

Image Processing and Forward Propagation using Binary Representations, and
Robust Audio Analysis Using Deep Learning

by

Fabrizio Pedersoli

B.Sc., University of Brescia, 2009

M.Sc., University of Brescia, 2012

A Dissertation Submitted in Partial Fulfillment of the
Requirements for the Degree of

DOCTOR OF PHILOSOPHY

in the Department of Computer Science

© Fabrizio Pedersoli, 2019
University of Victoria

All rights reserved. This dissertation may not be reproduced in whole or in part, by
photocopying or other means, without the permission of the author.

Image Processing and Forward Propagation using Binary Representations, and
Robust Audio Analysis Using Deep Learning

by

Fabrizio Pedersoli

B.Sc., University of Brescia, 2009

M.Sc., University of Brescia, 2012

Supervisory Committee

Dr. George Tzanetakis, Supervisor
(Department of Computer Science)

Dr. Kwang Moo Yi, Departmental Member
(Department of Compute Science)

Dr. Alexandra Branzan Albu, Outside Member
(Department of Electrical and Computer Engineering)

Supervisory Committee

Dr. George Tzanetakis, Supervisor
(Department of Computer Science)

Dr. Kwang Moo Yi, Departmental Member
(Department of Computer Science)

Dr. Alexandra Branzan Albu, Outside Member
(Department of Electrical and Computer Engineering)

ABSTRACT

The work presented in this thesis consists of three main topics: document segmentation and classification into text and score, efficient computation with binary representations, and deep learning architectures for polyphonic music transcription and classification.

Optical Character Recognition (OCR) and Optical Music Recognition (OMR) can be used to extract information from large collections of scanned documents. In the case of musical documents, an important problem is separating text from musical score by detecting the corresponding boundary boxes so that each process (OCR or OMR) can be applied to the correct type of data. Therefore, a new algorithm is proposed for pixel-wise classification of digital documents in musical score and text. It is based on a bag-of-visual-words approach and random forest classification. A robust technique for identifying bounding boxes of text and music score from the pixel-wise classification is also proposed.

For efficient processing of learned models, we turn our attention to binary representations. When dealing with binary data, the use of bit-packing and bit-wise computation can reduce computational time and memory requirements considerably. Efficiency is a key factor when processing large scale datasets and in industrial applications. For example OMR and OCR can benefit from efficient processing of bi-

nary images. SPmat is an optimized framework for binary image processing. We propose a bit-packed representation for binary images that encodes both pixels and square neighborhoods, and design SPmat, an optimized framework for binary image processing, around it. Using the SPmat representation, we define and evaluate optimized implementations of a variety of binary image processing algorithms such as: erosion/dilation, run-length extraction, contour extraction, and thinning.

Bit-packing and bit-wise computation can also be used for efficient forward propagation in deep neural networks. Quantified deep neural networks have recently been proposed with the goal of improving computational time performance and memory requirements while maintaining as much as possible classification performance. In such networks, the weights and activations are quantized to lower precision and integer arithmetic is used to speed-up computations. A particular type of quantized neural networks are binary neural networks in which the weights and activations are constrained to -1 and $+1$. In this thesis, we describe and evaluate Espresso, a novel optimized framework for fast inference of binary neural networks that takes advantage of bit-packing and bit-wise computations. Espresso is self contained, written in C/CUDA and provides optimized implementations of all the building blocks needed to perform forward propagation. In the context of Espresso, we also describe how binary techniques can be used for efficient forward propagation of convolutional neural networks, a case not covered by existing literature on binary neural networks.

Following the recent success, we further investigate Deep neural networks. They have achieved state-of-the-art results and outperformed traditional machine learning methods in many applications such as: computer vision, speech recognition, and machine translation. However, in the case of music information retrieval (MIR) and audio analysis, shallow neural networks are commonly used. The effectiveness of deep and very deep architectures for MIR and audio tasks has not been explored in detail. It is also not clear what is the best input representation for a particular task. We therefore investigate deep neural networks for the following audio analysis tasks: polyphonic music transcription, musical genre classification, and urban sound classification. We analyze the performance of common classification network architectures using different input representations, paying specific attention to residual networks. We also evaluate the robustness of these models in case of degraded audio using different combinations of training/testing data. Through experimental evaluation we show that residual networks provide consistent performance improvements when analyzing degraded audio across different representations and tasks. Finally,

we present a convolutional architecture based on U-Net that can improve polyphonic music transcription performance of different baseline transcription networks.

Contents

Supervisory Committee	ii
Abstract	iii
Table of Contents	vi
List of Tables	ix
List of Figures	xi
1 Introduction	1
1.1 Overview of the thesis material	3
1.1.1 Document segmentation and classification	3
1.1.2 Efficient computation with binary representations	4
1.1.3 Neural networks for music transcription and classification . . .	4
1.2 Contributions	6
1.2.1 Document segmentation and classification	6
1.2.2 Efficient computations with binary representations	6
1.2.3 Neural networks for music transcription and classification . . .	7
1.2.4 Reproducibility	8
1.3 History	8
1.4 Conclusion and structure of the thesis	9
2 Related work	10
2.1 Document segmentation and classification	10
2.1.1 Staff lines detection and removal	11
2.1.2 Musical feature detection	12
2.1.3 Document segmentation	13
2.2 Efficient computation with binary representations	14

2.2.1	Binary image processing	15
2.2.2	Binary neural networks	16
2.3	Neural networks for music transcription and classification	19
2.3.1	Traditional methods	21
2.3.2	Deep learning methods	23
3	Document segmentation and classification	26
3.1	Algorithm description	27
3.1.1	Random Block Voting (RBV)	28
3.1.2	Coarse segmentation	31
3.1.3	Final segmentation	32
3.2	Datasets	33
3.2.1	Artificial Dataset	34
3.2.2	Real Dataset	37
3.3	Experimental results	38
3.3.1	Baseline approach	39
3.3.2	Artificial Dataset Experiments	42
3.3.3	Real dataset experiments	46
4	Efficient computation with binary representations	51
4.1	SPmat	51
4.1.1	Proposed framework	52
4.1.2	Algorithms and optimized implementation	54
4.1.3	Experimental results	59
4.2	Espresso	64
4.2.1	The Espresso Framework	66
4.2.2	Binary Deep Neural Networks (BDNNs)	67
4.2.3	Espresso architecture	70
4.3	Conclusions	76
5	Neural network for music transcription and classification	77
5.1	Deep neural networks on degraded audio	78
5.1.1	Methodology	79
5.1.2	Neural Networks	80
5.1.3	Datasets	83
5.1.4	Experimental Results	85

5.1.5	Additional experiments	93
5.1.6	Experiments with binary neural networks	97
5.2	Improving music transcription with skip connections	99
5.2.1	U-Net architecture	100
5.2.2	Proposed solution	102
5.2.3	Methodology	104
5.2.4	Results	106
5.2.5	Additional results	108
5.3	Conclusion	109
6	Conclusion and future work	110
6.1	Document segmentation and classification	110
6.2	Efficient computation with binary representations	111
6.3	Neural networks for music transcription and classification	112
6.4	Future work	113
A	Publicly available software	115
A.1	Document segmentation and classification	115
A.2	SPmat	115
A.3	Espresso	115
A.4	Neural network for music transcription and classification	115
B	Publications	116
B.1	Publications not related to the thesis	116
C	Source code examples	118
C.1	SPmat	118
	Bibliography	123

List of Tables

Table 3.1	K grid search with 3 fold cross-validation.	36
Table 3.2	Performance of the baseline approach on the artificial dataset. .	42
Table 3.3	Performance results on the artificial dataset.	46
Table 3.4	Performance results on the real datasets.	48
Table 4.1	Comparison vs baseline implementation [speed-up / μ s]	61
Table 4.2	Comparison with state-of-the-art libraries (CPU) [speed-up / μ s]	63
Table 4.3	Averaged time of binary optimized matrix multiplication.	74
Table 4.4	Average prediction time of the BMLP.	74
Table 4.5	Average prediction time of the BCNN.	75
Table 5.1	Transcription results on “clean/clean” configuration.	86
Table 5.2	Genre results on “clean/clean” configuration.	87
Table 5.3	Transcription results on “phone” degradation.	88
Table 5.4	Transcription results on “hall” degradation	89
Table 5.5	Genre results on “phone” degradation.	89
Table 5.6	Genre results on “hall” degradation.	90
Table 5.7	Transcription results using “all” degradations.	92
Table 5.8	Genre results using “all” degradations.	92
Table 5.9	Sound results on “clean” configuration.	94
Table 5.10	Sound results on “phone” degradation.	95
Table 5.11	Sound results on “hall” degradation.	96
Table 5.12	Sound results using “all” degradations.	96
Table 5.13	Music transcription results with binary neural networks.	98
Table 5.14	Genre classification results with binary neural networks.	99
Table 5.15	“Instrument-agnostic” transcription results.	107
Table 5.16	“Piano”/“non-piano” transcription results.	107
Table 5.17	All instrument transcription results.	107
Table 5.18	Transcription results when skip-connections are removed.	108

Table 5.19 Transcription results when the front-end is an auto-encoder and the model is trained with double loss.	108
--	-----

List of Figures

Figure 2.1 Musical score and text segmentation example.	11
Figure 2.2 Neural network neuron.	17
Figure 2.3 Simple example of two layers feed forward neural network. . . .	17
Figure 2.4 Example of spectrogram and piano-roll notation.	20
Figure 3.1 Segmentation algorithm.	28
Figure 3.2 Segmentation steps of a test image.	28
(a) test image.	28
(b) ground truth.	28
(c) coarse segmentation.	28
(d) final segmentation.	28
Figure 3.3 Bag of visual word testing/training scheme. Where the testing flow is indicated in black and the training one in gray.	31
Figure 3.4 Final segmentation flowchart.	32
Figure 3.5 Examples of images.	35
Figure 3.6 Test set examples.	36
Figure 3.7 Examples of scanned image.	38
Figure 3.8 Examples of OCRopus page segmentation output.	40
Figure 3.9 Baseline approach evaluation for each image.	43
Figure 3.10 Detection performance of the proposed system.	44
Figure 3.11 Detection performance of the coarse segmentation.	45
Figure 3.12 Examples of document segmentation.	47
(a) good	47
(b) good	47
(c) good	47
(d) bad	47
Figure 3.13 Examples of modern book segmentation.	49
Figure 3.14 Examples of vintage book segmentation.	49

Figure 4.1 Example of the split and merge procedure	56
Figure 4.2 Example of the optimized “find the next contour pixel” procedure	58
Figure 4.3 <i>unrolling</i> and <i>lifting</i> operations for CNN layers	71
Figure 5.1 Residual block.	82
Figure 5.2 Results overview for music transcription task.	88
Figure 5.3 Results overview for genre classification task.	90
Figure 5.4 Transcription results using “all” degradations.	91
Figure 5.5 Genre results using “all” degradations.	93
Figure 5.6 Results overview for the sound classification task.	95
Figure 5.7 Sound results using “all” degradations.	97
Figure 5.8 The U-Net architecture (red arrows represent skip-connections).	101
Figure 5.9 Proposed transcription architecture.	102
Figure 5.10 Instrument-wise transcription architecture.	104

Chapter 1

Introduction

Deep Learning (DL) requires significant computational resources and therefore *efficient computation* with deep networks is important in many application contexts. DL models have achieved state-of-the-art performance in many challenging applications and achieved considerable improvements in accuracy compared to more “traditional” machine learning models [50, 90]. These advances caused by DL models increased the research focus on Artificial Intelligence (AI) in both academia and industry. The availability of a variety of DL frameworks and libraries [1, 83, 26] and of public large data-sets [34] have made experimentation with DL techniques much easier than it used to be. In fact, industry in many application domains has quickly adopted deep learning and have nearly abandoned traditional machine learning solutions. Impressive breakthroughs have been accomplished, up to a point where we are experiencing and interacting with AI on a daily basis, for example through self-driving cars and virtual assistants. Moreover for the first time, AI has been able to beat humans in extremely challenging games such as the game of Go [98].

However, DL models are notoriously heavy in terms of computational demands especially for the training stage but also at the inference stage. Indeed, powerful GPUs are frequently needed to effectively use these technologies. Therefore, in order to use these powerful models in real world applications, a lot of research [54, 44, 68, 48, 28] has been conducted for speeding up DL computations. The main strategy for making DL models more efficient consists of: reducing parameters with pruning or sharing techniques, and quantizing parameters such that more efficient machine instructions for computation can be used. In addition, custom hardware has been recently designed and manufactured for accelerating DL models, such as Google Tensor Process-

ing Unit ¹.

DL models are based on Deep Neural networks (DNNs), and Convolutional Neural Networks (CNNs) in particular. These biological inspired models are not new, they had already been studied and were quite popular, during the 70-80's. The main concepts have not dramatically changed since then. In fact, CNNs architectures are still very similar to the one proposed by LeCun for handwritten digit classification [64]. Moreover, the concept of using back-propagation algorithm for training, is still as it was initially proposed, although more powerful gradient descent optimizers [59] have been proposed.

Back then neural networks were shallow because of limited computational resources, lack of regularization schemes to avoid overfitting, and did not perform as well as we are experiencing today. Therefore during the 90's, neural networks lost interest in favour of more mathematically rigorous models such as Support Vector Machines (SVMs) [15]. Indeed, SVMs solve a convex optimization problem which is guaranteed to converge to a global minima.

In 2012, the seminal work of Krizhevsky et al. [62], vigorously brought the attention back to neural networks. The proposed deep architecture trained on the large scale Imagenet dataset [34] won the ILSVCR ² image classification competition with an impressive margin with respect to the previous state-of-the-art.

DNNs, large scale datasets, and powerful GPUs, were instrumental for this success. In fact, the recent technology of graphic accelerators allowed to train deeper and deeper models, unveiling the power of these models when trained on massive datasets. As opposite to the standard machine learning algorithms, DNNs are able to scale well as the data samples increase, without suffering an "accuracy plateau" as the other models do. DNNs became immediately popular, and were applied to other research problems achieving state-of-the-art results in different fields, such as speech recognition and machine translation [42, 4].

DL models have also started to be used in Music Information Retrieval (MIR). However, currently the effectiveness of DNNs is not as high compared to other techniques as in other fields. In fact, neural networks for music tend to be shallow, and mainly used as replacement for hand-crafted feature extraction and traditional classifiers. Very deep architectures, such as the ones proposed for image classification have not yet been explored for several MIR tasks.

¹<https://cloud.google.com/tpu/>

²<http://www.image-net.org/challenges/LSVRC/>

This thesis touches both the aspects of efficient computation, and DNNs for music. Efficient computation focuses on binary data and it is not solely related to DL, but also extended to image processing. In addition to that, a section of the thesis also focuses on a more traditional machine learning approach for document segmentation and classification. Regarding the music domain, we experiment with popular DNN architecture for image classification, with special emphasis on Residual Neural Networks, and focus on the tasks of: polyphonic music transcription and genre classification. For the same tasks, we also investigate the robustness of DNNs to degraded audio.

A more detailed overview of the thesis material is given in the following Section 1.1.

1.1 Overview of the thesis material

This section gives an introduction to the main topics of the thesis work: document segmentation and classification, Section 1.1.1; efficient computation with binary representations, Section 1.1.2; and finally neural networks architectures for music transcription and classification, Section 1.1.3.

1.1.1 Document segmentation and classification

A new algorithm for segmenting documents into regions containing musical scores and text is proposed. Such segmentation is required as a step prior to applying Optical Character Recognition (OCR) and Optical Music Recognition (OMR) on scanned pages that contain both music notation and text. Our segmentation technique is based on the Bag of Visual Words (BoVW) representation followed by Random Block Voting (RBV) in order to detect the bounding boxes containing the musical score and text within a document image. The RBV procedure consists of extracting a fixed number of blocks whose position and size are sampled from a discrete uniform distribution that “over”-covers the input image. Each block is automatically classified as either coming from musical-score or text and votes with a posterior probability of classification proportional to its spatial extent. An initial coarse segmentation is obtained by summarizing all the votes in a single image. Subsequently, the final segmentation is obtained by subdividing the image in microblocks and classifying them using a N -nearest neighbor classifier which is trained using the coarse segmentation. We demonstrate the potential of the proposed method by experiments on two different

datasets. The first dataset is a challenging collection of images artificially combined and manipulated for this project. The other one is a music dataset obtained by the scanning of two music books. The results are reported using precision/recall metrics of the overlapping area with respect to the ground truth. The proposed system achieves an overall averaged F-measure of 85%.

1.1.2 Efficient computation with binary representations

We propose an optimized framework for binary image processing, characterized by a highly bit-packed representation of pixels and their square neighbourhood. The Super-Packed (*SPmat*) representation for binary images enables the easy use of bit-wise computations for developing fast processing algorithms, such as: morphology, contours, run-length, and thinning, in a unified framework. With several experiments, we show that the aforementioned algorithms can be consistently sped-up, and outperform by a large margin available software implementations.

In addition to that, there are many applications scenarios for which the computational performance and memory footprint of the prediction phase of Deep Neural Networks (DNNs) need to be optimized. Binary Deep Neural Networks (BDNNs) have been shown to be an effective way of achieving this objective.

Espresso is a compact, yet powerful library written in C/CUDA that features all the functionalities required for the forward propagation of CNNs, in a binary file less than 400 kB, without any external dependencies. Although it is mainly designed to take advantage of massive GPU parallelism, *Espresso* also provides an equivalent CPU implementation for CNNs. *Espresso* provides special convolutional and dense layers for BCNNs, leveraging bit-packing and bit-wise computations for efficient execution. These techniques provide a speed-up of matrix-multiplication routines, and at the same time, reduce memory usage when storing parameters and activations. We experimentally show that *Espresso* is significantly faster than existing implementations of optimized binary neural networks (≈ 2 orders of magnitude).

1.1.3 Neural networks for music transcription and classification

Deep residual networks were originally proposed and shown to be successful in the context of image classification. A variety of deep learning systems have been used for

audio analysis tasks such as music transcription and classification as well as urban sound classification. When the analyzed audio signal is degraded (for example when played in a hall with reverberation or captured by a smart phone) the performance of these systems can be negatively affected. We present a detailed experimental evaluation of several Convolutional Neural Network (CNN) models for audio analysis tasks, and their performance in the presence of audio degradations. More specifically we focus on two fundamental tasks in Music Information Retrieval (MIR): polyphonic music transcription and musical genre classification. In addition, we also consider urban sound classification as a non-music audio analysis task. Different scenarios of training and testing with two types of audio degradations are investigated. We experiment with popular CNN architectures of different depth, ranging from: a shallow network with “long” mono-dimensional kernels, up to a very deep network with 3×3 kernels and residual connections using as input typical time-frequency representations based on spectrograms. Interestingly, we show that while different architectures provide the best performance on clean data, with degraded audio Residual Networks always provide the best results, and have on-par performance with respect to the best performing architecture. This suggests that residual connections provide robustness to audio degradations.

We further propose the use of U-Net as a way of improving polyphonic music transcription performance of various baseline CNNs. We propose a convolutional architecture composed by a *transformation* network (U-Net) which is put in front of a *transcription* network. Notably, we do not introduce any additional loss specific to this network, and instead the model is trained with the original loss functions that were designed for the back-end transcription network. We argue that this U-Net preprocess the input signal into a representation that is more effective for transcription, and thus enabling the enhancement. Indeed, we empirically confirm with exhaustive experiments on the MusicNet dataset that the proposed configuration unleashes the full potential of the transcription network. Moreover, we show that we can go beyond simple transcription and perform instrument-wise music transcription, easily with the proposed architecture. We show that by doing so, we can even increase the original holistic transcription performance.

1.2 Contributions

In this section, the main contributions of the thesis work in the respective areas are highlighted.

1.2.1 Document segmentation and classification

Traditional document segmentation techniques are specific for certain types of documents, and often make prior assumptions about the layout and content of the document. Being able to perform a segmentation without prior knowledge, that can scale up to a wide variety of contents and classes is a challenging problem. We propose a machine learning approach to document segmentation based on classification, which is scalable and does not require prior knowledge.

The main contributions of the document segmentation work are:

- Pixel-wise segmentation technique based on Random Block Voting (RBV).
- Robust procedure for extracting bounding-boxes from the pixel-wise classification.
- Content independent segmentation algorithm (extensible to other segmentation classes).

1.2.2 Efficient computations with binary representations

Providing optimized implementation is a challenging task because the developer needs to be aware, and understand low level hardware details. Moreover, optimized implementations require a significant amount of work and profiling before being deployed. In fact, optimized implementations are created from scratch, since algorithms are based on hand-crafted data structures.

The main contributions of SPmat for binary image processing are:

- Bit-packed image representation of pixels and square neighbours.
- Implementation of optimized binary image processing algorithms with bit-wise computations and look-up tables.

Computational efficiency of deep learning models is a hot research topic nowadays. The main challenges in this case is to deploy deep neural networks on low power

embedded devices. Optimized implementations are fundamental in order to achieve this goal. In fact, if we consider binary networks, the entire computational back-end, tensors and layers have to be redesigned for obtaining the best possible performance.

The main contributions of the Espresso framework for efficient binary neural networks forward propagation are:

- Optimized self-contained framework for binary data.
- Binary optimized neural network routines.
- Convolutional layer.
- State-of-the-art performance in terms of computational time.

1.2.3 Neural networks for music transcription and classification

Deep learning, and deep neural networks are the clear answer to the most of the computer vision and speech recognition problems. These models are fundamental for obtaining state-of-the-art performance. In music information retrieval, deep models have not been thoroughly evaluated yet. The main challenge in this case is the setup of a well engineered evaluation framework that can handle large scale datasets, logs and displays all the information for diagnosing the model training. In addition, the experimental framework must be able to run in a super-computer environment where many nodes and GPU can be used at the same time.

The main contributions of the neural networks for music transcription and classification work are summarized as follows:

- Evaluation and comparison of deep neural networks for some popular Music Information Retrieval (MIR) tasks.
- Evaluation and comparison of spectrogram input representations.
- Performance analysis on degraded audio.
- Effectiveness of ResNet architectures for music analysis.
- Front-end/back-end convolutional architecture for improving polyphonic music transcription.

1.2.4 Reproducibility

All the developed software during thesis work has been made available under open-source license on github. We strongly encourage researchers to share their implementations such that the results can be easily reproduced, further investigated and improved by future work of other researchers [84]. More details about the software repositories are provided in the Appendix A.

1.3 History

The thesis work initially started on the document segmentation and classification topic. The intended application scenario was large-scale document segmentation of historical digital music documents. For this reason, although not directly related to the initial segmentation work, performance issues and high processing throughput were in the back of my mind at the time. In the field of document processing, several processing techniques, such as OCR or OMR, process binary data. Available libraries for image processing do not differentiate between binary and non-binary data. However, huge advantages in terms of computational speed and memory can be achieved by using binary computation. With this goal in my mind, I proposed and developed SPmat for fast binary image processing.

The document segmentation work was based on traditional machine learning techniques. However, deep learning was already increasing in importance at that time, and I started being interested in this topic. The superiority of Deep Neural architectures was already clearly established back then. Therefore, I got interested on a more recent research area of deep learning that deals with computational efficiency. An emerging branch of the deep learning that focused on computational efficiency was *quantized* networks. A particular case of quantized networks is *binary* networks. Binary networks, while in many cases achieving classification performance comparable to their floating point counterpart, potentially allow for optimized deep learning implementations. However, almost nobody at the time really investigated optimized implementations for binary neural networks, mainly because this task requires the development of the entire computational framework from scratch. Thus, Espresso was developed for unleashing the full computational speed potential of binary neural networks.

Binary networks for image classification performed quite well, despite the loss

of information in the weights and activations due to binarization. One interesting research question is to apply such models to other domains, and assess their performance. The application of choice was MIR and audio analysis. Before investigating the effectiveness of binary neural networks for MIR tasks, some more fundamental questions needed to be investigated. Specifically, the effectiveness of deep architectures in MIR and audio analysis has not been clearly established. Currently, most of the neural networks that are being used are relatively shallow and there is no general agreement about the network specifications, and the input representation to use. In the neural networks for music transcription and classification theme of this thesis we try to answer these questions. We also further study the robustness of different deep neural architectures to audio degradation. In addition, we propose a front-end/back-end convolutional architecture for improving polyphonic music transcription. After clarifying these aspects of deep learning for MIR, we finally investigate the effectiveness of binary networks for music transcription and classification.

1.4 Conclusion and structure of the thesis

In this chapter the research described in this thesis was described and motivated. It is subdivided in three main areas, and we highlighted the main contributions made to each area. The rest of the thesis is organized as follows. Chapter 2 describes related work that provides context and informs this work. Chapter 3 describes the proposed algorithm for document segmentation and classification. Chapter 4 describes the proposed optimized frameworks for binary image processing and forward propagation for binary neural networks. Chapter 5 outlines the work on neural networks for music transcription and classification. Finally, Chapter 6 draws the thesis conclusions.

Chapter 2

Related work

This chapter discusses previous work related to the main of topics of this thesis. Section 2.1 describes related work on document segmentation. Section 2.2 focuses on related work to efficient computations with binary representation in the context of binary image processing and binary neural networks. Finally, Section 2.3 reports related work on: traditional and deep learning approaches for music transcription and classification, mainly focused on genre classification.

2.1 Document segmentation and classification

The document segmentation tasks consists of: given a digital image representing a document page, finding the bounding-boxes that are representative of particular classes. In our application we are interested in identifying regions of a document (image) related to musical score and text. In Figure 2.1 we show examples of document segmentations in musical score and text. More details will be presented in Chapter 3.

To the best of our knowledge, there is no previous work that deals with segmenting mixed documents containing both musical score and text regions . Related published work can be grouped into two categories: 1) papers dealing solely with musical score images and extracting features and information from them [39, 88], 2) papers dealing with more general segmentation of documents in text and non-text (graphics) regions, or document structure analysis [73, 95, 3]. The following subsections describe related work, which is focused on: the detection of Staff lines, (subsection 2.1.1), the detection of musical features, (subsection 2.1.2), and text/graphics document segmentation,

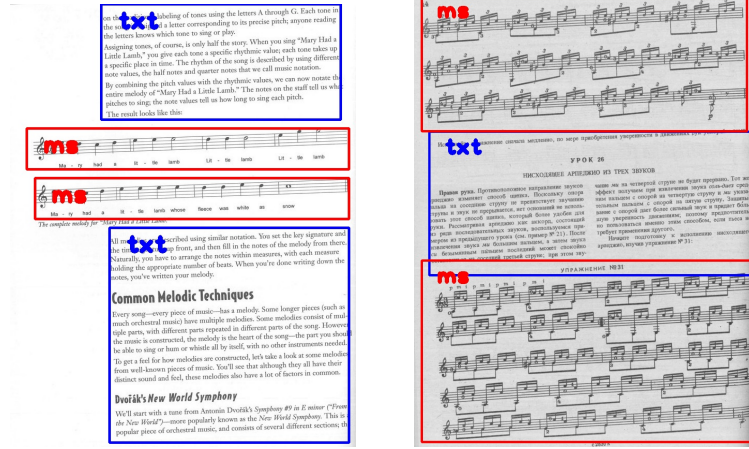


Figure 2.1: Musical score and text segmentation example.

(subsection 2.1.3).

2.1.1 Staff lines detection and removal

The generic scheme for finding staff lines is a generalization of the method described by Miyao et al. [76]. This technique operates on a set of staff segments. Methods for horizontally/vertically linking two staff segments, and for merging segments that are overlapping are proposed. Once all the links are added, the resulting graph is partitioned in subgraphs. Each subgraph that is wide and high enough corresponds to a staff line.

Based on this general method Dalitz et al. [31] have proposed a staff removal algorithm that uses skeleton information. The skeleton is split at branching points and corner points. Around each splitting point, a subset of pixels, taken from the distance transform at the splitting point, is removed. Staff line segments are selected as skeleton segments that satisfy predefined geometric criteria. Then, the generic method described above is applied to detect staff lines. Authors also propose several criteria, based on overlap and branch points similarity, to remove false positive staff lines. Finally, in order to remove staff lines, all vertical “black” runs around the detected staff skeletons are removed.

In addition, regarding the staff lines detection task, two methods based on connected paths have been proposed [22, 37]. In these two publications, the image is represented as a graph in which the pixels are the nodes and the edges connect neighboring pixels. The weight of each arc is a function of pixels values and relative

positions. Given this formulation, a staff line can be considered as a connected path of black pixels that minimizes a predefined distance from the left side of the image to the right. In Cardoso et al. [22] the path cost is given by the sum of all the edge weights belonging to the path. Therefore, a staff line is an 8-connected shortest path in the graph from the left to the right which contains one and only one pixel for each column of the image. Finally, the optimal staff line that minimizes the cost is found using dynamic programming. An improved version of this method has been developed by introducing the notion of a *stable* path [37]. A path $P_{s,t}$ from point s to point t and two regions Ω_1 and Ω_2 is defined as stable if it is the shortest path between $s \in \Omega_1$ and the whole region Ω_2 , as well as the shortest path between $t \in \Omega_2$ and the whole region Ω_1 . This simple assumption allows to define a more reliable method for detecting staff lines, which the authors claim to be robust to skewed images and discontinued/curved staff lines.

A different method for staff line detection has been presented by Su et al. [104]. This technique is based on the global information of the musical document which is used to model the staff line shape. The input to the system is a binary image where musical symbols are white and the rest is black. The first operation is the statistical analysis of black and white column run-length of pixels that permits to obtain the staff height and spacing. This information is then used to remove all the musical symbols that are not considered staff lines. Then, the staff line shape is modeled by analyzing the angular orientation of its pixels along the column of the image. Finally the estimated model is used to detect all staff locations within the document.

2.1.2 Musical feature detection

To detect musical features, such as Stems, Slurs, Staves, Beams, Noteheads, a technique based on Kalman filter has been proposed [32]. In this work, every music feature is considered as a segment with different geometric characteristics. A segment is modeled as a succession of connected run-lengths with the same color and thickness that have a certain direction of evolution in space. This information is used by a Kalman filter for tracking its evolution over space. The Kalman filter is capable of detecting both vertical and horizontal segments which can then be finally classified utilizing simple and intuitive geometric criteria. Another method that uses geometric criteria in order to detect musical features has been proposed by Sicard et al. [96].

Machine learning approaches for the detection of musical features have also been

proposed [52]. This work describes a system that employs a structured decision based neural network (DBNN) to recognize music notes. The first step consists in compensating for eventual rotation angles with a Principal Component Analysis (PCA), afterwards horizontal and vertical lines such as staves and stems are extracted using edge detection and profile projection. The DBNN is trained utilizing the regional density, subdivided in eight neighboring regions, of pixels around the stems, so that it is able to classify the note type.

2.1.3 Document segmentation

Regarding automatic document segmentation, Zirari et al. [122] have proposed a technique to identify text and non-text regions, which is based on graph modeling and structural analysis of connected components. The image is represented as a graph. Initially each pixel is a vertex and the weights between pairs of neighbors are given by the corresponding intensity difference. Connected components are identified by using a measure of homogeneity of intensity, namely the internal difference of a component. Subsequently, the histogram of components' sizes is computed and used to classify them. Text components correspond to the most significant peaks of the histogram. Possible noise and graphical components are then filtered out using the notion of vertical alignment of the components. Another technique that classifies text and non-text connected components has been presented by Bukhari et al. [20]. This technique utilizes a multi-layer perceptron network that employs the shape and context information of a component as features. The shape feature vector consists of a 40×40 rescaled version of the image region, plus the normalized length, height, number of foreground pixels and the aspect ratio. In a similar way, the context feature vector is obtained by rescaling components and its surrounding area (determined as a function of the component size) to a window of 40×40 . Finally an autoMLP, which is a self tuning classifier that automatically adjusts the learning parameters, is trained using these features vectors in order to classify text and non-text components.

Other methods for document segmentation, which are based on fuzzy classification and multi-resolution features, have been proposed [21, 72]. Caponetti et al. [21] have proposed a system for document segmentation that is able to differentiate between text, graphics and background. Initially pixels are classified into coherent regions, using a neuro-fuzzy classifier with multi-resolution features, which are based on the intensity and edge strength. These regions are then refined by shape analy-

sis. Similarly, Maji et al. [72] have presented a segmentation method that combines multi-resolution image analysis and rough fuzzy computing to detect both text and graphics regions of a document. The multiresolution analysis is done using an M-band wavelet that extracts scale-space features. The feature vector is further reduced in dimensionality through unsupervised feature selection to select the most relevant ones. Finally, the rough-fuzzy-probabilistic c-means algorithm is used to obtain the final segmentation.

A more advanced method for a document layout extraction based on Hierarchical Conditional Random Fields (HCRFs) has been proposed [23]. The first stage of this algorithm is the pixel-wise classification, in text, background and image, by using globally matched wavelets as features for a Fisher Linear Discriminant Analysis (LDA) classifier. In the next stages, HCRFs are employed at various levels enabling the learning of: local features (for text, background and images), contextual features (for classifying region blocks such as: title, author, heading, paragraph, etc), and document layout model (for encoding the relations of the previously described block regions). Finally, Cote et al. [27] have presented an algorithm for classification of pixels of business documents. Business documents contain multilayered mixture of textual, graphical and pictorial elements. In order to be able to classify them, an SVM trained on a low dimensional feature vector based on sparseness is used. The sparseness is computed by applying the Hoyer’s measure to the output of the Leung-Malik filter bank for texture analysis. The feature vector is composed by 10 elements; the first 5 measure the sparseness of a pixel at various resolution, and the last five are the mean sparseness in the pixel neighborhood also at different resolutions.

The related work references discussed above are not an exhaustive publication list about document segmentation research. In fact, only classification based techniques to document segmentation have been considered because these solutions are the most related to our algorithm.

2.2 Efficient computation with binary representations

Section 2.2.1 describes related work to optimized binary image processing techniques. Section 2.2.2 focuses on deep neural networks architectures for efficient computation.

2.2.1 Binary image processing

Binary image processing is a branch of image processing where the data to be processed is quantized to two logical values: “0” and “1”. Binary images are often used to represent simple concepts (bitmaps) that stand out from the background. An example of that is text documents. As happens in traditional image processing the goal is to extract some useful information from the input data. The main processing techniques for binary images include: binary morphology, contour extraction, run-length extraction, and thinning. When dealing with binary data, a considerable improvement in terms of computational speed and memory usage can be achieved by using “bit-packing” and “bitwise” computations. Related work on optimized implementations of binary image processing routines mainly apply these two concepts. More details will be presented in Chapter 4.

Considering binary morphology, Bloomberg [14] proposes optimized implementations using image rasterops and word accumulation methods for computing erosion and dilation. In his work, the image is packed into 32 pixels. Erosion and dilation are computed by translating the input image in all directions relative to the structuring element, and calculating respectively the bit-wise ‘and’ and ‘or’. The author shows that repeatedly applying the structuring element to small parts of the image is faster by a factor between 2 and 4 than successive full image rasterops.

Lien [69] developed an implementation for processing binary images by exhaustive table look-ups using the packed 3×3 neighborhood as index. The author shows that use of look-up tables reduces the number of computations needed for each set pixel, and can improve time performance of the Zhang-Suen thinning algorithm [118] by up to 2 times.

In the work of Van Den Boomgard and Van Balen [112], a binary image is represented as a bitmap that stores 32 consecutive horizontal pixels in a single word. Besides the immediate advantages of reducing memory usage, the authors present efficient algorithms for elementary morphological operations, that operate on 32 pixels in parallel. Moreover, they propose the logarithmic decomposition of structuring elements to further improve the computational time when large convex structuring elements are used. Focusing the attention onto the pixels neighbors and efficient ways to store them, reducing memory accesses, Kapela and Rybarczyk [58] proposed the neighboring pixel representation for binary images. With this representation, each pixel contains information of its 3×3 neighborhood, which is stored in a single

byte at the pixels address location. This allows for the implementation of efficient binary image algorithms such as morphological operations and contour tracking, by considerably reducing memory accesses.

A similar approach of handling the 3×3 pixel neighborhood was proposed by Sobel [103]. Also in this case the idea is to retrieve all of the 8 neighbors in a single memory access, assembling them in a compact 8-bit code. With this setup, the desired processing is implemented by computing the neighborhood function by a simple table lookup. More specifically, Sobel proposed a fast version of a simple contour follower algorithm that takes advantages of the proposed processing formulation, in order to demonstrate the potential improvements.

With the exception of Bloomberg’s [14] work, none of previous optimization techniques have been extended and incorporated into a binary image processing library. The Leptonica ¹ image library processing library developed by Bloomberg, makes use of pixel-packing in the underlying implementation of binary morphology. However, Leptonica does not provide bit-level optimizations of other binary image processing algorithms, such as: contour extraction, thinning, etc. Moreover, the most widely used libraries for image processing, OpenCV [17], do not provide optimized binary image processing algorithms at all.

In contrast to previous work, with the proposed optimized image representation and associated library we focus on defining a general way to encode binary information that does not depend on the specific algorithm, and therefore can be used for a variety of binary image processing algorithms. The bit-wise optimizations proposed in previous work are instead limited to particular applications, and require the programmer to explicitly implement them for each algorithm. Some of them are related to packing of pixels, others to packing of neighbours. It has not been recognized that these two types of optimization can be unified in a common framework that combines their advantages, as we show in our work. The proposed framework provides the necessary tools for developing optimized binary image processing applications through a clean and simple API.

2.2.2 Binary neural networks

Artificial Neural networks are a biological inspired model used for classification or regression. The fundamental component of a neural network is the neuron, Figure 2.2.

¹<http://tpgit.github.io/UnOfficialLeptDocs/leptonica/>

A neuron computes a linear combination of input values according to some weights, and produces a single output by applying a non-linearity. Neural networks are organized

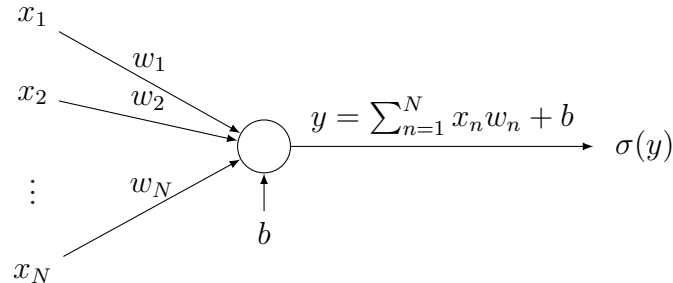


Figure 2.2: Neural network neuron.

in a cascade of layers, where each layer is composed of several neurons. Figure 2.3 shows a simple example of a neural network.

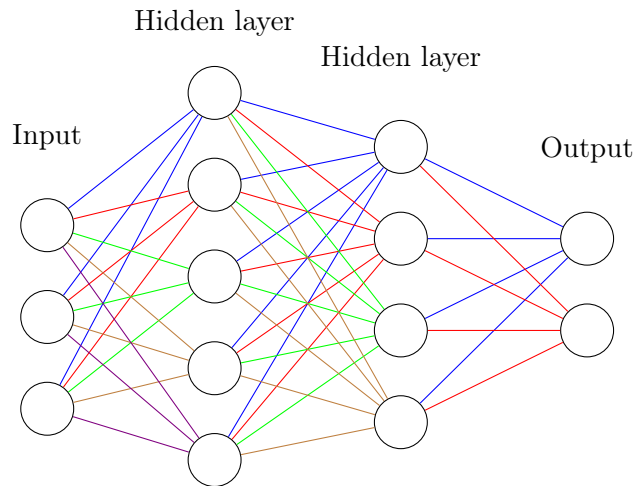


Figure 2.3: Simple example of two layers feed forward neural network.

Neural networks function in two modes: training (back-propagation) and inference (forward propagation). During training, the weights of each layer are optimized by gradient descent according to a loss function. The back-propagation algorithm computes through the chain-rule the derivative of each weight with respect to the loss function. Back-propagation is composed of two parts: forward propagation and backward propagation. Forward propagation computes the network output. During back-propagation the loss function is applied and derivatives are computed in backward order. Once the derivatives are computed, the weights are adjusted by gradient

descent according to some optimization algorithm. Once the networks is trained, the models is used to make predictions using the forward mode.

In the initial era, neural networks were very shallow (no more than a couple of layers) due to the limited computational capabilities available at that time. These shallow models were unable to learn complex concepts, however it was known that deeper model would have solved this limitation. In the Deep Learning era, thanks to powerful computational accelerators such as GPUs, deep neural networks can be used. The effectiveness of these model was immediately evident, and deep neural networks quickly achieved state-of-the-art performance in several application fields [62, 42, 4]. However, this models still require huge computational power, and research has been devoted to make deep neural networks more efficient in terms of computations.

Improving the performance of DNNs can be achieved at either the hardware or software level. At the hardware level, chipsets that are dedicated to DNN execution can outperform general-purpose CPUs/GPUs [57, 47]. At the software level one approach is to design simpler architectures, in terms of overall floating point operations, that can offer the same accuracy as the original model [54]. Another approach is to prune the weights [44], or even entire filters [68], that have low impact on the activations such that a simpler model can be derived. These simplified models can be further compressed by weight sharing [48]. Finally instead of removing connections, another approach is to *quantize* the network weights [28] such that computations can be executed more efficiently.

In quantized networks, the objective is to train DNNs whose (quantized) weights do not significantly impact the network’s classification accuracy. For example, [28] show that 10-bits are enough for Maxout Networks, and how more efficient multiplications can be performed with fixed-point arithmetic. Continuing this trend, more aggressive quantization schemes, up to ternary [121], have also been studied.

Recently, [29] showed that a network with *binary* $\{-1, +1\}$ weights can achieve near state-of-the-art results on several standard datasets. Binary DNNs (BDNNs) were shown to perform effectively on datasets with relatively small images, such as the permutation-invariant MNIST [63], CIFAR-10 [61] and SVHN [80]. Recently, Rastegari et al. [87] show that binarized CNNs can perform well even on massive datasets such as ImageNet [34] using binarized versions of well-known DNN architectures such as AlexNet [62], ResNet-18 [50], and GoogLeNet [107]. Similarly interesting results can be achieved by binarizing both DNN weights and activations as showed by [53]. In their work, the authors introduce *BinaryNet*, a technique to effectively train DNNs

where both weights and activations are constrained to $\{-1, +1\}$. *BinaryNet* achieves nearly state-of-the-art accuracy for MLP training on MNIST and CNN training on CIFAR-10. The authors also propose a binary optimized implementation of matrix multiplication which result in $7\times$ faster performance than the base-line non optimized implementation, and, almost $2\times$ faster than Theano [11]. Their core contributions, namely to replace Floating-point Multiply and Add operations (FMAs) with *XNORs* and *bit-counts*, represent the cornerstone over which we build our research on efficient forward propagation in BNNs.

2.3 Neural networks for music transcription and classification

Music Information Retrieval (MIR) objectives is to automatically analyze audio signal and extract by computer programs some high-level information, such as: Tempo, Beats, genre, fundamental Pitches. In this thesis, we focus on genre classification, and polyphonic music transcription tasks. Genre classification consist in analyzing a clip of audio, and predicting the correspondent genre. On the other hand, polyphonic music transcription consists in producing a piano-roll representation of a given music peace, where actives notes at depicted any given time. Music transcription is a challenging task, specially when the source is polyphonic, i.e. more than one independent melody occurs.

In many MIR applications, the input audio signal is often represented in terms of magnitude spectrograms. Spectrogram are a time-frequency representation of signals, where the DFT of subsequent window of audio is computed through time. More precisely, a spectrogram is defined by three fundamental parameters: window size, hop-length and number of DFT bins (usually matches the window size). The window size is the number of samples of the audio signal used to compute the DFT. The hop-length is number of samples the audio window is shifted through time.

Prior to Deep Learning, MIR tasks relied on hand-crafted features and a variety of traditional machine learning classifiers. Spectral features were mainly extracted from magnitude spectrograms, which statistically model the behaviour of how the energies of different frequencies evolve through time. In addition to such spectral features, “temporal” features were also computed from spectrograms, such as: beat/tempo and, zero crossing rate. Logarithmically spaced time-frequency representations such

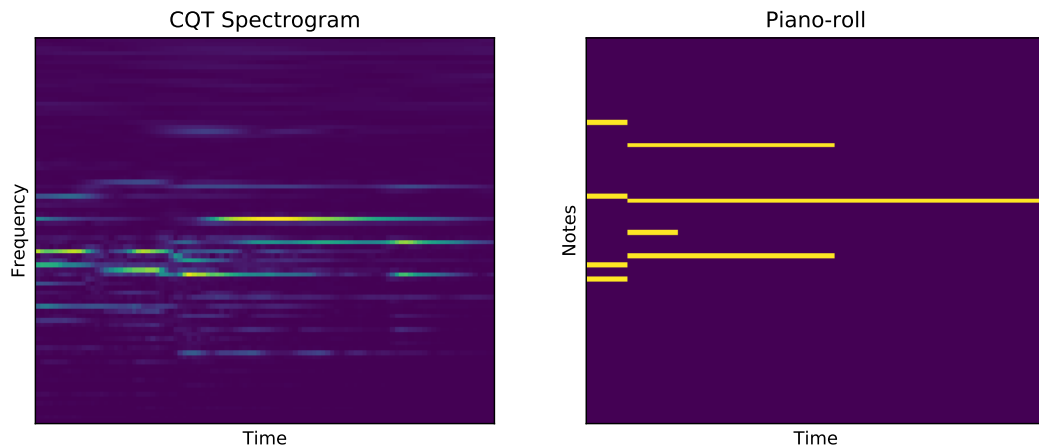


Figure 2.4: Example of spectrogram and piano-roll notation.

as mel scaled spectrograms or the Constant-Q Transform (CQT), have been shown to be effective for representing music. Musical pitch perception and discrete musical pitches in many music cultures are spaced logarithmically and such representations capture musical pitch patterns more accurately. Mel Frequency Cepstral Coefficients (MFCCs), extracted from mel-scaled spectrograms, are also widely used, not only for music classification, but also for speech recognition [70, 114].

Music classification models, and specifically music transcription models, are usually composed of two main parts: an acoustic model, and a musical language model. The acoustic model is trained on spectral features and provides classifications, or probability estimates, based on a single short time frame of the input signal. In other words, the acoustic model does not model temporal relationships of features. However, the music signal is characterized by peculiar temporal patterns according to its nature. Being able to model the temporal relationships of features, would be beneficial in providing a better performing model. For this reason, a music language model is often used and, yields better performance when combined with the acoustic model. The music language model is trained on top of the output of the acoustic model and, it refines the time-wise classifications by analyzing the entire sequence of predictions through time. Hidden Markov Models (HMMs), are widely utilized as music language models. Alternatively to the use of the acoustic/language model, another common setup for music classification is based on feature aggregation. Feature aggregation (or integration) consists of computing a single feature vector for a given analysis window of audio, by summarizing the features of each time frame by doing some statistical analysis over a longer time period.

Deep Learning (DL) models have achieved state-of-the-art results in many research fields, such as: computer vision, speech recognition and machine translation [50, 42, 106]. Motivated by these successes, there has been growing interest in applying these powerful models to music. Therefore in recent work, both the acoustic and language models have been replaced by Deep Neural Networks (DNNs).

Deep learning models, e.g. Convolutional Neural Networks (CNNs) typically do not rely on hand-crafted features and instead automatically perform feature extraction, which is embedded in the model itself and, optimized during training. Thus, these models are typically trained on low-level representations of audio signals, such as spectrograms. The music language model instead, has been replaced by sequence modelling networks such as Recurrent Neural Network (RNN) or derived models such as: Long-Short Term Memory models (LSTM), or Gated Recurrent Units (GRU).

2.3.1 Traditional methods

Regarding music genre classification, Tzanetakis and Cook [111] proposed three sets of audio features representing: timbral texture, rhythmic content, and pitch content for training a Gaussia Mixture Model (GMM) classifier on 10 genre classification task. Similarly in the work of Xu et al. [117], a multi-layer Support Vector Machine (SVM) classifier trained on spectral features, was used for genre classification. Feature extracted from Long-term modulation spectral analysis of spectral features and MFCCs, were used in the work of Lee et al. [65]. The modulation spectra are collected in a modulation spectrogram which is then decomposed in several logarithmically-spaced modulation subbands. From each subband, the modulation spectral contrast (MSC) and modulation spectral valley (MSV) are computed. Feature are extracted from MSC and MSV by means of statistical aggregations. Classification is done by feature fusion, and Linear Discriminant Analysis (LDA). Meng et al. [75] proposed a multivariate autoregressive model for temporal feature integration for music genre classification. This integration model is able to capture temporal dynamics and dependencies of independent features and it was shown to be superior to the more traditional mean, standard deviation feature integration.

Initial approaches to automatic music transcription were mainly unsupervised and based on spectral factorization techniques. In these approaches, the goal is to factorize the magnitude spectra into two components in such a way that one component is related to the frequency profile of each note, and the other one is related to the

activation in time of each note.

For instance, Smaragdis et al. [100] used Non-negative Matrix Factorization (NMF) approach to factorize the magnitude spectrogram. Although their method requires the prior knowledge of the number of note events present in the analyzed audio segment, it showed initial interesting results both in monophonic and polyphonic music. Smaragdis et al. [100] proposed the use of Probabilistic Latent Component Analysis (PLCA) for spectrogram decomposition. This statistical framework models the spectra as a multi-dimensional distribution, which is approximated by a mixture of marginals distribution products. These marginals are estimated using a variant of the Expectation Maximization (EM) algorithm. Smaragdis et al. [101] modified the standard PLCA model in a way that would make it possible to detect multiple local shift invariant patterns. According to this shift invariant model, the marginals distributions are defined in terms of convolutions. Grindlay et al. [43] extended the PLCA model to multiple polyphonic sources. A set of training instruments is used to learn a sparse model space with NMF. This model is then used to learn the distributions of pitches conditioned to the sources. Benetos et al. [8] extended the shift-invariant PLCA to support the use of multiple spectral template per pitch and per instrument. The time varying pitch contribution of each source is also considered by the proposed model extension.

Rather than using spectrogram factorization, Poliner et al. [85] proposed a discriminative model for polyphonic piano transcription. In their work, a Support Vector Machine (SMV) is trained on spectral features, and used to classify frame-level notes instances. In addition, an HMM is used to temporally constrain the SVM outputs.

Instead of relying on hand-crafted features, Nam et al. [77] used Deep Belief Networks (DBM) to learn feature representations of notes and jointly train classifiers for multiple notes. Similarly to Poliner et al. [85], the DBM output is temporally smoothed by an HMM.

In all the above methods a pitched sound source is modelled. However, in some cases the transcription of unpitched instruments, such as drums, is also required. Benetos et al. [9] proposed an extension of the PLCA that jointly transcribes pitched and unpitched sound, such as drum kit components from polyphonic music. In this case the marginal distributions are defined by two components: pitched and unpitched.

Other techniques for automatic music transcription rely on multiple- f_0 estimation, instead of spectrogram factorization. Multiple- f_0 estimation by itself, is not reliable

enough for providing good transcription results. For this reason, this processing stage is often combined with additional processing stages that model other musical aspects of the audio signal. Ryyanen et al. [91] proposed a music transcription system composed of: multiple- f_0 estimation, an acoustic model and, a musicological model. The acoustic model, takes as input three features extracted from the multiple- f_0 and calculates the likelihoods of different notes and performs temporal segmentation of notes. The musicological model instead, estimates the musical key and controls the transition between notes. The final transcription result is obtained by searching for the best paths through the models of the notes. Multiple- f_0 estimation is also used in the work of Benetos et al. [7], which is combined with note onset/offset detection. The input of the transcription system is the resonator time-frequency image [120]. A pitch salience function is extracted by each frame, and onset detection is computed through a spectral flux feature. Finally, a pitch set score function is used for each segment defined by two onsets to estimate the pitch of the current frame.

2.3.2 Deep learning methods

Lee et. al [66] used a shallow CNN to learn features from unlabeled data and evaluate them on different tasks, including music genre and artist classification. For genre classification, Jeong and Lee [56], instead of extracting spectral features, learn temporal features with a DNN by using the cepstral modulation spectrum.

Zhang et al. [119] proposed some architectural tweaks for improving performance of shallow CNN for music genre classification. In particular, authors combined max and average pooling to provide more statistical information, and also suggested the use of a shortcut connection [50].

Choi et al [24], started to experiment with larger datasets and deeper CNNs for the task of automatic music tagging. The authors showed that deeper networks scale better when dealing with larger datasets compared to networks with shallower architectures. In addition, a convolutional and recurrent network were investigated by Choi et al. [25]. The CNN was used for local feature extraction, while the recurrent network was responsible of the temporal summarizing the local features.

A multi-modal approach to music genre classification was proposed by Oramas et al. [81]. In their work, DNNs are used to extract features from audio tracks, text reviews, and cover art images, and they show that feature aggregation from multiple modalities can improve accuracy of classification.

Bittner et al. [12] proposed a fully convolutional neural network for learning the salience and estimation of fundamental frequencies. The network is trained on a large scale, semi-automatically generated f_0 dataset. In order to better capture harmonic relationship, authors used a harmonic constant-Q transform as the input representation. Instead of solely considering a time-frequency representation, Wu et al. [115] use a multi-channel input which includes spectrum, generalized cepstrum, and generalized cepstrum of spectrum, as input to a Convolutional Neural Network (CNN).

Sigtia et al. [97] proposed an architecture that comprises of an acoustic model and, a music model for polyphonic piano music transcription. The acoustic model is a neural network that estimates pitch probabilities for a given audio frame. The musical model is an RNN that models temporal dependencies of pitches. The predictions of the two models are combined using a probabilistic graphical model, and the beam search algorithm is used to perform inference.

The musical language model in all the above works predicts the expected notes at time t given the notes that are active at time $t - 1$. However, it would be preferable to model the conditional distribution of the next time given the previous. RNN and HMM are not able to handle high dimensional distributions. Energy based models, such as Restricted Boltzman Machines (RBMs) can overcome this limitation. Boulanger-Lewandowski et al. [16] proposed the use of Recurrent Temporal RBM (RTRBM), and a generalization of that called RNN RBM, as a musical model for polyphonic music transcription. RTRBM are an extension of RBM to model temporal sequences [105]. Boulanger-Lewandowski et al. [16] showed that the use of RNN RBM offers better performance than HMM, when using a language model on top of the acoustic model proposed by Nam et al. [77].

Thickstun et al. [109] proposed a convolutional architecture for polyphonic music transcription, that extracts features from raw audio rather than using a time-frequency representation as input. A convolutional layer is used as a learnable filterbank that computes a spectrogram-like representation of a chunk of audio signal. After a pooling layer, a linear classifier predicts the probabilities of notes active within the considered time window. However, feature extraction from raw audio was discovered to be: not as good as, feature extraction from log scaled spectrograms Thickstun et al. [35, 108]. In their work, authors showed that a two layers CNN, extracting features from log spectrograms performs better than the previously proposed model trained on raw audio. This two layer network extracts features in two stages. The first layer extracts timbre features with a mono-dimensional filter ori-

ented along the frequency axis. Similarly, the second layer learns temporal features with a mono-dimensional kernel oriented along the time axis.

Chapter 3

Document segmentation and classification

In the new “Big Data” era of the Internet, there is a large and diverse amount of data available online including text documents, videos, music, and images. Recommendation and retrieval systems are becoming increasingly important to effectively interact with this growing amount of data. In order to build such systems it is important to extract content information to support more effective interaction with humans. Using humans to extract this information is not practical given the large amounts of data involved. For this reason several systems for automatic annotation without requiring human intervention have been proposed.

The ability to scan books and then perform OCR on them for the purpose of searching their contents is well known from efforts such as Google Books. Books for teaching music frequently contain musical score examples as well as text on the same page as can be seen in Figure 3.7. When the book is related to music we would like to be able to search not only the associated text but also the musical score example. Both musical score and text are important and they need to be processed independently with the proper OMR/OCR techniques. The ability to segment a document into these two different sources of information is a key aspect. In fact, both OMR and OCR algorithms need as input an image that contains solely the information they are expecting to process, in order to provide reliable results.

Regarding this application scenario, we propose an algorithm for the automatic segmentation of digital documents into musical score and text regions. In our intended application we have a large amount of scanned documents originating from a

large digital archive. These images contain musical scores and text regions that vary in terms of their layout and organization, as well as the types of text and notation symbols used. They include both, typeset and hand-written examples. The aim of our system is to identify and characterize these regions providing a classification label (musical score or text) and a corresponding bounding-box. Our document segmentation system plays an important role in the processing chain used to extract and organize information from the raw digital archive. In scanned pages that contain both musical scores and text, the direct use of existing Optical Music Recognition (OMR) and Optical Character Recognition (OCR) systems is problematic and results in extensive errors. This is because existing OMR and OCR systems are typically designed with the assumption that their input consists solely of a musical score or text respectively. Our proposed segmentation system can be used as a pre-processing step to make the use of OMR and OCR systems on mixed scanned pages reliable.

The rest of this chapter is organized as follows. In Section 3.1, we describe the proposed algorithm for music and text document segmentation. Section 3.2 specifies the dataset used and, how the training is conducted. Section 3.3 shows experimental results evaluating the system performance.

3.1 Algorithm description

In this section we describe our proposed algorithm for musical score/text document segmentation. Our system is able to detect the structure of a document that is represented as a list of bounding boxes each belonging to a particular class: musical score or text. No assumptions about the number and layout organization of these bounding boxes are made. The algorithm is composed of three fundamental steps, as shown in Figure 3.1. The first step is the Random Block Voting (RBV) procedure. This procedure extracts and classifies a fixed number of blocks from the image, whose position and size are sampled from a 2D random uniform distribution. The classification posterior probability for each block is also computed. Hence, for each block we compute a *vote* that is characterized by the a posteriori probability as well as the size and position. The next step consists in the computation of a coarse segmentation of the document, which we call the labeled image, obtained by summarizing all the previously computed votes in a single image. The last step is the final segmentation of the document which uses this labeled image as a guideline for a finer segmentation. Figure 3.2 illustrates both the coarse and final segmentation of an example document.

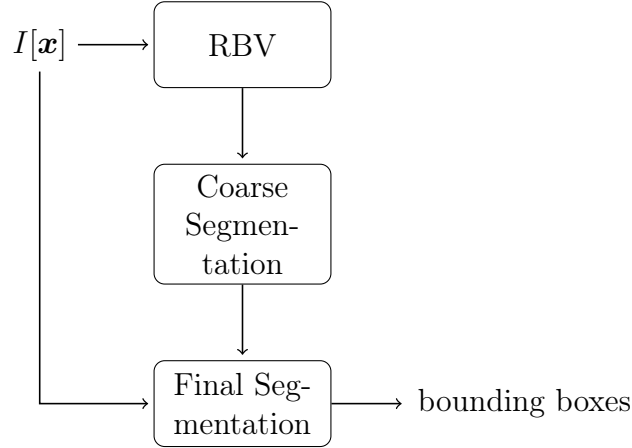


Figure 3.1: Segmentation algorithm.

In the following sections these three steps are described in more detail.

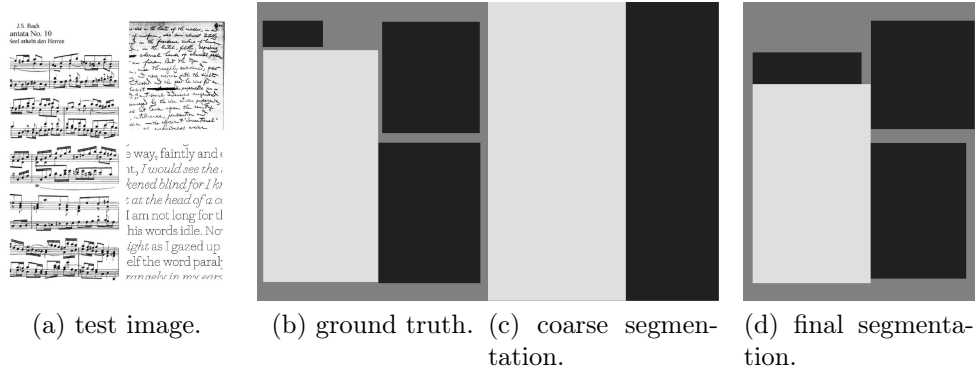


Figure 3.2: Segmentation steps of a test image.

3.1.1 Random Block Voting (RBV)

The RBV procedure consists in two steps: construction of random blocks and their classification into musical score or text. The two following sections describe these steps in more detail.

Random construction of blocks

The RBV procedure aims at obtaining a series of votes that gives a “local” and deliberately “redundant” classification of the image regions. To the best of our knowledge this procedure is a novel contribution especially in document image analysis. A vote

consists of: the a posteriori probability of classification, a size and a position. By using these votes it is possible to reconstruct, through the subsequent processing steps, the structure of the scanned image. The set of votes “over”-covers the whole area of the input image with various amounts of overlap. This redundancy is used to overcome possible errors in the classification of individual blocks as well as different scale factors. We have empirically observed that after a certain number of classification attempts of slightly different blocks, the overall probability of correct classification of that area is increased. In our context, classification errors may occur because a block is too small, too big, or because it contains a lot of empty space. In cases like that, multiple overlapping classifications end up correcting these errors. The next part of the section gives a more formal description of this RBV procedure.

First we extract a set N_{rb} of random blocks from a given image $I[\mathbf{x}]$, of dimension $\mathbf{S} = (H, W)$, where $\mathbf{x} = (i, j)$ indicates the spatial position, H is the height and W the width. The size $\mathbf{s} = (h, w)$ of these blocks is uniformly distributed between a minimum and maximum value. These values are

$$\mathbf{s}_{min} = \left\lfloor \frac{\mathbf{S}}{\mathbf{D}_{max}} \right\rfloor, \quad \mathbf{s}_{max} = \left\lfloor \frac{\mathbf{S}}{\mathbf{D}_{min}} \right\rfloor \quad (3.1)$$

where $\mathbf{D}_{max} = (D_{max}^y, D_{max}^x)$ is the maximum size divisor, and $\mathbf{D}_{min} = (D_{min}^y, D_{min}^x)$ is the minimum size divisor. Given those objects we can define the step size as

$$\mathbf{S}_{st} = \frac{\mathbf{s}_{max} - \mathbf{s}_{min}}{\#step} \quad (3.2)$$

and finally define the discrete uniform probability distribution for the block size, that assumes the form:

$$U(\mathbf{s}) \sim \{\mathbf{s}_{min}, \mathbf{s}_{min} + \mathbf{S}_{st}, \mathbf{s}_{min} + 2\mathbf{S}_{st}, \dots, \mathbf{s}_{max}\} \quad (3.3)$$

The position $\mathbf{p}$ of the blocks is also sampled from a discrete uniform distribution. The distribution takes values between 0 and $\mathbf{p}_{max} = \mathbf{S} - \mathbf{s}$, where $\mathbf{s}$ is the size of the current block. The step size here is given by $\mathbf{P}_{st} = \mathbf{p}_{max} / \#step$ and the corresponding distribution is equal to

$$U(\mathbf{p}) \sim \{0, \mathbf{P}_{st}, 2\mathbf{P}_{st}, \dots, \mathbf{p}_{max}\} \quad (3.4)$$

Using these definitions we can now formalize the notion of a random block $B[\mathbf{x}]$ of

an image by using the following notation:

$$B^{p,s}(I) = \{I[\mathbf{x}] \mid \mathbf{x} \in [\mathbf{p}, \mathbf{p} + \mathbf{s}]\} \quad (3.5)$$

Thus, we end up with the set $\mathcal{B} = \{B_n^{p,s}(I)\}_1^{N_{rb}}$ of blocks spanning and overlapping the full area of the document.

Classification with BoVW

Each random block is then classified by means of a Bag of Visual Words (BoVW) representation into musical score, text or blank regions. The BoVW [67] is an extension of the bag of words method used in text classification to images. In this case a document is represented as a sparse vector of occurrence counts of words i.e. a sparse histogram over a fixed dictionary. In the case of BoVW, an image (or image block) is represented similarly as a sparse vector of words, but the histogram represents the occurrence counts over a learned dictionary (codebook) of image features. In Figure 3.3 we show the general flowchart for calculating the BoVW representation. First, a set of keypoints is detected and used to describe the image. Here we use the SURF [6] features that are essentially a computationally efficient variant of the well known SIFT [71] features. SURF is a multi-scale technique for finding keypoints based on the gradient and gradient distribution around them, where keypoints are detected based on pixels that maximize the determinant of the Hessian matrix. The description is given by the Haar wavelet responses in both x and y directions within a circular neighborhood of radius 6ρ where ρ is current scale. Moreover, SURF features exhibits scale and rotation invariance. In our applications, this characteristic is fundamental to achieve robustness with respect to musical score/text dimension and orientation. The dictionary is computed by K-means clustering of the feature descriptions of all the keypoints detected for each image in the training set. K is the number of words in the dictionary. Once the dictionary is computed an image can be represented as a count of the dictionary words by means of a vector quantization of its features description. Finally, a Random Forest classifier is trained using the BoVW representation of the same samples used in the previous step of codebook generation. It is possible for a block to be either completely empty or almost empty. In such situations, the number of detected features that is lower than a fixed threshold τ_f may be not suitable for a reliable classification. When this happens the block is classified as blank.

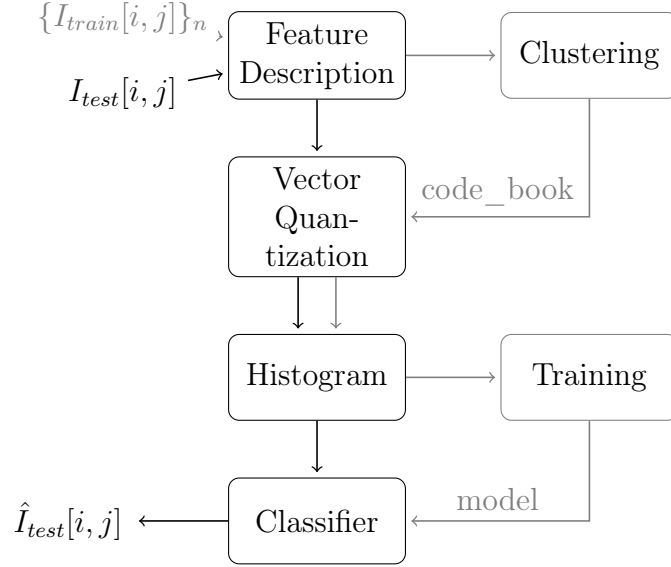


Figure 3.3: Bag of visual word testing/training scheme. Where the testing flow is indicated in black and the training one in gray.

Once the BoVW model is trained, we are able to predict the class of each random block and its posterior probability, according to the testing scheme depicted in Figure 3.3. More precisely we compute the set of votes:

$$\{v_n\}_1^{N_{rb}} = \{(\hat{y}_n, p_n(x|\hat{y}), (\mathbf{p}, \mathbf{s})_n)\}_1^{N_{rb}} \quad (3.6)$$

$$\hat{y}_n \in \{-1: \text{'blank'}, 0: \text{'text'}, 1: \text{'music'}\} \quad (3.7)$$

where $\hat{y}$ is the classification, $p(x|\hat{y})$ the posterior probability and $(\mathbf{p}, \mathbf{s})$ is the tuple consisting of position and size.

3.1.2 Coarse segmentation

The labeled image represents a coarse segmentation of the document where musical score/text regions are detected by combining the votes at the pixel level. The resulting image (of the same resolution as the original image) defines regions that are possibly bigger and noisier with respect to the actual content and layout of the document. That is because blocks are extracted randomly within the image without taking into account a predefined structure. Moreover, this stage doesn't consider a neighboring context for the classification of a particular pixel. These considerations motivated the final segmentation step described below.

The labeled image $L[\mathbf{x}] \in \{-1, 0, 1\}$ is created by summarizing all the votes $\{v_n\}_1^{N_{rb}}$ in a single image. This is accomplished by selecting for each pixel position, the class that maximizes the total probability of all the votes relative to the particular pixel (i.e from all overlapping random blocks that contain that pixel). More specifically the labeled image can be notated as follows:

$$L[\mathbf{x}] = \underset{\hat{y} \in \{-1, 0, 1\}}{\operatorname{argmax}} \sum_n p(x_n | \hat{y}_n) \Big|_{\mathbf{x}} \quad (3.8)$$

$$\forall \mathbf{x} \in [1, H] \times [1, W]$$

3.1.3 Final segmentation

The next step consists in obtaining the final segmentation of the document into musical score and text regions by utilizing the labeled image as a guideline. This process, depicted in Figure 3.4, is composed of four phases: subdivision of microblocks, validation of microblocks, nearest neighbor classification and connected components analysis. The first step consists of subdividing the image into 16×16 non-overlapping

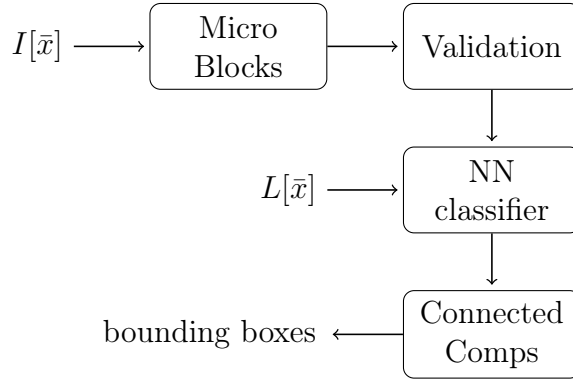


Figure 3.4: Final segmentation flowchart.

microblocks. The next step validates each one of them for the subsequent classification. Since these documents generally contain blank regions, before proceeding with the final classification, we discard the microblocks that contain low amount of information. The validation criterion is based on the ratio of *informative pixels* per region. More specifically, the image is binarized by using an adaptive gaussian thresholding in order to roughly separate background, which is set to 0, and foreground, which is set to 1. A microblock $B_n^\mu(I)$ is valid if the sum of the foreground elements normalized

by its dimension is greater than a fixed threshold Θ as in:

$$\frac{\sum_{\mathbf{x}} B_n^\mu(I[\mathbf{x}])}{16 \times 16} > \Theta \quad (3.9)$$

In the next step, each valid microblock is classified into musical score or text by a K Nearest Neighbor classifier (K-NN) which is trained using the labeled image. Neighbors are defined spatially in a grid over the labeled image. The non-valid microblocks are automatically set to blank. By using a K-NN classifier, the surrounding context of a microblock is taken into consideration. The final step consists of computing the bounding boxes of the musical score and text regions. For doing so, connected components are extracted from the fine microblock segmentation of the document by using the region growing algorithm [86]. The final output is further improved by removing components with a number of elements below a fixed threshold, or components with unreasonable aspect ratio such as “line shaped” ones.

3.2 Datasets

In order to evaluate the performance of our segmentation algorithm, we have created two different datasets. One is *artificial*, and is manually constructed by putting together regions of musical scores and text coming from a pool of samples images for each test image. The other dataset is *real*, and is constructed by the scanning two physical music books. These two datasets have different properties and are used to evaluate different aspects of the proposed segmentation algorithm.

The artificial dataset is used to asses the recognition performance among instances of the two classes, that present high intraclass variance. With this test we aim at evaluating the scalability of our algorithm up to a huge variety of typographic styles. This is an important aspect, and allows us to to deploy our algorithm as a general service that can be trained once and used many times. On the other hand, the scanned dataset is used to estimate the capability of our algorithm to perform good layout segmentation. What we mean with layout segmentation in this paper has a slightly different interpretation with respect to the classical definition of layout segmentation in document analysis[95]. Our main intereset in this work is to separate the contents of a document page in such a way they can be correctly given as input to an OMR/OCR system. Therefore, by layout segmentation we mean the ability of our algorithm to identify rectangular regions that isolate musical score and text. As mentioned before,

the artificial dataset is constructed in such a way that the resulting layout can not be considered a good representation of what happens in real music books. For this reason we scanned the most challenging page of two music books. One is recent, written in english, and follows a modern typographic layout. The other has a vintage layout and it is written in Russian. The following subsections discuss in more detail the characteristics of the datasets.

3.2.1 Artificial Dataset

The artificial dataset is created by downloading pictures of musical scores and text from Google Images. We chose images with height is at least 768 pixels. In our opinion this is a reasonable criterion since scanned documents at a good resolution will easily end up bigger than this. Moreover the BoVW representation, especially when operating on blocks as in our algorithm, needs a decent number of features (greater than 30 – 50) to be effective.

The *musical score* class consists of 171 RGB images. These images mainly depict musical scores that span multiple lines, sometimes along with a title with dimension/-font that is frequently different depending on the particular image. The average of the images size ¹ is 1110×1274 with a standard deviation of 503×621 . In order to be able to represent a large variety of musical scores, the images that define this class contain a considerable amount of variability in quality, font, type (handwritten/printed), and layout. Figure 3.5a shows some examples.

The *text* class is composed of 181 RGB images. In this case the images contain text which is organized either in a single or multiple columns, or as a sparse collection of words. The average size of the images is 965×1085 with a standard deviation of 418×588 . For the same reasons stated for the musical score class, the text class presents variability in: font, dimensions, layout, type (printed/handwritten). Figure 3.5b shows some examples.

Testset

With this dataset we aim at providing a challenging collection of documents that are artificially combined. The testset consists of 40 images created by putting together text and musical score in a single container. The content used to create these images was removed from the training set. Figure 3.6 shows some examples. Each image

¹width $\times$ height



(a) Musical score.

Bandolero Elegante
Cruz Handy Vera Cruz
Bandolera Swinger
Cantina Cruz Grafica
Cruz Brush Troubadour
Cruz Calligraphic CRUZ
Cruz Cañtera STENCIL
Fat Albert Cruz Ballpoint
Romantica Homeland

Handwritten text in German, likely a letter or a page from a manuscript. The text is written in a cursive script and is somewhat faded.

1. Die Bauernstube.

Wie einfach ist die Bauernstube ausgestattet! Ein hölzerner Tisch in der Mitte der Stube, daneben ein Holzhuhl und eine Bank, links eine Kommode und am Ofen ebenfalls eine Bank — das ist alles, was der Bauer für sich und seine Familie braucht. Nach des Tages Arbeit schmeckt ihm das einfache Essen trefflich, das ihm seine Frau aufgetragen hat. Auch der kleine Bursche auf der Bank ist kräftig mit; während der Hund sehnüchelt wartet, ob für ihn nicht auch etwas abfällt. Der Knecht hat noch zu thun gehabt; er kommt daher zu spät zum Essen. Sorglos schläft das Kindchen in der bunt bemalten Wiege; es weiß noch nichts von den Mühen und Sorgen dieses Lebens. Die Bauerfrau arbeitet nicht nur fleißig im Hause und auf dem Felde, sie denkt nicht nur eifrig das Spinnrad, sie hat auch Sinn für etwas Schönes: mehrere Silberstücken die Wand, und in den Fenstern stehen Blumen. Trotz vieler und schwerer Arbeit vergessen diese einfachen Leute doch ihren Gott nicht. Der Getreueste über dem Bette weist darauf hin, daß sie nicht nur arbeiten, sondern auch beten.

(b) Text.

Figure 3.5: Examples of images.

contains multiple instances of musical scores and text segments, and each one of these exhibits different characteristics such as dimension, font and scale. In some cases, we also apply distortion (affine transformation) to the musical score and text region in order to evaluate the robustness of the proposed algorithm. The ground truth is manually generated by a human annotator utilizing our custom annotation tool, that is part of the software implementation.

Training

The actual training is performed on random blocks extracted from the original full images. The number of random blocks per image is empirically set to $N_{rb} = 15$. The other parameters used in our experiments are: maximum divisor $D_{max} = 6$, minimum divisor $D_{min} = 3$, size step $S_{st} = 8$ and position step $P_{st} = 8$. Therefore the dataset ends up being composed of 2580 random samples for the musical score class and 2715 for the text class where each image corresponds to a random block.



Figure 3.6: Test set examples.

The number of codebook words K , is chosen performing a grid search evaluation. Cross-validation experiments of the random forest classifier, reported in Table 3.1, show that the best accuracy is achieved with $K = 80$. We see from Table 3.1, the

Table 3.1: K grid search with 3 fold cross-validation.

K	8	16	32	64	80	100
accuracy [%]	94.5	95.4	95.6	96	97.2	95.7

accuracy is very high in all cases, and does not vary too much, with values that increase as the number of codebook words increases, until it reaches a saturation point at $K = 100$ words. This behavior is explained by the nature of BoVW. Usually this representation is used to distinguish among a relatively large number of classes. In our application the BoVW is used to distinguish only between two classes: musical score and text. Due to the nature of musical score and text images, there is no a real need for a high dimensionality dictionary. Since there is no significant difference in terms of computational time during the tests between low and high K , we select the K that maximizes the accuracy. From this experiment we can note again how the chosen representation of information is powerful and scalable.

Finally a random forest classifier, with 40 estimators, is trained by using the the codeword occurrences histogram computed through vector quantization using the L^2 distance as the similarity measure.

3.2.2 Real Dataset

The real dataset is created by scanning pages of two books: one modern and one vintage. We selected the pages that present balanced amounts of text and musical scores in order to obtain an equal distribution between the two classes. All the pages are scanned at 300dpi of resolution, resized to a maximum height of 1024 pixels and finally converted to jpeg format. The same considerations regarding the number of features discussed in the previous section hold here. However, in this case we go from a big resolution to a smaller one. In fact, our algorithm doesn't need high resolution images to perform well. For these reason we prefer to operate on smaller images, speeding up the test time calculation.

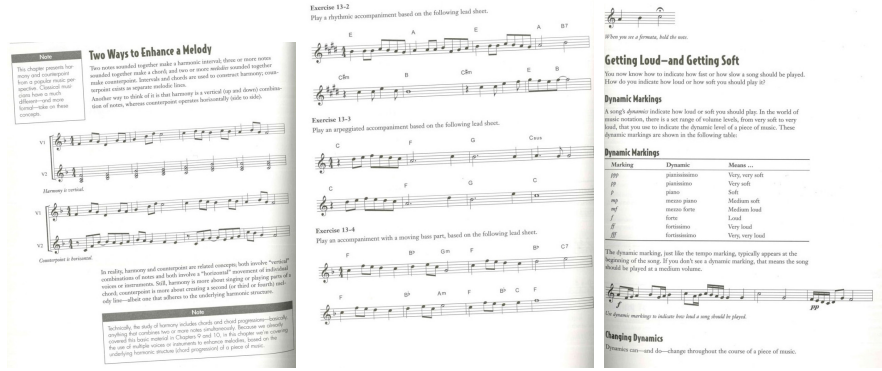
The modern book dataset consists of 52 images. The average of image size is 690×884 with a standard deviation of 20×27 . As we can see from Figure 3.7a, images of this book may contain musical scores, text, titles, side-notes and even tables. And they also present the typical common scanning artifacts such as skew and slope. The vintage book dataset contains 35 images. The average image size is 722×950 with a standard deviation of 12×21 . This time, we notice a considerably different typographic style, besides the different alphabet. Text is organized in columns and the titles are more difficult to differentiate from the general text, and the background tends to be yellowish because of paper aging.

As a common consideration, the typographic style remains constant throughout each book. This make the training procedure easy because each class can be described with just a few training samples.

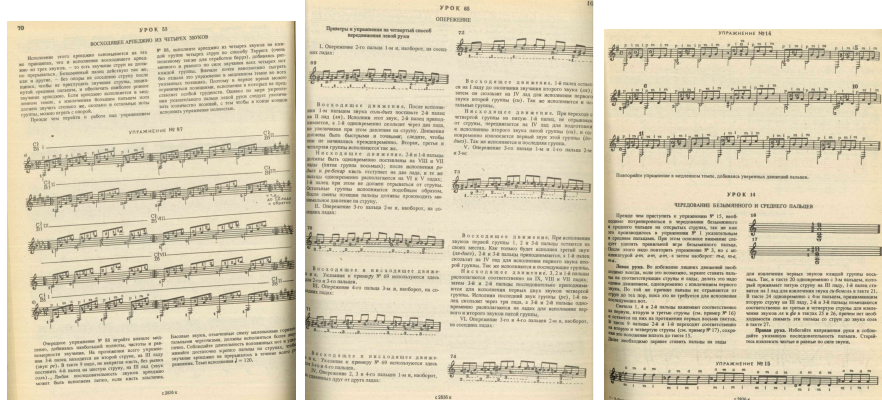
Training

The training of the BoVW model for the real books follows the same principles describe in Section 3.2.1. However in this case, training doesn't play the same fundamental role as it does with the artificial dataset. With the real dataset we are more interested in evaluating the layout segmentation performance of our algorithm. For this reason, we decided to train each model independently, in order to minimize other possible sources of errors.

Therefore, we create our training set extracting the musical score and textual information of the first 3 pages of each book. The number of random blocks per image is empirically set to $N_{rb} = 50$. The selection of other parameters in this case here has as bigger impact to the performance of the algorithm. The general idea to



(a) Modern book.



(b) Vintage book.

Figure 3.7: Examples of scanned image.

be able to detect small typographic details is to have: more blocks, more positions and a wider range of smaller sizes. Through some empirical tests we have found that $D_{max} = 12$, $D_{min} = 6$, $S_{st} = 8$, $P_{st} = 32$.

Regarding the choice of K , we selected the smallest value that allows us to obtain an accuracy greater than to 90% with a 3 fold cross-validation test, which results in $K = 16$.

3.3 Experimental results

In this section we evaluate the proposed algorithm by utilizing a challenging dataset of images combined artificially. These images are assembled by gathering multiple instances of text and musical score that present high intra and inter image variation and combining them in a single image. All the tests conducted in this section utilize the precision/recall of the overlapping area between detected bounding boxes and the

ground truth as evaluation metrics. A more formal definition of this metric is given below.

Let's first introduce some mathematical objects required by the definition. The ground truth G of a test image is represented as a list of n labeled bounding boxes defined according to the following formulation:

$$G(I_{test}[\mathbf{x}]) = \{\Omega, (\mathbf{p}, \mathbf{s})\}_1^n \quad \Omega \in \{\text{'text'}, \text{'score'}\} \quad (3.10)$$

The corresponded segmentation, $S(\cdot)$, produces the same kind of output which in turn consist of a list of labeled bounding boxes. The indicator function χ of the ground truth and segmentation with respect to a certain class is defined as:

$$\chi_k^\Omega(I[\mathbf{x}]) = \begin{cases} 1 & \text{if } \exists g = \{\omega, (\vec{p}, \vec{s})\} \in G \text{ s.t.} \\ & \omega = \Omega \wedge \mathbf{x} \in [\mathbf{p}, \mathbf{p} + \mathbf{s}] \\ 0 & \text{otherwise} \end{cases} \quad (3.11)$$

$$k \in \{G, S\}, \Omega \in \{\text{'text'}, \text{'score'}\}$$

Given these objects we can now define the precision, recall and F-measure of a certain class

$$\begin{aligned} p_\Omega &= \frac{\sum_{\mathbf{x}} \chi_G^\Omega \cap \chi_S^\Omega}{\sum_{\mathbf{x}} \chi_S^\Omega} \quad \Omega \in \{\text{'text'}, \text{'score'}\} \\ r_\Omega &= \frac{\sum_{\mathbf{x}} \chi_G^\Omega \cup \chi_S^\Omega}{\sum_{\mathbf{x}} \chi_G^\Omega} \quad \Omega \in \{\text{'text'}, \text{'score'}\} \\ F_\Omega &= 2 \frac{p_\Omega r_\Omega}{p_\Omega + r_\Omega} \quad \Omega \in \{\text{'text'}, \text{'score'}\} \end{aligned} \quad (3.12)$$

3.3.1 Baseline approach

The baseline approach aims at detecting only the musical score regions within a digital image without having the capability of identifying the text areas. Our initial intention was to use the page segmentation output of OCRopus [18], which ideally removes everything that is not detected as text within a document page, and use it as a deletion mask for the purpose of isolating musical scores. However, the layout segmentation provided by OCRopus, for our test images, was too unreliable for it to act as a reasonable baseline. Figure 3.8 shows to examples of OCRopus page segmentation, as can be seen, OCRopus fails at removing a large portion of musical score in both the documents. This is not surprising, this tool wasn't developed to be



Figure 3.8: Examples of OCRopus page segmentation output.

applied to this kind of documents. For this reason, we based our baseline approach on top of a well-known, more reliable, staff line detector.

Therefore, our baseline approach is an extension of a known technique for music staff detection that can identify the bounding box for musical score regions in images that contain both music scores and text. This can be viewed as an attempt to assess the scalability of a traditional staff line detection method for the purpose of documentation segmentation into musical score and text. The baseline method is based on the Gamera software framework for Optical Music Recognition (OMR) [38] and the music staves detection toolkit proposed in Dalitz et al. [31]. By using these tools it is possible to detect candidate staff lines which span most of the width of the image. However, since we are interested in detecting bounding boxes of musical scores we propose an extension of this method which is able to identify these regions by merging multiple sequences of staff lines. Ideally staff lines come in groups of five parallel lines which are separated by a larger space with respect to the intra-spacing of the group. Moreover due to the nature of typical musical scores, it is more common to observe a small, rather than a big, spacing between staff lines. For this reason the basic idea is to model the distribution of staff line spacing in terms of a mean and variance. These quantities will be used to define an intra-spacing threshold τ_s value. Lines whose intra-spacing is less than τ_s are grouped together in a single bounding

box. In particular, given an ordered set $\mathcal{S}$ of staff lines defined by its y coordinate,

$$\mathcal{S} = \{y_i \mid y_i < y_{i+1} \quad \forall i = 1, 2, \dots, N-1\} \quad (3.13)$$

we compute the discrete derivative

$$\mathcal{S}' = \{y_{i+1} - y_i \quad \forall i = 1, 2, \dots, N-1\}, \quad (3.14)$$

its mean $\mu(\mathcal{S}')$, and standard deviation $\sigma(\mathcal{S}')$. The threshold is defined by

$$\tau_s = \mu(\mathcal{S}') + \sigma(\mathcal{S}') \quad (3.15)$$

In order to identify the musical score bounding boxes we proceed by scanning the lines in sequential order according to Algorithm 1. At this point the detected bounding boxes have a width that is equal to the image width. In some cases this represents a coarse segmentation which can be further improved by identifying a tighter block width. We improve the segmentation by identifying two vertical lines that enclose the content of the document discarding eventual margins. The two lines define the x_{min} and x_{max} common coordinate that ultimately characterizes each block. These two coordinates are obtained by a vertical projection ψ_x of the image. The first step is to obtain a binary image by using the Otsu thresholding technique [82]. Given threshold τ_{otsu} the binary image is recovered by

$$I_{otsu}[i, j] = \begin{cases} 1 & \text{if } I[i, j] > \tau_{otsu} \\ 0 & \text{otherwise} \end{cases} \quad \forall i, j \quad (3.16)$$

which is used to compute the projection,

$$\psi_x[k] = \frac{1}{M} \sum_{i=1}^M I_{otsu}[i, k] \quad k = 1, 2, \dots, N \quad (3.17)$$

In order to identify x_{min} and x_{max} we define another threshold τ_x . This threshold represents the minimum amount of information of a column, which is required to be considered part of the documents content. We empirically set $\tau_x = 0.05$ and finally

obtain the x coordinates

$$\begin{aligned} x_{min} &= \underset{k}{\operatorname{argmin}} (\psi_x[k] > \tau_x) \\ x_{max} &= \underset{k}{\operatorname{argmax}} (\psi_x[k] > \tau_x) \end{aligned} \quad (3.18)$$

```

 $box_y \leftarrow [];$ 
 $start \leftarrow 1;$ 
 $i \leftarrow 2;$ 
while  $i \leq N - 1$  do
    if  $\mathcal{S}'_i \geq \tau_s$  then
        if  $\mathcal{S}'_{i-1} < \tau_s$  then
             $box_y \leftarrow [box_y, [\mathcal{S}_{start}, \mathcal{S}_i]];$ 
        end
         $start \leftarrow i + 1;$ 
    end
     $i \leftarrow i + 1;$ 
end

```

Algorithm 1. Staff line merging.

3.3.2 Artificial Dataset Experiments

Baseline algorithm

Since the baseline algorithm is limited to solely detect musical scores, in this experiment we consider only those regions of the test images. Table 3.2 reports the achieved mean, and standard deviation, precision/recall obtained over the entire dataset. In-

Table 3.2: Performance of the baseline approach on the artificial dataset.

	Precision [%]	Recall [%]	F-measure [%]
music	58.7 ± 24.5	57.1 ± 14.4	54.2 ± 15.5

stead, Figure 3.9 reports the performance results for each instance of the testset.

From this experiment it is clear that the baseline approach performs poorly in the segmentation task. The principal explanation of this low performance is related to the typical application domain assumptions of staff-line detection algorithms. First of all, these algorithms require as input images that solely contain musical scores, with the

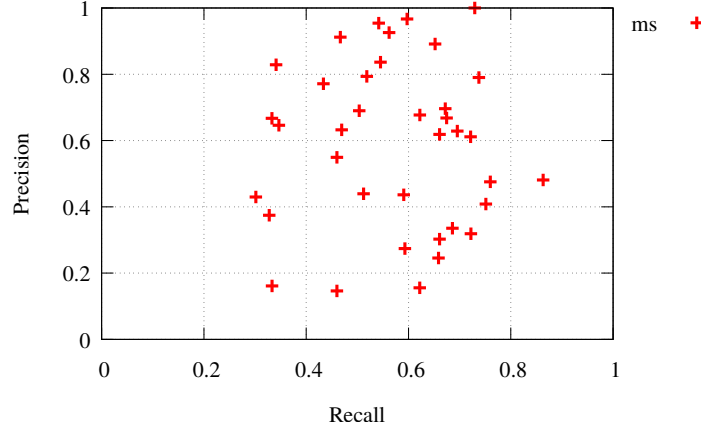


Figure 3.9: Baseline approach evaluation for each image.

reasonable common assumption that staff-lines horizontally span most of the column width. According to the design of our artificial dataset, where text and musical scores are placed without a layout assumption, these requirements are frequently not satisfied ². Moreover, our test images contain hand-written musical scores, at a relative low resolution, which can be considered the most challenging scenario for staff-line detection algorithms.

Finally, with this experiment we show that staff-line detection algorithms are not reliable enough to build on up more advanced processing techniques. However, in contrast, our proposed method for musical score and text document segmentation deals more effectively with these issues.

Proposed algorithm

The parameters used in our evaluation experiments are: number of random blocks per image $N_{rb} = 100$, number of nearest neighbors $N_{nn} = 100$, microblocks size 16×16 , maximum divisor $D_{max} = 7$, minimum divisor $D_{min} = 3$, size step $S_{st} = 8$, position step $P_{st} = 8$, number of features threshold $\tau_f = 40$ and microblocks validity threshold $\Theta = 0.1$. The evaluation results demonstrate the potential of the proposed algorithm. The 3-fold cross validation test conducted on the random blocks used during the training phase reports a mean accuracy of 97%.

Figure 3.12 shows four examples of segmentation of test images. Figure 3.12a, Figure 3.12b and Figure 3.12c show the good segmentation abilities of our algorithm in challenging images where all text and score regions are correctly detected and iden-

²Conversely, the real dataset meets the width requirement in the majority of the cases

tified. These examples also illustrate the robustness of the algorithm with respect to intra-class variability; in fact both musical scores and text, whether handwritten and printed, are correctly identified. On the other hand, Figure 3.12d depicts a partially bad output. The upper text block is both not entirely identified and incorrectly classified. This is due to the distortion that we apply, indeed in the training dataset there are no such distorted examples from which to learn. At the same time, one can observe the robustness against the rotation of the bottom left score. This is possible due to the rotation invariance of the SURF feature representation.

Finally we report in Table 3.3b the average precision and recall obtained by the proposed system on the entire test set. Figure 3.10 depicts the precision/recall evaluation metrics obtained for each image individually. As can be seen for the majority of test images the algorithms performs very well.

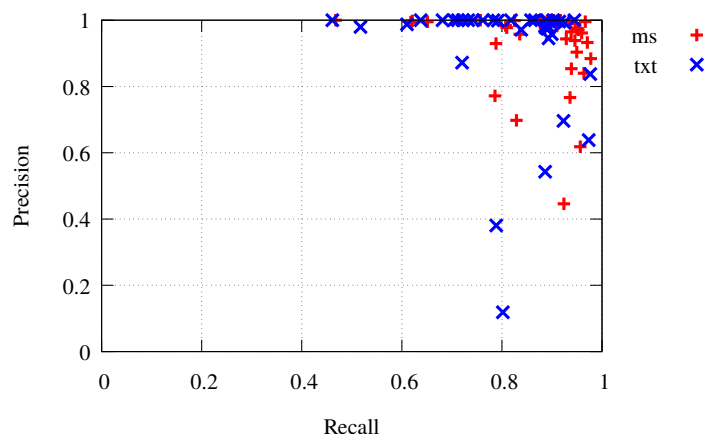


Figure 3.10: Detection performance of the proposed system.

From Table 3.3b, we can note good performance, both in terms of precision and recall, achieved by our method, that largely outperforms the baseline musical score detection approach. We can also observe how performance is balanced between classes. This highlights the capability of the system to distinguish, with the same ability, the two different type of content.

In addition, we also report the performance obtained just by the coarse segmentation without executing the final segmentation step. Since this segmentation is in general less precise with respect to the actual content of the document, the expectation is that the recall should be higher than the precision, and that the precision should be lower than the average scored by the full algorithm. Table 3.3a confirms this expectation and, Figure 3.11 shows the precision and recall evaluation metrics

obtained by the coarse segmentation for each individual image of the testset.

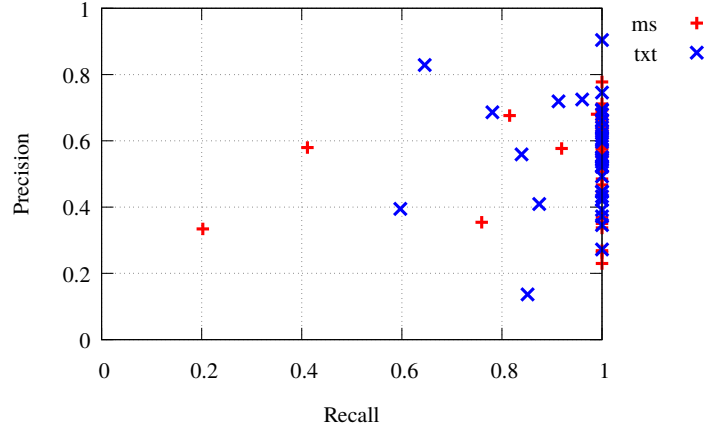


Figure 3.11: Detection performance of the coarse segmentation.

Finally, as our last experiment we report the performances obtained by our algorithm replacing the RBV procedure with the uniform block classification one. The input image is subdivided into non overlapping blocks of size $H_u \times W_u$, and each one of them is classified into musical score or text by means of the same BoVW classifier utilized in the previous experiments. The coarse segmentation is directly computed by the uniform block classification procedure. Subsequently, this intermediate result is further improved by the final segmentation phase which is tuned in the same way as the previous tests.

Table 3.3c reports the achieved precision/recall using three different block sizes: small, intermediate and large. From Table 3.3c, we can note how the block size parameter influences the performance of the uniform block segmentation. With small block sizes there is no capability of detecting text regions, while the detection of musical score has low precision. Instead the recall is pretty high since the majority of blocks are classified into musical score. By using an intermediate size, the uniform segmentation scores good precision/recall results, that are lower than the RBV segmentation but still comparable. On the other hand, with large block size the performances start to decrease, but in contrast to small block sizes, text regions are still detected although with low performance. From the prior discussion emerges that choosing the ideal block size is important to obtain good segmentation results. The optimal block size may change from document to document and it would be inconvenient to exhaustively search for it, image by image. The RBV procedure, in some sense, embeds the computation of this important parameter and makes it possible to obtain superior

segmentation results with respect to the uniform document segmentation. Finally, from Table 3.3c we can note that the standard deviation of precision/recall of the uniform segmentation approach is in general higher than the RBV one. This means that our algorithm in general assures better reliability of segmentation, and for that reason, can be more successfully employed in the processing of large volume of data.

Table 3.3: Performance results on the artificial dataset.

(a) Coarse segmentation				
		Precision [%]	Recall [%]	F-measure [%]
music		54.5 ± 14.0	94.9 ± 16.3	67.6 ± 14.8
text		55.6 ± 15.4	95.8 ± 9.7	68.8 ± 13.4
average		55.0 ± 14.7	95.4 ± 13.0	68.2 ± 14.1

(b) RBV segmentation				
		Precision [%]	Recall [%]	F-measure [%]
music		92.5 ± 12.1	86.8 ± 11.5	88.5 ± 9.2
text		91.4 ± 19.2	81.2 ± 12.0	83.8 ± 14.6
average		92.0 ± 15.7	84.0 ± 11.8	86.1 ± 11.9

(c) Uniform segmentation				
	$H_u \times W_u$	Precision [%]	Recall [%]	F-measure [%]
music	50×50	62.1 ± 23.4	90.8 ± 7.1	70.9 ± 18.2
	150×150	94.1 ± 9.8	80.0 ± 16.5	85.1 ± 12.0
	300×300	71.0 ± 30.0	77.4 ± 29.2	72.2 ± 27.6
text	50×50	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
	150×150	79.5 ± 24.9	82.0 ± 12.2	77.5 ± 19.0
	300×300	57.0 ± 41.4	49.9 ± 36.7	51.0 ± 36.3
average	50×50	31.1 ± 11.7	45.4 ± 3.6	35.5 ± 9.1
	150×150	86.8 ± 17.4	81.0 ± 14.4	81.3 ± 15.5
	300×300	64.0 ± 35.7	63.7 ± 33.0	61.6 ± 32.0

3.3.3 Real dataset experiments

In this section we discuss the performance obtained by our algorithm, and the baseline one, regarding the layout analysis of real scanned music books. In this experiments,

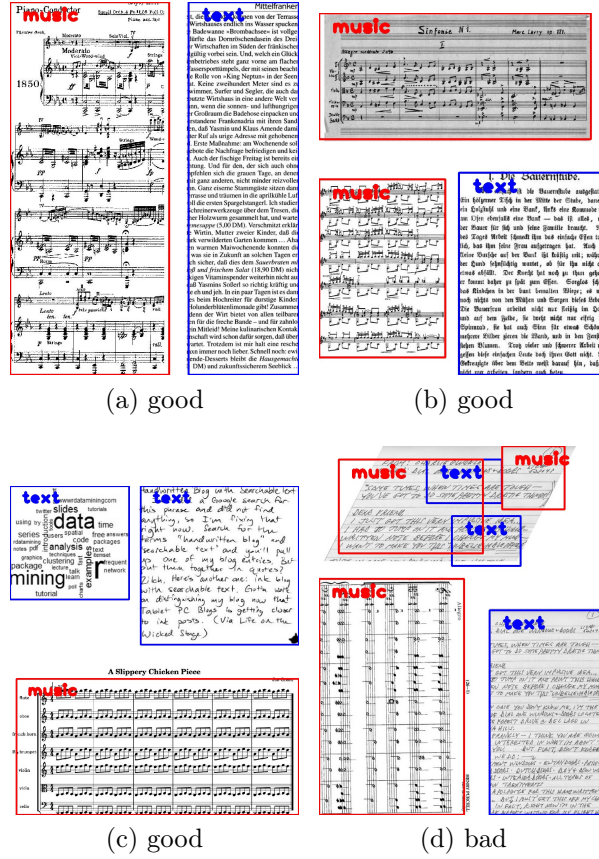


Figure 3.12: Examples of document segmentation.

the choice of the RBV parameters, especially the number of blocks and their dimension, has a big impact on the final performance. In particular, in order to perform a good segmentation, we need a large number of blocks with relative small sizes and wider range of values. We recall that, for the purpose of a reliable block classification, it is required to have a decent number of features inside each block. Therefore, the smallest block size can't be arbitrary small. The parameters used in our evaluation are: $N_{rb} = 400$, $N_{nn} = 50$, $D_{max} = 16$, $D_{min} = 8$, $S_{st} = 8$, $P_{st} = 32$, number of features threshold $\tau_f = 40$, validity threshold $\Theta = 0.1$. A constraint in parameter selection is to be able to perform a segmentation of an image in a reasonable computational time. For this reason, the chosen number of random blocks, which is the parameter that mostly affects time performance, is a trade-off between: segmentation time per image, and the capability of S to detect details of the images used in this experiment.

From Table 3.4b we can notice that, also in the case of real scanned documents,

Table 3.4: Performance results on the real datasets.

(a) Baseline approach				
		Precision [%]	Recall [%]	F-measure [%]
Modern	music	28.2 ± 23.3	46.5 ± 29.4	32.8 ± 19.6
Vintage	music	65.2 ± 25	54.4 ± 17.9	56.3 ± 18
Average		46.7 ± 24	50.4 ± 23.6	44.5 ± 18.8

(b) RBV segmentation				
		Prec. [%]	Recall [%]	F-meas. [%]
Modern	music	86.1 ± 17.3	81.2 ± 19.3	84.7 ± 8.6
	text	84.6 ± 14.4	90.7 ± 12.1	86.5 ± 12.2
	avg	85.3 ± 15.8	86 ± 15.7	85.6 ± 10.4
Vintage	music	83.6 ± 11	95.5 ± 7	88.3 ± 7.4
	text	80.5 ± 10.6	84.6 ± 16.2	81 ± 12.5
	avg	82 ± 10.8	90 ± 11.6	84.6 ± 10
Average		83.7 ± 13.6	88 ± 13.6	85.1 ± 10.2

our segmentation algorithm performs well. In fact, both for the modern and vintage book, we obtain an average F-measure which is higher than 85%.

Figure 3.13 depicts four examples of segmentation computed by our algorithm, relative to the modern book. In Figure 3.13a, 3.13b and 3.13c we see example of good segmentation. In all of those cases both musical score and text are correctly identified. Moreover, we can notice the capability of our algorithm to correctly classify both tables, notes and sidenotes as text regions. On the other hand, Figure 3.13d shows a bad segmentation result. In this example, text regions are mostly correctly identified. However, the large part of musical score is either miss classified or undetected. As we can see from Figure 3.13d, the musical scores are almost empty. In correspondence of those regions, when we perform RBV, we end up with blocks that may contain just 5 straight parallel lines. A very small number of features are detected in those cases, which has the consequence of obtaining lots of unlabeled blocks.

In Figure 3.14, where we report four examples of segmentation executed on the vintage scanned book. In general we obtain good segmentation results, where musical scores and text are correctly detected. Some of them are not perfect, such as the text box at middle of Figure 3.14c, but it can still be considered good, since the majority

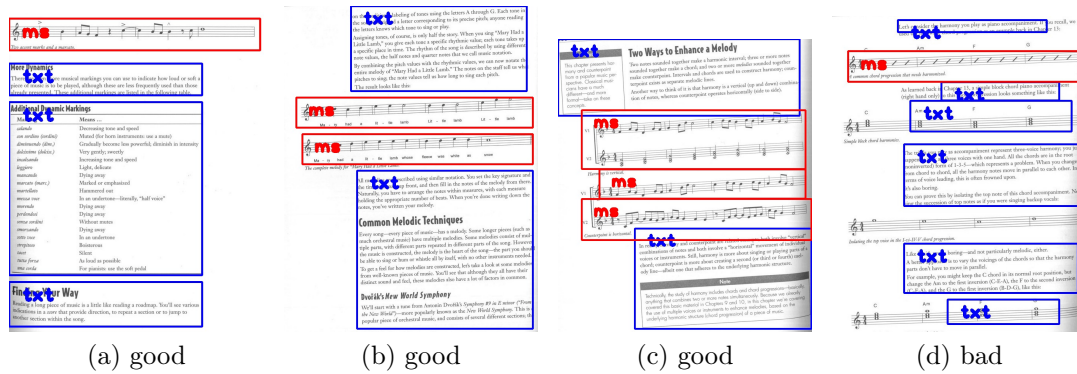


Figure 3.13: Examples of modern book segmentation.

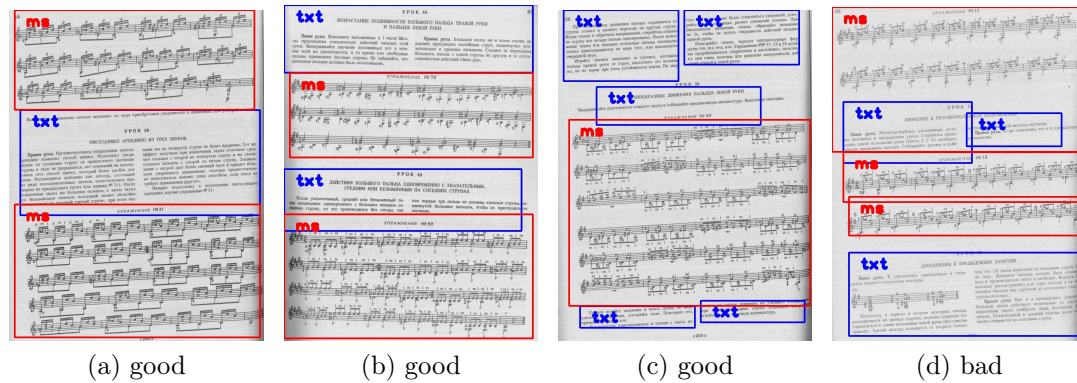


Figure 3.14: Examples of vintage book segmentation.

of the text is detected. Figure 3.14d shows a bad segmentation result where a musical score is miss classified into the text box at bottom. In addition, due to some boundary artifacts, the musical score box detected at top, ends up containing some unwanted text.

Finally, if we analyze the overall average results in term of F-measure, we notice that our algorithm performs well, with a similar score of 85%, in both the experiments (artificial and real). These experiments support that our algorithm is capable, at the same time, to perform a good classification of musical score and text under high intraclass variability, and a good layout segmentation.

Also in the case of real documents, the baseline approach, compared to our segmentation algorithm, performs poorly in the detection of musical scores. As Table 3.4a reports, the recall of the baseline method is comparable between both the books, on the other hand, the precision of detecting musical scores is consistently lower for the modern book. In this very case, the low precision value is mainly due to two reasons.

First, we noticed that in the modern book the probability of having text given a document image is in general higher than the probability of having musical scores. This has the consequence of making the baseline approach more likely to make a wrong classification. Second, the layout organization of the modern book presents lines of text that, as in the the case of staff lines, span the most of the columns of the image. These text lines can potentially be treated as staff “candidates” and in many cases are wrongly identified as staff lines. Conversely, the vintage book contains in average more musical scores than text, and the layout is organized in such a way (i.e breaking the text into multiple columns) that is more distinguishable, from the prospective of the baseline algorithm, than a staff line. For these reasons, the achieved precision of the baseline method in detecting musical scores is consistently higher than in the case of the modern book, but still overall largely inferior to our proposed segmentation algorithm.

Chapter 4

Efficient computation with binary representations

This chapter describes how binary representations can be used to provide memory efficient and fast implementations of algorithms in two different domains. Section 4.1 describes a proposed framework for binary image processing and Section 4.2 describes how binary techniques can be used for efficient binary neural networks.

4.1 SPmat

Binary image processing algorithms are often found in scenarios such as document analysis and industrial machine vision. In these applications, complex recognition procedures are constructed on top of fundamental algorithms on binary images such as binary morphology, run-length extraction, contour extraction and thinning.

Optimized fast processing is important in large scale image processing as well as on embedded devices that need to process data in real time with limited computational resources. For example, consider the scenario of digitizing a large collection of documents. Optical Character Recognition (OCR) systems convert scanned images that represent the physical documents into text. The required computations include binary image processing. Speeding-up these operations through optimized software implementations can increase the digitization throughput. Furthermore it opens the possibility of embedding the OCR process on the copying machine hardware. Efficient binary image processing is also important in industrial applications where limited computational resources are available and real time processing with

high throughput is desired.

The idea of bit-packing and the use of optimized bit-wise computations is not new, and it has been independently applied to a variety of binary image processing algorithms in the past [14] [69] [112] [58] [103]. The first immediate advantage of bit-packing is the reduction of the memory needed to store the image. In fact, instead of storing each pixel value in separate bytes, we can pack a sequence of pixels into a single broad computer word. By doing this, we are also able to process multiple pixels at the same time, gaining processing speed. The same concept of packing can be applied to the pixel neighbors as well. Many image processing algorithms when processing a single pixel, take into account the neighborhood of the pixel in order to compute the resulting output value. The advantage of neighbor packing in this case is two-fold: reducing memory access when processing a pixel, and opening up the possibility to use look-up tables in order to memorize computations. Our goal in this work is to combine the ideas of binary representation of pixels and binary representation of neighbours in a unified and general data representation that can be used for any binary image processing algorithm.

4.1.1 Proposed framework

The core of our framework is a data structure for effectively representing a binary image in bit-packed form, both for pixels and neighbors. Pixel-packing consists of storing 64 contiguous non-overlapping pixels of each image row in a single broad word. Similarly, we define the packed-neighbours which consists of storing, for each pixel, its $k \times k - 1$ neighbors. Packed-neighbors is actually a “super”-packed representation, because each single neighbors pack is also packed into a 64-bits word. Nevertheless, our API maintains the traditional bi-dimensional indexing scheme typical of matrices, so that the user does not have to deal with the underlying representations. Therefore functions for “setting” or “clearing” a pixel which maintain both packed pixels and packed neighbours in a consistent state are provided.

Regarding neighbourhood configurations, our framework supports the 3×3 configuration. In fact, this is the only configuration needed for implementing all the experiments. Although morphological processing tests require additional kernel sizes, such as the 5×5 and the 7×7 , the implementation in these cases does not rely on binary neighbourhood processing.

Packed and super-packed representations

The general idea of packed representations is to exploit, as much as possible, the register width of the most common CPU architectures. If we do not consider special instruction sets available for the x86_64 machines, such as MMX, SSE, and AVX, the most common register width nowadays is 64-bit. Our aim with bit-packing is to fit as much information as possible in the fastest parts of the memory hierarchy. The packed representation allows to bring into CPU registers 64-bit of information, that could be either pixels or neighbors, and process it “in place” without additional memory fetches. This is particularly useful in the cases when: the same operation is independently applied to blocks of pixels (e.g. morphological erosion/dilation), or when we need consider neighbouring values to make a pixel-wise decision (e.g. non look-up Guo-Hall thinning). Another important advantage of packed representations is the possible reduction of cache misses when accessing neighboring pixels of different rows. In fact, by having image rows that require 8 times less bytes of memory, it is less likely to have a cache miss. The latency of a cache miss usually requires lots of CPU cycles to fetch the needed data from main memory. Therefore, minimizing cache misses can substantially reduce the overall execution time.

The super-packed representation of neighbours provides the additional advantage of speeding-up the processing of consecutive packs of neighbors. This is in general true when we are updating the neighbours, after setting/deleting a pixel. These operations can update more neighbors at the same time within a super-pack. Additionally, super-packed neighbors are particularly useful when processing sequential neighbor packs. In fact, we can load a super-pack in a register and explicitly process the single neighbours packs in an efficient unrolled way.

As a final note, we clarify that the super-packed neighbors actually embeds the pixel information, which can be recovered by analyzing consecutive neighbor packs. This feature is particularly useful when the processing algorithm needs both the information of pixels and neighbours, as happens in the case of thinning. In this case, we can obtain the pixel information from the next neighbor pack, which is already cached, avoiding the memory access to an unrelated memory location, which would probably cause a cache miss.

Fast neighbours processing using table look-up

The packed-neighbors representation enables the speed-up of all processing algorithms that require access to pixel neighbors. This is a very common situation that happens in a many binary image processing algorithms. By doing this we can process a pixel with a single memory access, and use fast bit-wise techniques to access and process its neighborhood as required by the algorithm. In addition, this encoding scheme of neighbors leads to another efficient way of processing an image through look-up tables. By using look-up tables we can encode all the possible results of all the possible combinations of neighbors (when the neighborhood is small) in a single array. Instead of executing the required operations we can directly fetch the result at the table location defined by the neighbors themselves. This is especially useful when the neighborhood processing requires lots of computations, as is the case with thinning algorithms.

The packed-neighbors is efficiently constructed once, when the image is loaded into memory. During reading from file, for each ‘1’-pixel we call the `set` function, that does all the computations needed to properly set the data and neighbors fields. In fact, every time we change the value of a pixel it effects also the corresponding $k \times k - 1$ packed neighbors. To simplify this, we provide set/clear functions that automatically adjust the packed neighbours.

4.1.2 Algorithms and optimized implementation

Erosion and dilation

Erosion and dilation can be considered the basic elements of mathematical morphology from which all other morphological operations are based. In this work we limit ourselves to the analysis of erosion, Algorithm 2. In fact, dilation is the dual operation of erosion, which is equivalent to the erosion of the background.

An optimized implementation of the erosion algorithm takes more advantage of the packed-pixels representation and simultaneously computes the erosion of a pixel-pack with simple bit-wise computation of adjacent pixel-packs. In particular, if we do not consider what happens at the border between packs, the erosion by a $k \times k$ kernel can be computed in the following way. Fetch the $r = \lfloor \frac{k}{2} \rfloor$ pack at the top and the bottom with respect the current one. Shift each pack by $1, 2, \dots, r$ left and right, and AND them together. Our implementation also takes into account the border

input : A binary image I of size $M \times N$
output: The eroded image I_e by a $k \times k$ kernel

$I_e \leftarrow \text{copy}(I)$;

```

foreach  $(i, j)$  such that  $I[i, j] = 1$  do
    foreach  $(y, x) \in \{-\frac{k}{2}, \dots, \frac{k}{2}\} \times \{-\frac{k}{2}, \dots, \frac{k}{2}\}$  do
        if  $I[i + y, j + x] = 0$  then
             $I_e[i, j] \leftarrow 0$ ;
            break;
        end
    end
end

```

Algorithm 2. Erosion (the optimized parts are highlighted).

conditions.

Run length

The run-length of an image is the set all the consecutive non overlapping horizontal sequences of ‘1’-pixels. Algorithm 3 shows a simple procedure for extracting the run-lengths.

input : A binary image I of size $M \times N$
output: A set of run lengths Λ

```

 $\Lambda \leftarrow \{\}$ ;
for  $i \leftarrow 0$  to  $M - 1$  do
     $j \leftarrow 0$ ;
    while  $j < N$  do
        if  $I[i, j] = 1$  then
             $start \leftarrow (i, j)$ ;
             $k \leftarrow j + 1$ ;
            while  $I[i, k] = 1 \wedge k < N$  do
                 $k \leftarrow k + 1$ 
            end
             $end \leftarrow (i, k)$ ;
             $j \leftarrow k$ ;
             $\Lambda \leftarrow \Lambda + \{(start, end)\}$ ;
        end
    end
end

```

Algorithm 3. Run-lengths extraction (the optimized parts are highlighted).

By using our optimized representation we can construct run-lengths in the fol-

lowing way. For each row and for each 64-pixels block, we extract run-lengths by recursively splitting blocks in half until a condition is satisfied, and attempt to merge these intermediate results every time a row changes. In more detail, the recursive splitting procedure is used to find with few operations sub-blocks of consecutive ‘1’-s within a block of pixels.

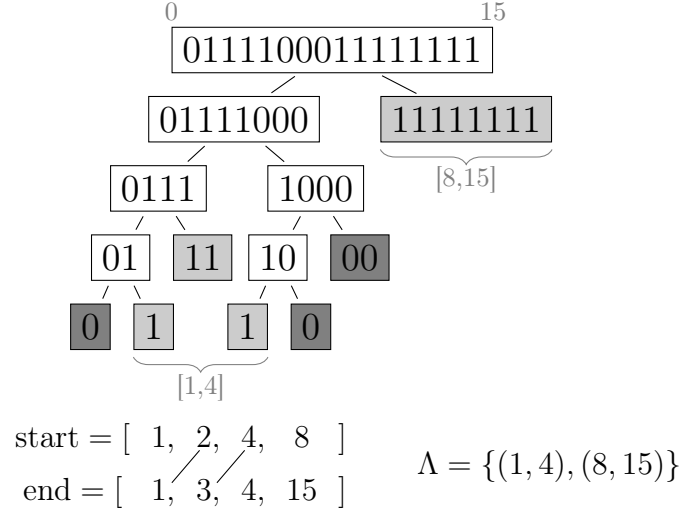


Figure 4.1: Example of the split and merge procedure

The recursion ends in two ways: when the current sub-block is zero, discarding the current interval, and when the current sub-block contains all ‘1’-s, keeping track of the starting and ending index. Figure 4.1 shows an example of the split procedure in the case of a 16 pixels block. Here we highlight: in light-gray the ‘1’-blocks where the recursion terminates keeping track of the starting and ending position, and in dark-gray the ‘0’-blocks where the recursion terminates without effects.

However, this procedure by its own might identify contiguous sub-runs which are not merged in a single one. Therefore, in order to detect the correct run-lengths we adopt the following strategy to merge consecutive sub-blocks. By means of two arrays we keep track of the starting and ending column index of each sub-run. When the scan of a row ends, the merging procedure iterates through the elements of these arrays and merges sub-runs for which the starting position of a new sub-run is equal to the ending position of the previous plus one, as shown in Figure 4.1.

Contour tracing

In this section we examine the optimized implementation of the Moore-neighbour contour tracing algorithm, Algorithm 4. In the following analysis, without loss of generality we limit ourselves to the detection of the contour of a single object.

input : A binary image I of size $M \times N$
output: A set of points Γ that defines the contour

```

 $\Gamma \leftarrow \{\}$ ;
 $start \leftarrow \text{first\_set\_pixel}(I)$ ;
 $end \leftarrow ()$ ;
 $cur \leftarrow start$ ;
 $off \leftarrow \{-1, 0, 1\} \times \{-1, 0, 1\}$ ;
 $dir \leftarrow off[0]$ ;
while  $start \neq end$  do
     $curr \leftarrow cur + dir$ ;
     $k \leftarrow \text{arg}(off = dir)$ ;
    while  $I[curr[0] + dir[0], curr[1] + dir[1]] = 0$  do
         $k \leftarrow k + 1 \bmod 8$ 
    end
     $end \leftarrow cur + off[k]$ ;
     $\Gamma \leftarrow \Gamma + \{end\}$ ;
     $dir = -1 \cdot off[k]$ ;
end

```

Algorithm 4. Moore contour tracing (the optimized parts are highlighted).

The Moore-neighbor starts from a given contour point and iteratively finds the next point by scanning in clock-wise direction the 3×3 neighbours starting from the position of the previous contour point, and it terminates when the starting point is visited again. The starting point is defined as the first ‘1’-pixel that is encountered during the image raster scan. In this case, our implementation extracts contours avoiding the clock-wise scan of neighbors, by processing the packed-neighbors with bit-wise operations. In particular we use a rotation, a left most bit finding and a table-look up. The look-up table consists of all the neighbors offsets with respect to the current (i, j) pixel. The objective of our procedure is to compute the index of this table that gives us the next contour pixel.

Figure 4.2 shows a concrete example of how a new contour point is found according to our efficient procedure.

At the left end side of the figure we represent the current (i, j) pixel neighborhood, where respectively we highlight in gray the previous contour pixel, in red the current

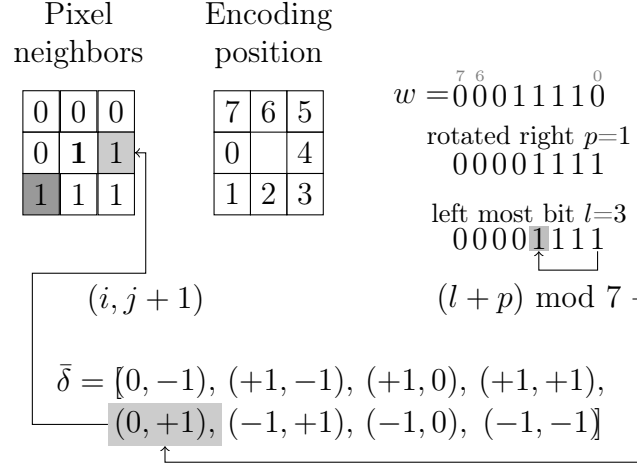


Figure 4.2: Example of the optimized “find the next contour pixel” procedure

contour pixel, and in green the next one. The order according to which the neighbours are encoded is specified in Figure 4.2, which is clock-wise starting from the top-left element.

Thinning

A popular algorithm for binary image thinning is the Guo-Hall algorithm [45], Algorithm 5.

The thinned image is computed by calling thinning iterations until no more deletion is possible. Two criteria are applied in order to delete a pixel, which are identical under a rotation of 180° , one for the even iterations and one for the odd iterations.

Our optimized implementation uses look-up tables in order to compute the deletion condition of a pixel. In particular, for all of the 2^8 possible configuration of the $3 \times 3 - 1$ neighborhood we pre-evaluate and store the deletion condition such that, at run-time, it can be computed by an efficient table look-up. In this case we process the image in the super-packed neighbors domain. The pixel value of the current processed neighborhood (i, j) is recovered from the next neighbor pack $(i, j + 1)$. Instead of introducing an additional branch in the nested loop, to check if the pixel is set, we AND the pixel value with the lookup result. If this condition holds, we delete the current (i, j) pixel. Those considerations are not limited to the Guo-Hall algorithm, in fact they can be extended to all the thinning algorithms that share a common setup with respect to the Guo-Hall algorithm, such as the one proposed by Zhang-Suen [118].

input : A binary image I of size $M \times N$

output: The thinned image I_t

Function GuoHall(I) **is**

```

     $iter \leftarrow 1$ ;
     $deleted \leftarrow 1$ ;
    while  $deleted > 0$  do
        |  $I, deleted \leftarrow \text{ThinIter}(I, iter)$ ;
        |  $iter \leftarrow iter + 1$  ;
    end
    return  $I$ 
end

```

Function ThinIter($I, iter$) **is**

```

     $I_t = \text{copy}(I)$ ;
     $deleted \leftarrow 0$ ;
    foreach  $(i, j)$  such that  $I_t[i, j] = 1$  do
        | if DeleteCond( $I, iter, i, j$ ) then
            | |  $I_t[i, j] \leftarrow 0$ ;
            | |  $deleted \leftarrow deleted + 1$ ;
        | end
    end
    return  $I_t, deleted$ 
end

```

Algorithm 5. Guo-Hall thinning (the optimized parts are highlighted).

4.1.3 Experimental results

In this section, we report on the results of several experiments with different binary image processing algorithms. We quantify the improvements that our efficient implementation offers compared to corresponding baseline standard implementations, and state-of-the-art libraries for image processing. The chosen algorithms are: erosion/dilation¹, run-length extraction, contour extraction, and thinning. These algorithms are fundamental components of binary image processing. Algorithms such as connected components extraction, although widely used, are not considered in this comparison because internally they operate on integer images.

Experiments are conducted on a small dataset of artificially created binary images of size 1024×1280 . Images are manually designed by using the GIMP² open-source software and its paintbrush tool. The images are composed of different shapes, mainly

¹We limit ourself to erosion and dilation because all the other morphological operation can be expressed in terms of these two basic ones [94]

²<https://www.gimp.org/>

```

Function DeleteCond( $I, iter, i, j$ ) is
   $c \leftarrow (\neg I[i-1, j] \vee I[i-1, j+1] \wedge I[i, j+1]) +$ 
     $(\neg I[i, j+1] \vee I[i+1, j+1] \wedge I[i+1, j]) +$ 
     $(\neg I[i+1, j] \vee I[i+1, j-1] \wedge I[i, j-1]) +$ 
     $(\neg I[i, j-1] \vee I[i-1, j-1] \wedge I[i-1, j]);$ 
   $n_1 \leftarrow (I[i-1, j-1] \wedge I[i-1, j]) + (I[i-1, j+1] \wedge I[i, j+1])$ 
     $(I[i+1, j+1] \wedge I[i+1, j]) + (I[i+1, j-1] \wedge I[i, j-1]);$ 
   $n_2 \leftarrow (I[i-1, j] \wedge I[i-1, j+1]) + (I[i, j+1] \wedge I[i+1, j+1]) +$ 
     $(I[i+1, j] \wedge I[i+1, j-1]) + (I[i, j-1] \wedge I[i-1, j-1]);$ 
   $n \leftarrow \min(n_1, n_2);$ 
  if  $iter \bmod 2$  then
     $d \leftarrow (I[i-1, j] \wedge I[i-1, j+1] \wedge \neg I[i+1, j+1]) \vee I[i, j+1];$ 
  else
     $d \leftarrow (I[i+1, j] \wedge I[i+1, j-1] \wedge \neg I[i-1, j-1]) \vee I[i, j-1];$ 
  end
  return  $(c = 1) \vee (2 \leq n \leq 3) \vee \neg d$ 
end

```

Algorithm 6. Guo-Hall delete condition.

arbitrary curves characterized by wide range of thickness, length, and complexity. Some examples are available at our github repository. Images are subdivided in three categories based on foreground pixel density: low, medium, and high. In average the three groups present roughly the following density values: $\rho_{low} \approx 0.15$, $\rho_{med} \approx 0.3$ and $\rho_{high} \approx 0.5$. We think that artificially created images, rather than natural images, are more appropriate for a thorough experimental evaluation that does not depend on the nature of images. Binary natural images are often related to a particular task and do not exhibit the high variability in composition we are able to offer with our dataset.

Image density affects performance during tests. Pixel density is independent from the image resolution, and the obtained performance results can be generalized to different resolutions by means of re-scaling. However, pixel density based experiments are not the most appropriate for contour extraction. Therefore, we test contour extraction implementations on a different dataset of artificial images. In this case, images are subdivided according to their contour length: which is chosen to be: short, average, and long with respect to the image resolution.

For each experiment we report the mean execution time of 1000 subsequent repeated executions, such that caches can be warmed up. In the following experiments all the code is compiled with `gcc` with optimization flag `-O3`, and executed on a Intel® Core™ 2 Duo CPU P7350@2.00 GHz.

Comparison with our baseline

Table 4.1 reports the execution time comparison between standard baseline implementations written by the author, and the SPmat implementations. By keeping the algorithms the same in all aspects other than the use of the optimized binary representations we ensure that this comparison is meaningful. The average performance improvement obtained by SPmat among the tested algorithms is $\approx 50\times$. Performance improvement clearly depends on the algorithm, and it is directly related to “amount of work” required per pixel. In general, the more “work per pixel” is required, the more likely it is to obtain a big improvement. With the term “work” we refer to both computations and memory accesses. In fact, the SPmat data-structure allows a more efficient use of memory as well as the possibility to process multiple pixels at the same time.

Table 4.1: Comparison vs baseline implementation [speed-up / μs]

Operation		Baseline	SPmat
$\text{contour}(l_{\text{short}})$	$1.2\times$	55	45
$\text{contour}(l_{\text{med}})$	$2.42\times$	410	169
$\text{contour}(l_{\text{long}})$	$2.61\times$	1123	431
$\text{runlen}(\rho_{\text{low}})$	$5.41\times$	3150	582
$\text{runlen}(\rho_{\text{med}})$	$5.75\times$	5777	1005
$\text{runlen}(\rho_{\text{high}})$	$4.67\times$	8928	1912
$\text{erode}(\rho, 3)$	$137\times$	23 468	171
$\text{erode}(\rho, 5)$	$130\times$	50 676	391
$\text{erode}(\rho, 7)$	$120\times$	87 420	731
$\text{thin}(\rho_{\text{low}})$	$51\times$	230 811	4443
$\text{thin}(\rho_{\text{med}})$	$73\times$	379 125	5159
$\text{thin}(\rho_{\text{high}})$	$81\times$	517 530	6366

Analyzing Table 4.1 from top to bottom, we see that contour extraction is the least improved algorithm. In this case the best obtained speed-up by SPmat is $\approx 2.5\times$, and it shows an increasing trend based on the length of the contour. The Moore algorithm itself only requires to scan (at most) the 3×3 neighborhood per pixel to find the next contour pixel. This is a low computational operation and can not be parallelized on a pixel-pack, because it is not guaranteed that the contour spans the entire pixel pack. In fact, given a contour point, in the worst case scenario there is a probability of $\frac{2}{3}$ that the next contour pixel is found on a different row, which as a consequence can

possibly generate more cache misses. Therefore, the only way to improve the baseline implementation is the ability to fetch and process the neighbourhood faster as is done by SPmat using packed-neighbours.

In the case of run-length the improvement offered by SPmat is on average $\approx 5\times$. Since the algorithm is row based, on average we have less cache misses per row, and more importantly we can process 64 pixels in a CPU register simultaneously. Based on the distribution of run-lengths, it is frequently the case that the number of computations per pixel-pack is considerably smaller than 64 (which is the baseline case). Ideally, when all the pixels in the pack are either ‘0’ or ‘1’, just one operation is needed to process it. The SPmat implementation performance does not (directly) depend on the pixel density, but it is somewhat correlated to it. The performance depends on the “regularity” of the image, that causes run-lengths to be constant, which is more likely to be the case in low-density images.

In the case of erosion, the SPmat implementation largely outperforms, $\approx 130\times$, the baseline implementation. This is possible because the erosion of 64 pixels is computed at the same time through few bit-wise AND operations, while having better cache efficiency. In the case of erosion the performance does not depend on pixel density. In fact, the same processing operations are performed for each pixel, independently of its value. Usually in high-performance code, branching inside large nested loops is avoided in order to not have a pipeline stall. However, the cost of a pipeline stall, mitigated in modern CPU architectures thanks to sophisticated branch prediction logic, is less than the cost of fetching the pixel neighbourhood from memory. In the case of low density images, where many ‘0’ pixels are skipped during erosion, performance is likely to increase. Intuitively, the best performance is achieved with small structuring elements and gradually decreases as the structuring element size increases, because the bigger neighbourhood requires more memory loads.

Also for thinning, the SPmat implementation achieves an important average speed-up of $\approx 70\times$. In this case, the SPmat implementation enables a very efficient computation of the pixel deletion condition, which is done with a single table lookup. Moreover, the use of super-packed neighbours reduces the 8 memory fetches to just one, and at the same time fits into register the other neighbours pack that belong to the super-pack. In this case, performance increases as the pixel density increases. Structured images with high pixel density usually require more deletions to compute the final result. Thus, the improvement in term of execution time directly depends on the number of deletions.

Comparison with available libraries

In this section, we compare SPmat performance against current version of the state-of-the-art libraries (publicly available) for image processing, namely: MATLABTM Image Processing Toolbox ³, OpenCV ⁴, and Leptonica ⁵. It is important to note that this evaluation is affected by the libraries design, and algorithm implementation. Therefore, perfect correspondence of the algorithm, as is the case with our baseline comparison, can not be guaranteed for these experiments. This is particularly the case with the thinning and contour following algorithms. Although thinning algorithms among the different implementations are not the same, they share a common approach that makes the computations comparable. In the case of the contour following algorithm, each implementation uses the same building block (Moore contour following) but other libraries provide implementations that take into account more complicated scenarios than the one we have implemented. For this reason, the reported speed-up does not give a fair indication of the performance improvement. Finally, not all the tested algorithms are implemented in the libraries used for the comparison.

Table 4.2: Comparison with state-of-the-art libraries (CPU) [speed-up / μ s]

Operation	MATLAB TM		OpenCV		Leptonica		SPmat	32-bit
$erode(\rho, 3)$	18×	3126	9×	1597	3×	511	171	262
$erode(\rho, 5)$	22×	8766	4×	1889	2×	835	391	556
$erode(\rho, 7)$	15×	11 160	3×	2456	1.5×	1146	731	1037
$thin(\rho_{low})$	39×	174 092	N/A		47×	211 455	4443	
$thin(\rho_{med})$	45×	233 914	N/A		68×	354 528	5159	
$thin(\rho_{high})$	42×	270 297	N/A		62×	397 328	6366	
$contour(l_{short})$	140×	6340	24×	1096	N/A		45	
$contour(l_{med})$	184×	31 089	29×	4893	N/A		169	
$contour(l_{long})$	79×	33 992	12×	5109	N/A		431	

Table 4.2 reports the comparison with CPU implementations from different libraries. Averaging all performance improvements we notice that SPmat implementation is approximatively: 65×, 13× and 30× faster than, respectively: MATLABTM, OpenCV, and Leptonica. Averaging is limited to the available implementations. This result highlights the bit-packing superiority as a way to improve performance com-

³<https://www.mathworks.com/products/image.html>

⁴<http://opencv.org/>

⁵<http://www.leptonica.com/>

pared to code optimization. In fact, the only alternative whose performance is close to SPmat is Leptonica erosion which features bit-packing. In addition, for the erosion experiment, we also report on the execution time obtained by SPmat using 32-bit pixel-packs (same size as Leptonica) instead of 64-bit. This comparison shows that our performance improvement with respect to Leptonica does not only depend on the bigger packing size. In fact, SPmat 32-bit still outperforms, even though with a smaller margin, the Leptonica implementation.

The MATLABTM image processing toolkit result is always the slowest alternative (with the exception of Leptonica’s thinning). MATLABTM is often used for prototyping and fast development, therefore, performance is not likely to be the primary concern. However, is not easy to justify why the Leptonica thinning implementation performs worse than MATLABTM. It may be due to the thinning algorithm, which requires more iterations or perform more computation per pixel, to produce the final result. MATLABTM implements the Zhang-Suen algorithm [118], while Leptonica implements an algorithm [13] developed by Dan Bloomberg, the author of the library. The performance achieved by OpenCV shows the benefit of SIMD optimization for erosion, which is consistent if compared to MATLABTM, but not as important as the binary optimization used in Leptonica and SPmat.

As already mentioned, the contour extraction comparison does not give a fair indication of the performance improvement obtained by SPmat. With SPmat we provide an optimized implementation of the Moore border following algorithm, while MATLABTM and OpenCV offer a more complete algorithm, built on top of that, which extracts all the contours and organizes them into an hierarchy. It is not obvious how to quantify the cost of the additional complexity introduced by the MATLABTM and OpenCV implementations. The large improvement of performance obtained by SPmat indicates that our implementation would likely still be faster even if the additional complexity was added.

4.2 Espresso

Convolutional Neural Networks have revolutionized computer vision, pushing the task of object recognition beyond human capabilities [62, 99, 107]. Deep Neural Networks (DNN), have also been successfully applied in other fields, such as speech recognition [42, 51] and automated translation [4, 106]. Despite achieving impressive classification accuracy results, DNNs require too much memory and power to

be used effectively on embedded or low-power devices. Many networks consume a considerable amount of memory. Memory remains a very limited resource on mobile platforms making harder the usage of trained DNNs ⁶. Even when memory is not an issue, DNNs remain very computationally intensive, and can quickly drain the battery. Reducing the computational load does not only improve energy efficiency, but can also enable further applications. For example, when processing real-time object classification on mobile devices, being able to perform faster predictions frees up computational resources that can be spent on other tasks such as speech recognition and analysis. Therefore, there is a substantial interest in reducing the computational and memory requirements of DNNs.

One way to achieve this target is to use specialized hardware for DNNs. Another strategy is to reduce the network’s memory footprint and associated computation, hence increasing its efficiency. Such solutions are preferable as they can be implemented in software without requiring specialized hardware. In our research we follow the software approach, and focus our attention to *quantized* networks. In this case, the parameters are stored as “small” integers (typically less than 8-bit) instead of single precision floating point numbers (32-bit). In particular, we consider the *binary* deep neural networks (BDNN) proposed by [53] where parameters and activations are 1-bit integers: $\{-1, +1\}$. At the expense of a relatively small decrease in accuracy, BDNNs can considerably reduce memory usage, and result in faster execution time (i.e. forward propagation). Furthermore, note that potential hardware implementation of BDNNs would also be cheaper due to the reduced number of required FPUs. While these results are highly promising, currently only *proof-of-concept* implementations of BinaryNets have been published [53]. Therefore, the availability of a flexible end-to-end framework, with particular emphasis placed on computational efficiency, can enable further research on BDNNs, as well as its application to practical scenarios.

With *Espresso* we provide an optimized framework for BDNNs capable of achieving state-of-the-art run-time *performance* with minimal *memory* footprint while being numerically equivalent to their non-optimized binary counterpart. *Espresso* provides a complete optimized framework for BDNNs supporting both the *dense* and the *convolutional* layer. Current state-of-the-art optimized BDNNs implementations are limited to the fully connected layer, with the serious drawback of not being able to run optimized state-of-art convolutional BDNNs (BCNNs). While our work is a necessary

⁶for example, the popular AlexNet [62] and VGG [99] architectures consume respectively ≈ 250 MB and ≈ 520 MB

stepping stone towards optimization of training routines, in this thesis we focus on the optimization of forward-propagation (i.e. testing), rather than back-propagation (i.e. training). *Espresso* is designed to have no external dependencies. This not only results in a highly optimized implementation of BDNNs, but also substantially simplifies its deployment in practical applications, such as those executing on mobile or embedded devices.

4.2.1 The Espresso Framework

Espresso provides the user with the necessary tools for executing forward-propagation of DNNs, with particular emphasis placed on convolutional neural networks due to their ubiquitousness in computer vision applications. As the complexity of these networks is cubic to the size of the problem, they are less memory efficient and more computationally intensive than traditional machine-learning algorithms. Identifying the memory and computational bottlenecks of DNNs is therefore essential to enable their practical application. In particular, our primary focus is *GPU-optimized* BDNN architectures, which we refer to as GPU^{opt} , but we also support the equivalent floating-point counterparts on heterogeneous architectures, which we refer to as *CPU* and *GPU*. The *CPU* and *GPU* implementations of *Espresso* do not feature binary optimizations because the data is encoded as single precision floating point numbers. However they still utilize an optimized library for matrix multiplication.

The *Espresso*'s implementations of tensors and layers come in three variants $\{CPU, GPU, GPU^{opt}\}$. A CPU-tensor is allocated in CPU memory, and is processed on the CPU using sequential code. A GPU-tensor is allocated on GPU main memory and is processed by CUDA kernels. *Espresso* provides functions for *converting* tensors and layers from one variant to the other, and different variants can also be interconnected with each other. Consequently, *Espresso* enables the design of hybrid DNNs consisting of a combination of $\{CPU, GPU, GPU^{opt}\}$ layers.

Dense linear algebra is at the heart of deep-learning as deep networks can be viewed as a composition of *matrix-matrix*, *matrix-vector* and *elementwise matrix-matrix or vector-vector* multiplications. The implementation of these dense linear algebra operations relies heavily on the efficient computation of the *dot-product*. The execution of this operator consists of (single precision) *Floating-point Multiply and Add* (FMA) operations. In modern architectures, floating-point multiplications executing on the FPU dominate the complexity of FMAs, and BDNNs address these

concerns by replacing FMAs with simpler *bitwise* operations; see Section 4.2.2.

The superior computational performance of *Espresso* derives from three main technical contributions: (1) the use of bit-packing in network layers, (2) better memory layout and management, and (3) the use of custom optimized CUDA kernels. Through the use of bit-packed layers, *Espresso* can execute a forward operation without the need for expensive memory re-arrangements employed by existing implementations. As dynamic memory allocation on GPUs is a performance bottleneck, *Espresso* implements a custom memory allocator that pre-allocates memory at start-up, and replaces the traditional *malloc* and *free* system calls. Finally, matrix multiplications are performed with CUDA kernels that have been adapted to bit-packing, and only resort to XNORs and bit-counts.

4.2.2 Binary Deep Neural Networks (BDNNs)

In this section, we overview the fundamental characteristics of BDNNs [53] that inform the basics of *Espresso*’s design. In BDNNs, computationally intensive FMA operations are replaced by *XNOR* (for multiplications) and *bit-count* (for additions), enabling significant computational speed-ups. In particular, XNOR is a simpler machine instruction compared to floating point multiplication, and therefore achieves much higher throughput on many architectures. More importantly, a single XNOR step can execute multiple 64-bit wide blocks of dot-products, further increasing the overall computational efficiency. In what follows, we describe how a network is binarized, detail a compressed memory layout enabling efficient execution of dot-products, show how to re-interpret input data to allow execution on fixed-precision input (e.g. images), and provide a few notes regarding the training procedure.

Network binarization

A BDNN is composed of a sequence of $k = 1, \dots, L$ layers whose weights W_k^b and activations a_k^b are binarized to the values $\{-1, +1\}$. The superscript b in the notation indicates binary quantities. Weights and activations are $\{-1, +1\}$, but at the hardware level they must be encoded as $\{0, 1\}$. Our convention is to encode $-1 \rightarrow 0$ and $+1 \rightarrow 1$. Amongst many possible choices, e.g. stochastic binarization [29], we

employ the following activation function due to its efficient implementation:

$$x^b = \text{sign}(x) = \begin{cases} +1 & x \geq 0 \\ -1 & \text{otherwise} \end{cases} \quad (4.1)$$

Bit-packing

The weights of a BDNN can be stored in the bits of a 64-bit word. One immediate advantage of bit-packing is to drastically reduce the memory usage by a $32\times$ factor. An even more significant advantage is the ability to process multiple values at the same time using registers. This is particularly useful for dot-products: with bit-packing we can compute a dot-product of 64 element vectors by using just one XNOR and one bit-count. Furthermore, modern computer architectures provide a hardware instruction for counting the number of bits set to 1 in a given word. Assuming binary vectors $a, b \in \mathbb{B}^{1 \times N}$ where N is a multiple of 64, the dot-product is then equivalent to:

$$a \cdot b \equiv N - \left(\sum_{i=1}^{N/64} \text{bitcount}(\text{XNOR}(a_i, b_i)) \right) \ll 1 \triangleq a \odot b \quad (4.2)$$

where $\ll$ represents the bit-shift operator. This simple computation becomes the building block of optimized BDNNs as binary matrix-matrix or matrix-vector operations are computed in this fashion.

Input data binarization

BDNNs require binary input data, which is not typically available at the first layer of the network. However, the input data usually comes in a fixed precision format (e.g. 8-bit/channel in RGB images). Therefore, the optimized computation of dot-products can still be applied if we split the input data according to bit-planes, and then sum back each contribution according to the corresponding weight. For instance, if with $\langle a \rangle_n$ we indicate the n -th bit of a fixed precision vector, and with i the corresponding bit-plane, we obtain:

$$a \cdot b \equiv \sum_{i=0}^{n-1} 2^i \langle a \odot b \rangle_i \quad (4.3)$$

Training

When training a BDNN (Algorithm 7) it is important to note that the gradient is computed with the binary weights, but is accumulated with floating point precision [53]. That is because the optimizer needs sufficient precision to make a reliable update. In addition, the derivative of the sign function, which is zero almost everywhere, cannot be used for back-propagation. To overcome these issues, the *straight-through estimator* [10] is employed, where 1 is back-propagated if the floating point argument $|x| \leq 1$, and 0 otherwise. Finally, during training weights are clipped to $[-1, 1]$ to avoid a large growth of the floating point weights that would not have an impact on the binary weights.

Data: minibatch of input and labels (a_0, y) , previous weights $W^{(t-1)}$, and learning rate η . Subscripts denote the layers, superscripts the time iteration.

Result: updated weights $W^{(t)}$

```

/* Forward propagation                                     */
for k = 1 to L do
     $W_k^b \leftarrow \text{sign}(W_k)$ ;
     $a_k \leftarrow a_{k-1}^b W_k^b$ ;
    if k < L then
         $a_k^b \leftarrow \text{sign}(a_k)$ ;
    end
end

/* Backward propagation                                     */
for k = L to 1 do
    if k < L then
         $\frac{\partial C}{\partial a_k} \leftarrow \frac{\partial C}{\partial a_k^b} \circ 1_{|a_k| \leq 1}$  ;           // element-wise product
    end
     $\frac{\partial C}{\partial a_{k-1}^b} \leftarrow \frac{\partial C}{\partial a_k} W_k^b$ ;
     $\frac{\partial C}{\partial W_k^b} \leftarrow \frac{\partial C}{\partial a_k}^T a_{k-1}^b$ ;
end

/* Accumulate minibatch gradient                           */
/* Parameters update                                        */
for k = 1 to L do
     $W_k^{(t)} \leftarrow \text{clip}\left(\text{opt}\left(W^{(t-1)}, \eta, \frac{\partial C}{\partial W_k^b}\right), -1, 1\right)$ ;
end

```

Algorithm 7. Training of a BDNN with L layers according to cost function C .

4.2.3 Espresso architecture

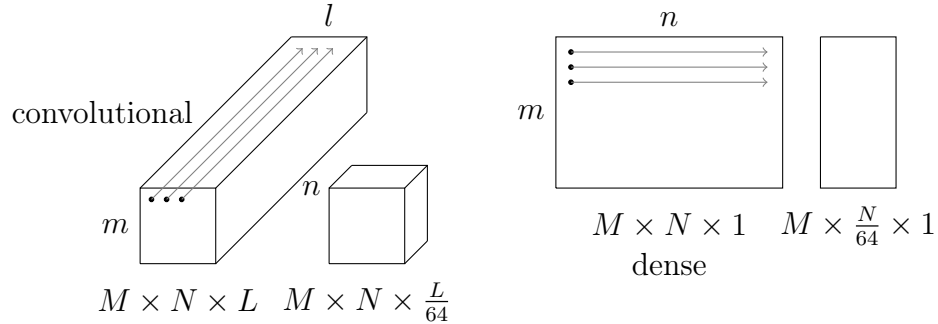
The principal components of our framework are *tensors*, *layers* and the *network*. These components are organized as a hierarchy. Tensors are n dimensional matrices used for storing *inputs*, *weights* and *activations* (outputs). A layer processes an input tensor and produces an output tensor, while a network consists of a concatenation of layers.

Tensors

In *Espresso*, each element of a tensor $A \in \mathbb{R}^{M \times N \times L}$ is identified by the triplet m, n, l , where $m \in [0, M)$ indicates the row, $n \in [0, N)$ indicates the column, and $l \in [0, L)$ indicates the channel. A tensor is stored in memory using row-major order with interleaved channels. Therefore, according to this layout, the element $A_{m,n,l}$ is found at position $(mN + n)L + l$ in linear memory.



We use the notation $A_{m,n,:}$ to indicate all the channels of the (m,n) -th element. Using the same storing scheme *Espresso* also defines bit-packed tensors for GPU^{opt} implementations but with the following changes to further increase its performance. Bit-packing is performed according to the number of channels: when $L > 1$ bit-packing is done along the l dimension; when $L = 1$ bit-packing is done along the n dimension. For *convolutional* layers this packing direction enables efficient memory



access when unrolling/lifting a tensor, which would have not been possible if either m or n had been chosen instead. More specifically, this layout is optimal for retrieving

a pixel neighborhood as needed by convolution without requiring the layout to be changed. Further, typically a large number of filters are used resulting in an increase of tensor dimension in the l direction, while the m and n dimensions are progressively shrunk by pooling layers. For other layer types, n is the most efficient packing direction, as neurons are stored along rows and their number decreases as we move toward later stages in the network.

Layers

Espresso provides the following layer types: *Input*, *Convolutional*, *Pooling*, *Dense* (i.e. fully connected) and *Batch-normalization*. Each layer is characterized by its size, tensor parameters and output. The *Espresso* API defines for each layer a *forward* function that computes the output of a layer given an input tensor, and a function for applying *non-linearity* to the outputs of convolutional and dense layers. Moreover, the convolutional layer features additional functions for *pooling* and *unrolling*.

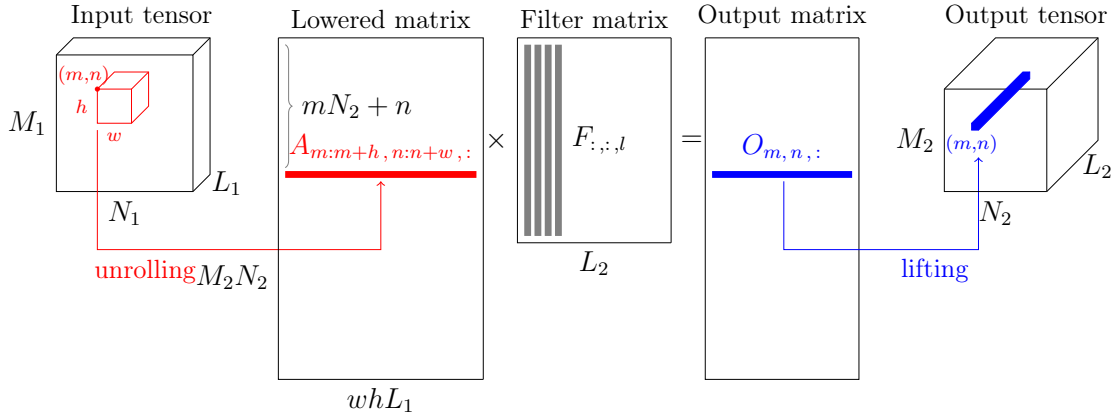


Figure 4.3: *unrolling* and *lifting* operations for CNN layers

In our framework, 2D convolutions are computed through matrix multiplications – an operation involving a very high reuse of data. For both *CPU* and *GPU*, this computation is performed by sectioning data in amounts that are cache-friendly [36], resulting in implementations attaining close to peak computational performance. However, in order to express convolution as matrix multiplication we need to re-organize the input memory appropriately. This is achieved through the *unrolling* procedure; see Figure 4.3. It consists of transforming a tensor into a matrix where each row is formed by unrolling the tensor data contained in each convolution sliding volume. The unrolled matrix is then multiplied by the filter matrix. Finally, the result of the

convolution is reordered back to tensor by using the *lifting* procedure. In *Espresso* we do not need to manually lift the convolution result in order to undo the unrolling: thanks to our tensor representation this happens automatically and at zero cost. *Espresso* provides CUDA kernels for the *unrolling* and *pooling* of tensors for both *GPU* and *GPU^{opt}* implementations.

Matrix-vector multiplications are fundamental operations of both dense and CNN layers. For the *CPU* architecture, we use the OpenBLAS library [116] to implement these operations. For *GPU* and *GPU^{opt}* architectures, the CUDA kernels are based on MAGMA(sgemv) [2], modified to make it compatible with our binary data representation. These kernels for matrix multiplication feature *register blocking* optimization: since the introduction of *Fermi* architectures the number of registers have been increased, while register access latency has been substantially reduced compared to shared-memory; hence caching at the register-memory level results in considerably faster throughput [78]. *Espresso* first fetches the tiles of the two matrices into shared-memory and then process sub-tiles using registers. In the *GPU^{opt}* variant, we modify the code by replacing blocks of 64 (or blocks of 32 for *GPU^{opt}* 32) single precision multiply and add (FMA) operations with XNOR and bit-count using packed tensors. We also re-tune the kernel block size parameters for improving the performance on reduced size matrices.

Typical CNN implementations apply a tensor convolution in a “same” configuration, where the sizes of input and output tensors matches. This is achieved by zero-padding input tensors, but in convolutional *GPU^{opt}* layers the *zero-padding* of the input introduces the side-effect of making the data ternary $\{-1, 0, +1\}$. We deal with this problem by treating the data as if it was binary (zero is considered a minus one) and fix the results of the convolution at these corner-cases in post-processing. This allows us to leave the convolution kernel code – the computational bottleneck of the code – untouched. The corner-cases are fixed using a highly efficient kernel which executes an element-wise sum between the results of the convolution and the correction matrix. The correction matrix is computed once, when the *GPU^{opt}* layer is loaded, and it simply consists of the convolution of the layer’s weights with a (+1)-padded zero-tensor.

A DNN in *Espresso* is defined as a combination of layers, which is loaded at run-time by reading its parameters file. The parameters file specifies the storage format of all the layers, as well as their weights. Therefore, it completely specifies a DNN as layers are stored sequentially. Training of the network is done by BinaryNet [53];

the resulting parameters are converted to the *Espresso* format by a utility script distributed together with the provided source code.

Evaluation

The performance of our framework is evaluated in terms of average computational time needed to perform a particular task. The execution times, averaged over 100 experiments, are obtained on a machine equipped with an NVIDIA GeForce GTX 960 with 2GB of RAM, and a Intel® dual-Xeon® X5660 @ 2.80 GHz. In *CPU* mode, we configure the OpenBLAS library for matrix multiplication to use all the 24 available cores.

We perform three quantitative evaluations: (Section 4.2.3) matrix multiplications of two dense square matrices of size 8192×8192 ; (Section 4.2.3) forward-propagations of a Multi-Layer Perceptron (MLP) trained on the MNIST dataset [63]; (Section 4.2.3) forward-propagations of a Convolutional Neural Network (CNN) trained on the CIFAR-10 dataset [61]. By using the same models and datasets, we compare *Espresso* with: (1) the author provided optimized implementation of BinaryNet [29]; (2) the optimized BDNN implemented in the Intel Nervana *neon* framework [79]; (3) a self-comparison across $\{CPU, GPU, GPU^{opt}\}$ as no binary-optimized implementations of convolutional layers are publicly available. *Espresso* is numerically equivalent to BinaryNet in terms of classification accuracy. Therefore our evaluation focuses on computation speed.

The MNIST dataset [63] consists of 60K instances for training and, 10K instances for testing. Each instance is a 28×28 grayscale image that depicts digits ranging from 0 to 9. The CIFAR-10 dataset [61], consists of 50K training instances and 10K testing instances of $32 \times 32 \times 3$ color images. Images are subdivided into 10 classes (airplanes, automobiles, birds, cats, deers, dogs, frogs, horses, ships and trucks). Since our interest is to assess the real-time performance of binary optimized DNNs, in those experiment we use a batch-size of one, and measure the averaged forward time for each image of the testing-sets for each dataset.

Binary dense matrix multiplication

In computing dense matrix multiplication, Table 4.3, *Espresso* outperforms BinaryNet by a $\approx 8\times$ factor.

Much of the gain can be attributed to our optimized kernels, and the use of

Table 4.3: Averaged time of binary optimized matrix multiplication.

BinaryNet	Espresso _{GPU^{opt}} (32-bit)	Espresso _{GPU^{opt}} (64-bit)
88 ms	16 ms (5.5×)	11 ms (8×)

register blocking: by fetching bigger data from main memory and shared memory, our kernel increases the bandwidth utilization by decreasing the number of memory fetch instructions. The use of 64-bit packing instead of the 32-bit (such as that of BinaryNet), introduces an additional performance improvement. The 64-bit kernel achieves a memory DRAM throughput of 40 GB s^{-1} for reads and 5 GB s^{-1} for writes, while the 32-bit kernel obtain 29.6 GB s^{-1} for reads and 3.6 GB s^{-1} for writes. This translates into the resulting $\approx 25\%$ speed improvement.

Multi-layer perceptron on MNIST

Table 4.4: Average prediction time of the BMLP.

BinaryNet	Nervana/Neon	Espresso _{CPU}	Espresso _{GPU}	Espresso _{GPU^{opt}}
18 ms	17 ms	37.4 ms	3.2 ms (5.6×)	0.26 ms (68×)

We evaluate the average classification execution time over the MNIST dataset, where we trained the MLP architecture from [30, Sec 2.1] with author-provided sources, and then converted it to *Espresso*’s format. In Table 4.4, *Espresso* achieves a consistent speed-up of $\approx 68\times$ when compared to BinaryNet. As the Nervana/neon implementation of binary network is a BinaryNet derivative, it is affected by the same drawbacks of BinaryNet, and hence achieves comparable performance. Both alternatives have the additional cost of running CUDA through Python/Theano which may introduce further latency in the process. In Table 4.4, the evaluation over the three variants of *Espresso* shows the expected outcome, with the *GPU^{opt}* implementation leading the ranking. Note that we are able to achieve a speedup of $\approx 12\times$ on an NVIDIA GTX 960 ($\approx 2.5 \text{ TFLOPs}$), although this device has only roughly four times more throughput than the Xeon X5660 ($\approx 500 \text{ GFLOPs}$ without turbo-boost). Through binary optimization, we are able to further increase the performance to $\approx 15\times$ with respect to the GPU implementation. We attribute our computational gains to (1) the use of *binary-optimized* layers, (2) our use of *optimized kernels* for

matrix multiplication and (3) *Espresso*'s ability to perform binary optimization of the first layer.

An evident drawback of Binary-Net is the need for binarizing/packing the layer's parameters *every time* a forward method is called. In the case of binary optimized networks, the cost of packing the parameters is closely related to the cost of multiplication itself. Therefore, the reduction of bit-packing function calls leads to a consistent improvement. This motivates our choice of designing specific layers, where bit-packing is done once during network loading.

BinaryNet employs two bit-packing kernels: one for row-packing, the other for column-packing. Although BinaryNet's pack-by-rows kernel is slightly slower than ours (8%), the pack-by-columns kernel is significantly slower ($\approx 4\times$) due to non-coalesced accesses to global memory. An additional performance gain of $\approx 15\%$ is achieved by swapping matrix-vector in favour of matrix-matrix multiplication kernels when appropriate (i.e. Dense layers with batch size equal to 1); for this reason, *Espresso* also includes the binary-optimized MAGMA(sgemv) kernel.

Another important advantage offered by *Espresso* is the ability to leverage binary optimization in the *first* layer. Since the first stage of a network processes non-binary data, BinaryNet does not feature binary optimization for this layer. However if the input data is split into its constituent bit-planes, binary optimization can still be applied. In particular, we split the input vector in a matrix of 8 rows, and recombine the result after multiplication by a weighted sum. Our experimental results report an overall $\approx 3\times$ performance boost when comparing the full binary optimized network with one in which the first layer is not binary optimized.

Finally, in terms of memory the GPU^{opt} implementation requires 4.57 MB instead of 140.6 MB as in the case of non binary optimized implementation, resulting in a saving $\approx 31\times$ amount of memory.

Convolutional Neural Network on CIFAR-10

Table 4.5: Average prediction time of the BCNN.

Espresso _{CPU}	Espresso _{GPU}	Espresso _{GPU^{opt}}
85.2 ms	5.2 ms (16 $\times$)	1.0 ms (85 $\times$)

To the best of our knowledge, no BDNN implementation of *binary-optimized* CNN layers is publicly available. Our self-evaluation implements the *VGGNet*-like CNN

architecture from Hubara et al. [53, Sec. 2.3], and evaluates it across our three modalities: as expected the GPU^{opt} implementation achieves significantly better performance.

Note how the GPU implementation offers a slightly better improvement over CPU with respect to the MLP test, with an $\approx 16\times$ speed-up. In this experiment, the inherent parallelism of unrolling and pooling, and the GPU higher memory throughput explain the behavior. Gains are marginal as FMA still represents the computational bottleneck.

The GPU^{opt} implementation results in a $\approx 5\times$ performance gain with the respect to GPU . These gains, to binary optimizations, are slightly smaller than those discussed for MLP in Section 4.2.3. The output of convolutional layers is significantly larger than those of MLP’s dense layers, therefore, the computation of bit-packing sign-activation requires more computational effort.

Finally, in terms of memory the GPU^{opt} implementation requires 1.73 MB instead of 53.54 MB as in the case of non binary optimized implementation, resulting in a saving $\approx 31\times$ amount of memory.

4.3 Conclusions

This chapters described two optimized frameworks for binary data: SPmat and Espresso. SPmat is related to binary image processing, while Espresso deals with binary neural networks. In both of these frameworks, *bit-packing* and *bit-wise* computations form the foundation for developing efficient data-structures, and associated computational routines. In the case of binary image processing, the super-packed data-structure enables optimized implementation of several algorithms. In the case of binary neural networks a complete computational framework for forward propagation, specific for binary data, was proposed. We showed that considerable improvements in terms of computational time and memory usage are achieved by the proposed frameworks compared to available software solutions.

Chapter 5

Neural network for music transcription and classification

Deep neural networks (DNNs) have achieved state-of-the-art results in image classification [62, 99, 50], speech recognition [42], and language translation [4, 106]. Their performance in these tasks is significantly better from what was possible prior to their development. The driving forces behind this advancement have been the availability of large scale annotated data-sets, powerful GPU accelerators, and easy to use libraries [1, 83, 26]. Another important factor has been the creation of several large-scale benchmarks [34, 5, 60]. This deep learning revolution has led to the use of such networks in a variety of application domains such as content recommendation systems, automatic tagging systems, language translation, and self driving cars. In these applications, traditional machine learning approaches are not used as much anymore, because they are unable to scale up to the large amounts of data that are available nowadays.

Music Information Retrieval (MIR) is an interdisciplinary research area that deals with all aspects of computationally extracting information from music signals. In MIR mainly due to the scarcity of well annotated large scale data-sets, the full potential of deep networks has not yet been realized. Recent work on music generation [113], where models can be trained with unlabeled data, has unveiled the effectiveness of DNNs for this specific task. Therefore, in order to encourage deep learning research in music, large scale annotated data-sets have recently been collected [109], or packaged in a convenient format [33].

Deep Learning (DL) in computer vision is usually done “end-to-end”, i.e. the raw

signal is directly fed into the network such that all the information can be retained by the model. In MIR, or in audio signal processing in general, “end-to-end” learning is, in many cases, not the most performing setup. In fact, domain knowledge when dealing with music signal is still relevant: time-frequency representations in general allow for superior classification performance [108, 35]. While very deep NNs (up to hundred layers) have been proven to be effective for image classification and segmentation, there is still a need to investigate how beneficial they can be in MIR tasks.

DNNs in MIR often tend to be relatively shallow: a couple of convolutional layers with max polling, followed by a couple of linear classifiers is a typical architecture.

In this chapter we investigate the used of DNNs for two MIR tasks: polyphonic music transcription, and music genre classification. In Section 5.1 we focus on the robustness of deep neural networks to input audio degradation. In Section 5.2.1 we describe a convolutional architecture for improving polyphonic music transcription performance of baseline transcription networks.

5.1 Deep neural networks on degraded audio

Robustness to audio degradation is an aspect that is not frequently considered in MIR research. However, consumer applications that interact with the real world are always affected by degradation, because production and/or acquisition of sound is frequently done in an uncontrolled acoustic environment. In addition, smart phones use microphones and audio systems of varying quality. Therefore, machine learning models that are more resistant to degradation of the input audio signal are desired in many application scenarios.

In our work we experiment with a set of popular Convolutional Neural Networks (CNNs) architectures, ranging from shallow up very deep, and evaluate their classification performance in different scenarios of training and testing with clean and degraded audio. More specifically, we investigate the effect of degradation on classification performance for the fundamental MIR tasks of: polyphonic music transcription, and music genre classification. The considered CNNs are informed by neural network architectures proposed for image classification. The spectrogram is treated as an image and the CNNs are used to learn features from it. In particular we use: a network proposed by Thickstun et. al [108], a VGG [99] network with eleven layers, and Residual Networks [50] (ResNet) with eighteen and thirty-four layers.

In this chapter, two types of degradation are investigated: playback through a

low quality phone speaker, and reverberation. In our opinion, these degradation are representative of real world scenarios and support a useful comparison of different training/testing strategies and DNN architectures. The degradation is computed by convolving with corresponding impulse responses. More specifically, we use the: Google Nexus speaker, and Great Hall degradation, available in the Audio Degradation Toolbox [74]. For convenience, we refer to these degradations respectively as “phone” and “hall”.

The most common scenario that has been used in the majority of research in music transcription and genre classification is the use of clean audio for both training and testing. This scenario forms the baseline of our experimental investigation. There are many cases where the exact nature of the audio degradation is not known and we are interested in the robustness of a trained model when applied to data that is different in terms of quality to the data it was trained on. We investigate the following combinations: “clean/clean”, “clean/phone”, “phone/clean”, “clean/hall”, “hall/clean”.

A common strategy to increase robustness to degradations is to apply a type data augmentation in which the possible degradations are known and the training data is augmented by applying them to clean data and increasing that way the size and diversity of the training set. In this scenario we train on the a combined train set of “clean/phone/hall” and test respectively each configuration.

5.1.1 Methodology

We experiment with four CNN architectures applied to the MIR tasks of: polyphonic music transcription (MT) and, genre classification (GC). The CNNs ranges from shallow, up to very deep, and are described in more detail in Section 5.1.2. The considered MIR tasks are formulated as multi-label classification problems. That means that for each given audio frame, or audio clip, multiple classes can be active at the same time, e.g multiple notes, multiple genres.

Input representations

Due to their convolutional nature, CNNs are most effective at extracting features which are spatial invariant. With traditional DFT magnitude spectrograms spatial invariance is not satisfied. In fact, pitches are organized in a logarithm scale which means that: the same frequency “pattern” appears compressed or dilated at different frequencies when a linear scale is used. Logarithmically spaced spectrograms solve

this problem, and allow to achieve space invariance enabling superior performance [108].

In particular, we use as input representations the following log-spaced magnitude spectrograms: Constant-Q Transform (CQT) [19] spectrograms, and mel-scaled spectrograms (which we refer as MEL). CQT spectrograms are computed on 7 octaves with 24 bins per octave with a minimum frequency 32.7 Hz. This yields a total of 168 frequency bins. On the other hand, MEL spectrograms are computed using 2048 DFT points and 128 mel filters. The Hamming window is applied to each audio frame while computing both kinds of spectrograms. Finally, spectrograms are normalized along the frequency axis to a unitary norm, and padded with a reflective pattern to be divisible by two at every pooling stage.

In all of our experiments audio (and labels in case of music transcription) are re-sampled to 11.025 kHz using an implementation of the band-limited *sinc* interpolation method for sampling rate conversion as described by Smith [102].

The spectrograms hop-length and context frames, are dependent on the particular MIR task, and chosen to be equivalent to the specifications used in the experimental section of the papers that introduced the data-sets we utilize in our experiments [109, 33].

Music Transcription: as the authors [108] originally used 44.1 kHz sampling rate, we rescale their specifications by a factor of 4, which results in an hop-length of 128 samples, (≈ 12 ms at 11.025 kHz) and a context window of 32 frames. After padding, we end up with the following input sizes: 192×32 for CQT, and 128×32 for MEL.

Genre Classification: in this case all the 30 s of audio are used to classify a music clip. For this particular task a fine granularity in time is less crucial, therefore, we adopt a hop-length of 1024 sample (≈ 90 ms at 11.025 kHz). After padding we end up with the following input shapes: 192×320 for CQT, and 128×320 for MEL.

5.1.2 Neural Networks

We use four CNNs that range from shallow to deep, which we respectively call: 2conv (2 layers), VGG (11 layers), ResNet18 (18 layers), and ResNet34 (34 layers). In the rest of this section we provide a more detailed description of the models. We use the notation $\#filters C_{stride}^{kernel}$, MP_{stride}^{kernel} , AP_{stride}^{kernel} , $\#filters R_{stride}^{kernel}$, and $\#units L$, to respectively

define: convolutional layer, max pooling layer, average pool layer, residual block, linear (dense) layer. The symbol $\circ$ denotes function composition, and raise to n -th power means the concatenation of the current layer n times.

2LR This network was proposed in the work of Thickstun et al. [108] for the polyphonic music transcription task, which we refer to as 2LR (2 LayeRs). It features two convolutional layers with filters designed to operate respectively on the frequency axis (kernel shape is $(h, 1)$), and on the time axis (kernel shape is $(1, w)$).

The first convolutional layer is supposed to capture timber patterns, while the second layer models temporal relationships of sound. Since only two layers are used, the kernel size is large in order to have an appropriate receptive filed. The kernel size w of the second layer is equal to the w size of its input, such that this dimension collapses to one. In other word, the second convolutional layer is looking at the entire frame context. Finally at the output a single linear classifier is used. Because this network was designed to process audio at 44.1kHz, we adapt its specifications of kernel size and stride to our scenario, as reported in Equation (5.1)

$$2LR := {}^{128}C_{1 \times 1}^{32 \times 1} \circ {}^{256}C_{1 \times 1}^{1 \times 32} \circ {}^{N_{\text{classes}}}L \quad (5.1)$$

VGG: this network was proposed for the image classification task, and led the way to more deep architectures [99]. It was shown that stacking many convolutional layers with a small kernel size of 3×3 allowed to achieve state-of-the-art performance at that time. In particular they increased: the number of convolution stacks (as well as their output features), the number of pooling stages, and the number of units of the final classifiers. The authors proposed different configurations of the VGG architecture in terms of total number of layers. In our work we chose the version with 11 layers, which is showed in Equation (5.2).

$$\begin{aligned} VGG := & \left({}^{64}C_{1 \times 1}^{3 \times 3} \circ MP_{2 \times 2}^{2 \times 2} \right)^2 \circ \left({}^{128}C_{1 \times 1}^{3 \times 3} \right)^2 \circ MP_{2 \times 2}^{2 \times 2} \\ & \circ \left({}^{256}C_{1 \times 1}^{3 \times 3} \right)^4 \circ MP_{2 \times 2}^{2 \times 2} \circ \left({}^{512}C_{1 \times 1}^{3 \times 3} \right)^2 \circ MP_{2 \times 2}^{2 \times 2} \\ & \circ {}^{4096}L \circ {}^{4096}L \circ {}^{N_{\text{classes}}}L \end{aligned} \quad (5.2)$$

ResNets

Making deeper and deeper architectures was the preferred way for improving the image classification performance. However, due to the gradient vanishing problem

there is a limit on how deep a network can be. Residual Networks (Resnets) were proposed as a new way of training these deep models based on residual blocks [50], which enable to effectively train very deep networks, up to one hundred layers. The residual block, Figure 5.1, is characterized by the identity connection, where the input of the block is summed to output of two convolutional layers. Residual connections

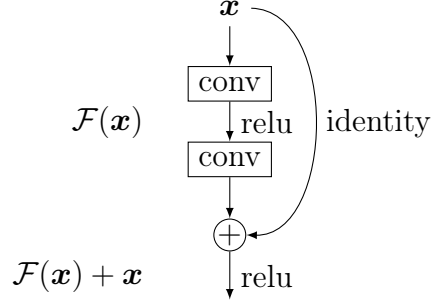


Figure 5.1: Residual block.

grant a better gradient flow, and thus enable training of very deep networks.

A variety of Resnet configuration were proposed as well as different types of residual blocks. In our work we use the residual block without bottleneck, and the 18 and 34, described respectively in Equation (5.3) and Equation (5.4).

$$\begin{aligned} \text{ResNet18} &:= {}^{64}C_{1 \times 1}^{7 \times 7} \circ {}^{64}R_{1 \times 1}^{3 \times 3} \circ {}^{64}R_{2 \times 2}^{3 \times 3} \\ &\quad \circ {}^{128}R_{1 \times 1}^{3 \times 3} \circ {}^{128}R_{2 \times 2}^{3 \times 3} \circ {}^{256}R_{1 \times 1}^{3 \times 3} \circ {}^{256}R_{2 \times 2}^{3 \times 3} \\ &\quad \circ {}^{512}R_{1 \times 1}^{3 \times 3} \circ {}^{512}R_{2 \times 2}^{3 \times 3} \circ AP \vdots \circ {}^{N_{\text{classes}}}L \end{aligned} \quad (5.3)$$

$$\begin{aligned} \text{ResNet34} &:= {}^{64}C_{1 \times 1}^{7 \times 7} \circ {}^{64}R_{1 \times 1}^{3 \times 3} \circ {}^{64}R_{2 \times 2}^{3 \times 3} \\ &\quad \circ ({}^{128}R_{1 \times 1}^{3 \times 3})^3 \circ {}^{128}R_{2 \times 2}^{3 \times 3} \circ ({}^{256}R_{1 \times 1}^{3 \times 3})^5 \circ {}^{256}R_{2 \times 2}^{3 \times 3} \\ &\quad \circ ({}^{512}R_{1 \times 1}^{3 \times 3})^2 \circ {}^{512}R_{2 \times 2}^{3 \times 3} \circ AP \vdots \circ {}^{N_{\text{classes}}}L \end{aligned} \quad (5.4)$$

Training

For adapting the networks to the multi-label classification scenario, instead of taking the softmax at the output, we compute the sigmoid and treat each element as probability. We use cross-entropy loss, and optimize the networks parameters with the Adam algorithm [59]. For all the experiments, the learning rate is fixed to 1×10^{-4} and the batch size to 32.

In the case of music transcription (MT), the training batch is composed by randomly choosing a track, and, randomly choosing a spectrogram frame within the track (with random uniform distribution). Instead, in the case of genre classification (GC) we just uniform randomly choose a clip, because we use the entire spectrogram.

The models are trained until convergence. It usually takes less than 100 epochs to converge in case of MT, and less than 10 for GC. In the case of MT an epoch is defined as the back-propagation of 1000 batches, while for GC, an epoch has the traditional meaning of training on all the samples available in the dataset.

The training of all the models for all the experimental configurations took approximately the equivalent of 1.6 GPU years on a powerful Compute Canada cluster.

Testing

Music transcription The test set consists of three recordings that cover most of the instruments present in the dataset, in a: small, medium and, large ensemble. For each track, a window spanning from the 1st second to 90th second is considered for prediction. The hop-length between consecutive predictions is set to 512 samples (in our case 128). 88 labels are used for this task.

Music genre classification The dataset comes with 80/10/10 split of training, validation and, testing. Therefore, since we use the large subset of the dataset, the test set is composed of $\approx 16k$ audio clips. We consider all the 30 s of audio for making a classification. 161 labels are used for this task.

Performance metric

We use micro average-precision (μAP) metrics to report classification performance. Each output class is treated independently as binary prediction. For all the possible thresholds value n we compute precision P_n and recall R_n . μAP summarizes the precision-recall curve as the weighted mean:

$$\mu AP = \sum_n P_n (R_n - R_{n-1}) \quad (5.5)$$

5.1.3 Datasets

In this section we describe the data-sets used in our experiments: Musicnet for the music transcription task and the Free Music Archive (FMA) for genre classification.

Musicnet

This is a large scale data-set of classical music for polyphonic music transcription [109]. The data-set consist of 330 freely licensed recordings (2048 minutes, 1 299 329 note labels) of classical music with a variety of instruments arranged in small chamber ensembles under various condition of studio and microphone.

The data-set is skewed towards Beethoven (1085 minutes, 736 072 labels) due to his popularity among performing ensembles. The data-set is also skewed to solo piano (1346 minutes, 794 532 labels) because of the large availability of digital scores for this particular instrument. The labels are structured according to the format: starting/ending time, instrument, note, measure, beat, and note value.

The labels are retrieved from digital MIDI scores, collected from various archives, and aligned to the recordings using techniques of Turetsky and Ellis [110] with an error rate of 4%.

Three recordings are used as the testing set which is a representative sampling of the full data-set: Bach's Prelude in D major for Solo piano, Mozart's Serenade in E-flat major, and Beethoven's String Quartet No. 13 in B-flat major.

Free Music Archive

This is a large scale data-set suitable for evaluating several MIR tasks, that comes directly with high quality audio ¹. It consists of 106 574 tracks from 16 341 artists and it is arranged by a taxonomy of 161 genres. There are 16 root genres and the remaining are sub-genres. This data-set is a dump of the Free Music Archive ² which is a free and open library where artists can upload their material under a permissive license.

The meta-data is provided by the artists at the moment of submission, and include: song title, duration, album, artist, and per-track genres; user data such as per-track/album/artist favorites, play counts, and comments; free-form text such as per-track/album/artist tags, album description and artist biography. The genre distribution of tracks is very skewed towards: experimental, electronic, rock and instrumental. The data-set is available in three different sets:

small 8000 30s clips from 8 top genres, balanced with 1000 clips per genre, 1 root

¹For other large scaled data-set e.g. the Million Songs Data-set, audio files are not directly available

²<http://freemusicarchive.org/>

genre per clip (similar to GTZAN [111]).

medium 25 000 30s clips, genre unbalanced with 21 up to 7103 clips per top genre, but only one of the 16 top genres per clip.

large entire data-set with audio limited to 30s.

full complete data-set with 161 genres, unbalanced with 1 up to 38 154 tracks per genre, and up to 31 genres per track.

5.1.4 Experimental Results

We investigate the classification performance of the considered CNNs for the MIR tasks of: polyphonic music transcription (MT) and, genre classification (GC). These tasks are characterized by: different types of sound sources and, different classification goals. This allows a more thorough evaluation that is not limited or biased to a particular task.

In particular, the chosen dataset for MT focuses on classical music, and it is mainly characterized by piano and strings instruments such as: violin, viola and, cello. On the other hand, the GC task covers all the remaining spectrum of music (although skewed to western music), with a wide variety of musical genres.

The time “resolution” of the classification is also very different for each of these two particular tasks. The MT task requires a fine resolution in time because we need to be able to detect notes potentially played at a fast rate. Therefore, the context window for this task is usually short and very frequent. On the other hand, the Genre classification task does not require any particular resolution in time, since the classification is done at clip level.

With the proposed experiments, our goal is to outline some “common” behaviours of convolutional architectures for music classification, which emerges from the joint analysis of multiple MIR tasks. In addition, we also investigate the performance of these models in presence of input signal degradation. In several real world applications the assumption of having “clean”, or high quality input audio data is not realistic. Different models may behave differently when the sound source is degraded. We report these results in the following parts of this section.

Experimental setup

The considered tasks can be formulated as multi-label classification problems. For MT task multiple notes can be simultaneously active in a given audio frame and, multiple genres are associated to a particular clip for the GC task/dataset.

All the data-sets are pre-processed as described in Sec. 5.1.1. We pre-compute the CQT, and the MEL version of each dataset, for the clean case as well as the “phone” and “hall” degradations. Although all the pre-processing can be done on-the-fly, precomputing it allow us to train the models considerably faster.

For each one of the considered CNN architectures we report performance in terms of average precision score for the following configurations of training and testing data: “clean/clean”, “clean/phone”, “phone/clean”, “clean/hall”, “hall/clean”.

Finally, we test our models in a data augmentation type of scenario, where we train on “clean”, “phone”, and “hall”, and test respectively on each one.

Clean

Experimental results on the clean configuration show that deeper models provides better classification performance. In particular, ResNets architectures and CQT input representation, are overall the best setup among the three different tasks.

Transcription Table 5.1 shows the classification performance for the MT task in the “clean/clean” configuration.

Table 5.1: Transcription results on “clean/clean” configuration.

	2LR	VGG	ResNet18	ResNet34
CQT	74.21	74.91	75.65	76.08
MEL	68.42	70.40	74.04	73.89

As can be observed, the CQT is the best input representation for all the architectures considered. This is indeed not surprisingly, because of the direct relationship of CQT bins to the note frequencies that are important in music transcription. ResNet models achieve the best performance for both the input representations, meaning that deeper networks are helpful for this task and configuration. The performance gap between ResNets and 2LR and VGG models is narrow in the CQT case, while it is more significant for the MEL spectrogram, especially when compared to the 2LR

model. This means that deeper models are less sensitive to the input representation for extracting “good” features for this task.

Genre Table 5.2 reports performance results on the “clean/clean” configuration for the genre classification task.

Table 5.2: Genre results on “clean/clean” configuration.

	2LR	VGG	ResNet18	ResNet34
CQT	35.32	39.60	38.96	39.09
MEL	33.28	38.04	38.49	38.19

For this task we notice a smaller gap between the CQT and MEL spectrograms for all the considered configurations. This means that the choice of a particular input feature has less impact for obtaining the best performance case, which is achieved with the CQT. Deep models perform similarly, within a 1.5% gap, meaning that deep models are better at modeling global temporal patterns among different music genres.

Degraded

In case of degraded audio with no data augmentation there is no complete agreement among different models and training/testing data configurations. On average among the task of MT and GC, we see that CQT offers better performance than MEL. The performance gap between CQT and MEL is particularly evident in the case of MT and the “clean/phone” data configuration. In all the other combinations, classification results are not affected by used input representation, something which is especially evident for the GC task.

Moreover, with the exception of a few combinations of models and data, ResNets are the best performing network across the two tasks; with not a noticeable difference between the 18, and 34 layer architectures.

Transcription At first instance we notice that, as expected, degradation penalizes the classification performance of all the models. The “hall” degradation is the most harmful one for this task, with considerably lower performance with respect to the “clean” scenario. Figure 5.2 gives an overall overview of the results for the music transcription task.

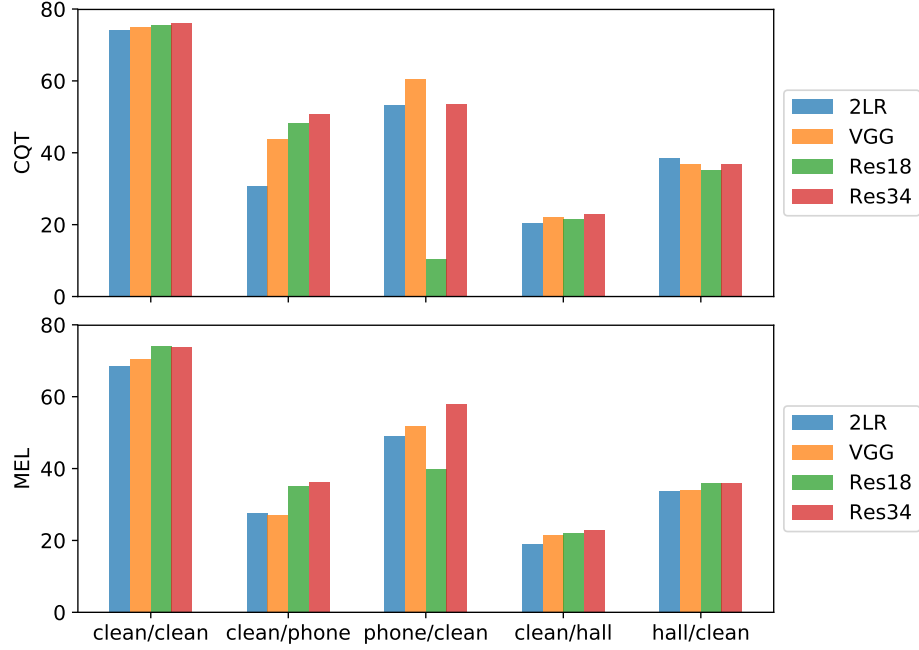


Figure 5.2: Results overview for music transcription task.

In the case of the “phone” degradation, Table 5.3, the CQT input spectrogram provides better performance compared to the MEL spectrogram. In addition, deeper networks are more robust to this type of degradation.

Table 5.3: Transcription results on “phone” degradation.

		2LR	VGG	ResNet18	ResNet34
CQT	clean/phone	30.63	43.82	48.14	50.82
	phone/clean	53.38	60.62	10.52	53.68
MEL	clean/phone	27.46	26.99	35.01	36.15
	phone/clean	48.96	51.88	39.89	57.99

The behaviour of the models, with the exception of Resnet34, is significantly influenced by the train/test configuration. Indeed, in the “phone/clean” configuration models perform consistently better than the “clean/phone”. The same trend is also true for the MEL spectrogram. In particular we see the severe performance drop of the 2LR network in the “clean/phone” configuration, for both input representations. The ResNet34 model offers the best performance for all the data configurations and input representations, with the exception of CQT spectrogram and “phone/clean”

configuration where VGG obtains the best result.

In the case of “hall” degradation, Table 5.4, performance results are less affected by the input representation. In fact all the models achieve similar performance for both

Table 5.4: Transcription results on “hall” degradation

		2LR	VGG	ResNet18	ResNet34
CQT	clean/hall	20.52	22.25	21.47	22.87
	hall/clean	38.67	36.77	35.28	36.81
MEL	clean/hall	19.08	21.50	22.16	22.99
	hall/clean	33.79	34.12	35.82	36.06

CQT and MEL spectrograms, as well as, for the two combinations “clean/hall” and “hall/clean”. The ResNet34 scores the best performance, although with a small margin, in all the cases with the exception of the CQT spectrogram and the “hall/clean” configuration for which the 2LR network provides the best result.

Genre

Figure 5.3 gives overall overview of the results for the genre classification task.

For this task, “phone” is the most harmful degradation. In the case of “phone”

Table 5.5: Genre results on “phone” degradation.

		2LR	VGG	ResNet18	ResNet34
CQT	clean/phone	21.81	25.86	25.87	24.13
	phone/clean	20.21	23.51	24.57	24.85
MEL	clean/phone	20.84	25.50	26.05	26.01
	phone/clean	19.60	24.09	25.59	25.19

degradation, Table 5.5, we notice a marginal performance difference between the CQT and MEL spectrogram, and almost symmetric results for the two configuration “clean/phone” and “phone/clean”. We also observe a clear gap of performance, $\approx 5\%$, between the shallow model and the others, leading us to the conclusion that deep models are more robust to degradation in this setup. In fact ResNets models achieve the best performance, even if with a small margin, across all the combinations of input, and data configuration.

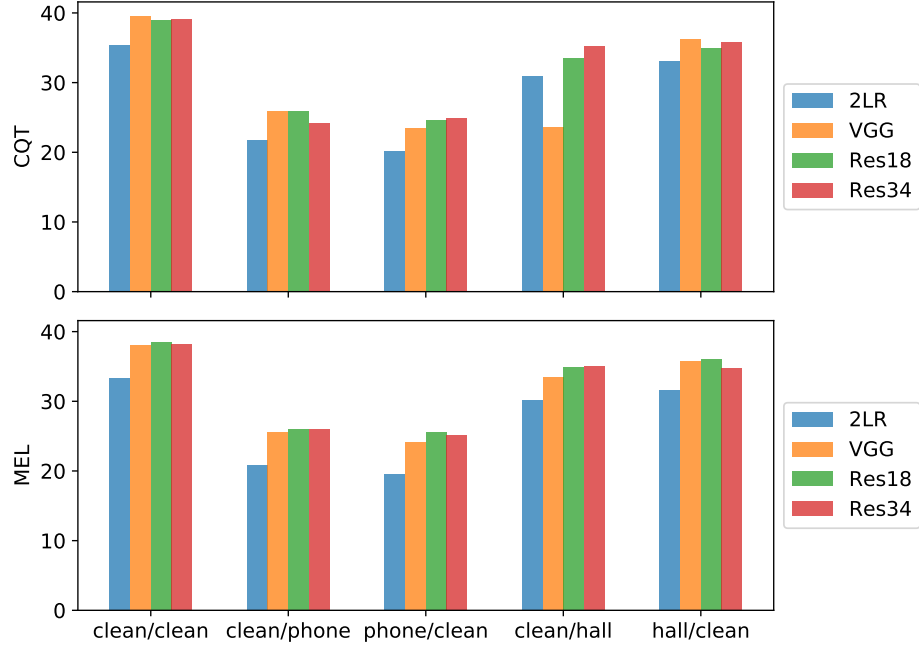


Figure 5.3: Results overview for genre classification task.

Similar conclusions are drawn for the “hall” degradation experiment, Table 5.6. Also in this case deeper models provide better performance. In particular, ResNets

Table 5.6: Genre results on “hall” degradation.

config	input	2LR	VGG	ResNet18	ResNet34
CQT	clean/hall	30.95	23.62	33.52	35.25
	hall/clean	33.03	36.30	34.93	35.81
MEL	clean/hall	30.23	33.46	34.93	34.98
	hall/clean	31.53	35.72	35.98	34.75

are the best performing architectures, with the exception of CQT input in “hall/clean” configuration, where VGG is slightly better.

Augmentation

In this section we discuss performance results in a data augmentation scenario where the degradations are known and can be applied to the training set. In this scenario “clean/phone/hall” are used as training data, and each of the “clean/phone/hall” data are in turn used as the testing data.

Also in this case, the CQT input representation yields better performance with respect to MEL. For the task of MT, the performance gap between CQT and MEL is $\approx 5\%$. For the GC task instead, the margin is lower at $\approx 1\%$.

The clear conclusion from this set of experiments is that ResNets models, in particular the 34 layers ResNet, achieves the best performance for all the tasks. This indicates that residual connections and deep architectures are beneficial when dealing with large datasets and degraded audio. In this scenario, ResNets model are able to scale better with respect to the others, and achieve performance compared to the original “clean/clean” configuration.

Music transcription For this task, Table 5.7 and Figure 5.4, we notice that the ResNet34 model in all the cases achieves the best performance.

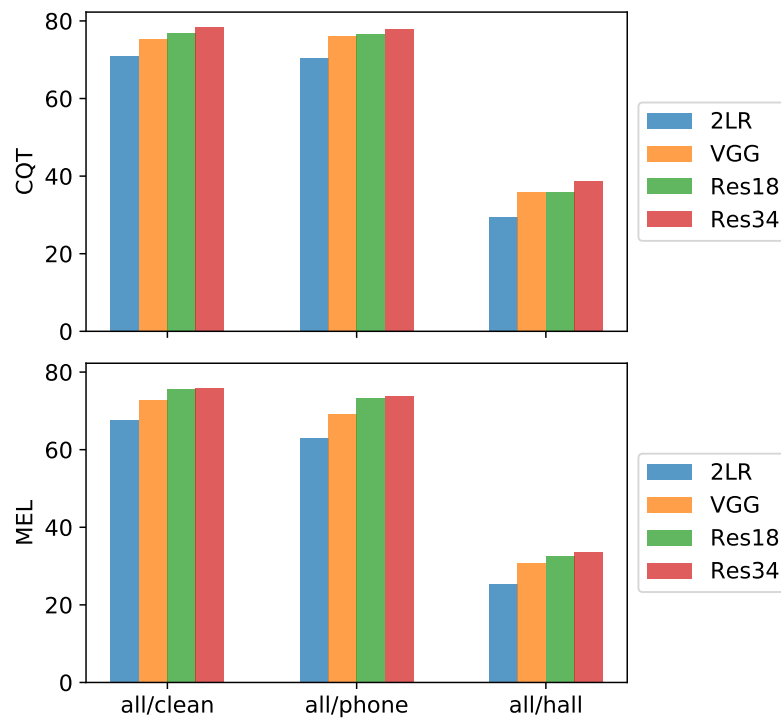


Figure 5.4: Transcription results using “all” degradations.

The performance margin is somewhat related to the network’s depth, and progressively decreases as the model gets deeper. The CQT spectrogram grants a significant improvement of performance, with respect to the MEL spectrogram.

Table 5.7: Transcription results using “all” degradations.

		2LR	VGG	ResNet18	ResNet34
CQT	all/clean	70.87	75.21	76.89	78.36
	all/phone	70.27	76.05	76.67	77.98
	all/hall	29.43	35.93	35.92	38.57
MEL	all/clean	67.75	72.88	75.64	75.93
	all/phone	63.04	69.08	73.38	73.78
	all/hall	25.28	30.77	32.67	33.69

In addition, we see that for VGG and ResNets models training on all the degradations serves as data augmentation, which increases the performance with respect to the “clean” case. However, this is not the case for the 2LR network, which is unable to improve over the baseline “clean” results. We observe the same behaviour for both CQT and MEL spectrogram.

Genre Also for this task, deeper models scale better with increasing data training samples, Table 5.8.

Table 5.8: Genre results using “all” degradations.

		2LR	VGG	ResNet18	ResNet34
CQT	all/clean	35.53	39.31	39.41	39.43
	all/phone	33.95	23.70	38.82	38.76
	all/hall	33.52	37.63	38.78	38.75
MEL	all/clean	34.32	38.79	38.90	39.26
	all/phone	31.17	37.62	37.57	37.62
	all/hall	32.32	37.51	38.27	38.93

Especially, we note that ResNets models allow for the best performance. Training on all the degradations is beneficial for the model which improves classification performance with respect to the “clean” training cases. CQT and MEL spectrograms allow for very close classification results, however, in average CQT performs slightly better.

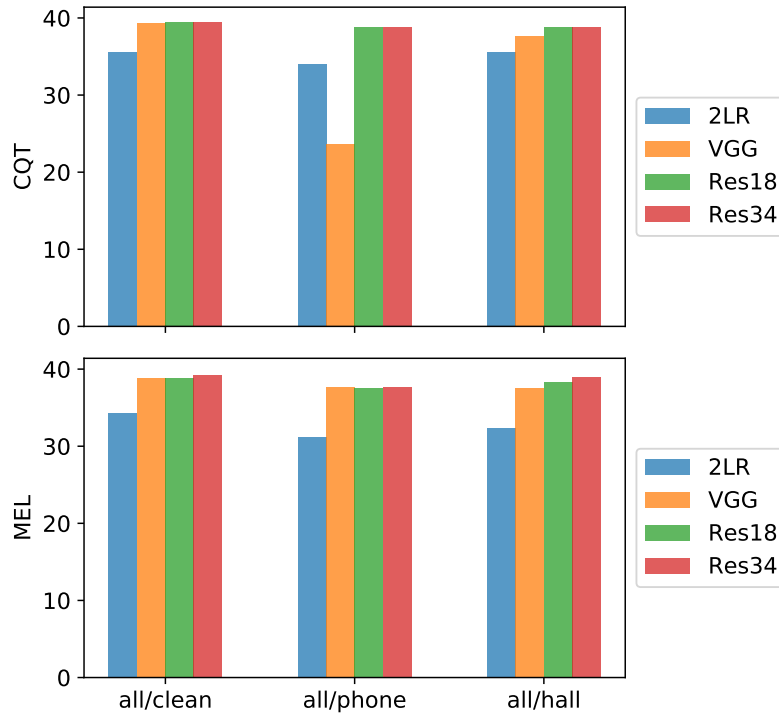


Figure 5.5: Genre results using “all” degradations.

5.1.5 Additional experiments

Although the main focus of the chapter is about DL architectures for music, we include an additional set of experiments using a different kind of audio source and analysis task. In this section, we report on experimental results for the sound event detection task, using the same evaluation scheme adopted in the previous section. In particular, we perform frame based multilabel classification of environmental urban sound clips. This additional set of experiments allowed us to investigate whether the invariance of the examined network architectures to degraded audio is also valid for non-musical audio input.

Dataset

This dataset was created with Scaper [93], starting from the UrbanSound8K [92], to generate a large dataset of 10 000 soundscapes for Sound Event Detection (SED). The dataset includes close to 50 000 annotated sound events. Each soundscape is 10 s long, and has a background of Brownian noise that resembles the typical “hum” often

heard in urban environments. Every soundscape contains between 1 to 9 sound events from the following classes: “air conditioner”, “car horn”, “children playing”, “dog bark”, “drilling”, “engine idling”, “gun shot”, “jackhammer”, “siren”, and “street music”. The dataset is split in: 6000 soundscapes for training, 2000 soundscapes for validation and, 2000 soundscapes for testing.

Input representation

For this specific data-set, authors make classifications of 1 s audio segments [93]. We use a hop length of 128 samples and therefore 86 frames of context are need to cover one second of audio. After padding we end up with the following input shapes: 192×86 for CQT, and 128×86 for MEL.

Clean

Table 5.9 summarizes classification results on the “clean/clean” configuration for the sound event classification task. In this case we see the ResNet34 is the best performing

Table 5.9: Sound results on “clean” configuration.

	2LR	VGG	ResNet18	ResNet34
CQT	49.76	51.12	55.35	57.51
MEL	45.58	53.68	53.89	55.87

model, for both input representations. Similarly to the MIR tasks, this implies that deeper models achieve better performance. The gap between the two types of input spectrogram is dependent on the model. With the exception of VGG, CQT gives the best performance for the remaining models with an improvement of $\approx 2\%$.

Degraded

Figure 5.6 gives overall overview of the results for the sound classification task. For the sound classification task, “phone” is the most harmful degradation, in particular when the CQT spectrogram is used. On average the MEL spectrogram offers better classification performance for this task.

In the case of “phone” degradation and CQT spectrogram, Table 5.11, the Resnet models achieve better performance, with a small margin with respect to the VGG model, and a little bit more consistent gap with respect to the 2LR. Results are

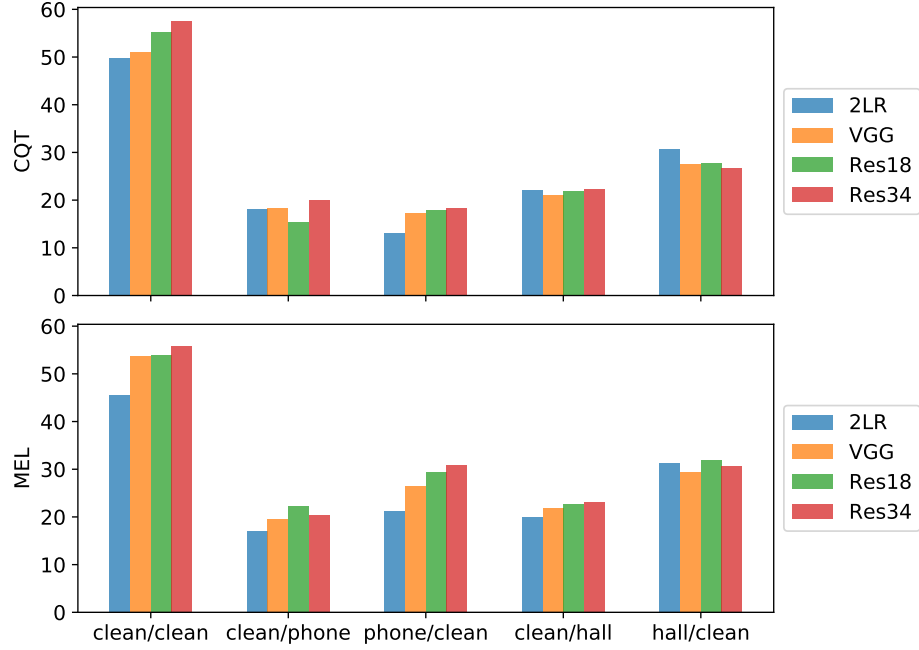


Figure 5.6: Results overview for the sound classification task.

Table 5.10: Sound results on “phone” degradation.

		2LR	VGG	ResNet18	ResNet34
CQT	clean/phone	18.14	18.38	15.38	20.11
	phone/clean	13.07	17.21	17.92	18.36
MEL	clean/phone	17.09	19.62	22.16	20.43
	phone/clean	21.21	26.45	29.30	30.84

roughly similar between the data configurations “clean/phone” and “phone/clean”, with the exception of 2LR, where we notice an $\approx 5\%$ gap.

When the MEL spectrogram is used as input, the models perform better, in particular for the “phone/clean” configuration. Also in this case, Resnet networks provide the best results. The performance gap between Resnets and the other models is particularly evident for the “phone/clean” configuration.

Also in the case of the “hall” degradation, MEL spectrograms are in average the best performing input representation, Table 5.11. The results in this case are heavily dependent on the data configuration. In particular, in the “hall/clean” configuration the models perform consistently better than in the “clean/hall” configuration. In

Table 5.11: Sound results on “hall” degradation.

		2LR	VGG	ResNet18	ResNet34
CQT	clean/hall	22.04	21.05	21.84	22.25
	hall/clean	30.76	27.49	27.82	26.67
MEL	clean/hall	20.00	21.91	22.60	23.01
	hall/clean	31.17	29.46	31.84	30.69

the “clean/hall” configuration we note similar performance with both CQT and MEL input representations. ResNet34 is the best performing model, although with a small margin. Conversely, in the case of “hall/clean”, 2LR achieves the best performance, with a small margin with MEL spectrogram, and a slightly bigger one when CQT is used as input.

Augmentation

Finally, also for this task we can draw similar conclusions as the previous two tasks. Data augmentation on average helps to improve the baseline performance of the models in the “clean” training scenario, Table 5.12 and Figure 5.7. In particular,

Table 5.12: Sound results using “all” degradations.

		2LR	VGG	ResNet18	ResNet34
CQT	all/clean	48.55	54.64	55.41	60.85
	all/phone	49.09	52.70	54.99	56.72
	all/hall	32.27	35.68	35.09	36.34
MEL	all/clean	45.95	55.17	58.31	59.07
	all/phone	45.46	53.87	54.96	56.74
	all/hall	29.40	35.24	36.32	36.21

we note that ResNet34 scores the best classification results in all the experimental configurations.

Regarding input representations, there is not a noticeable difference between CQT and MEL spectrograms. However, on average CQT performs slightly better than MEL.

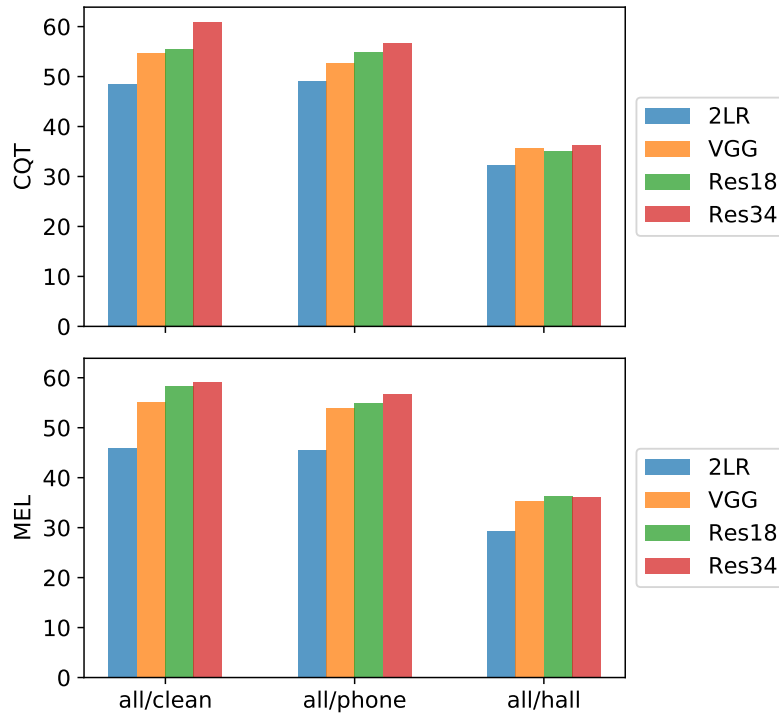


Figure 5.7: Sound results using “all” degradations.

5.1.6 Experiments with binary neural networks

In this section we evaluate the effectiveness of binary neural network for the MIR tasks of: polyphonic music transcription and genre classification.

We considered the binary counterparts of the CNN architectures utilized in the previous experiments. In this case we limit the experimental analysis to the configuration of “clean/clean” and CQT spectrograms.

Binary networks

A fundamental component of binary networks is batch normalization. In fact, the normalization of feature map activations to zero mean and standard deviation is what allows for binarization to not be “destructive”. Indeed, the dynamics of the signal after normalization is comparable to $-1, +1$ quantization values. For this reason, batch normalization is followed after each binary layer, both convolutional and linear. The hard-tanh is used as activation function, in contrast to relu activations. In addition, as usually happens for image classification tasks, the first layer of each network is

maintained in float-point precision because binary weights are not compatible with the input signal dynamics. Finally, it is important to use a floating-point final classifier output for the considered MIR tasks.

The models are trained in the same configurations as described in Section 5.1.2. Note that due to the presence of batch-normalization, the learning-rate could be increased for faster training.

Results

Tables 5.13 and 5.14 show the classification performance of the “binarized” CNNs, in terms of average precision score. One can notice that there is a significant drop in performance for the polyphonic music transcription task when the network is binarized. In contrast, for the genre classification task, the performance drop is more limited. Similarly to the case of music transcription, performance progressively decreases as the networks get deeper.

Music transcription Table 5.13 reports the classification results of the binary networks for the music transcription task. The best performing network in this case is the two convolutional layers 2LR. This result is not surprising because out of the three layers, just the second convolutional layer is binarized. We recall that the “input” (first), and “output” (last) layer of each network are in floating point precision. The performance drop for 2LR is $\approx 4\%$.

		Bin-2LR	Bin-VGG	Bin-ResNet18	Bin-ResNet34
CQT	clean/clean	69.63	37.72	17.17	8.61

Table 5.13: Music transcription results with binary neural networks.

In contrast, for the other networks, the performance drop is more extreme: $\approx 50\%$ for Bin-VGG, $\approx 80\%$ for Bin-ResNet18, and $\approx 90\%$ for Bin-ResNet34. The conclusion from this experiment is that deep binary networks, as proposed by Hubara et al. [53], simply do not work for the music transcription task. In contrast, it has been shown that binarization does not affect significantly the performance of networks in image classification tasks. A more thorough analysis should be done for better understanding what factors cause this failure of deep binary networks for music transcription compared to their success in image classification.

Genre Classification Table 5.14 reports classification results of the binary models for the genre classification task. In contrast to the transcription task, the performance drop in this case is less significant. case.

		Bin-2LR	Bin-VGG	Bin-ResNet18	Bin-ResNet34
CQT	clean/clean	27.71	34.96	31.23	23.27

Table 5.14: Genre classification results with binary neural networks.

The best performing model is Bin-VGG, with obtains a performance drop of $\approx 10\%$. Similarly to the previous case, the performance drop gets larger as the models get deeper. In fact, Bin-ResNet18 and Bin-ResNet34 respectively show a performance drop of $\approx 20\%$, and $\approx 40\%$. Finally, the Bin-2LR network is approximately as effective as the previous case, dropping in performance of $\approx 20\%$.

5.2 Improving music transcription with skip connections

Automatic Music Transcription (AMT) is the task of detecting the instruments' pitches, at any given time by analyzing the acoustic audio signal. In other word, AMT provides a way of converting *sound*, to *notes*, in a piano-roll notation. AMT is a challenging task, in particular when the input audio is polyphonic i.e multiple pitches in many cases played by different instruments can be present at any particular time. In fact, extracting pitches from polyphonic audio signals, without constraining the model to prior knowledge, is still an open problem. One immediate application of AMT is transcription of improvised performance, where musical scores are not available. In addition to that, AMT is also used in interactive music systems [41].

In the Deep Learning (DL) era, polyphonic music transcription has been mainly treated as a multi-label classification problem. According to this problem configuration, a Deep Neural Network (DNN) is trained on chunks of audio signal to predict the active notes within the corresponding analysis window. More specifically, Convolutional Neural Networks (CNNs) are used as classification models. These classification models are often trained on log-spaced (magnitude) spectrograms. Indeed, log-spaced spectrograms are the preferred input representation for music classification with CNNs, because patterns in the log-spaced frequency domain results are shift in-

variant when pitches change. There is no shift invariance of frequency patterns with linear-scale spectrograms.

In this section, we propose the use of the U-Net architecture as a way for improving transcription performance of existing neural network architectures for polyphonic music transcription. The U-Net architecture was initially developed, and mostly used, for medical image segmentation [90], because of its ability to reproduce tiny details. See Section 5.2.1 for a detailed description about this model.

We show the benefit of the proposed architecture through extensive experiments on the MusicNet dataset [109], focusing both on instrument and non-instrument based transcription. We use some popular neural networks as baseline transcription models, focusing in particular on deep architectures.

5.2.1 U-Net architecture

The U-Net architecture was initially proposed for medical image segmentation [90]. Image segmentation can be seen as a multi-class classification problem where each pixel of the input image is assigned to a particular class.

This architecture resembles an auto-encoder with skip-connections, where the feature maps of the encoder are copied and concatenated at the decoder. The introduction of skip-connections, enabled very precise segmentation results, and the U-Net architecture became the state-of-the-art model for this task.

This architecture is composed by an encoder and a decoder, and features skip-connections that connect the two sides, see Figure 5.8. We use the notation $\overset{\#filters}{C}^{\overset{kernel}{stride}}$, $MP^{\overset{kernel}{stride}}$, $AP^{\overset{kernel}{stride}}$, $\overset{\#filters}{R}^{\overset{kernel}{stride}}$, and $\overset{\#units}{L}$, to respectively define: convolutional, max pooling, average pool, residual block and, linear layer. The symbol $\circ$ means function and composition, $(.)^n$ means concatenating the considered layer n times, and $(.)^T$ stands for transposed convolution.

The encoder consists of blocks of two convolutional layers with 3×3 kernels, followed by ReLU activation. Each convolutional block is followed by max pooling layers of kernel and stride of 2×2 which halves the feature map size H and W . The first convolutional block has 64 filters, after each pooling step the number of filters is doubled.

The decoder instead, at each stage, takes the corresponding feature map and upsample it by a factor of 2, by means of transposed convolution with kernel and stride 2×2 . Feature map channels are instead halved at this step. After that,

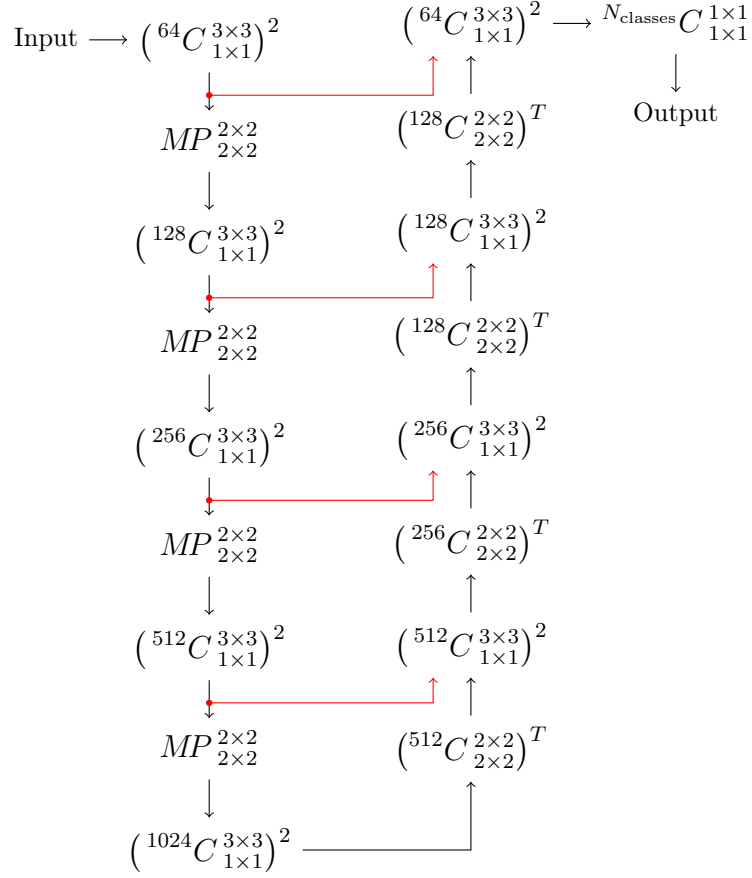


Figure 5.8: The U-Net architecture (red arrows represent skip-connections).

the corresponding feature map of the encoder is copied and concatenated, (skip-connections) along the channel dimension, with the transposed convolution output. A cascade of two 3×3 convolutional layers with ReLU activation is then applied.

Finally, at the last stage of the decoder, a 1×1 convolution is used to match the desired number of classes.

In addition to medical imaging, the U-Net architecture has also been successfully employed for MIR tasks. For example Jansson et al. [55] used U-Net for singing voice separation due to its ability of recreating fine details. In their work, the U-Net is configured to learn a spectrogram mask, which when applied to the input spectrogram isolates the voice component. U-Net has also been recently utilized for developing Optical Music Recognition (OMR) systems [46]. To the best of our knowledge the usage of the U-Net architecture has not been explored for automatic music transcription.

5.2.2 Proposed solution

The proposed solution consists in the combination of traditional CNNs for music transcription with a U-Net, where the U-Net is placed in front of a transcription network. More precisely, If we define U-Net as $f_\theta(\cdot)$, and the transcription network

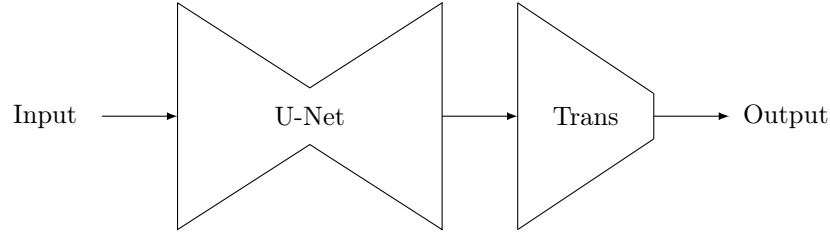


Figure 5.9: Proposed transcription architecture.

as $g_\phi(\cdot)$; then for a given input x , the proposed architecture computes:

$$z = g_\phi(f_\theta(x)) \quad (5.6)$$

In this configuration, the U-Net acts as a *transformation* network, which modifies the input signal to a “better” representation for the music transcription task.

This goal is achieved by training the model with a single cross-entropy loss at the output of the transcription back-end network. Specifically the loss is expressed by the following equation:

$$L_g = \frac{1}{N} \sum_{n=1}^N y_n \log(\hat{y}_n) + (1 - y_n) \log(1 - \hat{y}_n) \quad (5.7)$$

where y_n are the true labels, and $\hat{y}_n = 1/(1 - e^{z_n})$ for the n -th sample in each mini-batch.

By doing this we do not constrain U-Net’s output, leaving the network “free” to transform the input in a better representation for this task. We also investigated a configuration of the model that uses two losses: (i) a reconstruction loss at U-Net output, L_f , and (ii) the cross-entropy loss at transcription output, L_g :

$$L_{fg} = \frac{1}{N} \|x - f_\theta(x)\|_2^2 + L_g = L_f + L_g \quad (5.8)$$

However, we empirically found the double loss to not be as effective as the single loss. In fact, the proposed model trained with the double loss performs worse than

the same model trained with single loss. This leave us with the conclusion that: an unconstrained U-Net output is needed for achieving a “useful” transformation rather reconstruction. In addition to that, we empirically find skip connections to be fundamental for the proposed architecture to perform well in the single loss configuration. If the skip connections are removed, the proposed architecture achieves poor classification results. This means that, “copying” the features each convolutional layer of the encoder, and “concatenating” them to the input of each convolutional layer of the decoder, is a key step for obtaining a useful input transformation. In our opinion, skip-connections “guide” the transformation, which ensure the output not to drift too much from the original input. Specific experiments on these different configurations of the front-end network are described in Section 5.2.5.

We further investigated another front-end/back-end configuration. We remove the skip connections and introduce a reconstruction loss at the front-end output. In this configuration, the U-Net acts as an auto-encoder. However, through empirical evaluation, we notice that this configuration can not achieve the same performance of the proposed model with the single back-end loss.

In conclusion, a U-Net placed in front of a transcription network behaves as an “optimized” pre-processing step, that transforms the input in such a way that more meaningful features can be extracted by the transcription network, providing overall better classification performance when compared to the sole transcription network.

Transcription networks

The transcription networks used in this work are the same as the one used in Section 5.1. Please refer to Section 5.1.2 for a detailed description. Below we summarized the considered back-end networks: 2LR (5.9), VGG (5.10), ResNet18 (5.11), ResNet34 (5.12).

$$2LR := {}^{128}C_{1 \times 1}^{32 \times 1} \circ {}^{256}C_{1 \times 1}^{1 \times 32} \circ {}^{88}L \quad (5.9)$$

$$\begin{aligned} VGG := & \left({}^{64}C_{1 \times 1}^{3 \times 3} \circ MP_{2 \times 2}^{2 \times 2} \right)^2 \circ \left({}^{128}C_{1 \times 1}^{3 \times 3} \right)^2 \circ MP_{2 \times 2}^{2 \times 2} \\ & \circ \left({}^{256}C_{1 \times 1}^{3 \times 3} \right)^4 \circ MP_{2 \times 2}^{2 \times 2} \circ \left({}^{512}C_{1 \times 1}^{3 \times 3} \right)^2 \circ MP_{2 \times 2}^{2 \times 2} \\ & \circ {}^{4096}L \circ {}^{4096}L \circ {}^{88}L \end{aligned} \quad (5.10)$$

$$\begin{aligned} ResNet18 := & {}^{64}C_{1 \times 1}^{7 \times 7} \circ {}^{64}R_{1 \times 1}^{3 \times 3} \circ {}^{64}R_{2 \times 2}^{3 \times 3} \circ {}^{128}R_{1 \times 1}^{3 \times 3} \circ {}^{128}R_{2 \times 2}^{3 \times 3} \\ & \circ {}^{256}R_{1 \times 1}^{3 \times 3} \circ {}^{256}R_{2 \times 2}^{3 \times 3} \circ {}^{512}R_{1 \times 1}^{3 \times 3} \circ {}^{512}R_{2 \times 2}^{3 \times 3} \\ & \circ AP \circ {}^{88}L \end{aligned} \quad (5.11)$$

$$\begin{aligned}
\text{ResNet34} := & {}^{64}C_{1 \times 1}^{7 \times 7} \circ {}^{64}R_{1 \times 1}^{3 \times 3} \circ {}^{64}R_{2 \times 2}^{3 \times 3} \circ \left({}^{128}R_{1 \times 1}^{3 \times 3}\right)^3 \circ {}^{128}R_{2 \times 2}^{3 \times 3} \\
& \circ \left({}^{256}R_{1 \times 1}^{3 \times 3}\right)^5 \circ {}^{256}R_{2 \times 2}^{3 \times 3} \circ \left({}^{512}R_{1 \times 1}^{3 \times 3}\right)^2 \circ {}^{512}R_{2 \times 2}^{3 \times 3} \\
& \circ AP \vdots \circ {}^{88}L
\end{aligned} \tag{5.12}$$

Multi-instrument transcription

The dataset used in this work is labelled on an instrument basis, i.e. each note in the ground truth is paired with the corresponding instrument. Several tracks are also recordings of ensemble performance, where multiple instruments play together. In this work we leverage this additional information to extend music transcription to an instrument-wise transcription.

The proposed architecture is extended to instrument based transcription by replacing the single output classifier with a battery of classifiers, one for each instrument.

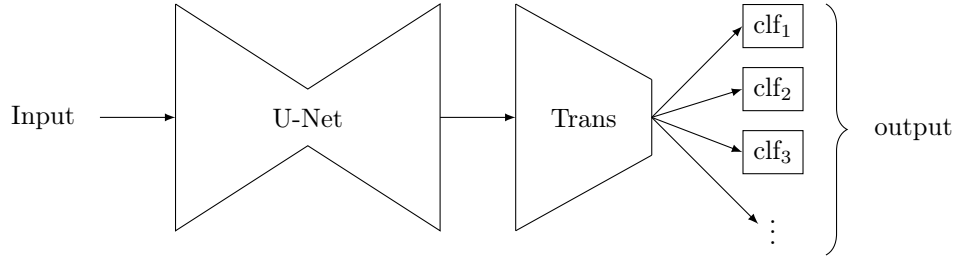


Figure 5.10: Instrument-wise transcription architecture.

The feature extraction part the of the transcription network is shared between all the classifiers. In addition, the number of output channels of the U-Net decoder matches the number of instruments involved in the transcription.

These instrument models are still trained with a single loss function, as described in Section 5.2.2.

5.2.3 Methodology

We experiment with four CNNs for polyphonic music transcription using the MusicNet dataset. Music transcription is formulated as a multi-label classification problem, where multiple notes can be active within a give analysis window.

First we establish the baseline transcription performance of the “plain” CNNs, and then evaluate the performance improvement of the proposed architecture.

Dataset

The MusicNet dataset [109] is a new large scale dataset of classical music specifically designed for the music transcription task. The dataset consists of 330 freely licensed recordings (2048 minutes, 1 299 329 labels) of classical music with a variety of instruments arranged in small chamber ensembles under various condition of studio and microphone.

The dataset is skewed towards Beethoven (1085 minutes, 736 072 labels) due to his popularity among performing ensembles. The dataset is also skewed to solo piano (1346 minutes, 794 532 labels) because of the large availability of digital scores for this particular instrument.

The MusicNet labels are structured according to the format: starting/ending time, instrument, note, measure, beat, and note value. The labels are retrieved from digital MIDI scores, collected from various archives, and aligned to the recordings using techniques of Turetsky and Ellis [110] with an error rate of 4%.

Three recordings are used as a testing set which is a representative sampling of the dataset: Bach’s Prelude in D major for Solo piano, Mozart’s Serenade in E-flat major, and Beethoven’s String Quartet No.13 in B-flat major.

Input representation

The dataset is preprocessed by computing the CQT magnitude spectrogram [19] of each recording. Indeed, as shown by experiments of Section 5.1.4, the CQT allows to obtain the best performance for music transcription task. CQT spectrograms are computed on 7 octaves with 24 bins per octave with a minimum frequency 32.7 Hz which yields 168 frequency bins in total.

In order to be comparable with the previous work of Thickstun et al.[109, 108] we adopt an equivalent setup for input preprocessing, rescaled to our sampling frequency of 11.025 kHz³. Therefore, the CQT spectrogram is computed with an hop length of 128 samples (≈ 12 ms). Finally, a context window of 32 frames is used for training and testing the neural networks.

³Audio and labels are re-sampled to 11.025 kHz using an implementation of the band-limited *sinc* interpolation method for sampling rate conversion as described by Smith [102].

Training

For adapting the networks to the multi-label classification scenario, instead of taking the softmax at the output, we compute the sigmoid, and treat each individual element as a probability value. We use cross-entropy loss, and optimize the network parameters with the Adam algorithm [59]. For all the experiments, the learning rate is fixed to 1×10^{-4} and the batch size is set to 32.

The training batch is composed by randomly choosing a track, and, randomly choosing a spectrogram frame (and associated context window) within the track with uniform distribution.

All the models are trained until convergence.

Performance metric

We use micro average-precision (μAP) metrics to report classification performance. Each output class is treated independently as binary prediction. For all the possible threshold values n , we compute precision P_n and recall R_n . μAP summarizes the precision-recall curve as the weighted mean:

$$\mu AP = \sum_n P_n (R_n - R_{n-1}) \quad (5.13)$$

5.2.4 Results

In this section we report and discuss music transcription results, in terms of average precision score. In the first part we focus on the “instrument-agnostic” based transcription. In this case instrument information is not utilized, which means that the transcribed notes are not associated with a particular instrument.

In the next part of this section, we focus on instrument based transcription. Due to data imbalances of the instrument labels, we first discuss transcription results considering only the “piano” and “non-piano” class. By doing so we obtain a roughly balanced dataset. Finally, we report performance results using all the instruments classes in the imbalanced data scenario.

Instrument-agnostic transcription

From Table 5.15 we notice that the proposed architecture based on U-Net, improves baseline results of the plain networks. The performance improvement is $\approx 1\%$, where

Table 5.15: “Instrument-agnostic” transcription results.

2LR(+Unet)	VGG(+Unet)	ResNet18(+Unet)	ResNet34(+Unet)
74.21 (75.63)	74.91 (75.05)	75.65 (76.40)	76.08 (76.83)

2LR, is the most improved network, and VGG the least improved network.

Instrument-wise transcription

Piano vs non-piano Table 5.16 reports transcription results for “piano”, and “non-piano” instruments. The proposed architecture based on U-Net improves classification

Table 5.16: “Piano”/“non-piano” transcription results.

	2LR(+Unet)	VGG(+Unet)	ResNet18(+Unet)	Resnet34(+Unet)
piano	72.96 (79.01)	78.60 (79.38)	79.75 (80.47)	78.20 (79.83)
non-piano	67.21 (70.03)	68.92 (70.66)	70.57 (72.85)	72.83 (72.28)
average	70.08 (74.52)	73.76 (75.02)	75.16 (76.66)	75.51 (76.06)

performance for all the considered CNNs. The performance improvement is large for the shallow models, $\approx 5\%$, and it reduces as the model gets deeper, becoming less than 1% for ResNet34.

All instruments Table 5.17 shows transcription results when all instrument labels are used. Also in this case the proposed architecture based on U-Net achieved an improvement with respect to the baseline models. The improvement is more evident

Table 5.17: All instrument transcription results.

	2LR(+Unet)	VGG(+Unet)	ResNet18(+Unet)	Resnet34(+Unet)
piano	67.98 (78.71)	74.60 (78.42)	77.70 (77.50)	80.19 (79.28)
violin	48.64 (49.30)	44.46 (50.10)	46.80 (47.30)	50.73 (52.40)
viola	32.53 (32.73)	31.80 (36.80)	36.35 (37.99)	33.72 (36.11)
cello	32.73 (37.16)	38.33 (43.37)	38.56 (37.80)	38.75 (40.55)
horn	67.53 (68.12)	64.38 (68.45)	70.38 (72.10)	63.20 (72.76)
bassoon	71.61 (69.35)	69.91 (72.94)	73.43 (68.09)	72.11 (66.71)
clarinet	64.79 (68.09)	63.98 (69.56)	67.89 (67.91)	68.53 (72.63)
average	55.12(57.85)	55.35(59.95)	58.73 (58.38)	58.18 (60.06)

when all the instruments are transcribed, and it ranges from $\approx 4\%$ for VGG network, to $\approx 2\%$ for ResNet34.

5.2.5 Additional results

This section reports additional experimental results using different configurations of the front-end networks. Table 5.18 shows transcription performance when skip connections are removed from the front-end network, and a single loss is used at the back-end output. In this configuration, the classification performance is very poor.

Table 5.18: Transcription results when skip-connections are removed.

	2LR+UAE	VGG+UAE	ResNet18+UAE	ResNet34+UAE
“agnostic”	9.02	9.28	9.57	9.10
“piano/non-piano”	5.09	4.86	4.98	5.10

The conclusion from this experiment is that skip-connections are fundamental for the proposed architecture to perform well. Due to the poor results of this configuration, we limit the analysis to “fused” instrument transcription and “piano”/“non-piano” instrument transcription.

Table 5.19 reports the transcription results when the U-Net front-end is replaced by an auto-encoder with the same specifications, and two losses are used for training. The one loss is the cross-entropy loss at the back-end output, and the other is the reconstruction loss at the front-end output, as shown in Equation 5.8. With the

Table 5.19: Transcription results when the front-end is an auto-encoder and the model is trained with double loss.

	2LR+AE	VGG+AE	ResNet18+AE	ResNet34+AE
“agnostic”	74.76	72.58	73.36	73.30
“piano/non-piano”	71.39	72.03	73.11	74.30
“all”	58.08	57.85	56.11	56.42

exception of some cases for the 2LR back-end network, we notice that this configuration of the front-end network is not as effective as the proposed architecture. The performance gap is particularly evident for VGG and ResNets. However, in the case of the simple two layers 2LR network performance gap is in average less evident.

5.3 Conclusion

This chapter investigated the use of DNNs for the MIR tasks of: polyphonic music transcription, and genre classification. To the best of our knowledge this is the first attempt in evaluating very deep networks for the aforementioned tasks. With thorough experiments we shown the effectiveness of deep architectures when the input sound source is degraded. Residual Neural Networks enable the training of deeper architecture and were shown to have superior performance in most configurations for both the scenario in which the audio degradation is not known and the scenario in which the audio degradations are known and can be used for data augmentation. In addition, the CQT input representation provides the best overall performance. Finally we proposed a front-end/back-end convolutional architecture based on the U-Net architecture for improving polyphonic music transcription performance. It was shown that using a single loss function provides better results than individual loss functions for both instrument-specific and instrument-agnostic transcription.

Chapter 6

Conclusion and future work

In this section we draw conclusions from the work presented in this thesis. In particular, Section 6.1 is related to the segmentation and classification of musical documents in text and music score. Section 6.2 deals with efficient computations with binary representation for binary image processing and binary neural networks. Finally, Section 6.3 reports conclusions related to neural networks architectures for music transcription and classification.

6.1 Document segmentation and classification

This thesis presents a novel document segmentation system capable of detecting musical scores and text within a digital image. Our system is characterized by a robust and extensible framework based on BoVW, for the analysis of information that doesn't make any prior assumptions about the content, and that can be potentially extended to treat other kinds of information besides scores and text. This is one of the contributions of our work. In fact, all the previous proposed methods for document segmentation make strong a priori assumptions about the content they want to model (such as graphics or text) that force the authors to design techniques which are too specific and do not scale outside that context.

Another contribution is the segmentation by random block voting. Here we propose a technique that is able to reconstruct the structure of a document by testing some random portion of it without making any assumption about the structure. The random votes are later summarized in a coarse segmentation that constitutes a guideline structure which is used to obtain a finer segmentation in the last stage of the

system.

The performance is evaluated conducting two experiments on different datasets. The metric used is the precision and recall of the overlapping area of detection with respect to the ground truth. One test-set is artificially created and consists of images that combine multiple instances of text and score with high variability of aspect, size, font and layout. In this challenging scenario our systems achieves an F-measure equal to 86%. The other test-set, comes from the scanning of two real music books, where one has a modern typographical layout and the other a vintage one. Also in this case the proposed algorithm achieves good results, namely an F-measure of 85%. These two experiments attest the capability of our algorithm to perform at the same time, a good classification into musical score and text under high variability, and a good layout segmentation.

6.2 Efficient computation with binary representations

This thesis presents an optimized data structure for binary image processing, and an easy to use framework built on top of it. Experiments applied to several common binary processing algorithms show how our library can be successfully employed, providing a consistent improvement while remaining easy to use. In particular, we show how the new proposed way of representing a binary image is particularly well suited to efficient bit-wise techniques of processing.

Regarding optimized binary neural networks, we present *Espresso*, a highly optimized forward-propagation framework for both traditional DNNs as well as BCNNs, that supports heterogeneous deployment on CPU and GPU. While BinaryNet and Nervana/neon BDNN implementations are limited to MLP networks, our framework also supports the popular CNN while simultaneously outperforming state-of-the-art implementations of MLP networks. *Espresso* is highly-efficient, light-weight and self-contained. Computation on the GPU side is done through specifically designed CUDA kernels, which, combined with a more careful handling of memory allocation and bit-packing, allows us to obtain considerable performance improvements.

6.3 Neural networks for music transcription and classification

We have conducted extensive experiments to investigate the performance of deep convolutional neural networks in combination with two log-scale spectrogram representations (CQT and MEL), for the MIR tasks of polyphonic music transcription and genre classification. In particular, we first establish the performance upper bound results on the “clean” (not degraded) scenario. We then evaluate the robustness of the considered CNNs to audio degradation using different combinations of training and testing data. Two types of representative audio degradations are utilized: low quality reproduction device, and a chamber reverb.

For both the “clean” and the “degraded” scenario we notice the effectiveness of deep architectures, specifically ResNets. In fact, in average, the ResNet is the best performing model across the two MIR tasks. The same conclusion is verified for the data augmentation experiment where clean and degraded audio are used for training. Also in this case, the ResNet is the superior model in terms of classification results.

Regarding the input representation, in the “clean” scenario the CQT representation obtains better performance compared to the MEL representation, with a more clear advantage in the music transcription case. In case of degradations, overall there is no preferred input representation across all configurations. The CQT representation tends to perform better when the degradation is on the testing set, while the MEL representation seems to be more useful when the degradation is on the training set.

Finally, we conduct an additional set of experiments on a non-musical dataset, for the task of urban sound classification. Similar conclusions apply for this type of audio source, making our conclusions potential applicable to a broader set of applications scenarios not limited to the musical domain.

We further propose a convolutional architecture for improving polyphonic music transcription results of the baseline architectures used in the aforementioned experiment. The proposed architecture is composed of a transformation network, U-Net, which is put in front the transcription networks. With thorough experiments we show that the proposed architecture improves transcription performance of the baseline, both in the case of “non-instrument” and “instrument”-wise classification scenarios.

6.4 Future work

Regarding the document segmentation and classification work, since the segmentation algorithm was designed to be content independent, one interesting extension would be to apply the proposed technique to other types of digital documents. An example could be the segmentation of text, natural images, and artificial images (such as charts or logos). These classes are the most common among general purpose documents, and it would be valuable to experiment in this scenario. Another extension of this work, could be to replace the traditional machine learning approach for classification with deep learning methods. A larger scale dataset of historical digitized music would need to be used for this extension to provide more training data. Another interesting future direction would be to experiment with promising segmentation techniques proposed in computer vision, such as R-CNN [40, 49] and YOLO [89].

Regarding the binary image processing work, more algorithms could be implemented, and the API could be refined and made clearer. This work is relatively self contained and we do not see much further research in this direction, with the exception of applying the proposed framework in real world applications.

Regarding the binary neural network research, an interesting future direction would be to investigate in more detail the use and effectiveness of binary networks for other tasks than image classification. Indeed, previous work on image classification has shown that binary neural networks, despite their aggressive quantization of weight and activations, can still provide decent classification performance while considerably simplifying the model. However, in our experiments on music classification and transcription, we observed that the classification performance of binary networks is significantly lower compared to their floating point counterparts. It would be useful to understand why this is the case, and propose architectural variations that make binary models perform better for music. One research direction could be to try less aggressive quantizations and see how classification performance changes related to that. Layer specific quantizations can also be investigated. In order to understand why binary networks (as proposed for image classification) achieve poor performance for music, a statistical analysis of feature map activations and gradients can be conducted, and compared between the two different domains. A possible reason for the significant performance drop of binarized networks for music, could be related to the precision required to represent the input signal. In fact, in the case of images, an 8-bit input is used. However, more precision (at least double) is required to represent an

high-quality audio signals, as the ones used in our experiments. With that being said, it is still true that the first convolutional layer operates with floating point precision, and therefore it should not be adversely affected by higher precision of the input signal. However, batch normalization and subsequent binarization might not be as effective in the case of music because of the different dynamics of the signals in contrast to images. In addition to that, in order to also speed-up also the training phase, probabilistic approximation of the back-propagation algorithm can be investigated. By introducing additional re-scaling terms, a binarized gradient can be back-propagated such that binary optimized matrix multiplication can be leveraged. Moreover, instead of training a binary network from scratch as required by the BinaryNet algorithm, knowledge transfer techniques can be investigated such that the binarized network can be derived from the correspondent floating point network without the need to retrain.

Regarding the work on neural networks for music, a possible extension could be to (i) extend the experimental analysis to more types of degradation and evaluate, and (ii) identify what a good trade-off between network depth and performance can be. In addition to that, the use of U-Net as a front-end for improving classification performance, can be extended to other domains, such as image classification. Preliminary experiments on image classification using the CIFAR10 dataset, suggest that the benefit of the front-end transformation network is not limited to the music transcription tasks. We further hypothesize that the proposed front-end/back-end architecture can be used for “un-engineered” data augmentation. In fact, the number of channels of U-Net output can be arbitrarily set. For example, if we start with a single channel input, the n outputs of U-Net can be seen as n input transformations, which expand the training set by a factor of n . Specifically related to the music transcription task, an interesting extension would be to focus on the instrument based source-separation. For instance, in our setup a multi-instrument signal can be fed to into transcription architectures that provide instrument-wise labels. Now, if we are interested in separating a single instrument, the labels related to all the other instruments can be set to zero. At this point, the L_2 loss between the multi-instrument and single-instrument (artificially created) can be used to optimize for input by gradient descent. Given this problem formulation we do not require perfect invertibility but only differentiability, thus, magnitude spectrogram can still be part of the processing pipeline.

Appendix A

Publicly available software

In this chapter we list the publicly available github repository where the thesis work is hosted. All the developed software during thesis work has been made available under open-source license on github. We strongly encourage researchers to share their implementations such that the results can be easily reproduced, further investigated and improved by future work of other researchers. The list of project repositories is provided in the following part of this chapter.

A.1 Document segmentation and classification

<https://github.com/fpeder/mscr>

A.2 SPmat

<https://github.com/fpeder/spmat>

A.3 Espresso

<https://github.com/fpeder/espresso>

A.4 Neural network for music transcription and classification

<https://github.com/fpeder/music>

Appendix B

Publications

- Pedersoli Fabrizio, Tzanetakis George. Document Segmentation and Classification into Musical Score and Text. In *International Journal on Document Analysis and Recognition (IJDAR)*, 19(4):289–304, 2016.
- Pedersoli Fabrizio, Tzanetakis George. SPmat: A Framework and Data Representation for Binary Image Processing. In *IEEE International Workshop on Multimedia Signal Processing (MMSP)*, 2018.
- Pedersoli Fabrizio, Tzanetakis George, Tagliasacchi Andrea. Espresso: Efficient Forward Propagation for Binary Deep Neural Networks. In *International Conference on Learning Representations (ICLR)*, 2018.
- Pedersoli Fabrizio, Yi Kwang Moo, Tzanetakis George. Residual Networks for Transcription and Classification of Degraded Audio. [TO BE SUBMITTED TO] *IEEE Transaction on Audio, Speech, and Language Processing*, 2019.
- Pedersoli Fabrizio, Yi Kwang Moo, Tzanetakis George. Improving Music Transcription With Skip-Connections. [TO BE SUBMITTED TO] *International Symposium on Music Information Retrieval (ISMIR)*, 2019.

B.1 Publications not related to the thesis

- Pedersoli Fabrizio, Benini Sergio, Adami Nicola, Leonardi Riccardo. XKin: an Open Source Framework for Hand Pose and Gesture Recognition using Kinect. In *The Visual Computer*, 30(10):1107-1122, 2014.

- Pedersoli Fabrizio, Adami Nicola, Benini Sergio, Leonardi Riccardo. XKin – Extendable Hand Pose and Gesture Recognition Library for Kinect. In *ACM International Conference on Multimedia*, 2012.
- Pedersoli Fabrizio, Benini Sergio, Adami Nicola, Okuda Masahiro, Leonardi Riccardo. Subpatch: Random KD-Tree on a Sub-Sampled Patch Set for Neighbor Field Estimation. In *International Conference on Machine Vision (ICMV)*, 2014.

Appendix C

Source code examples

C.1 SPmat

Listing C.1: Optimized 3×3 erosion

```

#define PPIX(m, i, k) \
    m.pix.data[(i) * m.pix.N + k]

#define E3_START(in, i, j) \
    PPIX(in, i, j) & \
    (PPIX(in, i, j) >> 1) & \
    ((PPIX(in, i, j) << 1) | \
    (PPIX(in, i, j + 1) >> 63))

#define E3_IN(in, i, j) \
    PPIX(in, i, j) & \
    ((PPIX(in, i, j) << 1) | \
    (PPIX(in, i, j + 1) >> 63)) & \
    ((PPIX(in, i, j) >> 1) | \
    (PPIX(in, i, j - 1) << 63))

#define E3_END(in, i, j) \
    PPIX(in, i, j) & \
    (PPIX(in, i, j) << 1) & \
    ((PPIX(in, i, j) >> 1) | \
    (PPIX(in, i, j - 1) << 63))

void

```

```

spmat_erode3(spmat in, spmat ou) {
    int i, j;

    for (i=1, j=0; i < in.pix.M - 1; i++) {
        PPIX(ou, i, j) = (E3_START(in, i-1, j) &
                          E3_START(in, i, j) &
                          E3_START(in, i+1, j));

        for (j = 1; j < in.pix.N - 1; j++) {
            PPIX(ou, i, j) = (E3_IN(in, i-1, j) &
                              E3_IN(in, i, j) &
                              E3_IN(in, i+1, j));
        }

        PPIX(ou, i, j) = (E3_END(in, i-1, j) &
                          E3_END(in, i, j) &
                          E3_END(in, i+1, j));
    }
}

```

Listing C.2: Optimized run length extraction

```

#define MID(s, e) (((e)-(s)) >> 1)
#define MASK64(s, e) \
    ((0xFFFFFFFFFFFFFFFF << (63-(e)+(s))) >> (s));

static int off = 0, id = 0, rl = 0;
static int16_t ss[RL_MAX_ROW];
static int16_t se[RL_MAX_ROW];

static void split(uint64_t v, int s, int e);
static void merge(int r, uint32_t *pt);

static void split(uint64_t v, int s, int e) {
    uint64_t mask = MASK64(s, e);
    uint64_t tmp = v & mask;
    if (!tmp)
        return;
    if (tmp == mask) {
        ss[id] = s + off;
        se[id] = e + off;
        id++;
    }
}

```

```

        return;
    } else {
        split(v, s, s + MID(s, e));
        split(v, s + MID(s, e) + 1, e);
    }
}

static void merge(int r, uint32_t *pt) {
    for (int i = 0; se[i] != -1; i++) {
        pt[rl++] = POINT_PACK(r, ss[i]);
        while (ss[i + 1] == se[i] + 1)
            i++;
        pt[rl++] = POINT_PACK(r, se[i]);
    }
}

void spmat_rl(spmat src, uint32_t *pt) {
    uint64_t ppix;
    id = 0;
    rl = 0;
    off = 0;
    for (int i = 0; i < src.pix.M; i++) {
        for (int j = 0; j < src.pix.N; j++) {
            ppix = src.pix.data[i * src.pix.N + j];
            split(ppix, 0, 63);
            off += OFF;
        }
        ss[id] = -1;
        se[id] = -1;
        id = 0;
        off = 0;
        merge(i, pt);
    }
    pt[rl] = 0;
}

```

Listing C.3: Optimized contour extraction

```

#define POINT_PACK(x, y) ((x) << 16) | y

const uint8_t map[] = {4,5,6,7,0,1,2,3};
const int8_t di[] = {0,1,1,1,0,1,1,1};

```

```

const int8_t dj[] = {-1,-1,0,1,1,1,0,-1};

inline uint8_t
find_next(uint8_t neigh, uint8_t prev) {
    __asm__ ("movb    %0,    %%cl    \n"
             "rorb    %%cl,  %1      \n"
             "movzbl %1,    %%eax    \n"
             "bsr     %%eax, %%eax    \n"
             "addb    %%al,  %0      \n"
             "andb    $7,%0          \n"
             : "+r"(prev)
             : "r"(neigh)
             : "%cl", "%eax");
    return prev;
}

void
spmat_contour(spmat in, uint32_t *contour) {
    uint8_t *data = (uint8_t *)in.ngh.data;
    uint8_t next, prev = 0;
    uint32_t start = 1, end = 0;
    int i, j, k = 0;

    spmat_first(in, &i, &j);
    start = POINT_PACK(i, j);
    while (start != end) {
        next = find_next(data[i * in.N + j], prev);
        i += di[next];
        j += dj[next];
        end = POINT_PACK(i, j);
        contour[k++] = end;
        prev = map[next];
    }
}

```

Listing C.4: Optimized Guo-Hall thinning

```

static int iter;
static const uint8_t rtab[] = {...};
static const uint8_t ltab[] = {...};

int gh_iter_lookup(spmat src, spmat dst) {

```

```

    const uint8_t *tab = iter ? rtab : ltab;
    uint8_t *in = (uint8_t *)src.ngh.data;
    uint8_t pix, ngh;
    int dels = 0;

    for (int i = 1; i < src.M - 1; i++) {
        for (int j = 1; j < src.N - 1; j++) {
            pix = in[i * src.N + j + 1] & 1;
            ngh = in[i * src.N + j];
            if (pix && tab[ngh]) {
                spmat_del(dst, i, j);
                dels++;
            }
        }
    }
    iter ^= 1;
    return dels;
}

void spmat_guohall(spmat src, spmat dst) {
    iter = 1;
    int dels = 0;
    spmat tmp = spmat_alloc_copy(src);
    do {
        dels = gh_iter_loookup(tmp, dst);
        spmat_copy(dst, tmp);
    } while (dels);

    spmat_free(tmp);
}

```

Bibliography

- [1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: a system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016.
- [2] Ahmad Abdelfattah, Azzam Haidar, Stanimire Tomov, and Jack Dongarra. Performance, design, and autotuning of batched GEMM for GPUs. In *International Conference on High Performance Computing*, pages 21–38. Springer, 2016. <http://icl.cs.utk.edu/magma/> (Link verified on May 11th, 2017).
- [3] Apostolos Antonacopoulos, Christian Clausner, Christos Papadopoulos, and Stefan Pletschacher. Icdar 2013 competition on historical newspaper layout analysis (hnla 2013). In *Document Analysis and Recognition (ICDAR), 2013 12th International Conference on*, pages 1454–1458. IEEE, 2013.
- [4] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.
- [5] Vassileios Balntas, Karel Lenc, Andrea Vedaldi, and Krystian Mikolajczyk. Hpatches: A benchmark and evaluation of handcrafted and learned local descriptors. In *CVPR*, 2017.
- [6] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *Computer vision–ECCV 2006*, pages 404–417. Springer, 2006.
- [7] Emmanouil Benetos and Simon Dixon. Polyphonic music transcription using note onset and offset detection. In *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*, pages 37–40. IEEE, 2011.

- [8] Emmanouil Benetos and Simon Dixon. A shift-invariant latent variable model for automatic music transcription. *Computer Music Journal*, 36(4):81–94, 2012.
- [9] Emmanouil Benetos, Sebastian Ewert, and Tillman Weyde. Automatic transcription of pitched and unpitched sounds from polyphonic music. In *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, pages 3107–3111. IEEE, 2014.
- [10] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.
- [11] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: A CPU and GPU math compiler in python. In *Proc. 9th Python in Science Conf*, pages 1–7, 2010.
- [12] Rachel M Bittner, Brian McFee, Justin Salamon, Peter Li, and Juan Pablo Bello. Deep salience representations for f0 estimation in polyphonic music. In *ISMIR*, pages 63–70, 2017.
- [13] Dan S Bloomberg. Connectivity-preserving morphological image transformations. *Xerox PARC Report EDL*, pages 91–1, 1991.
- [14] Dan S Bloomberg. Implementation efficiency of binary morphology. In *Proceedings of the international symposium of mathematical morphology VI*, pages 209–18, 2002.
- [15] Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152. ACM, 1992.
- [16] Nicolas Boulanger-Lewandowski, Yoshua Bengio, and Pascal Vincent. Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. *arXiv preprint arXiv:1206.6392*, 2012.
- [17] G. Bradski. The OpenCV Library. *Dr. Dobb’s Journal of Software Tools*, 2000.

- [18] Thomas M Breuel. The ocropus open source ocr system. In *Electronic Imaging 2008*, pages 68150F–68150F. International Society for Optics and Photonics, 2008.
- [19] Judith C Brown. Calculation of a constant q spectral transform. *The Journal of the Acoustical Society of America*, 89(1):425–434, 1991.
- [20] Syed Saqib Bukhari, Mayce Ibrahim Ali Al Azawi, Faisal Shafait, and Thomas M. Breuel. Document Image Segmentation Using Discriminative Learning over Connected Components. In *Proceedings of the 9th IAPR International Workshop on Document Analysis Systems*, pages 183–190. ACM, 2010.
- [21] Laura Caponetti, Ciro Castiello, and Przemysław Górecki. Document page segmentation using neuro-fuzzy approach. *Applied Soft Computing*, 8(1):118–126, January 2008.
- [22] J.S. Cardoso, A. Capela, A. Rebelo, and C. Guedes. A connected path approach for staff detection on a music score. In *Proceedings of International Conference on Image Processing. ICIP*, pages 1005–1008, October 2008.
- [23] Santanu Chaudhury, Megha Jindal, and Sumantra Dutta Roy. Model-Guided Segmentation and Layout Labelling of Document Images Using a Hierarchical Conditional Random Field. In *Pattern Recognition and Machine Intelligence*, number 5909, pages 375–380. Springer Berlin Heidelberg, 2009.
- [24] Keunwoo Choi, George Fazekas, and Mark Sandler. Automatic tagging using deep convolutional neural networks. *arXiv preprint arXiv:1606.00298*, 2016.
- [25] Keunwoo Choi, György Fazekas, Mark Sandler, and Kyunghyun Cho. Convolutional recurrent neural networks for music classification. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2392–2396. IEEE, 2017.
- [26] François Chollet et al. Keras: The python deep learning library. *Astrophysics Source Code Library*, 2018.
- [27] Melissa Cote and Alexandra Branzan Albu. Texture sparseness for pixel classification of business document images. *International Journal on Document Analysis and Recognition (IJDAR)*, 17(3):257–273, 2014.

- [28] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Training deep neural networks with low precision multiplications. *arXiv preprint arXiv:1412.7024*, 2014.
- [29] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in Neural Information Processing Systems*, pages 3123–3131, 2015.
- [30] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to ± 1 or -1 . *arXiv preprint arXiv:1602.02830*, 2016.
- [31] C. Dalitz, M. Droettboom, B. Pranzas, and I. Fujinaga. A Comparative Study of Staff Removal Algorithms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(5):753–766, May 2008.
- [32] V.P. d’Andecy, J. Camillerapp, and I. Leplumey. Kalman filtering for segment detection: application to music scores analysis. In *Proceedings of the 12th International Conference on Pattern Recognition. IAPR*, volume 1, pages 301–305 vol.1, October 1994.
- [33] Michaël Defferrard, Kirell Benzi, Pierre Vandergheynst, and Xavier Bresson. Fma: a dataset for music analysis. *arXiv preprint arXiv:1612.01840*, 2016.
- [34] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.
- [35] Sander Dieleman and Benjamin Schrauwen. End-to-end learning for music audio. In *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, pages 6964–6968. IEEE, 2014.
- [36] Jack J Dongarra, Jeremy Du Croz, Sven Hammarling, and Iain S Duff. A set of level 3 basic linear algebra subprograms. *ACM Transactions on Mathematical Software (TOMS)*, 16(1):1–17, 1990.
- [37] J. dos Santos Cardoso, A. Capela, A. Rebelo, C. Guedes, and J. Pinto da Costa. Staff Detection with Stable Paths. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(6):1134–1139, June 2009.

- [38] Michael Droettboom, Karl MacMillan, and Ichiro Fujinaga. The Gamera framework for building custom recognition systems. In *Symposium on Document Image Understanding Technologies*, pages 275–286. Citeseer, 2003.
- [39] Alicia Fornés and Gemma Sánchez. Analysis and Recognition of Music Scores. In *Handbook of Document Image Processing and Recognition*, pages 749–774. Springer, 2014.
- [40] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.
- [41] Masataka Goto. A real-time music-scene-description system: Predominant-f₀ estimation for detecting melody and bass lines in real-world audio signals. *Speech Communication*, 43(4):311–329, 2004.
- [42] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*, pages 6645–6649. IEEE, 2013.
- [43] Graham C Grindlay and Daniel PW Ellis. A probabilistic subspace model for multi-instrument polyphonic transcription. 2010.
- [44] Yiwen Guo, Anbang Yao, and Yurong Chen. Dynamic network surgery for efficient dnns. In *Advances In Neural Information Processing Systems*, pages 1379–1387, 2016.
- [45] Zicheng Guo and Richard W Hall. Parallel thinning with two-subiteration algorithms. *Communications of the ACM*, 32(3):359–373, 1989.
- [46] Jan Hajic jr, Matthias Dorfer, Gerhard Widmer, and Pavel Pecina. Towards full-pipeline handwritten omr with musical symbol detection by u-nets. In *Proceedings of the 19th International Society for Music Information Retrieval Conference, Paris, France*, pages 23–27, 2018.
- [47] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. EIE: efficient inference engine on compressed deep neural network. In *Proceedings of the 43rd International Symposium on Computer Architecture*, pages 243–254. IEEE Press, 2016.

- [48] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.
- [49] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, pages 2980–2988. IEEE, 2017.
- [50] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.
- [51] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.
- [52] Toyokazu Hori, Shinichiro Wada, Howzan Tai, and S. Y. Kung. Automatic music score recognition/play system based on decision based neural network. In *Multimedia Signal Processing, 1999 IEEE 3rd Workshop on*, pages 183–184. IEEE, 1999.
- [53] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In *Advances in Neural Information Processing Systems*, pages 4107–4115, 2016. <https://github.com/MatthieuCourbariaux/BinaryNet>.
- [54] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and < 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.
- [55] Andreas Jansson, Eric Humphrey, Nicola Montecchio, Rachel Bittner, Aparna Kumar, and Tillman Weyde. Singing voice separation with deep u-net convolutional networks. 2017.
- [56] Il-Young Jeong and Kyogu Lee. Learning temporal features using a deep neural network and its application to music genre classification. In *ISMIR*, pages 434–440, 2016.

- [57] Norm Jouppi. Google supercharges machine learning tasks with TPU custom chip, 2017.
- [58] R. Kapela and A. Rybarczyk. The Neighboring Pixel Representation for Efficient Binary Image Processing Operations. In *International Symposium on Parallel Computing in Electrical Engineering (PARELEC'06)*, pages 396–404, September 2006.
- [59] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [60] Matej Kristan, Jiri Matas, Aleš Leonardis, Tomas Vojir, Roman Pflugfelder, Gustavo Fernandez, Georg Nebel, Fatih Porikli, and Luka Čehovin. A novel performance evaluation methodology for single-target trackers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(11):2137–2155, Nov 2016.
- [61] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. The CIFAR-10 dataset. <https://www.cs.toronto.edu/~kriz/cifar.html>, 2009. Link verified on May 11th, 2017.
- [62] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.
- [63] Yann LeCun, Corinna Cortes, and Christopher J.C. Burges. The MNIST database of handwritten digits. <http://yann.lecun.com/exdb/mnist>, 1998. Link verified on May 11th, 2017.
- [64] Yann LeCun, LD Jackel, Leon Bottou, A Brunot, Corinna Cortes, JS Denker, Harris Drucker, I Guyon, UA Muller, Eduard Sackinger, et al. Comparison of learning algorithms for handwritten digit recognition. In *International conference on artificial neural networks*, volume 60, pages 53–60. Perth, Australia, 1995.
- [65] Chang-Hsing Lee, Jau-Ling Shih, Kun-Ming Yu, and Hwai-San Lin. Automatic music genre classification based on modulation spectral analysis of spectral and cepstral features. *IEEE Transactions on Multimedia*, 11(4):670–682, 2009.

- [66] Honglak Lee, Peter Pham, Yan Largman, and Andrew Y Ng. Unsupervised feature learning for audio classification using convolutional deep belief networks. In *Advances in neural information processing systems*, pages 1096–1104, 2009.
- [67] Fei-Fei Li, Rob Fergus, and Antonio Torralba. Recognizing and learning object categories. *Tutorial at ICCV*, 2005.
- [68] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.
- [69] Brian K Lien. Efficient implementation of binary morphological image processing. *Optical engineering*, 33(11):3733–3738, 1994.
- [70] Beth Logan et al. Mel frequency cepstral coefficients for music modeling. In *ISMIR*, volume 270, pages 1–11, 2000.
- [71] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.
- [72] Pradipta Maji and Shaswati Roy. Rough-fuzzy clustering and multiresolution image analysis for text-graphics segmentation. *Applied Soft Computing*, 30:705–721, May 2015.
- [73] Song Mao, Azriel Rosenfeld, and Tapas Kanungo. Document structure analysis algorithms: a literature survey. In *Electronic Imaging 2003*, pages 197–207. International Society for Optics and Photonics, 2003.
- [74] Matthias Mauch and Sebastian Ewert. The audio degradation toolbox and its application to robustness evaluation. In *Proceedings of the 14th International Society for Music Information Retrieval Conference (ISMIR 2013)*, Curitiba, Brazil, 2013. accepted.
- [75] Anders Meng, Peter Ahrendt, Jan Larsen, and Lars Kai Hansen. Temporal feature integration for music genre classification. *IEEE Transactions on Audio, Speech, and Language Processing*, 15(5):1654–1664, 2007.
- [76] Hidetoshi Miyao and Masayuki Okamoto. Stave extraction for printed music scores using dp matching. *JACIII*, 8(2):208–215, 2004.

- [77] Juhan Nam, Jiquan Ngiam, Honglak Lee, and Malcolm Slaney. A classification-based polyphonic piano transcription approach using learned feature representations. In *ISMIR*, pages 175–180, 2011.
- [78] Rajib Nath, Stanimire Tomov, and Jack Dongarra. An improved magma gemm for fermi graphics processing units. *The International Journal of High Performance Computing Applications*, 24(4):511–515, 2010.
- [79] Intel NervanaSystems. The Neon deep learning framework. <https://github.com/NervanaSystems/neon>. Link verified on May 11th, 2017.
- [80] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*, volume 2011, page 5, 2011.
- [81] Sergio Oramas, Francesco Barbieri, Oriol Nieto, and Xavier Serra. Multimodal deep learning for music genre classification. *Transactions of the International Society for Music Information Retrieval*. 2018; 1 (1): 4-21., 2018.
- [82] Nobuyuki Otsu. A threshold selection method from gray-level histograms. *Automatica*, 11(285-296):23–27, 1975.
- [83] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.
- [84] Roger D Peng. Reproducible research in computational science. *Science*, 334(6060):1226–1227, 2011.
- [85] Graham E Poliner and Daniel PW Ellis. A discriminative model for polyphonic piano transcription. *EURASIP Journal on Advances in Signal Processing*, 2007(1):048317, 2006.
- [86] W. K. Pratt. *Digital Image Processing 4th Edition*. Wiley, 2007.
- [87] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, pages 525–542. Springer, 2016.

- [88] Ana Rebelo, G. Capela, and Jaime S. Cardoso. Optical recognition of music symbols. *International Journal on Document Analysis and Recognition (IJ-DAR)*, 13(1):19–31, 2010.
- [89] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.
- [90] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.
- [91] Matti P Ryyanen and Anssi Klapuri. Polyphonic music transcription using note event modeling. In *Applications of Signal Processing to Audio and Acoustics, 2005. IEEE Workshop on*, pages 319–322. IEEE, 2005.
- [92] Justin Salamon, Christopher Jacoby, and Juan Pablo Bello. A dataset and taxonomy for urban sound research. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 1041–1044. ACM, 2014.
- [93] Justin Salamon, Duncan MacConnell, Mark Cartwright, Peter Li, and Juan Pablo Bello. Scaper: A library for soundscape synthesis and augmentation. In *Applications of Signal Processing to Audio and Acoustics (WASPAA), 2017 IEEE Workshop on*, pages 344–348. IEEE, 2017.
- [94] Jean Serra and Pierre Soille. *Mathematical morphology and its applications to image processing*, volume 2. Springer Science & Business Media, 2012.
- [95] Faisal Shafait, Daniel Keysers, and Thomas M. Breuel. Performance evaluation and benchmarking of six-page segmentation algorithms. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 30(6):941–954, 2008.
- [96] Etienne Sicard. An efficient method for the recognition of printed music. In *Proceedings of ICPR*, pages 573–573. IEEE, 1992.
- [97] Siddharth Sigtia, Emmanouil Benetos, and Simon Dixon. An end-to-end neural network for polyphonic piano music transcription. *IEEE/ACM Transactions on Audio, Speech and Language Processing (TASLP)*, 24(5):927–939, 2016.

- [98] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.
- [99] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.
- [100] Paris Smaragdis, Bhiksha Raj, and Madhusudana Shashanka. A probabilistic latent variable model for acoustic modeling. *Advances in models for acoustic processing, NIPS*, 148:8–1, 2006.
- [101] Paris Smaragdis, Bhiksha Raj, and Madhusudana Shashanka. Sparse and shift-invariant feature extraction from non-negative data. In *Acoustics, Speech and Signal Processing, 2008. ICASSP 2008. IEEE International Conference on*, pages 2069–2072. IEEE, 2008.
- [102] Julius O. Smith. *Digital Audio Resampling Home Page Center for Computer Research in Music and Acoustics*, 2015 (accessed Dec 12, 2018). <http://ccrma.stanford.edu/~jos/resample/>.
- [103] Irvin Sobel. Neighborhood coding of binary images for fast contour following and general binary array processing. *Computer Graphics and Image Processing*, 8(1):127–135, August 1978.
- [104] Bolan Su, Shijian Lu, U. Pal, and C.L. Tan. An Effective Staff Detection and Removal Technique for Musical Documents. In *IAPR International Workshop on Document Analysis Systems*, pages 160–164, March 2012.
- [105] Ilya Sutskever, Geoffrey E Hinton, and Graham W Taylor. The recurrent temporal restricted boltzmann machine. In *Advances in neural information processing systems*, pages 1601–1608, 2009.
- [106] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.
- [107] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabi-

- novich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.
- [108] John Thickstun, Zaid Harchaoui, Dean P Foster, and Sham M Kakade. Invariances and data augmentation for supervised music transcription. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2241–2245. IEEE, 2018.
- [109] John Thickstun, Zaid Harchaoui, and Sham Kakade. Learning features of music from scratch. *arXiv preprint arXiv:1611.09827*, 2016.
- [110] Robert J Turetsky and Daniel PW Ellis. Ground-truth transcriptions of real music from force-aligned midi syntheses. 2003.
- [111] George Tzanetakis and Perry Cook. Musical genre classification of audio signals. *IEEE Transactions on speech and audio processing*, 10(5):293–302, 2002.
- [112] Rein van den Boomgaard and Richard van Balen. Methods for fast morphological image transforms using bitmapped binary images. *CVGIP: Graphical Models and Image Processing*, 54(3):252–258, May 1992.
- [113] Aäron Van Den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. In *SSW*, page 125, 2016.
- [114] Olli Viikki and Kari Laurila. Cepstral domain segmental feature vector normalization for noise robust speech recognition. *Speech Communication*, 25(1-3):133–147, 1998.
- [115] Yu-Te Wu, Berlin Chen, and Li Su. Automatic music transcription leveraging generalized cepstral features and deep learning. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 401–405. IEEE, 2018.
- [116] Zhang Xianyi, Wang Qian, and Werner Saar. OpenBLAS: An optimized BLAS library. <http://www.openblas.net>. Link verified on May 11th, 2017.
- [117] Changsheng Xu, Namunu C Maddage, Xi Shao, Fang Cao, and Qi Tian. Musical genre classification using support vector machines. In *Acoustics, Speech, and*

- Signal Processing, 2003. Proceedings.(ICASSP'03). 2003 IEEE International Conference on*, volume 5, pages V–429. IEEE, 2003.
- [118] TY Zhang and Ching Y. Suen. A fast parallel algorithm for thinning digital patterns. *Communications of the ACM*, 27(3):236–239, 1984.
 - [119] Weibin Zhang, Wenkang Lei, Xiangmin Xu, and Xiaofeng Xing. Improved music genre classification with convolutional neural networks. In *INTERSPEECH*, pages 3304–3308, 2016.
 - [120] Ruohua Zhou and Marco Mattavelli. A new time-frequency representation for music signal analysis: Resonator time-frequency image. In *Signal Processing and Its Applications, 2007. ISSPA 2007. 9th International Symposium on*, pages 1–4. IEEE, 2007.
 - [121] Chenzhuo Zhu, Song Han, Huizi Mao, and William J Dally. Trained ternary quantization. *arXiv preprint arXiv:1612.01064*, 2016.
 - [122] F. Zirari, A. Ennaji, S. Nicolas, and D. Mammass. A Document Image Segmentation System Using Analysis of Connected Components. In *International Conference on Document Analysis and Recognition (ICDAR)*, pages 753–757, August 2013.

Glossary

Bag of Visual Words In computer vision, the bag-of-words model (BoW model) can be applied to image classification, by treating image features as words. In document classification, a bag of words is a sparse vector of occurrence counts of words; that is, a sparse histogram over the vocabulary. 3

Beam In musical notation, a beam is a horizontal or diagonal line used to connect multiple consecutive notes (and occasionally rests) to indicate rhythmic grouping. 12

Beat In music and music theory, the beat is the basic unit of time. The beat is often defined as the rhythm listeners would tap their toes to when listening to a piece of music, or the numbers a musician counts while performing, though in practice this may be technically incorrect (often the first multiple level). 19

Conditional Random Fields Conditional random fields (CRFs) are a class of statistical modeling method often applied in pattern recognition and machine learning and used for structured prediction. 14

Constant-Q Transform In mathematics and signal processing, the constant-Q transform transforms a data series to the frequency domain. The transform can be thought of as a series of logarithmically spaced filters f_k , with the k -th filter having a spectral width δf_k equal to a multiple of the previous filter's width. 20

Convolutional Neural Network In deep learning, a convolutional neural network is a class of deep neural networks, most commonly applied to analyzing visual imagery. CNNs use a variation of multilayer perceptrons designed to require minimal preprocessing. They are also known as shift invariant, based on their shared-weights architecture and translation invariance characteristics. 2

CUDA CUDA is a parallel computing platform and application programming interface (API) model created by Nvidia. It allows software developers and software engineers to use a CUDA-enabled graphics processing unit (GPU) for general purpose processing. 4

Deep Learning Deep learning is part of a broader family of machine learning methods based on learning data representations, as opposed to task-specific algorithms.. 1

Deep Neural network A deep neural network is an artificial neural network (ANN) with multiple layers between the input and output layers. 2

Expectation Maximization In statistics, an expectation–maximization (EM) algorithm is an iterative method to find maximum likelihood or maximum a posteriori (MAP) estimates of parameters in statistical models, where the model depends on unobserved latent variables. 22

GPU A graphics processing unit (GPU) is a specialized electronic circuit designed to rapidly manipulate and alter memory to accelerate the creation of images in a frame buffer intended for output to a display device. 1

Hidden Markov Models Hidden Markov Model (HMM) is a statistical Markov model in which the system being modeled is assumed to be a Markov process with unobserved (i.e. hidden) states. 20

K-means k-means clustering is a method of vector quantization, originally from signal processing, that is popular for cluster analysis in data mining. k-means clustering aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster. 30

LDA Linear discriminant analysis (LDA) is a generalization of Fisher’s linear discriminant, a method used in statistics, pattern recognition and machine learning to find a linear combination of features that characterizes or separates two or more classes of objects or events. 14

Mel Frequency Cepstral Coefficients In sound processing, the mel-frequency cepstrum (MFC) is a representation of the short-term power spectrum of a sound, based on a linear cosine transform of a log power spectrum on a nonlinear mel scale of frequency. 20

Non-negative Matrix Factorization Non-negative matrix factorization (NMF or NNMF), also non-negative matrix approximation is a group of algorithms in multivariate analysis and linear algebra where a matrix V is factorized into (usually) two matrices W and H , with the property that all three matrices have no negative elements. 22

Notehead In music, a notehead is the elliptical part of a note. Noteheads may be the same shape but colored completely black or white, indicating the note value (i.e., rhythmic duration). 12

Optical Character Recognition Optical character recognition or optical character reader, often abbreviated as OCR, is the mechanical or electronic conversion of images of typed, handwritten or printed text into machine-encoded text. 3

Optical Music Recognition Optical music recognition (OMR) or Music OCR is the application of optical character recognition to interpret sheet music or printed scores into editable or playable form.. 3

Pitch Pitch is a perceptual property of sounds that allows their ordering on a frequency-related scale, or more commonly, pitch is the quality that makes it possible to judge sounds as "higher" and "lower" in the sense associated with musical melodies. 19

Principal Component Analysis Principal component analysis (PCA) is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables (entities each of which takes on various numerical values) into a set of values of linearly uncorrelated variables called principal components. 13

Random Forest Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time and outputting the class

that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. 30

Recurrent Neural Network A recurrent neural network (RNN) is a class of artificial neural network where connections between nodes form a directed graph along a temporal sequence. This allows it to exhibit temporal dynamic behavior. Unlike feedforward neural networks, RNNs can use their internal state (memory) to process sequences of inputs. 21

Residual Neural Network A residual neural network (ResNet) is an artificial neural network (ANN) of a kind that builds on constructs known from pyramidal cells in the cerebral cortex. Residual neural networks do this by utilizing skip connections, or short-cuts to jump over some layers. 3

Slur A slur is a symbol in Western musical notation indicating that the notes it embraces are to be played without separation (that is, with legato articulation). 12

Staff In Western musical notation, the staff (US) or stave (UK) is a set of five horizontal lines and four spaces that each represent a different musical pitch or in the case of a percussion staff, different percussion instruments. 10, 12

Stem In musical notation, stems are the, "thin, vertical lines that are directly connected to the [note] head." 12

Support Vector Machine In machine learning, support-vector machines (SVMs) are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. 2

Tempo In musical terminology, tempo ("time" in Italian) is the speed or pace of a given piece. In classical music, tempo is typically indicated with an instruction at the start of a piece (often using conventional Italian terms) and is usually measured in beats per minute (or bpm). 19