

École polytechnique de Louvain

Exploratory study of machine learning classification techniques for historical bowed instruments

Author: **Romain FOCHON**

Supervisors: **François GLINEUR, Anne-Emanuelle CEULEMANS**

Readers: **Philémon BEGHIN, Paul FISETTE, John LEE**

Academic year 2025–2026

Master [120] in Computer Science

Abstract

Since the standardization of the violin family in the mid-eighteenth century, numerous instruments have undergone dimensional reductions in order to conform to emerging size conventions. While such reductions have traditionally been identified through qualitative visual inspection by luthiers and musicologists, no fully quantitative and automatic methodology has been established.

This thesis explores three machine learning approaches for the identification of violin reduction based on data obtained through photogrammetry on a corpus of historical instruments. The three different approaches investigated are Multiple Kernel Learning (MKL), Convolutional Neural Networks (CNN), and Multi-layer Perceptrons (MLP). The CNN approach constitutes the main focus of this work, while MKL constitutes more of an interpretability extension of the previous SVM attempt from [BCG25] and MLP are primarily used as simpler baseline for potential future work. The study discusses their respective strengths, limitations, interpretability, and robustness in the context of an extremely small, imbalanced dataset.

The results show that MKL achieves extremely high classification performance, with several configurations reaching up to 100% balanced accuracy. This approach confirms the importance of geometric descriptors derived from contour lines, and highlights the redundancy of the discriminative signal across multiple features. However, it also confirmed a strong sensitivity to hyperparameter tuning, raising concerns regarding model stability and generalization.

In contrast, the CNN approach demonstrates that elevation maps contain some useful discriminative information, but fails to achieve high classification performance, with a maximum balanced accuracy of 65.0%. The experiments show that full-resolution inputs introduce significant noise, and that simpler representations, such as resampled or reduced grids, lead to better results. The MLP approach confirms this observation, achieving slightly higher performance (up to 68.0%) with lower computational cost, suggesting that the essential discriminative information is already captured by more compact geometric descriptors.

Overall, this work highlights the trade-off between interpretability, performance, and stability across different modeling strategies. While kernel-based methods achieve the highest performance, their instability may limit their reliability. Neural networks approaches, although more flexible, remain constrained by the small dataset and the weak and subtle discriminative signal. These findings suggest that future work should focus on enlarging the dataset, improving input representations, and maybe developing hybrid approaches that combine the interpretability of geometric descriptors with the flexibility of learning-based models.

Acknowledgments

I would first like to express my sincere gratitude to Professor François Glineur, one of my supervisors, for his guidance, rigor, and constant support throughout this thesis. His expertise and critical thinking have greatly helped me to identify issues and to deliver such quality of work.

I'm also really grateful to Philémon Beghin for his assistance throughout the year. His prior work laid the foundations of this thesis, and all his help was essential in first navigating the data and several other aspects of this work.

I would also like to thank Professor Anne-Emmanuelle Ceulemans, my other supervisor, for sharing her deep knowledge of violins and historical instruments and for all the help she provided during this year.

On a more personal note, I want to thank Justine Mauel for her endless support and encouragement throughout this journey. Her presence has been a constant source of motivation.

Computational resources have been provided by the Consortium des Équipements de Calcul Intensif (CÉCI), funded by the Fonds de la Recherche Scientifique de Belgique (F.R.S.-FNRS) under Grant No. 2.5020.11 and by the Walloon Region

Finally, I acknowledge the use of ChatGPT, Grammarcheck, and Reverso tools for assistance with grammar, spelling, and language refinement. While all scientific content and interpretations remain my own, these tools helped improve clarity and readability.

Contents

Introduction	1
Background/context	1
Problem statement and objectives	2
Structure of this thesis	3
1 Related works and literature review	4
1.1 Background concepts	4
1.2 Previous work	8
1.2.1 Data acquitision and quality guarantee	8
1.2.2 Geometrically comparable objects	8
1.2.3 Contour lines and parametric function	8
1.2.4 Standardization of profiles length	10
1.2.5 First classification attempt and feature evaluation	10
1.2.6 Discussions and remaining challenges	11
1.3 State of the Art	12
1.3.1 Multiple Kernel Learning (MKL)	12
1.3.2 Convolutional Neural Network (CNN)	14
1.3.3 Cross-validation	16
2 Dataset	18
2.1 Data types/organization	18
2.2 Classes (imbalanced)	19
3 Multiple Kernel Learning (MKL) approach	20
3.1 Motivations	20
3.2 Algorithm and architecture	21
3.2.1 Feature organization and kernel definition	21
3.2.2 Construction of kernel blocks	22
3.2.3 EasyMKL algorithm	22
3.2.4 LOOCV evaluation	23
3.2.5 Extraction of kernel weights	24
3.3 Experiments and results	24
3.3.1 First experiment overview	24
3.3.2 Model parameters	25
3.3.3 Importance of zones and parameters	28
3.3.4 Overall results and discussion	31

4	Convolutional Neural Network (CNN) approach	32
4.1	Motivations	32
4.1.1	Limitations of parametric representations	32
4.2	Algorithm and architecture	33
4.2.1	Input representation and preprocessing	33
4.2.2	Data augmentation	34
4.2.3	CNN architectures	35
4.2.4	Training and evaluation	38
4.3	Experiments and results	39
4.3.1	First hyperparameters tuning (baseline)	39
4.3.2	Effect of hyperparameters	43
4.3.3	Architecture comparison	46
4.3.4	Influence of input representation	51
4.3.5	Effect of data augmentation	54
4.3.6	Overall comparison and discussion	55
5	Multi-Layer Perceptron (MLP) approach	58
5.1	Motivations	58
5.2	Algorithm and architecture	58
5.2.1	Input representation	58
5.2.2	MLP architectures	58
5.2.3	Training and evaluation	60
5.3	Experiments and results	60
5.3.1	First experiment overview	60
5.3.2	Effect of number of epoch	61
5.3.3	Effect of inner K-Fold strategy	62
5.3.4	Overall comparison and discussion	64
	Conclusion	65
	Summary and results of the different approaches	65
	Limitations and future work	66

Introduction

Background/context

The violin family emerged in Northern Italy at the end of the sixteenth century and spread relatively rapidly across Europe. During that early period (until mid-eighteenth century), instrument-making was characterized by enormous diversity. Instruments differed greatly across regions but also among the works of a single luthier. Around the mid-eighteenth century, more uniform size standards for violins, violas, and cellos became established. Instruments that did not respect these new standards were often physically modified. One possible type of modification consisted of reducing the width of the instrument by removing a longitudinal slice of wood from the instrument and gluing the two halves back together, as shown in Figure 1 [BCG25].

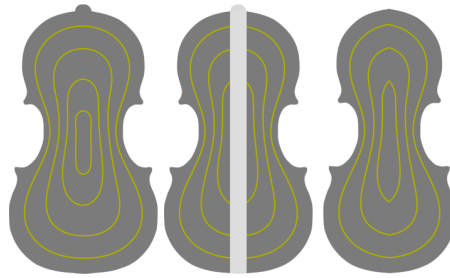


Figure 1: Removal of a longitudinal slice of wood in order to reduce width of instrument [BCG25]

From a musicological perspective, knowing whether an instrument has undergone such reduction is an important question as it affects historical interpretation, conservation strategies, and even economic valuation. However, as mentioned in previous work [BCG25], documentary evidence of reductions is often scarce and ambiguous. Traditionally, luthiers and experts rely on visual examination, tactile inspection, and internal observation (for example using endoscopic cameras) to detect traces of reduction. While expert judgment remains invaluable, the complexity of the task makes it lack reproducibility and quantitative grounding.

The digitization of historical instruments through photogrammetry and 3D scanning has opened the possibility of studying their geometry objectively and systematically. High-resolution meshes provide access to fine geometric details with submillimetric precision, enabling analyses that were previously impossible without invasive measurement. In previous work [BCG25], geometric features such as the set of contour lines of the instrument plates have been defined and studied. These lines and

geometric features seemed to suggest that there exists some consistent quantitative differences linked to the mechanical intervention of removing wood material along the central axis. In particular, contour lines of non-reduced instruments tend to exhibit smoother and more rounded "U-shaped" profiles, whereas reduced instruments often display sharper and more angular "V-shaped" profiles. For that reason, and instead of relying on handcrafted thresholds, previous work [BCG25] has tried to train and use a supervised classifier to detect patterns in the geometric profiles of instruments using reduction labels previously assessed by experts.

But, although these geometric differences were promising, several limitations and challenges still arise: (i) instruments are not uniform, they may contain different numbers of usable geometric descriptors extracted from the contour lines, and (ii) there exists a whole set of more or less naturally deformed violins with unreliable contour fits due to wood deformation, or other artifacts.

Problem statement and objectives

The work presented in this thesis takes place within a broader, long-term research effort whose objective is to develop a quantitative and reproducible tool for identifying violin reduction from geometric data. The specific contribution of this thesis is more focused. It consists of exploring and evaluating some common machine learning classification approaches on the available corpus. We consider here a corpus of historical bowed instruments obtained through photogrammetry, whose submillimeter accuracy has been previously demonstrated and analyzed in [Beg21] and [Beg+23]. Each instrument is associated with a label made by the visual inspection of an expert indicating whether it is considered reduced or non-reduced. Several characteristics on this available data make the evaluation non-trivial.

Limited sample size. The amount of digitized and labeled instruments is extremely small due to practical constraints. As a result, this thesis also operates with very few data. This constraint requires to be very cautious with conclusions, and also directly impacts model selection, hyperparameter tuning, and performance evaluation.

Class imbalance. The corpus is imbalanced: non-reduced instruments are about four times more present than reduced ones. In such conditions, standard accuracy can be misleading, since a classifier biased toward the majority class may achieve a seemingly good score while performing poorly on the minority class. This motivated the use of evaluation metrics that account for imbalance, such as balanced accuracy, and the use of training strategies that tries to mitigate bias.

Uncertainty in labels. Although the labels used in this study are based on expert inspection, some of them are nevertheless ambiguous. The manual classification of instruments as reduced or non-reduced may sometimes lead to uncertain or borderline cases. Some instruments are not clearly attributable to one category, resulting in some uncertainty in their labeling. This limitation may introduce potential noise in the dataset, making the whole training process and evaluation of the models even more challenging.

Given these constraints, the objective of this thesis is therefore to try and evaluate different methodological choices and approaches within this specific dataset and

context. More specifically, here are some concrete questions for which this work will suggest an answer: How do kernel-based methods such as Support Vector Machines (SVM) and Multiple Kernel Learning (MKL) perform when applied to contour-derived geometric features? Can convolutional neural networks (CNNs) effectively exploit elevation maps despite the very limited number of available training samples and other constraints? Which approach provides the best performance, stability, and interpretability? How stable are these performances across different hyperparameter configurations and seeds? Is it possible, in this context, to improve generalization and performance through additional techniques such as data augmentation, or data resampling?

Structure of this thesis

The thesis is organized into the five following chapters.

Chapter 1: Related works and literature review. First, it presents the necessary background concepts in machine learning, including validation strategies, evaluation metrics for imbalanced datasets, and fundamental classification models. Second, it summarizes previous work made on this violin geometric data. Third, it goes through more specific state of the art papers and techniques that will be useful for this paper.

Chapter 2: Dataset. It describes the dataset and the different data representation available and used in this thesis. It also discusses the challenges that arise from such small sample size and class imbalance.

Chapter 3: Multiple Kernel Learning (MKL) approach. It presents the first methodological approach of this paper, based on kernel methods. In particular, it introduces multiple kernel learning (MKL), as an extension of previous SVM attempts [BCG25], for combining heterogeneous feature groups. The motivations, architecture, experiments, and obtained results are discussed.

Chapter 4: Convolutional Neural Network (CNN) approach. It goes through the second methodological approach and main focus of this paper, based on convolutional neural networks. It describes the motivations for the use of CNNs, the architecture and training strategy, and discusses the results obtainable with the limited size of the corpus.

Chapter 5: Multi-Layer Perceptron (MLP) approach. It presents the third methodological approach of this paper based on multi-layer perceptrons (MLP). This approach is introduced as a simpler baseline model. It outlines the motivation behind its use and the associated training strategy. This chapter serves more as an initial exploration for potential future work, providing a foundation for further investigations using fully connected neural networks.

Chapter 1

Related works and literature review

1.1 Background concepts

This section introduces the main theoretical concepts required to understand the methodological choices made later in this thesis. These are the essential notions related to geometric data representations (elevation maps and contour lines), Support Vector Machines, Convolutional Neural Networks, validation strategies, and evaluation metrics adapted to imbalanced datasets.

Elevation maps. The geometric information used in this thesis originates from high-resolution three-dimensional meshes obtained through photogrammetry. While such meshes provide a detailed description of the instrument surface, they are irregular structures which are not directly suited for many classical machine learning techniques. To obtain a structured representation, the meshes were interpolated onto a regular two-dimensional grid defined on a reference plane of symmetry. Figure 1.1, from [BCG24], shows planes of the sound board and back before (top) and after (bottom) the plane of symmetry has been matched with the plane $z = 0$. For each grid point, the height of the instrument surface is measured along the direction orthogonal to the plane of symmetry. This procedure produces an elevation map (or height map).

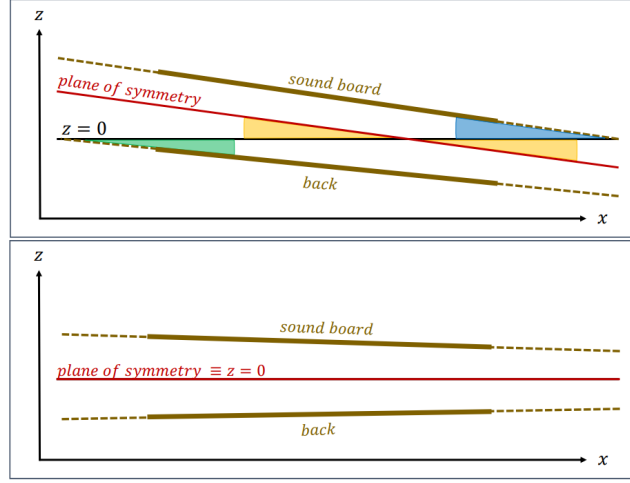


Figure 1.1: Planes of the sound board and back before (top) and after (bottom) the plane of symmetry has been matched with the plane $z = 0$, from [BCG24].

Contour lines. Contour lines correspond to the intersection between the surface of the instrument plate and horizontal planes parallel to the reference plane, similarly to what is used in topographic maps. Figure 1.2 shows contour lines of an instrument before (left) and after (right) the slicing of a longitudinal slice of wood.

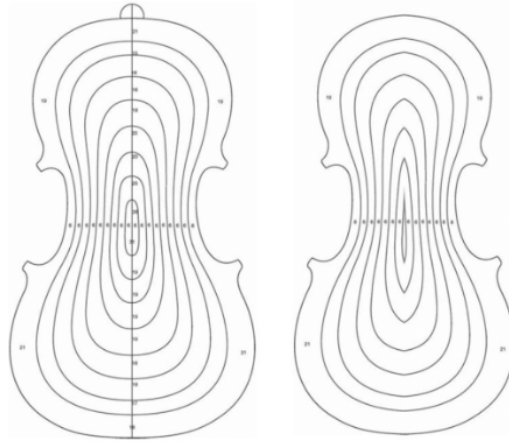


Figure 1.2: Contour lines of an instrument before (left) and after (right) the slicing of a longitudinal slice of wood, from [Beg21].

Support vector machines (SVM). SVM are among the most common kernel-based classifiers. In our binary setting, it seeks a decision function that maximizes the margin between the two classes while penalizing classification errors. Since perfectly linearly separable cases rarely hold in practice, SVM also uses a regularization parameter C , which can be seen as a trade-off between margin maximization and misclassification tolerance. Figure 1.3 [Alm] illustrates the effect of different values of parameter C (respectively, 100, 10, and 1).

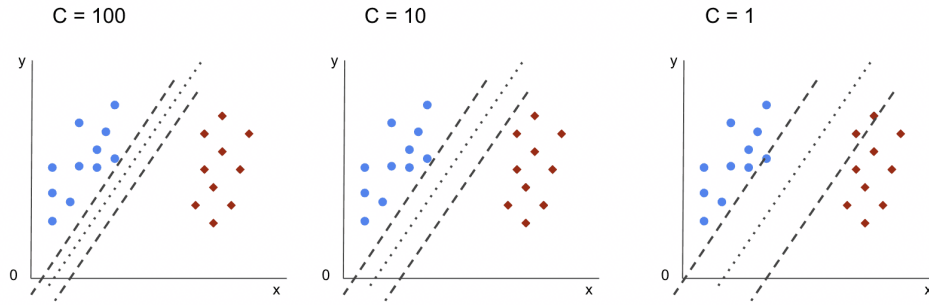


Figure 1.3: Example effect of different values of parameter C , from [Alm].

Convolutional Neural Networks (CNN). CNNs are a class of deep learning models commonly used to process structured grid-like data such as images. In this context, the elevation maps can be represented as two-dimensional arrays, which makes it very suitable for convolutional architectures.

As a reminder, a convolutional layer applies a set of learnable filters that slide over the input and compute local weighted sums. This reflects two important properties: (i) each output unit depends only on a local neighborhood of the input, and (ii) the same filter is applied across the entire spatial domain. Figure 1.4 shows one convolution operation step, turning input image into feature map.

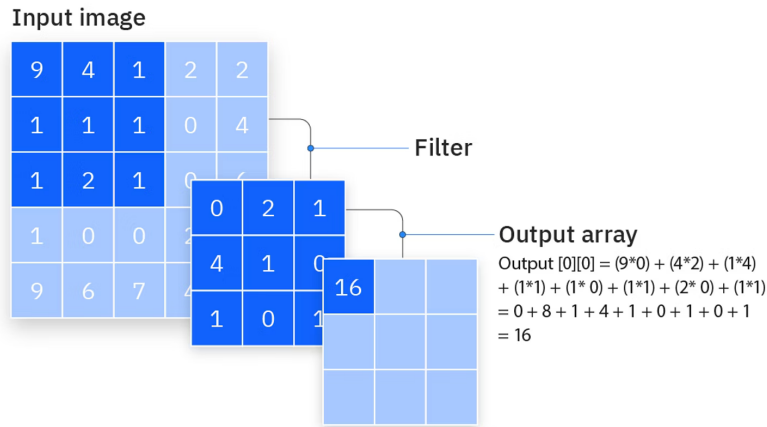


Figure 1.4: One convolution operation step, turning an input image into a feature map, from [Ibm].

Leave-One-Out Cross-Validation (LOOCV). Due to the small size of the dataset, model evaluation is particularly challenging. Creating independent subsets for training, validating and testing in a classical way could easily lead to unstable performance estimates [Vab+19] given the limited number of available samples, and simply not exploiting the most out of the available data.

To avoid as much as possible these issues, it has been decided from the start to use Leave-One-Out Cross-validation (LOOCV). This validation method is an extreme form of k -fold cross-validation in which k equals the number of sample n . In other

words, for each iteration, one sample is used as test instance while the model is trained on the remaining $n - 1$ samples. The procedure is then repeated n times. However, because the model must be retrained n times, it largely increases the computational cost compared to a more standard k-fold cross-validation approach [Lum+24].

Nested CV. In addition to LOOCV, a nested cross-validation strategy is employed to ensure an unbiased evaluation of the model when hyperparameter tuning is involved. Nested CV consists of two loops: an outer loop, used to estimate the generalization performance, and an inner loop, dedicated to hyperparameters tuning. This separation between hyperparameters tuning and evaluation prevents information leakage from the test data into the training process, which could otherwise lead to an optimistic bias in the performance estimates. Nested CV provides a more reliable assessment but further increases the computational cost, as the training procedure is repeated multiple times within each outer iteration. Figure 1.5, from [Bra+23], illustrates this concept.



Figure 1.5: Illustration of nested CV, from [Bra+23].

Stratified CV. Stratified CV is a variation of K-Fold CV, where the dataset is split into K subsets (folds), and the model is trained K times, each time using one fold for validation and the others for training. Stratified K-Fold ensures that each fold is representative of the overall classes distribution, which is very useful in such context where class are highly imbalanced. It addresses this issue by preserving as much as possible the proportion of samples from each class in every fold. This means that each training and validation subset reflects as much as possible the original class distribution of the dataset.

Balanced accuracy. Standard accuracy can be very misleading when class distribution is imbalanced. For example, a classifier that always predicts the majority class, on a dataset where one class represents 80% of the sample, would achieve that same 80% accuracy despite having no discriminative power at all. To deal with this problem and to have a better performance metric, balanced accuracy is often used. It gives as much importance as each class and is mathematically defined as the mean of sensitivity and specificity, where TP, TN, FP and FN denote respectively the true positives, true negatives, false positives and false negatives:

$$\text{Balanced Accuracy} = \frac{1}{2} \left(\frac{\text{TP}}{\text{TP} + \text{FN}} + \frac{\text{TN}}{\text{TN} + \text{FP}} \right).$$

The same naive classifier that always predicts the majority class would correctly classify all majority instances, but misclassify all minority ones. That would therefore give us a balanced accuracy of 50% as the specificity would be equal to 1, but the sensitivity equal to 0. That balanced accuracy score reflects better the absence of discriminative power.

1.2 Previous work

This section reviews the main contributions that formed the foundation of this thesis. All previous works considered here are from the same research project conducted at UCLouvain in collaboration with the Musical Instruments Museum (MIM) in Brussels. It includes (i) the validation of the photogrammetric acquisition protocol, (ii) the geometric analysis of violin contour lines, (iii) the first attempts at quantitative classification based on fitted parameters, and (iv) the documentation of the pre-processing pipeline and derived geometric descriptors.

1.2.1 Data acquisition and quality guarantee

This thesis is grounded in the availability of three-dimensional representations of historical bowed instruments. The transition from qualitative organological observations to quantitative geometric analysis would not be possible without a validated acquisition protocol capable of capturing fine morphological details. This foundational step was addressed in Beghin’s Master thesis (2021) [Beg21] and subsequently formalized in the publication by Beghin et al. (2023) [Beg+23]. These works describe the technical acquisition procedure and also validate the quality of the photogrammetric mesh, concluding that a sub-millimeter accuracy was achieved. This level of accuracy is very important as the geometric effects associated with reduction (such as subtle changes in contour sharpness) are themselves extremely small.

1.2.2 Geometrically comparable objects

The acquisition protocol also introduced a structured workflow for transforming raw meshes into geometrically comparable objects. A key conceptual element is the definition of a plane of symmetry between the sound board and the back. The mesh is then rotated so that this symmetry plane becomes horizontal. Finally, a translation along the vertical axis ensures that the symmetry plane coincides exactly with the plane defined by $z = 0$. As a result of this sequence of rotation and offset, elevation values can then be computed on a regularly spaced grid with a resolution of 0.25 mm. This availability of regular elevation maps opens the possibility of different modeling approaches, such as image-based or convolutional architectures.

1.2.3 Contour lines and parametric function

Another important foundational step that will serve for this thesis, from [BCG24], is (i) the computation of contour lines for each instrument (shown in 1.2), and (ii) the empirical observation that reduced instruments tend to show sharper, more V-shaped contour lines, while non-reduced instruments show rounder, more U-shaped profiles

(shown in Figure 1).

First, contour lines were obtained by drawing successive horizontal planes, parallel to the plane of symmetry, that intersected with the 3D meshes at 1 mm intervals. Figure 1.6 illustrates the plane of symmetry and the successive parallel horizontal planes, from [BCG24].

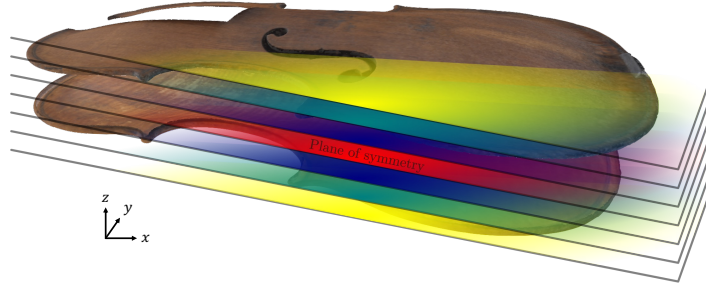


Figure 1.6: Plane of symmetry and the successive parallel horizontal planes, from [BCG24].

After preprocessing, including removal of artefacts, resampling, and filtering, each contour was fitted with the following parametric function:

$$y = \alpha \left| \frac{x - \delta}{\lambda/2} \right|^\beta + \gamma, \quad (1.1)$$

where α controls vertical stretch, β controls opening (sharpness), and γ , δ account for translations. The width λ of each contour normalises the function and can be easily computed. Figure 1.7 illustrates the effect of each of these parameters.

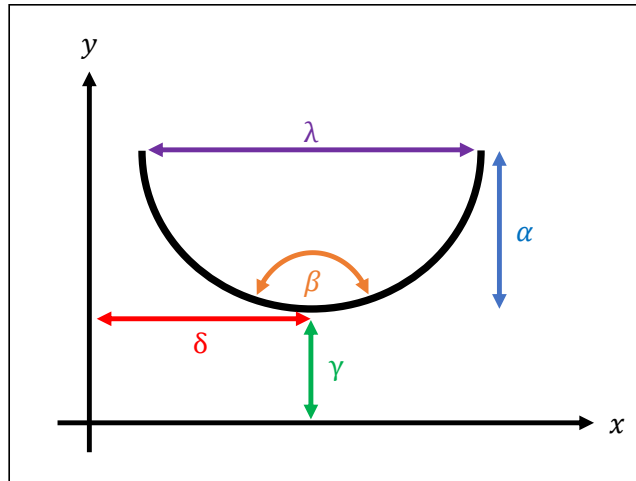


Figure 1.7: Representation of the effect of parameters of the parametric function, from [BCG25].

Each instrument was then characterized by level-wise profiles of the different parameters: α , β , γ , and δ . This work has been done and described in much more details in [BCG25]. Figure 1.8, from the same paper, shows the contour lines fitting of both a non-reduced violin sound board (on the left), and of a reduced violin sound board (on the right).

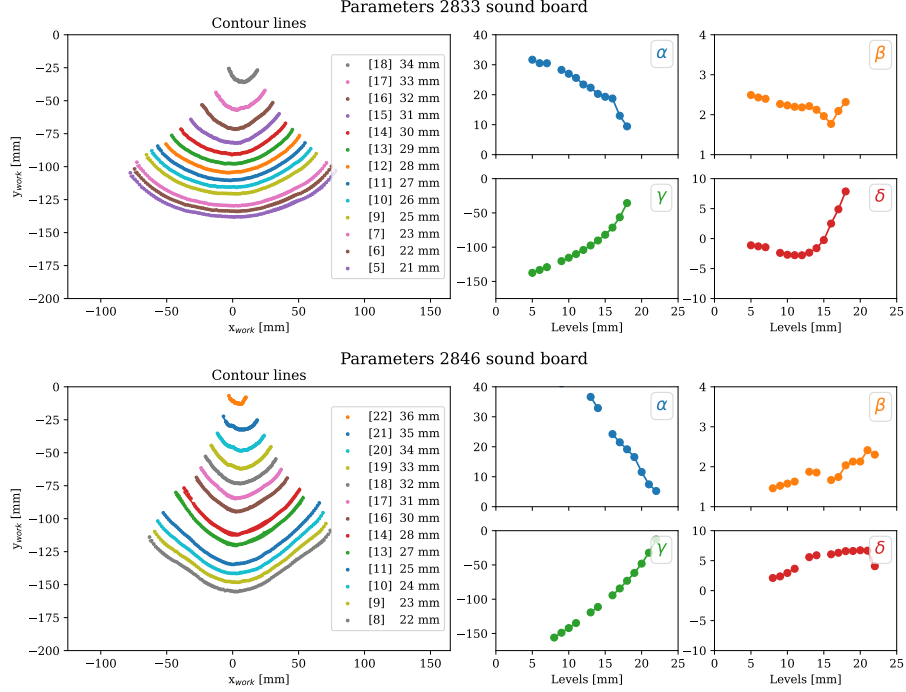


Figure 1.8: Contour lines fitting of an non-reduced violin sound board (top) and of a reduced violin sound board (bottom), from [BCG25].

1.2.4 Standardization of profiles length

One of the challenges identified in previous work was about the heterogeneity of contour levels across different instruments. Those levels varies from one instrument to another, mainly because instruments differ in height to begin with, but also because some levels have been discarded due to poor fitting quality.

To address this challenge, the contour-level parameters were resampled and standardized by mapping the first and last valid levels to the edges of the interval $[0, 1]$ and interpolating intermediate values. During the first classification attempt, samples of 50 points equally spaced per parameter had been arbitrarily chosen.

1.2.5 First classification attempt and feature evaluation

The paper [BCG25] also delivered the first attempt using machine learning. The dataset is composed of 25 instruments (20 non-reduced, 5 reduced). Given the small dataset size and class imbalance, the authors adopted a leave-one-out cross-validation (LOOCV) strategy. They also decided to use the well-known SVM classification

method as classifier.

Different configurations were tested: linear or radial basis function (RBF) kernels, the regularization hyperparameter C values going from 10^{-4} to 10^4 , and different combinations of number and types of input features. Each instrument was represented by four resampled vectors of 50 values, leading to a total of 200 input values when all parameters (α, β, γ and δ) were concatenated. Note that several others derived features based on the β profiles were also explored, including linear, quadratic, and piecewise-linear fittings, as well as threshold-based statistics on β values. Figure 1.9 shows the performances obtained with different C tuning strategies for both linear and RBF kernels, from the paper [BCG25].

Number and type of features	Balanced accuracy (in %)					
	Random C		max C (weak reg.)		min C (strong reg.)	
	RBF	Lin.	RBF	Lin.	RBF	Lin.
2 (linear fit.)	41.3	91.3	0	100	50	100
2 (second slope + number of $\beta \leq 2$)	75.4	75.2	50	90	100	90
2 (second slope + proportion of $\beta \leq 2$)	73.9	77.7	50	75	100	100
3 (quadratic fit.)	68.2	60.5	40	50	90	70
3 (number of $\beta \leq 2, \beta \leq 3$ and $\beta > 3$)	59.4	68.2	40	90	42.5	42.5
3 (proportion of $\beta \leq 2, \beta \leq 3$ and $\beta > 3$)	49.6	65.5	20	40	70	90
3 (second slope + number of $\beta \leq 2$ and $\beta \leq 3$)	69.9	63.4	40	70	90	70
3 (second slope + proportion of $\beta \leq 2$ and $\beta \leq 3$)	75.1	80.9	50	75	100	100
4 (second slope + number of $\beta \leq 2, \beta \leq 3$ and $\beta > 3$)	62.1	60.3	40	70	62.5	32.5
4 (second slope + proportion of $\beta \leq 2, \beta \leq 3$ and $\beta > 3$)	77	81.7	50	75	100	100
5 (linear fit. + number of $\beta \leq 2, \beta \leq 3$ and $\beta > 3$)	63.8	69.9	30	90	77.5	52.5
5 (linear fit. + proportion of $\beta \leq 2, \beta \leq 3$ and $\beta > 3$)	39.6	94.6	0	100	50	100
6 (quadratic fit. + number of $\beta \leq 2, \beta \leq 3$ and $\beta > 3$)	72.2	63.4	40	72.5	90	45
6 (quadratic fit. + proportion of $\beta \leq 2, \beta \leq 3$ and $\beta > 3$)	67.8	69.6	40	62.5	90	80
5 (piecewise linear fit.)	80.2	91.2	50	100	100	100
8 (piecewise linear fit. + number of $\beta \leq 2, \beta \leq 3$ and $\beta > 3$)	60.2	62.2	40	67.5	45	45
8 (piecewise linear fit. + proportion of $\beta \leq 2, \beta \leq 3$ and $\beta > 3$)	76.8	95	50	100	100	100
8 (piecewise lin. + quadratic fit.)	73.3	94.6	40	100	90	100
13 (lin. fit. + piecewise lin. fit. + quad. fit. + number of β)	64.4	71.8	40	80	90	70
13 (lin. fit. + piecewise lin. fit. + quad. fit. + proportion of β)	64.4	82.5	30	90	80	90
50 (resampled α profile)	75.8	100	50	100	100	100
50 (resampled β profile)	74.6	100	40	100	90	100
50 (resampled γ profile)	58.6	53.4	30	65	80	44.5
50 (resampled δ profile)	57.9	80	20	80	70	80
200 (resampled α, β, γ and δ profiles)	72.6	92.5	40	92.5	90	92.5

Figure 1.9: Performances obtained on two models with respectively RBF and linear kernel, from [BCG25].

1.2.6 Discussions and remaining challenges

These first promising performances were one of the motivations for the present work. Even though there are discussions and investigations about a potential lack of stability and/or potential overfit, they may still suggest that geometric features extracted from contour lines, and therefore the data obtained through the photogrammetry process, contain sufficient information to discriminate between reduced and non-reduced instruments. However, several important limitations were also identified in these previous papers and some open questions remained.

First, as reported in [BCG25], the classification results obtained with both kernel types in this first SVM attempt were extremely sensitive to the choice of the regularization parameter C , oscillating between near-perfect classification and complete failure for very small changes in the hyperparameter value.

Second, although it was observed empirically in [BCG25] that (i) the parameter β (controlling the sharpness of the contour) was the most discriminative feature, and (ii) reduced instruments often exhibit increasing β profiles across levels, whereas non-reduced instruments tend to show decreasing or more stable behaviors, no clear geometric threshold was identified that would allow a simple rule-based separation between all reduced and non-reduced instruments. A whole spectrum of intermediate cases exists. Some instruments display moderate deformations due to damage or simply aging, which may look like the geometric effect of reduction without corresponding to an actual historical modification.

Third, the standardization procedure and data representation themselves introduced modeling choices that influence the results. Profiles were resampled to 50 uniformly spaced points on $[0, 1]$, and levels with poor fitting quality (based on MAE thresholds) were discarded. These first classification attempts relied exclusively on derived parametric descriptors (the fitted parameters α , β , γ , δ and engineered features based on these profiles). Although this approach provides good interpretability, especially with linear kernel, it may bias the model through human decisions, compared to the initial, very precised and structured elevation maps. These elevation maps were not fully exploited in this first classification approach.

1.3 State of the Art

While a large body of state-of-the-art research exists in machine learning and computer vision, identifying methods that are directly transferable to this very specific context remains challenging. Many approaches have demonstrated strong performance on standard benchmarks such as natural image classification, but these problems typically involve well-defined visual patterns or semantic differences between classes. In contrast, the present work focuses on distinguishing between instruments with extremely subtle, maybe not perceptible, geometric variations. As a result, conclusions drawn from the literature can hardly be directly applied to this context. Consequently, rather than trying to provide an exhaustive overview of all existing methods, this section focuses on selected approaches that are both theoretically relevant and potentially adaptable to our context. It focuses on the theoretical and algorithmic foundations of the approaches themselves and the results discussed in the different papers. The motivation for selecting these methods, as well as their adaptation and implementation in this specific context, will be discussed in their respective chapters.

1.3.1 Multiple Kernel Learning (MKL)

In order to address complex classification problems involving heterogeneous and structured descriptors, several methods have been developed that extend Support vector machines (SVM), notably by combining several kernels. Among these approaches is Multiple Kernel Learning (MKL). The work of Gönen and Alpaydin [GA11] groups functional forms of the existing MKL algorithms into three basic categories: linear combinations, non-linear combinations, and data-dependent combination. The two latest will not be explained as they were not explored in this thesis.

In the linear combinations version of MKL with weighted sum, MKL seeks a classifier of the form:

$$K = \sum_{m=1}^M \eta_m K_m, \quad \text{with } \eta_m \geq 0,$$

where the coefficients η_m represent the non-negative and constrained ($\sum_m \eta_m = 1$) kernel weights, and K_m represents the available kernels. The learning task then consists not only in optimizing the SVM parameters, but also in learning the kernel weights η_m .

The authors of the paper *Supervised multiple kernel learning approaches for multi-omics data integration* [Bri+24] illustrate its use in a supervised setting for integrating heterogeneous biological data sources. Figure 1.10 shows one of the network architecture studied in [Bri+24].

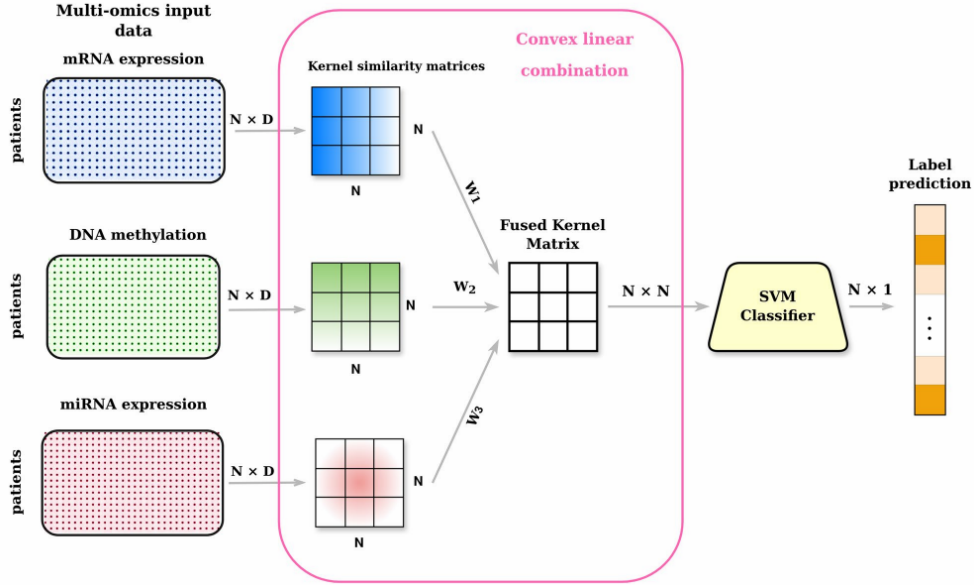


Figure 1.10: One MKL network architecture from [Bri+24]. A SVM algorithm is applied to the convex linear combination of the kernels.

Among the existing MKL approaches, EasyMKL [AD15] provides a scalable and efficient way of learning kernel weights in the linear combination setting. Instead of directly optimizing the weights within a costly joint optimization framework as in a more classical MKL approach, EasyMKL determines the optimal coefficients η_m by solving a min-max optimization problem. This optimization can be written as [AD15]:

$$\min_{\gamma \in \Gamma} \max_{\|\gamma\|=1} (1 - \lambda) \gamma^T Y \left(\sum_{m=1}^M \eta_m K_m \right) Y \gamma + \lambda \|\gamma\|^2, \quad (1.2)$$

where γ is a distribution over training examples. It assigns more importance to pairs of examples from opposite classes that are hard to separate. Given this distribution, the algorithm evaluates how well a given combination of kernels separates these critical examples. The minimization over γ therefore identifies the worst-case

(hardest) configuration of the data, while the maximization over η finds the kernel combination that best handles this difficult scenario.

The parameter λ controls the trade-off between maximizing class separation and ensuring stability in the solution. A higher value of λ leads to a more uniform distribution over the examples, while a lower value focuses more strongly on the hardest samples.

1.3.2 Convolutional Neural Network (CNN)

Convolutional Neural Networks (CNNs) are well-known and studied for visual classification tasks. Their relevance comes from their ability to learn spatial features directly from structured grid-like inputs, instead of relying only on manually designed descriptors. For the past decades a lot of different architectural variations and improvements have been proposed, including deeper networks, residual connections, multi-scale feature extraction, regularization techniques, and so on. Given the vast size of this field, providing an exhaustive state-of-the-art review would require a dedicated study. Therefore, this subsection focuses only on the concepts and approaches that are directly relevant to the context of this thesis.

CNN filter sizes. The literature highlights the importance of using relatively small convolution kernels. Most common filter sizes seem to go from 3×3 to 5×5 , and in their review and discussions of neural networks in computer vision, Zhao et al. reported that using small 3×3 filters is a widely adopted and effective choice that allows the network depth to be increased while reducing the number of parameters compared with larger filters [Zha+24]. This suggests that small kernels might be preferable when working with limited dataset in which over-parameterization must be avoided.

Dropout. CNNs are powerful models, but Zhao et al. also underline that deeper and more complex architectures are more likely to overfit, especially when the amount of training data is limited [Zha+24], hence the importance of regularization. Techniques such as dropout and batch normalization might therefore be used to improve generalization.

The core idea of dropout consists in randomly deactivating some neurons during training. More formally, each unit is retained with a fixed probability p , and removed otherwise. This stochastic process results in training a different sub-network at each iteration. Figure 1.11 [Sri+] illustrates this concept.

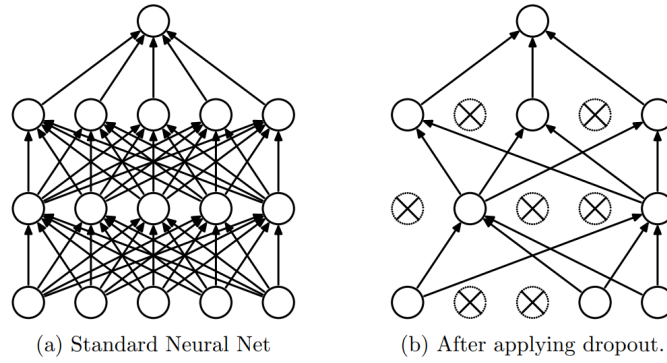


Figure 1.11: Dropout illustration from the paper [Sri+].

This technique prevents networks to rely too heavily on specific neurons, forcing them instead to learn more robust features. As highlighted in [Sri+], this significantly reduces overfitting and improves generalization across a wide range of tasks, including image classification, where substantial performance gains have been reported. For instance, on the MNIST dataset [Kag], the classification error can be reduced from approximately 1.60% with standard neural networks to around 1.25% when using dropout, and even down to 0.95% when combined with additional regularization strategies. Figure 1.12, also from [Sri+], compares the classification test error on several networks of many different architectures with and without dropout strategies.

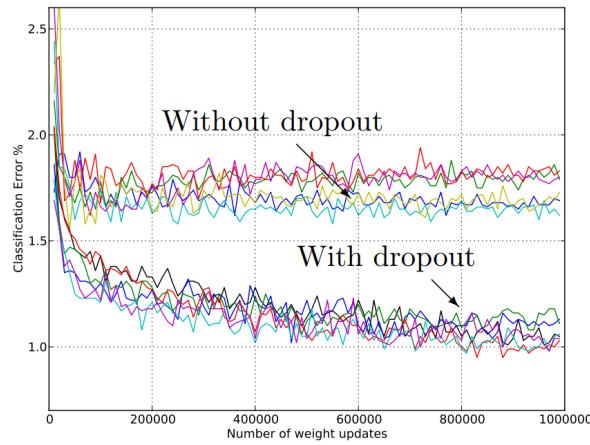


Figure 1.12: Comparison of the classification test errors on several networks of many different architectures with and without dropout strategies.

Batch normalization. It is a technique, introduced by Ioffe and Szegedy [IS15], used to stabilize and accelerate the training of deep neural networks. The general idea is to normalize the activations of each layer during training so that they maintain a stable distribution.

This normalization reduces the so-called *internal covariate shift*, that is the change in the distribution of layer inputs during training as the parameters of previous layers are updated. By keeping these distributions more stable, batch normalization allows

higher learning rates, speeds up convergence, and makes the training process less sensitive to initialization. Ioffe and Szegedy [IS15] reported significant improvements in both training speed and performance. They show that batch normalization enables faster convergence on large-scale image classification tasks, while also acting as a form of regularization that can sometimes complement (or reduce the need for) additional techniques such as dropout. This can be very valuable to optimize stability while working with small datasets.

Pre-trained architectures. Another important aspect of modern CNN based approaches is the use of pre-trained architectures. Many different pretrained models exist and they can be obviously particularly beneficial in contexts where the available dataset is limited. Instead of learning all features from scratch, the network can use knowledge from a previously trained model and only adapt the final layers to the specific task.

Among the commonly used architectures, VGG [SZ15] is characterized by its simple and homogeneous design based on stacked 3×3 convolutional layers, showing that depth alone can significantly improve performance. ResNet [He+15] introduces residual connections that allow very deep networks to be trained effectively by mitigating the vanishing gradient problem, making it one of the most influential architectures in deep learning. DenseNet [Hua+17] further extends this idea by connecting each layer to every other layer in a feed-forward manner, that encourages feature reuse and improving gradient flow while reducing the number of parameters. Finally, EfficientNet-B0 [TL19] proposes an efficient scaling method that has been shown to achieve strong performance with relatively low computational cost.

1.3.3 Cross-validation

A recent comparative study by Lumumba et al. (2024) [Lum+24] provides a detailed analysis of several cross-validation strategies, including Leave-One-Out Cross-Validation (LOOCV), k-fold cross-validation, and repeated k-fold cross-validation. Their results show that, on imbalanced datasets, k-fold cross-validation can achieve high balanced accuracy. With correct hyperparameter tuning and data balancing technique, performance can be even more improved using LOOCV. However, this came at an important computational cost. While more exhaustive validation strategies such as LOOCV can provide more accurate estimates, they are not scalable and may not be practical for complex models. Conversely, k-fold cross-validation seems to offer a good compromise between computational efficiency and reliable performance estimations, which explains its widespread use in practice.

The work of Bradshaw et al. (2023) [Bra+23] emphasizes the central role of cross-validation in preventing overoptimistic performance estimates in machine learning models. They highlight that cross-validation is a fundamental component of modern machine learning pipelines, both for performance estimate and hyperparameter tuning. They also stress that the choice of cross-validation strategy should depend on the dataset size and the objective of the study. For example, LOOCV is only recommended for performance estimate of small datasets, while K-fold (with holdout test set) is recommended for performance estimate on large datasets, hyperparameter tuning, and even algorithm selection. Figure 1.13 shows the CV recommendations depending on the objectives and the dataset from [Bra+23].

Recommendations for Using Different Cross-Validation Methods					
Method	Performance Evaluation	Hyperparameter Tuning	Algorithm Selection	Computational Cost and Time Complexity	Recommended Use
Holdout testing					
One-time train-test split	Yes	No*	No†	Low: $O(n)$	Large datasets; hyperparameter tuning and algorithm selection are not needed
One-time train-validation-test split	Yes	Yes	Yes	Low: $O(na)$	Large datasets; when hyperparameter tuning or algorithm selection is needed
K-fold, holdout test set	Yes	Yes	Yes	Medium: $O(nak)$	Large datasets; when hyperparameter tuning or algorithm selection is needed
Comprehensive testing					
K-fold, folded test set	Yes	No	No	Medium: $O(nk)$	Small or large datasets; when hyperparameter tuning and algorithm selection are not needed
Nested	Yes	Yes	Yes	High: $O(nakj)$	Small datasets; when algorithm selection is needed; small or lightweight models
Leave-one-out	Yes	No	No	High: $O(n^2)$	Small datasets; when hyperparameter tuning and algorithm selection are not needed; small or lightweight models
Select-shuffle-test	Yes	Yes	Yes	Medium: $O(nak)$	Small or large datasets; when algorithm selection is needed
Leave-one-center-out	Yes	No	No	Medium: $O(nc)$	Multicenter datasets
Random sample testing					
Random sampling	Yes	Yes	Yes	Medium: $O(nas)$	Small or large datasets; with or without hyperparameter tuning and algorithm selection
Bootstrap	Yes	No	No	Medium: $O(ns)$	Small or large datasets, when hyperparameter tuning and algorithm selection are not needed
Note.—a = number of algorithms or hyperparameter sets, c = centers, j = folds (inner), k = folds (outer), n = number of samples, O = order of, s = sessions or iterations. * Model hyperparameters must be preselected and remain fixed. † Only a single algorithm should be assessed.					

Figure 1.13: CV recommendations from [Bra+23].

Chapter 2

Dataset

This chapter describes the dataset used in this thesis and the different types of geometric data available. As the acquisition procedure itself has already been discussed in Chapter 2, this chapter rather explains how the different representations are structured, and how they were used throughout this thesis.

2.1 Data types/organization

The dataset is structured as a sequence of transformations applied to the original three-dimensional meshes. Each step of this transformation pipeline produced new data files describing different geometric aspects of each of the 25 instruments.

Original meshes and preprocessing. The starting point of the dataset consists of photogrammetric meshes representing the full instrument geometry. These meshes therefore correspond to the complete instruments at real scale and without any specific orientation. These raw meshes were not really suitable for machine learning as they were not aligned to any reference frame. They were therefore processed to isolate the two main plates of the instrument: the sound board, and the back. Since the instrument meshes had initially arbitrary orientations, a geometric alignment procedure was applied to make the dataset comparable across instruments. This procedure determines the plane of symmetry of the instrument based on the contours of the sound board and the back plate. The meshes were then rotated so that this symmetry plane becomes horizontal and aligned with the coordinate system. As mentioned earlier in Section 2.1, the vertical offset of these meshes were also adjusted in order to correspond exactly to the plane $z = 0$.

As a reminder, Figure 1.1 shows planes of the sound board and back before (top) and after (bottom) the plane of symmetry has been matched with the plane $z = 0$.

Elevation maps. An inconvenient still existed: vertices were distributed unevenly across the surface of the meshes. For a large amount of computational methods it is more convenient to work with data defined on a regular grid. To address this inconvenient, the meshes were interpolated on a two-dimensional grid defined on the symmetry plane. The height of the plate above this plane was then computed for each grid point. The grid spacing was fixed to 0.25mm.

At this point, each file could therefore be interpreted as a two-dimensional matrix representing the elevation of the plate (sound board or back) over a regular grid. Notably, this representation is particularly suitable for image-like processing methods such as the convolutional neural network approach of this thesis.

Contour lines and parameter fitting. Another type of representation is obtained by analysing the contour lines of the plates. As described earlier in Subsection 2.2.3, these contour lines correspond to the intersection between the plate mesh and horizontal planes spaced every millimetre along the vertical axis, and for each of these lines a parametric curve was fitted using a least-squares approach. As a reminder, four parameters were used: α , β , γ , and δ , respectively controlling the vertical scaling, the opening, and the horizontal and vertical translations of the curve.

Because the number of contour levels varies from one instrument to another, the resulting parameter profiles were resampled to obtain a fixed-length representation. Arbitrarily, each parameter profile was interpolated using 50 evenly spaced points on the interval $[0, 1]$.

Data used in this thesis. Although many types of geometric descriptors are available in the dataset, this thesis used two main type of data representations. First, for the kernel-based classification approach (MKL) and for the neural network (MLP), the input features consisted of the resampled parameter profiles extracted from the contour lines. Second, for the convolutional neural network approach, the input data consists of elevation maps of the sound board and back.

2.2 Classes (imbalanced)

The classification task addressed in this thesis consists in distinguishing between two categories of historical instruments: reduced (R) and non-reduced (NR) instruments. As a reminder, in this context, a reduction refers to a physical modification applied to an instrument after its initial construction.

The labels used in this work originated from prior expert analyses. Based on these expert assessments, each instrument of the corpus was assigned to one of the two classes. It is important to keep in mind that these manual classifications were in some case ambiguous, and there is no guarantee that all instruments are correctly classified in the first place.

The dataset used in this thesis contains a total of 25 instruments. Among them, 20 instruments are considered non-reduced, while 5 instruments are considered as reduced. The distribution of the classes is therefore highly imbalanced, with 80% of the instruments belonging to the non-reduced category and only 20% belonging to the reduced one. As mentioned before, such an imbalance raises important methodological challenges for the machine learning approaches used in this thesis. In this work, we therefore rely primarily on balanced accuracy rather than standard accuracy.

Chapter 3

Multiple Kernel Learning (MKL) approach

This chapter presents the first approach explored in this work, namely Multiple Kernel Learning (MKL), which can be seen as an extension of the SVM-based method previously explored in [BCG25]. This chapter starts by presenting the motivations for such approach. It then presents the architecture and technical choices that were made in our specific context, and end up with the experimental results, analysis, and a discussion on these results.

3.1 Motivations

As discussed in Chapter 2, previous experiments based on Support Vector Machines (SVM) had already been made before this thesis. Those experiments suggested that the geometry of contour lines could probably contain meaningful information for distinguishing between reduced and non-reduced instruments. In particular, parameters such as the opening β and the vertical stretch α showed great potential at capturing differences between the two categories [BCG25].

However the previous SVM approach also revealed several limitations. First, the classification performance strongly depended on the choice of the kernel and the regularization parameter, notably C . Slight variations of this parameter could lead to large and unpredictable changes in the classification results. Second, as different combinations of parameters extracted from the contour fitting process had been tested, it seemed that some of them contributed very poorly to the classification task. While β and α seemed to carry significant information, the relevance of other parameters such as δ or γ is more ambiguous. These parameters might seem less informative individually, but they could still contain complementary information, or information otherwise missing by more dominant features.

These limitations motivated this approach. Multiple Kernel Learning allows several kernels, which in our context correspond to each parameter, to be defined and then determines how to combine these kernels in order to maximize the performance of the classifier. This approach allows to use the different features of data with different weight, and can also provide insight into which geometric descriptors are indeed most relevant for the classification task.

3.2 Algorithm and architecture

As the theoretical foundations of MKL have already been introduced in subsection 2.3.1, here the focus will be on the specific architectural and algorithmic choices that were made in order to adapt the methodology described in [AD15] to our specific context and data. All implementations of this paper are written in Python and the core computation relies on sklearn as well as a dedicated MKL library called MKLpy.

3.2.1 Feature organization and kernel definition

The MKL framework requires the definition of several kernels, each corresponding to a specific representation of the data. In our case, the available descriptors originate from the parametric fitting of contour lines described in Chapter 3. As a quick reminder, for each instrument and each contour level, four parameters were obtained: α , β , γ , and δ . The number of levels being different from one instrument to another, the parameter profiles were resampled to a fixed length representation of 50 points.

Moreover, the contour lines were extracted from different regions of the instruments themselves. In the dataset used, three geometric zones were considered: the sound board (denoted *sb*), the lower part of the back plate (*back bottom*), and the upper part of the back plate (*back top*). Figure 3.1 illustrates these three geometric zones for the same example instrument.

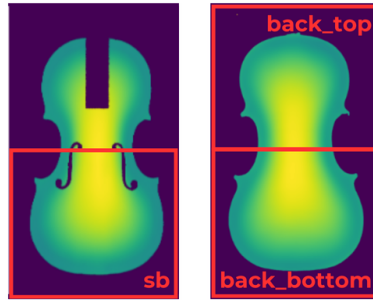


Figure 3.1: Illustration of the three geometric zones considered

Each zone is therefore associated with four parameter profiles corresponding to α , β , γ , and δ . Combining these elements lead to twelve distinct feature sets per instrument:

$$3 \text{ zones} \times 4 \text{ parameters} = 12 \text{ feature sets.}$$

In practice, each feature set is treated independently and is associated to a separate kernel. For a given feature group (for instance, the α parameter of the *back bottom* zone), the similarity between two instruments x_i and x_j is computed through a kernel function $K_m(x_i, x_j)$.

It has been decided to limit the experiments to two very common kernel types: the linear kernel and the radial basis function (RBF) kernel. This choice was motivated by (i) remaining aligned with the initial motivations, (ii) ensuring consistency and meaningful comparison with the SVM approach explored in previous work, and (iii) avoiding the introduction of unnecessary model complexity.

3.2.2 Construction of kernel blocks

The computation of the kernel matrices is implemented using the build-in functions `linear_kernel()` and `RBF_kernel()` from the `sklearn.metrics.pairwise` package. These functions are used with each feature set to create one distinct kernel matrix.

Each of these kernel matrices are then normalized to ensure numerical stability and fair comparison across the different feature sets. The same normalization factors computed from the training set are subsequently applied to the test kernel matrix to stay consistent between training and testing representations. The pseudo-code of this straightforward kernel blocks construction is presented by Algorithm 1.

Algorithm 1 Construction of EasyMKL kernel blocks

Data: Feature dictionary, training indices, combinations, test index, kernel

Result: Lists of training and test kernel blocks

```

foreach (zone, parameter) in combinations do
    Extract training and test samples for this combination
    if kernel is linear then
        | Compute training and test kernels using linear_kernel()
    else
        | Compute training and test kernels using RBF_kernel()
    Compute normalization factors from the diagonal of the training kernel
    Normalize the training and test kernels using these factors
    Append the training and test kernels to their corresponding lists

```

3.2.3 EasyMKL algorithm

Once the kernel blocks are constructed, the learning phase is performed using the EasyMKL algorithm introduced by Aioli and Donini [AD15] and presented in Section 2.3.1. EasyMKL is implemented in the MKLpy library. It takes, as input, the set of kernel matrices created with our kernel construction function just described and tries to find weights $\eta_1, \dots, \eta_m$ defining the combined kernel:

$$K = \sum_{m=1}^M \eta_m K_m, \quad \text{with } \eta_m \geq 0, \quad (3.1)$$

The coefficients η_m are constrained to be non-negative and normalized so that their sum equals one. In practice, EasyMKL optimizes these weights in order to maximize the margin of the classifier constructed in the feature space.

In the current implementation of this thesis, EasyMKL is used with a SVM base classifier. More specifically, the base learner provided to EasyMKL is an instance of the SVC class from the `sklearn.svm` library. This classifier operates directly on the precomputed kernel matrices and implements the same classical SVM optimization procedure described in Section 2.1.

While we still have the parameter C of the SVM that controls the trade-off between margin maximization and classification errors, EasyMKL introduces a new parameter λ that controls the regularization of the kernel combination itself. This parameter influences how the algorithm distributes the weights across the different kernels. Smaller

values encourage a more selective combination of kernels, while larger values tend to distribute weights more evenly.

3.2.4 LOOCV evaluation

Given the limited number of instruments available in the dataset, model evaluation relies on nested LOOCVs. As explained earlier in Chapter 2, this validation strategy consists of combining two cross-validation loops. For each (outer) fold, the training subset, of $n - 1$ instrument data, enters a second leave-one-out loop used to select the best values of the hyperparameters. Two parameters, the SVM regularization parameter C and the EasyMKL regularization parameter λ , are tuned during this process. A third parameter, γ is also tuned when using RBF kernels. It should be noted that the parameters λ and γ here refer respectively to the EasyMKL regularization coefficient and the RBF kernel parameter, and are unrelated to the contour line parameters introduced previously.

For each candidate combinations of parameter values (C , λ , and potentially γ), the inner cross-validation evaluates the balanced accuracy obtained across the validation instruments. The pair of hyperparameters that achieves the highest validation balanced accuracy is then selected. Once these optimal hyperparameters have been determined, the model is retrained on the full outer training set using these parameter values, and makes a prediction for the held-out test instrument. This procedure is repeated for each instrument in the dataset, and the predictions are evaluated. This methodology is kept very similar to the one used in previous work, described in section 2.2.5. Figure 3.2 illustrates this nested CV methodology, from [Xin+18] and Algorithm 2 presents a slightly simplified pseudo-code for clarity of the actual implementation.

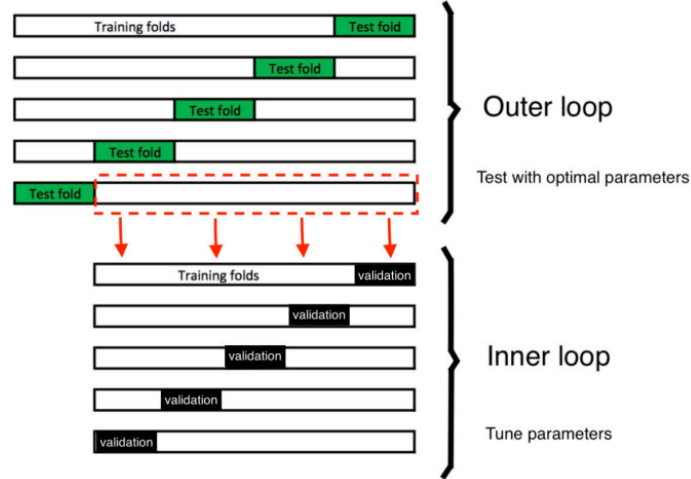


Figure 3.2: Illustration of the nested CV methodology, from [Xin+18]

Algorithm 2 Nested LOOCV with EasyMKL

Data: Feature parameters, hyperparameter grids, kernel

Result: Predictions, selected hyperparameters, performance metrics

Load features for all instruments

foreach *test instrument* **do**

 Define all remaining instruments as training set

 Select best hyperparameters (C , λ , γ) using inner LOOCV

 Build outer training and test kernel blocks

 Train EasyMKL model with selected hyperparameters

└ Predict label for test instrument

Compute performances and metrics

3.2.5 Extraction of kernel weights

One of the motivations for exploring such MKL approach was that it allows the model to assign different importance to the various feature groups. After training, the EasyMKL model contains a set of weights associated with each kernel. These weights provide an indication of how much each geometric descriptor has contributed to the final decision function. Once easily retrieved, thanks to the MKLpy library, the weights are normalized to ensure fair comparisons between different kernels and provides a direct interpretation of their relative contributions.

3.3 Experiments and results

This section presents the experiments and the results obtained with the MKL approach just described. The objectives were to report: the best score obtained, the behaviour and stability of the model across different hyperparameter settings, C , λ , and γ , the robustness of the results obtained, and furthermore what information can be interpreted from the learned kernel weights.

3.3.1 First experiment overview

As the first experiments, all twelve kernels (made of the 4 resampled parameters for each 3 instrument zones) were used. The ranges of values tested for each parameters are shown in Table 3.1.

Parameters	Ranges
MKL regularization λ	[0.0, 0.1, 0.2, 0.5, 0.8, 1.0]
regularization C	[10^{-4} , 10^{-3} , ..., 10^3 , 10^4]
RBF γ parameter	[10^{-4} , 10^{-3} , ..., 10^3 , 10^4]

Table 3.1: Parameter ranges used for the first MKL experiments

During the tuning of the regularization parameter C , in the case of ties, the two same selection strategies from [BCG25] were considered: either choosing the smallest C value (corresponding to the strongest regularization) or the largest C value (corresponding to the weakest regularization) among the tied candidates. Both linear and RBF kernels were tested which therefore resulted in four distinct experimental configurations, combining both types of kernel functions as MKL base learner and

both tie-breaking strategies for the parameter C . The results obtained from these first experiments, and the average computational times are reported in Table 3.2.

Kernel	min C (BA, %)	max C (BA, %)	Avg. Time (s)
Linear	70	100	413
RBF	80	100	1565

Table 3.2: Performances of first experiment, with both kernel base and both C tie-selection strategy

A clear distinction can be observed between the two tie-breaking strategies for the parameter C . Selecting the largest value of ties consistently leads to higher performances, reaching perfect classification (100% BA) for both kernel types, while choosing the smallest value of ties results in noticeably lower balanced accuracy (70% for linear and 80% for RBF). That may suggests that, in this context, less regularized models (higher C) are better able to capture the discriminative structure of the data.

A significant difference in computational time can also be observed between the two kernel types. The RBF-based configurations require substantially more time than the linear. This can be simply explained by the fact that this experiment used a full grid search, testing every possible combination, so additional hyperparameter γ involved in the RBF kernel created much more combinations in the hyperparameter search space.

3.3.2 Model parameters

Role of C and λ in the linear MKL setting

For these next experiments the values of C and λ have been iteratively fixed to be able to compare the evolution of the performance. Figure 3.3 shows the evolution of the balanced accuracy obtained with the linear MKL model for each tested value of C and λ . Each curve corresponds to a different λ value (the black curve represents the median of all curves), and the x-axis represents the value of C .

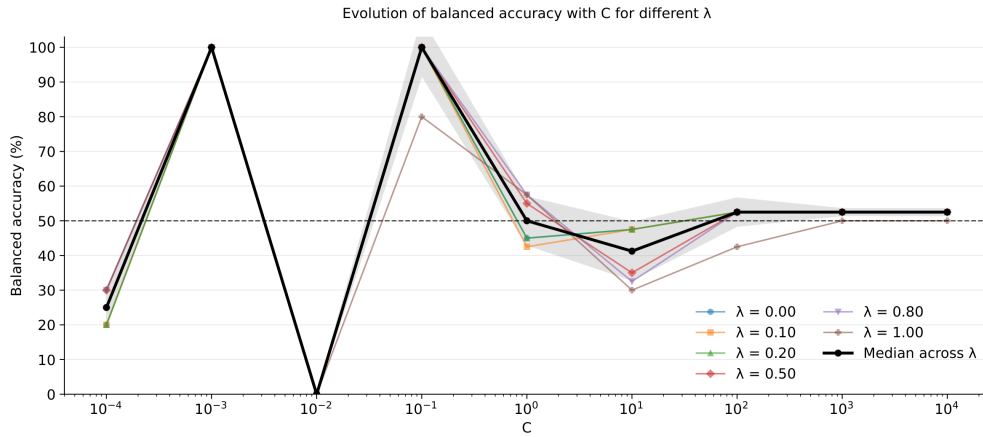


Figure 3.3: Evolution of balanced accuracy with C for different λ

The most striking aspect of this graph is that the overall shape of the curves is almost entirely driven by C , while λ only introduces very small variations. The model

seems to be capable of reaching very high BA performances but is entirely dependent on the C parameter value. A balanced accuracy of 100% is obtained for $C = 10^{-3}$ for all tested λ values, and the same perfect score is also obtained for all but one λ values when $C = 10^{-1}$.

This observation is consistent with the interpretation of the EasyMKL λ regularization. As a reminder, small λ values allow the model to concentrate more strongly on a subset of kernels, while large λ values enforce a more uniform distribution of kernel weights. The results suggest that, once the classifier is placed in a good C -regime, several weighting strategies remain viable. Conversely, when C induces a poor decision boundary, no reasonable reallocation of kernel weights through λ is sufficient to rescue the model performances. In that sense, λ is conditional on C : it can very slightly improve or degrade an already favorable or unfavorable solution, but it does not determine the global performance on its own.

In the next experiments, λ was fixed to 0.2 because this value belongs to the group of non-extreme values that systematically achieved the best performances when C is well chosen. With that parameter value being fixed, it is also informative to analyze how predictions themselves evolved according to the value of C . Figure 3.4 shows, for each value of C (y-axis), whether the prediction for each instrument is correct (in green) or incorrect (in red). The x-axis corresponds to the instrument inventory numbers. The first five instruments on the left belong to the reduced class and the other twenty belongs to the non-reduced class.

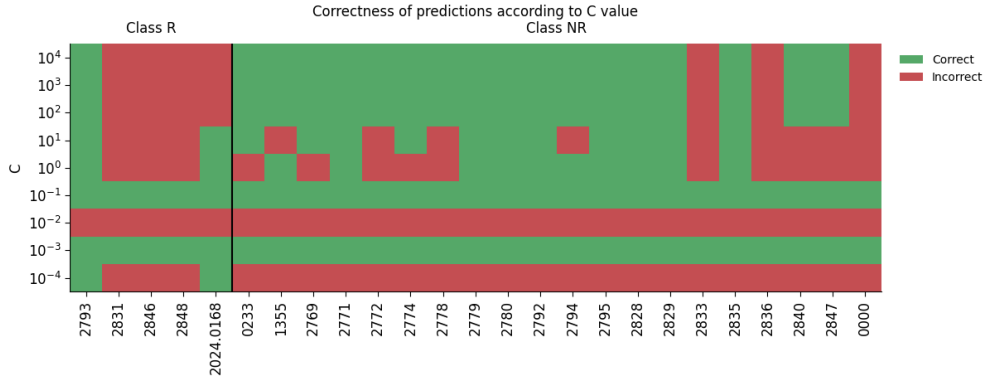


Figure 3.4: Correctness of each instrument prediction, per C value

For $C = 10^{-4}$, only two of the five reduced instruments were correctly identified, while all non-reduced ones were misclassified, resulting in a balanced accuracy of 20%. At $C = 10^{-2}$, the balanced accuracy drops to 0%. This behaviour is particularly surprising, as it indicates that the classifier consistently predicts the wrong class for all instruments rather than behaving randomly. In other words, the decision boundary learned at this value of C appears to be incorrect but very systematic. The fact that neighboring values, such as $C = 10^{-3}$ and $C = 10^{-1}$, gave perfect classification, highlights once again the strong sensitivity of the model to the regularization C parameter.

Role of γ in the RBF MKL setting

A similar experiment and analysis have been done for the RBF version of this MKL approach. In this setting, λ was fixed to 0.2, and the evolution of the performance was analyzed according to both C and the RBF kernel parameter γ . Figure 3.5 shows the balanced accuracy as a function of C (x-axis), with one curve corresponding to each value of γ . The black curve represents the median balanced accuracy across all γ values for each C .

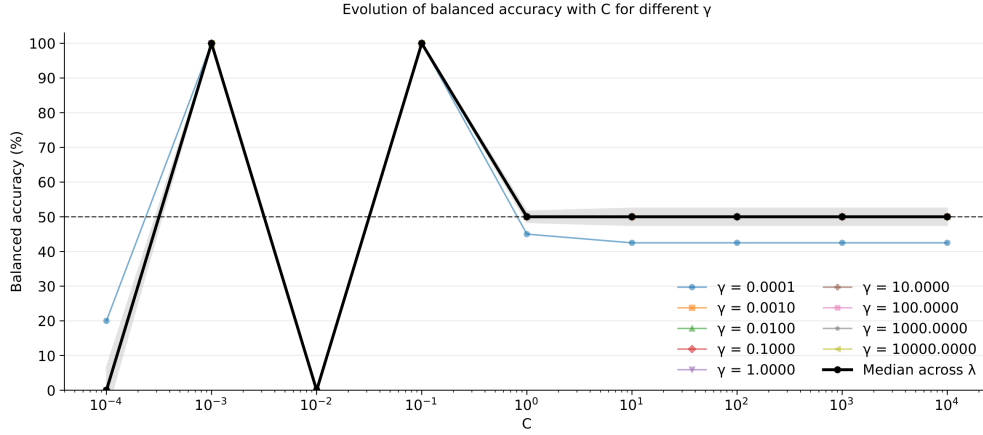


Figure 3.5: Evolution of balanced accuracy with C for different γ

Here, the difference between performances are even less pronounced. Except for the smallest value $\gamma = 10^{-4}$, all the curves are identical. For $\gamma \in \{10^{-3}, 10^{-2}, 10^{-1}, 1, 10, 10^2, 10^3, 10^4\}$, the model yields exactly the same balanced accuracy for every tested C : 100% at $C = 10^{-3}$, 0% at $C = 10^{-2}$, 100% again at $C = 10^{-1}$, and then 50% for $C \geq 1$. The smallest value $\gamma = 10^{-4}$ differs only marginally, mainly by producing slightly lower scores in the higher C region.

This behavior suggests that, in this context, this MKL framework is insensitive to the choice of γ over a wide range of values. Once again, the regularization parameter C remains the dominant factor controlling the model performance, while the impact of γ appears negligible. Similarly to what was done for λ , the value of γ was then fixed to 0.1 for the remaining RBF experiments. This value belongs to the large set of values that produce identical results while remaining commonly used in literature.

With $\lambda = 0.2$ and $\gamma = 0.1$, the evolution of the predictions according to the value of C with RBF as base learner has been recorded. Figure 3.6 shows, for each value of C , whether the prediction for each instrument is correct (in green) or incorrect (in red). The x-axis corresponds to the instrument ids. The first five instruments on the left belong to the reduced class and the other twenty belongs to the non-reduced class.

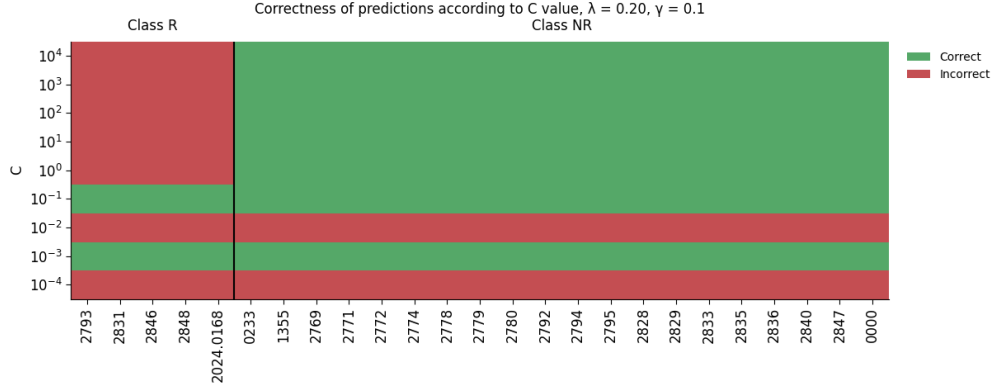


Figure 3.6: Correctness of each instrument prediction, per C value, with RBF as base kernel

The behavior observed here is different from the one observed with the linear kernel SVM as base learner. For values of C ranging from 10^{-4} to 10^{-1} , the classifier alternates between perfect classification and complete failure: $C = 10^{-4}$ gives 100% balanced accuracy, $C = 10^{-3}$ yields 0%, $C = 10^{-2}$ returns to 100%, and $C = 10^{-1}$ again drops to 0%. As previously discussed, this oscillatory behavior is particularly surprising because in such binary classification setting, consistently predicting the wrong class is, in a sense, as difficult as consistently predicting the correct one, which reflects a very systematic decision boundary. For $C \geq 1$, this oscillation disappears and the classifier seems to lose all predictive power, as it always predicts the majority class for all samples, resulting in a constant balanced accuracy of 50%.

Hyperparameter selection and interpretation

The nested LOOCV procedure described earlier was introduced precisely because of this hyperparameter sensitivity. The experiments of Figures 3.3 and 3.5 are useful for understanding the behaviour of the model across the parameter space, but they do not constitute a fair evaluation protocol on their own. Such one-at-a-time approach is useful to see the direct impact of a parameter, but generally speaking not enough to rule out interactions (as it will completely miss more complex, less direct influence that a parameter or combinations of parameters may have).

Nevertheless, the analysis of the hyperparameters leads to a clear conclusion. This MKL approach does not remove the need for careful regularization, especially for the SVM parameter C as it remains by far the most influential. λ has a secondary but clearly weaker direct effect, and γ has almost no practical direct influence in the tested RBF setting. This hierarchy of importance will be useful when interpreting the weight distributions and ablation results below.

3.3.3 Importance of zones and parameters

As mentioned before, this MKL approach offers an important advantage over a simple SVM approach: it explicitly learns a weight for each kernel block. Since each block corresponds here to a specific pair (zone, parameter), the learned weights provide a direct indication of which geometric descriptors are the most used by the model. Looking at the weights alone is not enough to determine the importance of a zone or

parameter, as there may be plenty of weight configurations that are equivalent. This is why, instead, ablations experiments were performed. Here we considered importance as how well the model could compensate for the ablation of a zone or parameter.

First, the kernel weights learned by the full model, with all twelve combinations of $(zone, param)$, were averaged and regrouped by zone and by parameter. Second, ablation experiments were carried out by removing either one entire zone or one entire parameter from the start and retraining the MKL classifier on the remaining block features. The corresponding weight redistributions and balanced accuracies are reported in Tables 3.3 and 3.4, respectively for the linear and RBF based kernels.

Linear MKL weights and ablation experiments

Table 3.3 shows the results of these ablation experiments with linear based kernels. On the left columns are the runs: *Abl.* (X) represents a run with zone or parameter X removed from the set of features. In the middle are the average weights assigned by the EasyMKL package to each zone and parameter kernel across the LOOCV folds, with “RM” indicating the removed zone/parameter. These weights are constrained to sum to 1 and reflect the relative importance given by the MKL model to each kernel when building the final combined decision function. Higher weights therefore indicate that the corresponding zone or parameter contributed more strongly to the classification. On the right column are the performances obtained for each run.

Runs	zones			params				BA (%)
	sb	back_bottom	back_top	alpha	beta	delta	gamma	
Full MKL	0.37	0.40	0.23	0.11	0.44	0.39	0.05	100.00
Abl. (sb)	RM	0.69	0.31	0.13	0.58	0.27	0.03	100.00
Abl. (back_bottom)	0.64	RM	0.36	0.09	0.18	0.67	0.06	100.00
Abl. (back_top)	0.45	0.55	RM	0.09	0.64	0.21	0.07	97.5
Abl. (alpha)	0.37	0.39	0.23	RM	0.47	0.47	0.05	100.00
Abl. (beta)	0.49	0.23	0.28	0.21	RM	0.69	0.10	100.00
Abl. (delta)	0.31	0.60	0.09	0.19	0.71	RM	0.10	80.00
Abl. (gamma)	0.35	0.41	0.24	0.12	0.48	0.40	RM	100.00

Table 3.3: Ablation experiments results for linear base learner

For the full linear MKL model (first line), the weight distribution is not uniform. At the zone level, the back bottom region receives the highest total weight (0.40), slightly ahead of the sound board (0.37), while the back top receives substantially less (0.23). At the parameter level, β receives the largest weight (0.44), followed closely by δ (0.39), whereas α (0.11) and especially γ (0.05) contribute much less.

In accord with what was suggested in [BCG25], β already emerged as the most informative contour descriptor, which was geometrically coherent because it controls the opening of the fitted curve and therefore directly reflects the “U-shape versus V-shape” distinction between non-reduced and reduced instruments mentioned before. It confirms that β remain central even when the model is allowed to combine twelve separate blocks with different weights.

At the same time, the weight assigned to δ (0.39) is noteworthy. On a purely geometric level, δ corresponds to horizontal translation of the fitted contour. One might initially expect such a parameter to be less directly related to reduction than β . Yet, being close to β on average, the MKL weights may indicate that δ carries important information as well. This is plausible because horizontal shifts in the fitted curves may capture asymmetries or systematic deformations associated with the

reduction process, with instrument aging, or with the way geometric alterations propagate across the plate.

By contrast, γ appears largely negligible. Its small weight in the full model and every other run, combined with the fact that removing it does not change the balanced accuracy at all strongly suggest that vertical translation contributes very little discriminative information in the present representation. Removing it keeps the balanced accuracy at 100%, and the redistributed weights remain very close to those of the full model. Removing α also leaves performance unchanged, although it induces a stronger redistribution of weight toward β and δ , which then share the same (and almost all) parameter-level importance. This seems to indicate that α is informative but not necessary. In contrast, removing δ causes the most substantial degradation, with balanced accuracy falling to 80%. This is the only parameter ablation that produces such a clear loss in the linear setting. Hence, although β is the largest single component in the full model, δ appears to be the most difficult parameter to replace or compensate.

This leads to an important nuance. *Most weighted* does not necessarily mean *most indispensable*. The full model assigns the largest share to β , but the ablation results show that δ plays a crucial complementary role. It may be that δ is the only feature that can help separate some ambiguous cases.

At the zone level, the most weighted regions (the sound board or the lower back) seem to be the most dispensable. Removing either of them still preserves 100% balanced accuracy. By contrast, removing the upper back slightly reduces the score to 97.5%. This difference is small, but it suggests that the upper back region is the only one containing at least one useful information that is not perfectly redundant with the other two zones. It is not globally dominant (0.23 on the full MKL run), but may help resolve a small number of difficult folds. On the opposite, both β and α seem to contain important information but are fully redundant with other zones.

RBF MKL weights and ablation experiments

Table 3.4 reports the results of the same experiments conducted this time with RBF base kernels.

Runs	zones			params				BA (%)
	sb	back_bottom	back_top	alpha	beta	delta	gamma	
Full MKL	0.34	0.38	0.28	0.28	0.13	0.24	0.35	100.00
Abl. (sb)	RM	0.55	0.45	0.24	0.10	0.23	0.42	100.00
Abl. (back_bottom)	0.55	RM	0.45	0.31	0.05	0.27	0.37	100.00
Abl. (back_top)	0.47	0.53	RM	0.27	0.17	0.22	0.34	100.00
Abl. (alpha)	0.33	0.39	0.28	RM	0.15	0.33	0.53	100.00
Abl. (beta)	0.36	0.32	0.32	0.32	RM	0.28	0.40	100.00
Abl. (delta)	0.33	0.40	0.26	0.38	0.17	RM	0.46	100.00
Abl. (gamma)	0.35	0.39	0.26	0.47	0.17	0.36	RM	100.00

Table 3.4: Ablation experiments results for RBF base learner

In the full RBF model, the weights are much more evenly spread. At the zone level, the back bottom (0.38) remains slightly ahead, but the three regions are overall closer than in the linear setting. At the parameter level, γ becomes the most weighted descriptor (0.35), followed by α (0.28) and δ (0.24), while β drops to only 0.13.

This reversal should be interpreted with caution. It does not mean that γ is suddenly geometrically more meaningful than β . Rather, it reflects the fact that kernel weights in a non-linear model may be much more redundant. This caution is confirmed by the ablation results: every single RBF ablation reported in Table 3.4 still achieves 100% balanced accuracy. In other words, none of the removed zones or parameters produces any measurable performance degradation. This means that the RBF model is either strongly redundant, or that its perfect score in this experiments and configurations is not discriminative enough to reveal which blocks really matter. In both cases, the interpretation of the weights becomes weaker. A descriptor can receive a high weight without being necessary, and conversely a descriptor with low weight may still be part of a important set of multiple features.

For that reason, the linear MKL results appear more informative than the RBF ones from the perspective of scientific interpretation. The linear model not only performs as well in the best cases, but also provides a more meaningful decomposition of importance across zones and parameters.

3.3.4 Overall results and discussion

Taken together, the previous experiments show that the MKL approach is able to classify reduced and non-reduced instruments with extremely high performance under appropriate settings. In both the linear and RBF versions, several configurations reach 100% balanced accuracy. This suggests that the contour-derived parameter profiles might contain a very strong discriminative signal. In that sense, the MKL approach clearly validates the main working hypothesis of previous work [BCG25]: the geometric transformations induced by reduction seems to leave measurable traces in the contour structure, and these traces can be exploited automatically by a supervised classifier.

However, these results are not uniformly robust across the hyperparameter space. In both kernel families, slightly changing C moves the model from perfect classification to complete failure. Even though the nested LOOCV procedure is used to mitigate the methodological issue by selecting hyperparameters only from training folds, this sensitivity remains important. This observation is in fact consistent with the previous SVM attempt presented in Chapter 2 in which the contour-based representation seemed highly informative, especially through β , but performance was strongly dependent on regularization choices and on the kernel. The present MKL approach does not contradict those findings.

In summary, this MKL approach provides an interesting way to combine feature groups without forcing them all into a single vector (which makes sense as they are not expected to contribute equally). It gives access to kernel weights, which turn the classifier into an even more interpretable tool. And the linear MKL ablation results confirmed the importance of β , highlighted the major complementary role of δ , and showed that some descriptors such as γ contribute very little and seem negligible. It shows that perfect performance is still attainable after removing several individual components. This indicates that the signal of reduction is redundant and not confined to a single zone or parameter. Nevertheless, because of the limited size of the dataset and the marked C hyperparameter sensitivity, these conclusions should be interpreted with caution. Further experiments would be necessary to assess how far these conclusions generalize beyond the present sample.

Chapter 4

Convolutional Neural Network (CNN) approach

4.1 Motivations

While the parametric analysis of contour lines provided meaningful descriptors of the instruments, several limitations motivated the exploration of alternative approaches, in particular convolutional neural networks (CNNs).

4.1.1 Limitations of parametric representations

The contour-line fitting approach relies on a set of parameters (α, β , δ , and γ) to describe each level curve. Although it has been shown that these parameters allow the capture of important geometric properties and make useful interpretations, they remain an approximation of the original data and shapes.

Several issues may arise from that: (i) the fitting procedure may fail on some naturally different instruments, (ii) the number of valid contour levels varies between instruments, requiring resampling and interpolation steps that may, in a sense, distort the original geometric information, and (iii) other human choices (such as the choice of the parametric function) may not fully capture subtle or irregular deformations, or may simply make the data biased by human choices.

These limitations motivated a CNN approach. Applying such models to elevation maps of instruments would help avoid the need for handcrafted descriptors as the models could theoretically learn relevant patterns directly from the almost-raw data. The robustness of such an approach could make it more suitable for handling intermediate or ambiguous cases. And the spatial awareness of CNN models might better exploit the spatial relationships between neighboring points.

These properties make CNNs particularly appealing for our problem, where the distinction between reduced and non-reduced instruments relies on very small, subtle distinctions. In some sense, the parametric analysis from the MKL approach provides interpretability and domain knowledge, while CNNs offer a potentially more flexible framework to leverage the full complexity of the data. Moreover, the use of elevation grids as input data aligns quite well with the previous preprocessing pipeline, which already produces regularized 2D representations of the instrument surfaces, without requiring additional transformations.

4.2 Algorithm and architecture

As the theoretical foundations of Convolutional Neural Networks (CNNs) have already been introduced in Chapter 2, the purpose of this section is to describe the concrete architectural and algorithmic choices that were made for the present classification task. The implementation relies mainly on common Python packages such as TensorFlow/Keras, with the help of NumPy, pandas, and scikit-learn for training, data manipulation, and evaluation.

4.2.1 Input representation and preprocessing

Resizing strategies. The CNN approach operates on 2D geometric maps extracted from the violin surfaces. For each instrument, the loading function first reads the sound board or back maps from data files. Since the original maps do not all share the same spatial dimensions, the code first determines the maximum dimensions observed in the dataset and then resizes all instruments to a common size. Two resizing strategies were implemented and tested: an *absolute* strategy based on symmetric zero-padding, and a *relative* strategy based on 2D interpolation.

The difference between these two strategies is important. The *absolute* resizing preserves the original spatial resolution of each instrument by embedding it into a larger grid through symmetric zero-padding. In that case, the physical distances and proportions between points remain unchanged, and smaller instruments occupy a smaller region of the final grid. By contrast, the *relative* resizing rescales each map to a fixed resolution using interpolation. This enforces a common spatial support across all instruments, normalizing their size and allowing the CNN to focus on relative shape variations rather than absolute dimensions. Figure 4.1 illustrates these two resizing strategies on the same example instrument sound board.

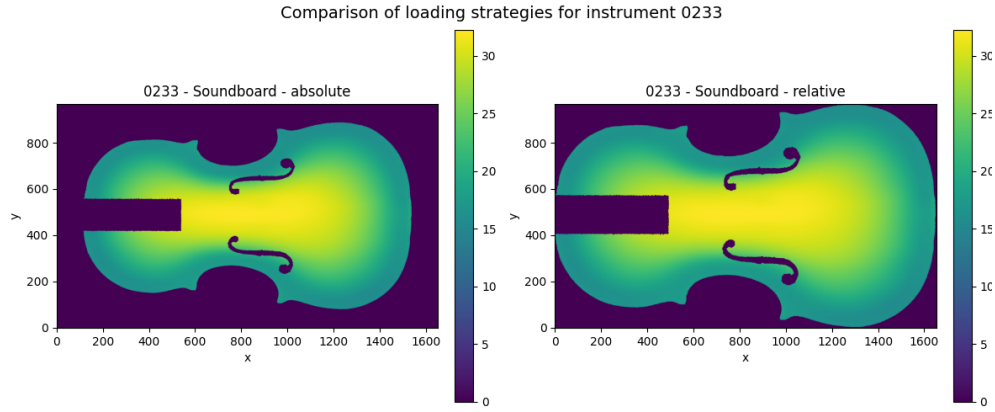


Figure 4.1: Comparison of the two resizing strategies (absolute on the left, relative on the right) for the same instrument sound board data.

Other input representations. In addition to these two resizing strategies, alternative input configurations were also explored in order to better control the spatial representation of the data.

An additional preprocessing variant was investigated by splitting the input maps along the height of the instrument. More specifically, the height dimension of each instrument could be divided into two halves, allowing the model to be trained either on the upper or lower part of the instrument independently. These experiments and results will be further described in a later section, but the idea was to restrict the field of view of the CNN to a specific region of the surface, making it possible to analyze whether discriminative information is concentrated in particular areas. Although this modification reduces the amount of available spatial information, it can also help the model focus on more localized geometric patterns and reduce potential noise coming from less informative regions.

Another variation of the loading function based on resampled grids was tested. In that setting, a specific region of all instruments was interpolated beforehand to a fixed resolution (for example 100×250), ensuring a perfectly consistent input size across the dataset without requiring dynamic resizing during loading. This approach, illustrated in Figure 4.2 simplifies the preprocessing pipeline and guarantees strict alignment of the spatial structures, at the significant cost of introducing an additional interpolation step and potentially losing details in data.

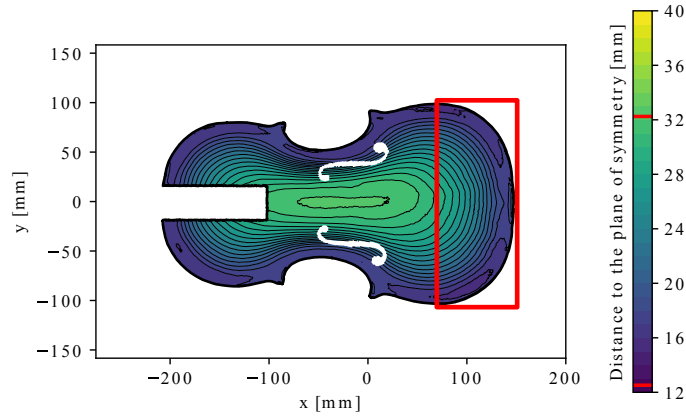


Figure 4.2: Selected subregion of violins for resampling (red rectangle), from [BCG26]

4.2.2 Data augmentation

Because the dataset is very small and imbalanced, the implementation includes a configurable data augmentation procedure applied only to the training folds. Three types of transformations have been used: (slight) rotations, horizontal mirroring, and (slight) shifts. These have been tested on their own and through combinations of multiple transformations.

Figure 4.3 illustrates the three augmentations on one instrument’s sound board. Note that here, for visualization purposes only, high transformation values were used.

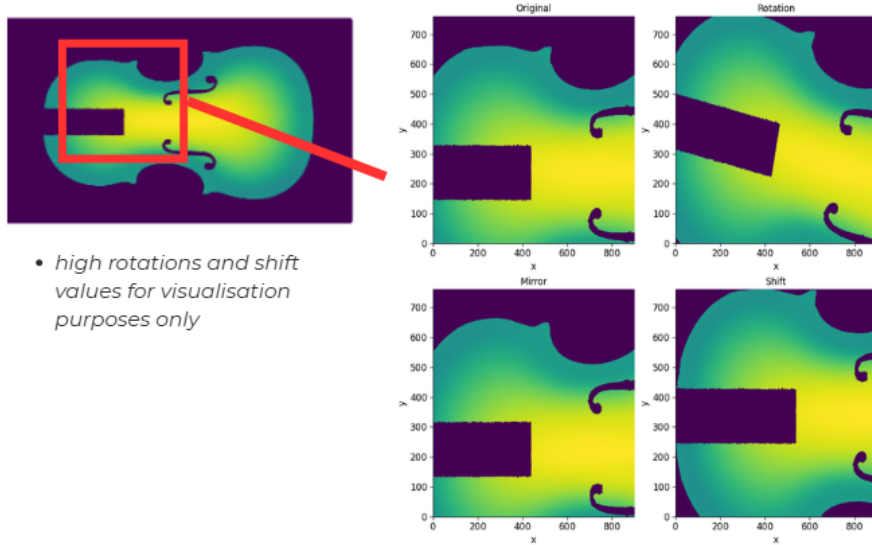


Figure 4.3: Illustration example of different augmentations on one instrument's sound board.

4.2.3 CNN architectures

In total, 8 different CNN architectures have been explored. Four of these architectures are fully customized and trained, the four others are based on common, widely used pretrained architectures on top of which a simple classification head has been added. Among the fully customized architectures are one simple baseline CNN architecture, and 3 slightly more complex ones derived directly from the baseline.

Customized CNN architectures

Simple baseline CNN. The simplest CNN model used in the implementation is a compact sequential architecture designed to serve as a baseline with the limited size of the dataset. It is composed of three convolutional blocks (including a convolutional layer, ReLU activation, and max-pooling) followed by a small fully connected classification head. Figure 4.1 shows the architecture in more detail, where k is the size of the kernel.

SimpleCNN architecture
• Convolutional layer with 16 filters, k, ReLU activation, and L_2 regularization • 2×2 max-pooling layer • Convolutional layer with 32 filters, k, ReLU activation, and L_2 regularization • 2×2 max-pooling layer • Convolutional layer with 64 filters, k, ReLU activation, and L_2 regularization • Global average pooling layer • Dense layer with 64 units, ReLU activation, and L_2 regularization • Dropout layer • Dense layer with 2 output units

Table 4.1: SimpleCNN architecture

CNN 5 architecture. This architecture is a direct extension of the simple baseline CNN. It follows the same design but increases the depth of the network by including two additional convolutional blocks. The goal of this variation is to observe the impact of increased model capacity on performance. Figure 4.2 shows the detailed architecture.

Simple CNN architecture
• Input layer • Convolutional layer with 16 filters, k, ReLU activation, and L_2 regularization • 2×2 max-pooling layer • Convolutional layer with 32 filters, k, ReLU activation, and L_2 regularization • 2×2 max-pooling layer • Convolutional layer with 64 filters, k, ReLU activation, and L_2 regularization • 2×2 max-pooling layer • Convolutional layer with 128 filters, k, ReLU activation, and L_2 regularization • 2×2 max-pooling layer • Convolutional layer with 256 filters, k, ReLU activation, and L_2 regularization • Global average pooling layer • Dense layer with 256 units, ReLU activation, and L_2 regularization • Dropout layer • Dense layer with 2 output units

Table 4.2: Simple CNN 5 architecture

BatchN CNN architecture. This architecture extends the baseline CNN architecture by including batch normalization layers before every activation layer. Batch normalization has been explained in section 2.3.2 and in much more detail in [IS15]. As a reminder, it can be used to stabilize and accelerate the training of deep neural networks. The general idea is to normalize the activations of each layer during training so that they maintain a more stable distribution. Figure 4.3 shows the different layers of this architecture.

BatchN CNN architecture
• Input layer • Convolutional layer with 16 filters, k, and L_2 regularization • Batch normalization layer • ReLU activation layer • 2×2 max-pooling layer • Convolutional layer with 32 filters, k, and L_2 regularization • Batch normalization layer • ReLU activation layer • 2×2 max-pooling layer • Convolutional layer with 64 filters, k, and L_2 regularization • Batch normalization layer • ReLU activation layer • Global average pooling layer • Dense layer with 64 units, ReLU activation, and L_2 regularization • Dropout layer • Dense layer with 2 output units

Table 4.3: BatchN CNN architecture

Multiscale CNN architecture. This architecture extends the baseline CNN by introducing a multiscale feature extraction mechanism in the first layer. Instead of applying a single convolution, three parallel convolutional layers with different kernel sizes (3×3 , 5×5 , and 7×7) are applied to the input. The resulting feature maps are then concatenated, so that the network can capture multiple scales simultaneously. Figure 4.4 shows the complete architecture.

Multiscale CNN architecture
• Parallel convolutional layers with 16 filters: 3×3, 5×5, 7×7, ReLU activation, L_2 regularization • Concatenation of 3 resulting feature maps • 2×2 max-pooling layer • Convolutional layer with 16 filters, k, ReLU activation, L_2 regularization • 2×2 max-pooling layer • Convolutional layer with 32 filters, k, ReLU activation, L_2 regularization • 2×2 max-pooling layer • Convolutional layer with 64 filters, k, ReLU activation, L_2 regularization • Global average pooling layer • Dense layer with 64 units, ReLU activation, and L_2 regularization • Dropout layer • Dense layer with 2 output units (logits)

Table 4.4: Multiscale CNN architecture

Pretrained CNN architectures.

Four of the explored architectures were pretrained models: VGG [SZ15], ResNet [He+15], DenseNet [Hua+17], and EfficientNet-B0 [TL19]. They were selected because they have all been extensively studied and have demonstrated strong performances, while relying on fundamentally different design principles. VGG is based on a simpler and uniform architecture using stacked small 3×3 convolutional filters, using depth as the main source of performance improvement. ResNet introduced residual connections, essentially allowing some information to bypass layers. DenseNet further extended this idea by connecting each layer to all previous ones in order to encourage feature reuse, and EfficientNet-B0 relies on an optimized scaling strategy, achieving a strong trade-off between performance and computational efficiency.

However, it is important to note that, although these architectures have demonstrated strong performance on common datasets, their effectiveness has not been studied on data comparable to the one considered in this thesis. Most existing works in computer vision focus on tasks such as object detection or classification, where the goal is to distinguish between visually different categories. In contrast, the present problem involves identifying extremely subtle, maybe not perceptible, geometric differences within highly imbalanced classes of violins.

These four architectures were also selected for their practical convenience, as they are readily available through the `tensorflow.keras.applications` library. In this work, they were all used following the same transfer learning strategy: the pretrained convolutional base was kept as a base model with frozen weights, and a small classification head was added on top. Only the weights of the added classification head were trained. Table 4.5 shows the way we used each of these four pretrained architectures.

Pretrained CNN architecture setup
• Input layer • Pretrained convolutional base (VGG / ResNet / DenseNet / EfficientNet-B0) • Global average pooling layer • Dense layer with 128 units, ReLU activation, and L_2 regularization • Dropout layer • Dense layer with 2 output units (logits)

Table 4.5: Pretrained CNN architecture used with transfer learning

4.2.4 Training and evaluation

Similarly to what was done in the MKL approach and in previous work [BCG25], this CNN approach relies on a nested cross-validation strategy. At each iteration, one instrument is held out as the test sample, while the remaining $n - 1$ samples are used for training. These training data are then passed to an inner cross-validation loop dedicated to hyperparameter tuning. More specifically, the inner loop uses a Stratified K-Fold CV procedure in order to preserve as much as possible the original class distribution within each fold. For each candidate combination of hyperparameters, the model is trained and evaluated across all folds for different numbers of epochs, and the corresponding balanced accuracies are averaged. The combination of hyperparameters that gives the highest mean validation balanced accuracy is selected, as well as its corresponding number of epochs. The number of random candidate combinations of hyperparameters was generally set to 10.

Using this selected configuration, the model is then retrained on the full outer training set (the $n - 1$ samples) and evaluated on the held-out test sample. This process is repeated for each instrument, ensuring that every sample is used once for testing. This evaluation protocol follows the recommendations of [Bra+23], discussed previously in subsection 2.3.3, combining an outer LOOCV loop for the evaluation part of our small datasets with an inner (stratified) K-Fold loop for robust hyperparameter tuning. As a reminder, Figure 3.2 illustrates this nested CV methodology. Algorithm 3 shows a simplified pseudocode, focusing on the main steps for clarity and brevity, of the training and evaluation procedure, where k is the number of folds used in the inner Stratified K-Fold.

Algorithm 3 Training and evaluation procedure

Data: Hyperparameter grids, K , max_epoch, seed

Result: Predictions, selected hyperparameters, performance metrics

Load data for all instruments

foreach *test instrument* **do**

 Define all remaining instruments as training set

 Select best hyperparameters using inner Stratified k-Fold CV

 Train CNN model with selected hyperparameters

└ Predict label for test instrument

Compute performances and metrics

For all CNN-based experiments, the models were trained using the AdamW optimizer. This provides a stable and effective regularization, especially when combined with L_2 penalties present in the network. The learning rate and weight decay val-

ues were treated as hyperparameters. The models were compiled using the sparse categorical cross-entropy loss function. This formulation avoids the need for explicit softmax activation in the final layer, as it is internally handled by the loss function.

To better handle class imbalance, a specific initialization strategy was adopted for the output layer bias. As random initialization of this bias can lead to poor initial predictions in imbalanced settings, the approach proposed in *A Recipe for Training Neural Networks* [Kar] and recommended in the official TensorFlow documentation was used [Ten]. The initial bias was set to $\log(\frac{NR}{R})$, where NR and R are the numbers of samples in each non-reduced (NR) and reduced (R) class.

In addition, class weighting was applied during training to compensate for imbalance. The weight of the non-reduced class was fixed to 1, while the weight of the reduced class was set to the ratio $\frac{\#non-reduced}{\#reduced}$ so that both classes contribute more evenly to the optimization process.

4.3 Experiments and results

4.3.1 First hyperparameters tuning (baseline)

The first experiment aimed at defining an initial baseline in order to assess the performance that could be directly obtained with such a convolutional neural network approach. This baseline also serves as a reference point for following comparisons between architectural variations, input representations, and the introduction of additional techniques such as data augmentation. This was conducted in two parts. First, the goal was to estimate the balanced accuracy (BA) performance of the model, through the robust nested cross-validation evaluation procedure described in the previous subsection. Second, it aimed at identifying a set of baseline hyperparameter values in order to be able to fix them and make fair comparisons in subsequent experiments.

Evaluation of the Simple CNN architecture. This first part focused on evaluating the performance that could be achieved with such a simple convolutional neural network. To do that, the nested cross-validation strategy, described in Section 5.2.4, was used. It combined a LOOCV outer loop for evaluation with a StratifiedKFold inner loop for hyperparameter tuning.

For this baseline CNN architecture, six hyperparameters were tuned: the learning rate, batch size, dropout rate, weight decay, kernel size, and regularization strength. Each of these plays a specific role in the training and for the generalization ability of the model. The learning rate controls how big a step can be when the model learns. Making it too small can make the learning very slow, but making it too large can make the model unstable and prevent convergence. The batch size simply represents the number of sample before updating the model. Smaller batches may introduce more noise but potentially improve generalization. The dropout rate is a regularization technique that randomly turns neurons off during training, which is intended to force the network not to rely too heavily on specific neurons. Weight decay introduces an L2 penalty on the model parameters, especially on large weights. It encourages smaller weights and smoother solutions. The kernel size determines the size of the convolutional window and therefore the scale of spatial features captured from the input representations, which in our context is particularly important as we work with elevation maps. Finally, the regularization strength controls additional L2 penalties applied directly to the network layers' parameters, as opposed to the weight decay

that is applied during optimization.

The number of random candidate combinations of hyperparameters for the inner fold grid search was set to 10. And the number of inner validation folds was set to 4. To account for variability due to random initialization, the entire process was repeated seven times with different random seeds. The ranges associated with each hyperparameter for the random grid search are reported in the following Table 4.6. They represent values commonly used in deep learning practice, which is supposed to be broad but not too extreme.

Hyperparameters	Ranges
Learning rate	$[10^{-4}, 5 \cdot 10^{-4}, 10^{-3}, 5 \cdot 10^{-3}, 10^{-2}, 5 \cdot 10^{-2}]$
Batch size	$[4, 8, 12, 16, 24]$
Dropout rate	$[0.0, 0.1, 0.3, 0.5]$
Weight decay	$[10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 0.0]$
Kernel size	$[3, 4, 5]$
Regularization strength	$[10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 0.0]$

Table 4.6: Hyperparameter ranges used for baseline’s random grid search

The grid search was initially performed on 10 random combinations of hyperparameters, and with the maximal number of epochs set to 100. The average balanced accuracy obtained across all seeds was 61.7%, with a highest value of 75% and a lowest value of 50%. Table 4.7 reports the mean validation performance (Val. BA) as well as the test performance obtained from these experiments. The standard deviations and average computational time are also reported.

Models	Val. BA (%)	Test BA (%)	Time (s.)
Simple CNN	67.5 ($\pm$ 2.88)	61.7 ($\pm$ 11.7)	5785

Table 4.7: Performances of the first overview experiment, with max_epoch=100

These results indicate moderate classification performances for the simple CNN architecture. While the validation balanced accuracy reaches on average 67.5%, the corresponding test performance drops to 61.7%, and is accompanied by a relatively large standard deviation (11.7%). This variability may suggest that the model struggles to converge to a stable solution.

One possible explanation lies in the intrinsic characteristics of the dataset. Each instrument is represented by a large amount of structured geometric data, while the total number of available samples remains very small. This imbalance between input dimensionality and dataset size may make the learning task particularly challenging, as the model may have difficulty identifying generalizable patterns without overfitting to specific training instances. In addition, the choice of the maximum number of training epochs may also contribute to this behavior. The value of `max_epoch=100` was selected arbitrarily after some exploratory analysis. Training for too many epochs could lead to an artificial improvement of the validation performance due to overfitting, which would result in a decreased generalization ability on unseen data. This could explain both the gap observed between validation and test performances and the instability reflected in the high variance of the test results.

Selecting the baseline hyperparameter values. This second set of experiments was made in order to define a set of hyperparameters to be used as a baseline configuration, to be able to make fair comparisons across subsequent experiments. A variation of the methodology used to evaluate the Simple CNN architecture performance was applied. For each candidate set of hyperparameters, the mean validation balanced accuracy (BA) was computed this time across all outer folds.

It is important to note that here the associated performance is subject to bias since the selection is based on results obtained from the outer cross-validation loop, meaning that each instrument, including those used as test samples, indirectly influenced the choice of hyperparameter values. That being said, even though the performance accuracy may be slightly biased, this procedure nevertheless provides a well-performing set of hyperparameters as, on average, they were the one performing best. This distinction is important but seems acceptable here since the objective is not to use them to report an unbiased estimate of the model performance but rather to define a strong and consistent baseline configuration for comparison purposes.

The random search was performed on the ranges of hyperparameters in Table 4.6. This grid search was initially performed on 30 random combinations of hyperparameters. The average balanced accuracy obtained across these configurations was 59.1%, with a highest value of 70% and a lowest value of 50.9%. These results highlight the noteworthy sensitivity of the model to hyperparameter choices in this context, and how impactful they can be on the performance.

The 3 best-performing configurations that gave the highest validation balanced accuracy (BA) performances, as well as the 3 worst-performing configurations are reported in Table 4.8. The **Epoch** column indicates the epoch at which the highest performance was obtained for each configuration, while the BA columns correspond to the associated balanced accuracies, respectively BA obtained on the validation set (**vBA**) and BA obtained on the test set (**tBA**). The remaining columns describe the hyperparameters of each configuration, namely the kernel size, learning rate, batch size, dropout rate, weight decay, and regularization strength.

Epoch	vBA (%)	tBA (%)	kernel	learning	batch	dropout	decay	reg.
100	70.0	70.0	3	0.001	24	0.0	0.0001	0.01
50	68.7	70.0	3	0.0005	8	0.0	0.00001	0.001
100	68.1	57.5	3	0.005	24	0.0	0.00001	0.001
...	...	...	...	...	...	...	...	...
15	51.5	50	3	0.005	24	0.5	0.001	0.01
1	51.1	25.0	3	0.05	24	0.5	0.001	0.1
50	50.9	47.5	3	0.01	4	0.8	0.0	0.01

Table 4.8: 3 best-performing and 3 worst-performing hyperparameter configurations obtained.

As the goal is not to only identify configurations that show one single peak performance at a specific epoch, but rather to assess their stability and consistency over the training epochs, the evolution of the BA performances of the top 3 candidate configurations have been reported according to the number of epochs. Figure 4.4 illustrates how the validation and test performances evolve for these three configurations reported in Table 4.8. Note that, each candidate configuration is represented by two curves of the same color. The dotted curve represents the evolution of the validation

performance (vBA) across epochs, while the plain curve represents the evolution of the test performance (tBA) across epochs.

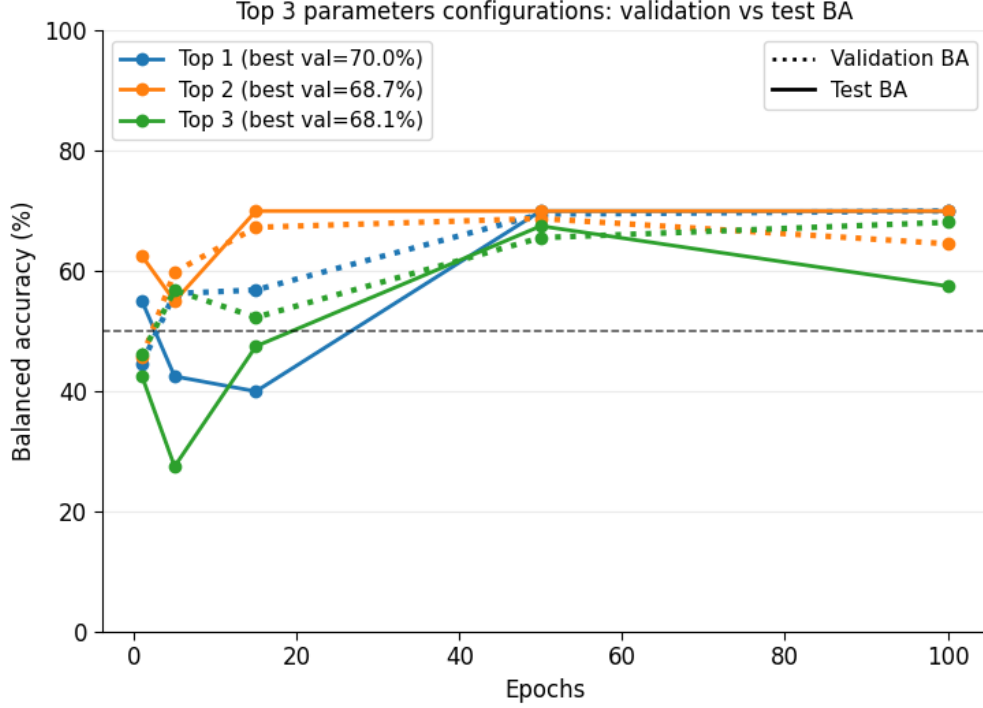


Figure 4.4: Evolution of the 3 best combinations obtained through random grid search

Configuration 2 (orange curve) appears to reach a plateau relatively early, around epoch 15. This behavior may be due to overfitting induced by the hyperparameter selection process, the effect of regularization combined with less training batches, or even a combination of both. In contrast, Configuration 1 (blue curve) achieves its peak performance at epoch 50, where both validation and test balanced accuracies stabilize around 70%. We can observe a strong alignment between validation and test curves for all three configurations at epochs 50.

The choice of hyperparameter values for the baseline is reported in Table 4.9. The selection was conducted by the behavior observed in these experiments as well as by general considerations regarding generalization and overfitting. The learning rate was set to 10^{-3} as it appeared in the best-performing configuration, and is a common choice in the literature. A relatively large batch size of 24 was selected. The dropout rate was set to 0.0, indicating that additional regularization was not necessary in this context. This may be explained by the already strong regularization induced by the training protocol itself (nested CV), the L2 regularization parameter, as well as the weight decay. In contrast, a small but non-zero weight decay of 10^{-4} was selected. The kernel size was fixed to 3, which corresponds to the smallest receptive field among the tested values. Finally, a relatively strong regularization strength of 10^{-2} was selected for the layers, reinforcing the control over model complexity and contributing to the observed alignment between validation and test performances.

Hyperparameters	Ranges	Selected
Learning rate	$[10^{-4}, 5 \cdot 10^{-4}, 10^{-3}, 5 \cdot 10^{-3}, 10^{-2}, 5 \cdot 10^{-2}]$	10^{-3}
Batch size	$[4, 8, 12, 16, 24]$	24
Dropout rate	$[0.0, 0.1, 0.3, 0.5, 0.8]$	0.0
Weight decay	$[0.0, 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}]$	10^{-4}
Kernel size	$[3, 4, 5]$	3
Regularization strength	$[0.0, 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}]$	10^{-2}

Table 4.9: Hyperparameter tuning selection

4.3.2 Effect of hyperparameters

From the previously selected hyperparameter values, a series of one-at-a-time experiments was conducted in order to better understand the individual impact of each parameter on the model’s performance and to ensure that the first random search did not miss important values within or outside the tested ranges. In this setting, only one hyperparameter was varied at a time while all others were kept fixed to the selected baseline values just reported in Table 4.9.

For each hyperparameter, the evolution of the balanced accuracy as a function of the number of training epochs is reported in Figure 4.5. The performance is the average performance out of 5 tested seeds. Each subplot corresponds to a specific hyperparameter, while each curve represents a different value within the predefined range. For these experiments, the maximal number of epoch was set to 600. To ensure a better clarity, and since nothing really noteworthy was observed between epoch 100 and epoch 600, all subplots have been stopped at epoch 100.

Baseline vs one-at-a-time variations (Test BA)

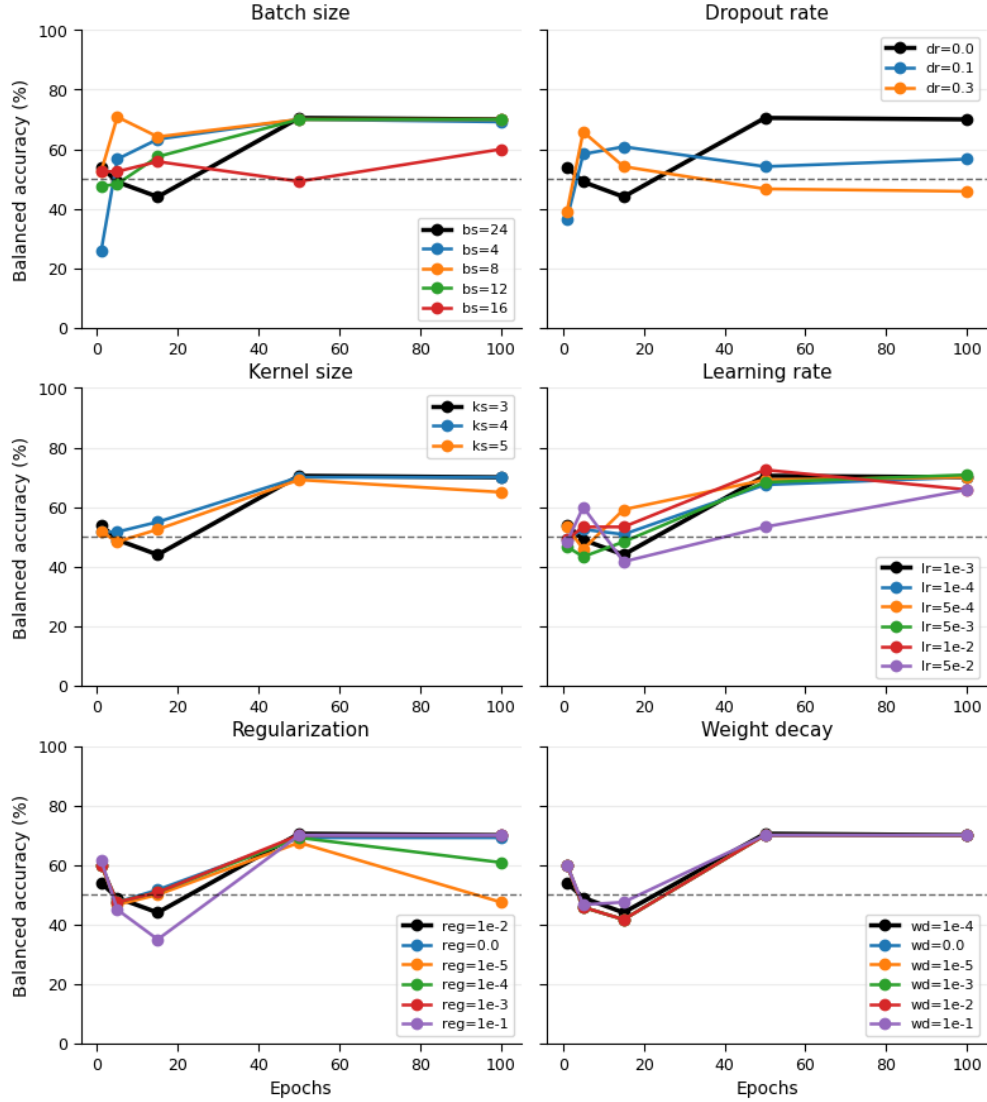


Figure 4.5: Evolution of performance according to small one-at-a-time hyperparameter variations, each curve represent the performance of a tested variation

Several observations can be made from these results. First, regarding the batch size, most tested values eventually converge towards similar performance levels, particularly around epoch 50. However, smaller batch sizes appear to reach this performance plateau more rapidly. This behavior can be explained by the fact that, for a fixed number of epochs, a smaller batch size results in a higher number of parameter updates, technically leading to more training iterations. This effect is particularly visible around epoch 15, where performance seems to correlate most to batch size. Interestingly, a batch size of 16 gave lower performance compared to the other tested values, although no clear explanation emerges as to why.

The effect of the dropout rate appears consistent with previous observations and discussion. Increasing the dropout rate systematically leads to lower performances. In this context, the combination of the training nested CV procedure and the presence of strong L2 regularization likely already provides sufficient regularization, making dropout unnecessary and even detrimental to the learning process.

The observed influence of kernel size, learning rate, and weight decay within the tested ranges was small. Apart from minor variations, these hyperparameters lead to very similar performance across epochs. This might suggest that the explored ranges are located in regions where performance is already close to a plateau.

In contrast, given the importance of regularization and the sensitivity observed in previous experiments, it was further investigated and discussed in more detail in dedicated experiments.

Effect of regularization. As regularization plays an important role in neural network training, and given the impact previously observed in this context, a dedicated set of one-at-a-time experiments was conducted to further investigate its effect. Figure 4.6 shows a more detailed view of the regularization behavior from Figure 4.5, extended to 600 epochs.

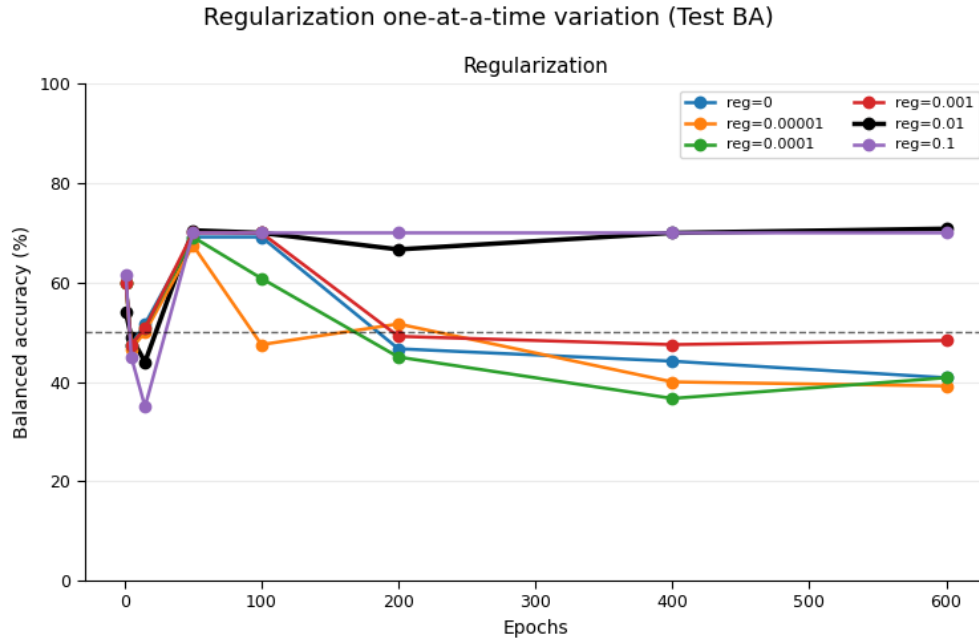


Figure 4.6: Evolution of performance according to small one-at-a-time hyperparameter variations on regularization only

The strong influence of regularization on both performance and stability can be observed from this experiment. Lower levels of regularization tend in this context to lead to a noticeable degradation on the test set as the number of epochs increases, which may suggest overfitting. By contrast, stronger regularization seems to result in more stable behavior, reaching a plateau at epoch 50.

A similar experiment was conducted using a more detailed range of regularization values in order to better identify the best performing range in our context. In addition to the previously tested values, intermediate regularization strengths of 0.5, 0.05, and 0.005 were considered, along with higher values such as 1, 10, and 100. Figure 4.7 shows the evolution of the performance for each of these variations as a function of the number of epochs.

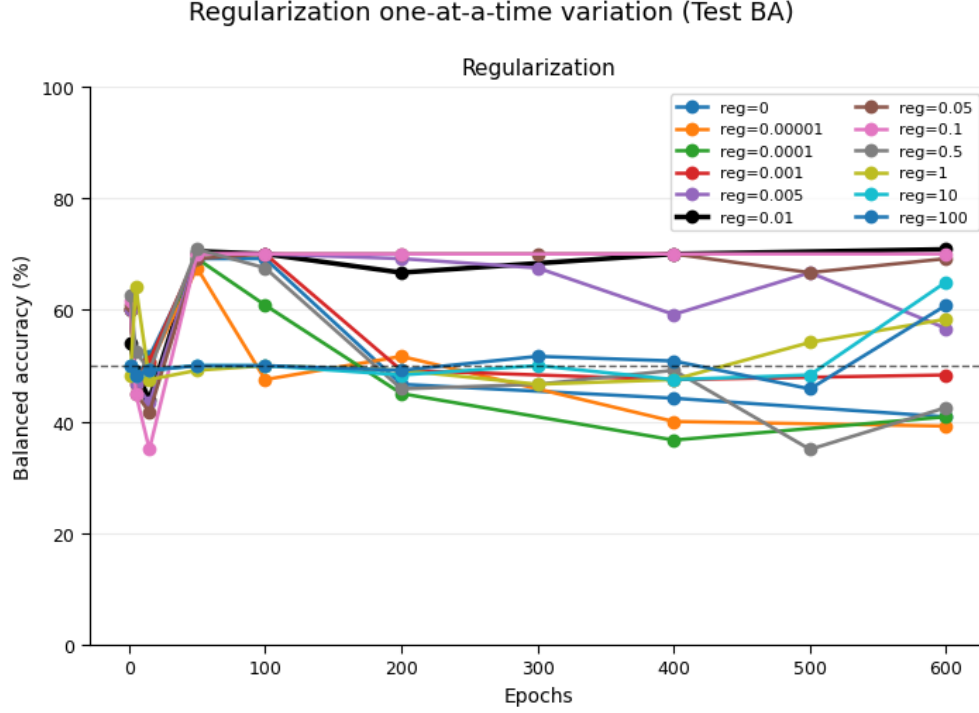


Figure 4.7: Evolution of performance according to small one-at-a-time hyperparameter variations on regularization only

Although the graph becomes visually dense due to the number of curves, several trends can be identified. First, increasing the regularization strength beyond 0.1 does not lead to improved performance, and even strongly degrades it. Second, the intermediate values tested (0.5, 0.05, and 0.005) achieved performances very similar to those previously observed with 0.1 and 0.01, especially 0.5 and 0.05. The lowest value of the three (0.005) gave similar results at first, then degraded quite significantly. Overall, these results suggest that the most effective range of regularization in this context, and with the previously selected baseline hyperparameters, lies approximately between 0.1 and 0.001.

4.3.3 Architecture comparison

Variations of baseline architecture

As the baseline architecture and the associated hyperparameter values had been previously defined, several variations of this architecture were explored in order to assess the impact of structural modifications on performance. The point of these varia-

tions was to investigate whether improvements could be achieved by modifying model capacity, increasing the number of layers, or even capturing features at different scales.

More specifically, the three custom architecture variants described in section 5.2.3 were considered. As a reminder, the first one, referred to as **CNN5**, extends the baseline model by increasing the depth of the network from three to five convolutional layers. This modification is intended to allow the model to learn more complex hierarchical representations of the input data. The second variant, **BatchN**, incorporates batch normalization layers within the baseline architecture. Batch normalization is used to stabilize training, and can also have a regularization effect. Finally, the **multiscale** architecture introduces three parallel convolutional branches with different kernel sizes (3x3, 5x5, 7x7), whose outputs are then concatenated. This design tries to capture features at multiple scales simultaneously, which could be relevant if both local and more global geometric patterns may be informative.

These architectural variations were evaluated under the same experimental procedures as the baseline in order to ensure a fair comparison. First, each model was assessed using the outer LOOCV procedure only, with the baseline hyperparameter configuration, to isolate the effect of architectural changes without introducing additional variability due to hyperparameter tuning. This is a direct comparison of how each architecture performs under identical training conditions. In a second step, an independent random grid search is performed for each architecture, following the same nested CV strategy as described for the baseline. This has for objective to evaluate the potential of each architecture variation when its hyperparameters are more specifically tuned.

Comparison of variant architectures with baseline hyperparameters. As mentioned previously, this first set of experiments was conducted to isolate the effect of architectural modifications without introducing additional variability due to hyperparameter tuning. In order to do that, all models were trained using the same hyperparameter configuration as the baseline, and only the outer LOOCV procedure was used. Figure 4.8 shows the evolution of the performance according to the number of epochs. Each curve corresponds to the mean balanced accuracy computed over five different seeds, while the associated variability is represented through standard deviation bands. The black curve corresponds to the baseline (SimpleCNN) architecture.

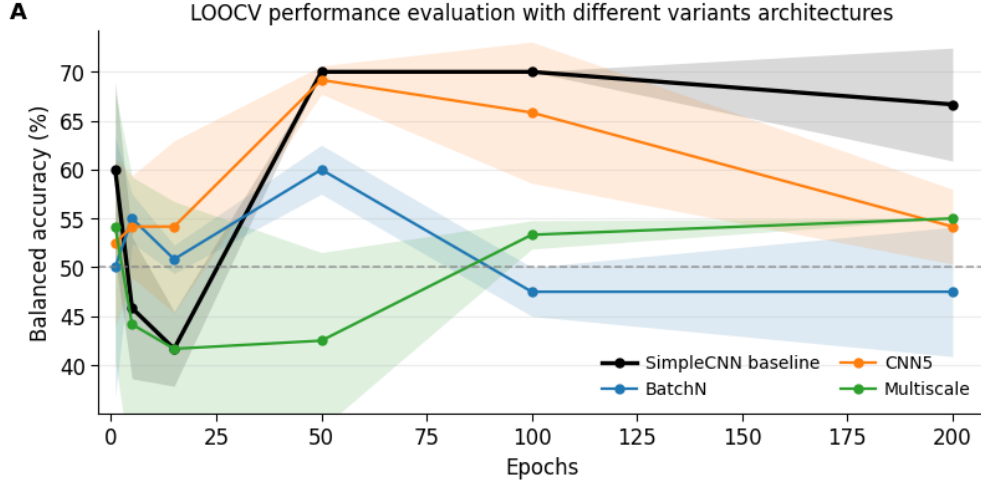


Figure 4.8: Comparison of the evolution of performance according to the number of epochs for 4 CNN architectures.

The **CNN5** architecture, despite its increased number of layers, does not seem to lead to any clear improvement in performance. It appears to perform slightly better than the baseline at low epoch values (below 50), but this comes at the cost of a substantially higher standard deviation across seeds. At around 50 epochs, its performance becomes comparable to the baseline, after which it starts to degrade significantly. This behaviour may suggest that increasing model depth alone is not sufficient in this context, and may even introduce additional optimization difficulties given the limited size of the dataset and the extreme subtlety of potential discriminative information. The **BatchN** architecture exhibits a slightly improved stability during training, particularly at earlier epochs, which is consistent with the expected effect of batch normalization. However, this increased stability does not translate into any meaningful improvement in test performance, as the achieved balanced accuracy remains close to 50% across most epoch values. The **multiscale** architecture gave the weakest results among the tested variants. It is associated with a lower mean performance and a higher variability across seeds compared to the other models. This might suggest that introducing multiple convolutional branches with different kernel sizes does not provide a benefit in this setting. One possible explanation could be that the relevant geometric information is already sufficiently captured by the simpler convolutional structure of the baseline model, and that the additional complexity introduced by the multiscale design only leads to overhead, less stable training without improving generalization at all.

Overall, these results suggest that, when trained under identical hyperparameter settings, these simple architectural variations alone do not lead to significant performance improvements. In the context of a very limited dataset, the additional complexity introduced by these models appears to degrade performance more than it improves it. It is important to note, however, that the performances reported here are not completely unbiased, as the hyperparameters were fixed based on the baseline architecture and then applied uniformly across all outer folds and all variants. This setting favours a controlled comparison of architectures, but does not necessarily reflect the optimal configuration for each model. This motivated the second part of the analysis, in which each architecture was evaluated under an independent

hyperparameter tuning procedure to better assess its full potential.

Evaluation of the baseline architecture variations. This second set of experiments was conducted in order to obtain a more accurate evaluation of the actual potential of each of the architectural variants. The experimental procedure follows the same protocol as the one used for the baseline architecture. More specifically, it used a nested cross-validation scheme, combining an inner Stratified K-Fold procedure with four folds for hyperparameter tuning, and an outer LOOCV for performance evaluation. The ranges of hyperparameter in the grid space were identical to the ones used for the baseline, shown in Table 4.6.

Figure 4.10 shows the validation and test performances obtained from this second set of experiments, as well as the corresponding computation time for each of the three architectural variants. The reported values correspond to the mean balanced accuracy across seeds, with the standard deviation indicated in parentheses.

Models	Val. BA (%)	Test BA (%)	Time (s.)
Simple CNN	67.5 ($\pm$ 2.88)	61.7 ($\pm$ 11.7)	5785
CNN5	67.8 ($\pm$ 0.6)	57.5 ($\pm$ 2.5)	9991
Batchn	62.8 ($\pm$ 3.2)	55.0 ($\pm$ 2.5)	8847
Multiscale	60.5 ($\pm$ 5.7)	50.0 ($\pm$ 14.1)	16401

Table 4.10: Validation and test performances obtained for each architecture

From these results, none of the explored architectural variants provided a clear improvement over the baseline Simple CNN. On the opposite, increasing the architectural complexity seems to degrade the performance while significantly increasing the computational cost. The CNN5 architecture achieves a validation performance similar to the baseline but results in a lower test balanced accuracy, suggesting a reduced generalization ability despite its increased depth. The BatchNorm-based architecture further decreases both validation and test performances, the addition of normalization layers does not seem to bring any benefits in this specific context. Finally, the Multiscale architecture, which is the most complex among the tested variants, gave the lowest performances and a high variability across seeds, while also requiring the longest computation time.

These observations suggest that more complex architectures might not necessarily be better suited for this task. Given the small size of the dataset, the relatively high dimensionality of the input representations, and the extremely subtle discriminative information, increasing the model capacity may lead to more difficulties in learning stable and generalizable patterns. Instead of capturing more relevant features, the additional parameters seem to amplify noise and lead to poorer generalization.

Pretrained architectures

In addition to these four custom-designed architectures, four well-established pre-trained convolutional neural networks were also evaluated, namely EfficientNetB0, VGG16, ResNet50, and DenseNet121. These models were originally trained on large-scale image datasets and are well-known and studied. They are all conveniently available in the applications package of TensorFlow. Their use in this work is motivated by the hypothesis that more generic visual features learned on natural images

may still capture relevant geometric patterns in elevation maps, despite the important domain difference. These architectures are described thoroughly in the papers they originated from, mentioned in section 5.2.3.

In our experiments, these pretrained models were kept frozen: their weights were not updated during training. On top of each architecture, a fully connected (dense) classification layer with regularization was added. Only the parameters of this final layer were updated during training, while the pretrained base remained fixed. As a consequence, regularization is applied solely at this final stage, which results in a more limited impact compared to custom architectures where regularization is distributed across almost all layers.

Comparison of pretrained architectures with baseline hyperparameters.

Similarly to the comparison performed for the custom architectural variants, a first set of experiments was conducted to isolate the effect of architectural changes for pretrained models under fixed hyperparameter settings. In this setting, the same baseline hyperparameter values were reused for all architectures, and the evolution of the balanced accuracy was evaluated according to the number of training epochs. The point of this first set of experiments was, once again, to be able to compare architectures without any additional variability due to hyperparameter tuning.

Figure 4.9 shows the test balanced accuracy obtained for each of the four pre-trained architectures, along with the baseline CNN represented by the black curve. Each curve corresponds to the mean performance across five different seeds, while the associated standard deviations are also displayed.

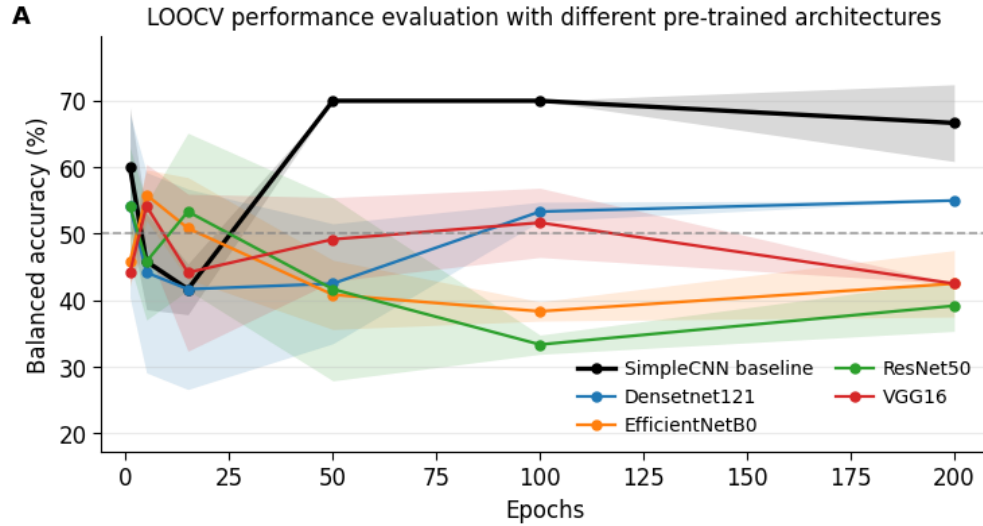


Figure 4.9: Comparison of the performance evolution of the pretrained architectures across training epochs

Unfortunately, none of the tested pretrained architectures provided any improvement over the baseline model. In fact, none of them achieved a balanced accuracy above 55% over the explored range of epochs, which indicates a complete absence of discriminative power.

These poor results might be due to several reasons: i) the very limited size of the dataset might make it difficult for such large pretrained models to adapt effectively without overfitting or underfitting, ii) the significant domain mismatch between the images typically used to pretrain these architectures and the geometric map representations used in this work might result in a poor transfer to the present task, and iii) it is also possible that the fixed hyperparameter configuration, inherited from the baseline model, is not suitable at all for these more complex architectures.

Evaluation of the pretrained architectures. Once again, to obtain a more accurate evaluation of the actual potential of each of these pre-trained architectures, the same nested cross-validation procedure was used. The ranges of hyperparameters in the grid space were identical to the ones used for the baseline, shown in Table 4.6.

Figure 4.11 shows the validation and test performances obtained from this second set of experiments, as well as the corresponding computation time for each of the four pretrained architectures. The reported values correspond to the mean balanced accuracy across seeds, with the standard deviation indicated in parentheses.

Models	Val. BA (%)	Test BA (%)	Time (s.)
Simple CNN	67.5 ($\pm$ 2.88)	61.7 ($\pm$ 11.7)	5785
EfficientNetB0	57.0 ($\pm$ 3.53)	49.2 ($\pm$ 9.43)	12260
DenseNet121	59.9 ($\pm$ 3.5)	44.17 ($\pm$ 16.2)	14153
ResNet50	58.7 ($\pm$ 1.9)	47.5 ($\pm$ 16.2)	15853
VGG16	63.0 ($\pm$ 4.6)	46.7 ($\pm$ 11.5)	19440

Table 4.11: Validation and test performances obtained for each pretrained architectures

The results obtained support even more strongly the idea that architectural complexity seems to degrade the performance while significantly increasing the computational cost. EfficientNetB0, DenseNet121, ResNet50, and VGG16 all performed slightly below random-level balanced accuracy. None of these pretrained architectures managed to achieve any meaningful discriminative power at all. These results suggest then that, although no pretrained architecture closer to the data was found, transfer learning from natural images is not well suited to the present data. In this context, a small, not too complex CNN architecture appears more appropriate.

4.3.4 Influence of input representation

In addition to the architectural comparisons, several experiments were conducted to evaluate the influence of the input representation on the models' performances. As mentioned before, all instruments originally have slightly different dimensions. In all previous experiments, the *relative* resizing strategy of the data was used, where all instruments are interpolated to match a common reference size. In the following experiments, all models will be compared using the balanced accuracy obtained through the nested CV procedure.

Resizing strategy: absolute vs. relative

The resizing strategy (absolute vs. relative) of the input representation was first investigated. As a reminder, the relative representation consists here of interpolating

each instrument so that all inputs have the exact same size as the largest dimensions of the dataset. In contrast, the absolute representation uses zero-padding to match the largest dimensions. This means that the absolute approach preserves the exact original proportions of each instrument but creates larger regions of padded zero values, while the relative approach reduces that amount of padding at the cost of slightly distorting the geometry through interpolation.

To perform this analysis, the baseline model SimpleCNN was used. Two input configurations were considered: relative resizing, corresponding to the baseline setting, and absolute resizing.

Reduced grids

The next sets of experiments conducted focused on specific regions of the instrument, rather than using the full input, could improve the model’s performance. This idea was motivated by previous observations on the resampled β profiles, which appeared to be particularly informative in the SVM approach proposed in [BCG25], and consequently in the MKL approach developed in this work. These results suggest that certain areas of the instrument may carry more discriminative information than others. Consequently, restricting the input to subregions could reduce computational cost while potentially maintaining or even improving the classification performance.

To explore this hypothesis, the input grids were first simply split into two halves along the horizontal axis, resulting in “top” and “bottom” regions of the sound boards. Three relative-resizing configurations were therefore considered: full sound board (baseline), bottom half only, and top half only. The idea was to evaluate the possibility of reducing the amount of input data while maintaining a similar (or better) level of classification performance, and to check whether relevant discriminative information appears to be sufficiently present in both the top and bottom regions individually, or not.

Resampled grids

Another type of input representation investigated was the use of resampled grids of a specific subregion of the violins. More precisely, the bottom of the sound board, illustrated in Figure 4.2, were used and resampled as input data. This specific subregion was selected because of the promising results obtained in [BCG25], where contour lines of the sound board seemed to be the most indicative. In contrast to the reduced grids, which aimed at targeting larger unspecific regions of the instruments, these experiments were conducted to evaluate whether reducing the spatial resolution of the promising subregion [BCG25] of violins itself could impact the model’s performance. In other words, instead of selecting half subregions, the entire specifically-selected region was uniformly downsampled to smaller sizes.

Behind these experiments were two main motivations. First, reducing the resolution obviously decreases the amount of input data, and therefore makes the computational cost and time significantly lower. Second, it allows to verify how much fine-grained geometric details are necessary for the classification task, and if less detailed representations are sufficient to capture the same amount of distinction.

The initial size of the selected region of the data, corresponding to the red rectangle of Figure 4.2, was roughly (500, 1000) for the sound boards. This common bounding box for all instruments was selected using the absolute approach described in [BCG26].

Seven resampling sizes were then tested: (5, 10), (10, 50), (10, 100), (50, 100), (100, 250), (250, 500), and (500, 1000).

Evaluation of all input representations

This set of experiments aimed at comparing and evaluating the potential performance of all the different input representations considered in this work (resizing strategies, reduced grids, and resampled grids) when used with the SimpleCNN architecture. In order to obtain the least biased assessment possible, the same nested cross-validation procedure as previously described was employed. This consists of an outer leave-one-out cross-validation (LOOCV) loop for evaluation, combined with an inner Stratified K-Fold procedure with four splits for hyperparameter tuning. The maximal number of training epochs was fixed to 100. For each input representation, the complete procedure is repeated five times, using the same 10 fixed combinations of hyperparameters selected from the explored search space, ensuring that each representation was evaluated under consistent and controlled conditions for the fairest comparison.

Both validation and test performances, as well as the computation times associated with each input representation, are reported in Table 4.12. The validation and test performances that outperformed the baseline are highlighted in bold.

Input representations	Val. BA (%)	Test BA (%)	Time (s.)
Baseline	67.5 ($\pm$ 2.88)	61.7 ($\pm$ 11.7)	5785
Resizing (absolute)	67.0 ($\pm$ 2.7)	58.5 ($\pm$ 6.6)	5892
Reduced (top half only)	68.7 ($\pm$ 1.2)	51.7 ($\pm$ 21.3)	4220
Reduced (bottom half only)	70.2 ($\pm$ 0.9)	61.7 ($\pm$ 8.8)	4266
Resampled (500, 1000)	64.4 ($\pm$ 2.6)	60.8 ($\pm$ 3.8)	3487
Resampled (250, 500)	68.0 ($\pm$ 2.2)	65.0 ($\pm$ 10.9)	2144
Resampled (100, 250)	66.3 ($\pm$ 0.1)	41.2 ($\pm$ 1.8)	1134
Resampled (50, 100)	65.4 ($\pm$ 1.4)	51.7 ($\pm$ 7.2)	1246
Resampled (10, 100)	62.3 ($\pm$ 0.5)	57.5 ($\pm$ 7.5)	1160
Resampled (10, 50)	61.8 ($\pm$ 1.3)	35.8 ($\pm$ 5.8)	1164
Resampled (5, 10)	67.5 ($\pm$ 4.6)	55.0 ($\pm$ 6.6)	1195

Table 4.12: SimpleCNN performances for each explored input representation

Multiple insights can be observed from Table 4.12. First, two input representations stand out in terms of overall performance: the reduced grid focusing on the bottom half of the instrument, and the resampled grid with resolution (250, 500). Both configurations match or outperform the baseline in terms of test balanced accuracy, while also improving stability, as reflected by lower standard deviations. The reduced bottom-half representation achieved the same test performance as the baseline (61.7%) but with a lower variability across seeds. At the same time, it reduces the computational cost by approximately 26%. The resampled (250, 500) configuration goes one step further by improving the test performance to 65.0%, while also reducing both the variance and the computation time significantly (more than 60% reduction compared to the baseline). This might suggest that the initial input data might contain too much irrelevant information and noise, and that an appropriate level of downsampling (through moderate resampling) and/or reduction (through subregions) helps the model focus on more relevant patterns.

In contrast, the top-half reduced representation, although achieving a higher validation performance, exhibits a strong degradation and high variability on the test set. This can be simply explained by the fact that the top-half of the sound board contains less information due to the presence of the instrument neck. Similarly, overly aggressive resampling (e.g., (100, 250) or lower) leads to a significant drop in performance, indicating that too much information is lost when the resolution becomes insufficient. Interestingly, the absolute resizing strategy also performed worse than the relative baseline, confirming previous observations that zero-padding might introduce less informative representations for convolutional models.

4.3.5 Effect of data augmentation

The final set of experiments was conducted in order to evaluate the impact of data augmentation on the performance of the SimpleCNN model. Given the very limited size of the dataset, data scarcity represents one of the main constraints of this context. It is therefore of particular interest to assess whether artificially increasing the diversity of the training data through augmentation techniques could improve the generalization capabilities of the model.

To this end, three types of augmentations were explored, as described in section 5.2.2, including rotations, horizontal mirroring, as well as horizontal and vertical shifts. These transformations do preserve the overall geometric structure of the input while introducing controlled variability in the training samples. The effect of these augmentations was evaluated under the same experimental protocol as previously described: the baseline hyperparameters were kept fixed, and a LOOCV procedure was used for evaluation.

Five different augmentation strategies were considered. The first one consists of applying only horizontal mirroring to each instrument, doubling the dataset size ($\times 2$). The second strategy applied two random horizontal and vertical shifts per instrument, increasing the dataset size by a factor of three ($\times 3$). The third strategy used a single slight rotation per instrument, also resulting in a dataset size doubled ($\times 2$). The fourth strategy applied four slight random rotations per instrument, leading to a five-fold increase in dataset size ($\times 5$). Finally, a combined strategy was tested, consisting of horizontal mirroring, two random shifts, and one rotation per instrument, resulting in a sixteen fold increase in dataset size ($\times 16$).

Figure 4.10 shows the mean performances according to the number of epochs, obtained for each of these five strategies and averaged across 5 seeds. The baseline (without augmentation) is also displayed and corresponds to the black curve. The standard deviation associated with each curve is also shown.

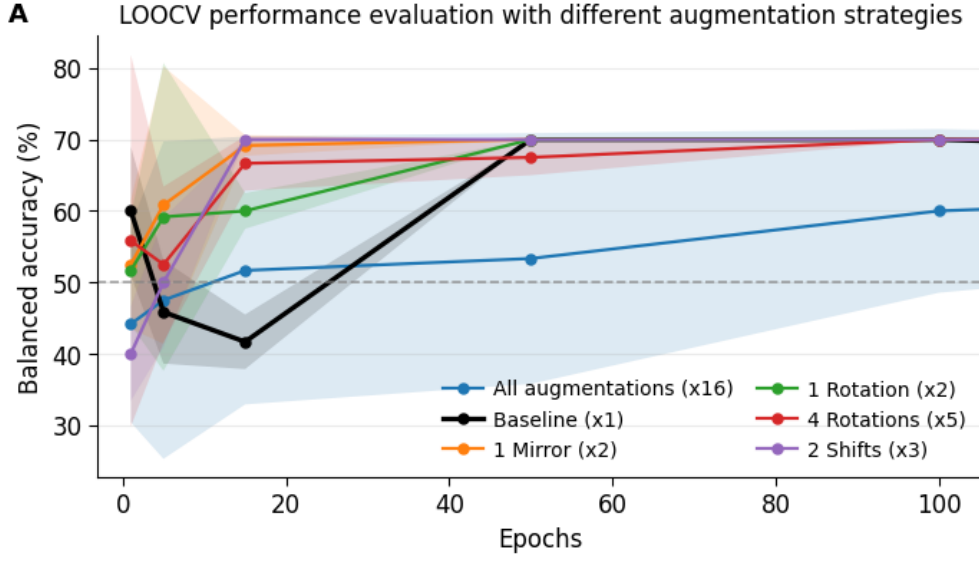


Figure 4.10: Evolution of performance according to the number of epochs, for different augmentation strategies.

These results show that all augmentation strategies eventually converge towards a similar performance level, around 70% balanced accuracy. However, the speed at which this plateau is reached varies depending on the augmentation strategy. The mirroring-only and shift-only configurations appear to reach this performance relatively quickly, already at around epoch 15. In contrast, strategies involving rotations (both with 1 rotation and 4 rotations) require slightly more epochs before achieving comparable performance levels. When combining all augmentation techniques, convergence becomes even slower. These observations suggest that while data augmentation does not significantly improve the final performance in this context, it affects the learning dynamics by slowing down convergence and increasing variability across seeds. This behavior may be explained by the increased variability introduced in the training data, which requires the model to learn more general representations before stabilizing.

4.3.6 Overall comparison and discussion

Table 4.13 summarizes the main results obtained throughout this CNN chapter. Overall, these experiments suggest that while the CNN approach seems to be able to extract some discriminative information from elevation maps, its performance remains quite moderate compared to the performances obtained on previous SVM-based approaches. The best overall results were obtained with the SimpleCNN architecture using the resampled grid of size (250, 500), reaching a test balanced accuracy of 65.0%, and the reduced bottom-half grid that reached the same test performance as the baseline, but with a lower standard deviation and a reduced computation time. This result supports the idea that the most useful discriminative information may not require the full surface of the instrument. In fact, using the complete grid seems to introduce a lot of less informative regions or even redundancy, which seems to result in poorer learning, both in terms of performance and stability, and way longer computational times.

Architecture	Input repr.	Val. BA (%)	Test BA (%)	Time (s.)
SimpleCNN	Relative, full grid	67.5 ($\pm$ 2.88)	61.7 ($\pm$ 11.7)	5785
CNN5	Relative, full grid	67.8 ($\pm$ 0.6)	57.5 ($\pm$ 2.5)	9991
BatchNorm	Relative, full grid	62.8 ($\pm$ 3.2)	55.0 ($\pm$ 2.5)	8847
Multiscale	Relative, full grid	60.5 ($\pm$ 5.7)	50.0 ($\pm$ 14.1)	16401
EfficientNetB0	Relative, full grid	57.0 ($\pm$ 3.53)	49.2 ($\pm$ 9.43)	12260
DenseNet121	Relative, full grid	59.9 ($\pm$ 3.5)	44.17 ($\pm$ 16.2)	14153
ResNet50	Relative, full grid	58.7 ($\pm$ 1.9)	47.5 ($\pm$ 16.2)	15853
VGG16	Relative, full grid	63.0 ($\pm$ 4.6)	46.7 ($\pm$ 11.5)	19440
SimpleCNN	Absolute, full grid	67.0 ($\pm$ 2.7)	58.5 ($\pm$ 6.6)	5892
SimpleCNN	Relative, top half	68.7 ($\pm$ 1.2)	51.7 ($\pm$ 21.3)	4220
SimpleCNN	Relative, bottom half	70.2 ($\pm$ 0.9)	61.7 ($\pm$ 8.8)	4266
SimpleCNN	Relative, 500 $\times$ 1000	64.4 ($\pm$ 2.6)	60.8 ($\pm$ 3.8)	3487
SimpleCNN	Relative, 250 $\times$ 500	68.0 ($\pm$ 2.2)	65.0 ($\pm$ 10.9)	2144
SimpleCNN	Relative, 100 $\times$ 250	66.3 ($\pm$ 0.1)	41.2 ($\pm$ 1.8)	1134
SimpleCNN	Relative, 50 $\times$ 100	65.4 ($\pm$ 1.4)	51.7 ($\pm$ 7.2)	1246

Table 4.13: Overall performances of different architectures and input representations

In particular, the bottom part of the sound board seems to preserve enough relevant information for the classification task. This is coherent with the previous observations made from the contour-line and β -profile analyses, where specific regions of the instrument seemed especially informative. A moderate downsampling of specific regions of the instruments might also be beneficial, not only because it reduces the computational cost, but also because it may help the model focus on more stable geometric structures. Also, the fact that the absolute resizing strategy performed slightly worse than with the relative baseline supports the idea that preserving the original proportions of the instruments might be less important in this context than helping the model to focus on very specific regions.

The architectural comparison also gave a clear result: increasing the model complexity did not improve performance at all. The deeper CNN5 architecture, the BatchNorm variant, and the Multiscale CNN all performed worse than the SimpleCNN baseline, while requiring more computation time. In this setting, with so little data and so many features per data, it seems like making the models more complex by adding layers, or normalization mechanisms may only increase the difficulty of learning stable patterns instead of improving generalization. The same conclusion applies even more strongly to pretrained architectures. EfficientNetB0, DenseNet121, ResNet50, and VGG16 all performed slightly below random-level balanced accuracy. These results suggest that transfer learning from those natural images is not well adapted to the present data. The elevation maps used here are very different from those natural images, both in visual structure and in semantic content. In this context, a small, not too complex CNN architecture appears more appropriate.

The data augmentation experiments also provide useful insight. Although augmentation strategies such as mirroring, shifts, and rotations eventually reached similar performance levels, they did not clearly improve the final classification performance. Instead, they mainly affected the learning dynamics, often slowing down convergence and increasing variability. This may indicate that the available dataset is too small for augmentation alone to compensate for the lack of real instruments. It is also possible that some transformations introduce geometric variations that are not fully meaningful for the reduction problem.

Overall, the CNN approach confirmed that elevation maps contain some useful information for distinguishing reduced and non-reduced instruments, but these alone do not seem sufficient to easily achieve consistently accurate classification across all instruments through CNN. The best configurations reached 65% balanced accuracy

with the LOOCV procedure, which shows that the model can learn something beyond random classification, but not enough to provide a robust standalone classification tool. The sometimes high variability across seeds and input representations also highlights the sensitivity of CNNs in this very small-data setting. The main conclusion of this approach is therefore that CNNs are promising as an exploratory approach, but not yet sufficient as the primary classification method for this task. Their main advantage and motivation was their ability to exploit almost raw geometric data without requiring handcrafted contour descriptors, but in this context this resulted in higher computational cost, greater instability, and lower overall performance. In contrast, simpler representations based on contour-line parameters seem to remain more interpretable and better aligned with the known geometric effect of reduction.

Chapter 5

Multi-Layer Perceptron (MLP) approach

5.1 Motivations

This MLP approach can be seen as an intermediate strategy between the two previously explored approaches. The motivation behind this choice is to combine the strengths of both approaches: using the best of the learning capacity and flexibility of neural networks, similarly to what has been done in the CNN-based approach, but relying on engineered input representations of geometric descriptors, introduced in [BCG25], and already used in the MKL approach. This first approach gave high, promising results so using the same inputs to feed a neural network seems like a natural extension to explore.

It should be noted that, due to time constraints, this approach has been less extensively investigated than the previous ones. The experiments conducted should therefore be seen as an initial exploratory analysis to gain insight into its potential, rather than a complete and exhaustive set of experiments and evaluation.

5.2 Algorithm and architecture

5.2.1 Input representation

The input representation used for this approach is identical to the one used in the MKL experiments. It consists of the resampled values of the four contour parameters α , β , δ , and γ , computed for each of the three instrument zones: sound board, back bottom, and back top. Each parameter is represented by a fixed number of resampled points, resulting in 50 values per parameter. This leads to a total of, at most, 600 input values (12×50) when all parameters and zones are used, compared to the CNN approach, which used high-resolution elevation maps containing a bit less than 2 million values (1000×1800). These 600 values are concatenated into a single one-dimensional feature vector, which is used as input for the MLP model.

5.2.2 MLP architectures

Similarly to the customized convolutional neural networks explored in the previous approach, three simple and customized MLP architectures were explored: namely

Simple3, Simple4, and BatchN. These models differ mainly in their depth and the use of normalization layers, but each of them keeps a relatively simple design.

Simple3 MLP. The first and simplest MLP model investigated is composed of two hidden dense layers followed by the final classification layer. The two hidden layers contain respectively 128 and 64 units, both using ReLU activation and L_2 regularization. A dropout layer is also used after each hidden layer. The final dense layer contains 2 output units corresponding to the two classes. Figure 5.1 shows the architecture in more detail.

Simple3 MLP architecture
• Dense layer with 128 units, ReLU activation, and L_2 regularization • Dropout layer • Dense layer with 64 units, ReLU activation, and L_2 regularization • Dropout layer • Dense layer with 2 output units

Table 5.1: Simple3 MLP architecture

Simple4 MLP. The Simple4 architecture extends Simple3 by adding one additional hidden layer and increasing the number of units in the first layers. It is therefore composed of three dense hidden layers with 256, 128, and 64 units respectively, each using ReLU activation and L_2 regularization. As in the Simple3 model, dropout is applied after each hidden layer. Figure 5.2 shows the architecture in more details.

Simple4 MLP architecture
• Dense layer with 256 units, ReLU activation, and L_2 regularization • Dropout layer • Dense layer with 128 units, ReLU activation, and L_2 regularization • Dropout layer • Dense layer with 64 units, ReLU activation, and L_2 regularization • Dropout layer • Dense layer with 2 output units

Table 5.2: Simple4 MLP architecture

BatchN MLP. The BatchN architecture follows the same general depth as Simple4, with three hidden dense layers of 256, 128, and 64 units. However, similarly to the BatchN CNN architecture, each dense layer is followed by a batch normalization layer before the ReLU activation. Dropout is then applied after the activation. Figure 5.3 shows the architecture in more details.

BatchN MLP architecture
• Dense layer with 256 units, no bias, and L_2 regularization • Batch normalization layer • ReLU activation layer • Dropout layer • Dense layer with 128 units, no bias, and L_2 regularization • Batch normalization layer • ReLU activation layer • Dropout layer • Dense layer with 64 units, no bias, and L_2 regularization • Batch normalization layer • ReLU activation layer • Dropout layer • Dense layer with 2 output units (logits)

Table 5.3: BatchN MLP architecture

5.2.3 Training and evaluation

This approach follows the same training and evaluation procedure as the one used for the CNN approach. A nested cross-validation strategy is used, combining an outer LOOCV loop for evaluation with an inner Stratified K-Fold loop for hyperparameter tuning. The whole procedure and algorithm are more thoroughly described in Section 5.2.4. The hyperparameters considered here are also the same as in the CNN approach, including learning rate, dropout rate, regularization strength, batch size, and weight decay. The only exception is the convolution kernel size, that is obviously not used in the context of MLP architectures. The number of epochs was treated the same way as for the CNN approach. For each configuration, the validation balanced accuracy was evaluated at different epochs, and the epoch giving the best validation performance was selected. This selected epoch was then used for the final prediction of the held-out test instrument.

5.3 Experiments and results

5.3.1 First experiment overview

The first experiments were conducted to get an initial assessment of the performance achievable with such MLP architectures in this context. The three models described in subsection 6.2.2 were compared with an inner 3-Fold CV for hyperparameter tuning. The number of random combinations of hyperparameter value within each fold was fixed to 10.

Evaluation of the three architectures with `max_epoch=100`. The maximal number of epoch was first set to 100 according to the behaviors observed in the previous CNN experiments. More precisely the epochs tested were: 1, 5, 15, 50, and 100. The results are reported in Table 5.4. The column *Val. BA* represents the mean of the highest validation balanced accuracy obtained within each inner K-Fold loop, while *Test BA* corresponds to the final performance computed through the outer LOOCV procedure. To ensure fair comparisons and account for variability due to random initialization, these experiments were repeated across 10 different seeds

using the same randomly selected hyperparameter combinations. The reported values therefore include the corresponding standard deviations.

Models	Val. BA (%)	Test BA (%)	Time (s.)
Simple3	82.1 ($\pm$ 5.8)	68.0 ($\pm$ 5.7)	3392
Simple4	82.3 ($\pm$ 6.7)	66.7 ($\pm$ 5.7)	4287
BatchN	82.2 ($\pm$ 5.6)	62.5 ($\pm$ 10.0)	5963

Table 5.4: Performances of the first experiment, with max_epoch=100

The three architectures gave very similar performances both in terms of validation and test balanced accuracy. This is consistent with their design, as they are only close variations of each other. The main noticeable difference is with the BatchN architecture, which resulted in slightly lower test performance and with higher standard deviation, suggesting less stability across different runs and less performance.

A clear increase in computational time can also be observed. The BatchN model, being the most complex due to the addition of normalization layers, requires significantly more time than the simpler architectures. Given these observations, the Simple3 model appears to offer the best trade-off between performance and computational efficiency, as it achieves very similar results while requiring the least training time.

That being said, all three architectures show a noticeable gap between validation and test performances. Such a difference may suggest overfitting caused by a maximum number of training epochs that is too high.

5.3.2 Effect of number of epoch

To investigate the hypothesis of overfitting due to a max_epoch too high, multiple experiments were conducted with max_epoch set respectively to 1, 5, 15, 25, and 50. Figure 5.1 shows the results of these experiments averaged over 5 seeds each. Each color represents an architecture, with dotted curves as validation performance, and plain curve as test performance.

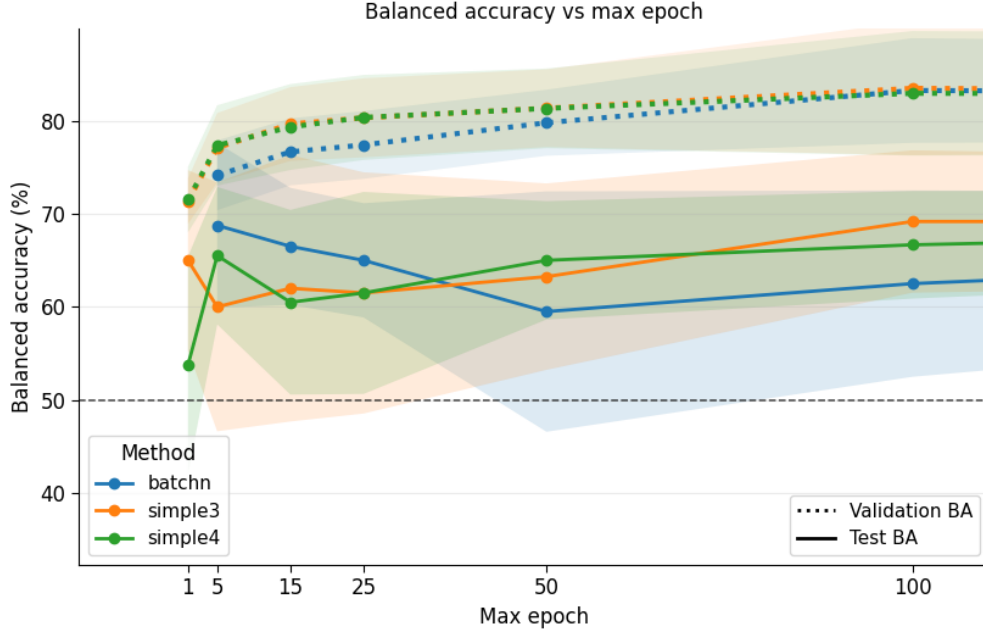


Figure 5.1: Evolution of performance according to the maximal number of epoch for the three explored MLP architectures

These results indicate that, for each of the three architectures, the validation balanced accuracy consistently remains higher than the test balanced accuracy, no matter the chosen maximal number of epochs. For the Simple3 and Simple4 architectures, the gap between validation and test performance appears relatively stable across all tested configurations, with a difference of approximately 10% in balanced accuracy. In contrast, the BatchN architecture shows a slightly better performance at lower numbers of epochs (up to around 25), but then underperformed compared to the other architectures as the number of epochs increases.

These observations suggest that the initial hypothesis is not supported by the experimental results. The difference between validation and test balanced accuracy does not seem to be caused by an excessively large number of training epochs. Increasing the number of epochs does not lead to a noticeable degradation of test performance relative to validation performance, which would typically be expected in the presence of overfitting.

Note that additional experiments were conducted with a significantly larger number of epochs (up to 300). These experiments did not reveal any further evolution of the results compared to those obtained with 100 epochs. Both validation and test balanced accuracies remained essentially constant, as if the training process had already converged.

5.3.3 Effect of inner K-Fold strategy

Given the observations from the previous experiments, an additional experiment was conducted to investigate whether the number of splits used in the inner Stratified K-Fold procedure could influence the observed gap between validation and test per-

formances.

Theoretically, increasing the number of inner folds should provide a more reliable estimate of the validation performance for each hyperparameter configuration, as it reduces the variance associated with each single train/validation split. This could potentially lead to better hyperparameter selection at the cost of a higher computational time, since the number of model trainings increases proportionally with the number of folds.

Figure 5.2 shows the results obtained from these experiments. Once again, each color represents one architecture, the dotted curves represent the average validation performances recorded, while plain curves represent the test performances.

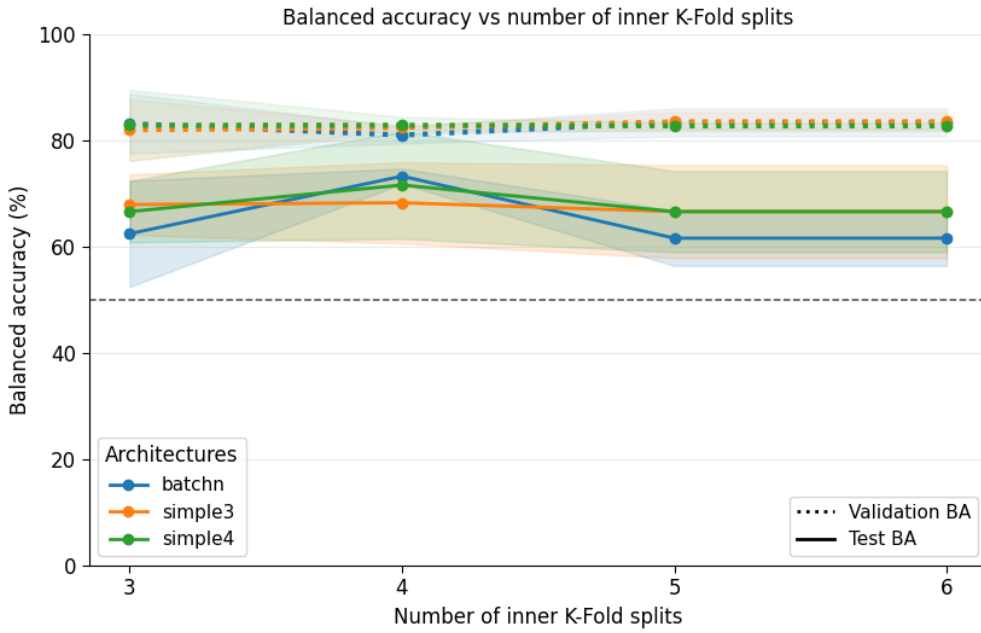


Figure 5.2: Evolution of performance according to the number of inner K-Fold splits for the three explored MLP architectures

The experimental results show that varying the number of inner folds has only a very limited impact on the final performances. While small variations can be observed in both validation and test balanced accuracies, no clear trend indicates a systematic improvement when increasing the number of folds. The gap between validation and test performances remains present regardless of the chosen number of splits.

These results suggest that the inner cross-validation strategy is not the primary factor explaining the difference between validation and test performances. Although using more folds may slightly stabilize hyperparameter selection, it does not significantly improve generalization in this context. Consequently, a moderate number of folds appears sufficient to balance computational cost and performance reliability.

5.3.4 Overall comparison and discussion

Overall, the MLP approach seems to represent a promising and encouraging intermediate strategy, but with still moderate results. Among the three architectures tested, the Simple3 model appeared to be the most appropriate choice. It obtained the highest test balanced accuracy, with 68.0%, while also requiring the lowest computation time.

The comparison between validation and test performances also revealed an important limitation of the approach. All three architectures reach validation balanced accuracies around 82%, but their test performances remain between approximately 62% and 68%. This persistent gap indicates that the selected hyperparameters perform well on the inner validation folds, but that this does not entirely translate to unseen instruments. Additional experiments showed that this gap is not mainly caused by the maximum number of training epochs, since reducing or increasing the number of epochs did not substantially change the results. Similarly, changing the number of inner K-Fold splits did not clearly improve generalization.

Compared with the CNN approach, the MLP approach gives slightly better results while requiring less computational cost. Using engineered contour-line descriptors seems more effective than training directly on almost raw elevation maps, at least with the current dataset. The MLP therefore confirms the relevance of the parametric contour representation introduced in previous work and reused in the MKL approach. It might also suggest that the main discriminative information is likely already present in these extracted geometric profiles. However, the MLP approach remains less interpretable than the MKL or SVM-based approaches. While it uses meaningful input features, the decision process is distributed across dense layers, making it much more difficult to identify which parameter, zone, or contour level is the most important for the classification. This limits its usefulness as an interpretability tool.

Conclusion

Summary and results of the different approaches

This thesis explored three machine learning approaches for the classification of violin instruments based on geometric features extracted from contour lines, and on elevation maps. These three approaches are Multiple Kernel Learning (MKL), which can be seen as an extension of the previous SVM approach from [BCG25], Convolutional Neural Networks (CNNs), and Multi-Layer Perceptrons (MLPs).

First, the MKL approach was able to classify instruments with extremely high performance. In both the linear and RBF versions, several configurations of input features reached 100% balanced accuracy. This approach also offered strong interpretability and supported the hypothesis from previous work [BCG25], highlighting the importance of specific geometric descriptors, such as the importance of the β parameter, and the complementary role of δ . Moreover, it showed that perfect performance is still attainable after removing several individual components. This indicated that the signal of reduction is redundant and not confined to a single zone or parameter. Nevertheless, similarly to the SVM approach in [BCG25], this approach remained extremely sensitive to the regularization hyperparameter C . A slight change in the value of C moved the model from perfect classification to complete failure. This should not be overlooked as it may suggest poor generalization and high sensitivity to the data, even though procedures such as LOOCV and nested CV were used to mitigate overfitting.

Second, while the CNN approach seemed to be able to extract some discriminative information from elevation maps, the performance obtained remained very moderate. The best performances were obtained using the resampled grid of size (250, 500) as input data, giving a balanced accuracy of 65.0%. These experiments clearly demonstrated that using the complete elevation maps as input in this context introduced a lot of unnecessary noise and uninformative regions, overall leading to poorer classification. In that sense, increase architecture complexity only made the performance and the computation time worse. These experiments also confirmed that, given the extremely small dataset and the extremely small discriminative information potentially contained in the input, using a simple CNN is not enough to obtain high performance classification.

Finally, the MLP approach, based on both previous approaches, seems to be a promising intermediate, but with still moderate results. The highest BA obtained was 68.0%, slightly higher than what was obtained with the CNN approach, while also requiring significantly less computation time. That confirmed the idea that the detailed elevation maps, used as input data for the baseline CNN architecture, only

made the classification task harder. It also suggests that the main discriminative information is likely already present in these extracted geometric profiles.

Limitations and future work

A first major limitation is obviously the size of the dataset. With only 25 instruments available, the learning process remains highly constrained, regardless of the chosen methodology. This very small sample size increases the risk of overfitting and limits the ability to draw strong conclusions about generalization performance. Although validation strategies such as LOOCV and nested cross-validation were used to mitigate these issues, they cannot fully compensate for the lack of data. Extending the dataset, either by acquiring new instruments or by incorporating additional types of bowed instruments such as cellos, could be a very useful step toward more robust and reliable models.

A second limitation concerns the sensitivity of certain approaches to hyperparameters. As mentioned before, the MKL (and more generally SVM-based) approach showed a strong dependence on the regularization parameter C , where small variations could lead to completely different results. This instability raises important concerns about the robustness of these models and suggests that further work should study the stability of such models.

Finally, another important aspect is related to the choice of input representations. The experiments conducted in this thesis showed that using full-resolution elevation maps introduces a lot of noisy information, making the learning task very difficult. While reduced grids and resampled representations improved both efficiency and performance, the selection of relevant regions in this work remained relatively limited. Future work could investigate more targeted approaches, such as focusing more deeply on geometrically meaningful regions.

Bibliography

- [AD15] Fabio Aioli and Michele Donini. “EasyMKL: A Scalable Multiple Kernel Learning Algorithm”. In: *Neurocomputing* 169 (2015), pp. 215–224. ISSN: 0925-2312. DOI: 10.1016/j.neucom.2014.11.078. URL: <https://www.sciencedirect.com/science/article/pii/S0925231215003653>.
- [Alm] *A Comprehensive Guide to Support Vector Machines (SVMs)*. AlmaBetter. URL: <https://www.almabetter.com/bytes/articles/support-vector-machines>.
- [BCG24] Philémon Beghin, Anne-Emmanuelle Ceulemans, and François Glineur. “A Discussion About Violin Reduction: Geometric Analysis of Contour Lines and Channel of Minima”. In: *The Future of Heritage Science and Technologies II*. Ed. by Rocco Furferi et al. Cham: Springer Nature Switzerland, 2024, pp. 432–446. ISBN: 978-3-031-98379-5. DOI: 10.1007/978-3-031-98379-5_33.
- [BCG25] Philémon Beghin, Anne-Emmanuelle Ceulemans, and François Glineur. “Identification of Violin Reduction via Contour Lines Classification”. In: *Digital Heritage* (2025). DOI: 10.2312/dh.20253083. arXiv: 2507.07743 [cs.AI]. URL: <http://arxiv.org/abs/2507.07743>.
- [BCG26] Philémon Beghin, Anne-Emmanuelle Ceulemans, and François Glineur. *From Elevation Maps To Contour Lines: SVM and Decision Trees to Detect Violin Width Reduction*. Version 1. 2026. DOI: 10.48550/ARXIV.2604.02446. URL: <https://arxiv.org/abs/2604.02446>. Pre-published.
- [Beg+23] Philémon Beghin et al. “Validation of a Photogrammetric Approach for the Objective Study of Early Bowed Instruments”. In: *Heritage Science* 11.1 (Aug. 11, 2023), p. 170. ISSN: 2050-7445. DOI: 10.1186/s40494-023-00979-4. URL: <https://www.nature.com/articles/s40494-023-00979-4>.
- [Beg21] Philémon Beghin. “A Digital Tool at the Service of Organology : Validation of a Photogrammetric Approach”. Master’s thesis. UCLouvain, 2021. HDL: 2078.2/23214. URL: <https://hdl.handle.net/2078.2/23214>.
- [Bra+23] Tyler J. Bradshaw et al. “A Guide to Cross-Validation for Artificial Intelligence in Medical Imaging”. In: *Radiology: Artificial Intelligence* 5.4 (July 1, 2023), e220232. ISSN: 2638-6100. DOI: 10.1148/ryai.220232. URL: <http://pubs.rsna.org/doi/10.1148/ryai.220232>.
- [Bri+24] Mitja Briscik et al. “Supervised Multiple Kernel Learning Approaches for Multi-Omics Data Integration”. In: *BioData Mining* 17.1 (Nov. 23, 2024), p. 53. ISSN: 1756-0381. DOI: 10.1186/s13040-024-00406-9. URL: <https://biodatamining.biomedcentral.com/articles/10.1186/s13040-024-00406-9>.

- [GA11] Mehmet Gönen and Ethem Alpaydin. “Multiple Kernel Learning Algorithms”. In: *Journal of Machine Learning Research* 12.64 (2011), pp. 2211–2268. URL: <http://jmlr.org/papers/v12/gonen11a.html>.
- [He+15] Kaiming He et al. *Deep Residual Learning for Image Recognition*. Dec. 10, 2015. DOI: 10.48550/arXiv.1512.03385. arXiv: 1512.03385 [cs.CV]. URL: <http://arxiv.org/abs/1512.03385> (visited on 05/22/2026). Pre-published.
- [Hua+17] Gao Huang et al. “Densely Connected Convolutional Networks”. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). Honolulu, HI: IEEE, July 2017, pp. 2261–2269. ISBN: 978-1-5386-0457-1. DOI: 10.1109/CVPR.2017.243. URL: <https://ieeexplore.ieee.org/document/8099726/>.
- [Ibm] *What Are Convolutional Neural Networks? — IBM*. IBM. URL: <https://www.ibm.com/think/topics/convolutional-neural-networks>.
- [IS15] Sergey Ioffe and Christian Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: *Proceedings of the 32nd International Conference on Machine Learning*. Ed. by Francis Bach and David Blei. Vol. 37. Proceedings of Machine Learning Research. Lille, France: PMLR, July 7–9, 2015, pp. 448–456. URL: <https://proceedings.mlr.press/v37/ioffe15.html>.
- [Kag] *MNIST Dataset*. URL: <https://www.kaggle.com/datasets/hojjatk/mnist-dataset>.
- [Kar] *A Recipe for Training Neural Networks*. URL: <https://karpathy.github.io/2019/04/25/recipe/>.
- [Lum+24] Victor Lumumba et al. “Comparative Analysis of Cross-Validation Techniques: LOOCV, K-folds Cross-Validation, and Repeated K-folds Cross-Validation in Machine Learning Models”. In: *American Journal of Theoretical and Applied Statistics* 13.5 (Oct. 10, 2024), pp. 127–137. ISSN: 2326-9006, 2326-8999. DOI: 10.11648/j.ajtas.20241305.13. URL: <https://www.sciencepublishinggroup.com/article/10.11648/j.ajtas.20241305.13>.
- [Sri+] Nitish Srivastava et al. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”. In: ().
- [SZ15] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. Apr. 10, 2015. DOI: 10.48550/arXiv.1409.1556. arXiv: 1409.1556 [cs.CV]. URL: <http://arxiv.org/abs/1409.1556>. Pre-published.
- [Ten] *Classification on Imbalanced Data — TensorFlow Core*. TensorFlow. URL: https://www.tensorflow.org/tutorials/structured_data/imbalanced_data.
- [TL19] Mingxing Tan and Quoc Le. “EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks”. In: *Proceedings of the 36th International Conference on Machine Learning*. International Conference on Machine Learning. PMLR, May 24, 2019, pp. 6105–6114. URL: <https://proceedings.mlr.press/v97/tan19a.html>.

- [Vab+19] Andrius Vabalas et al. “Machine Learning Algorithm Validation with a Limited Sample Size”. In: *PLOS ONE* 14.11 (Nov. 7, 2019). Ed. by Enrique Hernandez-Lemus, e0224365. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0224365. URL: <https://dx.plos.org/10.1371/journal.pone.0224365>.
- [Xin+18] Lin Xing et al. “Improving Predictability of Multisensor Data with Non-linear Statistical Methodologies”. In: *Crop Science* 58.2 (Mar. 2018), pp. 972–981. ISSN: 0011-183X, 1435-0653. DOI: 10.2135/cropsci2017.09.0537. URL: <https://acsess.onlinelibrary.wiley.com/doi/10.2135/cropsci2017.09.0537>.
- [Zha+24] Xia Zhao et al. “A Review of Convolutional Neural Networks in Computer Vision”. In: *Artificial Intelligence Review* 57.4 (Mar. 23, 2024), p. 99. ISSN: 1573-7462. DOI: 10.1007/s10462-024-10721-6. URL: <https://link.springer.com/10.1007/s10462-024-10721-6>.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain

Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | www.uclouvain.be/epl