul is finalized at identifying and preserving pixels that most likely belong to the background and not at exactly identifying foreground pixels. One inaccurate binarization is therefore acceptable for the subsequent steps. The aim is therefore to find pixels to erase replacing them with a generated background. In the tool it is possible to use two different algorithms for the binarization task, the choice depends on the noise we want to preserve for the background extraction. The tool can use the Otsu [94] and the Sauvola [105] binarization algorithms. By identifying the foreground pixels it is possible to substitute them with suitable values that resemble the background. In particular, we "clean" the foreground pixels by replacing them with the average value of background pixels in a W $\times$ W window centered over each foreground pix-

[2] `https://github.com/scstech85/DocEmul`

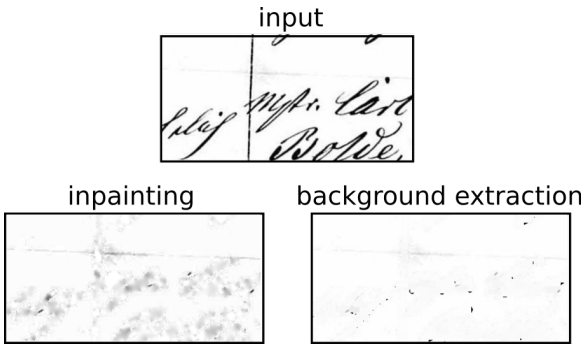


Figure 4.8: An example for the background extraction step. We compare the inpainting by biharmonic functions with our approach.

els. Usually, the window size is 20×20, but this is a changeable parameter of the tool.

By using this module, it is possible to obtain some background pages from a few pages in the collection which we want to model. We have compared this approach with image inpainting by biharmonic functions [29] from scikit-image [117], we can see in Figure 4.8 as the inpainting technique produces not usable background for our task.

In Chapter 5 we will see as the background extraction from real pages can help for training the models.

Structuring Handwritten Pages

To produce handwritten structured pages, the idea is to define one "ad hoc" general structure to model one collection adding random variability in the generation phase. Following the approach proposed in [11], we propose a model-driven generation technique defining a flexible model used to create synthetic documents adding some visual variations.

Having a look at real document collections that we want to emulate we can understand the general page structure for a given collection. Some of the features that can be inferred from the collection and that can be modeled in the tool are the fixed structure of the records and the presence of preprinted structures (e.g. vertical lines). These are examples of features that should be defined to produce synthetic pages that resemble as most as possible the real collection.

We aim to create one flexible description to define the most important document features easily. We use an XML file to define the configurations needed to create the synthetic pages. This file is used to characterize the header structure, the record structure, the fonts, the dictionaries, and the graphic objects which need to be used.

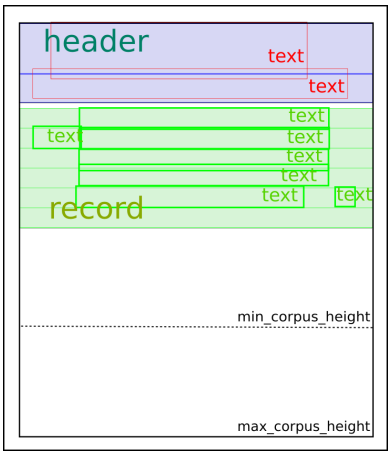


Figure 4.9: Page structure defined in the XML file.

Record and Header Structures

We model the record as one group of text lines where each line contains several textboxes defined as cells. We can set the height for each text line and the interline spacing between consecutive text lines. It is also possible to define a probability to write the text line on the page, in this way we add some variability to the document.

Each text line contains cells that extend the text line attributes defined above. It is also possible to define other attributes for each text box. For instance, the font and the dictionary used to write the text, the horizontal position and the width of the text box. Moreover, we can add some position variability to simulate a random variation of the text flow. It is also possible to define the probability of adding one cell to the structure, therefore, generating records with a variable number of items according to the model.

After defining the record structure it is possible to define one or more groups of records where each type of record has associated one probability of appearing in the document. In this way, we can define multiple structures of records and the morphology of the pattern which we want to model. We describe the document header like other records, however, in this case, it is not repeatable.

Fonts and Dictionaries

We can assign different options for each cell, in particular, we can define the font to write the text in these cells. It is possible to use different fonts downloaded from web sites. We can also define a set of dictionaries to use during the production phase and we can associate one dictionary to each cell.

Graphic Objects

To model some types of documents it is possible to add graphic objects to the document. Using this tool we can add objects as lines or boxes that can be filled with a fixed color or with salt and pepper noise.

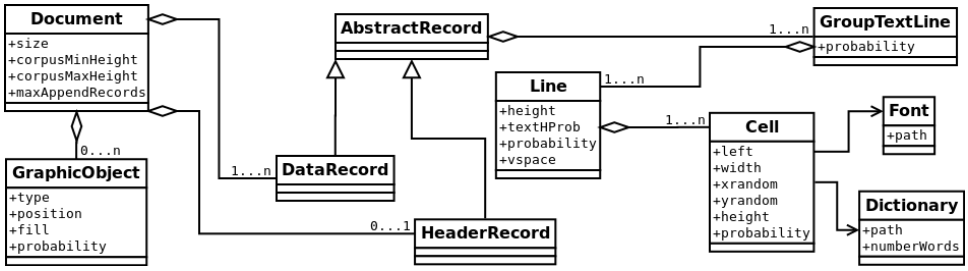


Figure 4.10: UML diagram to define the document structure used by the toolkit.

Generating Documents

In the document generation, some areas are defined to specify the regions where we are going to print the document contents. In Figure 4.9 we graphically illustrate the main areas. The blue area describes the header of the page. The record of the header, like other records, can contain mandatory fields (e.g. the page number) and other fields that are added according to a specified probability distribution.

The corpus area defines the part of the page where records are printed. Any page generated by the program will contain records in the area below the header and ending between the min_corpus_height and the max_corpus_height delimiters. The area depicted in green defines the zone on the page where one record is generated.

Data Augmentation

One module of the toolkit allows augmenting an input dataset modifying real images with some artificial transformations. In particular, we can add salt and pepper noise in the whole page or random sub-areas of the page. We can randomly rotate the page choosing the angle in a fixed range $[-A^\circ, A^\circ]$. It is also possible to add a scale variability to the document.

Model Specification

To better understand how to model one collection, we present the architecture used to define the page structure. In Figure 4.10 we show the UML diagram describing the XML model used to define the page structure.

We can define a document (`Document`) setting the size (height, width) and the corpus area between the `minCorpusHeight` and the `maxCorpusHeight` delimiters. Moreover, the tool generates a random number of records (`maxAppendRecords`) to fill the document in the previously defined corpus area. A document could contain graphic objects as lines or rectangles. It is possible to define them using the related class (`GraphicObject`) which defines the absolute position and also the probability to appear in the document.

To obtain an abstract document as a record container, we have defined an abstract concept for the record (`AbstractRecord`). This class is used to define the structure of the record. In structured documents, we can have a header record (`HeaderRecord`) which appears at most one time for a page containing several data records (`DataRecord`). Using the previous definition, an abstract record is composed of several groups of text lines (`GroupTextLine`) which are used to define a structured record. The text lines (`Line`) are associated with the group, we can define the height with a random variation (`textHProb`) for each line and also the probability to appear during the generation.

Defining a document as a container of text lines, the tool generates a document starting from the top to the bottom of the page, for each text line we define the interline space (`vspace`) concerning the next text line. A text line is composed of several text boxes (`Cell`) to fill with text. To obtain more variability, a cell is associated with a random position and the probability to appear in the document. A cell is associated with a font (`Font`) used to write the text extracted randomly from a dictionary (`Dictionary`).

The generation phase creates synthetic documents composed of only text and graphic objects over a transparency layer saved in RGBA file format. It is useful to generate different versions of the same collection but with different background pages.

4.4 Modeling datasets

We used the *Esposalles* collection (containing 200 pages) and the *Brandenburg* (containing 4,956 pages) to evaluate the toolkit capability to generate synthetic documents emulating the real collections. These are only examples of possible uses of the toolkit that is designed to be general and suitable to model also other collections.

Esposalles

This collection is one benchmark dataset proposed by [5] containing historical handwritten documents. These structured documents are composed of records that we

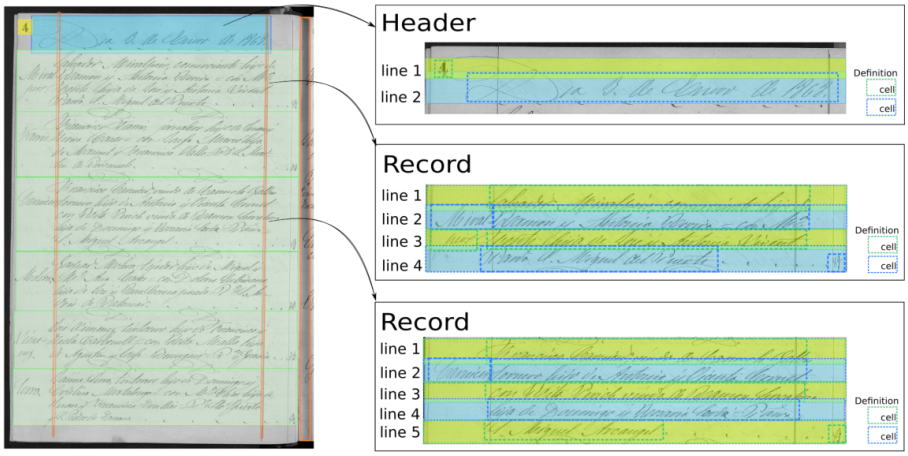


Figure 4.11: *Esposalles* dataset

can model with our toolkit. In Figure 4.12 we show one example page from the dataset. We will see in Chapter 5 this toolkit in action, in the experiments, when we generate 81,060 synthetic documents images containing several records between 3 and 9. The inputs have a resolution of size 366×256 pixels with black/white values.

Figure 4.11 is depicted as an image for the dataset which we want to emulate. Having a look at the page, we can see how it is structured. In particular, on the top, we can find the header composed of two lines. We can, therefore, model the header as a record with two lines where the top line has only one cell to show the page number. The second line contains the text as a title. The page contains 6 records and each record is structured differently. In Figure 4.11 we also show some extracted records and their structure. Having a look at these records we can observe that the structure for each record could change on the same page.

It is possible to simulate the production process using the toolkit and defining the document structure in the XML format and using some cursive fonts: "Scriptina", "A glitch in time" and "Love letter tw"; downloaded from `http://www.dafont.com` and Italian text as the dictionary.

In Figure 4.12 we show some synthetic images generated to emulate the *Esposalles* dataset. The two images are generated with different fonts.

Brandenburg

This collection is a private dataset from the Ancestry company and it is composed of structured documents. In Figure 4.14 we show one example page from the collection. We will see in Chapter 5 this toolkit in action, in the experiments, when we generate 61,914 synthetic images containing several records between 1 and 10.

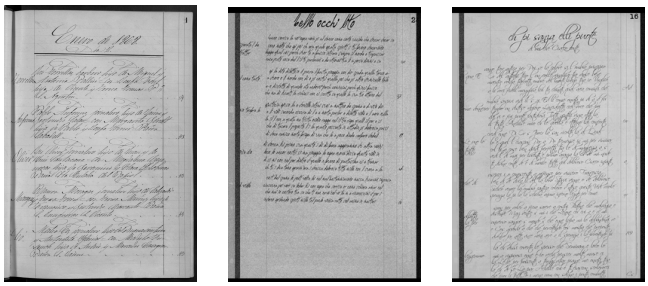


Figure 4.12: *Esposalles* dataset. One real image (left) and two generated pages using the same resulution in input to the networks.

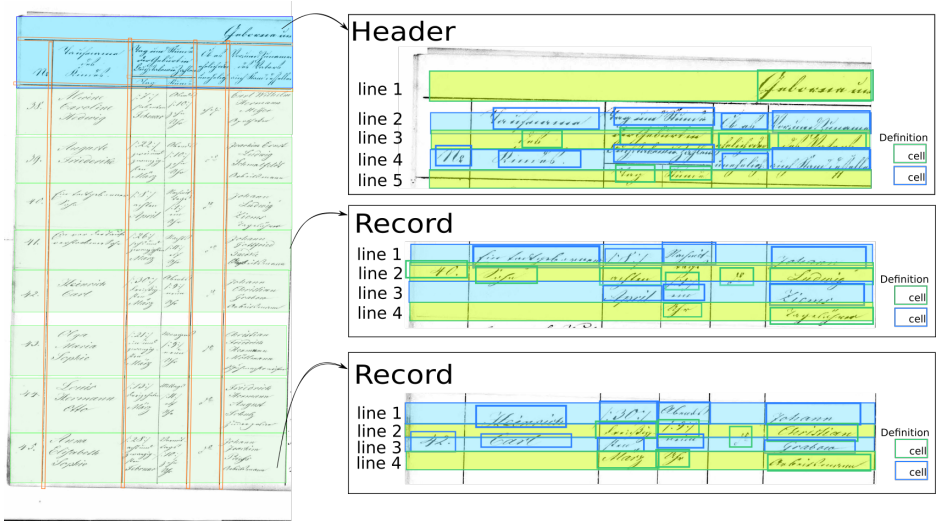


Figure 4.13: *Brandenburg* dataset

We can observe that the document structure is more complex than in the previous collection.

Figure 4.13 is depicted as one image from the dataset which we want to emulate. Again, on the top, we can find the header record. This page contains 8 records and each record is structured differently. Moreover, we also show some extracted records and their structure.

It is possible to simulate the production process using the toolkit and defining the document structure in the XML format using various cursive fonts “Scriptina”, “A glitch in time”, “Love letter tw”, and “Taken by vultures”; downloaded from `http://www.dafont.com` and Italian text as the dictionary.

In Figure 4.14 we show some synthetic image generated to emulate the *Brandenburg* dataset.

We show in Chapter 5 how to use the documents generated by DocEmul toolkit to train a deep neural network to solve the proposed record counting task.

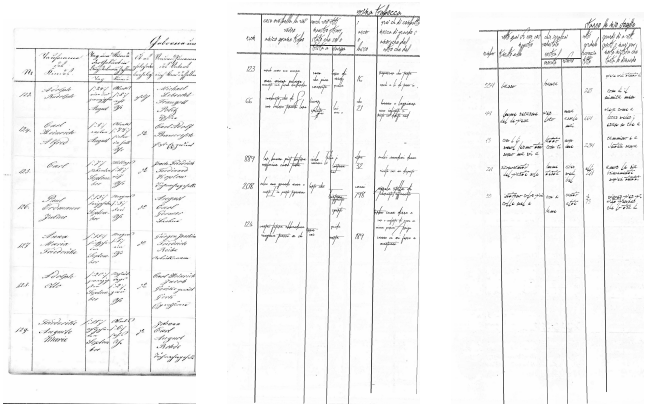


Figure 4.14: *Brandenburg* dataset. One real image (left) and two generated pages

4.5 Generating graphical symbols

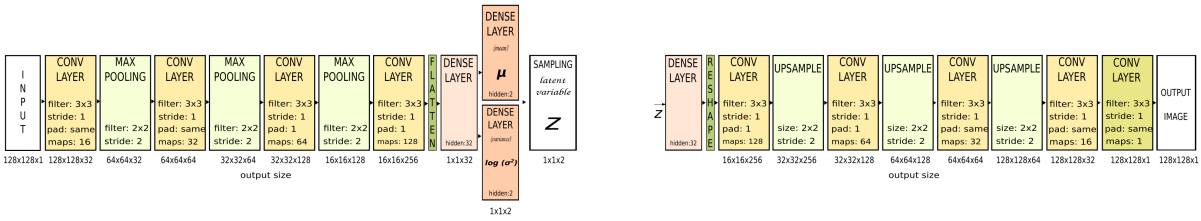
The generation phase proposed in Section 4.3 is based on a document structure defined by user experience. Recently, another solution to generate synthetic data uses neural network as generative models. These networks can model the intrinsic structure of objects or symbols given an input dataset.

Recently, various works in document image analysis use generative models to generate objects in this domain. In [62], the authors propose to use VAEs to generate different classes of handwritten numbers starting from the MNIST dataset of handwritten digits. In handwriting writer identification, VAEs are used to learn the generative model of a Japanese character dataset [126]. The authors propose to use Convolutional VAE to improve character generation accuracy.

One generative model based on GANs has been proposed for staff line removal as a preprocessing step in music score recognition [64]. The authors propose to use a U-Net network as Generator to produce staff-less images at the output similar to denoising autoencoders. Then the discriminator tries to differentiate between the generated images and ground-truth staff-less images.

Generating fonts automatically is another task where GANs have been applied successfully. In [1] Abe et al. propose a class discriminative Deep Convolution GAN (DCGAN) with the support of a classification network to refine the generated data. Including class representation inside the DCGAN architecture the model can produce robust fonts. Moreover, introducing a classification network along the generative model improves the generation ability of the system.

Overall, from these papers, we can notice that it is very difficult to generate symbols resembling those made by designers using deep learning techniques. Human designers have different skills to represent symbols that are difficult to generate properly by machine learning approaches.



(a) Encoder

(b) Decoder

Figure 4.15: The model architecture. Different transformations are depicted in different colors.

A Variational Convolutional Autoencoder

In this section, we describe how to prepare the data to train the Convolution Varational Autoencoder (ConvVAE).

Considering the difficulties in the symbol dataset, we build a VAE following the guidelines in Section 2.6 adding convolutional operators to learn better representations. To address the floor plan symbols generation, we propose the following architecture as depicted in Figure 4.15.

The encoder is composed of three convolutional layers with kernel 3×3 stride 1 followed by max-pooling with kernel 2×2 stride 2, respectively. In the end, we have a convolutional layer with kernel 3×3 followed by a fully-connected layer. The learned features are fed into two fully connected layers which produce mean and variance representations related to the latent variable which we want to model. The sampling operation takes as input the computed mean and variance, then it computes the ϵ value from a normal distribution $\epsilon \sim \mathcal{N}(0, 1)$. Other information related to the filter and network structure are in Figure 4.15 (encoder).

The decoder network aims to generate the input data from the computed z latent variable. To reduce checkerboard pattern artifacts we defined a decoder network as a series of "resize-convolution" operations [93]. The latent variable (z) is fed into a fully connected layer for starting the reconstruction phase. After that, the decoder continues to reconstruct the data with several convolutional and upsampling layers mirroring the encoder process but in reverse order.

All the layers use Rectified Linear Units (ReLUs) activation function. Finally, the last convolutional layer computes the reconstructed input by sigmoid activation function. More details on this network are shown in Figure 4.15 (decoder).

For the training phase, we use the ADAM optimizer with a learning rate of 0.0001. The loss function has been described in equation 2.6. We use the early stopping criterion to stop the training phase evaluating the loss score on validation dataset [46].

Figure 4.16: Selected symbols.

Evaluate the Generated Data

In this section we describe the experiments done on different floor plan datasets explaining how to extract graphical symbols from labeled floor plan images. We train one model for each category and then we show the results evaluating the trained models and their different performances.

For these experiment, we extract graphical symbols from labeled floor plan images. We train one model for each category and then we show the results evaluating the trained models and their different performances.

Symbol selection

The symbol dataset presented here contains many object categories, we decide to select only some categories. We select three different object categories including bed, armchair, and toilet because they are present in all of our floor plans with different aspect variations.

In one experiment we want to evaluate how VAEs can manage the possible data variations. We select first bed symbols considering their size and shape, and selecting images composed by bed objects with single or double size (or other graphical variations). Moreover, we decide to set a fixed orientation to bed images with vertical shape and the pillow on the top part.

Then, we select toilet objects with a fixed vertical position having only two orientations (base part on the top or the bottom shape), in this way the dataset has only this hidden peculiarity to learn during the training phase. In the end, we collect armchair objects maintaining their original shapes and orientations.

To fully understand structures and variabilities on the data, in Figure 4.16 we show some examples extracted from the two corresponding datasets (bed, toilet, and armchair). Before to start experiments, for each category we split the related dataset into different training and validation sets avoiding any ambiguity for the training and testing phases.

Considering how we have collected symbols from each category, we define one model for each semantic class to understand how the model can code the input symbol into embedded space evaluating the quality of reconstruction data. In particular, we want to put more emphasis on how the learned variable latent space can organize different graphical aspects in an unsupervised manner. We explore the variable latent space computed from an input image used as a seed to discover how the model represents data. We compare the results obtained by VAEs to results obtained by an autoencoder without variational inference regularization.

Results

In the following, we analyze the results obtained in the experiments, considering the capability proved by the VAE. We propose a subsection for each experiment.

Bed

The extracted symbols from *flo2plan* are not that much uniform, the trained model generates bed symbols which are affected by small distortions, this effect comes from the high variability on the data which makes the training much harder.

In ISTA dataset, bed symbols are very similar to each other, in this way it is easier for the model to train and also generate the related patterns. After the training phase, we have seen as the model is able to encode similar objects very close inside the latent space. After the inference step, we use the encoded input to generate a series of symbols sampling from its neighborhood. In Figure 4.17 we represent the reconstructed input (red rectangle) surrounded by other reconstructed symbols sampled from the neighborhood of encoded input. Apparently, although such images seems to be identical or very similar, they are affected by very small perturbations appearing plausible symbols.

Computing the inference of test set samples, we can inspect how the model maps input samples inside the latent space. In Figure 4.18 is shown the computed latent space where each point is related to the input image. We can see that double and single beds are very distinct and the details are also differentiated.

Armchair

The experiments for these symbols are quite their original from those described above. The seats taken from the ISTA dataset are very uniform and there are not

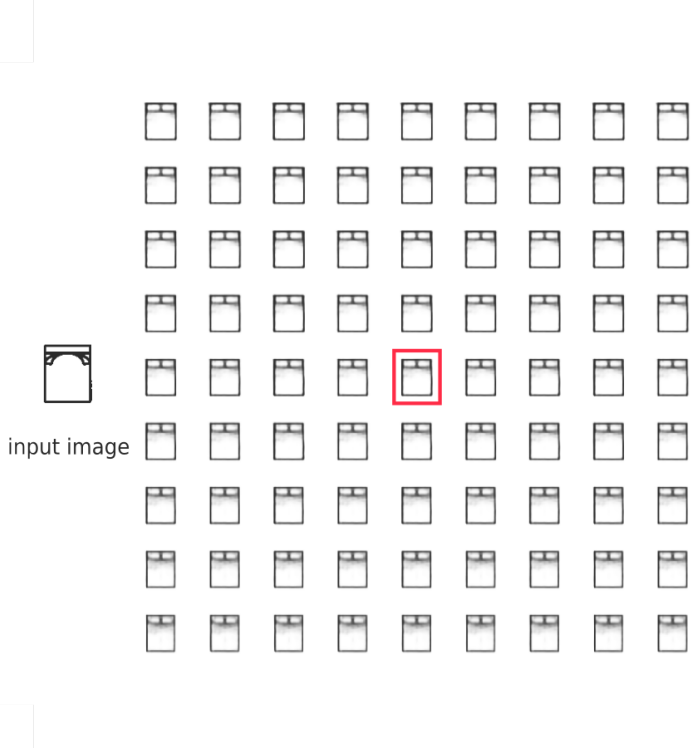


Figure 4.17: Grid of generated beds from flo2plan

many differences as in the case of beds and toilets. Therefore, the images representing the seats have been left in different positions, to verify the VAEs ability to discriminate the orientation of each image. In Figure 4.19 we show the computed latent space for armchair symbols (ISTA test set), notice different underlined clusters related to precise positions of the seats, as in the upper left corner or in lower right.

In *flo2plan* dataset we have less armchair than in the other dataset (Table 3.1), moreover the armchair instances are very different in position and shape. In the learned latent space there is not much homogeneity. We see how between the two features that the VAE is expected to learn, the different position and shape, it has basically only learned one, namely the shape, since the amount of data is limited to train the model properly. This is observable in the generation phase, where, even with a small neighborhood, images are generated similar to the input symbol, but with differences in positions, as can be seen in Figure 4.20.

Toilet

For this category of floor plan symbols, we study the VAEs ability to discriminate two possible positions of toilet: facing upwards and downwards. In Figure 4.21 we show the learned latent space: in the upper part there are all the toilets turned

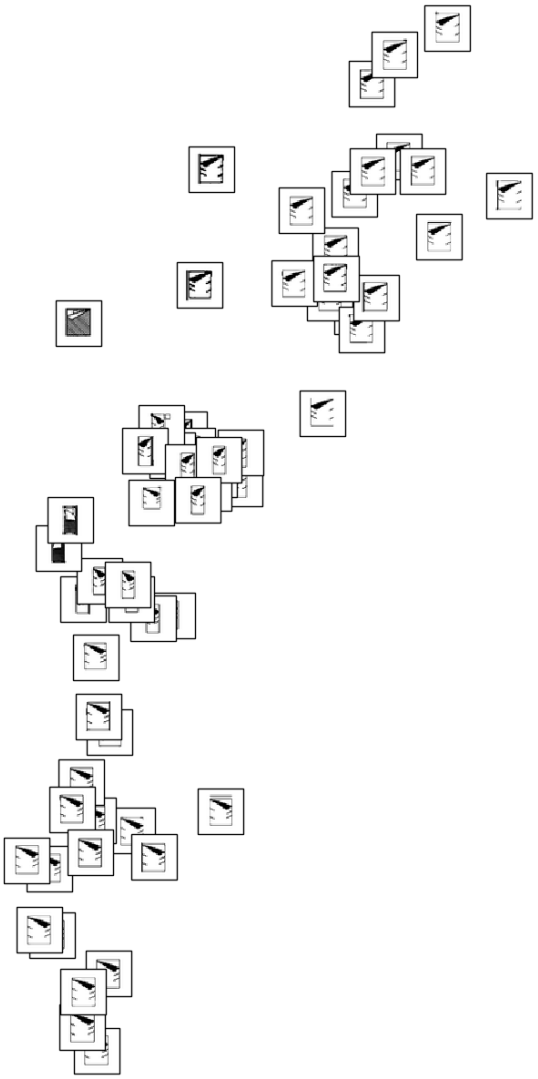


Figure 4.18: The embedded representations computed by VAE over bed symbols from ISTA dataset

downwards and in the lower part those upwards. Have a look to Figure 4.22, you can see another behaviour of our model during the reconstruction phase. The VAE is able to denoise the input symbol (#rec1), and also in its neighborhood there exist different scales (#rec2) and without any noise (#rec3).

Variational Autoecoder vs Autoecoder

We want to describe the differences between using VAE architecture over Autoencoder (AE). To do so, we define an AE where the latent variable (z) is not a Guassian variable and does not use the VB inference to approximate the true posterior. For these reasons, the learned embedding space does not follow a normal distribution.

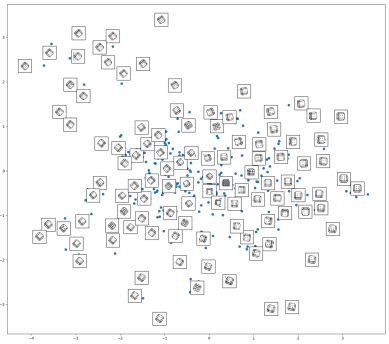


Figure 4.19: Latent distribution of ISTA armchair

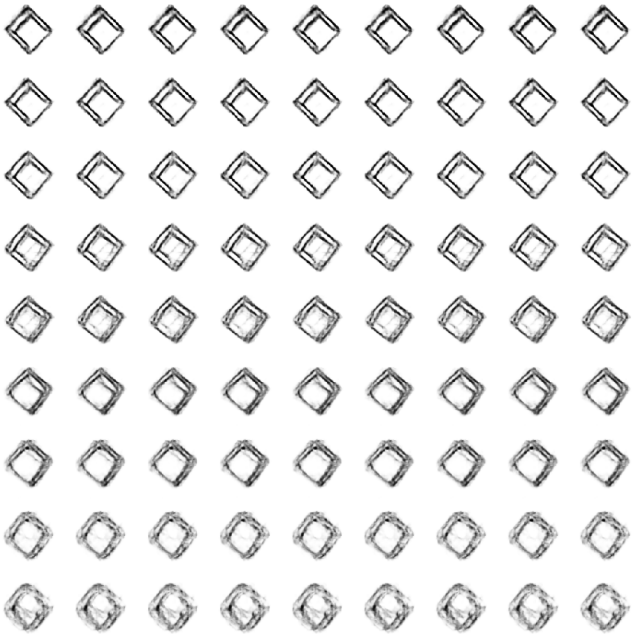


Figure 4.20: Grid of generated armchair from flo2plan

Considering Figure 4.23, we can see that any feature discrimination is performed in the latent space.

The proposed model has reported satisfactory results. It has the capability to map the data correctly and to discriminate, in the latent space, the various types of processed objects almost correctly. What follows is a generation that produces examples similar to those in input, and also improves some defects of the input images, such as drawing a continuous line in images that had a non-continuous outline. The VAE has also the ability to remove the noise in the input image and to generate, moving in its neighborhood, new images similar to the input one without

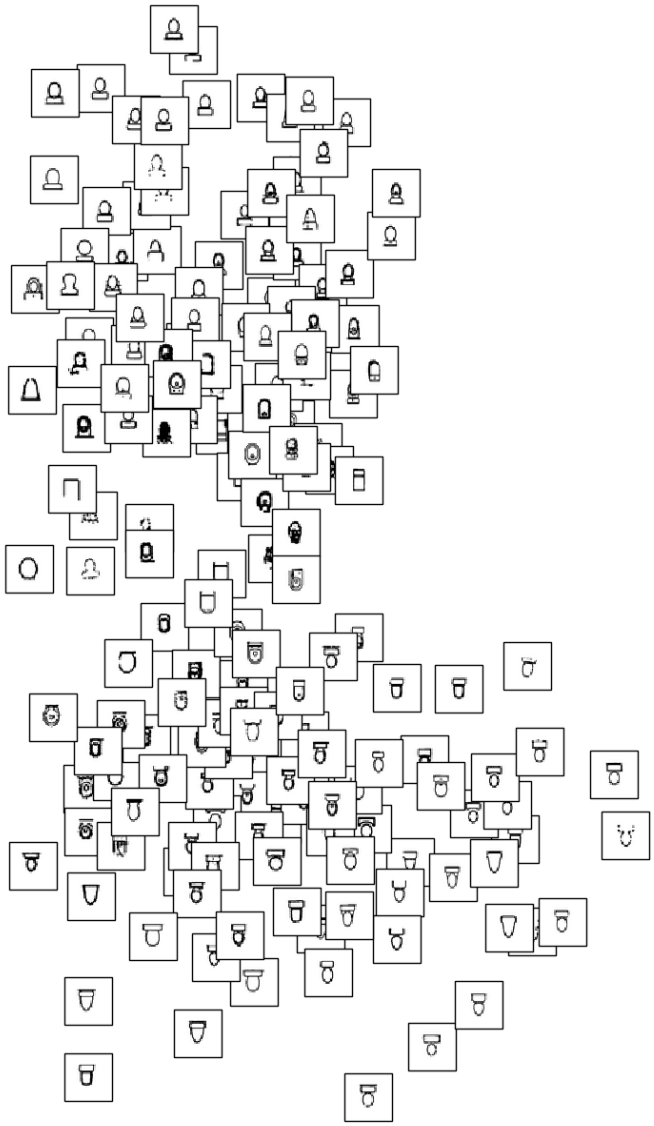


Figure 4.21: Latent distribution of flo2plan toilet

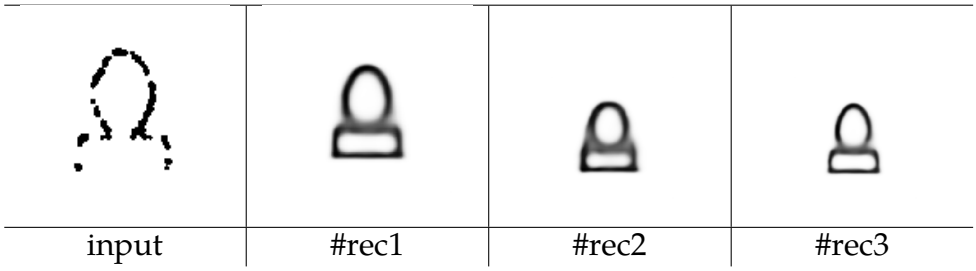


Figure 4.22: Using VAE as a denoising decoder

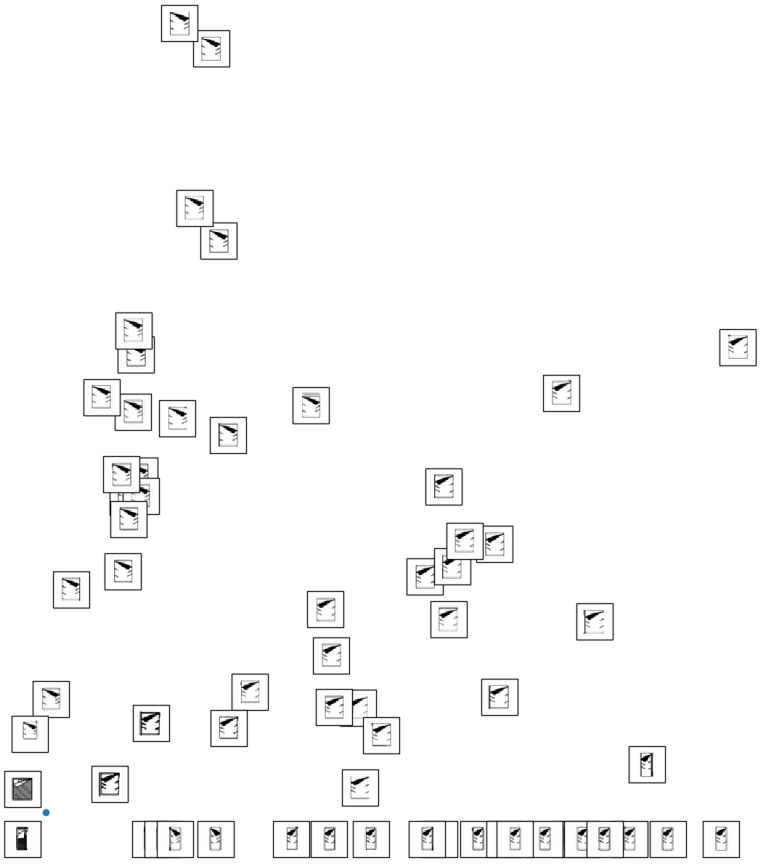


Figure 4.23: The embedded representations computed by AE over bed symbols from ISTA dataset

the noise and with different scale. We prove also the advantages of using VAE, for its ability to learn better learning space than an AutoEncoder.

What could further improve the developed VAE would be the availability of additional data in order to make the model able to recognize more different features at the same time, as we shown for armchairs. It would also be possible to assess whether the images generated are recognizable by an Object Recognition model,

trained to recognize the components of a house; this could also lead to data augmentation for floor plans. It will be possible to draw new ones replacing the old components with those generated by the model.

4.6 Summary

In this chapter we have seen several techniques to create training data. We can summarize them as follows:

- Labeling floor plan images is a labor-intensive task. It is possible to label these images by using the proposed interactive annotation tool to reduce the time of labeling.
- Detect symbols in floor plans is a difficult task, especially when we have a small training dataset. The old-fashioned techniques used to address symbol spotting based on the segmentation-recognition paradigm are very complex to use in any document image structure. Using deep architectures we could detect objects having good performances and defining a general framework.
- It is possible to cluster structured handwritten documents by using a rule-based method to select similar pages from a very large collection.
- Generating synthetic documents from a few labeled examples it is possible by using the DocEmul toolkit. The toolkit has data augmentation, background extraction, and other useful features to generate structured handwritten documents.
- Generating floor plan symbols with graphical variation is possible by training a Variational Autoencoder. This method learns a good representation in the latent space obtaining also interesting side effects as denoising function.

Chapter 5

Record counting in historical handwritten documents

One important application area for the processing of handwritten documents is related to information extraction from historical documents coming from census, birth records, and other public or private collections. The pages in these documents are often semi-structured and contain a variable number of records, each record follows a general structure but does not have a fixed number of lines or items. For instance, in a given document there might be records with more or fewer details according to the information available. By completely recognizing the content of these documents it is possible to reconstruct genealogies and perform demographic studies [17] [28] [38] [91].

When analyzing handwritten historical records it is possible to explicitly segment each record identifying its position in the page or to count the number of records on each page. It is clear that the former task provides more information, but is also more complex. On the other hand, often the solution of some sub-tasks can provide valuable information to the users before, or instead of, recognizing the whole content. In particular, when the recognition of handwriting is difficult, and the transcription is performed by human annotators, one accurate count of the number of records in one collection can provide useful information to foresee the amount of data available in the digitized documents and therefore give an estimate of the conversion costs. One example of page with records highlighted is shown in Figure 5.1.

One solution to the record segmentation task has been proposed in [5] where the authors perform structure detection and page segmentation applied to marriage license books. The latter books are handwritten documents where each page contains a variable number of records. Each record is composed of three main logical entities (body, name, and tax) and the number of records in each page is variable. In [5] different approaches to perform record and cell segmentation (including 2D Stochastic

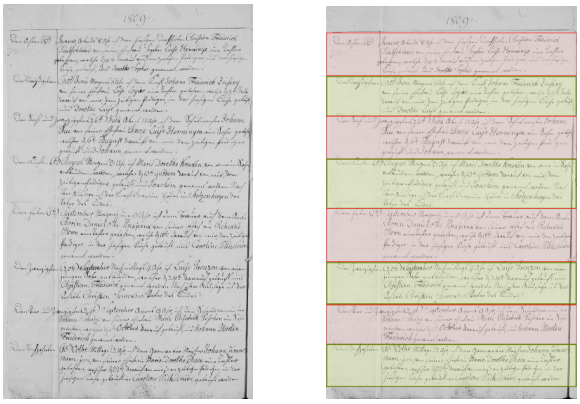


Figure 5.1: The records in the document on the left are shown with different colors on the right.

Context Free Grammars) are compared. As an indirect result of this segmentation step, it is possible to count the number of records on each page.

5.1 Record counting system

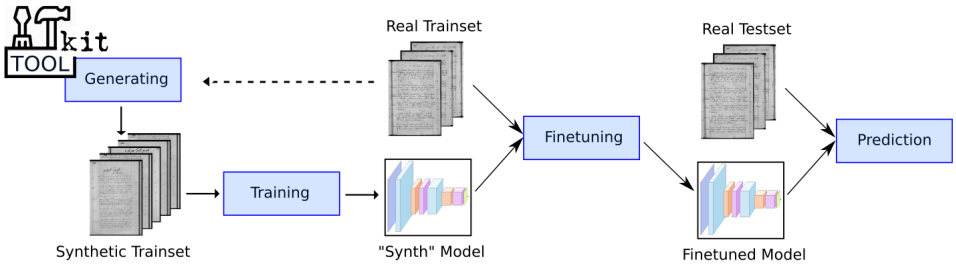


Figure 5.2: An overview of the proposed system to train and predict the number of records in an handwritten collection.

Object counting in an image is a relevant task in computer vision, with several applications in real-world problems. One solution for object counting has been proposed in [113] where the authors aim to recover a density function related to one input image and enumerate the objects. The density function is a real-value function over the pixel grid and the integral over this representation is expected to match the overall objects count.

The authors also highlight two approaches for object counting. *Counting by detection* is based on detecting the objects and then counting them. In *counting by regression* the objects are counted by using only input image features and object counting is cast as a classification (or regression) problem and to cover all the dataset variabilities a large training set is usually required.

In recent years, deep architectures have been proposed to address many computer vision tasks and object counting is not an exception. More specifically, several tasks in document image analysis applications have been often addressed with artificial neural networks [87].

We address the record counting problem by using deep Convolutional Neural Networks (CNNs) [71]. To the best of our knowledge, this is the first time that these techniques have been applied to this task. One related work [106] investigates the task of even digits counting in synthetic images generated from the MNIST dataset employing CNNs. The authors apply the same approach also to count pedestrians by creating synthetic images to extend the initial dataset. People counting by using convolutional neural networks have been addressed also by [122] [124] [128]. In particular [122] and [128] compute the density function (like in [73]) by using CNNs and then the integral over the image domain of the density function is used to count the number of items. On the other hand in [124] there is no density map and one regression neuron is the network output whose value is expected to identify the number of people in the image patch. One similar approach (with one regression neuron in output) is proposed in [96] for the task of fruit counting. What is common to all these methods is the assumption that individual items to be counted (i.e. people, fruits or cells in microscopy images) cover a relatively small portion of the image. Moreover, single items can be often modeled as Gaussian kernels centered on the location of the objects (or slightly more complex models, like in [128]). In several works, it is also assumed, for training and testing purposes, that the position of each item is known and can be used for the network training.

On the other hand in the application addressed in this chapter the items (records) span large portions of the image, that are often nearly completely covered by records. Also, the information about the position of each record is not available for the most challenging dataset used in our experiments. Therefore it was not possible to use the above methods for attaching the records counting task described in this work.

It is well known that to train CNNs it is important to use a large dataset of labeled instances. To the best of our knowledge, there is no large data set annotated on the basis of the number of records in the pages and therefore we need to find a solution to this lack of data. This is not an uncommon problem and, when dealing with pattern recognition tasks by using a learning system, common solutions to the scarcity of data are based on the generation of synthetic data and the use of data augmentation. In the first case, synthetic data that emulate real ones are generated with one suitable application. In data augmentation, the number of real data is increased by adding distortions or noise to the existing ones. The best practice requires to use the generated data only for training the system, while the performance should be always computed on real data. In the item counting task synthetic images of fruits are for instance generated in [96].

In the area of document image analysis and recognition, synthetic data generation has been used for instance to model the character degradation [82] or to generate synthetic documents for performance evaluation of symbol recognition systems in the area of graphics recognition [32]. In the field of handwriting processing, handwritten documents have been generated by using standard cursive fonts with an approach that is important for our research [30].

We present in Figure 5.2 one overview of the proposed approach for counting records, this work has been published in [22]. As we can notice, one Toolkit [20] (explained at Section 4.3) is used to generate a large dataset of pages used for the preliminary training of the CNN. Real data are then used to fine-tune the model and to asses the system performance in the test set. Concerning the architecture (presented at Section 2.3), we modified the *AlexNet* architecture [65], the *Network in Network* (NIN) architecture [76], and the $VGG16$ architecture [111]. In all the cases we modified the input size and the final and output layers in order to fit to the record counting problem.

To evaluate the proposed approach we used two collections: one Benchmark dataset proposed in [5] for addressing the segmentation of historical handwritten documents (*Esposalles* collection explained at Section 3.1); one collection composed by images provided by Ancestry (*Brandenburg* collection explained at Section 3.1).

Considering *Brandenburg*, we need to select a useful subset of handwritten pages to use in the following Experiments. At Section 4.2 we have seen a rule-based technique used to select similar documents automatically.

5.2 Convolutional models

The first model used to count the number of records is based on the well-known Alexnet architecture [113], where we changed the input size and replaced the output classification layer with one regression neuron, trained to count the number of records in the page. According to [106] we cast the counting problem as a regression one. In Figure 5.3 (top) we graphically depict the corresponding architecture.

In the second model, we extended the *Network in Network [76]* (NIN) architecture, originally the output of the global average pooling is fed into one softmax layer. In this architecture, we have one single feature map as an output map that is followed by one global average pooling to compute the prediction about the number of records. In Figure 5.3 (bottom) we graphically depict the corresponding architecture.

The third model used to count the number of records is based on the VGG16 architecture [111], where we changed the input size and replaced the output classification layer with one regression neuron, trained to count the number of records in the page.

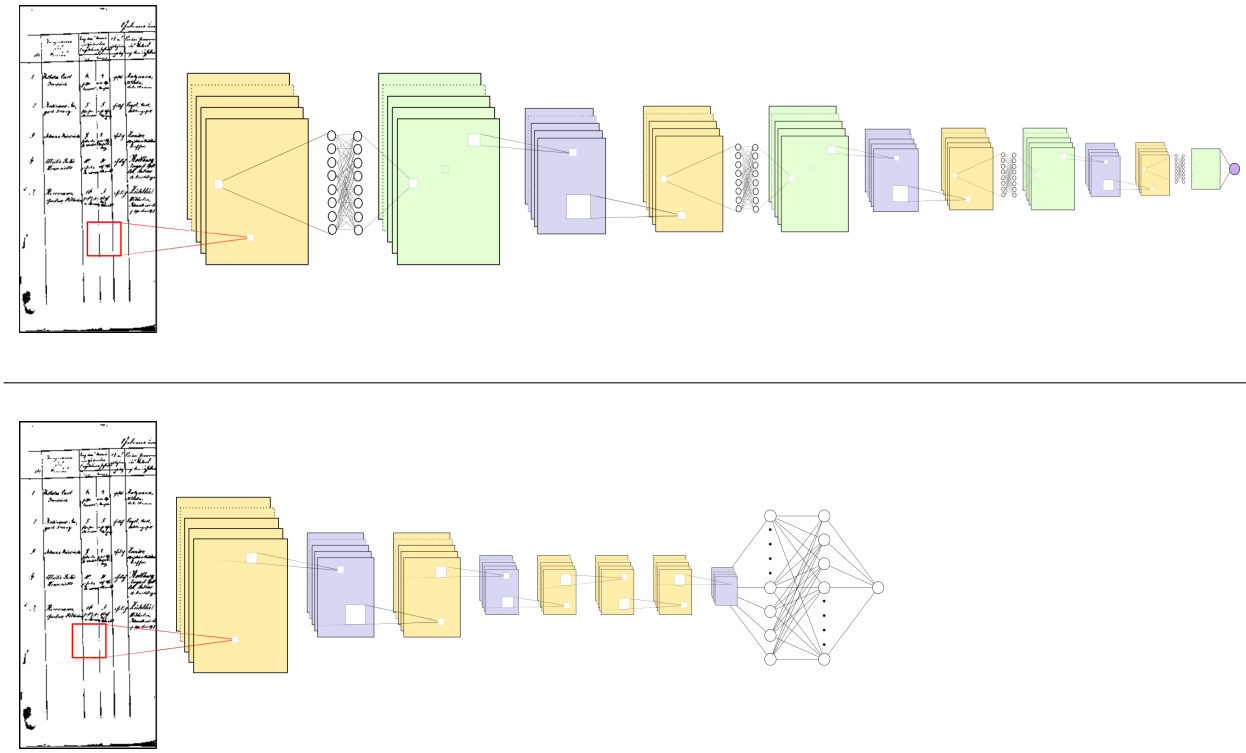


Figure 5.3: Architectures used in the experiments. Top: Regression layer added to the NIN model. Bottom: regression layer added to the AlexNet model. A similar change is considered for the VGG16 architecture.

The input dimensions slightly change according to the dataset. In all the cases we scale the images preserving the original proportions to reduce the network input size and improve generalization. With the *Esposalles* dataset the input images have been scaled to 366×256, while with the *Brandenburg* dataset the images have been scaled to 450×190. In both cases, we used binarized images (obtained with the Sauvola algorithm [105]) as input for both the training and test.

5.3 Evaluating system

As previously mentioned, to evaluate the proposed techniques we used the *Esposalles* collection (containing 200 pages) and the *Brandenburg* one (containing $4,956$ pages after clustering technique presented at Section 4.2). In both cases, we generated synthetic data as described in Section 4.3.

For the *Esposalles* collection we generated $81,060$ synthetic pages with associated information about the number of records on each page. The synthetic dataset contains pages with several records comprised between 3 and 9 even if the Benchmark collection in [5] contains only pages with $5, 6$, or 7 records each.

The distribution of the 4,956 pages in the *Brandenburg* collection is as follows (where (X, Y) means X pages with Y records): $(127, 1)$, $(273, 2)$, $(356, 3)$, $(485, 4)$, $(574, 5)$, $(806, 6)$, $(813, 7)$, $(833, 8)$, $(689, 9)$. The distribution is not balanced, but there is a large number of pages for each number of records. For this collections we generated 61,914 synthetic pages with a number of records comprised between 1 and 11.

The variability of the *Brandenburg* collection is significantly higher than the *Esposalles* one not only because the number of records in each page range from 1 to 9 but mostly because the pages come from different sources and the layout is variable (some examples are shown in Figure 5.5).

5.4 Experiments

In this section we describe the experiments performed to analyze the record counting. Two main approaches are considered for training the models (Figure 4.7).
In the first case, we use only real data for the training phase considering the cross-validation data for stopping the training.
In the second case, we first train the network using only synthetic images (real images are used as a validation set to stop the learning). This pre-trained network is subsequently fine-tuned using the real images in the training set that are different from those used as a validation set.
In both cases, the test is made with one disjoint set of real images and suitable measures are computed for performance evaluation.

Performance evaluation

As presented at Section 3.3, the system performance are measured with two values. The *Accuracy* is the percentage of pages where the number of records is correctly identified.
The *Error*, similar to Mean Absolute Error (MAE), is the percentage of errors in the record count when deciding on one page at a time. This value is defined according to Equation (5.1) where r_i is the actual number of records in page i, p_i is the predicted value ($\lfloor p_i + \frac{1}{2} \rfloor$ is the rounded predicted value), and N is the number of test pages.

$$Error = \frac{\sum_{i=1}^{N} \left| \lfloor p_i + \frac{1}{2} \rfloor - r_i \right|}{\sum_{i=1}^{N} r_i} \tag{5.1}$$

Esposalles

In the *Esposalles* collection we performed several experiments to identify the best training strategy and compare the results obtained by the proposed approach with those described in [5] considering the same split of the data in training, validation, and test datasets (we refer to this partitioning of data as Benchmark Split). Due to the limited size of the dataset in [5] we performed also stratified cross-validation to estimate the error rate on a larger dataset.

Esposalles: Benchmark Split

In the Benchmark experiments, we compare the results obtained by our system with those presented in [5] using the same split (150 pages for training, 10 for validation, and 40 for test).

Table 5.1: *Esposalles* dataset. Comparing different backgrounds in the Benchmark split.

Model	Initial weights	Training	Accuracy (%)	Error (%)
AlexNet	Random	Synth/W	85.0	2.6
AlexNet	Synth/W	Real+DA	95.0	0.8
AlexNet	Random	Synth/E	90.0	1.7
AlexNet	Synth/E	Real+DA	97.5	0.4

In the first experiment (Table 5.1) we evaluated the impact of the use of a white background (Synth/W) instead of the background extracted with the procedure described in Section 4.3 (Synth/E) when building the synthetic dataset. In both cases, the initial training is made with the synthetic images and the 160 training and validation images are used to stop the training. In the fine-tuning, we first augmented the 160 training and validation images (see [18] for additional details) and therefore used 1350 pages for fine-tuning and 90 as a validation set. We compared the two training data using the AlexNet architecture and concluded that in general, it is better to use a real background than a white one. In the subsequent experiments, we always used extracted backgrounds.

In the second experiment, we verified that training a network pre-trained on Imagenet dataset might provide better results concerning a random initialization of the weights (Table 5.2). In particular, for both the AlexNet and the NIN architectures one 100% accuracy is achieved by training the pre-trained network with synthetic data only. On the other hand, when using randomly initialized weights we need to perform a fine-tuning with real data to achieve the 100% accuracy with NIN

Table 5.2: *Esposalles* dataset. Random initialization vs pretrained Imagenet in the Benchmark split.

Model	Initial weights	Training	Accuracy (%)	Error (%)
AlexNet	Imagenet	Synth	100.0	0.0
AlexNet	Random	Synth	90.0	1.7
AlexNet	Synth	Real+DA	97.5	0.4
NIN	Imagenet	Synth	100.0	0.0
NIN	Random	Synth	95.0	0.9
NIN	Synth	Real+DA	100.0	0.0

(97.5% with AlexNet). In the subsequent experiments with the *Esposalles* dataset, we trained networks pre-trained on Imagenet.

Table 5.3: *Esposalles* dataset. Results on Benchmark split.

Model	Initial weights	Training	Accuracy (%)	Error (%)
AlexNet	Imagenet	Real	87.5	2.14
AlexNet	Imagenet	Synth	100.0	0.0
NIN	Imagenet	Real	100.0	0.0
NIN	Imagenet	Synth	100.0	0.0
VGG16	Imagenet	Real	92.5	1.29
VGG16	Imagenet	Synth	100.0	0.0
[Alvaro et al][5]			80.0	-

In the third experiment, we considered one recent deep model still pre-trained on Imagenet (the $VGG16$ architecture [111]). In Table 5.3 we report the results obtained on the benchmark split when considering three architectures: AlexNet, NIN, and VGG16. For each architecture, we compare the performance when fine-tuning the Imagenet network with real data (without the data augmentation considered in the first two experiments) and when training it with synthetic data like in the second experiment, but without performing a fine-tuning. We can notice that the training with real data has in general worst performance concerning the training with the synthetic data.

In Table 5.3 we also compare the results achieved by our system with that reported in [5] where the right number of records is predicted for 80% of the test documents. It is very important to observe that the system in [5] is designed to segment

the records and therefore the record counting is only one additional information that is computed from the segmentation.

From this experiment, we can notice that the networks trained with Synthetic data provide one 100% accuracy also without fine-tuning. With these data, the NIN slightly outperforms the other architectures.

Table 5.4: *Esposalles* dataset. Average results with five fold cross-validation.

Model	Initial weights	Training	Accuracy (%)	Error (%)
NIN	Imagenet	Real	94.5	0.9
NIN	Imagenet	Synth	97.1	0.5
VGG16	Imagenet	Real	86.1	2.4
VGG16	Imagenet	Synth	93.5	1.1

***Esposalles*: Five Fold Cross-validation**

Since the number of labeled pages in the *Esposalles* collection is limited, in this experiment we tested the system with a fivefold stratified cross-validation. The *Esposalles* dataset contains 44 pages with 5 records, 152 with 6 records and only 4 with 7 records. The five folders contain, on average, 40 pages each. In each fold we used 160 pages to perform the fine-tuning using 142 pages for training and 18 as validation set; we left the remaining 40 pages for evaluating the performance on real data.

In Table 5.4 we report the average values of the results obtained in the five folders. In this experiment, we considered the two most promising architectures: NIN and $VGG16$. Like in Table 5.3 we compared the training using only real data of networks pre-trained on Imagenet and training the same networks using synthetic data. In both cases, one network trained using only synthetic data outperforms one network trained using a very limited number of real data. It is worth recalling here that for each folder we use for the Real training on average 142 real training data (the validation set contains 18 pages). In the training (Synth) we use $81,060$ synthetic pages while the validation and test sets are the same as the Real training.

Brandenburg

The second main group of experiments is made using the $4,956$ pages of the *Brandenburg* collection described in Section 5.3. Since this dataset is significantly larger than the *Esposalles*, we made more detailed experiments. In particular, we analyze the change of performance when varying the training set size. We split the pages into three sets: $3,165$ pages for training, 796 for validation and 995 for the test.

Table 5.5: *Brandeburg* dataset. Comparison of different architectures (Random initialization of weights).

Model	Initial weights	Training	Accuracy (%)	Error (%)
Alexnet	Random	Real	66.53	6.19
Alexnet	Random	Synth	46.63	12.40
Alexnet	Synth	Real	74.87	4.40
NIN	Random	Real	74.47	4.53
NIN	Random	Synth	51.26	9.84
NIN	Synth	Real	77.79	3.96
NIN + FC	Random	Real	76.38	4.18
NIN + FC	Random	Synth	48.34	10.72
NIN + FC	Synth	Real	76.98	4.06
NIN softmax	Random	Real	69.35	6.93
NIN softmax	Random	Synth	45.60	14.26
NIN softmax	Synth	Real	69.09	6.98

Once again the first experiments have been performed to identify the most promising architectures (Table 5.5). We considered architectures based on the AlexNet and NIN models using a random initialization of weights. For NIN we considered one version with the global average pooling as last layer (NIN in Table 5.6) and one with a fully connected layer as last layer (NIN+FC in Table 5.6). In the last case, the weights are initialized to compute the average over the input features map. The best results are obtained with the NIN with global average pooling. We also considered one architecture with the output based on one softmax layer with 9 outputs corresponding to the number of classes (NIN softmax in Table 5.6). Not surprisingly, in the latter case, we obtained the worst results.

Also in this first experiment with the *Brandeburg* dataset the NIN architecture outperforms the AlexNet one confirming the results for the Esposalles collection. In almost all cases, the finetuning of a network initially trained on synthetic data improves the results on the test set concerning the training only on real data (without data augmentation).

One exception is the NIN softmax experiment. However, it is worth noticing that the *Accuracy* and *Error* are worst than in the other tests. Apart from the softmax NIN in all the cases the finetuning of a network trained with synthetic data has higher *Accuracy* and lower *Error* with respect to a network trained on real data only.

Since this collection is larger than the *Esposalles* one we considered also one experiment where all the training and test are made with real data and the networks

are pre-trained on Imagenet (Table 5.6). For the Alexnet and NIN architectures, the best results are obtained when finetuning with real data one network initially trained with synthetic data. On the other hand, the training with only real data on VGG16 pre-trained on Imagenet provides the best overall results.

Table 5.6: *Brandeburg* dataset. Comparison of different architectures (Imagenet initialization of weights).

Model	Initial weights	Training	Accuracy (%)	Error (%)
Alexnet	Imagenet	Real	70.55	5.26
Alexnet	Imagenet	Synth	55.58	10.09
Alexnet	Synth	Real	79.90	3.66
NIN	Imagenet	Real	82.51	2.98
NIN	Imagenet	Synth	57.59	9.14
NIN	Synth	Real	83.22	3.08
VGG16	Imagenet	Real	86.23	2.35
VGG16	Imagenet	Synth	60.20	7.46
VGG16	Synth	Real	85.93	2.48

Table 5.7: *Brandeburg* dataset. Varying the size of the data used for training or for finetuning.

Model	Measure(%)	Initial weights	Training	Trainset size 1/8	1/4	1/2	1/1
Alexnet	Accuracy	Imagenet	Real	44.22	51.15	57.39	70.55
		Synth	Real	65.93	71.96	77.89	79.90
	Error	Imagenet	Real	11.55	9.61	7.99	5.26
		Synth	Real	6.89	5.43	4.13	3.66
NIN	Accuracy	Imagenet	Real	56.08	61.81	73.57	82.51
		Synth	Real	67.44	71.76	76.18	83.22
	Error	Imagenet	Real	8.49	7.04	4.61	2.98
		Synth	Real	6.29	5.48	4.51	3.08
VGG16	Accuracy	Imagenet	Real	70.45	72.96	79.19	86.23
		Synth	Real	74.07	76.98	80.60	85.93
	Error	Imagenet	Real	5.56	4.89	3.60	2.35
		Synth	Real	4.83	4.26	3.46	2.48

One of the claims of this research is that when the number of training data available is limited, then the performance can be improved by using synthetic data. To investigate the effect of the number of training pages used for finetuning, in Table 5.7 we show the results of two of the experiments described in Table 5.6 (training with real data one network pre-trained on Imagenet and finetuning with real data one network trained on synthetic data) when using different sizes for the real training set. For example, the smallest training set contains 396 images (1/8 of the total 3,165 pages available for training) and the corresponding validation set contains 104 images. The test sets are left unchanged with respect to the experiments in Table 5.6.

Except for VGG16 trained with 100% of the data, in all the cases the finetuning of a network trained with synthetic data outperforms the same model only trained on real data. We can also notice that in general, the gap between the two approaches narrows when the size of the training set increases.

To provide additional information on the results we show in Figure 5.4 the confusion matrix for the results obtained with the VGG16 model. It is worth to notice that for nearly all the types of pages the results are in a range of ± 1 with respect to the correct record count.

To further analyze the results in Figure 5.5 we show some pages in the collection with five records. From these examples, we can notice how variable are the pages in the collection. The pages are grouped in the Figure according to the results achieved when training one VGG16 model with the two experiments summarized in Table 5.7. The number of records for pages in the first group is correctly identified with both approaches. On the other hand, the number of records for pages in the second group is not identified by either method. For instance, the first two pages in the group have a long comment at the bottom, while in the last one the first record has been canceled and therefore is not counted in the ground-truth.
For the groups in the bottom part of Figure 5.5 the two approaches provide different results. In the first two pages, the first record is very difficult to read and faded and this type of degradation was probably not properly modeled by the synthetic document generator. The latter tool was, however, able to model the irregular distribution of records in the last three examples, that are probably rare in real training pages.

5.5 Summary

In this chapter we have explained how to use a CNN-based architecture to count the number of records contained in handwritten documents. We can summarize this chapter as follows:

- Having structured documents it is useful to perform record counting to eval-

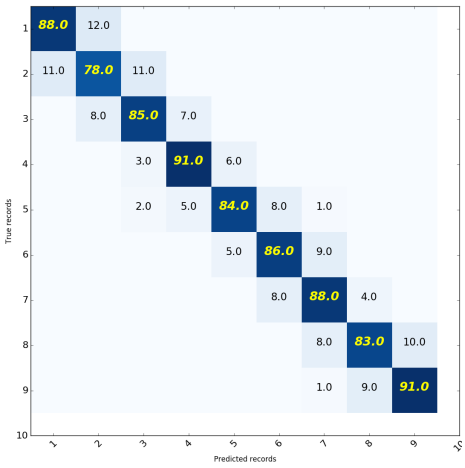


Figure 5.4: Confusion matrix for the best VGG16 model.

uate the collection content.

- It is possible to use CNN-based architecture to count the number of records contained inside a page only if we have enough labeled data for training.
- Using DocEmul is possible to produce synthetic data to train a deep network properly.
- Pre-trained networks and synthetic data to extend the original dataset are good combinations to train deep models to prevent over-fitting.

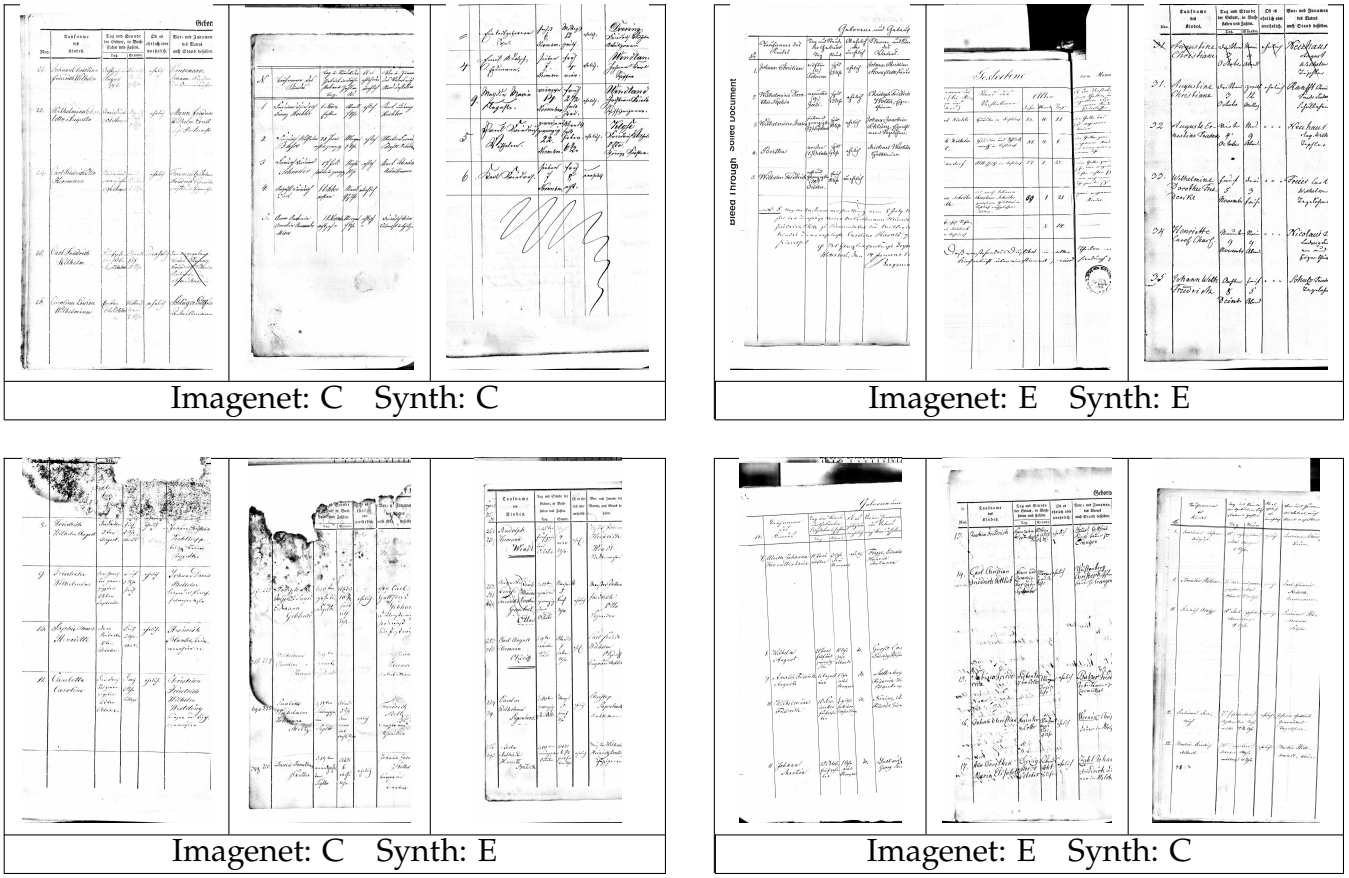


Figure 5.5: *Brandenburg* dataset. Examples of pages and record counting results for the VGG16 model and training starting from Imagenet or Synthetic initial weights (as in Table 5.7). C means a correct record count, while E means an error in the record count.

Chapter 6

Historical handwritten page analysis

Handwritten historical documents are widely available in Digital Libraries and archives. As we have seen in Chapter 5, understanding handwritten historical documents is a challenging task that includes several sub-problems. One of the first steps is to segment and extract text lines which could be recognized in subsequent phases to understand the document content. The layout analysis of handwritten documents can be very difficult, because of the variable layout structure, the presence of decorations, different writing styles and degradations due to the aging of documents. In the last years, different techniques have been proposed to address this task [24, 99, 119]. In particular, to extract text lines from handwritten documents we can consider two types of related problems. Considering the page segmentation task the target is to split a document image into regions of interest [24]. These approaches extract regions of interest which are considered as text lines and often provide similar results when considering handwritten documents.

Among several solutions proposed to solve this task, some use assumptions simplify the approach. In [90] the authors assume that for each text line there is one path from one side of the image to the other that crosses only one text line. Based on this assumption, they trace the text line after the blurred image transformation extracting directly the text lines. One old-fashioned technique to detect text lines in the historical document is based on the Run Length Smearing Algorithm (RLSA) [121]. In [92] the authors propose to extract text lines with a modified version of RLSA, in this case, they add connected components, whitespaces, punctuation marks, and skeleton of the strokes knowledge to the smearing process.

Later, it has been proposed another solution [41] where the authors can extract text lines from handwritten pages using Hough transform and the page structure as prior knowledge. Another great solution [8] has been proposed to segment text lines constructing distance transform directly on the gray-scale images and computing two types of seams (medial seams and separating seams) on that. The medial seam means the text area of a text line, the separating seam means the path that

passes between two consecutive rows. The medial and separating seams are computed using dynamic programming which relies on generating an energy map. This energy map is based on the constructed distance transform and encodes the minimal cost of the valid paths.

In Section 2.2, many different CNN architectures have been presented to solve several computer vision tasks. One important task is the image semantic segmentation whose goal is to classify each image pixel into one of several different classes highlighting homogeneous object regions. One interesting solution adopts a Fully Convolutional Network (FCN) [108] composed only by convolutional and pooling operations used to learn representations based on local spatial input to compute pixel-wise predictions. The FCNs concerning CNNs architectures do not use fully connected layers and use upsampling layers as deconvolutional operators.

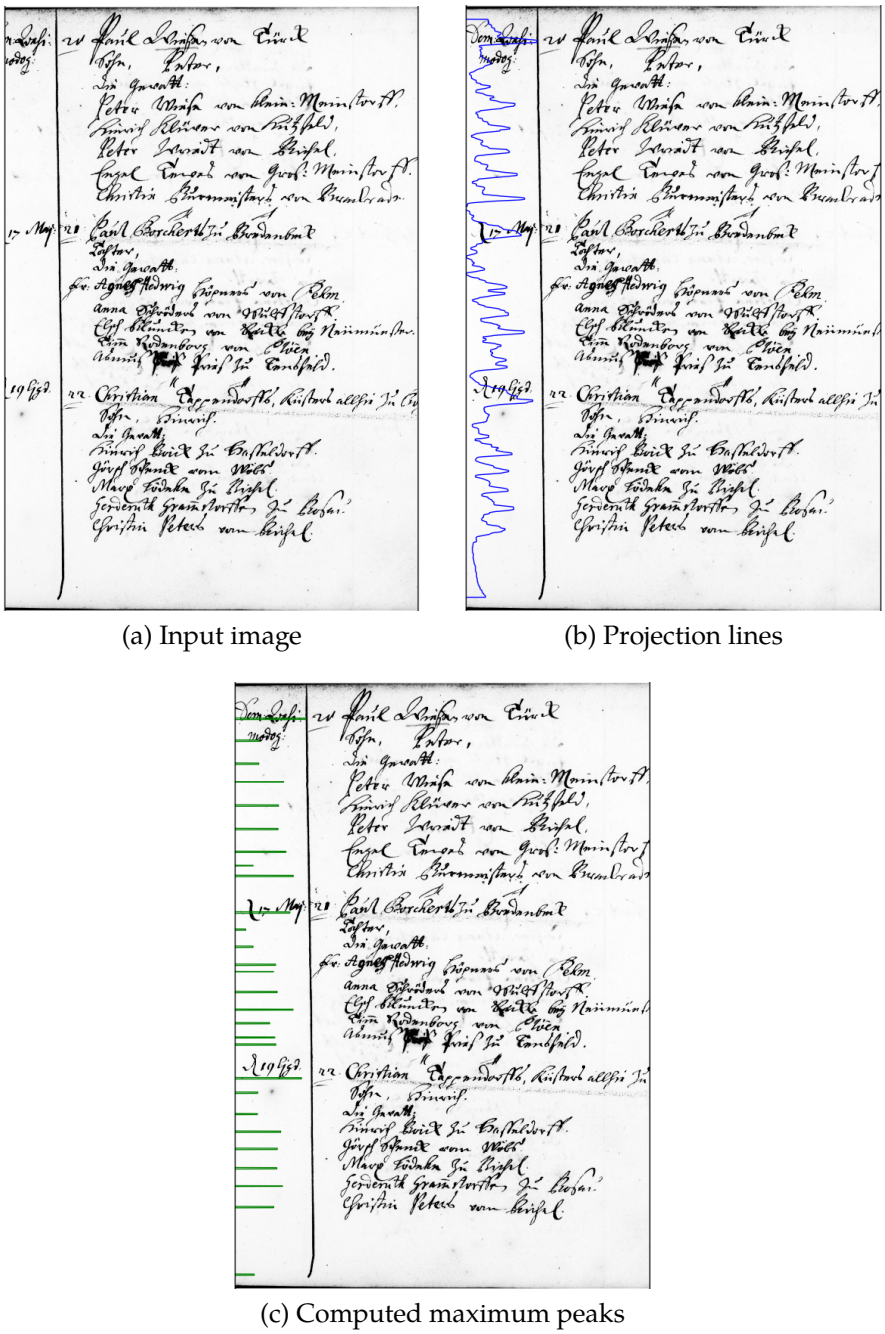
In this chapter, we show two different approaches to layout problems which use prior information to perform document analysis effectively.
The first approach addresses text line extraction. We define a CNN model to classify extracted patches from an input image to detect different document regions as text lines. This approach has been published in [21]
The second system is based on an FCN network to segment input image into document semantic regions. This system has been described in [23]

6.1 Patch-based system for text line extraction

In our research we deal with handwritten documents having different layouts. As we know, on each page, the distance between text lines is quite regular. We can use this prior knowledge to streamline the document analysis, then we can localize the text lines and the line separators by using a suitably trained neural network model.

Firstly, to train a patch-based CNN model we need to create a suitable document representation. To this purpose, for each training page, we estimate the average distance between text lines, then we extract rectangle random patches with height related to the previously estimated line distance. The extracted patches are labeled according to ground truth information and used to train the convolutional network. Subsequently, we design a text line separator. For each test page, we estimate the average distance between text lines and use this information to extract dense patches that are classified using the trained model. The classified patches are then combined to obtain an overall page segmentation.

In the following sections, we describe in detail how we estimate the patch dimension to create the dataset for training the CNN model. Then, we analyze the proposed application scenario and discuss the results obtained from experimental datasets.



(a) Input image

(b) Projection lines

(c) Computed maximum peaks

Figure 6.1: Steps to estimate the dsistance between text lines in one page.

Adaptive estimation of patch dimension

This text line segmentation approach is based on a patch-wise representation of document images. A patch is expected to cover an image region containing text or background. Therefore, in the pre-processing phase, we need to estimate the average distance between text lines. As we can notice in Figure 6.1 there is a certain regu-

larity in the distance between two contiguous text lines, and this regularity can be used to set the height of patch size for each page. Subsequently, we can define two semantic regions: a patch could be labeled as text line separator (the patch covers separators between contiguous text lines); a patch could be labeled as text (the text is contained in the patch).

Estimate average text line distance

To estimate the average text line distance we need to compute several steps. In the first step, we remove black bands at the top and bottom borders of the page. Then, we compute the projection profile along the vertical dimension and find its local maxima. So, we retain only the peaks larger than 20% of the maximum.

Considering these peaks, we can compute the distribution of distances between pairs of neighboring peaks. This set of values can be fitted by two centroids using the k-means algorithm. In the end, we take the minimum centroid which defines the average distance between lines. The main steps are depicted in Figure 6.1 where we show one input image, the computed projection profiles, and the peaks used to define the distance distribution.

With this procedure, it is possible to estimate the text line distance for each page. This distance is subsequently used to define the size of the patches extracted from the page.

Patch definition

After the average text line distance estimation, $\hat{h}$, we define the patch size (height, width) as $(\hat{h} \cdot \sqrt{2}, \frac{\hat{h}}{\sqrt{2}})$. The patch height is defined to capture almost two contiguous text lines with one separator (space or other) in between. The patch size has been chosen after some preliminary tests. In general, we can observe that the classification of variable-size patches is a good option to be independent of the text line distance in the input image. A dense patch extraction with overlapping can cover all the lines providing a good hint in the presence of text line separators.

Patch-based classification system

Taking into account the physical structure of the documents we can define two types of areas: the text line and the separator (page area between two contiguous text lines). For the subsequent steps, it is also useful to define the median line and the separator line. The median line on the text is the middle line between the top profile and the bottom profile. The separator line is the middle line between two consecutive median lines in the text area. We can see these lines in Figure 6.2, in particular, we show three text lines extracted from one page representing the top and bottom

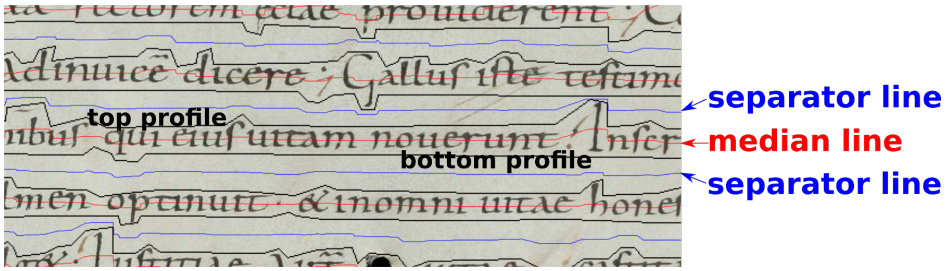


Figure 6.2: Examples of separator and median lines and the corresponding top and bottom profiles.

profiles with black lines, the separator and median lines in blue and red. These two types of lines can be used to label the extracted patches and subsequently train a CNN-based system for line segmentation task. However, the previous information is not always available in the ground truth of the document dataset. Therefore we need to compute this representation from the available ground truth data.

For example, the ground truth of the *Saint Gall* dataset (more details in Section 3.1) is available as pixel-label images defining the text area. In this case, we can compute the median line for each text line starting from the pixel-level information. After that, we compute the separator line as the midpoint between two consecutive median lines. In Figure 6.4 we can see one example of the *Saint Gall* dataset and the corresponding ground truth information. Using this information we can compute the previously defined guidelines to label the extracted patches.

Unfortunately, sometimes the ground truth data does not contain this pixel-level labeling. In these cases, we manually define only the separator lines and use this information to identify also text lines. More details on how to extract and label patches are described in Section 6.2. Using the document structure and the ground truth information, we can label each patch as separator, text, or background. The assigned class depends on the area covered by the patch on the document.

Network architecture

In Section 2.2 we have seen several convolutional network architectures, for this topic we have extended *Network in Network* [76] (NIN) model. This architecture (Figure 6.3) consists of some stacked blocks (*mlpconv* layer) where each block is composed by a classical convolutional operator followed by a Multi Layer Perceptron (Figure 2.4).

For this topic, we use a modified version of the original architecture. In the first *mlpconv* layer we have a convolutional layer with a kernel dimension of $3 \times 7 \times 7$ computing 96 feature maps with stride 2 and padding 3. In the second *mlpconv* layer we have a convolutional layer with a kernel dimension $96 \times 5 \times 5$ computing 256 feature maps with stride 1 and padding 2. In the third *mlpconv* layer we have

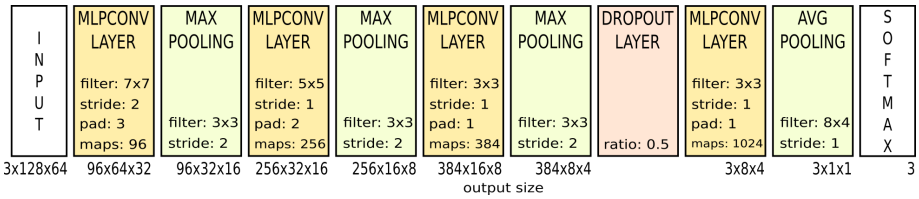


Figure 6.3: The architecture of the model. Different transformations are depicted with different colors. The *mlpconv* transformation is depicted in orange, the pooling undersampling in green and the input and output layer in white. The output result is composed by three classes.

a convolutional layer with a kernel dimension $256 \times 3 \times 3$ computing 384 feature maps with stride 1 and padding 1. After that, we have a dropout layer [56] which randomly selects 50% of input neurons during the training phase. In the fourth *mlpconv* layer we have a convolutional layer with a kernel dimension $384 \times 3 \times 3$ computing 1024 feature maps with stride 1 and padding 1. For the last *mlpconv* layer, we define the number of feature maps according to the number of classes in our task. We compute the global average pooling over the previous feature maps providing them to the softmax layer which models the distribution over the class labels. We use the stochastic gradient descent to train a model initialized with random weight values. The training is stopped on the best classification accuracy on a validation set. We use the Adam optimization algorithm [61] with a fixed learning rate of 0.0001 with momentum (0.9, 0.999) for the training phase.

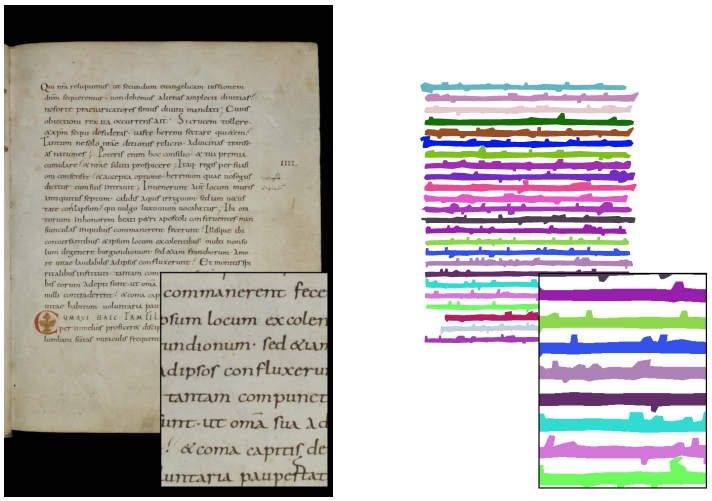
The overall architecture can be seen in Figure 6.3. The model is composed of four *mlpconv* and pooling layers. In orange, we denote the *mlpconv* transformation, in green the pooling transformation layer. For the last *mlpconv* we compute 1024 feature maps until the second inner product, instead, for the last inner product we have the number of neurons according to the output classes.

6.2 Evaluating patch-based system

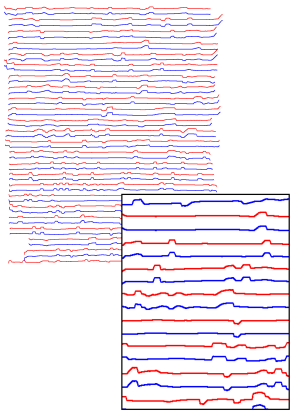
In this section, we describe the experiments on two different datasets comparing the results obtained by the proposed system.

Extracted patches

As previously mentioned, the first step is to extract patches needed to train the model. This extraction is based on the ground truth of the training set. As previously mentioned, for this research we consider two datasets having different ground truth information.



(a) sample (b) text lines

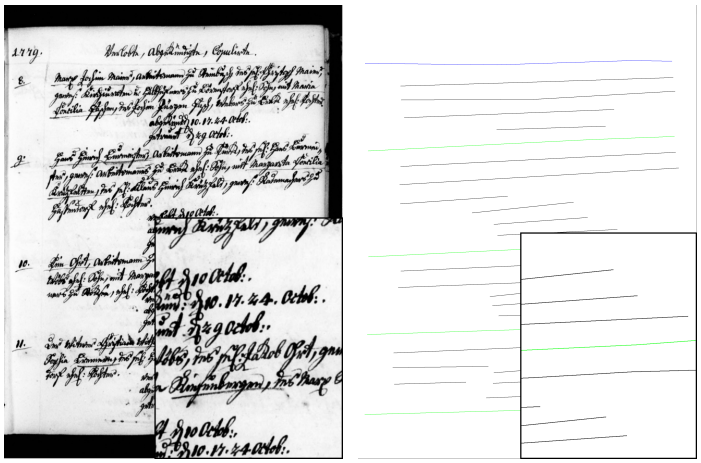


(c) median and separator lines

Figure 6.4: In (*a*) one page from the *Saint Gall* dataset, in (*b*) the corresponding ground truth pixel-wise representation for each text line, in (*c*) the computed median text lines in red and the separator text lines in blue.

In the *Saint Gall* dataset, we have one XML file for each document that describes very detailed layout information. The XML file contains a pixel-wise representation for each text line. One example of this dataset is presented in Figure 6.4 where we can see an input document, a pixel-wise representation of the ground truth for each text line, the computed median text line, and the text separator.

The ground truth produced for the *Branden-ancestry* dataset defines only the separator line between two adjacent text lines. In Figure 6.5 we can see one image example and its ground truth image.



(a) Input (b) Ground truth

Figure 6.5: An input image from *Branden-ancestry* collection with its ground truth image.

Considering these labeled datasets we can build a large patch dataset to learn the proposed CNN model. We have seen how to compute the average distance between text lines; using this information we can extract a patch with suitable dimensions to understand the document structure. Following the ground truth guidelines, in the first dataset, we can define patches for three classes (text, separator, and background) while only two classes can be considered for the second dataset. We can easily control the number of training patches for each class defining a uniform distribution of labeled patches. We then scale each extracted patch to a fixed size of 128×64 pixels.

In these two experiments for each class, we considered $100,000$ training patches and $30,000$ validation patches. We extract patches from each test image as described in the following sub-section. For instance in the *Saint Gall* dataset the number of patches for each page is around $600,000$ (the number is not fixed since the patches are not extracted for background areas).

Textline identification

The text line identification is made by extracting dense patches on the test images. Using a sliding window with the estimated dimensions, we classify the patches. The model used gives us an estimate of the probability for each class. This information can be used to infer the position of the lines in the image. We move the sliding window on the image with a stride size equal to 10% of the patch width and 1% of the patch height. In this way, we can compute several scores for the same position. We then compute the average probability along the horizontal direction concerning

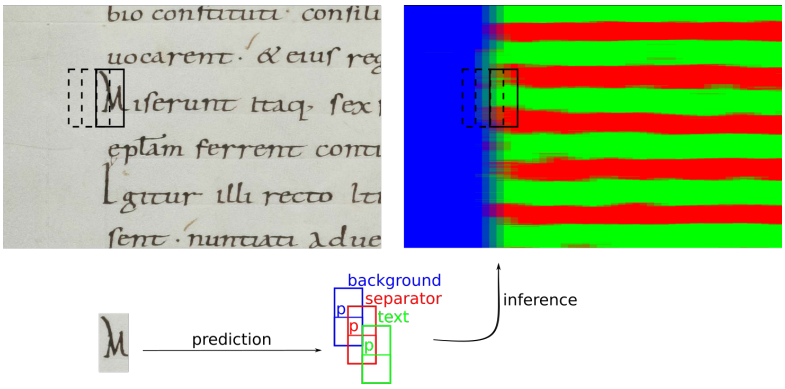
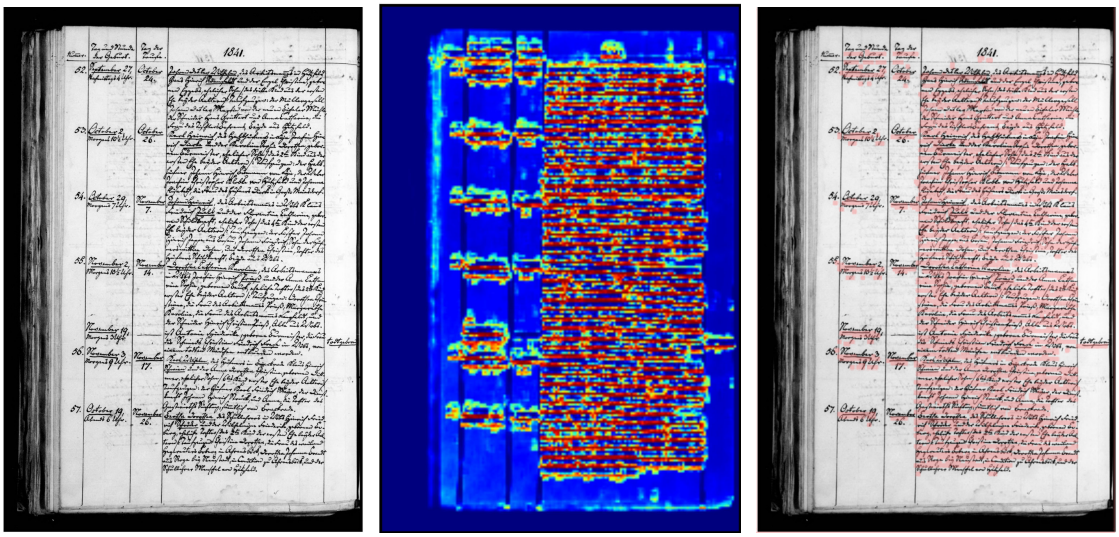


Figure 6.6: Textline identification: for each patch one prediction is computed and results are combined to segment the page into three classes: background in blue, text line in green and separator in red.



(a) input (b) probability map (c) results over input image

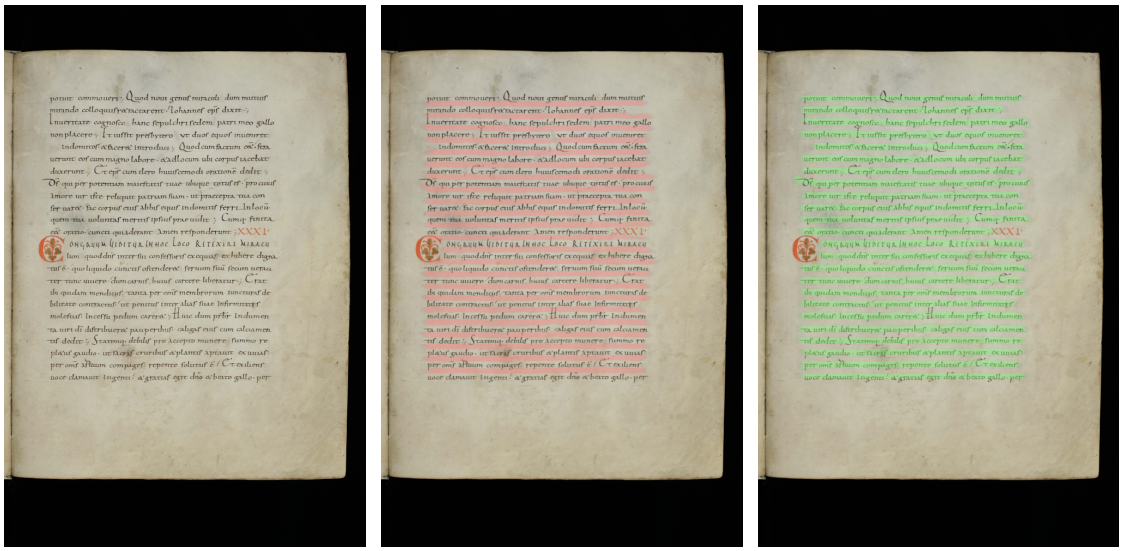
Figure 6.7: Probability Map to localize text line separators. In (*a*) the input image, in (*b*) the probability map computed by our framework and in (*c*) the results after a threshold over the input image.

the patch dimension. This approach is depicted in Figure 6.6 where we describe the test procedure. In particular, we can see how we extract dense patches form the input image and also how we compute the probability score for the whole page.

Preliminary Results

We present preliminary results obtained by our proposed system.

In the *Branden-ancestry* dataset, the ground truth considers only text line separators, therefore we can have only two classes: *text separator* and *no text separator*

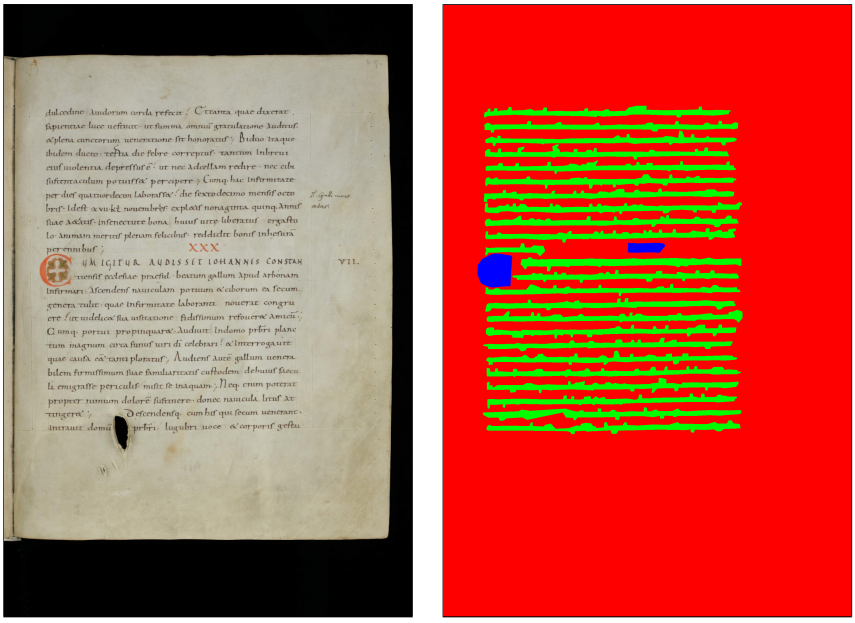


(a) input image (b) results for separator lines (c) results for text lines

Figure 6.8: Results obtained on the dataset with three classes. In (*a*) the input image, in (*b*) the results for separator lines and in (*c*) the results for text lines.

(background or text). The architecture used for this task is the same presented before. However, in the last *mlpconv* we have one output from two computed feature maps. After the testing phase, in Figure 6.7 we show the model results displaying only the probability score computed by the softmax operator. Moving a window over the input image, we compute a probability score for each position. We can define the text areas computing the average probability score as we can see in Figure 6.7*b*. This probability score represents the text line separator presence. The final result will be a probability map where we can discover the separator position. In this case, having only two classes, we define a global threshold on the probability density map predicting the text line separator. The prediction results considering a predefined global threshold can be seen in Figure 6.7*c* where we choose 0.7 as a threshold value to detect the text line separators.

In the HisDoc project, we have a more detailed ground truth where we can define median text lines and separator text lines. Using this information, we can use the proposed solution to predict three classes: text, separator, and background. With this information, we can infer the text and separator line positions. An example of this approach is shown in Figure 6.8 where we can see two different results, one for each class. In Figure 6.8*b* we highlight the area between text line (separator class), while in Figure 6.8*c* we we highlight the text line area (text line class).



(a) Original page (b) Original ground truth

Figure 6.9: One document example from the dataset and its ground truth image. The background area is in red, decoration in blue and text in green.

6.3 Pixel-wise page segmentation

In this section, we address the page segmentation using one Fully Convolutional Network and assigning different weights to the image pixels to compute an ad-hoc training loss designed to address the proposed task. In this way, we aim at classifying with better results in some areas of the image that are more critical to perform the text line extraction, without using dedicated post-processing techniques.

We segment historical handwritten pages by using one FCN-based network which is trained directly on document image (large) patches to produce a pixel-wise prediction to recognize different regions of interest in the input image. The documents addressed in our experiments have three different semantic classes: background, text, and decoration. An example of one input image with its ground truth can be seen in Figure 6.9, differently from Section 6.1, we use all the information from ground truth data to produce a pixel-wise classification.

Several architectures have been proposed to address the semantic segmentation. One model that gained attention in biomedical image segmentation is the *U-net* [101]. We propose a neural network that is strongly inspired by the *U-net* model to segment each page in different semantic regions. To map the features into a classification score we use the Softmax operator to predict the probability score related to the semantic segmentation. In particular, we compute pixel-wise classification scores to determine a class for each input pixel. In the basic approach, we use the

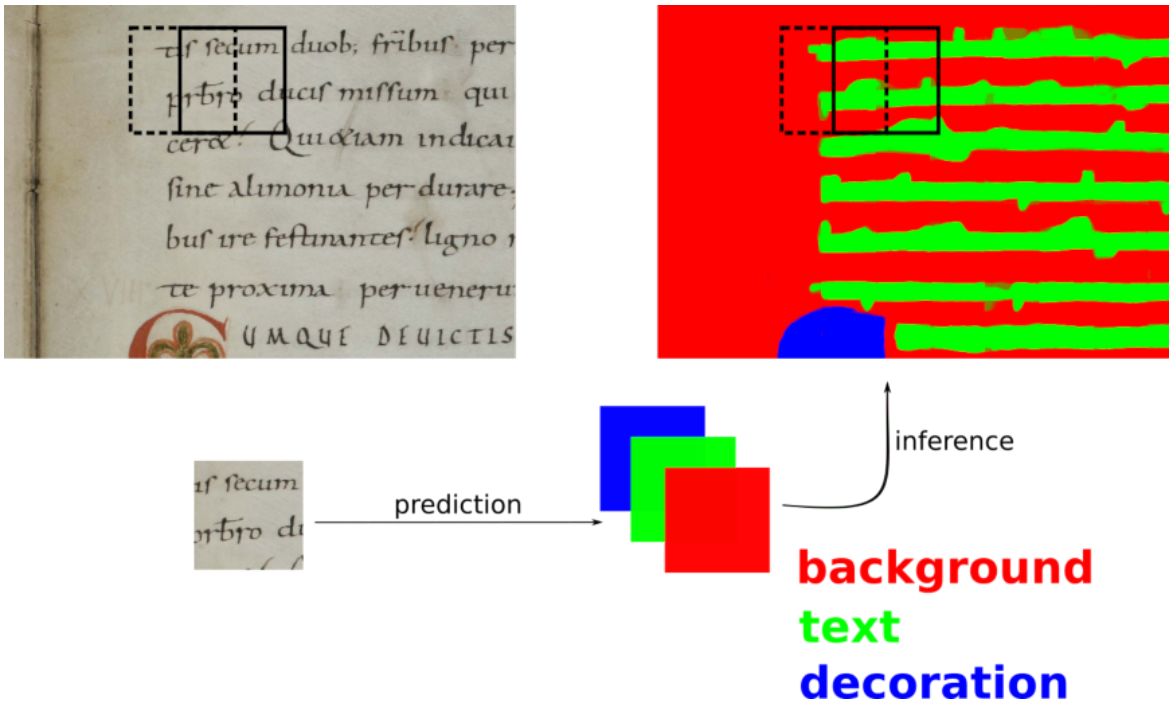


Figure 6.10: Moving a sliding window over the input image, we compute a pixel-wise classification score for each patch. The results are combined by averaging the scores of overlapping patches.

cross-entropy loss function to train the model from random weights initialized using the technique proposed by [45]. This loss function is then modified to take into account the peculiarities of the proposed problem which we want to explain in this section.

To build the training set we randomly crop several patches with a fixed size from each document image. To maximize the differences between training patches the maximum overlap between patches is set to 25%. Similarly to Section 6.1 for the testing phase, we systematically extract document patches from the input image with an overlap of 50%. For each pixel, the final prediction is the average of the probability scores computed by the neural network for all the overlapping probability maps as illustrated in Figure 6.10.

Weighting the loss function

Performing a good page segmentation is a difficult operation caused by several related issues. One significant problem is the unbalanced pixel class distribution. Having a look at Figure 6.9 we can see that the pixel distribution is highly unbalanced for background pixels with respect to the foreground pixels (considering foreground as text and decoration parts). We can notice also that some background pixels are very important to segment text lines. Often the text lines are very close to

each other and in this case, some misclassification errors of pixels between two text lines could give rise to significant problems for properly segmenting contiguous text lines.

The model is trained using a categorical cross entropy. One possibility to give different cost values to the input during the training is to add one weighted map to the loss function

$$WCE = -\sum_{x \in \Phi} w(x) \log p_{q(x)}(x) \tag{6.1}$$

where $\Phi \subset \mathbb{Z}^2$ is the set of pixel positions, $q : \Phi \rightarrow 1, \ldots, K$ maps input pixels to the class label of the predicted distribution p (K is the number of classes), $w : \Phi \rightarrow \mathbb{R}^+$ is the weight function that maps each pixel x to a suitable weight.

Considering Equation 6.1, we define a weighted map function $w(x)$ which assigns a cost to each pixel considering the class frequency and the contribution which could provide in the segmentation task. Considering the set of pixels Φ in the mini-batch used to compute the loss function, we define a weight map to balance the class frequency and also to put more attention in specific areas that are useful to segment different regions properly. Therefore, the weight map includes two aspects of the document, background and foreground areas. Formally, the weight map assigns to a pixel x one weight balancing the pixel class frequency with a factor α and managing the background pixels with a predefined weight mask $\beta(x)$ (described in the following section), such that:

$$w(x) = \begin{cases} \alpha & x \in \Phi_f \\ \beta(x) & x \in \Phi_b \end{cases} \tag{6.2}$$

where the foreground pixels $\Phi_f \subset \Phi$ represent text and decoration areas, while $\Phi_b \subset \Phi$ represent background pixels.

Considering Φ the set of pixels for each mini-batch, the foreground pixel frequency is a variable number (usually $|\Phi_f| < |\Phi_b|$). To balance the foreground areas, we apply a factor α as $\frac{|\Phi_b|}{|\Phi_f|}$ computed for each mini-batch. Having a pixel weight related to the class frequency we can balance the loss function improving the training.

As previously mentioned, not all background pixels have the same importance concerning the overall performance. In particular, misclassification errors between contiguous text lines could give rise to improper segmentation of the text lines. To address this problem, we define one training rule weighting more the background pixels between different regions (text lines or decorations). This topological constraint is a rule which could be defined directly into the weighted map (Equation 6.2) defining a weighed mask $\beta(x)$ for the background area as described in the following.

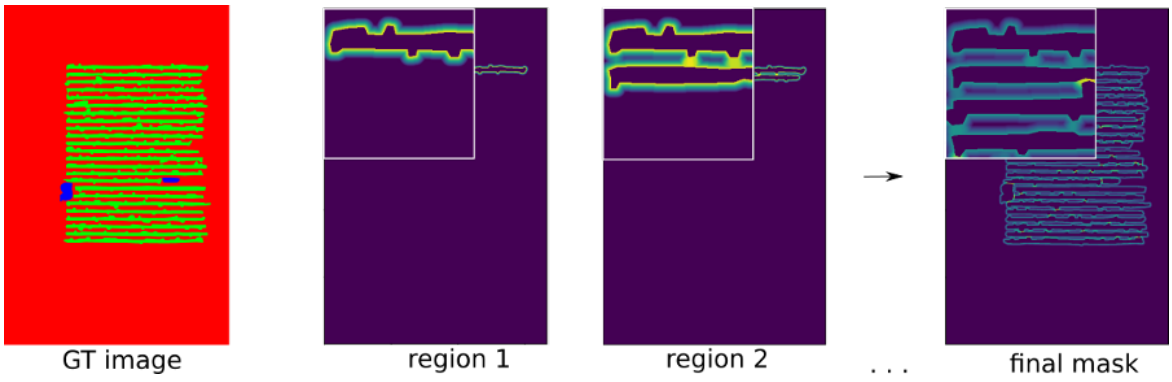


Figure 6.11: Creation of the weighted mask of a ground truth page. In the third image, after merging the GT for region 1 and region 2 we can see pixels closer to both regions give a larger contribution to the weight mask.

Weighting background pixels

The weighted mask $\beta(x)$ gives more emphasis on background pixels considering the distance between two contiguous lines. The background pixels have a classification cost inversely proportional to the distance between two contiguous text lines. To this purpose, the weight mask assigns to each background pixel one value considering the distance to the nearest line (a larger distance gives a smaller value and vice versa). For the other background pixels, the weight masks $\beta(x)$ returns a fixed (neutral) weight value.

To compute the weight mask $\beta(x)$ we first transform the ground truth image representation from three to two region categories by merging text and decorator as foreground and the rest are background pixels. Considering this representation, taking a foreground region per time, we compute the *distance transform* which designs level curves from the region borders to a defined maximum distance d. An example of these level curves is shown in Figure 6.11 (region one) where the level curve value (in false colors) decreases when increasing the distance from the closest region border.

These level curves encode one information useful to consider the distance to the nearest regions. Iteratively, computing a level curve for each region and summing-up these values we can produce an overall weight mask. In this way, when the regions are close to each other, the level curves are summed providing a larger value when the regions are closer. The largest value is obtained when the distance between the two regions is only one pixel. We force the range of values for the level curves to be between 0 and 1. By using a factor λ to multiply α (Section 6.3) we obtain that some pixels (close to text border) in the background area have the mask values larger than foreground regions.

Considering a binary representation I of the ground truth image, for each re-

gion r_i at time i, we compute the level curves on the basis of the distance transform $dist_d(r_i)$ limiting this representation until a max distance d. We can consider the area around all the region borders with a maximum distance of d is a dilation operation with kernel d.

In this way, the mask for an image with N regions is:

$$\beta(x) = \begin{cases} 1 + \frac{\lambda\alpha}{2d} \cdot (\sum_i^N dist_d(r_i)) & x \in dilate_d(I) \\ 1 & otherwise \end{cases} \tag{6.3}$$

where $dilate_d$ is the morphological dilation operator with kernel d useful to consider the area where the weight mask has a variable number. For the remaining pixels in the page, the weight mask maps pixels to a neutral value.

We illustrate in Figure 6.11 the approach to compute the weight mask. Starting from a ground truth image we compute a binary representation with foreground regions and background. For each region, we compute the distance curve levels as $dist(r_i)$ which are sequentially summed with the next region representations. The final result is the computed mask for all the pixels $x \in dilate_d(I)$ which are the critical pixels where we want to put more emphasis during the training to learn background representation. To provide a better idea about the critical pixels, in Figure 6.12 we highlighted in red the critical pixel areas.

6.4 Evaluating pixel-wise page segmentation

In this section we describe the experiments performed to test the previously proposed model to segment historical document images. The tests have been made on the *Saint Gall* dataset (Section 3.1).

We evaluate the model performance using four metrics applied to semantic segmentation proposed by [108]. These measures are based on pixel accuracy and region intersection over union (IU). In particular, we evaluate the performance using: pixel accuracy, mean pixel accuracy, mean IU, and frequency weighted IU (f.w. IoU).

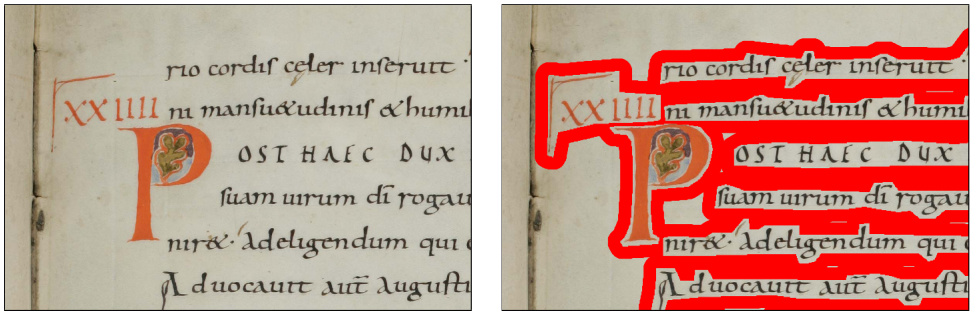
Let n_{ij} be the number of pixels of class i predicted to belong to class j (in total there are n_{cl} classes), and $t_i = \sum_j n_{ij}$ be the total number of pixels of class i. We can express the measures as:

- Pixel accuracy

$$pix.acc. = \frac{\sum_i n_{ii}}{\sum_i t_i} \tag{6.4}$$

- Mean accuracy

$$mean.acc = \frac{1}{n_{cl}} \sum_i \frac{n_{ii}}{t_i} \tag{6.5}$$



(a) input (b) crical pixels

Figure 6.12: Given a input page, we can define the critical pixel areas (red) around the semantic regions found into the ground truth representation.

Model	pix.acc.	mean.acc.	mean.IoU	f.w.IoU
Baseline (CE Loss)	98.07	95.12	90.80	96.27
Weighted (CE Loss)	98.03	94.75	90.88	96.18
Chen et al. [24]	98	90	87	96

Table 6.1: Results for overall performance measures.

- Mean IoU (Intersection over Union)

$$mean.IoU = \frac{1}{n_{cl}} \sum_i \frac{n_{ii}}{t_i + \sum_j (n_{ji} - n_{ii})} \tag{6.6}$$

- Frequency weighted IoU

$$f.w.IoU = \frac{1}{\sum_k t_k} \sum_i \frac{t_i n_{ii}}{t_i + \sum_j (n_{ji} - n_{ii})} \tag{6.7}$$

The previous metrics are used to define a global evaluation for whole pages. To better evaluate the performance, we also define one local pixel accuracy considering only the area around the foreground regions. In Figure 6.12 we depict in red the area around foreground regions where the local pixel accuracy is computed. This area is important to extract text lines because misclassification pixels in it could give rise to a wrong layout analysis.

In the experiments, we trained the proposed model using the extracted patches from the original training pages and learning the parameters by the Stochastic Gradient Descent algorithm. The training dataset is composed of several patches of size 256×256 pixels randomly extracted from the input pages. Overall the training dataset contains $299,756$ patches. We use the test set pages to evaluate the models comparing the proposed learning methods.

Model	Critical Pixel accuracy	DR	RA	FM
Baseline (CE Loss)	95.65	77.81	83.48	80.55
Weighted (CE Loss)	96.25	81.28	86.18	83.65

Table 6.2: Results for critical pixel classification and text lines extraction [95].

In Table 6.1 we report the results for page segmentation on the *Saint gall* dataset and compare with previous results on the same dataset reported by Chen et al. [24]. The proposed model obtains good results compared to [24] the use of the standard cross-entropy loss. We improve the mean IoU by using the proposed weighted loss. These metrics evaluate the page segmentation globally, but as we previously mentioned some misclassification errors have more importance in the final segmentation results.

The results reported in Table 6.2 show the critical pixel accuracy. This measure is useful to evaluate the models after the training by different losses. Using the proposed weighted loss function we can obtain better results which could be useful to extract text line directly after the page segmentation.

For a qualitative evaluation of results, we show in Figure 6.13 a part of one page and two results, one from a model trained with cross-entropy loss and the other from a model trained with the proposed weighted loss. We can notice that the model trained with the weighted map can better perform the text line segmentation task.

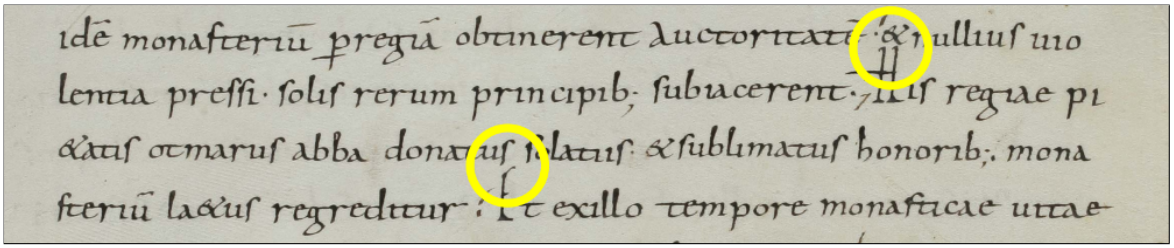
To evaluate the trained models with the measure proposed by [95] the model trained by weighted loss obtains Detection Rate (DR) and Recognition Accuracy (RA) respectively better than the model trained by cross-entropy loss. Comparing these scores in Table 6.2 the model trained by the proposed approach can extract more accurate text lines than the baseline model.

The proposed approaches are very good for a pixel-wise classification task; in the text line extraction, solutions based on deep architectures are not so good compared to an old-fashioned solution[8]. In [8] the author proposed a method to generate an energy map encoding the minimal cost of the global valid seams.

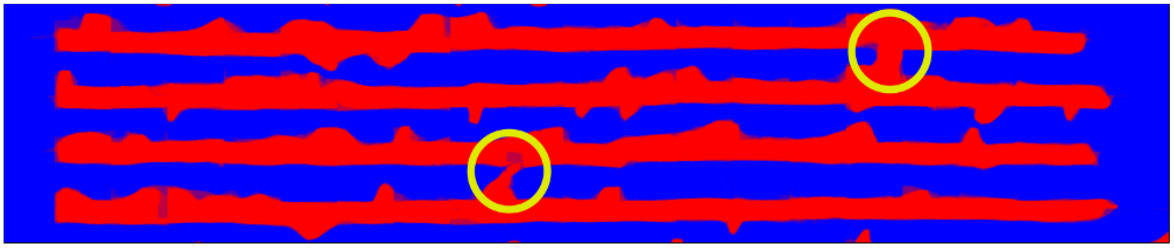
Recently, in [4] the authors propose a solution based on an FCN architecture with post-processing phase based on seam curving. We could extend our pipeline computing seam curving on the resulted feature maps, this post-processing step could help the system to extract text lines properly improving the overall performances.

6.5 Summary

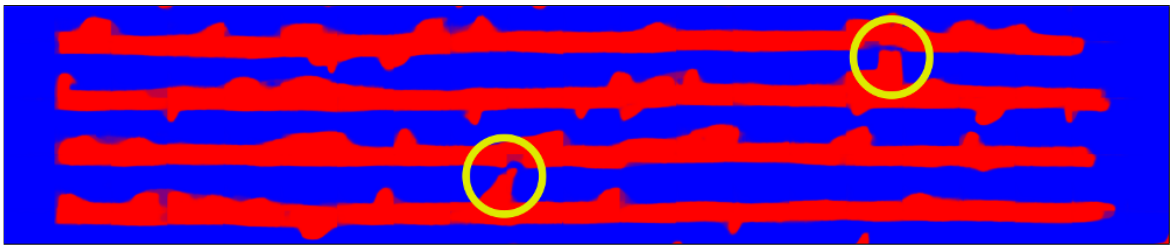
In this chapter, we presented our results for layout analysis on historical handwritten documents. The major points to recall are as follows:



(a) Input



(b) CE



(c) Weighted CE

Figure 6.13: Different results obtained with one model trained using cross entropy loss and one with weighted cross entropy loss,

- It is possible to extract whole text line from handwritten pages by using a patch-based system after an estimating the average distance between contiguous text lines.
- Patch-based convolutional network can classify patches into text or no text regions to detect text lines in historical documents.
- The preliminary results obtained by the patch-based system are a good starting point to extract text lines showing that CNNs are suitable for this task.
- Fully convolutional based networks can produce very good results for the page segmentation task
- Defining a domain-specific loss function we can weight pixel classification errors related to their position in page layout. In this way, we can guide the learning using prior information as soft-constraints.

Chapter 7

Conclusions

In this Ph.D. work, we have addressed document image understanding by using deep learning methods. We have investigated how to use these techniques on layout analysis of historical handwritten documents and on graphical document understanding. Thanks to a large number of parameters, deep networks can fit a variety of complex datasets. Unfortunately, for the proposed topics, the scarcity of labeled data for training deep networks in a supervised manner has been one of the major issues to be addressed in this research. We have proposed several solutions to solve overfitting by extending the labeled dataset or adapting the learning schema.

In Chapter 2 we presented a brief introduction on document image understanding applications followed by an explanation of recent deep learning methods for various application domains. Some of these architectures have been applied and adapted to our research. Chapter 3 was devoted to present the datasets used to evaluate the proposed methods. Also, suitable metrics have been presented to compare results guiding a better understanding of model performance.

In Chapter 4 we tackled the problem of data scarcity proposing an interactive annotating system that helps users for labeling floor plan images for object detection. We trained deep network architectures for object detection using our dataset showing good preliminary results. Our deep learning solution gives detection results in one step differently to old-fashioned solutions based on the segmentation-recognition paradigm. Still in Chapter 4 we proposed a dataset selection technique useful to create a better training data for layout analysis. Since it is important to reduce intra-class variance in datasets to improve the learning process, we exploited out page clustering technique for data selection. In the end, we presented a toolkit to generate synthetic handwritten documents using a few examples to emulate. With a similar aim, we proposed a generative model to produce floor plan symbols to be used for automatic floor plan generation. All these methods are useful to select or generate data to train deep architectures.

Chapter 5 described a solution for layout analysis in historical handwritten docu-

ments that addressed record counting. The obtained results demonstrated how the document generator proposed in Chapter 4 is able to produce synthetic documents to extend original training data useful to improve the performance of trained deep networks. We also compared different deep architectures showing their peculiarities.

In Chapter 6 we presented two different approaches for layout analysis in historical handwritten documents by segmenting text regions. The first was a CNN-based method for a patch-wise classification to extract contiguous text lines. The second was a FCN-based method to classify pixels in semantic categories to produce a complete page segmentation.

In this research, we have seen how deep learning methodologies could be considered as a general framework that could be adopted in different applications. Of course, the main factor in the learning is data, we have seen also how to work around this factor generating the data distribution which we want to capture in our prediction tasks. It is possible to use deep networks as general-purpose solutions that can be specified to a particular need adding ad-hoc operations in the same computational process.

7.1 Future works

Layout analysis in historical handwritten structured documents and its associated sub-tasks, such record counting, text line extraction and page analysis have gained interest from many researchers. As presented in this thesis, new toolkits and deep network methodologies show very promising generalization capabilities. However, there is still a long path to explore in order to extend the capability of actual automatic systems to reach a full document image understanding.

Related to layout analysis, we are extending the DocEmul toolkit to identify a technique to generate the XML configuration file semi-automatically starting from few real documents. Moreover, we are studying the information that can be extracted from the feature maps in the convolutional neural networks after the training on the produced synthetic data. This type of information can be of interest for understanding which parts of the input image are more informative to perform the record counting task. The latter study may lead at the end to address the record segmentation with deep convolutional networks.

Related to graphical document understanding, we will apply modern object detectors to our labeled floor plan images in order to add also the information related to the spatial relationship between objects trying to improve the results.

7.2 Contributions

In this Ph.D. dissertation, we have made both practical and methodological contributions to document image understanding, in terms of historical handwritten layout analysis and graphical document processing. We have studied the following aspects of learning deep networks methods for the proposed topics:

- an interactive method to help the user in the graphical document annotation process;
- clustering techniques on historical handwritten documents;
- rule-based system to emulate the handwriting process;
- generative model to produce graphical symbols;
- generating synthetic data to extend dataset with a small amount of labeled data;
- define custom loss with prior knowledge for page segmentation to prevent local errors;
- use deep architectures for object detection in floor plan images.

For document image understanding we dealt with the problems behind historical handwritten layout analysis and graphical document processing. For historical handwritten layout analysis, we proposed a novel solution to address the scarcity of labeled data for training deep networks. These contributions are summarized as follows:

- we proposed a novel method to generate synthetic handwritten and structured documents for training deep network architectures;
- we extended a small labeled dataset with generated handwritten documents for training deep architectures;
- we proposed a novel domain-specific training loss for page segmentation task;

For graphical documents the two major contributions could be summarized as follows:

- we implemented an interactive annotation tool to reduce the time for labeling documents;
- we made experiments on an object detection system for floor plan image;

Appendix A

Publications

The work of this dissertation has led to a number of peer-reviewed publications and service to scientific community.

Journal papers

1. Samuele Capobianco, Simone Marinai "Deep neural networks for record counting in historical handwritten documents.", *Pattern Recognition Letters*, pages:103–111, 2019. **Candidate's contributions**: designed algorithms, designed software toolkit, learning procedure

Peer reviewed conference papers

1. Simone Marinai, Samuele Capobianco, Zahra Ziran, Andrea Giuntini and Pierluigi Mansueto, "Recognition of Concordances for Indexing in Digital Libraries." *IRCDL,Accepted*, 2020. **Candidate's contributions**: document analysis

2. Samuele Capobianco, Simone Marinai "DocEmul: A Toolkit to Generate Structured Historical Documents." *ICDAR*, pages:1186–1191, 2017. **Candidate's contributions**: designed algorithms and software toolkit

3. Samuele Capobianco, Simone Marinai "Text Line Extraction in Handwritten Historical Documents." *IRCDL*, pages:68–79, 2017. **Candidate's contributions**: designed algorithms

4. Samuele Capobianco, Luca Facheris, Fabrizio Cuccoli and Simone Marinai "Vehicle Classification Based on Convolutional Networks Applied to FMCW Radar Signals." *TRAP*, pages:115–128, 2017. **Candidate's contributions**: designed algorithms

5. Muhammad Zeshan Afzal, Samuele Capobianco, Muhammad Imran Malik, Simone Marinai, Thomas M. Breuel, Andreas Dengel and Marcus Liwicki "Deep-

docclassifier: Document classification with deep Convolutional Neural Network." *ICDAR*, pages:1111–1115, 2015. **Candidate's contributions**: designed model for experimets

Workshop papers

1. Samuele Capobianco, Leonardo Scommegna, Simone Marinai "Historical Handwritten Document Segmentation by Using a Weighted Loss.", *ANNPR*, pages:395–406, 2018. **Candidate's contributions**: designed algorithms, desiged loss function

2. Samuele Capobianco, Simone Marinai "Record Counting in Historical Handwritten Documents with Convolutional Neural Networks.", *DLPR*, 2016. **Candidate's contributions**: designed algorithms

Papers under review

1. Anuradha Madugalla, Kim Marriott, Simone Marinai, Samuele Capobianco, Cagatay Goncu "Creating Accessible Online Floor Plans for Visually Impaired", *TACCESS*. **Candidate's contributions**: labeling system

Bibliography

[1] Abe, K., Iwana, B. K., Holmer, V. G., and Uchida, S. (2017). Font creation using class discriminative deep convolutional generative adversarial networks. In *ACPR 2017*, pages 232–237.

[2] Afzal, M. Z., Capobianco, S., Malik, M. I., Marinai, S., Breuel, T. M., Dengel, A., and Liwicki, M. (2015). Deepdocclassifier: Document classification with deep convolutional neural network. In *13th International Conference on Document Analysis and Recognition, ICDAR 2015, Nancy, France, August 23-26, 2015*, pages 1111–1115.

[3] Ahmed, S., Liwicki, M., Weber, M., and Dengel, A. (2011). Improved automatic analysis of architectural floor plans. In *2011 International Conference on Document Analysis and Recognition*, pages 864–869. IEEE.

[4] Alberti, M., Vögtlin, L., Pondenkandath, V., Seuret, M., Ingold, R., and Liwicki, M. (2019). Labeling, cutting, grouping: an efficient text line segmentation method for medieval manuscripts. *CoRR*, abs/1906.11894.

[5] Alvaro, F., Fernandez, F. C., Sánchez, J., Terrades, O. R., and Benedí, J. (2015). Structure detection and segmentation of documents using 2d stochastic context-free grammars. *Neurocomputing*, 150:147–154.

[6] An, G. (1996). The effects of adding noise during backpropagation training on a generalization performance. *Neural Computation*, 8(3):643–674.

[7] Arora, S., Bhaskara, A., Ge, R., and Ma, T. (2014). Provable bounds for learning some deep representations. In *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, pages 584–592.

[8] Asi, A., Saabni, R., and El-Sana, J. (2011). Text line segmentation for gray scale historical document images. In *Proceedings of the 2011 Workshop on Historical Document Imaging and Processing, HIP@ICDAR 2011, Beijing, China, September 16-17, 2011*, pages 120–126.

[9] Baird, H. S. (2007). The state of the art of document image degradation modelling. In *Digital Document Processing*, pages 261–279. Springer.

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

[11] Becker, C. and Duretec, K. (2013). Free benchmark corpora for preservation experiments: Using model-driven engineering to generate data sets. In *Proceedings of the 13th ACM/IEEE-CS Joint Conference on Digital Libraries*, JCDL '13, pages 349–358. ACM.

[12] Bengio, Y., Courville, A., and Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE Trans. Pattern Anal. Mach. Intell.*, 35(8):1798–1828.

[13] Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166.

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

[15] Bhattacharya, U., Plamondon, R., Dutta Chowdhury, S., Goyal, P., and Parui, S. K. (2017). A sigma-lognormal model-based approach to generating large synthetic online handwriting sample databases. *International Journal on Document Analysis and Recognition (IJDAR)*, 20(3):155–171.

[16] Blei, D. M., Kucukelbir, A., and McAuliffe, J. D. (2016). Variational inference: A review for statisticians. *CoRR*, abs/1601.00670.

[17] Bulacu, M., van Koert, R., Schomaker, L., and van der Zant, T. (2007). Layout analysis of handwritten historical documents for searching the archive of the cabinet of the dutch queen. In *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*, volume 1, pages 357–361.

[18] Capobianco, S. and Marinai, S. (2016). Record counting in historical handwritten documents with convolutional neural networks. In *Proc. 1st Int.l Workshop on Deep Learning for Pattern Recognition, DLPR*, volume abs/1610.07393.

[19] Capobianco, S. and Marinai, S. (2017a). Docemul: a toolkit to generate structured historical documents. In *14th IAPR International Conference on Document Analysis and Recognition, ICDAR 2017, Kyoto, Japan, November 9-15*.

[20] Capobianco, S. and Marinai, S. (2017b). Docemul: a toolkit to generate structured historical documents. `https://github.com/scstech85/DocEmul`.

[21] Capobianco, S. and Marinai, S. (2017c). Text line extraction in handwritten historical documents. In *Digital Libraries and Archives - 13th Italian Research Conference on Digital Libraries, IRCDL 2017, Modena, Italy, January 26-27, 2017, Revised Selected Papers*, pages 68–79.

[22] Capobianco, S. and Marinai, S. (2019). Deep neural networks for record counting in historical handwritten documents. *Pattern Recognition Letters*, 119:103–111.

[23] Capobianco, S., Scommegna, L., and Marinai, S. (2018). Historical handwritten document segmentation by using a weighted loss. In *Artificial Neural Networks in Pattern Recognition - 8th IAPR TC3 Workshop, ANNPR 2018, Siena, Italy, September 19-21, 2018, Proceedings*, pages 395–406.

[24] Chen, K., Seuret, M., Hennebert, J., and Ingold, R. (2017). Convolutional neural networks for page segmentation of historical document images. In *14th IAPR International Conference on Document Analysis and Recognition, ICDAR 2017, Kyoto, Japan, November 9-15, 2017*, pages 965–970.

[25] Chhabra, A. K. (1997). Graphic symbol recognition: An overview. In *Graphics Recognition, Algorithms and Systems, Second International Workshop, GREC'97, Nancy, France, August 22-23, 1997, Selected Papers*, pages 68–79.

[26] Cireşan, D. C., Meier, U., Masci, J., Gambardella, L. M., and Schmidhuber, J. (2011). Flexible, high performance convolutional neural networks for image classification. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Two*, IJCAI'11, pages 1237–1242. AAAI Press.

[27] Cordella, L. P. and Vento, M. (2000). Symbol recognition in documents: a collection of techniques? *IJDAR*, 3(2):73–88.

[28] Cruz, F. and Terrades, O. R. (2014). Em-based layout analysis method for structured documents. In *Pattern Recognition (ICPR), 2014 22nd International Conference on*, pages 315–320.

[29] Damelin, S. B. and Hoang, N. S. (2017). On surface completion and image inpainting by biharmonic functions: Numerical aspects. *CoRR*, eprint arXiv:1707.06567.

[30] de França Pereira e Silva, G., Lins, R. D., and Gomes, C. (2014). Automatic training set generation for better historic document transcription and compression. In *11th IAPR International Workshop on Document Analysis Systems, DAS 2014, Tours, France, April 7-10, 2014*, pages 277–281.

[31] de las Heras, L.-P., Terrades, O. R., Robles, S., and Sánchez, G. (2015). Cvc-fp and sgt: a new database for structural floor plan analysis and its groundtruthing

tool. *International Journal on Document Analysis and Recognition (IJDAR)*, 18(1):15–30.

[32] Delalandre, M., Valveny, E., Pridmore, T., and Karatzas, D. (2010). Generation of synthetic documents for performance evaluation of symbol recognition & spotting systems. *International Journal on Document Analysis and Recognition (IJDAR)*, 13(3):187–207.

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

[34] Donahue, J., Jia, Y., Vinyals, O., Hoffman, J., Zhang, N., Tzeng, E., and Darrell, T. (2014). Decaf: A deep convolutional activation feature for generic visual recognition. In *International Conference in Machine Learning (ICML)*.

[35] Dutta, A., Lladós, J., and Pal, U. (2013). A symbol spotting approach in graphical documents by hashing serialized graphs. *Pattern Recognition*, 46(3):752–768.

[36] Erhan, D., Bengio, Y., Courville, A., Manzagol, P.-A., Vincent, P., and Bengio, S. (2010). Why does unsupervised pre-training help deep learning? *J. Mach. Learn. Res.*, 11:625–660.

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

[38] Fernández, D., Marinai, S., Lladós, J., and Fornés, A. (2013). Contextual word spotting in historical manuscripts using markov logic networks. In *Proceedings of the 2nd International Workshop on Historical Document Imaging and Processing, HIP@ICDAR 2013, Washington, DC, USA, August 24, 2013*, pages 36–43.

[39] Fischer, A., Bunke, H., Naji, N., Savoy, J., Baechler, M., and Ingold, R. (2014). *The HisDoc project. Automatic analysis, recognition, and retrieval of handwritten historical documents for digital libraries*, volume 38, page 91–106. De Gruyter.

[40] Fischer, A., Frinken, V., Fornés, A., and Bunke, H. (2011). Transcription alignment of latin manuscripts using hidden markov models. In *Proceedings of the 2011 Workshop on Historical Document Imaging and Processing*, HIP '11, pages 29–36, New York, NY, USA. ACM.

[41] Gatos, B., Louloudis, G., and Stamatopoulos, N. (2014). Segmentation of historical handwritten documents into text zones and text lines. In *14th International Conference on Frontiers in Handwriting Recognition, ICFHR 2014, Crete, Greece, September 1-4, 2014*, pages 464–469.

[42] Giro-i Nieto, X., Camps, N., and Marques, F. (2010). Gat: a graphical annotation tool for semantic regions. *Multimedia Tools and Applications*, 46(2):155–174.

[43] Girshick, R. (2015). Fast r-cnn. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, ICCV '15, pages 1440–1448, Washington, DC, USA. IEEE Computer Society.

[44] Girshick, R., Donahue, J., Darrell, T., and Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '14, pages 580–587, Washington, DC, USA. IEEE Computer Society.

[45] Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS'10). Society for Artificial Intelligence and Statistics*.

[46] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

[47] Goodfellow, I., Lee, H., Le, Q. V., Saxe, A., and Ng, A. Y. (2009). Measuring invariances in deep networks. In Bengio, Y., Schuurmans, D., Lafferty, J. D., Williams, C. K. I., and Culotta, A., editors, *Advances in Neural Information Processing Systems 22*, pages 646–654. Curran Associates, Inc.

[48] Goodfellow, I. J. (2013). Piecewise linear multilayer perceptrons and dropout. *CoRR*, abs/1301.5088.

[49] Goodfellow, I. J., Warde-Farley, D., Mirza, M., Courville, A., and Bengio, Y. (2013). Maxout networks. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, page III–1319–III–1327. JMLR.org.

[50] Gupta, A., Vedaldi, A., and Zisserman, A. (2016). Synthetic data for text localisation in natural images. In *2016 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2315–2324.

[51] Harley, A. W., Ufkes, A., and Derpanis, K. G. (2015). Evaluation of deep convolutional nets for document image classification and retrieval. In *13th International Conference on Document Analysis and Recognition, ICDAR 2015, Tunis, Tunisia, August 23-26, 2015*, pages 991–995.

[52] He, K., Gkioxari, G., Dollár, P., and Girshick, R. (2017). Mask r-cnn. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2980–2988.

[53] He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778.

[54] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778.

[55] Héroux, P., Barbu, E., Adam, S., and Trupin, É. (2007). Automatic ground-truth generation for document image analysis and understanding. In *9th International Conference on Document Analysis and Recognition (ICDAR 2007), 23-26 September, Curitiba, Paraná, Brazil*, pages 476–480.

[56] Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580.

[57] Hubel, D. H. and Wiesel, T. N. (1959). Receptive fields of single neurons in the cat's striate cortex. *Journal of Physiology*, 148:574–591.

[58] Jarrett, K., Kavukcuoglu, K., and Lecun, Y. (2009). What is the best multi-stage architecture for object recognition?

[59] Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., and Darrell, T. (2014). Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22Nd ACM International Conference on Multimedia*, MM '14, pages 675–678, New York, NY, USA. ACM.

[60] Joutel, G., Eglin, V., Bres, S., and Emptoz, H. (2007). Curvelets based queries for cbir application in handwriting collections. In *ICDAR*, pages 649–653. IEEE Computer Society.

[61] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

[62] Kingma, D. P. and Welling, M. (2014). Auto-encoding variational bayes. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*.

[63] Konidaris, T., Gatos, B., Ntzios, K., Pratikakis, I., Theodoridis, S., and Perantonis, S. J. (2007). Keyword-guided word spotting in historical printed documents using synthetic data and user feedback. *International Journal of Document Analysis and Recognition (IJDAR)*, 9(2-4):167–177.

[64] Konwer, A., Bhunia, A. K., Bhowmick, A., Bhunia, A. K., Banerjee, P., Roy, P. P., and Pal, U. (2018). Staff line removal using generative adversarial networks. In *ICPR 2018*, pages 1103–1108.

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

[66] Kullback, S. and Leibler, R. A. (1951). On information and sufficiency. *Ann. Math. Statist.*, 22(1):79–86.

[67] Kumar, J., Ye, P., and Doermann, D. S. (2012). Learning document structure for retrieval and classification. In *ICPR*, pages 1558–1561. IEEE Computer Society.

[68] Larochelle, H., Bengio, Y., Louradour, J., and Lamblin, P. (2009). Exploring strategies for training deep neural networks. *J. Mach. Learn. Res.*, 10:1–40.

[69] Le Kang, Jayant Kumar, Peng Ye, Yi Li, and David Doermann (2014). Convolutional Neural Networks for Document Image Classification. In *International Conference on Pattern Recognition (ICPR 2014)*, pages 3168–3172.

[70] Lecun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436–444.

[71] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324.

[72] Lee, J., Shridhar, K., Hayashi, H., Iwana, B. K., Kang, S., and Uchida, S. (2019). Probact: A probabilistic activation function for deep neural networks. *arXiv preprint arXiv:1905.10761*.

[73] Lempitsky, V. and Zisserman, A. (2010). Learning to count objects in images. In Lafferty, J. D., Williams, C. K. I., Shawe-Taylor, J., Zemel, R. S., and Culotta, A., editors, *Advances in Neural Information Processing Systems 23*, pages 1324–1332. Curran Associates, Inc.

[74] Lewis, D., Agam, G., Argamon, S., Frieder, O., Grossman, D., and Heard, J. (2006). Building a test collection for complex document information processing. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '06, pages 665–666, New York, NY, USA. ACM.

[75] Likforman-Sulem, L., Zahour, A., and Taconet, B. (2007). Text line segmentation of historical documents: A survey. *Int. J. Doc. Anal. Recognit.*, 9(2):123–138.

[76] Lin, M., Chen, Q., and Yan, S. (2014a). Network in network. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*.

[77] Lin, T., Dollár, P., Girshick, R. B., He, K., Hariharan, B., and Belongie, S. J. (2017). Feature pyramid networks for object detection. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 936–944.

[78] Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L. (2014b). Microsoft coco: Common objects in context. In Fleet, D., Pajdla, T., Schiele, B., and Tuytelaars, T., editors, *Computer Vision – ECCV 2014*, pages 740–755, Cham. Springer International Publishing.

[79] Liu, C., Wu, J., Kohli, P., and Furukawa, Y. (2017). Raster-to-vector: Revisiting floorplan transformation. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, pages 2214–2222.

[80] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S. E., Fu, C., and Berg, A. C. (2016). SSD: single shot multibox detector. In *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part I*, pages 21–37.

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

[82] Luyen, D. T., Carel, E., Ogier, J. M., and Burie, J. C. (2015). A character degradation model for color document images. In *Document Analysis and Recognition (ICDAR), 2015 13th International Conference on*, pages 806–810.

[83] Madugalla, A., Marriott, K., and Marinai, S. (2017). Partitioning open plan areas in floor plans. In *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, volume 1, pages 47–52. IEEE.

[84] Madugalla, A., Marriott, K., Marinai, S., Capobianco, S., and Goncu, C. (2019). Creating accessible online floor plans for visually impaired readers. *ACM Transactions on Accessible Computing*, under review.

[85] Manning, C. D., Raghavan, P., and Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA.

[86] Marinai, S. (2008). *Introduction to Document Analysis and Recognition*, pages 1–20. Springer Berlin Heidelberg, Berlin, Heidelberg.

[87] Marinai, S., Gori, M., and Soda, G. (2005). Artificial neural networks for document analysis and recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 27(1):23–35.

[88] Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In Fürnkranz, J. and Joachims, T., editors, *ICML*, pages 807–814. Omnipress.

[89] Neubeck, A. and Van Gool, L. (2006). Efficient non-maximum suppression. In *Proceedings of the 18th International Conference on Pattern Recognition - Volume 03*, ICPR '06, pages 850–855, Washington, DC, USA. IEEE Computer Society.

[90] Nicolaou, A. and Gatos, B. (2009). Handwritten text line segmentation by shredding text into its lines. In *10th International Conference on Document Analysis and Recognition, ICDAR 2009, Barcelona, Spain, 26-29 July 2009*, pages 626–630.

[91] Nielson, H. and Barrett, W. (2006). Consensus-based table form recognition of low-quality historical documents. *International Journal of Document Analysis and Recognition (IJDAR)*, 8(2):183–200.

[92] Nikolaou, N., Makridis, M., Gatos, B., Stamatopoulos, N., and Papamarkos, N. (2010). Segmentation of historical machine-printed documents using adaptive run length smoothing and skeleton segmentation paths. *Image Vision Comput.*, 28(4):590–604.

[93] Odena, A., Dumoulin, V., and Olah, C. (2016). Deconvolution and checkerboard artifacts. *Distill*.

[94] Otsu, N. (1979). A Threshold Selection Method from Gray-level Histograms. *IEEE Transactions on Systems, Man and Cybernetics*, 9(1):62–66.

[95] Pastor-Pellicer, J., Afzal, M. Z., Liwicki, M., and Bleda, M. J. C. (2016). Complete system for text line extraction using convolutional neural networks and watershed transform. In *12th IAPR Workshop on Document Analysis Systems, DAS 2016, Santorini, Greece, April 11-14, 2016*, pages 30–35.

[96] Rahnemoonfar, M. and Sheppard, C. (2017). Deep count: Fruit counting based on deep simulated learning. *Sensors*, 17(4):905.

[97] Ranzato, M., Boureau, Y., and LeCun, Y. (2007). Sparse feature learning for deep belief networks. In *NIPS 2007*, pages 1185–1192.

[98] Ren, S., He, K., Girshick, R. B., and Sun, J. (2017). Faster R-CNN: towards real-time object detection with region proposal networks. *IEEE Trans. Pattern Anal. Mach. Intell.*, 39(6):1137–1149.

[99] Renton, G., Chatelain, C., Adam, S., Kermorvant, C., and Paquet, T. (2017). Handwritten text line segmentation using fully convolutional network. In *First Workshop of Machine Learning, 14th IAPR International Conference on Document Analysis and Recognition, WML@ICDAR 2017, Kyoto, Japan, November 9-15, 2017*, pages 5–9.

[100] Romero, V., Fornés, A., Serrano, N., Sánchez, J.-A., Toselli, A. H., Frinken, V., Vidal, E., and Lladós, J. (2013). The esposalles database: An ancient marriage license corpus for off-line handwriting recognition. *Pattern Recognition*.

[101] Ronneberger, O., Fischer, P., and Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention - MICCAI 2015 - 18th International Conference Munich, Germany, October 5 - 9, 2015, Proceedings, Part III*, pages 234–241.

[102] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1988). Neurocomputing: Foundations of research. chapter Learning Representations by Back-propagating Errors, pages 696–699. MIT Press, Cambridge, MA, USA.

[103] Rusiñol, M. and Lladós, J. (2007). A region-based hashing approach for symbol spotting in technical documents. In *Graphics Recognition. Recent Advances and New Opportunities, 7th International Workshop, GREC 2007, Curitiba, Brazil, September 20-21, 2007. Selected Papers*, pages 104–113.

[104] Russell, B. C., Torralba, A., Murphy, K. P., and Freeman, W. T. (2008). Labelme: A database and web-based tool for image annotation. *International Journal of Computer Vision*, 77(1):157–173.

[105] Sauvola, J. J. and Pietikäinen, M. (2000). Adaptive document image binarization. *Pattern Recognition*, 33(2):225–236.

[106] Segui, S., Pujol, O., and Vitria, J. (2015). Learning to count with deep object features. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*.

[107] Shalev-Shwartz, S. and Ben-David, S. (2014). *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, New York, NY, USA.

[108] Shelhamer, E., Long, J., and Darrell, T. (2017). Fully convolutional networks for semantic segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 39(4):640–651.

[109] Simon, M., Rodner, E., and Denzler, J. (2014). Part detector discovery in deep convolutional neural networks. In *ACCV*.

[110] Simonyan, K., Vedaldi, A., and Zisserman, A. (2013). Deep inside convolutional networks: Visualising image classification models and saliency maps. *CoRR*, abs/1312.6034.

[111] Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. In *Proceedings of ICLR 2014*, volume abs/1409.1556.

[112] Smith, R. (2007). An overview of the tesseract ocr engine. In *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*, volume 2, pages 629–633.

[113] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *CVPR 2015*.

[114] Tombre, K. and Lamiroy, B. (2003). Graphics recognition - from re-engineering to retrieval. In *Proceedings of the Seventh International Conference on Document Analysis and Recognition - Volume 1*, ICDAR '03, page 148, USA. IEEE Computer Society.

[115] Tombre, K., Tabbone, S., and Dosch, P. (2005). Musings on symbol recognition. In *Graphics Recognition. Ten Years Review and Future Perspectives, 6th Internation Workshop, GREC 2005, Hong Kong, China, August 25-26, 2005, Revised Selected Papers*, pages 23–34.

[116] Uijlings, J., van de Sande, K., Gevers, T., and Smeulders, A. (2013). Selective search for object recognition. *International Journal of Computer Vision*.

[117] van der Walt, S., Schönberger, J. L., Nunez-Iglesias, J., Boulogne, F., Warner, J. D., Yager, N., Gouillart, E., Yu, T., and the scikit-image contributors (2014). scikit-image: image processing in Python. *PeerJ*, 2:e453.

[118] Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., and Manzagol, P. (2010). Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *JMLR*, 11:3371–3408.

[119] Vo, Q. N., Kim, S., Yang, H. J., and Lee, G. (2018). Text line segmentation using a fully convolutional network in handwritten document images. *IET Image Processing*, 12(3):438–446.

[120] Wada, K. (2016). labelme: Image Polygonal Annotation with Python. `https://github.com/wkentaro/labelme`.

[121] Wahl, F. M., Wong, K. Y., and Casey, R. G. (1982). Block segmentation and text extraction in mixed text/image documents. *Computer Graphics and Image Processing*, 20:375–390.

[122] Walach, E. and Wolf, L. (2016). *Learning to Count with CNN Boosting*, pages 660–676. Springer International Publishing, Cham.

[123] Wan, L., Zeiler, M., Zhang, S., Cun, Y. L., and Fergus, R. (2013). Regularization of neural networks using dropconnect. In Dasgupta, S. and Mcallester, D., editors, *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, volume 28, pages 1058–1066. JMLR Workshop and Conference Proceedings.

[124] Wang, C., Zhang, H., Yang, L., Liu, S., and Cao, X. (2015). Deep people counting in extremely dense crowds. In *Proc. 23rd ACM Int.l Conf. Multimedia*, MM '15, pages 1299–1302.

[125] Wong, Y.-S., Chu, H.-K., and Mitra, N. J. (2015). Smartannotator an interactive tool for annotating indoor rgbd images. *Computer Graphics Forum*, 34(2):447–457.

[126] Yamada, T., Hosoe, M., Kato, K., and Yamamoto, K. (2017). The character generation in handwriting feature extraction using variational autoencoder. In *ICDAR 2017*, pages 1019–1024.

[127] Zeiler, M. D. and Fergus, R. (2014). Visualizing and understanding convolutional networks. In *Computer Vision - ECCV 2014 - 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I*, pages 818–833.

[128] Zhang, C., Li, H., Wang, X., and Yang, X. (2015). Cross-scene crowd counting via deep convolutional neural networks. In *IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, pages 833–841.

[129] Ziran, Z. and Marinai, S. (2018). Object detection in floor plan images. In *ANNPR, 2018*, LNCS, pages 383–394.