Machine Learning for Multi-Sensory Data

A DISSERTATION

SUBMITTED TO THE GRADUATE SCHOOL
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

for the degree

DOCTOR OF PHILOSOPHY

Field of Electrical Engineering and Computer Science

By

Neda Rohani

EVANSTON, ILLINOIS

December 2018
© Copyright by Neda Rohani 2018

All Rights Reserved
ABSTRACT

Machine Learning for Multi-Sensory Data

Neda Rohani

In this dissertation, we study different machine learning algorithms including probabilistic, sparse and deep learning based models applied to multi-sensory datasets. In many machine learning problems, samples are collected from more than one source or modality. Also, various feature extraction methods can be used to provide more than one set of feature vectors per sample. Such extra sources or feature vectors are referred to as “views”. Using multiple views can improve performance as they may provide complementary or redundant information. Integrating multiple sources of data is a challenging task.

First, we develop a new model using probabilistic modeling and variational Bayesian inference for multi-sensory data classification. A Gaussian Process (GP) classifier is built upon the introduced modeling. Its posterior distribution is approximated using variational Bayesian inference. Finally, test point labels are predicted employing this classifier.

In the next chapter, traditional and advanced machine learning techniques have been applied to classify non-cosmic, non-Gaussian disturbances known as “glitches”. Glitches show up in gravitational-wave data of the Advanced Laser Interferometer Gravitational-wave Observatory, or aLIGO. We also develop deep multi-view convolutional neural networks to
classify glitches automatically. Lastly, we present the algorithms for glitch image classification using multiple citizen labels and machine learning classifier outputs.

In the next chapter, first we study the problem of automatic pigment identification, then, pigment unmixing problem in hyperspectral paintings is explored. Here, we cast the problem of pigment identification in a novel way by decomposing the spectrum into pure pigments. With the intent of expanding the current data treatment of hyperspectral datasets, an innovative approach for data reduction and visualization is presented next. It uses a statistical embedding method known as t-SNE (t-Distributed Stochastic Neighbor Embedding) to provide a nonlinear representation of spectral features in a lower 2-dimensional space. Then, the problem of automatic nonlinear unmixing of hyperspectral reflectance data using works of art as test cases are described. Our approach is to interrogate a hyperspectral image cube by, first, decomposing it into a set of pure pigments and, second, to estimate the scattering and absorption coefficients of each individual pigment to produce semi-quantitative values for the component fractions. This is accomplished through deep neural networks (DNN) used to qualitatively identify the constituent pigments in an unknown spectrum. Based on the pigment(s) present in a given pixel, Kubelka-Munk theory is employed to estimate its semi-quantitative concentration. Next, we replace the gradient descent based approach for pigment unmixing task by a DNN which is a much faster approach.

In the last chapter, we argue that learning a hierarchy of features in a hierarchical dataset requires lower layers to approach convergence faster than layers above them. We show that, if this assumption holds, we can analytically approximate the outcome of stochastic gradient descent (SGD) for each layer. We find that the weights should converge to a class-based PCA, with some weights in every layer dedicated to principal components of each label class. The class-based PCA allows us to train layers directly, without SGD, often leading to a dramatic
decrease in training complexity. We demonstrate the effectiveness of this by using our results to replace one and two convolutional layers in networks trained on MNIST, CIFAR10 and CIFAR100 datasets, showing that our method achieves performance superior or comparable to similar architectures trained using SGD.
Table of Contents

ABSTRACT 3

Chapter 1. Introduction 9

1.1. Variational Gaussian Process for Multisensor Classification Problems 9
1.2. Machine Learning for Gravity Spy: Dataset and Glitch Classification 10
1.3. Machine Learning for Cultural Heritage 11
1.4. Convergence of Weights in Deep Neural Networks 13
1.5. Outline of the Dissertation 14

Chapter 2. Variational Gaussian Process for Multisensor Classification Problems 16

2.1. Introduction 16
2.2. Bayesian modeling 20
2.3. Variational inference 23
2.4. Classification rule 27
2.5. Related models 28
2.6. Experimental results 30
2.7. Conclusions 45

Chapter 3. Machine Learning for Gravity Spy: Dataset and Glitch Classification 46

3.1. Introduction 46
3.2. Gravity Spy Data 47
3.3. Classification Methods

3.4. Glitch Classification using Machine Learning and Citizen Science

3.5. Conclusions

Chapter 4. Machine Learning for Cultural Heritage

4.1. Introduction

4.2. Automatic Pigment Identification on Roman Egyptian Paintings by using Sparse Modeling of Hyperspectral Images

4.3. Innovative data reduction and visualization strategy for hyperspectral imaging datasets using t-SNE approach


4.5. Pigment unmixing of hyperspectral images of paintings using deep neural networks

4.6. Conclusions

Chapter 5. Convergence of Weights in Deep Neural Networks

5.1. Introduction

5.2. Stochasticity and the learning dynamics

5.3. Setup and notation

5.4. Fast drift phase and singular values of $w^{(k)}$

5.5. Conditions for Decoupling of Layer Dynamics during SGD

5.6. Relaxation phase

5.7. Evolution of covariance and density matrix

5.8. Training Layers Using Class-based PCA
5.9. Simple PCA layers 187
5.10. Conclusion 190
Chapter 6. Conclusions 192
References 194
CHAPTER 1

Introduction

In this dissertation, we apply different traditional and advanced machine learning techniques including probabilistic, sparse and deep learning based models applied to multi-sensory data scenario.

Fusion of multiple sources of information has been used in many applications such as emotion recognition [70], recommendation systems [156], speech recognition [14, 80], and biometric verification [9]. Integrating multiple sources of data is a challenging task, and various approaches have been proposed in the literature [80, 130]. More recently, deep learning techniques have shown promising performance for multi-modal fusion [114, 139, 140]. Moreover, deep learning methods have shown superb performance for many classification problems including image classification.

1.1. Variational Gaussian Process for Multisensor Classification Problems

In the first chapter, we develop a new model for multi-sensory data classification. To tackle this problem, probabilistic modeling and variational Bayesian inference are used. A Gaussian Process (GP) classifier is built upon the introduced modeling. Its posterior distribution is approximated using variational Bayesian inference. Finally, test point labels are predicted employing this classifier. Very importantly, and in contrast to alternative approaches, the developed method does not discard samples with missing features and utilizes all the available information. Furthermore, to take into account that the quality of the information provided
by each sensor may differ (some modalities/sensors may provide more reliable/distinctive information than others), we introduce two versions of the algorithm. In the first one, the parameters modeling each sensor performance are shared while in the second one, each sensor parameters are estimated independently. Synthetic and real datasets are utilized to examine the validity of the developed models. The results obtained for binary classification problems justify their use and confirm their superiority over existing fusion architectures. Using one-vs-all scheme we can extend the developed algorithm to multi-class classification problem. The results of the developed algorithm have been published in [130] and [131].

1.2. Machine Learning for Gravity Spy: Dataset and Glitch Classification

In next chapter, traditional and advanced machine learning techniques have been applied to glitches which show up in gravitational-wave data of aLIGO. The detection of gravitational waves with ground-based laser-interferometric detectors requires sensitivity to changes in distance much smaller than the diameter of atomic nuclei. Though sophisticated machinery and techniques have been developed over the past few decades to isolate such instruments from non-astrophysical noise, the detectors are still susceptible to instrumental and environmental noise transients, glitches, which hinder searches for transient gravitational waves. The Gravity Spy project is an effort to comprehensively classify the glitches that afflict gravitational-wave detectors into morphological families by combining the strengths of machine learning algorithms and citizen scientists.

This chapter presents the initial Gravity Spy dataset used for citizen scientist and machine learning classification – a static, accessible, documented dataset for testing machine learning supervised classification. This set consists of time-frequency images of LIGO glitches and their associated metadata. The primary purpose of classifying glitches is to understand their
characteristics and origin, which facilitates their removal from the data or from the detector entirely. Results from the application of state-of-the-art supervised classification methods to this dataset are presented in order to provide baselines for future glitch classification work. The baseline methods are selected from both traditional and more recent deep learning approaches. We suggest two multi-view deep neural network models (a parallel view and a merged view) and compare their performances to deep single view models. The suggested classifiers exploit four different views for classification.

Later in this chapter, we present and define the protocols and algorithms for (i) glitch image classification using multiple citizen labels and machine learning classifier outputs, (ii) evaluation of the performance of citizens during the classification process so as to promote them to higher levels of expertise, and (iii) enrichment of a golden set of images with certain labels. The results of the developed algorithms have been published in [154], [16], [15] and [17].

1.3. Machine Learning for Cultural Heritage

In the next chapter, we first, study the problem of automatic pigment identification and then, pigment unmixing problem in hyperspectral paintings is explored. Visible hyperspectral imaging (HSI) is a fast and non-invasive imaging method that has been adapted by the field of conservation science to study painted surfaces. By collecting reflectance spectra from a 2D surface, the resulting 3D hyperspectral data cube contains millions of recorded spectra. While processing such large amounts of spectra poses an analytical and computational challenge, it also opens new opportunities to apply powerful methods of multivariate analysis for data evaluation.
Here, we cast the problem of pigment identification in a novel way by decomposing the spectrum into pure pigments. The pure pigment exemplars, chosen and prepared in our laboratory based on historic sources and archaeological examples, closely resemble the materials used to make ancient paintings. To validate our algorithm, we created a set of mock-up paintings in our laboratory consisting of a broad palette of mixtures of pure pigments. Our results clearly demonstrate more accurate estimation of pigment composition than purely distance-based methods such as spectral angle mapping (SAM) and spectral correlation mapping (SCM). In addition, we studied hyperspectral imagery acquired of a Roman-Egyptian portrait, excavated from the site of Tebtunis in the Fayum region of Egypt, and dated to about the 2nd century CE. Using ground truth information obtained using Raman spectroscopy, we show qualitatively that our method accurately detects pigment composition for the specific pigments hematite and indigo.

With the intent of expanding current data treatment of hyperspectral datasets, an innovative approach for data reduction and visualization is presented next. It uses a statistical embedding method known as t-SNE (t-Distributed Stochastic Neighbor Embedding) to provide a nonlinear representation of spectral features in a lower 2-dimensional space. The efficiency of the developed method for painted surfaces from cultural heritage is established through the study of laboratory prepared paint mock-ups, and medieval French illuminated manuscript.

Next in this chapter, the problem of automatic nonlinear unmixing of hyperspectral reflectance data using works of art as test cases is described. Our approach is to interrogate a hyperspectral image cube by, first, decomposing it into a set of pure pigments and, second, to estimate the scattering and absorption coefficients of each individual pigment to produce semi-quantitative values for the component fractions. This is accomplished through deep neural networks (DNN) used to qualitatively identify the constituent pigments in any unknown
spectrum. Based on the pigment(s) present in a given pixel, Kubelka-Munk theory is employed to estimate its semi-quantitative concentration. We also use DNN to estimate the quantitative concentration of pigments in a given spectrum. To evaluate the performance of the developed algorithm, a set of mock-up paintings consisting of a broad palette of pigment mixtures, and pure pigment exemplars, were measured using diffuse reflectance hyperspectral imaging. We compare the results of our approach to previous state-of-the-art algorithms. We also apply the developed method to a real painting of Islamic works of art which is a 15th century example and present the coefficient maps for exemplar pigments. Next, we develop a DNN to learn the mixing function and estimate the coefficients. We apply this model to estimate the concentration values of the pigments in a real Fayum dataset. The results of the developed algorithms have been published in [132], [122] and [133].

1.4. Convergence of Weights in Deep Neural Networks

In the last chapter, we show that for many datasets it may be possible to compute weights of layers in a neural network directly from training data, without stochastic gradient descent (SGD). Assuming that the dataset contains a hierarchy of features and that the architecture learns the hierarchy, we find that the training dynamics of layers may approximately decouple, with layers closer to input showing faster training dynamics than layers above. We exploit this to analytically calculate the approximate outcome of SGD for each layer and find that it should converge to a class-based PCA, with some weights in every layer dedicated to principal components of each label class. The class-based PCA allows us to train layers directly, without SGD, often leading to a dramatic decrease in training complexity. We demonstrate the effectiveness of this by using our results to replace one and two convolutional layers in networks trained on MNIST, CIFAR10 and CIFAR100 datasets, showing that our
method achieves performance superior or comparable to ConvNet. We also show that in our method often a fraction of the training data is sufficient to train layers.

1.5. Outline of the Dissertation

In this prospectus, we focus on data fusion approaches which could be applied to different applications and datasets. We will make extensive use of the learning techniques throughout this prospectus. The rest of this prospectus is outlined as follows:

- In Chapter 2, we develop a probabilistic model which can be used in multi-sensor classification problems. We also apply the model to synthetic and real datasets and compare the performance of our algorithm with the state-of-the-work algorithms in the fusion literature.

- In Chapter 3, traditional and advanced machine learning algorithms have been applied to transient, non-Gaussian noise sources known as glitches. We also introduce a standard dataset for future use and develop two multi-view deep learning frameworks. At the end of the chapter, we introduce probabilistic frameworks used for glitch image classification based on multiple citizen labels and machine learning classifier outputs.

- In Chapter 4, we study two well-known problems in cultural heritage: pigment identification and pigment unmixing. We tackle the first problem by applying sparse unmixing model. We also apply tsne algorithm to map our dataset to a lower dimensional space and apply state-of-the-art algorithms in the literature to find the clusters. For the second problem, we use deep learning frameworks to first find the constituent pigments of any given spectrum. Then, we develop two methods to find the concentration values of the pigments. In the first one, we apply nonlinear
mixing function and gradient descent approach to find the concentration values of the pigments. In the latter method, a DNN is used to estimate the concentration values of the pigments. Next, we replace the second step with another DNN.

• In Chapter 5, we analyze the dynamics of SGD and show that, indeed, direct computation of solutions is possible in many datasets. We show that best performing setting used in DNNs introduce a separation of time-scales in the training dynamics, allowing SGD to train layers separately from lowest (closest to input) to highest. We find that for each layer, the distribution of solutions found by SGD can be estimated using a class-based principal component analysis (PCA) of the layer’s input. This allows us to forgo SGD entirely and directly derive the DNN parameters using this class-based PCA. We also confirm that the class-based PCA often converges using a fraction of the data required for SGD. Additionally, our method reduces the training time complexity of higher layers by eliminating layers from the costly back-propagation step of the training.

• Finally, we draw conclusion remarks in Chapter 6.
CHAPTER 2

Variational Gaussian Process for Multisensor Classification Problems

2.1. Introduction

There are numerous machine learning problems where different views of a single object exist and multimodal information can be used to provide more global information on the object of interest. In such problems, different sensors (modalities) capture information and data fusion is employed to combine the information gathered by all sources, which should lead to a more accurate understanding of the environment. The more sensors, the greater amount of available information, and therefore, the better for the performance of the system (e.g. accuracy in a classification problem). However, fusion techniques become specially useful when the information provided by different sensors is complementary [80]. In these cases, the combination of the information results in an extra improvement of the performance, which would not be possible if the information of each sensor is processed separately.

Kernel based methods such as Support Vector Machines (SVM) [40] or Gaussian Processes (GP) [127] are currently two of the most utilized fusion tools. In [29], the authors propose a composite kernel machine framework for the enhanced classification of hyperspectral images. They construct a family of composite kernels by combining spatial and spectral information linearly and using local cross-information. The authors justify their approach by presenting the classification accuracies when different kernel based models are applied to an AVIRIS
image. The obtained results confirm that using composite kernels leads to better results than using spectral information only.

In [148], the authors introduce simple closed form kernels and use them with Gaussian Processes to discover patterns and enable extrapolation over long ranges beyond the available data. In order to justify their method, they use synthetic examples, an atmospheric $CO_2$ trends [127] and an airline passenger data [72].

In [45], the authors use an additive Gaussian Process that generalizes Generalized Additive Models (GAM) and Squared-Exponential Gaussian Processes (SE-GP). In their experiments, they show that additive structure is present in real datasets which enhances the performance of the proposed model compared to that of standard GP models. Inference is carried out using Expectation Propagation (EP) [108].

In [57], the authors use Gaussian Process to build high-fidelity 3D representations of the environment for robot sensing tasks. By using Gaussian Process Implicit Surfaces (GPIS), a distinct continuous surface representation is built for each modality. Then, a local consistency test is performed between the surface representations to separate consistent data from inconsistent ones. After this test, the consistent data are combined to make the fused surface more accurate than each sensor surface. They apply their algorithm to synthetic data of Stanford bunny [94] and real data of a mobile robot [121] equipped with a laser scanner and a radar.

In [12], the authors tackle urban tree species classification problem using both AVIRIS and LIDAR data. Initially, they process AVIRIS and Lidar data separately, then segmentation algorithms are applied to each data set to obtain regions of interest (ROI), followed by feature extraction techniques. After these three steps, the extracted features are fused (concatenated) and canonical discriminant analysis (CDA) classifier is applied to find the labels of each
individual pixel. Finally, by majority voting each ROI is labeled. The studied dataset is downtown Santa Barbara with 29 tree species. Applying their method to this dataset, the authors show that the classification accuracy improves when fusion techniques are used.

Multiple feature learning for hyperspectral image classification is studied in [96]. In this article, the authors use both linear and nonlinear sets of features extracted from the original spectral features. They propose to use a combination of both types of features to cope with linear and nonlinear boundaries between different data classes. In the proposed approach, no regularization parameters are needed to control the weights of different sets of features. Logistic regression via a variable splitting and augmented Lagrangian (LORSAL) algorithm [95] has been selected as the classifier for this framework. Four different hyperspectral images (AVIRIS, ROSIS Pavia University, ROSIS Pavia Centre and HYDICE Washington DC) have been studied in this article and the classification accuracies using different set of features including original spectral information, morphological features, and kernel-based features have been reported.

In [63], the authors propose a Nonlinear Multiple Kernel Learning algorithm which uses spectral and spatial features of hyperspectral images. Principal Component Analysis is performed on the original features and spatial features are extracted. Multiple kernels are nonlinearly combined. This algorithm is used for hyperspectral image classification and is applied to three hyperspectral images: University of Pavia, Indian Pine and Salinas.

During the last few years, Deep Learning (DL) has been shown to be a powerful tool for solving fusion problems (see [125] for an extensive survey). For instance, in [76], the authors use Convolutional Neural Networks (CNNs) for fault diagnosis on a planetary gearbox. The main problem of using DL for fusion is that most of the proposed methods in the literature
cannot deal with missing samples. Only generative methods, such as [31] or [71] can simulate the missing modality and use it for classification.

Regarding the model we develop in the dissertation, the most similar works in the literature are [79], [151] and [35]. In [79], the authors introduce one GP for modeling each sensor. The data fusion is performed in the likelihood function which is a mixture of cumulative distribution functions of a standard normal distribution. EP inference is used to approximate the posterior distribution of the unknowns. However, this formulation requires an extra step for estimating the weights of each sensor for classifying a new sample. Although, the proposed model shares the same structure, we utilize a new likelihood function and Variational Inference which allow a joint estimation of all the model unknowns. In [151], the authors also consider one GP for modeling each sensor. For data fusion, the authors introduce a consensus function. In section 2.5, we show that this consensus function is very sensitive to noisy sensors, which can lead to poor performances in some scenarios. In [35], the authors introduce one GP for two sensors, whose prior covariance matrix is a sum of a linear and squared exponential kernels [127], that is, one kernel for each sensor. We see in section 2.5, that it can be formulated as a particular case of the proposed method, however the formulation proposed by the authors in [35] does not deal with the missing samples case.

The rest of this chapter is organized as follows. First, we introduce a Bayesian modeling of the fusion for classification problem in section 2.2. Variational inference is used to derive the training algorithm in section 2.3. In section 2.4, we introduce the classification rule. In section 2.5, we discuss the relationship of the developed model with Early and Late fusion as well as the state-of-the-art methods Bayesian Co-Training. Experimental results are presented in section 2.6, and finally section 2.7 concludes the chapter.
2.2. Bayesian modeling

The main goal in this work is to solve a classification problem where data are acquired by \( P \) different sensors. The input (feature) training set is defined by the following matrix

\[
X = [X_1 | X_2 | \ldots | X_P] = \begin{bmatrix}
X_{11} & X_{12} & \ldots & X_{1P} \\
X_{21} & X_{22} & \ldots & X_{2P} \\
\vdots & \vdots & \ddots & \vdots \\
X_{N1} & X_{N2} & \ldots & X_{NP}
\end{bmatrix} \in \mathbb{R}^{N \times (D_1+D_2+\ldots+D_P)} \tag{2.1}
\]

where \( x_{ji} \in \mathbb{R}^{1 \times D_i} \) represents the \( j \)-th training feature vector with dimension \( D_i \), acquired by the \( i \)-th sensor. The corresponding training labels associated to each row of \( X \) are given by the vector \( y = (y_1, \ldots, y_N)^T \in \{0, 1\}^{N \times 1} \).

Given \( X \), two classical fusion strategies are possible. We can build a classifier by concatenating the features observed by all sensors, \( i.e., \) by using \( x_j = (x_{j1}, \ldots, x_{jP}) \) with associated label \( y_j \) (this is called “early fusion” method). Associated to each \( X_i, i = 1, \ldots, P \) and using \( y \), we can also build \( P \) independent classifiers, these classifiers are later combined in decision level (this is called “late fusion” method). While both approaches have some interest, the second one makes an independence assumption which is unrealistic in many real problems while the first one does not include explicit cross-relations between sensors whose knowledge may be of interest for the problem at hand.

We now describe the approach we develop for the multi-sensor fusion problem. To relate samples and labels, we introduce a set of latent variables for each sensor, that is, \( f_1, \ldots, f_P \in \mathbb{R}^{N \times 1} \). For the \( i \)-th sensor, the corresponding set of latent variables \( f_i \) follows a Gaussian distribution \( \mathcal{N}(0, \alpha_i K_i + \gamma_i^2 I) \), where \( \alpha_i \) is the so-called “signal variance” parameter,
\( \gamma_i \) is the “noise variance” parameter, and \( K_i \in \mathbb{R}^{N \times N} \) is a kernel matrix depending on a set of parameters \( \Omega_i \). The component \((n, m)\) of \( K_i(n, m) \) is calculated as \( K_i(n, m) = k_{\Omega_i}(x_{ni}, x_{mi}) \) where \( k_{\Omega_i}(\cdot, \cdot) \) is a kernel function depending on the parameters \( \Omega_i \).

Concatenating all latent variables, we obtain the vector \( f = [f^T_1, \ldots, f^T_p]^T \in \mathbb{R}^{PN \times 1} \), which follows a Gaussian distribution \( \mathcal{N}(0, K) \), where \( K \in \mathbb{R}^{PN \times PN} \) is a block-diagonal matrix

\[
K = \begin{bmatrix}
\alpha_1 K_1 + \gamma_1^2 I & 0 & \ldots & 0 \\
0 & \alpha_2 K_2 + \gamma_2^2 I & \ldots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \ldots & \alpha_P K_P + \gamma_P^2 I
\end{bmatrix}
\] (2.2)

Eventually, during the acquisition procedure, we may have sensors that do not work appropriately, which sometimes generate features with missing entries. Most of the proposed methods in the fusion literature are not capable of dealing with this problem, and perform the training stage by discarding all the samples with missing entries, furthermore they cannot make predictions for samples with missing features. The model developed in this work is trained using all the available information and can make predictions for data with missing features. Since \( x_{ji} \) corresponds to the latent variable \( f_{ij} \), we introduce a zero degenerate prior distribution on the latent variables corresponding to the missing data point. That is, if the \( i \)-th sensor missed the information of the \( j \)-th sample, we set the corresponding \( j \)-th row and column of the matrix \( \alpha_i K_i + \gamma_i^2 I \) to zero.

To relate the labels \( y \) to the latent variables \( f \), we introduce the following likelihood function

\[
p(y|f) = \prod_{j=1}^{N} p(y_j|f_{.,j}) = \prod_{j=1}^{N} \sigma \left( \sum_{i=1}^{P} f_{ij} \right)^{y_j} \sigma \left( - \sum_{i=1}^{P} f_{ij} \right)^{1-y_j},
\]
where $\sigma(\cdot)$ is the sigmoid function and $f_{\cdot,j} = (f_{1j}, \ldots, f_{Pj})^T$. The rationale behind this model is that each sensor is capable of independently providing a classifier from all the information it gathers. For a given sample $x_j$, adding the GP values associated to the sensors $i$ and $i'$, $f_{ij}$ and $f_{i'j}$, respectively will increase (decrease) the likelihood of the observed label if they are in agreement (disagreement) in their labels.

The joint distribution can be written as

$$p(y, f, \alpha, \gamma, \Omega) = p(y|f)p(f|\alpha, \gamma, \Omega)p(\alpha)p(\gamma)p(\Omega)$$  \hspace{1cm} (2.4)$$

where $\Omega = \{\Omega_1, \ldots, \Omega_P\}$, $\alpha = (\alpha_1, \ldots, \alpha_P)^T$, $\gamma = (\gamma_1^2, \ldots, \gamma_P^2)^T$, and we use improper flat priors for $p(\alpha)$, $p(\gamma)$ and $p(\Omega)$.

From a Bayesian perspective, this model has an interesting interpretation. For a given sensor, the prior distribution introduces the correlations between the samples, however, it considers that the sensors are not correlated a priori as indicates eq. (2.2). The likelihood function in eq. (2.3) models how to combine the information provided by all the sensor, to classify a sample. The inference procedure, that we introduce in section 2.3, will result in a posterior distribution approximation, which will take into account both, correlations between samples and correlations between sensors.

We have developed two versions of the intermediate fusion algorithm. In the first version, we assume that the signal variance parameters of all sensors are the same. In the second version, we assume that there are $P$ different signal variance variables. The modeling of intermediate fusion architecture for two versions is shown in Figure 2.1.
2.3. Variational inference

The posterior distribution of the unknowns given the observations is given by

\[ p(f, \alpha, \gamma, \Omega | y) = \frac{p(y, \alpha, \gamma, f, \Omega)}{p(y)} \]  \quad (2.5) \]

However, for our problem, this posterior cannot be analytically calculated because the following integral is not tractable

\[ p(y) = \int p(y, f, \alpha, \gamma, \Omega) d(f, \alpha, \gamma, \Omega) \]  \quad (2.6) \]
Variational Bayesian Inference (VBI) approximates the posterior distribution by minimizing the Kulback-Leibler (KL) divergence

$$KL(q(f, \alpha, \gamma, \Omega)||p(f, \alpha, \gamma, \Omega|y)) = \int q(f, \alpha, \gamma, \Omega) \log \left( \frac{q(f, \alpha, \gamma, \Omega)}{p(f, \alpha, \gamma, \Omega|y)} \right) df, d\alpha, d\gamma, d\Omega$$

$$= \int q(f, \alpha, \gamma, \Omega) \log \left( \frac{q(f, \alpha, \gamma, \Omega)}{p(y, f, \alpha, \gamma, \Omega)} \right) df, d\alpha, d\gamma, d\Omega$$

$$+ \text{const} \quad (2.7)$$

The KL divergence is always non-negative and is equal to zero if and only if \(q(f, \alpha, \gamma, \Omega)\) and \(p(f, \alpha, \gamma, \Omega|y)\) coincide. Unfortunately, the functional form of \(p(y|f)\) does not allow the direct evaluation of the KL divergence. To alleviate this problem, we use the following lower bound \([24]\)

$$\log(1 + e^f) \leq \lambda(\xi)(f^2 - \xi^2) + \frac{f - \xi}{2} + \log(1 + e^\xi) \quad (2.8)$$

which produces the following lower bound for the logarithm of the joint distribution

$$\log p(y, f, \alpha, \gamma, \Omega) \geq \log M(y, f, \alpha, \gamma, \Omega, \xi) = \text{const}+$$

$$\log p(f|\alpha, \gamma, \Omega) + \sum_{j=1}^{N} \left\{ \left(y_j - \frac{1}{2}\right) 1^T f_{j} - \lambda(\xi_j) f_{j}^T 1 1^T f_{j} + \lambda(\xi_j) \xi_j^2 + \frac{\xi_j}{2} - \log(1 + e^{\xi_j}) \right\}$$

where \(\lambda(\xi) = \frac{1}{2\xi} \left( \frac{1}{1+e^{-\xi}} - \frac{1}{2} \right)\) and \(\xi = (\xi_1, \ldots, \xi_N)^T\) is a vector of additional positive parameters to be estimated.

Using the variational bound in eq. (2.9), the KL divergence is bounded as

$$KL(q(f, \alpha, \gamma, \Omega)||p(f, \alpha, \gamma, \Omega|y)) \leq KL(q(f, \alpha, \gamma, \Omega)||M(y, f, \alpha, \gamma, \Omega, \xi)) \quad (2.10)$$
then the right-hand side of eq. (2.10) is minimized with respect to \( q(f, \alpha, \gamma, \Omega) \) and \( \xi \), to push the left-hand side to a minimum.

Additional assumptions on \( q(f, \alpha, \gamma, \Omega) \) are imposed in order to find the solution for this minimization problem. The mean field theory [119] considers the following factorization for the posterior approximation

\[
q(f, \alpha, \gamma, \Omega) = q(f)q(\alpha, \gamma, \Omega) \tag{2.11}
\]

where \( q(\alpha, \gamma, \Omega) \) is restricted to be a degenerate distribution. The joint posterior approximation \( q(f, \alpha, \gamma, \Omega) \) can then be sequentially estimated by alternating between the estimations of \( q(f) \) and \( q(\alpha, \gamma, \Omega) \).

Let \( \hat{q}(\alpha, \gamma, \Omega) \) and \( \hat{\xi} \) be the current estimations of \( q(\alpha, \gamma, \Omega) \) and \( \xi \), respectively. Then, the estimation of \( q(f) \) is given as

\[
\log \hat{q}(f) = \mathbb{E}_{\hat{q}(\alpha, \gamma, \Omega)} \left[ \log M(y, f, \alpha, \gamma, \Omega, \hat{\xi}) \right] + \text{const} \tag{2.12}
\]

which is a quadratic function of \( f \). That means \( \hat{q}(f) \) is a Gaussian distribution whose mean and covariance matrix can be calculated respectively by taking the first and the second derivatives of eq. (2.12) with respect to \( f \), that leads to

\[
\hat{q}(f) = \mathcal{N}(f | \bar{f}, \Sigma) \tag{2.13}
\]

where

\[
\bar{f} = \Sigma \left( 1 \otimes \left( y - \frac{1}{2} 1 \right) \right) \quad \text{and} \quad \Sigma = (\hat{K}^{-1} + \hat{W})^{-1} \tag{2.14}
\]

with \( \hat{W} = 2(11^T \otimes \hat{\Lambda}) \), \( \hat{\Lambda} = \text{diag}[\lambda(\hat{\xi}_1), \ldots, \lambda(\hat{\xi}_N)] \) and \( \otimes \) denotes the Kronecker product.
Given $\xi$, the approximated likelihood function $q(y|\alpha, \gamma, \Omega)$ \[127\] can be calculated by integrating $M(y, f, \alpha, \gamma, \Omega, \hat{\xi})$ on $f$ resulting in

$$q(y|\alpha, \gamma, \Omega) = \mathcal{N}\left(y|\frac{1}{2}\mathbf{1}, 2\hat{\Lambda} + 4\sum_{i=1}^{P} \hat{\Lambda} (\alpha_i K_i + \gamma_i^2 \mathbf{I}) \hat{\Lambda}\right)$$ \hspace{1cm} (2.15)

which is used to calculate the point where $\hat{q}(\alpha, \gamma, \Omega)$ degenerates as

$$\hat{\alpha}, \hat{\gamma}, \hat{\Omega} = \arg\max_{\alpha, \gamma, \Omega} q(y|\alpha, \gamma, \Omega)$$ \hspace{1cm} (2.16)

Here, $\hat{\alpha}$, $\hat{\gamma}$, and $\hat{\Omega}$ are the optimal parameters of the model.

To estimate the variational parameters $\xi$, we maximize $E_{q(f, \alpha, \gamma, \Omega)} [\log M(y, f, \alpha, \gamma, \Omega, \xi)]$. Taking derivatives with respect to $\xi_j$ and equating to zero, we obtain

$$\xi_j = \sqrt{1^T (f_{\cdot, j} f_{\cdot, j}^T + \Sigma_j) 1},$$ \hspace{1cm} (2.17)

where $\Sigma_j$ is obtained from $\Sigma$ by removing the rows and columns which do not correspond to the components of $f_{\cdot, j}$.

The inference procedure is summarized in Algorithm 1.

---

**Algorithm 1** Intermediate Fusion Training

**input:** $X, y$, initials $\hat{\alpha}$, $\hat{\gamma}$, $\hat{\Omega}$ and $\hat{\xi}_j = 1, \forall j = 1, \ldots, N$.  

1: repeat  
2: Update $\hat{q}(f)$ using eq. (2.13).  
3: Update $\hat{\alpha}, \hat{\gamma}, \hat{\Omega}$ by solving the maximization problem in eq. (2.16).  
4: Update $\hat{\xi}$ using eq. (2.17).  
5: until convergence
2.4. Classification rule

Given a new sample \( x_* = [x_{*1}, \ldots, x_{*P}] \), the classification rule is based on the posterior probability \( p(y_*|y) \), which can be written as

\[
p(y_*|y) = \int p(y_*|f_*)p(f_*|f, \alpha, \gamma, \Omega)p(f, \alpha, \gamma, \Omega|y)d(f_*, f, \alpha, \gamma, \Omega) \tag{2.18}
\]

where \( f_* = (f_{1*}, \ldots, f_{P*})^T \).

The probability \( p(y_*|f_*) \) is given by eq. (2.3), meanwhile the posterior distribution, \( p(f, \alpha, \gamma, \Omega|y) \), can be approximated by \( \hat{q}(f, \alpha, \gamma, \Omega) \) obtained by Algorithm 1 at convergence.

Since the vector \( (f_*, f^T)^T \) follows a Gaussian distribution

\[
\begin{pmatrix}
  f_* \\
  f
\end{pmatrix}
\sim \mathcal{N}
\begin{pmatrix}
  0 \\
  0
\end{pmatrix},
\begin{bmatrix}
  C & H^T \\
  H & K
\end{bmatrix}
\tag{2.19}
\]

where

\[
H = \begin{bmatrix}
  h_1 & 0 & \ldots & 0 \\
  0 & h_2 & \ldots & 0 \\
  \vdots & \vdots & \ddots & \vdots \\
  0 & 0 & \ldots & h_P
\end{bmatrix}, \quad C = \text{diag}[c_1, c_2, \ldots, c_P] \tag{2.20}
\]

with

\[
h_i = (\alpha_i k_{\Omega_i}(x_{1i}, x_{*1}), \ldots, \alpha_i k_{\Omega_i}(x_{Ni}, x_{*i}))^T, \quad c_i = \alpha_i k_{\Omega_i}(x_{*i}, x_{*i}) + \gamma_i^2 \tag{2.21}
\]

the conditional distribution \( p(f_*|f, \alpha, \gamma, \Omega) \) is obtained as

\[
p(f_*|f, \alpha, \gamma, \Omega) = \mathcal{N}(f_*|H^TK^{-1}f, C - H^TK^{-1}H) \tag{2.22}
\]
By substituting in eq. (2.18) we obtain

$$p(y_\ast = 1|y) \approx \int \sigma \left( 1^T f_{\ast, \ast} \right) \mathcal{N}(f_{\ast, \ast}|m(x_\ast), S(x_\ast)) df_{\ast, \ast}$$  \hspace{1cm} (2.23)$$

where $m(x_\ast) = \hat{H}^T \hat{K}^{-1} \hat{f}$ and $S(x_\ast) = \hat{C} - \hat{H}^T (\hat{K} + \hat{W}^{-1})^{-1} \hat{H}$.

The integral in eq. (2.23) is approximated as in [24] resulting in

$$p(y_\ast = 1|y) \approx \sigma \left( m(x_\ast) \kappa \left( v^2(x_\ast) \right) \right)$$  \hspace{1cm} (2.24)$$

where $m(x_\ast) = 1^T m(x_\ast)$, $v^2(x_\ast) = 1^T S(x_\ast) 1$ and $\kappa(v^2) = (1 + \frac{1}{8} \pi v^2)^{-1/2}$.

Finally, given a threshold $\delta$, the classification rule is given as

$$y_\ast = \begin{cases} 1 & \text{if } p(y_\ast = 1|y) \geq \delta \\ 0 & \text{if } p(y_\ast = 1|y) < \delta \end{cases}$$  \hspace{1cm} (2.25)$$

Notice that if at testing phase, $x_{\ast i}$ is not observed, the developed model can still provide a prediction for the sample $x_\ast$. To do that, we set the corresponding $h_i$ and $c_i$ in eq. (2.20) equal to zero.

### 2.5. Related models

In this section, we discuss the relationship between the developed model and alternative fusion models based on Gaussian Processes.

By defining $g_j = f_{1j} + \ldots + f_{Pj}$ in eq. (2.3), the developed model in section 2.2 corresponds to a Gaussian Process Classifier [127], with latent variables $g = (g_1, \ldots, g_N)^T$ and prior distribution

$$p(g|\alpha, \gamma, \Omega) = \mathcal{N} \left( g|0, \bar{K} = \sum_{i=1}^{P} \alpha_i K_i + \gamma^2 I \right)$$  \hspace{1cm} (2.26)$$
As we will see in the experimental section, the formulation introduced in section 2.2 allows to understand how our model learns the correlation between different sensors, as well as, an intuitive modeling for the missing sample case. However, we can use eq. (2.26) to understand how the developed model is related to [151] and [35].

The prior model proposed in [35], is a particular case of the prior model in eq. (2.26) when \( P = 2, \gamma_1 = \gamma_2 = 0, \) \( K_1 \) is a linear kernel and \( K_2 \) is a squared exponential kernel.

In [151], the authors introduce the following prior model on the latent variables

\[
p(g, f_1, \ldots, f_P | \alpha, \gamma, \Omega) \propto \prod_{i=1}^{P} \mathcal{N}(f_i | 0, \alpha_i K_i) \exp \left\{ -\frac{||f_i - g||^2}{2\gamma_i^2} \right\}
\]  

where \( g \in \mathbb{R}^{P \times 1} \) is a consensus function. As in our model, the factors \( \mathcal{N}(f_i | 0, \alpha_i K_i) \) model the prior correlations between samples when the kernel matrices \( K_1, \ldots, K_P \) associated to the \( P \) sensors are used. The second factor, which can be considered a regularizer, models the relationship between the latent variables associated to the sensors a the consensus function. Notice that each vector \( f_i \) is forced to be similar to the latent variables \( g \). Unfortunately, as we will see in the experiments, when a sensor is not discriminative, the regularizer can lead to a poor \( g \) behavior when integrating on \( f_1, \ldots, f_P \).

By integrating in eq. (2.27) on \( f_1, \ldots, f_P \), the authors in [151] obtain the following prior model on \( g \)

\[
p(g | \alpha, \gamma, \Omega) = \mathcal{N} \left( g | 0, K_c = \left( \sum_{i=1}^{P} (\alpha_i K_i + \gamma_i^2 I)^{-1} \right)^{-1} \right)
\]  

where \( K_c \) is called the Co-training kernel.
Using basic properties of positive definite matrices, the following relationship between the precision matrices of both prior models (the utilized one in this work in eq. (2.26) and the proposed in [151] in eq. (2.28)) can be established

\[ v^T K^{-1} c v \geq v^T \tilde{K}^{-1} v, \quad \forall v \in \mathbb{R}^N. \]

(2.29)

The differences between both approaches are now clear. The Co-Training model assumes a stronger prior knowledge than our model. Our model gives more weight to the information provided by the observed labels.

### 2.6. Experimental results

In this section, the developed approach is evaluated on both, synthetic and real problems. For each sensor, we use the squared exponential kernel defined by

\[ k_{\beta_i}(x_{ji}, x_{ki}) = \exp\{-\|x_{ji} - x_{ki}\|^2 / 2\beta_i^2\}. \]

The length scale parameters \( \beta_i \) and signal variance parameters \( \alpha_i \) for all sensors are estimated during the training step. *Inter 1* is used to denote our developed fusion model. *Inter 2* is used for the case \( \alpha_i = \alpha, i = 1, \ldots, P \). Both methods are compared with [29], which combines different kernels to train a SVM classifier. We name this method *CK-SVM*. Parameters are estimated following the settings proposed by the authors in [29], that is, \( \{\beta_i\} \) are estimated by cross-validation from the set \( \beta_i \in \{10^{-3}, 10^{-2}, 10^{-1}, 1, 10\} \) for \( i = 1, \ldots, P \). The Bayesian co-training method proposed in [151] is also compared with our results. In order to perform a fair comparison, we consider the case when all signal variance parameters take different values for each sensor (*Co-Tr1*), and the case when all signal variance parameters take the same value (*Co-Tr2*). In the experiments, we also include early and late fusion methods. The early method first stacks all features and then builds the classifier (notice that this method cannot deal with missing features). The late method
fuses the *Class 1* posterior probability (see eq. (2.24)) provided by each of the *P* sensors by calculating their mean. These methods are denoted *Early* and *Late*, respectively. We also provide the results obtained by a GP classifier applied to each sensor separately. *Sensor* *i* is used to denote the results obtained only by the *i*th sensor. The results of deep neural network (*DNN*) are also reported for the comparison. The network consists of three fully connected, *dense* layers. The activation functions of the first two layers are *relu* and the activation function of the last layer is a *sigmoid*. Here, the fusion is performed in the hidden layers of the deep network as is explained in [125]. The number of epochs is set to 100 and the Adam optimizer with *binary cross entropy* loss function is used.

### 2.6.1. Synthetic experiment

Figure 2.2 displays the synthetic dataset which is used in our experiments. This dataset is called Two-moon and was introduced in [157]. The top (red) and bottom (blue) half moons correspond to two different classes, and as it can be observed from Figure 2.2, they cannot be linearly separated. This is a typical example of binary classification problem where non-linear classifiers are needed to obtain a good performance. Following the experiments presented in [79], we associate a different sensor to each coordinate (dimension). “Sensor 1” measures the horizontal component of each sample (X coordinate) while “Sensor 2” measures the vertical one (Y coordinate). The dataset contains 200 samples, 100 from each class. For training, 40 samples from each class are randomly selected, and the remaining 120 samples are used for testing. To avoid biased results, the experiment is repeated 10 times.

Figure 2.3 illustrates an example of how the developed method can combine the information provided by both sensors to obtain a better classification performance. Figure 2.3 (a) is the figure related to the sets used in the 5th realization in Table 2.1. From each class, 40 points
are randomly selected and used for training (depicted by green triangles). The remaining points are used for testing (60 samples from each class). Figure 2.3 (b) shows the classification results obtained by Sensor 1. As expected, many points are misclassified because of the overlap in X dimension of points in both classes. Figure 2.3 (c) shows the classification results obtained by Sensor 2. We can observe that the number of misclassified points is lower than in the previous case, because less points overlap in Y direction. We can conclude that the information provided by the second sensor is more discriminative for classifying samples (as we have seen in Table 2.1). We also observe from Figs. 2.3 (b) and 2.3 (c), that both classes cannot be perfectly separated using the information of one coordinate, because they overlap in both X and Y axes. Therefore, this problem only can be solved by taking into account the relationship between both sources of information. Figure 2.3 (d) shows the classification obtained by Inter 1 where we observe that all points are correctly classified.
In Table 2.1, we report the area under ROC curve (AUC), and Overall Accuracy (OA) obtained by setting the threshold value $\delta = 0.5$, for 10 random realizations, as well as, the corresponding mean values reported in the last column. First and second rows report the results obtained by Sensor 1 and Sensor 2, respectively. The third, fourth, fifth and sixth rows report the results obtained by Early, Inter 1, Inter 2, and Late fusion algorithms, respectively. Finally, the last three rows report the results for the state-of-the-art methods CK-SVM, Co-Tr1 and Co-Tr2, respectively.

The developed method Inter 1 obtains 0.99 and 99.00 of mean AUC and OA, respectively, and Inter 2 obtains 0.99 and 99.25 of mean AUC and OA, respectively. Therefore, the
developed methods can classify all samples almost perfectly. We observe that the mean OA for Inter 2 is slightly better than Inter 1. Notice that, in this case, it is realistic to assume that the scale parameters are the same for the two sensors and so Inter 2 performs slightly better than Inter 1.

Sensor 1 obtains 0.85 and 76.16 of mean AUC and mean OA, respectively, and Sensor 2 obtains 0.96 and 88.00 of mean AUC and mean OA, respectively. We observe that the obtained results are much worse than Inter 1. This means that information provided by each sensor must be combined to obtain a good classification performance. Early fusion obtains 0.99 and 99.16 of mean AUC and OA, respectively, and Late fusion obtains 0.98 and 94.41 of mean AUC and OA, respectively. In this case, Inter 2 obtains a higher mean AUC and OA than Early and Late, which proves that performing fusion in the latent variables can lead to a better performance, than stacking the features (Early) or combining the classifiers outputs (Late). CK-SVM obtains 0.99 and 97.91 of mean AUC and mean OA, respectively, which represents 1% less than Inter 1. We think this happens because CK-SVM selects the parameters by cross-validation from a small set of values. However, the developed parameter estimation method of Inter 1 and Inter 2 provided a finer values of the parameters which results in a better performance. Co-Tr1 obtains 0.96 and 87.50 of mean AUC and mean OA, respectively, and Co-Tr2 obtains 0.96 and 87.91 of mean AUC and mean OA, respectively. We observe that Co-Tr2, which uses the same value for all signal variance parameters, obtains slightly better results than Co-Tr1 which is consistent with the results obtained by Inter 1 and Inter 2. We also observe that Bayesian Co-Training obtains poor results, which are even worse than the obtained by Sensor 2. Notice that the information provided by “Sensor 1” is not very discriminative (Sensor 1 reaches only 0.85 of mean AUC) and the consensus function is built on contradictory information.
Table 2.1. Classification Overall Accuracies (OA) and Areas under the ROC Curves (AUC) for the compared methods.

Table 2.2 shows the estimated signal variance parameters for *Inter 1* and *Inter 2* fusion methods in 10 realizations. For *Inter 1*, we can observe that in all cases $\alpha_2 > \alpha_1$. In this case, the maximum signal variance parameter coincide with the most informative sensor. For *Inter 1*, we observe that all values for $\alpha$ are higher than $\alpha_1$ and lower than $\alpha_2$, which means that when we constrain both sensors to have the same value of signal variance, the system returns a weighted mean of the obtained values for *Inter 2*.

To compare our method to the method proposed in [151], we consider that besides the sensors corresponding to X and Y coordinates, we have $n \in \{0, 2, 4, 6, 8\}$ sensors which provide only noise features. These features are assumed to be realizations of a $\mathcal{N}(0, 0.01)$ distribution. In Figure 2.4, the mean AUC and mean OA for 10 random realization based on the number of noise sensors are provided. As it can be observed from Figure 2.4 (a) and 2.4 (b), the developed methods *Inter 1* and *Inter 2* are robust to noise and the mean OA and AUC do not degrade by adding non informative sensors. However, *Co-Tr1* and *Co-Tr2*
Table 2.2. Estimated signal variance values for Inter 1 and Inter 2 fusion algorithms applied to Two-Moon Dataset for 10 random realizations.

<table>
<thead>
<tr>
<th>Realizations</th>
<th>$\alpha_1$</th>
<th>$\alpha_2$</th>
<th>$\alpha$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>19.64</td>
<td>114.78</td>
<td>46.50</td>
</tr>
<tr>
<td>2</td>
<td>16.26</td>
<td>144.67</td>
<td>47.29</td>
</tr>
<tr>
<td>3</td>
<td>20.93</td>
<td>132.17</td>
<td>54.04</td>
</tr>
<tr>
<td>4</td>
<td>14.81</td>
<td>139.04</td>
<td>47.44</td>
</tr>
<tr>
<td>5</td>
<td>31.15</td>
<td>155.96</td>
<td>63.29</td>
</tr>
<tr>
<td>6</td>
<td>22.22</td>
<td>137.11</td>
<td>58.51</td>
</tr>
<tr>
<td>7</td>
<td>11.86</td>
<td>120.20</td>
<td>45.74</td>
</tr>
<tr>
<td>8</td>
<td>23.64</td>
<td>149.30</td>
<td>56.62</td>
</tr>
<tr>
<td>9</td>
<td>18.32</td>
<td>100.52</td>
<td>38.52</td>
</tr>
<tr>
<td>10</td>
<td>47.98</td>
<td>223.70</td>
<td>95.52</td>
</tr>
</tbody>
</table>

Figure 2.4. (a) mean AUC of different approaches vs number of noisy sensors, (b) mean OA of different approaches vs number of noise sensors.

perform worse when we add the information provided by these sensors. This, as we indicated in section 2.5, is due to the regularizer the consensus function uses.

2.6.2. Radar + Multispectral image classification

In this section, we investigate the use of the developed fusion algorithms on a real dataset, where the information is provided by two sensors. We use an image from Rome (Italy) captured in 1995, the goal is to classify the pixels as belonging to Urban vs. Non-Urban.
classes. This image has been provided by the authors of [60] and was acquired in the context of the Urban Expansion Monitoring Project (UrbEx) of the European Space Agency.

The first sensor (ERS2 SAR) captures 2 backscattering intensities images with 35 days of difference, and returns only one intensities image ($D_1 = 1$) representing the coherence between both observations. The second sensor (Landsat TM) provides a multispectral image with $D_2 = 7$ bands. In Figure 2.5 (a), we plot a small part ($400 \times 200$ pixels) of the coherence image captured by ERS2 SAR sensor. Figure 2.5 (b) shows the RGB bands captured by Landsat TM sensor for the same area. Finally, a reference land cover map provided by the Italian Institute of Statistics is also available. In Figure 2.5 (c), we show the region of interest corresponding to Figs. 2.5 (a) and 2.5 (b), where yellow corresponds to pixels belonging to class Urban, blue corresponds to pixels belonging to class Non-Urban and red corresponds to pixels whose class is unknown. Comparing the coherence band with the reference map, we can note a correspondence between pixels with high coherence values and pixels belonging to the class “Urban”. In the RGB image, we can also note that most of the pixels belonging to the class “No-urban” seem to have different color than pixels belonging to class “Urban”. So, both sensors seem to provide discriminative information for solving this classification problem. 100 pixels (50 from each class) are randomly selected for training, and 1000 pixels (500 from each class) are randomly selected for testing. To obtain unbiased results, the experiment is repeated 10 times with random training and testing sets.

The first row in Table 2.3 shows the performance of the fusion algorithms when both sensors provide information for all the training samples, i.e., the probability of malfunctioning when collecting a feature vector is $p = 0$ for both sensors. Sensor 1 obtained 0.88 and 80.54 of mean AUC and mean OA, respectively, and Sensor 2 obtained 0.95 and 87.50 of mean AUC and mean OA, respectively. In this case, we observe that the Landsat TM sensor has
Figure 2.5. Multi-spectral image classification: a) Coherence band provided by ERS2 SAR sensor, b) Original RGB image provided by Landsat TM sensor, c) Available groundtruth with Urban (YELLOW), No-urban (BLUE) and unknown pixels (RED).

more discriminative information than the ERS2 SAR sensor. Early fusion obtained 0.97 and 91.74 of mean AUC and mean OA, respectively. Inter 1 obtained 0.97 and 91.97, and Inter 2 obtained 0.97 and 91.96 of mean AUC and mean OA, respectively. Finally, Late fusion obtained 0.97 and 92.29 of mean AUC and mean OA, respectively. In this case, Late
fusion has the best overall accuracy. \textit{Inter 1} and \textit{Inter 2} obtained a minimum improvement over \textit{Early} fusion. We observe that all these results are better than the obtained ones by each sensor separately. The best mean OA was obtained by \textit{Late} fusion which was around 0.3\% better than the other fusion methods. \textit{CK-SVM} obtained 0.95 and 90.43 of mean AUC and mean OA, respectively, which means that our developed methods were around 2\% better. Bayesian Co-training algorithms have the worst performance among fusion algorithms. \textit{Co-Tr1} obtained 0.86 and 80.39 of mean AUC and mean OA, respectively. \textit{Co-Tr2} obtained 0.91 and 83.84 of mean AUC and mean OA, respectively.

Let us now investigate the use of sensors which do not always observe all the features. For each of the 10 training sets used for the no-missing samples case, we simulate the loss of information with probability $p \in \{0.4, 0.8\}$. That is, the probability of each sensor independently missing a sample is $p$. The results are reported in Table 2.3. We observe that mean AUC and mean OA drop for all methods when $p$ increases. For $p = 0.8$, we observe that \textit{Early} fusion obtains 2\% and 5\% of mean AUC and OA less than for the no missing sample case. \textit{Late} fusion obtains 17\% and 20\% of mean AUC and OA less than for the no missing sample case. However, \textit{Inter 1} and \textit{Inter 2} obtain similar values to the no-missing samples case, with a difference lower than 1\% of mean AUC and mean OA. For the \textit{CK-SVM} method, we observe that it obtains around 12\% mean AUC and mean OA less than the no missing samples case. Unlike the other methods, \textit{Inter 1} and \textit{Inter 2} are capable of managing all the information provided by both sensors. \textit{Early} and \textit{Late} fusion algorithms and \textit{CK-SVM} are forced to discard all incomplete samples which leads to a poor classification performance. The missing sample case is also handled by \textit{Co-Tr1} and \textit{Co-Tr2}. We observe that for \textit{Co-Tr1} the results for $p = \{0, 0.4, 0.8\}$ are very similar, notice that for $p = 0.4$ results are around 1\% better than those for $p = 0$. However, the Co-training mean AUC and mean OA are 9\%
<table>
<thead>
<tr>
<th></th>
<th>Sensor 1</th>
<th>Sensor 2</th>
<th>Early</th>
<th>Inter 1</th>
<th>Inter 2</th>
<th>Late</th>
<th>CK-SVM</th>
<th>Co-Tr1</th>
<th>Co-Tr2</th>
<th>DNN</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OA %</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p=0</td>
<td>80.54</td>
<td>87.50</td>
<td>91.74</td>
<td>91.97</td>
<td>91.96</td>
<td>92.29</td>
<td>90.43</td>
<td>80.39</td>
<td>83.84</td>
<td>87.31</td>
</tr>
<tr>
<td>AUC</td>
<td>0.88</td>
<td>0.95</td>
<td>0.97</td>
<td>0.97</td>
<td>0.97</td>
<td>0.97</td>
<td>0.95</td>
<td>0.86</td>
<td>0.91</td>
<td>0.93</td>
</tr>
<tr>
<td>p=0.4</td>
<td>80.53</td>
<td>87.42</td>
<td>91.26</td>
<td>91.84</td>
<td>91.87</td>
<td>80.13</td>
<td>87.49</td>
<td>81.63</td>
<td>81.43</td>
<td>77.18</td>
</tr>
<tr>
<td>OA %</td>
<td>0.88</td>
<td>0.94</td>
<td>0.97</td>
<td>0.97</td>
<td>0.97</td>
<td>0.87</td>
<td>0.94</td>
<td>0.88</td>
<td>0.88</td>
<td>0.82</td>
</tr>
<tr>
<td>p=0.8</td>
<td>79.99</td>
<td>86.86</td>
<td>86.35</td>
<td>91.28</td>
<td>91.37</td>
<td>72.53</td>
<td>78.29</td>
<td>79.15</td>
<td>76.12</td>
<td>72.18</td>
</tr>
<tr>
<td>AUC</td>
<td>0.87</td>
<td>0.94</td>
<td>0.95</td>
<td>0.97</td>
<td>0.97</td>
<td>0.80</td>
<td>0.84</td>
<td>0.86</td>
<td>0.82</td>
<td>0.78</td>
</tr>
</tbody>
</table>

Table 2.3. Mean classification accuracies and AUCs for probabilities $p=0, p=0.4$ and $p=0.8$ of a sensor fails acquiring a sample.

and 6% lower than the obtained by our method. It can be observed that the performance of DNN also degrades drastically in missing samples scenario when $p$ increases. Here, the algorithm discards incomplete samples and is trained using less number of training samples.

We study now the behavior of the different methods when the size of the training set increases. We randomly pick training sets of sizes $100, 200, \ldots, 1000$ samples. To obtain unbiased results, each experiment is repeated 10 times. We also consider missing samples with loss probability $p \in \{0, 0.4, 0.8\}$. The averaged OA and AUC for each case are shown in Fig. 2.6. The x-axis represents the size of the training set and the y-axis shows the mean OA (left) or mean AUC (right). Fig. 2.6 (a), corresponds to the perfect sensor case ($p = 0$). We observe that our developed algorithms have the highest performance, which increases when the size of the training set also increases. The performance of the DNN is worse than the our developed method. We believe that this happens due to the fact that here the size of the training set is not very large and also the dimension of the dataset is low. Note that the slope of the black line is larger which confirms that the improvement observed by the DNN is larger when we have a larger number of training samples. With all algorithms, the performance improves with the number of training samples. In Figs. 2.6 (b) and (c), we plot the mean OA and mean AUC for $p = 0.4$ and $p = 0.8$, respectively. We observe that
the performance of our developed algorithms does not decrease drastically when incomplete training samples are present. However, other algorithms suffer from that, more specifically the performance of the DNN becomes much worse compared to the perfect sensor scenario which again verifies its dependency on the number of training samples. As expected, in all cases the performance of the algorithms improves by adding more samples to the training set.

2.6.3. Pervasive change detection

In this experiment, the dataset was provided by the authors of [100]. The dataset is a pair of multi-spectral images captured by Quickbird satellite. Both 864 × 1060 images were acquired over Denver (USA) on July 17th 2002 and August 22nd 2008, respectively. The goal of this experiment is to detect pervasive changes in the images. To carry out this task, a set of 2280 labeled pixels is available, where 1140 correspond to the class $C_1 =$ “pervasive change”, and 1140 correspond to the class $C_0 =$ “non-pervasive change”.

Quickbird multispectral images are acquired by $P = 4$ sensors capturing different wavelengths: blue, green, red and near-infrared. For each sensor, we have a $D_j = 2, j = 1, 2, 3, 4$, feature vector where the first component corresponds to the image captured on Jul 17th 2002, and the second component corresponds to the image captured on Aug 22nd 2008.

For training, 280 samples (140 from each class) are randomly selected and the remaining 2000 labeled samples (1000 from each class) are used for testing. To obtain unbiased results, the experiment is repeated 10 times.

Table 2.4 shows the results obtained by the different algorithms. Sensor 1 obtained 0.87 and 78.30 of mean AUC and mean OA, respectively, Sensor 2 obtained 0.86 and 78.05 of mean AUC and mean OA, respectively, Sensor 3 obtained 0.86 and 77.84 of mean AUC and mean OA, respectively and Sensor 4 obtained 0.78 and 72.12 of mean AUC and mean
Figure 2.6. Overall accuracy and area under the ROC curve vs training set size for missing case scenario with three different missing probabilities.
OA, respectively. From these results, we observe that the first sensor provides the most
discriminative information and the fourth sensor is the least accurate one. Early fusion
obtained 0.89 and 80.98 of mean AUC and mean OA, respectively. Inter 1 obtained 0.89 and
81.83, and Inter 2 obtained 0.90 and 81.96 of mean AUC and mean OA, respectively. Late
fusion obtained 0.88 and 79.72 of mean AUC and mean OA, respectively. In this case, Inter
1 and Inter 2 obtained a minimum improvement over Early and Late fusion methods. The
accuracies of three fusion algorithms, Early, Inter 1 and Inter 2 are very close to each other
in all runs. The last three rows of Table 2.4 show the results of CK-SVM, Co-Tr1 and Co-Tr2
algorithms. CK-SVM obtained 0.84 and 77.65 of mean AUC and mean OA, respectively.
The other two performed similarly, which means that our developed method was around
5% better than the ones compared with. Finally, DNN method obtained 0.85 and 77.42 of
mean AUC and mean OA, respectively. It works similar to Bayesian co-training methods
and worse than our developed method and CK-SVM. We believe this happens due to small
size of training set, low dimensionality of the space ($d = 8$) and more number of parameters
to be estimated in DNN algorithm.

2.6.4. Occupancy detection

The dataset for this experiment was collected by the authors of [30] and is available in UCI
Machine Learning Repository. An office room with approximate dimensions of 5.85m ×
3.50m × 3.53m (W × D × H) was monitored with different sensors to obtain the following
variables: temperature, humidity, light and CO2 levels [30]. The goal is to detect if the room
is occupied or not according to the values of the four acquired features.

To study the behavior of the different methods when the size of the training set varies, we
randomly pick training sets of sizes 100, 200, . . . , 1000 samples. To obtain unbiased results for
Table 2.4. Classification accuracies and area under the curve of different level fusion algorithms and composite kernel applied to multispectral image with perfect simulated sensors for 10 random realizations.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>OA%</td>
<td>AUC</td>
<td>OA%</td>
<td>AUC</td>
<td>OA%</td>
<td>AUC</td>
<td>OA%</td>
<td>AUC</td>
<td>OA%</td>
<td>AUC</td>
<td>OA%</td>
</tr>
<tr>
<td>Sen. 1</td>
<td>78.55</td>
<td>0.88</td>
<td>79.60</td>
<td>0.87</td>
<td>74.95</td>
<td>0.86</td>
<td>76.70</td>
<td>0.87</td>
<td>79.00</td>
<td>0.87</td>
<td>79.85</td>
</tr>
<tr>
<td></td>
<td>79.20</td>
<td>0.87</td>
<td>77.20</td>
<td>0.87</td>
<td>79.20</td>
<td>0.87</td>
<td>80.40</td>
<td>0.87</td>
<td>77.65</td>
<td>0.87</td>
<td>78.30</td>
</tr>
<tr>
<td>Sen. 2</td>
<td>77.30</td>
<td>0.87</td>
<td>79.75</td>
<td>0.86</td>
<td>74.75</td>
<td>0.86</td>
<td>77.95</td>
<td>0.87</td>
<td>79.95</td>
<td>0.87</td>
<td>78.05</td>
</tr>
<tr>
<td></td>
<td>79.15</td>
<td>0.86</td>
<td>78.35</td>
<td>0.86</td>
<td>78.35</td>
<td>0.86</td>
<td>76.65</td>
<td>0.87</td>
<td>76.65</td>
<td>0.87</td>
<td>77.84</td>
</tr>
<tr>
<td>Sen. 3</td>
<td>78.00</td>
<td>0.87</td>
<td>77.75</td>
<td>0.86</td>
<td>74.70</td>
<td>0.86</td>
<td>79.95</td>
<td>0.87</td>
<td>79.50</td>
<td>0.87</td>
<td>78.05</td>
</tr>
<tr>
<td></td>
<td>79.15</td>
<td>0.86</td>
<td>77.65</td>
<td>0.86</td>
<td>76.60</td>
<td>0.85</td>
<td>78.90</td>
<td>0.86</td>
<td>76.65</td>
<td>0.87</td>
<td>77.84</td>
</tr>
<tr>
<td>Sen. 4</td>
<td>72.95</td>
<td>0.78</td>
<td>72.50</td>
<td>0.78</td>
<td>71.85</td>
<td>0.78</td>
<td>72.20</td>
<td>0.78</td>
<td>71.60</td>
<td>0.78</td>
<td>72.05</td>
</tr>
<tr>
<td></td>
<td>72.00</td>
<td>0.78</td>
<td>71.50</td>
<td>0.78</td>
<td>71.15</td>
<td>0.78</td>
<td>71.50</td>
<td>0.79</td>
<td>72.65</td>
<td>0.79</td>
<td>73.05</td>
</tr>
<tr>
<td>Early</td>
<td>81.75</td>
<td>0.80</td>
<td>80.40</td>
<td>0.80</td>
<td>76.65</td>
<td>0.80</td>
<td>82.05</td>
<td>0.80</td>
<td>78.65</td>
<td>0.80</td>
<td>82.85</td>
</tr>
<tr>
<td></td>
<td>80.40</td>
<td>0.80</td>
<td>78.15</td>
<td>0.80</td>
<td>78.65</td>
<td>0.80</td>
<td>82.85</td>
<td>0.80</td>
<td>82.60</td>
<td>0.80</td>
<td>80.98</td>
</tr>
<tr>
<td>Inter 1</td>
<td>82.95</td>
<td>0.90</td>
<td>81.40</td>
<td>0.89</td>
<td>79.55</td>
<td>0.89</td>
<td>82.60</td>
<td>0.89</td>
<td>78.80</td>
<td>0.89</td>
<td>82.65</td>
</tr>
<tr>
<td></td>
<td>81.40</td>
<td>0.89</td>
<td>79.15</td>
<td>0.89</td>
<td>80.10</td>
<td>0.89</td>
<td>82.65</td>
<td>0.89</td>
<td>84.60</td>
<td>0.89</td>
<td>81.83</td>
</tr>
<tr>
<td>Inter 2</td>
<td>83.15</td>
<td>0.90</td>
<td>81.90</td>
<td>0.90</td>
<td>79.30</td>
<td>0.90</td>
<td>82.85</td>
<td>0.90</td>
<td>80.00</td>
<td>0.90</td>
<td>83.10</td>
</tr>
<tr>
<td></td>
<td>81.90</td>
<td>0.90</td>
<td>79.30</td>
<td>0.90</td>
<td>81.55</td>
<td>0.90</td>
<td>82.85</td>
<td>0.90</td>
<td>83.35</td>
<td>0.90</td>
<td>81.96</td>
</tr>
<tr>
<td>Late</td>
<td>81.65</td>
<td>0.89</td>
<td>81.15</td>
<td>0.89</td>
<td>75.55</td>
<td>0.89</td>
<td>80.95</td>
<td>0.89</td>
<td>78.85</td>
<td>0.89</td>
<td>81.50</td>
</tr>
<tr>
<td></td>
<td>81.15</td>
<td>0.89</td>
<td>75.55</td>
<td>0.89</td>
<td>78.85</td>
<td>0.89</td>
<td>81.50</td>
<td>0.89</td>
<td>78.95</td>
<td>0.89</td>
<td>79.15</td>
</tr>
<tr>
<td>CK-SVM</td>
<td>79.20</td>
<td>0.83</td>
<td>79.70</td>
<td>0.82</td>
<td>75.55</td>
<td>0.82</td>
<td>75.40</td>
<td>0.82</td>
<td>78.30</td>
<td>0.82</td>
<td>79.75</td>
</tr>
<tr>
<td></td>
<td>79.70</td>
<td>0.82</td>
<td>75.55</td>
<td>0.82</td>
<td>75.40</td>
<td>0.82</td>
<td>78.30</td>
<td>0.82</td>
<td>77.90</td>
<td>0.82</td>
<td>77.65</td>
</tr>
<tr>
<td>Co-Tr1</td>
<td>78.65</td>
<td>0.85</td>
<td>77.60</td>
<td>0.85</td>
<td>79.10</td>
<td>0.85</td>
<td>77.15</td>
<td>0.85</td>
<td>76.85</td>
<td>0.85</td>
<td>78.60</td>
</tr>
<tr>
<td></td>
<td>77.60</td>
<td>0.85</td>
<td>79.10</td>
<td>0.85</td>
<td>77.15</td>
<td>0.85</td>
<td>76.85</td>
<td>0.85</td>
<td>78.60</td>
<td>0.85</td>
<td>77.72</td>
</tr>
<tr>
<td>Co-Tr2</td>
<td>76.55</td>
<td>0.86</td>
<td>77.85</td>
<td>0.86</td>
<td>79.40</td>
<td>0.86</td>
<td>77.15</td>
<td>0.86</td>
<td>76.85</td>
<td>0.86</td>
<td>78.95</td>
</tr>
<tr>
<td></td>
<td>77.85</td>
<td>0.86</td>
<td>79.40</td>
<td>0.86</td>
<td>77.15</td>
<td>0.86</td>
<td>76.85</td>
<td>0.86</td>
<td>79.00</td>
<td>0.86</td>
<td>77.47</td>
</tr>
<tr>
<td>DNN</td>
<td>76.35</td>
<td>0.85</td>
<td>79.20</td>
<td>0.85</td>
<td>75.70</td>
<td>0.85</td>
<td>74.20</td>
<td>0.85</td>
<td>76.50</td>
<td>0.85</td>
<td>77.84</td>
</tr>
<tr>
<td></td>
<td>79.20</td>
<td>0.85</td>
<td>75.70</td>
<td>0.85</td>
<td>74.20</td>
<td>0.85</td>
<td>76.50</td>
<td>0.85</td>
<td>77.84</td>
<td>0.85</td>
<td>77.42</td>
</tr>
</tbody>
</table>

As we can observe in Fig. 2.7, our developed algorithm outperforms the rest of the fusion algorithms. For this dataset, the Bayesian Co-training algorithm demonstrated the worst performance, whereas for the larger training sets, the performance of the DNN is comparable to the intermediate fusion algorithms.
Figure 2.7. Overall accuracy and Area under ROC curve vs training set size for Occupancy Detection dataset.

2.7. Conclusions

In this chapter, we use Gaussian Process theory to model a classification problem where information is provided by different sensors. We introduce a prior model which exploits the correlations between the samples provided by each sensor and a likelihood function which links the information provided by the sensors. Variational Bayes inference is used to approximate the posterior distribution of the model unknowns. In contrast to other methods in the literature, our developed model can handle sensors which do not always observe all their associated features. The method is trained with the available information and provides predictions for complete as well as incomplete testing samples. We also study the relationship with the Bayesian Co-training model and demonstrate which are the main advantages of the developed model. In the experimental section, the developed method is evaluated on both synthetic and real datasets, and is compared with other methods in the literature. The results justify the applicability of our developed algorithm.
CHAPTER 3

Machine Learning for Gravity Spy: Dataset and Glitch Classification

3.1. Introduction

The recent observations of gravitational waves from binary black hole mergers [4-6, 142] have inaugurated a new field of observational astronomy by providing a new method with which to explore the cosmos. These observations, made by the advanced Laser Interferometer Gravitational-wave Observatory (LIGO, [98]), require sensitivity to fractional changes of distance on the order of $10^{-21}$. To achieve this unprecedented sensitivity, all sensitive components of LIGO are exquisitely isolated from non-gravitational-wave disturbances. Even with this isolation, the LIGO detectors are susceptible to disturbances that cause noise in the detectors and can afflict searches for gravitational waves.

Of particular concern are transient, non-Gaussian noise sources known colloquially as glitches. Glitches occur at a significant rate, come in many morphologies, and can mask or mimic gravitational-wave signals. Work has been done in assessing whether or not a bit of excess noise is, in fact, a glitch [25], but a comprehensive classification and characterization of these noise features could allow their origin to be identified and their root cause to be removed from the instruments. Attempts to use machine learning algorithms have shown promise in glitch classification endeavors [112, 113, 123, 124, 126], however these techniques do not yet capture the full range of glitch morphologies present in LIGO data. In addition to
the above methods, *Gravity Spy*\(^1\) [154], a citizen science project hosted by the Zooniverse platform [27] that combines the classification power of machine learning and crowd-sourcing, provides a solution for addressing this problem. A critical component to the Gravity Spy method is the dataset used in training both the machine learning algorithm and the citizens. More details about the Gravity Spy project can be found in [154].

In this chapter, we present the Gravity Spy dataset, which is a collection of images of glitches and their associated metadata, in the context of machine learning tasks. We discuss the characteristics of the glitch classes within this data and provide an example for each class. To illustrate the complexity of the data and provide a better understanding of the relationship between various glitch classes, we visualize the feature space of Gravity Spy dataset. We apply state-of-the-art machine learning algorithms such as support vector machines and deep neural networks on this dataset to provide baselines for future works on this dataset.

In the following sections, we describe the process of producing the images of glitches from raw gravitational-wave detector data and investigate the feature space of all classes in the dataset (Section 3.2). Finally, different machine learning baselines, with performance evaluation and data analysis for supervised classification tasks, are presented in Section 3.3. In Section 3.4, a framework is introduced for glitch image classification using machine learning and Citizen science. Concluding remarks are made in Section 3.5.

### 3.2. Gravity Spy Data

Gravitational-wave data, including transient noise in the detectors, is often visualized as time-frequency spectrograms. The images in the Gravity Spy dataset (used for both human classification and machine learning tasks) are a particular type of spectrogram based on

\(^1\)www.gravityspy.org
decomposition using sine-Gaussian templates, a process known as the \textit{Q-transform} [33]. The Gravity Spy dataset is composed of Q-transform images of any transients recorded by the gravitational-wave channels of the detectors that exceed a certain threshold in loudness, specifically the signal-to-noise ratio (SNR), and pass the standard set of data quality criteria [3] used by LIGO’s real-time gravitational-wave searches.

Additionally, the Gravity Spy machine learning algorithms required a large training set of example images that belong to classes of morphologically distinct glitches to be constructed in order to allow machine learning pre-classification of the images that would be presented to citizen scientists [154]. To accomplish this, 22 different morphologically distinct classes of glitches were selected (the names and morphology of many of these classes had already been identified by the broader LIGO Scientific Collaboration [1,3,117]) and tens to hundreds of example images were hand selected (often with input from algorithms, such as the Hierarchical Veto [137] that identify classes of glitches by their relationship with other types of disturbances, such as seismic noise). These classes are also the classification choices (buttons) that citizen scientists have to choose from the Gravity Spy project interface.

Over time, these training sets have updated/expanded. When both the machine learning and volunteer classification of a given unlabeled image passes a certain confidence threshold [154], these images are referred to as “retired” images and added to the training set. Furthermore, new glitch classes identified by citizen scientists, volunteers, or clustering algorithms are manually (following discussion by the science team) added to the training set. More detailed information about this process can be found in [154]. Because the training set is continually evolving, this chapter presents only the initial training dataset which does not include the “retired” images for the purpose of creating a static reference for other groups interested in glitch classification.
3.2.1. Data Preparation

To construct the dataset, Gravity Spy relies on glitch triggers (parameters of known glitches) provided by the Omicron transient search algorithm [101, 129]. Omicron, which itself is based on the Q-transform, uses sine-Gaussian wavelets to identify transient excess noise (and gravitational-wave signals) in the data and clustering to identify metadata features about this excess noise such as peak frequency and SNR. In order to focus on the excess noise that is most troubling to gravitational wave searches, the following filters are applied to the glitch triggers. First, glitches that occurred outside of time periods when their detector was in “observing mode” are rejected. Observing mode indicates that the configuration and state of the detector is nominal and the data is ready to be searched for gravitational waves [3]. Second, glitches with SNR (as reported by Omicron) below 7.5 dB are rejected, since glitches below this threshold are difficult to see in the images and thus difficult to classify based on their morphology. Third, glitches whose peak frequency falls outside of LIGO’s most sensitive frequency band, from 10 Hz to 2048 Hz, are also rejected.

After applying these filters, time-frequency representations, called Omega Scans [33] and again based on the Q-transform, of the raw data for the remaining glitches are created and saved as image files. Omega Scans, which originated as part of a pipeline for the detection of gravitational wave transients [33], are outstanding at visualizing glitches with similar time-frequency morphology as transient gravitational-wave signals. Omega Scans represent a generic signal as a combination of sine-Gaussians. Omega Scans perform an unmodeled SNR calculation with the template for a signal defined by its ‘Q’ value, where Q is the quality factor of a sine-Gaussian waveform. An Omega Scan searches over a range of time-frequency tilings constructed using different Q templates and identifies the template that gives the loudest
Figure 3.1. Omega Scan images, with four different durations, of a glitch in the Koi Fish class.

SNR value, then generates a spectrogram image of the time-frequency tiling corresponding to that Q. The color axis of this image is the normalized energy (directly related to the SNR), defined as the square of a given tile’s Q transform magnitude divided by the mean squared magnitude in the presence of stationary white noise:

\[ Z = \frac{|X|^2}{\langle |X|^2 \rangle} \]  

(3.1)

where \( Z \) is the normalized energy and \( |X| \) is the Q transform magnitude of a tile [33].

Figure 3.1 shows example Omega Scan images for a glitch from the Koi Fish class (described below) at the LIGO Hanford observatory (LHO). The second LIGO observatory is located at Livingston (LLO). Each Omega Scan image is centered on the time of maximum energy of the glitch, and each glitch is visualized using four different time windows (± 0.25, 0.5, 1.0, and 2.0 seconds, referred to as view 1, view 2, view 3, and view 4, respectively in the rest of this chapter) to accommodate the durations that are most common for the different glitch classes. For the classification of a given glitch, both citizen scientists and machine learning algorithms are presented with images of all four time durations. For more details about views, i.e., time durations, see [16].
3.2.2. Dataset Specifications

Below are listed the 22 classes of glitches included in this dataset. Examples from each class are shown in Figure 3.2. The names associated with these classes and the typical morphology of the glitches belonging to each class were set by a combination of LIGO scientists, Gravity Spy scientists, and in some cases, citizen scientists. Many of the glitch classes listed here, and in some cases their physical causes, were previously identified in work to characterize the LIGO data [1, 3, 117]. Glitch classes for which LIGO scientists have uncovered or fixed the cause are also highlighted below.

These classes are not exhaustive and they are not static. Because environmental conditions at the sites change with weather and seasons and because the LIGO detectors are under active commissioning to bring them to their designed levels of sensitivity and robustness, the classes of glitches change with time. There are many possible sub-classes of glitches and there are sometimes new or short-lived classes of glitches. These 22 classes are an attempt at delineating the most representative and distinguishable classes as a snapshot in time, circa 2015-2017.

3.2.3. Gravity Spy Dataset Feature Space

To better understand the Gravity Spy dataset, particularly the (dis)similarity between members of these glitch classes, we visualize the glitch images in the raw pixel feature space. The original glitch images are 570 × 470 pixels. We downsample them (to reduce the computation cost) and then employ the t-distributed Stochastic Neighbor Embedding (t-SNE) method [103] to project them into a two dimension feature space (see Figure 3.3).
Figure 3.2. Omega Scan images for example members of each class within the Gravity Spy dataset. From top left to bottom right: 1080Lines, 1400Ripples, Air Compressor, Blip, Chirp, Extremely Loud, Helix, Koi Fish, Light Modulation, Low Frequency Burst, Low Frequency Lines, No Glitch, Paired Doves, Power Line, Repeating Blips, Scattered Light, Scratchy, Tomte, Violin Mode, Wandering Line, Whistle, None of the Above (one possible example, this class can have various forms).
t-SNE is a nonlinear method for dimensionality reduction to visualize high-dimensional data (which was described briefly in section 4.3.2). It projects the original high-dimensional feature space into a new lower dimensional (usually two) feature space such that the similarities between points would be preserved. This means that points that are close in the original high-dimensional feature space should remain close in the low-dimensional feature space and dissimilar points in the original space should be far from each other in the projected space.

Examining this low dimensional representation obtained by t-SNE, we observe that some classes are well separated in this space while others overlap. For instance, members of the Extremely Loud class are separated into three distinct groups, one of which overlaps with the Koi Fish class (these are the extremely loud Koi Fish). Similarly, there is a large cluster of Blip glitches that are mixed with Repeating Blips which are known to be morphologically similar. We also observe that few samples from the Koi Fish class are close to the Blip class which makes sense for the less loud Koi Fish glitches, which look similar to Blips. Scattered Light and Low Frequency Line also have a small overlap which is again consistent with citizen scientists’ experience with these classes. Other points of Scattered Light are well clustered on the top right of the plot. In general, None of The Above glitches are spread all over the space and overlap with other classes. This make sense because this catch-all group can have varied morphology. However, that also makes the classification of glitches into this class challenging and highlights the need for creating more classes that better capture the glitches currently put into None of The Above.

Near the top of the plot, a cluster of Tomte glitches is very close to the None of The Above glitches. This lack of separation needs further investigation. On the mid left of the plot, we observe that Light Modulation glitches and Low Frequency Burst glitches are close to each other and have some overlap, likely due to the fact that Light Modulation glitches are often
accompanied by low-frequency glitches (indeed our citizen scientists often mix classifications between these two classes). On mid bottom, there is a small green cluster (Air Compressor) which has some pink ‘+’ samples (Power Line), these glitches do look morphologically similar;
however, the former occurs usually at 50 Hz while the later occurs at 60 Hz. The most dense area of overlap is in the low-mid center of the plot where Whistle, Violin Mode, 1080Lines, and Wandering Line glitches are found. These are all relatively high-frequency effects, but should have morphological distinctions, however, especially for whistles while the shape of each glitch usually looks like a “V” or a “W” these may be narrow or wide, and thus may resemble these other categories. Overlaps between these classes may be improved in the future through better class definitions and improvements to the glitch classification training set.

3.3. Classification Methods

In this section, we first describe the application of three classification methods on the GravitySpy dataset. It has a total of 8583 glitch samples from 22 classes. The dataset is split into 6008 train, 1288 validation, and 1287 test samples. As the distributions of the samples in various classes are highly skewed, we force all classes to be populated in all training, validation, and test sets proportional to their distribution in the whole dataset.

The methods employed here are well-known and state-of-the-art from two main categories of machine learning algorithms: shallow models, such as logistic regression and support vector machines, which can be considered as traditional approaches in machine learning and the more recent deep neural network models [92].

3.3.1. Logistic Regression

Logistic regression is a discriminative linear classifier that finds a hyperplane to separate the samples [147]. We use the multinomial logistic regression which is the extension of the standard binary logistic regression to multi-class scenarios [90, 147]. We denote the training
set of \( N \) labeled examples as \( \mathcal{D} = \{(x_1, y_1), \ldots, (x_N, y_N)\} \), with \( x_i \) representing the input samples and \( y_i \in \{1, \ldots, C\} \) are the corresponding labels, where \( C \) is the total number of glitch classes. Multinomial logistic regression minimizes the following objective function with respect to the unknown hyperplane parameters \( \theta^c \) per class \( c \)

\[
\mathcal{J}(\theta) = \sum_{i=1}^{N} \sum_{c=1}^{C} 1\{y_i = c\} \log \frac{\exp((\theta^c)^T \times x_i)}{\sum_{c=1}^{C} \exp((\theta^c)^T \times x_i)}
\]  

(3.2)

where \( 1\{a = b\} \) is the indicator function that returns one if the two input arguments are equal and zero otherwise.

The model is trained using the training set and evaluated over the test set. As logistic regression does not have any hyperparameters, we do not use the validation set here. We use the scikit-learn \([28]\) implementation of multinomial logistic regression with newton-cg as the solver.

The original resolution of the Gravity Spy spectrogram is \( 470 \times 570 \). As a first step, we investigated the effect of the resolution of the spectrograms on the performance of the classifier. If a lower resolution were to perform better (or at least as good as a higher resolution) then a double benefit would be obtained of higher performance and a lower computation cost.

The performance of multinomial logistic regression for different resolutions and different views is shown in Figure 3.4. The original glitch images are down-sampled by a factor of 0.1, 0.2, 0.3, 0.4 and 0.6 resulting in glitch images of size \( 47 \times 57 \), \( 93 \times 113 \), \( 140 \times 170 \), \( 186 \times 226 \), and \( 282 \times 342 \) pixels, respectively. Downsampling was performed with the ‘rescale’ function and the glitches were converted to gray-scale with the ‘rgb2gray’ function. Both functions are in skimage toolbox. As can be observed from Figure 3.4, the classification accuracy initially improves by increasing the resolution but after a point decreases (exclusively view 1 which
Figure 3.4. Performance of multinomial logistic regression applied to different views using glitch images with various resolutions.

performs poorly). Considering also the performance of the classifier across views, we observe that the performance of view 1 (shortest duration) is poor, while the performance for view 4 (longest duration) is inconsistent across resolutions and lower than views 2 and 3 for all other than one resolution. Based on the specification of different glitches, some glitches have short duration and others have long duration. Therefore, using only view 1, may not provide enough information regarding long duration glitches. Similarly, using only view 4 may not provide enough resolution for short duration glitches. Based on these observations we adopt resolution $140 \times 170$ pixels for the remaining of this chapter as it seems that this resolution does not sacrifice accuracy for the computation cost.
3.3.2. Support Vector Machine (SVM)

SVM was proposed in 1992 and it quickly became the state-of-the-art of machine learning methods for classification for many years \[107\]. These days, although deep learning approaches outperform SVM in many applications, SVM is still competitive in some cases, especially for a small dataset.

This classifier finds the optimal set of hyperplanes with the largest minimum distance, i.e., maximum “margin”, between classes. Two types of SVMs, e.g., linear and kernel, are used here.

3.3.2.1. Linear SVM. Having a training set of \(N\) samples \(\{x_i\}_{i=1}^{N}\) and training labels \(y \in \{1, -1\}\), SVM solves the following optimization problem

\[
\min_{w,b} \frac{1}{2} \| w \|^2_2 + C \sum_i \zeta_i \quad \text{s.t.} \quad y_i(w^T x_i + b) \geq 1 - \zeta_i \quad \text{and} \quad \zeta_i \geq 0, \quad \text{for} \ i = 1, \cdots, N \quad (3.3)
\]

where \(C\) is the capacity constant, \(w\) and \(b\) are the hyperplane parameters (\(w\) is the vector of coefficients and \(b\) is a constant), and \(\zeta\) is a slack variable for handling non-separable inputs and allowing approximate solutions when there is not a feasible one. For a small value of \(C\), we do not penalize slack variables \(\zeta\), and as we increase \(C\), we obtain a large margin for SVM. \(C\) is a parameter which controls the trade-off between penalizing the slack variables \(\zeta_i\) and a large margin for the classifier \[147\]. The model’s hyperparameters are tuned by grid search and n-fold cross validation. After finding the optimal hyperparameters, the classifier is trained by using just the training set and the classification performance on the test set is reported.

We use the scikit-learn \[28\] implementation of linear SVM, “LinearSVC” with internal implementation of liblinear \[51\]. LinearSVC uses one-vs-the-rest scheme in which \(M\) classifiers
Table 3.1. Overall accuracy of linear SVM. Hyperparameter $C = 0.1$ (obtained by grid search and n-fold cross validation)

<table>
<thead>
<tr>
<th>view 1 (0.5 sec)</th>
<th>view 2 (1 sec)</th>
<th>view 3 (2 sec)</th>
<th>view 4 (4 sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>93.93</td>
<td><strong>96.19</strong></td>
<td>95.88</td>
<td>93.16</td>
</tr>
</tbody>
</table>

are trained, where $M$ is the number of classes. The $i^{th}$ SVM classifier is trained with the examples in the $i^{th}$ class with positive labels, and the examples from all other classes with negative labels. For each test sample, having the decision values from all $M$ classifiers, the resulting label would correspond to the class which has the largest decision function value.

The classification accuracies of trained linear SVM applied to test images of different views are reported in Table 3.1. As can be seen, linear SVM performs the best when applied to view 2 and the worst when applied to view 4 which is inline with our prior investigation for logistic regression and deep models [16].

3.3.2.2. Kernel SVM. In many cases, the complexity present in the data cannot be captured by linear models. The kernel trick is applied in such scenarios as it projects the samples into another feature space where they can be linearly separated. We use SVM with the Radial Basis Function (RBF) kernel to capture the nonlinearity of Gravity Spy samples. Kernel SVM optimizes the following objective function

$$\min_{w,b} \frac{1}{2} \| w \|^2 + C \sum_i \zeta_i \quad \text{s.t.} \quad y_i (w^T \phi(x_i) + b) \geq 1 - \zeta_i \quad \text{and} \quad \zeta_i \geq 0 \quad \text{for } i = 1, \cdots, N$$

(3.4)

The function $\phi(x_i)$ does not need to be explicitly defined. Instead, only the kernel $K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$ needs to be defined.

The RBF kernel function is given by $K(x_i, x_j) = \exp(-\gamma \| x_i - x_j \|^2_2)$. The hyperparameter $\gamma$ controls the trade-off between error due to bias and variance in the model. A very
Table 3.2. Overall accuracy of RBF kernel on different views. The RBF kernel hyperparameters are $C = 5.65$ and $\gamma = 4e^{-5}$

<table>
<thead>
<tr>
<th>view 1 (0.5 sec)</th>
<th>view 2 (1 sec)</th>
<th>view 3 (2 sec)</th>
<th>view 4 (4 sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>95.41</td>
<td>97.12</td>
<td>96.50</td>
<td>93.93</td>
</tr>
</tbody>
</table>

large value of $\gamma$ can lead to a very complex boundary with low bias/high variance and also having an overfitting problem. On the other hand, a very small value of $\gamma$ has the risk of a model with high bias/low variance. Here, for multi-class scenario, we use the one-vs-one scheme. This method constructs $\frac{M \times (M-1)}{2}$ classifiers, where $M$ is the number of classes. Each classifier is trained on a pair of two classes. For a given test point, each classifier gives a vote and the final label is assigned to the class with the largest number of votes (majority voting scheme). The hyperparameters of the RBF classifier, $C$ and $\gamma$, are obtained using grid search and n-fold cross validation.

The classification accuracies of trained kernel SVMs applied to test images in different views are reported in Table 3.2. The optimal hyperparameters are found to be $C = 5.65$ and $\gamma = 4e^{-5}$. As can be observed from Table 3.2, kernel SVM has the best performance when applied to view 2 and has the worst performance when applied to view 4.

3.3.3. Deep Neural Networks (DNNs)

Motivated by the impressive performance of DNNs for solving many classification problems [92], we choose them as one of the baselines for glitch classification for the Gravity Spy dataset.

Deep Convolutional Neural Networks (CNNs) [91] have shown superior performance on image data [84]. When CNNs are used for classification, they are usually composed of convolutional and max-pooling layers, followed by fully connected and softmax layers.
In the following, we describe the layers used in CNN architecture, briefly.

### 3.3.3.1. Layer types.

- **Convolution layer**

  Convolutional layer has a number of filters, i.e., kernels, that slide over the input matrix and multiply their corresponding weights to the input matrix and produce the so-called feature maps. Assume the input to the convolutional layer is a $m \times n \times c$ matrix, where $m$ and $n$ correspond to the height and weight of image, respectively, and $c$ is the number of channels. For example, if the input is the original image (given to the first layer), $c$ is 3 in RGB scale and 1 in gray-scale format. The size of filters are smaller than the input. Using $K$ filters with the size of $f \times f \times c$, the output dimension is $K \times (m - f + 1) \times (n - f + 1) \times c$. Each $(m - f + 1) \times (n - f + 1)$.

- **Activation function and pooling layer**

  A layer of activation function is used after the convolution layer to determine the neuron’s output. A popular activation function is rectified linear unit (ReLU) which is $\max(0, x)$ and we have used that in our model as well. Feature maps are usually subsampled using max (mean) operation. A square matrix slides over the feature map and gives the maximum (mean) value among the elements inside it. In our network, we used max-pooling layer. The size of the max-pooling matrix is usually between $2 \times 2$ and $5 \times 5$. Therefore, it can subsample the feature maps between 2 to 5 times, depending on its size.

- **Fully connected layer**

  Each node in fully connected (FC) layer is connected to all nodes of the previous layer. The dimension of the weight matrix between a FC layer and its previous layer is $m_{n-1} \times m_n$, where $m_n$ is the number of nodes of the $n^{th}$ layer. Hence, the number
of parameters in FC layers are much higher than in convolution layers, and careful consideration is sometimes needed to prevent overfitting.

- **Softmax layer**

  Softmax is a FC layer popular for multi-class classification problems. The softmax layer size is equal to the number of the target classes. The output of softmax layer is defined by Eq. (3.5).

\[
o^i_c = \frac{e^{w_c^T x}}{\sum_{c=1}^{C} e^{w_c^T x}} \quad \text{for } c = 1, \ldots, C
\]

(3.5)

where \( o^i_c \) denotes the \( c^{\text{th}} \) class score for the \( i^{\text{th}} \) image as input to the model, \( x \) is the input given to the softmax layer (the output of the layer before softmax), \( w_c \) is the weight vector connecting to the \( c^{\text{th}} \) node in the softmax layer and \( C \) is the total number of classes.

### 3.3.3.2. Training

The models presented above optimize a loss function defined on the training data. For training the model, we can use either the mean squared error or the average cross-entropy error as the loss function. Due to the advantages of the average cross-entropy error over the mean squared error, e.g., the derivation of a “better” gradient for back propagation, for multi-class classification problems [59], in our model we use a cross-entropy based loss function defined as follows:

\[
\text{objective function} = - \sum_{i=1}^{N} \sum_{c=1}^{C} y^i_c \log o^i_c
\]

(3.6)

where \( y^i_c \) denotes the binary label for sample \( i \) (it is one only when the sample \( i \) belongs to class \( c \)).
There exits many optimization techniques [81, 116, 135, 153] that we can use to optimize the objective function. We use the Adadelta [153] optimizer. It monotonically decreases the learning rate and shows good performance in our experiments.

We use Keras [37] library with Theano [20] back-end for deep learning implementations. For Adadelta parameters, the default values of Keras 2 are used. The initializer for the convolutional layer is ‘glorot uniform’ [58] that draws samples from a truncated normal distribution. We use $l^2$ regularization. The weight of the regularization for the layers in Table 3.3 is $w_{\text{reg}} = 2 \times 10^{-4}$. Using Keras Model Check Point, the performance of the model on the validation set is checked at the end of each epoch and if there is an improvement over the best model so far, the weights are saved as the new best weights. At the end of all iterations, the best weights are loaded for the final model. In our experiments, the number of epoch is set to 200, and the batch size is equal to 32. The accuracy of glitch classification for different durations is shown in Table 3.4.

A straightforward approach is to use just one glitch duration, as is done in a traditional single view approach. We use this as a baseline to compare the performance of our multi-view

<table>
<thead>
<tr>
<th>Single view model</th>
<th>Parallel view model</th>
<th>Merged-view model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input $140 \times 170$</td>
<td>Input four $140 \times 170$</td>
<td>Input $280 \times 340$</td>
</tr>
<tr>
<td>5 × 5 Conv. layer (32) with reg.</td>
<td>Four 5 × 5 Conv. layer (32) with reg.</td>
<td>5 × 5 Conv. layer (16) with reg.</td>
</tr>
<tr>
<td>2 × 2 Maxpooling, 0.5 drop-out</td>
<td>2 × 2 Maxpooling, 0.5 drop-out</td>
<td>2 × 2 Maxpooling, 0.5 drop-out</td>
</tr>
<tr>
<td>5 × 5 Conv. layer (64) with reg.</td>
<td>5 × 5 Conv. layer (64)</td>
<td>5 × 5 Conv. layer (32) with reg.</td>
</tr>
<tr>
<td>2 × 2 Maxpooling, 0.5 drop-out</td>
<td>2 × 2 Maxpooling, 0.5 drop-out</td>
<td>2 × 2 Maxpooling, 0.5 drop-out</td>
</tr>
<tr>
<td>FC (256), 0.5 drop-out</td>
<td>FC (256), 0.5 drop-out</td>
<td>5 × 5 Conv. layer (64) with reg.</td>
</tr>
<tr>
<td>Softmax (22)</td>
<td>Softmax (22)</td>
<td>Softmax (22)</td>
</tr>
</tbody>
</table>

Table 3.3. Specifications for single and multi-view CNN used for glitch classification. We use the abbreviation of ‘Reg’ for regularizer. The number in parenthesis in front of convolutional and fully connected layer show the number of kernels and nodes, respectively.
Table 3.4. Accuracy of CNN on different glitch durations, parallel-view and merged-view model in [16] with new architecture.

<table>
<thead>
<tr>
<th>view 1 (0.5 sec)</th>
<th>view 2 (1 sec)</th>
<th>view 3 (2 sec)</th>
<th>view 4 (4 sec)</th>
<th>parallel-view</th>
<th>merged-view</th>
</tr>
</thead>
<tbody>
<tr>
<td>95.10</td>
<td>96.81</td>
<td>96.58</td>
<td>95.65</td>
<td>97.12</td>
<td>97.67</td>
</tr>
</tbody>
</table>

deep models. For single view models, we use CNNs with the structure shown in the left column of Table 3.3. The architecture of CNNs is optimized for the best classification accuracy. We also report the classification accuracies of two developed multi-view models [16] which are going to be described in section 3.3.3.4.

3.3.3.3. Analysis. As the results in Table 3.4 show, the performance of multi-view deep models is better than single view models. An example of a misclassified sample by all single view models that was classified correctly by the multi-view models is shown in Figure 3.5. Such examples show that in many cases, single view models do not have the needed sight and horizon for recognizing glitches correctly. Glitch classes are divided into short and long duration based on the glitch duration. In Table 3.5, we show the category of each class plus the accuracy of two of the single view models; the 0.5-second model (Classifier 1) and 4-second model (Classifier 4). As can be seen in this Table, Classifier 1 performs at least as good as Classifier 4 for short duration glitches, while the opposite is true for long duration glitches, as expected for some classes (e.g., “Air Compressor” and “Tomte”) the performance is perfect with both classifiers. Clearly the multi-view models which use all durations can capture the needed information to classify all types of glitches (according to their duration) more accurately.

3.3.3.4. Architectures. The main motivation for this study is to exploit multiple views for glitch classification instead of depending on just a single view. We investigate this by combining views’ information at two points as we go through the deep network layers.
Figure 3.5. An example of a glitch that was misclassified by all four single view models, but correctly classified with both of the multi-view models. The single view model, which is trained with 0.5 second duration images, classifies it as “Blip” class. The predicted class is “Repeating Blips”, “Koi fish”, and “Koi fish” for the single view models trained with 1, 2, and 4 second duration glitches, respectively. Multi-view models predict the sample correctly as belongs to the “Light Modulation” class.

Table 3.5. Classification accuracy of single view model trained with half a second duration (classifier 1) and four second duration (classifier 4) for each class. The first column shows whether the class is a short (Sh) or long (L) duration.

<table>
<thead>
<tr>
<th>Duration</th>
<th>Class</th>
<th>Classifier 1 Acc. (%)</th>
<th>Classifier 4 Acc. (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sh</td>
<td>Air Compressor</td>
<td>100.00</td>
<td>80.00</td>
</tr>
<tr>
<td>Sh</td>
<td>Blip</td>
<td>98.09</td>
<td>97.61</td>
</tr>
<tr>
<td>Sh</td>
<td>Helix</td>
<td>96.66</td>
<td>96.66</td>
</tr>
<tr>
<td>Sh</td>
<td>Power Line</td>
<td>100.00</td>
<td>100.00</td>
</tr>
<tr>
<td>Sh</td>
<td>Repeating Blips</td>
<td>80.00</td>
<td>76.00</td>
</tr>
<tr>
<td>Sh</td>
<td>Tomte</td>
<td>100.00</td>
<td>100.00</td>
</tr>
<tr>
<td>L</td>
<td>Extremely Loud</td>
<td>96.77</td>
<td>98.38</td>
</tr>
<tr>
<td>L</td>
<td>Light Modulation</td>
<td>83.14</td>
<td>86.51</td>
</tr>
<tr>
<td>L</td>
<td>Low Frequency Lines</td>
<td>89.79</td>
<td>89.79</td>
</tr>
<tr>
<td>L</td>
<td>Scattered Light</td>
<td>98.36</td>
<td>98.36</td>
</tr>
<tr>
<td>L</td>
<td>Wandering Line</td>
<td>57.14</td>
<td>71.42</td>
</tr>
</tbody>
</table>

thus develop one model in which fusion take place at an early step (referred to as “merged view”) and one in which information is integrated at the middle level (referred to as “parallel view”). In the following, we explain these two architectures in detail.
• Parallel view model

The idea behind the parallel view model is to project each view into a feature space which is based more on the statistical properties of the samples than their view-specific properties. This projection makes the views interact with each other and be presented in a common feature space more efficiently. We illustrate the construction of the parallel view model using the four durations as views in Figure 3.6. At the very first layer, each view passes through a convolutional layer, followed by max-pooling and ReLU activation. Then, we introduce a shared layer (merger layer) to map all views into a common feature space. Another set of convolutional, max-pooling, and activation layers is used after the merger layer to model the obtained common features from the previous layers. In the end, a fully connected layer and a softmax layer are employed.

• Merged view model

In this model, we introduce the network layers on top of the merged views. We merge the views by forming a $2m \times 2k$ matrix by placing next to each other four $m \times k$
images. After merging the views, a set of convolutional layer followed by activation and max-pooling layers, and then a fully connected layer and a softmax layer are exploited (see Fig. 3.7). This approach clearly models jointly the distribution of the different views in their original feature space. This seems a reasonable approach since the correlations among the views in our problem are not highly non-linear compared to other tasks where the views are very different, such as image and text [139], or audio and video [114]. Therefore, it is possible for the model to learn such correlations even in the original space of data.

The accuracy obtained by the merged-view model is shown in the right most column of Table 3.4 and the detailed specification of the model is reported in the last column of Table 3.3 and drawn in Figure 3.7. All the other hyper-parameters and configuration settings for the “merged-view” model are the same as with the single view CNNs described earlier in this section.

3.4. Glitch Classification using Machine Learning and Citizen Science

The aim of this section is to present and define the protocols and algorithms for (i) glitch image classification using multiple citizen labels and machine learning classifier outputs, (ii)
3.4.1. Citizen Scientist Modeling

Based on the expertise and reliability level, the citizens are divided into three levels: Beginner, Intermediate, and Advanced as shown in Figure 3.8. The higher the level, the more challenging
the classification task. As shown in the diagram above, each citizen starts as a Beginner and can be promoted to higher levels based on their performance.

We define a matrix for each annotator to record their previous labeling performance for different classes of the golden set. Let \( M^k \in \mathbb{N}^{C \times C} \) be the matrix defined for the \( k \)th annotator, where \( C \) denotes the total number of classes. This matrix is motivated from confusion matrix defined for classification problems in machine learning, and is defined as:

\[
M^k = \begin{bmatrix}
m_{11}^k & m_{12}^k & \cdots & m_{1C}^k \\
m_{21}^k & m_{22}^k & \cdots & m_{2C}^k \\
\vdots & \vdots & \ddots & \vdots \\
m_{C1}^k & m_{C2}^k & \cdots & m_{CC}^k 
\end{bmatrix}
\] (3.7)

where \( m_{ij}^k \) is the number of samples belonging to class \( i \) and labeled as belonging to class \( j \) by the \( k \)th citizen. It will be initiated by 0 for all entries and it is updated in two situations: 1) when a training image from the golden set is labeled by annotator \( k \) and 2) when the label of a test image is finalized, (i.e. when a test image is retired as shown by the solid blue arrows returning to the training set in Figure 3.8).

Using the confusion matrix \( M^k \), we define a reliability measure for citizen \( k \) as the vector \( \mathbf{a}^k = [\alpha_1^k, \cdots, \alpha_c^k, \cdots, \alpha_C^k] \in \mathbb{R}^{C \times 1} \), where its \( c \)th element, denoted by \( \alpha_c^k \) quantifies the reliability of annotator \( k \) in classifying samples of class \( c \), (or the sensitivity of \( k \)th citizen for the class \( c \)). It is defined as:

\[
\alpha_c^k = \frac{m_{cc}^k}{\sum_{i=1}^C m_{ci}^k} = p(\hat{y}^k = c|y = c) \quad \text{for} \quad c \in \{1, \cdots, C\}
\] (3.8)
\( \alpha_c^k \) is also equal to the probability that the \( k^{th} \) citizen provides a label \( \hat{y}^k \) for an image, as belonging to class \( c \) class, given the true label \( y \) is indeed equal to \( c \).

### 3.4.2. Images Classification from Multiple Annotations and Machine Learning

Having discussed how to model the citizens’ ability in recognizing different classes in the previous section, we discuss here how to classify an unknown test sample using multiple annotations.

Let the training set be \( \mathbb{D}^{tr} = \{(x^{tr}_i, y_i), i = 1, \ldots, N_{tr}\} \), where \( N_{tr} \) is the number of images in the training set and \( x^{tr}_i \) is the \( i^{th} \) training image vector, \( y_i \in \{1, \ldots, C\} \) denotes the known true label of the \( i^{th} \) image; This set is used for different purposes: 1) training the ML classifier and citizens and 2) building the confusion matrix for each citizen.

As can also be seen in Figure 3.8, a test image is initially provided to the ML classifier. The ML classifier outputs a probability vector of the given image belonging to each class. We denote this vector for image \( i \) by \( \mathbf{p}_i^{ML} \in \mathbb{R}^{C \times 1} \). Depending on this probability vector, this test sample is forwarded to the citizens at a given level (see blue arrows emanating from the ML classifier in Figure 3.8). Then, citizens provide hard labels to this image. We denote the assigned label by the \( k^{th} \) annotator to the test sample \( i \) by \( \hat{y}_i^k \in \{1, \ldots, C\} \). Let the test set be \( \mathbb{D}^{ts} = \{(x^{ts}_i, \hat{y}_1^i, \ldots, \hat{y}_R_i^i, \mathbf{p}_i^{ML}), i = 1, \ldots, N_{ts}\} \), where \( N_{ts} \) is the total number of images in the test set and \( R_i \) is the number of citizens that have labeled image \( i \). Our algorithm uses the ML probabilities and the hard labels given by the annotators to predict the true label. Having the assigned labels by \( R_i \) annotators for a given image \( i \), the goal is to fuse these labels and find the posterior probabilities \( p(y_i^{cr} = j|\hat{y}_1^i, \ldots, \hat{y}_R_i^i) \) for \( j \in \{1, \ldots, C\} \), where \( y_i^{cr} \) is the predicted label from crowdsourcing and ML information. We find the final
predicted label $\tilde{y}_i$ as:

$$\tilde{y}_i = \arg\max_j \frac{p(y_i^c = j | \hat{y}_i^1, \cdots, \hat{y}_i^R) + p_{i}^{ML}(j)}{\sum_{j=1}^C p(y_i^c = j | \hat{y}_i^1, \cdots, \hat{y}_i^R) + p_{i}^{ML}(j)}$$

(3.9)

Where $p_{i}^{ML}(j)$ denotes the $j^{th}$ component of $p_{i}^{ML}$.

Before explaining how these posteriors are estimated, we want to clarify how the updating system works. The posterior probabilities are updated after specified time intervals. The images utilized during such intervals form a “batch”. There are two algorithms for updating posteriors. A parallel algorithm considers all the annotations at once inside a batch, “intra-batch” algorithm. After updating is performed inside the batch, the posterior probabilities are propagated to the next time window if needed. To do this, we introduce another algorithm, referred to as “inter-batch” algorithm which uses the posteriors from the previous batch as prior knowledge. In the following sections, we discuss these algorithms in more details. Clearly this time interval can be as long or short we desire, and if made too short the updating of the probabilities can take place after each “click”.

3.4.2.1. Intra-batch Algorithm. To estimate the posteriors inside a batch, first we convert the hard labels $\hat{y}_i^k$, given by annotator $k$, to soft labels, that is, we compute the probability $p(y_i^c = j | \hat{y}_i^k = l)$ and estimate the posterior probability of an image being labeled as belonging to a specific class, provided the hard label assigned by the $k^{th}$ annotator. In this way, we utilize the expertise level of annotators in our fusion framework. The posterior probability is calculated from the confusion matrix according to

$$p(y_i^c = j | \hat{y}_i^k = l) = \frac{p(\hat{y}_i^k = l | y_i^c = j) \cdot p(y_i^c = j)}{\sum_{j=1}^C p(\hat{y}_i^k = l | y_i^c = j) \cdot p(y_i^c = j)},$$

(3.10)
where \( l \) is the annotator label and \( l \in \{1, \ldots, C\} \). To calculate the posterior probability of Eq. 3.10, we need to find \( p(\hat{y}_i^k = l|\hat{y}_i^{cr} = j) \) which can be obtained from the confusion matrix of the \( k^{th} \) annotator, \( M^k \) and \( p(\hat{y}_i^{cr} = j) \) which is the prior knowledge on the \( j^{th} \) class we can obtain from our training set \( \mathbb{D}^{tr} \). We define the prior probability \( p(\hat{y}_i^{cr} = j) \) as:

\[
p(\hat{y}_i^{cr} = j) = \frac{\# \text{ images in the } j^{th} \text{ class in } \mathbb{D}^{tr}}{\# \text{ images in } \mathbb{D}^{tr}}, \quad j \in \{1, \ldots, C\} \tag{3.11}
\]

Alternatively a flat prior can be used.

To calculate \( p(\hat{y}_i^k = l|\hat{y}_i^{cr} = j) \), we use the confusion matrix entries \( M^k \) according to

\[
p(\hat{y}_i^k = l|\hat{y}_i^{cr} = j) = \frac{m_{jc}^k}{\sum_{c=1}^C m_{jc}^k} \tag{3.12}
\]

After the posterior probability per class and annotator is calculated, which we rename as \( p_j^i = p(\hat{y}_i^{cr} = j|\hat{y}_i^k = l) \), we form a posterior probability matrix \( \mathcal{P}_i \in \mathbb{R}^{C \times R_i} \) for each image \( i \), that is,

\[
\mathcal{P}_i = \begin{bmatrix}
p_1^1 & p_1^2 & \cdots & p_1^{R_i} \\
p_2^1 & p_2^2 & \cdots & p_2^{R_i} \\
\vdots & \vdots & \ddots & \vdots \\
p_C^1 & p_C^2 & \cdots & p_C^{R_i}
\end{bmatrix}
\tag{3.13}
\]

To perform fusion, for each image, we take the summation of posterior probabilities over the annotators (rows in matrix \( \mathcal{P}_i \)) in a vector \( \mathbf{z}^i = [z_1^i, \cdots, z_C^i] \), in which \( z_j^i \) is defined according to

\[
z_j^i = \sum_{k=1}^{R_i} p_j^k \quad j \in \{1, \cdots, C\} \tag{3.14}
\]

i.e. \( z_j^i \) is the summation of the probabilities of \( j^{th} \) row of matrix \( \mathcal{P}_i \).
To find the posterior probability in Eq. 3.9, we normalize the $z^i$ entries, that is,
\[ p(y^c_r = j | \hat{y}_1^i, \cdots, \hat{y}_n^i) = \frac{z^j_i}{\sum_{j=1}^{C} z^j_i} \]  
(3.15)

Adding the information of the ML classifier, the final predicted label $\tilde{y}_i$ is obtained from Eq. 3.9. For simplicity, we rename the posterior probability of image $i$, belonging to class $j$ by $v^i_j$ defined as
\[ v^i_j = \frac{p(y^c_r = j | \hat{y}_1^i, \cdots, \hat{y}_n^i) + p_{ML}^i(j)}{\sum_{j=1}^{C} p(y^c_r = j | \hat{y}_1^i, \cdots, \hat{y}_n^i) + p_{ML}^i(j)} \]  
(3.16)

3.4.2.2. Inter-batch Algorithm. The posterior probabilities from the previous batch should be transmitted to the new batch. To do that, the priors are replaced by the posterior probability of each class from the previous time window. Assume annotator $k+1$ labels the image and we have obtained the posterior probability $p(y^c_r = j | \hat{y}_1^i, \cdots, \hat{y}_n^i) = p(y^c_r = j | D_k)$ by using the information provided by the previous $k$ annotators, Where $D_k$ denotes the information provided by the previous $k$ annotators.
\[ p(y_i = j | \hat{y}_i^{k+1} = l, D_k) = \frac{p(\hat{y}_i^{k+1} = l | y_i = j, D_k) \cdot p(y_i = j | D_k)}{\sum_{j=1}^{C} p(\hat{y}_i^{k+1} = l | y_i = j, D_k) \cdot p(y_i = j | D_k)}, \]  
(3.17)

We continue updating the posterior probabilities until an image is retired or the maximum number of annotators have labeled an image and it should be investigated by citizens in higher levels.

3.4.3. Retiring an Image

To decide on the retirement of the test image, a threshold $t_j$ is defined per class based on the difficulty of each class. The threshold vector can be thus defined as $t = [t_1, t_2, \cdots, t_C]^T$
Having the posterior probabilities of all the classes from Eq. 3.16 and putting them in a vector $\mathbf{v}^i = [v_i^1, \cdots, v_i^C] \in \mathbb{R}^C$, we can compare the posterior probability vector $\mathbf{v}^i$ with threshold vector $\mathbf{t}$.

$$\mathbf{v}^i \geq \mathbf{t}$$ (3.18)

If an entry of $\mathbf{v}^i$ is greater than the corresponding entry of $\mathbf{t}$, the image is retired with label $j$ for which $v_i^j \geq t_j$. Then this retired images is sent to training set with label $j$ as its true label.

Otherwise, if no entry of $\mathbf{v}^i$ is greater than the corresponding entry of $\mathbf{t}$, further action is needed based on the number of annotators $R_i$ who have labeled the $i^{\text{th}}$ image. We define a threshold value on the number of annotators $R$ who should label the image in each level. If $R_i$ is smaller than $R$, we wait for more annotators at the same level to label the image. Otherwise this image should be passed to more skilled people, the citizens in the upper level.

### 3.4.4. Promoting the Citizens

As the citizens label the training data, their confusion matrices are updated. Also, as some test images from each batch are retired, the training set is updated. Then, the citizen labels are compared with the labels of the retired images and the confusion matrices are also updated. With Eq. 3.8, the $\mathbf{a}^k$ vector is calculated from the confusion matrix $\mathcal{M}^k$. We define threshold values on reliability values for each class ($T_j$ for $j \in \{1, \cdots, C\}$). Let it be $\mathbf{T}_a = [T_1, \cdots, T_C]$. The following decision rule is used:

$$\mathbf{a}^k \geq \mathbf{T}_a$$ (3.19)
If all the values of $a^k$ vector exceeds the threshold values in $T_\alpha$, the citizen is promoted to next level. If not, (s)he should do more correct classification tasks to get promoted.

### 3.4.5. Summary

Here is the summary of the developed algorithm. We denote by $\hat{y}^k = \{\hat{y}_1^k, \ldots, \hat{y}_N^k\}$ the labels provided by the $k$–th citizen, and $\hat{Y}^a = \{\hat{y}^1, \ldots, \hat{y}^R\}$ denotes the labels provided by all the annotators.

---

**Algorithm 2** Classification of the Test Images using crowdsourcing and ML

```plaintext
input: $\hat{Y}^a$, $t$, $R$ and $T_\alpha$. 
1: for $j = 1$ to $C$ do
2:   Calculate $p(y_{icr} = j)$ using (3.11)
3: end for
4: for $i = 1$ to $N_{ts}$ do
5:   for $j = 1$ to $C$ do
6:     Calculate $p(y_{icr} = j | \hat{y}_i^k = 1)$ using (3.10) and (3.12)
7:   end for
8:   Calculate $v$ using (3.16)
9:   Decide using (3.18)
10: if $v_i \geq t$ then
11:   Retire
12: else
13:   if $R_i \geq R$ then
14:     Forward to upper class;
15:   else
16:     Wait for more labels from the next batch;
17:     Replace $p(y_{icr} = j)$ with $p(y_{icr} = j | D_k)$;
18:     Repeat from Step 6;
19:   end if
20: end if
21: end for
```
3.5. Conclusions

In this chapter, we present the Gravity Spy dataset which includes thousands of images of LIGO glitches and the specifications for 22 glitch classes. Employing state-of-the-art classification methods, we developed several baselines for the supervised glitch classification problem. All classifiers were trained using the standard training set and the validation set was used for tuning the hyper-parameters wherever needed (the validation set samples were never added to the training set). The glitch classes present in LIGO data evolve with time and we expect to produce and study more such datasets in the future. We also develop an algorithm to classify test images fusing the output of machine learning classifier and citizen labels. This algorithm can be used to promote the citizens to higher levels and retire test images.
CHAPTER 4

Machine Learning for Cultural Heritage

4.1. Introduction

In this chapter, we study two well-known problems in cultural heritage: pigment identification and pigment unmixing. To tackle these problems, we study the state-of-the-art algorithms and also develop new algorithms to solve them.

The primary aim of identifying the pigments used on artworks is to characterize the palette of the artist, understand how these materials may change over time, to better inform conservation treatment of this invaluable cultural heritage, and to route out forgeries. More specifically, for the 2nd century AD Roman Egyptian portraits comprising this study (currently housed at the Phoebe A. Hearst Museum of Anthropology), not only help us to better understand the painting technology of this early period but also they are helping us to better characterize the global exchange and trade economy of the Roman era [145].

Reflectance spectroscopy and hyperspectral imaging are analytical methods that are becoming increasingly important for the documentation and characterization of artworks. Previous work in this area has employed a broad range of spectral information which relates the physical mineralogy of the pigment to its unique molecular spectral response. Hyperspectral imaging is inherently non-destructive and non-invasive and can be applied in-situ to recover reflectance curves on a per pixel basis. Since each pigment has a characteristic reflectance response, these data can be interrogated using dictionaries of known spectra. These spectral dictionaries may either be physically modeled from the known molecular structure of the
pigment mineralogy, learned directly from spectral samples of an artwork (e.g. learning so-called “endmembers” [43]), or they may be formed from previously collected spectra of representative pigment samples [11]. In this chapter, we investigate the latter case, where ground truth measurements of a spectral library are provided. In particular, we form an over-complete dictionary that incorporates the inherent variability in the spectral response of a given pigment due to the non-uniformity in chemical compositions, paint thicknesses, pigment-binder ratios, sensor noise, and so on. We show quantitatively that such an over-complete dictionary representation produces more accurate estimation of pigment composition than using a dictionary consisting of a small set of pure endmembers alone.

Later in this chapter, we focus on pigment unmixing using nonlinear methods. One of the primary challenges to using HSI for materials identification is that the reflectance response of a heterogeneous mixture of pigments is nonlinear. This fact has long been understood and described analytically by Kubelka-Munk theory [21]. Previous approaches to nonlinear spectral unmixing have used a simplified opaque case of Kubelka-Munk (KM) [21, 22, 56, 88, 150] that models the non-linear effects of pigment and binder mixtures. Using this KM model, reflectance spectra collected from a painted surface are fitted to dictionaries of pure compounds [44]. While this approach is computationally straightforward for pigment identification/quantification, its application is limited to single-point analyses, rather than every pixel of a hyperspectral image cube, due to extended memory requirements and computation times necessary to undertake these KM calculations. Recent work [150] has attempted to overcome this limitation by segmenting and clustering the 2D image space, based on chromatic information, thereby producing a reduced number of representative spectra prior to KM modeling. Here, we describe a novel approach to solve the pixel-wise pigment unmixing problem to produce data-rich and semi-quantitative maps of pigment
distributions. We also study the applicability of deep neural networks in pigment unmixing problem. We show that by using the Kullback–Leibler divergence loss leads to accurate coefficient estimated values.

The rest of this section is as following. First, automatic pigment identification is solved by using sparse unmixing algorithm. In Section 4.3, data reduction is described and clustering is applied. Next, we develop an algorithm for automatic pigment unmixing using gradient based method. In section 4.5, deep neural networks are used to decompose any unknown spectrum to its constituent pigments first qualitatively and then quantitatively. Finally, Section 4.6 concludes this chapter.

4.2. Automatic Pigment Identification on Roman Egyptian Paintings by using Sparse Modeling of Hyperspectral Images

Sparse modeling has been used in a diverse set of problems such as speech recognition [80] and remote sensing [74]. Here, we cast pigment identification as an unmixing problem and solve it using sparse modeling. To the best of our knowledge, we are the first to apply sparse modeling to the problem of pigment identification in paintings. We build an over-complete dictionary from hyperspectral images of a reference pigment library composed of color swatches painted out on paper and canvas supports. These pigments were selected on the basis of what we already know about Roman painting from previous studies [75] and interrogating the portraits themselves using multi-analytical approach [54]. We then use our over-complete dictionary to decompose the measured test spectra of a pigment into a sparse coefficient vector using the $\ell_1$ minimization solver [23]. The values in the sparse coefficient vector then directly correspond to abundances of pure pigments in the dictionary. We show that this approach produces quantitative performance advantages for estimating the composition of
mixed pigments, and does so without the need for an explicit threshold selection strategy. To evaluate the performance of our algorithm, we applied our method to hyperspectral images captured from both our color swatch library and the mummy portrait itself. We compare our results with commonly used spectral angle mapping (SAM) and spectral correlation mapping (SCM) techniques and show that our algorithm outperforms these methods. While we do not have exhaustive ground truth data for the pigment composition of the Fayum portrait, we compare against the ground truth pigment information of point samples measured using Raman spectroscopy. We also quantitatively evaluate the performance of our method using back-projection reconstruction error. Our method shows a significant improvement in both back-projection error and pigment decomposition relative to previous approaches.

4.2.1. Previous Works

Hyperspectral and multispectral data have already been used for pigment identification in previous studies such as [11, 41, 43, 120]. Many of them [11, 41, 43, 120] can be seen as a method just based on nearest neighbor information to classify pigments. They compare the unknown spectrum to the spectra of some reference spectra of pure pigments in a library. The whole procedure can be summarized in three steps: (i) making a spectral library (ii) defining a distance measure, (iii) assigning a label to an unknown pigment.

To build the spectral library, an endmember detection algorithm such as the pixel purity index algorithm [34] is usually used. These algorithms find the spectral signatures of the painting as pure pigments. However, they have important drawbacks. They usually need to know the number of endmembers beforehand [46]. Also, the obtained endmembers sometimes do not have any physical meanings. Once the endmembers are identified, the next step is to compare against the spectrum of a given pixel. A distance metric is used to assign the
unknown pigment to the closest endmember in the library. To assign the label, a threshold value should be defined beforehand. For a given pixel, if the maximum similarity is higher than the threshold, the pixel is labeled as a member of the class with the smallest distance. Otherwise, the pixel is assigned as unlabeled (unknown). Such approaches suffer from several drawbacks. First, defining an appropriate distance function is not straightforward. Also, the value of the threshold is another crucial parameter. It has been shown that to detect different pigments in the hyperspectral paintings, sometimes more than one threshold value should be defined [41]. Lower threshold values are suitable for some pigments and to detect other pigments, higher threshold values are used. Furthermore, the pixel is not labeled if its maximum spectral similarity is lower than the threshold. These metrics can be used for homogeneous areas and not highly mixed regions. In real paintings, there are numerous mixed pixels which are (non)linear combinations of two or more pigments.

In [41], a fully constrained spectral unmixing algorithm combined with linear mixing model [67] is applied to find the concentration of each endmember in each pixel. This work assumes that pigment spectra can be decomposed into a linear combination of endmembers from a known dictionary.

4.2.2. Model Definition

We use sparse modeling to analyze pigment spectra. A sparse vector is a vector with only few non-zero elements. The sparse representation $\alpha$ of an unknown observation $x \in \mathbb{R}^P$ with respect to a dictionary $D \in \mathbb{R}^{P \times N}$, is the result of the constrained optimization problem:

$$\text{Minimize } ||\alpha||_0 \text{ such that } x = D\alpha$$

(4.1)
where the $\ell_0$ pseudo-norm counts the number of non-zero elements in $\alpha$. The problem in Eq. (4.1) can be solved by either involving greedy algorithms such as Matching Pursuit (MP) or Orthogonal Matching Pursuit (OMP) or by using relaxation techniques [47]. One such relaxation technique we adopt here is to replace the $\ell_0$ norm with the $\ell_1$ norm defined by:

$$
\| \alpha \|_1 = \sum_{i=1}^{N} |\alpha_i |
$$

(4.2)

where $\alpha_i$ is the $i^{th}$ component of vector $\alpha$. In our experiments, we use the SUnSAL (a LASSO solver) that has shown its efficiency for sparse unmixing for remote sensing [23]. It solves the constrained optimization problem:

$$
\hat{\alpha} = \text{argmin}_{\alpha} \| x - D\alpha \|_2^2 + \lambda \| \alpha \|_1 \text{ such that } \alpha \geq 0
$$

(4.3)

where $\lambda$ controls the importance of the regularization term.

In this chapter, we represent the unknown observation, i.e., the spectrum of a pigment, from a painting by $x$. We estimate the sparse representation $\alpha$ corresponding to it by making a dictionary matrix $D$ and solving a convex optimization problem. Given $D$ and an unknown observation $x$, we identify the constituent components of $x$. In making our library or dictionary matrix $D$, all the possible colors are considered and no further information about the number of pure pigments or their spectra is required. We assume that the spectrum of each pixel, i.e., observation, in the painting can be described as a linear combination of the atoms of the dictionary. Since each pixel in the real painting is composed of a small number of pigments compared to the total number of pigments in the composition as a whole, the obtained representation $\alpha$ should be a sparse vector.
4.2.3. Sparse Representation for Pigment Identification

Let us assume we have $P$ spectral samples for each pigment. $N$ sets of spectral samples are measured by capturing hyperspectral images of a library of pigments. We denote the number of pigments as $K$. From the measurements, we create $K$ sub-dictionaries $D_k \in \mathbb{R}^{P \times n_k}, k = 1, \cdots, K$. Each sub-dictionary is formed as:

$$D_k = [d_{k,1}, d_{k,2}, \cdots, d_{k,n_k}]$$ (4.4)

where $d_{k,i}$ is the $i$-th spectrum of the $k$-th color and $n_k$ is the number of the training samples for class $k$. Also, we define sub-coefficient vectors $\alpha_k$ corresponding to the coefficients of sub-dictionary $D_k$ as:

$$\alpha_k = [\alpha_{k,1}, \alpha_{k,2}, \cdots, \alpha_{k,n_k}]$$ (4.5)

where $\alpha_{k,i}$ is the corresponding coefficient of $d_{k,i}$. The full dictionary or library $D$ is composed of all such sub-dictionaries, according to:

$$D = [D_1, D_2, \cdots, D_K] \in \mathbb{R}^{P \times N}$$ (4.6)

The full coefficient vector $\alpha$ is composed of all such sub-coefficient vectors, according to:

$$\alpha = [\alpha_1, \alpha_2, \cdots, \alpha_K] \in \mathbb{R}^{P \times N}$$ (4.7)

where $N = \sum_{k=1}^{K} n_k$ in Eq. (4.6) and (4.7).

Assuming a sample from a class can be estimated as a sparse linear combination of the training samples from that class, we can write
\[ x_k \approx D_k \alpha_k = \alpha_{k,1} d_{k,1} + \cdots + \alpha_{k,n_k} d_{k,n_k} \]  \hspace{1cm} (4.8)

where \( x_k \) is the observation or test sample from class \( k \), \( \alpha_{k,i} \) coefficients are defined as in Eq. (4.5), and \( d_{k,i} \) atoms are defined as in Eq. (4.4). For any test sample, the non-zero components of \( \alpha \) correspond to the concentration of each pigments present in the mixture.

4.2.4. Building the Pigment Dictionary

To form our dictionary \( D \), we use a set of pure commercial pigment exemplars, chosen and prepared as paint in our laboratory based on historic sources and archaeological examples, closely resembling the materials used to make ancient paintings. We captured the hyperspectral response of these pure exemplars and used these spectra to form a reference dictionary allowing us to study pigment identification via sparse regression. In addition, we used the samples from two on-line dictionaries: FORS Library and USGS Digital Spectral Library. To obtain a set of representative samples for each class and make the size of the dictionary reasonable, we apply the k-means clustering algorithm and use only the centroids of each cluster as atoms to populate the dictionary. Our final dictionary consists of \( N = 406 \) atoms and \( K = 23 \) unique pigments.

4.2.5. Pre-processing

Our captured spectra consist of 240 bands with 2nm resolution from 383 to 893nm. We observe that the low wavelength channels (20 bands) in the UV tend to be quite noisy, and are omitted. To reduce the effects of noise in the remaining bands, we average each four consecutive bands together, resulting in \( P = 55 \) channels in the each atom. The last step
before solving the optimization problem is to normalize the atoms in the dictionary and test samples to have Euclidean norm equal to one.

4.2.6. Evaluation Measures

To evaluate the performance of our algorithm, we define a new coefficient vector, $\alpha' \in \mathbb{R}^K$ with elements:

$$\alpha'_k = \sum_{j=1}^{n_k} \alpha_{k,j} \quad \text{for} \quad k \in \{1, \cdots, K\}$$  \hspace{1cm} (4.9)

where $\alpha'_k$ is the $k$th component of vector $\alpha'$, and $\alpha_{k,j}$ is the $j$th component of vector $\alpha_k$. The new coefficient vector $\alpha'$ is simply the sum of the coefficients for the atoms corresponding to each pigment. Unfortunately, in our experiments we do not have access to ground truth information on the concentration of pigments in the spectra we measured. Instead, we use reconstruction error (RMSE) as a metric of evaluation.

$$RMSE = \sqrt{\frac{1}{N} \sum_{i=1}^{N} \| x_i - D \cdot \alpha_i \|_2^2}$$  \hspace{1cm} (4.10)

4.2.7. Experiments

To evaluate the performance of the developed algorithm, we organize two sets of experiments. To compare our results with previous methods for pigment identification, we use SAM, SCM and linear unmixing [67]. In this section, we refer to our algorithm as the sparse unmixing method. In the linear unmixing algorithm, we refer to the atoms of the dictionary as endmembers.

4.2.7.1. Real Pigment Mixtures. In our first experiment, we use seven pure pigments: “gypsum, calcite, indigo, lead white, Egyptian blue, goethite and hematite”. We create 17 mixtures of these pigments by mixing two/three together and painting onto a color
Table 4.1. Average reconstruction error For 17 mixtures (our algorithm vs. linear unmixing)

<table>
<thead>
<tr>
<th>Mixture</th>
<th>Our Algorithm</th>
<th>Linear Unmixing</th>
</tr>
</thead>
<tbody>
<tr>
<td>gypsum+lead white</td>
<td>0.010</td>
<td>0.011</td>
</tr>
<tr>
<td>calcite+goethite</td>
<td>0.045</td>
<td>0.081</td>
</tr>
<tr>
<td>calcite+hematite</td>
<td>0.038</td>
<td>0.096</td>
</tr>
<tr>
<td>calcite+lead white</td>
<td>0.056</td>
<td>0.125</td>
</tr>
<tr>
<td>indigo+lead white</td>
<td>0.053</td>
<td>0.112</td>
</tr>
<tr>
<td>indigo+hematite</td>
<td>0.013</td>
<td>3.226</td>
</tr>
<tr>
<td>indigo+goethite</td>
<td>0.010</td>
<td>0.011</td>
</tr>
<tr>
<td>hematite+lead white+goethite</td>
<td>0.045</td>
<td>0.081</td>
</tr>
<tr>
<td>Egyptian blue+lead white+goethite</td>
<td>0.038</td>
<td>0.096</td>
</tr>
<tr>
<td>Egyptian blue+hematite+goethite</td>
<td>0.056</td>
<td>0.125</td>
</tr>
<tr>
<td>Egyptian blue+hematite+lead white</td>
<td>0.053</td>
<td>0.112</td>
</tr>
<tr>
<td>hematite+goethite</td>
<td>0.013</td>
<td>3.226</td>
</tr>
<tr>
<td>hematite+Egyptian blue</td>
<td>0.010</td>
<td>0.011</td>
</tr>
<tr>
<td>hematite+lead white</td>
<td>0.045</td>
<td>0.081</td>
</tr>
<tr>
<td>Egyptian blue+lead white</td>
<td>0.038</td>
<td>0.096</td>
</tr>
<tr>
<td>goethite+lead white</td>
<td>0.056</td>
<td>0.125</td>
</tr>
<tr>
<td>goethite+Egyptian blue</td>
<td>0.053</td>
<td>0.112</td>
</tr>
</tbody>
</table>

We analyze the results of four algorithms (SAM, SCM, linear and sparse unmixing) on mixture 6 which is composed of two pure pigments: indigo and hematite. The average spectra of seven pure pigments are shown in Figure 4.1 (a). For reference, the average spectrum of
the mixture with that of its pure pigments, (indigo, hematite and mixture) are shown in Figure 4.1 (b). As can be observed, the average spectrum of the mixture is very similar to the average spectrum of the indigo curve shown in the figure, indicating a greater concentration of indigo than hematite. We calculate the spectral angle and spectral correlation similarities of each pixel in the mixture with the endmembers in the dictionary. We also apply the linear and sparse unmixing algorithms to estimate the coefficients of each pixel of the mixture. To compare the results of four algorithms, we take the average of the similarity coefficients of all pixels of the mixture for seven colors. In Figure 4.1 (c), the numbers on the horizontal axis refer to the pigment in our pure pigment list. The vertical axis shows the average similarity coefficient obtained by each algorithm for each color. As it can be observed from Figure 4.1, with both unmixing algorithms, the estimated coefficients for the first two colors (gypsum and calcite) are very small ($< 10^{-4}$). However, linear unmixing estimates goethite (color 6) to be greater than indigo (color 3) and hematite (color 7). As discussed previously, SAM and SCM are not suitable to determine pigment concentration in highly mixed regions. By studying the SAM result in Figure 4.1 (c), it is unclear how to choose the proper threshold to identify the constituent pigments in the mixture. The most similar pigments to the mixture pixels appear to be goethite, hematite and gypsum. Using SCM, the most similar pigments are goethite, gypsum and lead white. Only our spectral unmixing algorithm correctly estimates that the sole pigments in the mixture are hematite and indigo.

4.2.7.2. Real Portrait. In our second experiment, we applied our method to a Roman-Egyptian portrait, excavated from the site of Tebtunis in the Fayum region of Egypt. For this painting, ground truth measurements of pigment composition at a small set of point locations were measured using Raman spectroscopy. In Figure 4.2 (a), the studied portrait is shown. The red arrows indicate the pixels for which ground truth information has indicated the
Figure 4.1. (a) Endmembers spectra, (b) Average spectra of indigo, hematite and the mixture, (c) Comparison of the estimated similarity coefficient of four algorithms. Only our spectral unmixing algorithm correctly estimates that the sole pigments in the mixture are hematite and indigo.

Figure 4.2. (a) Studied Portrait (arrows point to locations where ground truth pigment composition is known), (b) Coefficient map of indigo, (c) Coefficient map of hematite. Our sparse unmixing algorithm correctly estimates pigment composition in the select areas where ground truth information is known.

presence of either hematite or indigo. As can be observed from Figure 4.2 (b), our method correctly estimates the presence of indigo in the lozenge of the crown and clavus regions, which is confirmed by our analysis via Raman. In Figure 4.2 (c), we observe that hematite is used more in the face region, which is also confirmed by our ground truth measurements.
4.3. Innovative data reduction and visualization strategy for hyperspectral imaging datasets using t-SNE approach

The analytical identification and classification of pigments augments our understanding of artistic practice and informs how works of art are conserved. In this context, multi- or hyperspectral reflectance imaging has become a workhorse technique for the characterization of painted materials [10, 52, 89, 97]. This success may be attributed to the merits of the technique: it is non-invasive, relatively inexpensive, and allows for the wide field imaging of an artwork in under a few minutes. Particularly, visible hyperspectral imaging or HSI (400 to 900 nm) has proven to be a powerful method to map the distribution of colorants across a painted surface. However, the contiguous set of 2D images collected through the visible range produces a three-dimensional data cube containing millions of spectra that poses an analytical and computational challenge. Therefore, extracting meaningful information from such large spectral volumes requires an efficient and robust approach to undertake unsupervised and/or supervised reduction of the multivariate data [32]. Data reduction is expected to facilitate visualization and classification of the data as well as to reduce computation time. Thus it requires that the multivariate structure of the original dataset is maintained in the lower-dimensional representation in order for the visualization and the resulting coordinates of the reduced space to have meaning to the user. As described in more depth below (Section 4.3.1), most multivariate data reduction methods in cultural heritage studies rely on either principal component analysis (PCA) or minimum noise fraction (MNF) analyses [32, 42, 43, 49, 110, 111, 118, 128, 134]. As well-established techniques for data reduction, their effectiveness for high-dimensional dataset of non-linear mixtures of pure reflectance spectra present several limitations. First, both methods assume that the
underlying data manifold is linear [109]; an erroneous assumption since it is well established through the Kubelka-Munk theory of reflectance that pigment mixtures absorb and scatter light non-linearly [65, 88, 97]. Moreover, for these techniques, dimensionality reduction is obtained by discarding the components with the lowest variance which does not always accurately model chemical variation of the high-dimensional dataset [53, 103]. Finally, the visualization and segmentation of high-dimensional data is compromised when examining only several selected 3D projections, as is a common practice. To overcome these limitations, in this section we develop a novel technique for data reduction called t-distributed stochastic neighbor embedding (t-SNE) [103]. To highlight the advantages of this innovative approach for cultural heritage, the results obtained on mock-up paint samples are compared to two established data reduction and segmentation methods, namely i) PCA combined with K-means clustering, ii) and MNF combined with Pure Pixel Index endmember extraction. The potential of the developed approach is further examined for the study of a historical object: an illuminated page from a French medieval Book of Hours. These examples demonstrate the potential of t-SNE to classify, and map pure and mixed pigments in artistic material.

4.3.1. Previous Work

Within the field of cultural heritage, data reduction and segmentation is often undertaken using statistical techniques borrowed from remote sensing applications [97]. For instance, Baronti et al. [134] and more recently Mounier et al. [111] applied principal component analysis (PCA) for detecting areas of paintings characterized by similar chemical composition or/and physical properties. The eigenvectors and eigenimages of the first principal components are used to identify and map spectral features characteristic of specific pure pigment or pigment mixture. In the context of illuminated manuscript and painting studies, two additional
multivariate analysis protocols have been recently proposed [42, 43, 118]. The first approach uses an algorithm called the ‘hourglass paradigm’ implemented in the ENVI software [43].

This approach can be broken into two steps. First, the reduction of dimensionality and denoising of the data is performed using MNF, where the data cube is transformed into a contiguous set of images with regularly increasing noise levels. In general, only the few first MNF images are kept for further data processing (≈ 10 of them). Second, the reduced data are projected onto several random unit vectors and the extreme pixels in each projection are tallied to record a per-pixel extremity-score which directly corresponds to pixel purity and is thus called a Pure Pixel Index or PPI [26, 77]. From the pixels presenting the highest purity, endmember spectra are extracted by manual clustering. These endmembers are dominated by a single chemical component based on the hypothesis that pixel spectra from pure components present more significant differences compared to mixed component spectra. The match between the reflectance spectra of the dataset and the mutually independent endmembers is finally estimated using the spectral angle-mapping (SAM) algorithm [50, 128].

This method computes an angle between the endmember considered as reference and each pixel spectrum, the smaller angles represent closer matches to the reference endmember spectrum. The number of pixels displayed as belonging to a specific endmember class is a function of a threshold applied during this classification step. A second approach [32] uses an inverse MNF analysis that aims to first denoise and then interrogate the dataset by PCA combined with i) manual clustering of the data in the PC space, and ii) Iterative Key Set Factor Analysis (IKSFA). IKSFA is a method proposed by Malinowski [106] which seeks to find the minimum number of spectra required to reproduce the entire dataset through the characterization of the most orthogonal spectra that typify the original data matrix. In IKSFA, the goal is to find a set of rows (columns) from the data which are most orthogonal to each other. In HSI, this
method translates to finding the spectra of pure elements. As with pixel-purity, the IKSFA algorithm assumes that the purest spectra are more dissimilar than the corresponding mixture spectra on a per pixel basis and separates the spectra of pure elements from the spectra of mixtures by factor analysis. Comparable to the hourglass paradigm approach, the key spectra extracted using IKSFA are mapped using SAM. Using these different approaches, classification and mapping are driven by the data themselves. However, pigment spectral libraries are used during the last step of data evaluation to correlate the extracted endmember spectra with pure pigment spectral signatures. For these different approaches, pigment identification is solely based on visual comparison of the absorption edge positions (in wavelengths) and spectral shape. A third common approach to interrogating hyperspectral datasets of painted surfaces uses databases built a-priori from materials expected to be within the painting. This approach relies on spectral libraries of pigments and binder computed from Kubelka-Munk (KM) theory \[21, 22, 56, 65, 88, 150\] that models the nonlinear effects of spectral mixing. When the nonlinear library (i.e. nonlinear combinations of pigments reflectance spectra) is compared to the data, the best match is found using a Least Squares Linear Combination (LSLC) approach \[44, 65\]. This is a straightforward approach for pigment identification and mapping, but is limited by the extensive computation time and memory allocation necessary for imaging an entire painting. One study tackled this limitation by segmenting the 2D image space based on chromatic information; for each segment a limited number of representative spectra were then selected to perform pigment mapping pixel-wise. The spectra from the spectral library, in this case, that presented the closest linear fit are deemed the pigment combination in a given pixel \[150\].
4.3.2. t-Distributed Stochastic Neighbor Embedding (t-SNE) approach

t-Distributed Stochastic Neighbor Embedding (t-SNE) is an innovative technique for multidimensional scaling [103] inspired by earlier work on Stochastic Neighbor Embedding (SNE) [39, 68], that is particularly well suited to data reduction and visualization of high-dimensional datasets. SNE first converts the high-dimensional distances between datapoints into conditional probabilities that represent similarities. The similarity of datapoint to datapoint is expressed by the conditional probability \( p_{ji} \), that is, the probability that \( x_i \) would pick \( x_j \) as its neighbor if neighbors were picked proportionally to their probability density under a Gaussian centered at \( x_i \). It is given by

\[
p_{ji} = \frac{\exp\left(\frac{\|x_i - x_j\|^2}{2\sigma_i^2}\right)}{\sum_{k \neq i} \exp\left(\frac{\|x_i - x_k\|^2}{2\sigma_i^2}\right)}
\]  (4.11)

where \( \sigma_i \) is the variance of the Gaussian that is centered on datapoint \( x_i \). The probability \( p_{ij} \) is proportional to the similarity of two points \( x_i \) and \( x_j \) and is estimated by \( p_{ij} = \frac{p_{ji} + p_{ij}}{2N} \) where \( N \) denotes the total number of points. t-SNE projects a sample in the high dimensional feature space (original dataset) \( \mathcal{X} = \{x_1, \cdots, x_N\} \) to a low-dimensional one \( \mathcal{Y} = \{y_1, \cdots, y_N\} \). For the low-dimensional counterparts \( y_j \) and \( y_i \) of the high-dimensional datapoints \( x_j \) and \( x_i \), the same expression for the computation of the conditional probability is utilized as in Eq. 4.11, resulting in \( q_{ji} \).

If the points \( y_j \) and \( y_i \) correctly model the similarity between the high-dimensional datapoints \( x_j \) and \( x_i \), the conditional probabilities \( p_{ji} \) and \( q_{ji} \) will be equal. Motivated by this, SNE aims to find a low-dimensional representation that minimizes the mismatch between \( p_{ji} \) and \( q_{ji} \). The Kullback-Leibler (KL) divergence is the natural measure of the similarity of two distributions. SNE minimizes the sum of the KL divergences over all datapoints, using
a gradient descent approach. Since the KL divergence is not symmetric, different types of errors are introduced. As an alternative to minimizing the sum of the KL divergence between the conditional distributions $p_{ji}$ and $q_{ji}$, a single KL divergence between a joint probability distribution $P$, in the high-dimensional space, and a joint probability distribution $Q$, in the low-dimensional space is minimized. The cost function in this case is given by

$$C = KL(P||Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}}$$ (4.12)

where $p_{ij}$ is given by

$$p_{ij} = \frac{\exp\left(\frac{||x_i-x_j||^2}{2\sigma_i^2}\right)}{\sum_{k \neq i} \exp\left(\frac{||x_i-x_k||^2}{2\sigma_i^2}\right)}$$ (4.13)

and $q_{ij}$ is defined also using Eq. 4.13. Minimization of the cost in Eq. 4.12 is now referred to as symmetric SNE. It is mentioned here that alternative definitions of $p_{ij}$ exist.

One of the problems present when mapping a high-dimensional to a low-dimensional space is the so called “crowding problem”. It is due to the fact that the volume of a sphere centered on a datapoint $i$ scales as $r^m$, where $r$ is the radius and $m$ is the dimensionality of the sphere. So if the datapoints are approximately uniformly distributed in the region around $i$ on the high-dimensional manifold, and we try to model the distances from $i$ to the other datapoints in the two-dimensional map, we have the “crowding problem”, that is, the area of the two-dimensional map that is available to accommodate moderately distant datapoints will not be nearly large enough compared with the area available to accommodate nearby datapoints. Therefore, in order to model the small distances accurately in the map, most of the points that are at a moderate distance from datapoint $i$ will have to be placed much too far away in the two-dimensional map.
In order to address the crowding problem, while a Gaussian distribution is utilized in the high-dimensional space to convert distances into probabilities as shown in Eq. 4.11, the Student-t distribution, which is a heavy-tailed distribution is utilized for the same purpose in the two-dimensional space. That is the joint probabilities are now defined as

\[ q_{ij} = \frac{(1 + \|y_i - y_j\|^2)^{-1}}{\sum_{k \neq i} (1 + \|y_i - y_j\|^2)^{-1}} \]  

(4.14)

The Student-t distribution (with a single degree of freedom) shown in Eq. 4.14 allows a moderate distance in the high-dimensional space to be faithfully modeled by a much larger distance in the map.

Thus by preserving the local data structure in the low-dimensional embedding, this approach represents a new opportunity for data reduction and visualization, through segmentation, of large non-linear HSI datasets.

### 4.3.3. Experimental Datasets

#### 4.3.3.1. Samples.
To test the feasibility of quantitative characterization of pigment mixtures based upon HSI, reference paints were analyzed. A set of mock-up paintings were prepared composed of 12 pure colors and 16 mixtures. There are fifteen mixtures of two colors and one of three colors. Among the two-color mixtures, eleven are called tints and are a mixture of pure pigment and white color (namely lead white \((2PbCO_3.Pb(OH)_2)\)), the four others being a mixture of two colored pigments. To build the mock-ups, the pigment weight ratio percentages were i) 50/50 for the two pigments mixture, and ii) 33/33/33 for three the pigments mixture. The pigments were then mixed with a gum arabic to bind the paint layers to a commercially primed canvas. The layers were applied thick enough to prevent
the priming layer from interfering with the measured reflectance spectra. Suffrages from a Book of Hours (Ms. 6.T.6) manuscript part of the Isabella Garner Museum collection were also analyzed. The decorative program of this book comprises a range of schools of French illumination, apparently executed in two campaigns in the mid-1400s and around 1500. At least three different “hands” or styles have been identified among the pages of the manuscript. Additional modern restoration by the 19th-century English illuminator and copyist of manuscripts, Caleb William Wing, were also visually identified by scholars through the manuscript. To redefine artistic hands and practice among one or different workshops and modern add-in, the painting technique and the materials used for the manufacturing have been analyzed using a combination of in-situ techniques namely: HSI and X-Ray Fluorescence (XRF). In order to focus this study on the efficiency of the proposed data treatment approach, the results of a single page of the full book are reported (page 34v).

4.3.4. Experimental

Hyperspectral images were recorded using a Resonon Pika II Pushbroom system in the 400 – 900 nm range with spectral resolution of 2nm. The system was connected to a stage allowing the scanning of the entire width of the samples, with a pixel size of $125 \times 125 \mu m^2$ in the case of the historic object and $375 \times 375 \mu m^2$ for the mock-up paintings. During acquisition, the object was illuminated using two broad-spectrum tungsten halogen lamps placed at $45^\circ$ of the object normal. A Lambertian reference reflector (Epson UltraSmooth Fine Art Paper, 13” $\times$ 19”, A3+, 325g/m2) was used as a calibration target to convert the image cubes to diffuse reflectance. Hyperspectral acquisition was performed using the SpectrononPro software. The macro XGLab’s ELIO XRF imaging spectrometer system (MA-XRF) was used in combination with HSI for characterizing the pigments palette of the
illuminated manuscript page. The instrument is equipped with a transmission Rh anode X-Ray tube, the polychromatic beam presenting an incoming angle of 63.5° prior to the sample plane. A collimator allowing a 1 mm diameter focused spot size at the surface of the object was used to acquire XRF maps at the surface of the book. The instrument was operated at 50$kV$ and 40$\mu A$. The 2D surface rastering was executed with acquisition times of 1s per point and with $1 \times 1 mm^2$ step size.

### 4.3.5. Data treatments

Principal component analyses (PCA) and subsequent k-means clustering were performed using the TXM-Wizard software [149]. Pixels with similar reflectance spectra were pooled in Principal Component space, effectively segmenting the image based on the variance in the recorded reflectance features into a pre-defined number of regions (k areas) consisting of pixels with a similar spectral signature. The number of clusters was defined as the number of pure pigments or mixtures present into the data to analyze, i.e. each paint composition being accounted as a single cluster. The first eight Principal Components represented more than 98% of the variance of the total stacks, consequently the clustering was run on these 8 components only. Matlab hyperMnf function [62] was used to perform the MNF transform on the data. Subsequently, the pixel purity index (PPI) algorithm hyperPpi was used to determine endmembers within the dataset (the number of “skewer” vectors to project data was fixed to $10^6$ iterations). For each datapoint, we counted the number of times it was being picked as the extreme point in random projections. This count number for each point was used as the PPI index, the higher the index is, the higher is the probability for the point to be a pure pixel. The Matlab implementation of the standard t-SNE method was used. The function performed symmetric t-SNE on the $N \times N$ pairwise Euclidean distance matrix to
construct an embedding with two dimensions. The perplexity of the Gaussian kernel that is employed can be specified through perplexity, for this study it was fixed at 50. For both mock-up and historical sample datasets, the pixel number was downsized by a factor of 4; instead of binning that would lead to pixels signal averaging, 1 pixel over 4 was kept for data processing. This downsampling was necessary to allow reasonable computing time (typically shorter than 10min for the full dataset) with standard computer resources.

4.3.6. Experiments and Results

4.3.6.1. Mock-up pigment classification using Principal Component Analysis combined with K mean clustering. To segment and classify the different pigment signatures of the mock-up samples, the PCA results have been combined with k-means clustering. The efficiency of this approach for data reduction and classification has been determined for the full dataset composed of pure pigments, tints (lead white) and colored pigment mixtures.

When the full dataset is used as an input for the reduction and segmentation processes, several limitations are observed. The pure Prussian Blue ($Fe_4(Fe[CN]_6)_3$, ivory black (composed of about 10% carbon, C, and 83% calcium hydroxyapatite, $Ca_5(PO_4)_3(OH)$, along with smaller amounts of magnesium phosphate, $Mg(H_2PO_4)_2$, and calcium carbonate, $CaCO_3$) and ivory black mixed with lead white, all highly absorbing pigments within the visible range, do not cluster separately (Figure 4.3 (b), (c)). The low reflectance index of these two pigments together with the absence of specific spectral features over the range studied prevent their differentiation by the proposed method (Figure 4.4 (a)). Whereas the add-in of white pigment allows for a slight increase of the reflectance value of the ivory black tint (Figure 4.4 (a)), these variations remain too low in comparison with the total dataset variance to allow for a specific differentiation of their fingerprints. Similarly, cobalt blue
Figure 4.3. a) Visible picture of the mock-up samples analyzed, showing the color aspect of pure pigment and pigment mixture layers - pigment name followed by (*) corresponds to tint, i.e. mixture of coloring pigment with lead white; b) Datapoints represented into PC01, PC02 and PC03 space, pixels are colored based on their cluster assignment; c) the same color scale is used to represent the data clustering into real sample space (the cluster number is provided for each paint layer to ease data interpretation).

$(CoO.Al_2O_3)$ and cobalt blue mixed with lead white were not clustered separately (Figure 4.3 (b), (c)); the add-in of lead white being barely observable as the reflectance intensity of the paint tint is almost unchanged compared to the pure pigment (Figure 4.4 (b)). More interestingly, the paint tint of ultramarine $(Na_8[Al_6Si_6O_{24}]S_n$, mineral lazurite) with lead white, is split in two different clusters (Figure 4.3 (b), (c)). These two clusters being explained by a heterogeneous thickness of paint application through the area studied; for some part of the paint out the colored layer is thin enough to be partially transparent and thus presents a reflectance spectra resulting of the nonlinear mixture of the white preparation layer and the blue pigment. For this example, thickness effects present variations that were more easily clustered than changes in pigment composition. Based on these results, the approach
Figure 4.4. a) Comparison of the mean spectra collected in the paint out area corresponding to Prussian Blue and ivory black pure pigments, and tints; b) Comparison of the mean spectra collected in the paint out area corresponding to cobalt blue pure pigment and tint.

represents a fast and easy solution for the segmentation of different pure pigments and pigment mixtures, respectively, with distinct spectral signature. However, small spectral variation in shape and intensity are difficult to visualize and extract when looking at the linearly reduced dataset. As previously outlined by focusing on describing the largest variance in the dataset, low-dimensional variations are harder to cluster with any segmentation methods – the approach being more sensitive to low Signal To Noise ratio (SNR) area, and paint layer heterogeneities containing pigments with strong absorption edges which will result in more variance in the spectral domain. Moreover, the segmentation step of the approach requires a good a priori knowledge of the number of pure pigments and mixtures present, as an expected number of clusters is required prior to the clustering. Thus, the data reduction does not provide a straightforward data representation to define the different spectral signatures present in the dataset.
4.3.6.2. Minimum Noise Function combined with Pure Pixel Index algorithm.

Endmember extraction using MNF combined with Pure Pixel Index (PPI) algorithm is an established approach to solve the unmixing problem of hyperspectral datasets, where MNF is used to perform dimensionality reduction to ease computational complexity as well as compact information in the transformed components to perform the endmember extraction. Figure 4.5 (a) presents the dataset in the two first component spaces identified by PPI. Figure 4.5 (b) presents for each pure pigment box after PPI calculation i) the index of the first appearance in the purity score of one single pixel of the pure pigment area, ii) the maximum voting that refers to the purity score of the point selected as the most extreme, which corresponds to the first occurrence pixel, and iii) the sum of the purity score of all pixels.

Similarly to PCA results, by focusing on high variance between datapoints, smallest variance between spectra is statistically less represented in the unmixed dataset. As an example, none of the pure Prussian blue, ivory black, azurite \((2CuCO_3.Cu(OH)_2)\) and malachite \((CuCO_3.Cu(OH)_2)\) pigment spectra are identified as pure pixels after PPI calculation. On the contrary, lead white, lead tin yellow \((Pb_2SnO_4, \text{type 1})\), cobalt blue, smalt (a potash glass with color given by cobalt ions), vermilion \((HgS)\), and verdigris \((Cu(C_2H_3O_2)_2.2Cu(OH)_2)\) which appear as maxima clusters in the first and second component values of the PPI projection (Figure 4.5 (a)), are identified as pure pixel between the first and 2222th projections of the PPI protocol (Figure 4.5 (b)), value of index of first appearance in PPI). Based on these results, defining the cut-off threshold value for the scores produced by the PPI to extract candidate pixel vectors for final selection of endmembers is not straightforward. It also calls into question the efficiency of a manual selection of the final set of endmembers by using the purity map index as a visualization tool. Indeed, some pigments seem to be more efficiently identified as single component than others. Thus, using the sum of voting
Figure 4.5. a) Dataset represented in the two first space components of the PPI calculation, b) Table presenting the results of the PPI calculation for each pure pigment paint out: i) index value of the first appearance as a pure component of a single pixel, ii) purity score of the single pixel that first appeared as a pure component, iii) sum purity score of all pixels.

and maximum voting values presented in Figure 4.5 (b), the determination of pure pigment as endmember seems compromised as only 8 pure pigments over 12 present are identified as pure pixels. Moreover, when comparing the sum of voting per pigment boxes, the tints of vermilion, lead tin, orpiment, smalt, and verdigris present a higher purity score value than their pure pigments. Similarly, mixtures of lead tin with either verdigris or malachite present a higher purity score values than their respective pure verdigris or malachite pigments. With this example, the presence of highly reflective pigment seems to strongly affect the result of the unmixing, as instead of identifying pure pigment area, the PPI identify higher reflectance values as more extreme pigment signatures.

Whereas this approach is presented as an efficient approach for spectral unmixing, this example demonstrates that it shares similar limitations with the previous approach when used for data reduction and segmentation. In particular to perform endmember identification from dataset three main limitations can be emphasized, the loss of non-liner relationship in
the data reduced space, the loss of low-variance variation of the dataset, and the difficulty to identify rare events that can reflect a few pixels of the entire dataset [34]. These limitations ultimately determine the quality and efficiency of the approach to determine endmember spectra of pure pigment as observed for this dataset.

4.3.6.3. t-SNE. The function performs symmetric t-SNE on the $N \times N$ pairwise Euclidean distance matrix $D$ to construct an embedding with 2 dimensions. To ease visualization, in the scatterplot each pixel of the dataset is represented in the RGB color domain using three channels of the HSI wavelength range: red is assigned to channel 118 (637.6nm), green to channel 75 (547.8nm) and blue to channel 32 (458.1nm). The results obtained using the t-SNE approach are presented in Figure 4.6 (a). For comparison, a manual clustering of the dataset in the reduced data space is provided in Figure 4.6 (b) together with its result in the sample map space.

For studying mock-up systems, the technique performs an efficient dimension reduction of the dataset while keeping the local data structure (Figure 4.6 (a)). Pure pigments, tint of colored pigments mixed with lead white, and colored pigment mixtures appear as distinct pixel groups in the reduced space (Figure 4.6 (b)). Thus, the 2-dimensional representation of the dataset provides a straightforward visual segmentation and classification of the pigment and pigment mixture signatures present within the data cube. This data reduction not only preserves similarities within the dataset in a 2D reduced dimension but acts as an efficient method to spatially cluster spectra with similar signatures (Figure 4.6 (b)). In comparison to PCA and MNF, t-SNE efficiently differentiates the pure pigments from their corresponding tints. As an example, by focusing on small pairwise distances, this approach successfully distinguishes spectra of Prussian blue and ivory black. Similarly, pure cobalt blue is efficiently separated from its mixture with lead white (Figure 4.6 (a)). Whereas, the distance of one
Figure 4.6. Visualization of 60x105 pixels map from the mock-up HSI dataset produced by symmetric t-SNE approach a) using RGB pixel values for labelling; and b) combined with user-guided manual clustering - the same color scale is used to represent the result of data clustering into sample map space (the cluster number is provided for each paint layer).

Cluster to another cannot be used as quantitative information on the similarities between pixel groups (as t-SNE provides a non-linear representation of those similarities), the presence of pixels centered between the cluster of pure cobalt blue pigment and cobalt blue mixed
with lead white highlight the possibility that some pixels of the mixture contain pure pigment in higher concentration than in the rest of the mixture. Interestingly this effect is observed for a pure pigment that was not differentiated from its tint with the two other approaches.

t-SNE provides efficient data reduction with a clear visualization of the similarities between the spectral features of both pure pigment and pigment mixtures. As the clustering of the 2D representation of dataset is straightforward, this method offers the possibility of performing the data segmentation using user-guided manual clustering, rather than more complicated and computationally slow statistical algorithms. This drastically simplifies the approach compared to classical data analyses protocol, which requires segmentation after data reduction to extract meaningful information on the dataset.

4.3.6.4. Results of t-SNE approach for historical samples. t-SNE algorithm has been used to process hyperspectral dataset acquired on an illuminated manuscript page (detailed above) to assess the capabilities of this innovative approach in the case of historical painted materials. Similarly to the work performed on the mock-up samples, symmetric t-SNE on the $N \times N$ pairwise Euclidean distance matrix D was performed to construct a data embedding in two dimensions. The original dataset contains $100 \times 149$ pixels encompassing the 240 wavelengths of the HSI spectra. For this dataset, the processing time represented 460 seconds in total. For each pixel its RGB color was assigned in the t-SNE 2D representation using three specific wavelengths: red to channel 118 (637.6nm), green to channel 75 (547.8nm) and blue to channel 32 (458.1nm). The results obtained using the t-SNE approach are presented in Figure 4.7 (a). Here no prior information on the pigments was provided apart from the elemental mapping using MA-XRF method. As presented for the mock-ups example, the 2D dataset was further segmented using the inherent 2D structure of the reduced dataset, Figure 4.7 (b). A user-guided manual clustering was used to select and assign a characteristic
Figure 4.7. a) Visible picture of the area analyzed by t-SNE; b) 2D representation of the algorithm result; c) assignment of the red and pink clusters in the t-SNE component space (top) and transposed into data space (bottom); d) centroids spectra of data cluster 1, 2, 3 and 4 compared to reflectance spectra of red lead and vermilion pigments, and brazil wood and red madder lakes; e) assignment of the blue clusters in the t-SNE component space (top) and transposed into data space (bottom); f) centroids spectra of data cluster 6 and 7 compared to reflectance spectra of azurite and Fra Angelico pigments; g) assignment of the green clusters in the t-SNE component space (top) and transposed into data space (bottom); h) centroids spectra of data cluster 11 and 10 compared to reflectance spectra of verdigris, malachite and mixture of verdigris and saffron.

Color to a set of pixels presenting the highest similarities in the 2D space. This information was returned into the original image space where the different clusters appeared with different colors. In addition, the corresponding centroid spectrum was extracted. For more clarity, the cluster identification is provided by color hue, Figure 4.7 (c-e).
Assignment of the red and pink clusters Cluster 1 is present in both the mantle of the left character and red flowers present on both side of the main figure (Figure 4.7 (c)). The spectra for the corresponding area reveal shapes similar to semiconductors pigment, i.e. a sigmoid with a steeper rise around the inflection point \( [8] \). For cluster 1, the inflection point being around 590 – 605nm indicates the presence of vermillion, confirmed by the identification of mercury and sulfur by XRF analyses \([42] \). Cluster 4, present in the localized area of the white and red flower surrounded by gilding and blue decoration, present a similar semi-conductor fingerprint (Figure 4.7 (c)). However, with an inflection point around 565nm, the presence of red lead \((Pb_3O_4,\) mineral form minium) is proposed in this single flower \([8] \). This has been confirmed by the presence of high content of lead together with the absence of additional elements that could refer to other semi-conductor red pigments. Cluster 2 is present in all vegetable dark red decoration observed on the side of the main figure (Figure 4.7 (c)). The first derivative of its representative spectra has a broad asymmetric band with a maximum centered at 590nm. For both spectra a rise in reflectance in the red part of the spectra is also observed. XRF spectra from these sites show mostly the presence of calcium, which seems consistent with the presence of a red dye with chalk \((CaCO_3)\) as possible substrate. Moreover the presence of a weak absorption structured in two bands at 510 – 515 and 540 – 545nm, might suggest the presence of a red madder dye \([8,42]\). Similarly, Cluster 3 refers to vegetal pink decoration, with an XRF spectra dominated by Ca signal and a broad asymmetric band with a maximum centered at 590nm with a constant absorption intensity in the red part (Figure 4.7 (c)). A single absorption band centered at 560nm is observed for this cluster, pointing toward the use a pink dye namely brazilwood applied on a chalk substrate \([8,42]\).

- Assignment of the blue clusters

Two clusters are identified among the blue areas: clusters 6, mostly present in the robe
of the characters, and 7, mostly present in the blue vegetable decoration identified in
the vicinity of the central scene (pink and light pink respectively) (Figure 4.7 (d)).
Cluster 6 presents a good match with the mineral ultramarine ($\text{Na}_8[\text{Al}_6\text{Si}_6\text{O}_{24}]\text{S}_n$, mineral lazurite), with a reflectance spectrum presenting a maximum of absorption around 600nm and a transition to high reflectance around 700nm. In the case of cluster 7, a decrease in reflectance between 700 and 900nm is observed, that can be attributed to the presence of azurite pigment together with ultramarine. XRF spectra confirms the significant amount of Cu, consistent with the presence of azurite, together with dominant Al, Si and K signals, consistent with the presence of ultramarine. These results can be interpreted as a combined use of both pigments to paint vegetable blue area of the illumination. Together with lead white, Pb being identified by XRF, different shades of blue can be achieved as observed in the RGB image of the illumination. In previous publications the application of a thin layer of ultramarine on top of azurite is proposed to create deeper blue hues, instead of a mixture of both pigments, since ultramarine glazing over azurite was a common technique during this time period [13, 42]. Two other clusters refer to blue area: clusters 9, identified in blue brocades in the central figure, and 8, identified at the edge of the blue vegetable decorations (red light and light blue respectively - Figure 4.7 (d)). In the case of cluster 9, the presence of ultramarine is proposed based on the same spectral features described for cluster 6, however its presence as a single cluster is explained by the fact that the blue paint is applied on top of a gold leaf. The blue paint being relatively thin, the reflectance from the gold leaf underneath modified the signal of the pure pigment, mostly by increasing the reflectance intensity of these pixels between 600 – 900nm. Similarly, cluster 8 refers
to blue paint that diffuses into the parchment, here again the high and constant reflectance of the prepared parchment mostly flatten the reflectance spectra of the blue pixels leading to a distinct spectral fingerprint.

- Assignment of the green clusters

The green garments are clustered in two clusters: cluster 10 and 11 (Figure 4.7 (e)). Cluster 11 is limited to the leaves painted within the edge section. Its characteristic spectrum presents a peak reflectance around 535 – 540nm and a slow rise from 800nm. Similarly cluster 10 presents a peak reflectance at 535nm, however the peak is broader than cluster 10, accompanied by a maximum of absorption shifted toward higher wavelengths (760nm) and the absence of reflectance rise after 800nm. These spectral features point out the use of Cu-containing green such as verdigris, copper-resinate ($C_{19}H_{29}COOH$), or malachite, confirmed by the presence of Cu identified as the main element entering in the composition of the green paints by XRF technique [42]. The difference in peak broadening in the case of cluster 10 can be attributed to many different factors, however a mixture of verdigris with a yellow dye, e.g. type saffron, to achieve a different type of green can be proposed. Within cluster 11 both dark and light greens are present - they both present similar reflectance peaks however the overall intensity of the spectra in the darker area is lower suggesting the add-in of a dark pigment in that area (e.g. carbon black).

Two additional clusters are attributed to the gilded area (cluster 13), and to the parchment without colored paint layers (clusters 12).

In conclusion, the combination of t-SNE 2D representation with manual clustering allows for the segmentation of the datacube into pure pigments and pigment mixture clusters that confirm the first results obtained by XRF mapping. It represented
a relatively fast and easy multivariate approach to extract and differentiate the different spectral fingerprints present into the dataset—the use of centroids spectra for each cluster together with an adapted database allow identifying the main pigment and dyes palette used by the artist to depict the illumination.


In HSI, visible reflected light from painted surfaces are spatially recorded resulting in a spectrum at every pixel location in an image. The entire dataset often contains several million spectra making the processing of these data computationally challenging. On the other hand, such large datasets also offer new opportunities to innovate methods of multivariate analysis utilizing the latest algorithms developed for processing “big data”. Previous works in this area have employed a broad range of spectral information which relate the physical mineralogy of the pigment to its unique molecular spectral response. Since individual pigments have a characteristic reflectance, these data may be compared to dictionaries of known spectra to make identifications [132], at least in the trivial case where only one type of pigment is present in a given pixel.

In this section, we study the automatic pigment unmixing problem and develop an innovative approach using deep learning and Kubelka-Munk theory to first identify the constituent pigments and then find their abundance values for any given spectrum. Here, we describe a novel approach to solve the pixel-wise pigment unmixing problem to produce data-rich and semi-quantitative maps of pigment distributions. First, we use a supervised classification scheme in which a deep neural network is trained to qualitatively identify the pigments present in a given spectrum. Having found the pigments, the concentration
values of the individual components are approximated using both an indirect and direct approach. The indirect approach requires the construction of a dictionary of non-linear samples with different concentrations of the basic pigments. The KM coefficient values of the given spectrum are estimated by finding the closest match in the dictionary. In the direct approach, combinations of pure compound spectra are fitted on a pixel by pixel basis using a gradient descent implementation of Kubelka-Munk to calculate the coefficient values.

The rest of this section is organized as follows. First, we describe the experimental dataset which has also been used in Section 4.4.1 and the preprocessing step. In Section 4.4.3, the problem is defined. Then, two steps of the algorithm: pigment identification (Section 4.4.4) and nonlinear unmixing (Section 4.4.6) are described.

### 4.4.1. Dataset

We have prepared a set of mock-up paintings composed of 12 pure colors and 16 mixtures. The average spectra of 12 pure pigments in the reflectance space are shown in Figure 4.8. As can be observed, each color has its unique spectrum shape. Lead white has almost a flat signal in reflectance space. There are 15 mixtures of two colors. 11 of these mixtures are tint (mixtures of pure pigment and white color) and the other 4 mixtures are composed of two non-white colors. There is one mixture of three non-white colors. To build the mixtures, the same coefficients of colors have been used ($\frac{1}{2}$ for mixtures of two colors and $\frac{1}{3}$ for mixtures of three colors), theoretically. However, as this experiment has been done in a real setting, it is impossible to control such an assumption that in every position at the mock-up square, the same concentration of specific color is used. Unfortunately, this is the only ground truth information that we can have.
4.4.2. Pre-processing

Our captured spectra consist of 240 bands with 2nm resolution from 383 to 893 nm. We observe that the first low wavelength channels (10 bands) and the last wavelength channels (10 bands) tend to be quite noisy, and are omitted. Also, all the pixel values are normalized by the reflectance scale factor of the hyperspectral camera. We build our hyperspectral image by concatenating 12 pure pigments and 16 mixtures in a rectangle. From each pure/mixture box, we select a $60 \times 60$ square to make all mixtures have the same number of pixels (3600 pixels). The pure/mixture compositions of different classes are demonstrated in Table 4.2 and the six exemplar mixture boxes in our mock-up painting are shown in Figure 4.9. The whole mock-up image is shown in Figure 4.3 (a).

4.4.3. Problem and Model Definitions

Assume there is a hyperspectral painting $I \in \mathbb{R}^{N \times P}$ with $N$ number of pixels and $P$ wavelengths. Besides the hyperspectral test data, there is a dictionary $D \in \mathbb{R}^{M \times P}$ composing of
<table>
<thead>
<tr>
<th>Class</th>
<th>Multi-label vectors</th>
<th>Composition</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>[1,0,0,0,0,0,0,0,0,0,0,0]</td>
<td>Prussian blue</td>
</tr>
<tr>
<td>2</td>
<td>[0,1,0,0,0,0,0,0,0,0,0,0]</td>
<td>Cobalt blue</td>
</tr>
<tr>
<td>3</td>
<td>[0,0,1,0,0,0,0,0,0,0,0,0]</td>
<td>Orpiment</td>
</tr>
<tr>
<td>4</td>
<td>[0,0,0,1,0,0,0,0,0,0,0,0]</td>
<td>Vermilion</td>
</tr>
<tr>
<td>5</td>
<td>[0,0,0,0,1,0,0,0,0,0,0,0]</td>
<td>Ivory Black</td>
</tr>
<tr>
<td>6</td>
<td>[0,0,0,0,0,1,0,0,0,0,0,0]</td>
<td>Lead Tin Yellow</td>
</tr>
<tr>
<td>7</td>
<td>[0,0,0,0,0,0,1,0,0,0,0,0]</td>
<td>Verdigris</td>
</tr>
<tr>
<td>8</td>
<td>[0,0,0,0,0,0,0,1,0,0,0,0]</td>
<td>Azurite</td>
</tr>
<tr>
<td>9</td>
<td>[0,0,0,0,0,0,0,0,1,0,0,0]</td>
<td>Smalt</td>
</tr>
<tr>
<td>10</td>
<td>[0,0,0,0,0,0,0,0,0,1,0,0]</td>
<td>Ultramarine</td>
</tr>
<tr>
<td>11</td>
<td>[0,0,0,0,0,0,0,0,0,0,1,0]</td>
<td>Malachite</td>
</tr>
<tr>
<td>12</td>
<td>[0,0,0,0,0,0,0,0,0,0,0,1]</td>
<td>Lead White</td>
</tr>
<tr>
<td>13</td>
<td>[1,0,0,0,0,0,0,0,0,0,0,0]</td>
<td>Persian Blue+W</td>
</tr>
<tr>
<td>14</td>
<td>[0,1,0,0,0,0,0,0,0,0,0,0]</td>
<td>Cobalt Blue+W</td>
</tr>
<tr>
<td>15</td>
<td>[0,0,1,0,0,0,0,0,0,0,0,0]</td>
<td>Orpiment+W</td>
</tr>
<tr>
<td>16</td>
<td>[0,0,0,1,0,0,0,0,0,0,0,0]</td>
<td>Vermilion+W</td>
</tr>
<tr>
<td>17</td>
<td>[0,0,0,0,1,0,0,0,0,0,0,0]</td>
<td>Ivory Black+W</td>
</tr>
<tr>
<td>18</td>
<td>[0,0,0,0,1,0,0,0,0,0,0,0]</td>
<td>Lead Tin Yellow+W</td>
</tr>
<tr>
<td>19</td>
<td>[0,0,0,0,0,1,0,0,0,0,0,0]</td>
<td>Verdigris+W</td>
</tr>
<tr>
<td>20</td>
<td>[0,0,0,0,0,0,1,0,0,0,0,0]</td>
<td>Azurite+W</td>
</tr>
<tr>
<td>21</td>
<td>[0,0,0,0,0,0,0,1,0,0,0,0]</td>
<td>Smalt+W</td>
</tr>
<tr>
<td>22</td>
<td>[0,0,0,0,0,0,0,0,1,0,0,0]</td>
<td>Ultramarine+W</td>
</tr>
<tr>
<td>23</td>
<td>[0,0,0,0,0,0,0,0,0,1,0,0]</td>
<td>Malachite+W</td>
</tr>
<tr>
<td>24</td>
<td>[0,0,0,0,0,1,0,0,0,0,0,0]</td>
<td>Lead Tin Yellow+Malachite</td>
</tr>
<tr>
<td>25</td>
<td>[0,0,0,0,0,1,0,0,0,0,0,0]</td>
<td>Lead Tin Yellow+Verdigris+Malachite</td>
</tr>
<tr>
<td>26</td>
<td>[0,0,0,0,0,1,0,0,0,0,0,0]</td>
<td>Lead Tin Yellow+Verdigris</td>
</tr>
<tr>
<td>27</td>
<td>[0,0,0,0,0,1,0,0,0,0,0,0]</td>
<td>Verdigris+Malachite</td>
</tr>
<tr>
<td>28</td>
<td>[0,0,0,0,0,0,1,0,0,0,0,0]</td>
<td>Azurite+Ultramarine</td>
</tr>
</tbody>
</table>

Table 4.2. Class compositions: the first 12 classes correspond to 12 pure pigments and the last 16 classes correspond to 16 mixture pigments.

(non) linear sample(s) of $m$ pure pigments. Dictionary (D) can be under-complete (set of endmembers, where $M < P$) or over-complete (several samples for each endmember, where $M > P$). Let $x$ be a pixel in the image $I$ with an unknown composition. We are interested in decomposing the spectrum of the unknown pixel $x \in \mathbb{R}^P$ as a function of atoms in the
Figure 4.9. Visible picture of the mock-up samples analyzed, showing the color aspect of six pigment mixture layers - pigment name followed by (*) corresponds to tint, i.e. mixture of coloring pigment with lead white.

dictionary $D$ as in Eq. 4.15.

$$ \mathbf{x} = f(D, \alpha) \quad \text{s.t.} \quad \sum_{i=1}^{M} \alpha_i = 1 \quad \text{and} \quad \alpha_i \geq 0$$

(4.15)

where function $f$ can be linear or nonlinear and $\alpha \in \mathbb{R}^M$ is the concentration values of atoms of the dictionary in an unknown spectrum $\mathbf{x}$. In [132], Rohani et. al. have shown that using over-complete dictionary and applying sparse unmixing algorithm leads to more accurate pigment identification versus using under-complete dictionary and applying linear unmixing algorithm. They assumed $f$ as a linear function and by using kmeans clustering algorithm on pure pigment boxes, for each atom used different linear samples to capture variations inside each pigment. However, the mixing model of pigments is defined by a nonlinear function based on Kubelka-Munk theory which is described in the following.

Typically, the spectrum captured by a hyperspectral camera is a reflectance spectrum. Under some assumptions on opaque film [21] (If the opaque film is thick enough), we can map the reflectance spectrum to an absorption/scattering curve and vice versa by using Eq. 4.17,
respectively. Notice that these equations hold for each wavelength $\lambda$.

$$R_\lambda = 1 + \left(\frac{K}{S}\right)_\lambda - \left[\left(\frac{K}{S}\right)_\lambda^2 + 2\left(\frac{K}{S}\right)_\lambda\right]^{\frac{1}{2}},$$ \hspace{1cm} (4.16)

$$\left(\frac{K}{S}\right)_\lambda = \frac{(1 - R_\lambda)^2}{2R_\lambda}$$ \hspace{1cm} (4.17)

where $R_\lambda$, $\left(\frac{K}{S}\right)_\lambda$ are the reflectance and the ratio of absorption and scattering, respectively, for a given wavelength, according to the Kubelka-Munk (K-M), theory [115]. Clearly, the K-M function is a nonlinear one in both the reflectance space $R_\lambda$ and the $\left(\frac{K}{S}\right)_\lambda$ space since $R_\lambda$ is related nonlinearly to $\left(\frac{K}{S}\right)_\lambda$ in both Eqs. 4.16 and 4.17. Now, according to K-M theory, the $\left(\frac{K}{S}\right)_\lambda$ ratio for a mixture denoted by $\left(\frac{K}{S}\right)_{\lambda,\text{mixture}}$ is given by

$$\left(\frac{K}{S}\right)_{\lambda,\text{mixture}} = \frac{\sum_{i=1}^{m} \alpha_i k_{\lambda,i}}{\sum_{i=1}^{m} \alpha_i s_{\lambda,i}}$$ \hspace{1cm} (4.18)

where $k_{\lambda,i}$ and $s_{\lambda,i}$ are respectively the absorption and scattering coefficients for each constituent $i$ and $m$ is the number of pure pigments in the mixture.

To model the mixing function, there have been two simplified models of the Kubelka-Munk function, based on single- and two-constant simplifications [21]. Berns and Mohammadi [21] have shown that two-constant approach better models the nonlinear function of mixtures. Here, we briefly describe the two-constant K-M model which is more accurate for our experiments. We will use this model to build our dictionary next. In the two-constant K-M model, Eq. 4.18 is used to describe the mixing function for any mixture. However, to find $k_{\lambda,i}$ and $s_{\lambda,i}$ values of each i-th color, some simplifications are considered. Having the ratio of absorption and scattering of white pigment, $\left(\frac{K}{S}\right)_{\lambda,w}$, we find the absorption and scattering
coefficients of white pigment denoted by $k_{\lambda,w}$ and $s_{\lambda,w}$, respectively as

$$s_{\lambda,w} = 1, \quad k_{\lambda,w} = \left( \frac{K}{S} \right)_{\lambda,w}$$  \hfill (4.19)

For other pure pigments (non-white), having the ratio of absorption and scattering of pure pigment and tint denoted by $\left( \frac{K}{S} \right)_{\lambda,\text{pigment}}$, $\left( \frac{K}{S} \right)_{\lambda,\text{tint}}$, respectively, we use the following equations to find absorption and scattering coefficient values of pure pigment denoted by $k_{\lambda,\text{pigment}}$ and $s_{\lambda,\text{pigment}}$, respectively as $k_{\lambda,\text{pigment}}$ and $s_{\lambda,\text{pigment}}$ vectors.

$$s_{\lambda,0} = \frac{\left( \frac{K}{S} \right)_{\lambda,\text{tint}} \times s_{\lambda,w} - k_{\lambda,w}}{\left( \frac{K}{S} \right)_{\lambda,\text{pigment}} - \left( \frac{K}{S} \right)_{\lambda,\text{tint}}}, \quad s_{\lambda,\text{pigment}} = \frac{1 - c_{\text{tint}}}{c_{\text{tint}}} \times s_{\lambda,0}$$

$$k_{\lambda,\text{pigment}} = s_{\lambda,\text{pigment}} \times \left( \frac{K}{S} \right)_{\lambda,\text{pigment}}$$  \hfill (4.20)

where $\alpha_{\text{tint}}$ is the concentration value of the pure pigment ($\alpha_{\text{tint}} + \alpha_w = 1$).

After finding $k_{\lambda,i}$ and $s_{\lambda,i}$ values for each $i$-th color, we can simulate nonlinear mixed pixels by combining different concentrations of any colors using Eq. 4.18.

As mentioned earlier, the linear mixing function does not describe the spectrum of a mixed pixel appropriately. In Fig. 4.10, the spectra obtained by using different concentration values of Ultramarine and Lead White using a linear and a nonlinear (K-M) model are shown. Also the spectrum of Ultramarine tint from the mock-up painting which is composed of 50% Ultramarine and 50% Lead white is depicted by the orange curve for the comparison. As can be observed, the mock-up spectrum is closer to two mixtures with only 10% and 20% Lead white if we use a linear mixing function ($R_{\text{mixture}} = \alpha_{\text{lead white}} \times R_{\text{lead white}} + \alpha_{\text{ultramarine}} \times R_{\text{ultramarine}}$, s.t. to $\alpha_{\text{lead white}} + \alpha_{\text{ultramarine}} = 1$, $0 \leq \alpha_{\text{lead white}} \leq 1$ and $0 \leq \alpha_{\text{ultramarine}} \leq 1$). However, using the nonlinear function (Eq. 4.18), the mock-up spectrum overlaps completely with curve with 50% lead white and is closer to two curves with 40% and 60% Lead white.
Figure 4.10. Comparison of spectra obtained by (a) linear and (b) nonlinear mixing functions using different coefficients of Ultramarine and Lead white. As can be observed the tint spectrum is closer to what we know from the ground-truth information using K-M nonlinear function.

In order to find the coefficient of constituent pigments for any unknown pixel, we simplify the problem by dividing it into two steps:

1. We first find the pigments available in the pixel. Here, we model the problem as either a multi-class or a multi-label classification task.

2. We then approximate the concentration of the constituent pigments. Here, we can use an over-complete dictionary composed of nonlinear atoms using two-constant KM theory with different concentration values of found pigments in the first step and find the closest match between the unknown spectrum and the atoms of dictionary. Another approach is to use the K-M model as the nonlinear function and use the gradient descent method to minimize the reconstruction error of any given spectrum.
4.4.4. Pigment identification

To tackle this problem, we can find the pigments by supervised classification methods. In the first approach, the labels of possible pure/mixture pigments are considered and the multi-class classification approaches can be used to identify the pure pigments for any given spectrum. The description of different pure/mixed pigments are given in Table 4.2. We can use traditional algorithms such as support vector machines or more advanced techniques such as deep neural networks to find the labels. Another approach is multi-label classification approach where each unknown spectrum can have multiple labels. This approach is suitable for situations where two or more categories can be present simultaneously. Assume there are $m$ pure pigments. Here, each output $y$ is a vector of zeros and ones $y \in \mathbb{R}^m$. For our
set of mock-ups, such output vectors are given in the second column of Table 4.2. Here, we train $m$ binary classifiers and each $i$-th classifier predicts if the $i$-th pure pigment is present in the unknown spectrum or not. The main difference between multi-class classification and multi-label classification is in the output format by trained models. In multi-class classification, the classifier returns only one value which corresponds to a class while in a multi-label classification task, the classifier returns a vector of output values $\{0, 1\}^m$ ($m$ is equal to the number of pure pigments).

4.4.5. Multi-class and multi-label classification tasks and the architectures of trained deep neural networks used in pigment identification step

In a multi-class classification task [32], we consider each combination of pure pigments as a separate class. In this task, assuming there are $C$ classes, the output $y$ has only one value from the set $y \in [1, \cdots, C]$. For instance, in the mock-up painting experiment with all possible pairs of mixtures from 12 pigments, we have $C = \binom{12}{2}$ classes. To solve a multi-class classification problem, we can use traditional algorithms such as logistic regression [35] or more advanced techniques such as deep neural networks [39]. Here, given an unknown spectrum, we predict which combination of pigments it belongs to. The limitation here is that we should know all possible classes beforehand and there should be enough samples from each class to be able to train the classifier appropriately. This may not be possible in some real scenarios.

For two classification tasks, we trained a deep feed-forward neural network consisting of several dense layers.

The deep neural network used in the multi-class classification task (third column in table 4.3) is composed of four fully connected layers with 256, 128, 64 and 32 number of
hidden nodes, respectively with sigmoid nonlinear activation function. The last layer is the classification layer with softmax nonlinear activation function.

This model optimizes a loss function defined on the training data. For training the model, we use the average cross-entropy error as the loss function which has been used in multi-class classification problems. A cross-entropy based loss function is defined as follow

$$L = - \sum_{n=1}^{N} \sum_{i=1}^{C} y_{n}^{i} \log p_{i}^{n}$$  \hspace{1cm} (4.21)

where $p_{i}^{n}$ is the model’s probability output for class $i$ when the $n^{th}$ training sample is given to the network, $y_{n}^{i}$ is one if the $n^{th}$ sample is from class $i$, otherwise it is zero, and $N$ and $C$ represent the total number of the training samples and classes, respectively.

Another approach is multi-label classification [33] where each unknown spectrum can have multiple labels simultaneously. In this model, we predict which pigments are present in any given spectrum. In a multi-label classification task, assuming there are $m$ pure pigments, the output $y \in \mathbb{R}^{m}$ of the classifier is a vector of zeros and ones. For our set of mock-ups, such output vectors are given in the second column of table 4.2 in sec. 4.4.2. Here, $m = 12$ and in each row, if the i-th entry has a value of one it determined the presence of the i-th color. Otherwise, a zero value at i-th entry shows the lack of i-th color. Here, we train $m$ binary classifiers and each i-th classifier predicts if the i-th pure pigment is present in the unknown spectrum or not. We do not need to have samples from all possible combinations of pigments and, thus we can train the classifiers with a subset of possible combinations of pigments.

The deep neural network used in multi-label classification task (last column in table 4.3) is composed of four fully connected layers with 256, 128, 64, 32 and 16 number of hidden nodes,
respectively with sigmoid nonlinear activation function. The last layer is the classification layer with sigmoid nonlinear activation function.

This model optimizes a loss function defined on the training data. For training the model, we use the average binary cross-entropy error as the loss function which has been used in binary classification and multi-label classifications problems. A binary cross-entropy based loss function is defined as

\[ L = -\sum_{n=1}^{N} y^n \log p^n + (1 - y)^n \log(1 - p)^n \]  

(4.22)

where \( p^n \) is the model’s probability output for class 1 when the \( n^{th} \) training sample is given to the network, \( y^n \) is one if the \( n^{th} \) sample is from class 1, otherwise it is zero, and \( N \) is the total number of training samples.

For training both deep neural networks, the number of epochs is set to 100 and the batch size to 32. To avoid overfitting, we use early stopping condition with patience set to 5. We use Adam optimizer [40] to optimize the objective function in both architectures which monotonically decreases the learning rate and shows good performance in our experiments. For all our deep learning implementations, we use Python utilizing the Keras [41] library with Tensorflow [42] as the back-end.

As mentioned before, the nonlinear activation functions used in the trained deep neural networks are sigmoid and softmax functions which are defined, respectively as

\[ S(x) = \frac{1}{1 + e^{-x}} \]  

(4.23)

\[ \sigma(x)_j = \frac{e^x_j}{\sum_{j=1}^{C} e^x_j} \]  

(4.24)

where \( x \) is a vector of the inputs to the output layer and \( C \) is the number of classes.
The main difference between multi-class and multi-label classification is in the output format of the trained models. In multi-class classification, the classifier returns only one value which corresponds to a class while in a multi-label classification task, the classifier returns a vector of output values \( \{0, 1\}^m \) (where again \( m \) is equal to the number of pure pigments).

4.4.5.1. Experiment set up. To find the hyperparameters of the classifier, train the classifier and apply it to unseen dataset, for any experiment, first, we split the dataset to training/validation/test sets with 64/16/20 split. During the training step, the model is trained using the training set and it is tested on the validation set to find the best set of hyperparameters of the classifier. In test phase, we apply the trained classifier to unseen test set to check how generalizable the classifier is.

4.4.6. Nonlinear unmixing

The first approach to find the coefficients of constituent pigments for any given spectrum is to compare it with the spectra of reference samples. To do that, we build a reference dictionary where atoms are found by using two-constant K-M model to build nonlinear combinations of pigments with different abundance values. After building the dictionary, we compare the given spectrum with the spectra of atoms of the dictionary using a distance function and approximate the coefficients as the coefficients of the closest match in the dictionary. To find the closest match, we can model the problem as:

\[
\begin{align*}
    d_{\text{closest}} &= \arg\min_j \| \left( \frac{K}{S} \right)_{\lambda, \text{mixture}} - d_j \|_2^2 \\
    \text{with} & \quad d_j = \frac{\sum_{i=1}^{n} \alpha_{i,j} k_{\lambda,i}}{\sum_{i=1}^{n} \alpha_{i,j} s_{\lambda,i}}, \\
    \text{s.t.} & \quad \sum_{i=1}^{n} \alpha_{i,j} = 1, \quad 0 \leq \alpha_{i,j} \leq 1
\end{align*}
\]
where $d_j$ denotes the j-th atom in the dictionary which is composed of $n$ pure pigments with abundance values $\alpha_{i,j}$, for a fixed $j$. The atoms $d_j$ with different coefficients are placed in the dictionary and by finding closest match, the corresponding concentrations are approximated as the coefficients of $d_{\text{closest}}$. We could also use cosine distance function instead of Euclidean distance function in Eq. 4.25 and find the closest match. In the second approach, in order to find the concentration of the pure pigments, using K-M theory, we define the problem as a constrained nonlinear multivariable function. Here, we try to minimize the reconstruction error subject to two constraints, summing up to one and non-negativity constraints for the coefficients.

$$
\min_{\alpha} \left\| \left( \frac{K}{S} \right)_{\text{mixture}} - \frac{\sum_{i=1}^{n} \alpha_i k_{\lambda,i}}{\sum_{i=1}^{n} \alpha_i s_{\lambda,i}} \right\|_2^2 \quad \text{s.t.} \quad \sum_{i=1}^{n} \alpha_i = 1 \quad \text{and} \quad 0 \leq \alpha_i \leq 1 \quad (4.26)
$$

We use gradient-based method to find the concentration values for any given spectrum.

### 4.4.7. Experiments and Results

To find the pigments, we simulated different experiments. For two different classification tasks (multi-class and multi-label), we have used deep feedforward neural network composed of several fully connected layers.

#### 4.4.7.1. Mock-up Experiment

In the first experiment, we only use the mock-up hyperspectral painting. As was mentioned before, there are 28 colors boxes with 16 mixtures of two and three colors. For each class, we select 100 random pixels from each color box. This is the simplest experiment, where the chosen samples from different classes are the most similar to each other compared to next experiments.
Table 4.3. Multi-class classification accuracy (%) of different classifiers for different experiments. SVM, LR and NN refer to support vector machine, multinominal logistic regression and deep neural network.

<table>
<thead>
<tr>
<th>Experiment</th>
<th># of classes</th>
<th>SVM</th>
<th>LR</th>
<th>NN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exp. 1</td>
<td>28</td>
<td>76.78</td>
<td>89.46</td>
<td>99.11</td>
</tr>
<tr>
<td>Exp. 2</td>
<td>28</td>
<td>66.06</td>
<td>70.62</td>
<td>99.08</td>
</tr>
<tr>
<td>Exp. 3</td>
<td>56</td>
<td>78.62</td>
<td>90.70</td>
<td>98.21</td>
</tr>
<tr>
<td>Exp. 4</td>
<td>165</td>
<td>53.91</td>
<td>69.52</td>
<td>96.27</td>
</tr>
</tbody>
</table>

In the second experiment, first, we select random samples from each color box as in the first experiment. Then, for each composition of 16 mixtures, we select 100 random combinations of coefficient values and use K-M model to build random nonlinear mixed pixels and add them to our dataset.

In the third experiment, we generalize the dataset to all possible mixtures of two pure pigments. For each pair of pigments, we select 500 combinations of random coefficient values and use K-M model to build random nonlinear mixed pixels and add to our dataset.

In the last experiment, we build all possible mixtures of three pure pigments. For each mixture, we select 500 random combinations of coefficient values and use K-M model to build random nonlinear mixed pixels and add to our dataset.

To compare the results of deep neural networks with other algorithms, we also trained SVM and multinomial logistic regression classifiers. The logistic regression is modeled by a one layer neural network and the best setting is found by optimizing the hyperparameters of the model. The classification accuracies of trained classifiers for different experiments are given in Table 4.3. As can be observed from the numerical results, the deep neural network has a very high classification accuracy and outperforms other algorithms.

We also studied the classification accuracy of deep neural network for multi-labeling classification task. The results are provided in Table 4.4.
Table 4.4. Multi-label classification accuracy of deep neural networks for different experiments.

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Accuracy (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exp. 1</td>
<td>97.85</td>
</tr>
<tr>
<td>Exp. 2</td>
<td>98.41</td>
</tr>
<tr>
<td>Exp. 3</td>
<td>99.53</td>
</tr>
<tr>
<td>Exp. 4</td>
<td>98.64</td>
</tr>
</tbody>
</table>

Multi-label classification task is a more difficult task as we do not have modeled all possible classes and \( m \) individual binary classifiers are trained for detecting \( m \) pure pigments. However, from Table 4.4, we observe that the classification accuracy for this task is very high and can identify the constituent pigments for majority of the pixels in all experiments.

After identifying the pigments for a given spectrum, we apply two different approaches to find their concentration values. We also apply linear unmixing model to estimate the coefficient values and report the results for three approaches in Figure 4.12 and Table 4.5.

In the first approach, as was mentioned before, we use two-constant K-M model to build a nonlinear dictionary and find the closest match in the dictionary to the given spectrum. The unknown concentrations are approximated as the concentration values of the closest match. Here, the estimated concentration values of six mixtures Verdegris+Lead White, Ultramarine+Lead White, Malachite+Lead White, Lead Tin Yellow+Verdegris+Malachite, Verdegris+Malachite and Azurite+Ultramarine are provided in Figure 4.12. Figure 4.12 (a) shows the average of the estimated coefficients by linear unmixing algorithm for each mixture have been plotted. In Figure 4.12 (b) and (c), the average of the estimated coefficients for each mixture have been plotted finding closest match using Euclidean and cosine distance functions, respectively.

In the second approach, we minimize the reconstruction error using gradient descent and estimate the coefficient values. In Figure 4.12 (d), the average estimated coefficients of
Table 4.5. Numerical results of different approaches (Average Estimated Coefficient): Linear unmixing (FCLS), Closest Match (Euclidean and Cosine distance functions), Nonlinear unmixing.

<table>
<thead>
<tr>
<th>Mixture</th>
<th>Color</th>
<th>FCLS</th>
<th>Euclidean</th>
<th>Cosine</th>
<th>Nonlinear Unmixing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Verdegris+Lead White</td>
<td>Verdegris</td>
<td>0.827</td>
<td>0.5</td>
<td>0.4999</td>
<td>0.501</td>
</tr>
<tr>
<td></td>
<td>Lead White</td>
<td>0.173</td>
<td>0.5</td>
<td>0.5041</td>
<td>0.499</td>
</tr>
<tr>
<td>Ultramarine+Lead White</td>
<td>Ultramarine</td>
<td>0.8514</td>
<td>0.5</td>
<td>0.5</td>
<td>0.501</td>
</tr>
<tr>
<td></td>
<td>Lead White</td>
<td>0.1486</td>
<td>0.5</td>
<td>0.5</td>
<td>0.499</td>
</tr>
<tr>
<td>Malachite+Lead White</td>
<td>Malachite</td>
<td>0.7814</td>
<td>0.502</td>
<td>0.4563</td>
<td>0.5015</td>
</tr>
<tr>
<td></td>
<td>Lead White</td>
<td>0.2186</td>
<td>0.498</td>
<td>0.5437</td>
<td>0.4985</td>
</tr>
<tr>
<td>Lead Tin Yellow+Verdegris+Malachite</td>
<td>Lead Tin Yellow</td>
<td>0.1344</td>
<td>0.5204</td>
<td>0.1929</td>
<td>0.3367</td>
</tr>
<tr>
<td></td>
<td>Verdegris</td>
<td>0</td>
<td>0.1816</td>
<td>0.2918</td>
<td>0.3146</td>
</tr>
<tr>
<td></td>
<td>Malachite</td>
<td>0.8656</td>
<td>0.298</td>
<td>0.5153</td>
<td>0.3487</td>
</tr>
<tr>
<td>Verdegris+Malachite</td>
<td>Verdegris</td>
<td>0.4997</td>
<td>0.3706</td>
<td>0.4335</td>
<td>0.4973</td>
</tr>
<tr>
<td></td>
<td>Malachite</td>
<td>0.5003</td>
<td>0.6294</td>
<td>0.5665</td>
<td>0.5027</td>
</tr>
<tr>
<td>Azurite+Ultramarine</td>
<td>Azurite</td>
<td>0.3889</td>
<td>0.7898</td>
<td>0.5894</td>
<td>0.5132</td>
</tr>
<tr>
<td></td>
<td>Ultramarine</td>
<td>0.6111</td>
<td>0.2102</td>
<td>0.4106</td>
<td>0.4868</td>
</tr>
</tbody>
</table>

each pigment for six sample mixtures have been plotted. As can be observed the estimated concentration values are very close to what we knew from ground truth information. For mixtures of two colors, the estimated coefficient values are very close to 0.5 and for the mixture of three colors, the concentration values are very close to 0.33. The average estimated coefficient values by different algorithms are given in Table 4.5, we also added the numerical results obtained by using cosine distance function and finding the closest match.

As shown in Table 4.5, the estimated coefficients estimated by nonlinear mixing approach are closest to the ground-truth information. Linear unmixing (fully constraint least square method) leads to the worst estimation of abundance values as this algorithm models the mixing function as a linear function. Finding closest match using cosine distance function also leads to decent results except for the mixture composed of three colors.

Without identifying the pigments first and considering all 12 colors, we applied linear and nonlinear unmixing algorithms. The average estimated coefficient values for linear and nonlinear mixing algorithms are reported in tables 4.6 and 4.7, respectively. Mixtures 1 to 6 correspond to Verdegris+Lead White, Ultramarine+Lead White, Malachite+Lead White,
Figure 4.12. Estimated coefficients for six exemplar mixtures by applying different approaches (a) fully constrained least square method, (b) closest match (Euclidean similarity), (c) closest match (cosine similarity), (d) K-M model and gradient descent approach.

Lead Tin Yellow+Verdegris+Malachite, Verdegris+Malachite, Azurite+Ultramarine. As expected, both algorithms assign non-zero values to other colors than the ones actually present in the mixtures which justifies the necessity of applying the first step and pigment identification before pigment unmixing step.
### Table 4.6. Numerical results of applying linear unmixing without identifying the pigments first.

<table>
<thead>
<tr>
<th>Color</th>
<th>Mixture 1</th>
<th>Mixture 2</th>
<th>Mixture 3</th>
<th>Mixture 4</th>
<th>Mixture 5</th>
<th>Mixture 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prussian blue</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0211</td>
</tr>
<tr>
<td>Cobalt blue</td>
<td>0.0000</td>
<td>0.1462</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0263</td>
</tr>
<tr>
<td>Orpiment</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>Vermilion</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0024</td>
</tr>
<tr>
<td>Ivory Black</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0001</td>
<td>0.0161</td>
</tr>
<tr>
<td>Lead Tin Yellow</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0045</td>
<td>0.0000</td>
<td>0.0020</td>
</tr>
<tr>
<td>Ivory Black</td>
<td>0.0000</td>
<td>0.2155</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.1089</td>
</tr>
<tr>
<td>Lead White</td>
<td>0.1545</td>
<td>0.0928</td>
<td>0.2186</td>
<td>0.1110</td>
<td>0.0146</td>
<td>0.0083</td>
</tr>
</tbody>
</table>

### Table 4.7. Numerical results of applying Nonlinear unmixing without identifying the pigments first.

<table>
<thead>
<tr>
<th>Color</th>
<th>Mixture 1</th>
<th>Mixture 2</th>
<th>Mixture 3</th>
<th>Mixture 4</th>
<th>Mixture 5</th>
<th>Mixture 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prussian blue</td>
<td>0.0667</td>
<td>0.0800</td>
<td>0.0605</td>
<td>0.0635</td>
<td>0.0871</td>
<td>0.1243</td>
</tr>
<tr>
<td>Cobalt blue</td>
<td>0.0099</td>
<td>0.1975</td>
<td>0.0088</td>
<td>0.0102</td>
<td>0.0123</td>
<td>0.0351</td>
</tr>
<tr>
<td>Orpiment</td>
<td>0.1016</td>
<td>0.0009</td>
<td>0.0419</td>
<td>0.0898</td>
<td>0.2545</td>
<td>0.0000</td>
</tr>
<tr>
<td>Vermilion</td>
<td>0.0000</td>
<td>0.0009</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>Ivory Black</td>
<td>0.0000</td>
<td>0.2155</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.1089</td>
</tr>
<tr>
<td>Lead Tin Yellow</td>
<td>0.0000</td>
<td>0.0008</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>Verdigris</td>
<td>0.4295</td>
<td>0.0775</td>
<td>0.3103</td>
<td>0.4155</td>
<td>0.4953</td>
<td>0.6293</td>
</tr>
<tr>
<td>Azurite</td>
<td>0.0000</td>
<td>0.0016</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0001</td>
</tr>
<tr>
<td>Smalt</td>
<td>0.0000</td>
<td>0.0017</td>
<td>0.0000</td>
<td>0.0019</td>
<td>0.0045</td>
<td>0.0000</td>
</tr>
<tr>
<td>Ultramarine</td>
<td>0.0000</td>
<td>0.4213</td>
<td>0.0000</td>
<td>0.0070</td>
<td>0.1108</td>
<td>0.0903</td>
</tr>
<tr>
<td>Malachite</td>
<td>0.3923</td>
<td>0.0015</td>
<td>0.5784</td>
<td>0.4121</td>
<td>0.0000</td>
<td>0.0119</td>
</tr>
<tr>
<td>Lead White</td>
<td>0.0000</td>
<td>0.0009</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0355</td>
<td>0.0000</td>
</tr>
</tbody>
</table>

### 4.4.7.2. Real Painting.

The pigment distribution maps obtained following the strategy developed above (first multi-label classification, second estimating coefficient values) are presented in Fig. 4.13b, and are discussed by color hue (Fig. 4.13a).
The red mantle of the character and red flowers present in the decoration area (Fig. 4.13b) present high content of vermilion (HgS). This information is confirmed as spectra reveal shapes similar to semiconductors pigment type vermilion, i.e. a sigmoid with a steeper rise around the inflection point centered at 590-605nm. Present as single pigment in the flower (point 1, (Fig. 4.14b and c), it is mixed with a few amount of lead white (0-30%) in the mantle area. This distribution is confirmed by XRF results probing the presence of mercury and sulfur in similar area (Fig. 4.13c).

The presence of hematite ($\text{Fe}_2\text{O}_3$) pigment is determined in two interior framing lines of the page (Fig. 4.13b). The spectra match with the hematite reflectance reference spectrum, with an inflection peak at 580nm, a minor reflectance peak at 600nm and a more pronounced one at 750nm (Fig. 4.14b and c). Iron is also confirmed by XRF analyses however the spatial resolution offered by this method alone did not allow to propose accurate location and identification of the pigment (Fig. 4.13c).

The blue dress of the character presents a good match with the mineral ultramarine ($\text{Na}_8[\text{Al}_6\text{Si}_6\text{O}_{24}]\text{S}_n$, mineral lazurite), with a reflectance spectrum presenting a maximum of absorption around 600nm and a transition to high reflectance around 700nm. In the case of the blue flowers, a decrease in reflectance between 700 and 900nm is observed, that can be attributed to the presence of azurite ($2\text{CuCO}_3\cdot\text{Cu(OH)}_2$) pigment together with ultramarine (Fig. 4.14b). XRF spectra confirms the significant amount of Cu, consistent with the presence of azurite, together with dominant Al, Si and K signals, consistent with the presence of ultramarine (Fig. 4.13c). These results can be interpreted as a combined use of both pigments to paint vegetable blue area of the illumination. Together with lead white, Pb being identified by XRF, different shades of blue can be achieved as observed in the RGB image of the illumination. Using the KM approach, the different ratio of ultramarine, azurite and lead
white used to paint these decorations have been approximated, showing an increased use of ultramarine in the darker area of the paint, azurite is mixed with ultramarine and lead white mostly in the light blue area (as an example, point 3 presents 36% azurite, 23% ultramarine, and 41% lead white - Fig. 4.14b and c).

![Image](image.jpg)

Figure 4.13. a) Visible picture of the area analyzed; b) Distribution maps of vermilion, hematite, ultramarine, azurite, malachite, lead tin yellow, lead white pigments extracted from hyperspectral datacube using developed approach; c) Elemental map distribution of mercury, iron, silicon, copper, gold, tin and lead from XRF scanning of the same area.

The pigment distribution maps obtained following the strategy developed above (first multi-label classification, second estimating coefficient values) are presented in Fig. 4.13b, and are discussed by color hue (Fig. 4.13a).

The red mantle of the character and red flowers present in the decoration area (Fig. 4.13b) present high content of vermilion (HgS). This information is confirmed as spectra reveal shapes similar to semiconductors pigment type vermilion, i.e. a sigmoid with a steeper rise around the inflection point centered at 590-605nm. Present as single pigment in the flower (point 1, (Fig. 4.14b and c), it is mixed with a few amount of lead white (0-30%)
in the mantle area. This distribution is confirmed by XRF results probing the presence of mercury and sulfur in similar area (Fig. 4.13c). The presence of hematite (Fe$_2$O$_3$) pigment is determined in two interior framing lines of the page (Fig. 4.13b). The spectra match with the hematite reflectance reference spectrum, with an inflection peak at 580nm, a minor reflectance peak at 600nm and a more pronounced one at 750nm (Fig. 4.14b and c). Iron is also confirmed by XRF analyses however the spatial resolution offered by this method alone did not allow to propose accurate location and identification of the pigment (Fig. 4.13c).

The blue dress of the character presents a good match with the mineral ultramarine (Na$_8$[Al$_6$Si$_6$O$_{24}$]S$_n$, mineral lazurite), with a reflectance spectrum presenting a maximum of absorption around 600nm and a transition to high reflectance around 700nm. In the case of the blue flowers, a decrease in reflectance between 700 and 900nm is observed, that can be attributed to the presence of azurite (2CuCO$_3$.Cu(OH)$_2$) pigment together with ultramarine (Fig. 4.14b). XRF spectra confirms the significant amount of Cu, consistent with the presence of azurite, together with dominant Al, Si and K signals, consistent with the presence of ultramarine (Fig. 4.13c). These results can be interpreted as a combined use of both pigments to paint vegetable blue area of the illumination. Together with lead white, Pb being identified by XRF, different shades of blue can be achieved as observed in the RGB image of the illumination. Using the KM approach, the different ratio of ultramarine, azurite and lead white used to paint these decorations have been approximated, showing an increased use of ultramarine in the darker area of the paint, azurite is mixed with ultramarine and lead white mostly in the light blue area (as an example, point 3 presents 36% azurite, 23% ultramarine, and 41% lead white - Fig. 4.14b and c).

The green leaves spectra are characterized by a peak reflectance around 535-540nm and a slow rise from 800nm (Fig. 4.14b). These spectral features point out the use of Cu-containing.
This hypothesis is confirmed by the presence of Cu identified as the main element entering in the composition of the green paints by XRF technique (Fig. 4.13c). For this sample, malachite (CuCO$_3$·Cu(OH)$_2$) presented the best fit with our experimental data (Fig. 4.14b)-however this pigment was not found alone in this area, as lead tin yellow pigment was also identified (the two pigments being found with a 43%, 47% ratios respectively in point 2, Fig. 4.14c. Interestingly, whereas the use of a yellow dye was proposed by the same author in the
leaves area, these results allow establishing a clear distribution correlation between the lead tin yellow found in the leaves and the relatively low but still differentiable Sn XRF signal identified in the same area. These results thus confirm the use of green and yellow mixture in the green leaves.

As the database used to calculate pigment ratios is determined \textit{a priori}, pigment references can be missing, and thus some pixel spectra can be poorly reconstructed. In order to identify unsuccessful pixel spectrum fitting Root Mean Square Error (RMSE) calculation per pixel is calculated.

The results of the RMSE calculation are presented in Fig. 4.14a. Let alone paint edges, and paper support three main area present high RMSE values. First, the gilded area is poorly fitted (point 6, Fig. 4.14b) - whereas similarities can be found with lead tin yellow reflectance spectrum, reference for gold leaves and/or gold-painting are missing. This is a first limitation of the approach as it models diffuse reflections, but is limited for spectral surface treatments like metal foils. Second, the red flower surrounded by a gilded macaroon on the bottom right of the image is poorly fitted (point 8, Fig. 4.14b). With an inflection point around 565nm, the presence of red lead ($\text{Pb}_3\text{O}_4$, mineral form minium) is proposed in this single flower. This has been confirmed by the presence of high content of lead together with the absence of additional elements that could refer to other semi-conductor red pigments. Third, the pink decorations present also very high RMSE values (point 7, Fig. 4.14b). With an XRF spectra dominated by Ca signal and a broad asymmetric band with a maximum centered at 590nm with a constant absorption intensity in the red part. A single absorption band centered at 560nm is observed for this cluster, pointing toward the use a pink dye namely brazilwood applied on a chalk substrate.
The pigment and dye, respectively Red Lead and Brazilwood, are missing in our reference library, resulting in poor reconstruction of the signal where they are present. This highlights the second limitation of the approach: the use of an overdetermined database, this means that for the method to be successful in each pixel of the image the database should contain all references used for the paint system - this might be an iterative process where RMSE is used to determine the area poorly fitted and thus the references to be included. However the use of classification allows to create a much larger database that what actually needed; thus building a shared database of absorption and scattering coefficient for pigments and dyes in the cultural heritage field will be a perspective to consider in order to tackle this kind of study.

4.5. Pigment unmixing of hyperspectral images of paintings using deep neural networks

In this section, the problem of automatic nonlinear unmixing of hyperspectral reflectance data using works of art as test cases is described. We use a deep neural network to decompose a given spectrum quantitatively to the abundance values of pure pigments. We again show that adding another step to identify the constituent pigments of a given spectrum leads to more accurate unmixing results. Towards this, we use another deep neural network to identify pigments first the same as the one used in section 4.4.5 and integrate this information to different layers of the network used for pigment unmixing. As a test set, the hyperspectral images of a set of mock-up paintings consisting of a broad palette of pigment mixtures, and pure pigment exemplars, were measured. The results of the algorithm on the mock-up test set are reported and analyzed.
Deep neural networks (DNNs) have been proven to be successful in determining the nonlinear mappings between an input and the corresponding output automatically. In remote sensing and hyperspectral imaging, different architectures of neural networks have been used in many applications such as feature extraction [36, 155], classification [105, 144, 152] and unmixing [64]. Despite the numerous advantages of DNNs, they have not been applied widely to cultural heritage and only few works have utilized such advanced techniques, such as, [61, 133]. Both of these methods use DNNs for pigment identification. In addition to this, we use a DNN for pigment unmixing as well. This network is trained to learn automatically the nonlinear pigment mixing function using the Kullback-Leibler divergence loss function and is used for the pigment unmixing task. This model can be applied to other nonlinear mixing functions as well, under the condition of having the training sets being built according to the mixing function.

4.5.1. The Pigment Unmixing Problem

Here, we focus on the same problem as mentioned in section 4.4.3 (Eq. 4.15).

In this section, we assume that the painting is composed of $M$ pure pigments or “basic elements”. We refer to the set of $M$ pure pigments as $D$. Let $x$ be the hyperspectral value of a pixel in the image $I$ with an unknown composition. We are interested in decomposing each pixel’s spectrum as a function of the set of pure pigments $D$, according to Eq. 4.15. The function $f$ can be linear or nonlinear and the vector $\alpha \in \mathbb{R}^M$ represents the concentration values of the pure pigments in the representation of the unknown spectrum $x$. It has been shown in the literature that the mixing model of the spectra of pigments is defined by a nonlinear function based on KM theory [87]. In [132, 133], we use sparse linear and nonlinear KM functions, respectively, to decompose any given spectrum into its basic elements. We show
that the sparse unmixing algorithm yields more accurate results than the linear unmixing algorithm using a fully constrained least square method [7]. In [133], we first detect the pigments in a given spectrum using a DNN and then by a using gradient-based algorithm [138], we estimate the coefficients of the detected pigments for the given spectrum. Here, we utilize a DNN to learn the pigment nonlinear mixing function and find the coefficients of the pigments, \( \alpha \), in a given spectrum.

### 4.5.2. Developed Model

As mentioned in section 4.5.1, to find the concentration values of the pigments in any spectrum, a DNN is trained. The developed DNN is a feed-forward network, composed of four fully connected layers with 256, 128, 64 and 32 number of hidden nodes, with relu/sigmoid nonlinear activation functions (Fig. 4.15(a)). The softmax nonlinear function is used as the activation function of the last layer with \( M = 11 \) number of hidden nodes, so that the summing up to one constraint on the coefficients holds (Eq. 4.15). The input to the network is the reflectance(absorption/scattering) spectrum vector of a pixel and the output is the estimated coefficient vector \( \alpha \) of pigment concentrations for the given spectrum. To obtain a more precise result, we can first identify the pigments for any unknown spectrum and use this information to decompose the given spectrum into the concentration values of the detected pigments. As shown in section 4.4.7 (table 4.6), the performance of the pigment unmixing algorithm improves when prior determination of expected pigments is used. To identify the pigments for a given spectrum, we use a deep feed-forward network as described in section 4.4.5. Pigment identification can be modeled as a multi-label classification [143] problem. Here, each spectrum can have multiple binary labels due to the presence of multiple pigments in it.
Figure 4.15. Architectures of developed networks (a) baseline network with the spectrum as the input, (b) a network with the spectrum and labels concatenated in the input layer (Early fusion), (c) a network with the spectrum as the input to the first layer and labels concatenated to the output of the third layer (Intermediate fusion), (d) a network with the spectrum as the input to the first layer and the output of the fifth layer multiplied by labels (Late fusion), (e) Top: Reflectance spectra of 11 colors used in dataset. Bottom: Visible picture of exemplar mock-up mixtures. The mixtures are: 1) Minium + Lead White 2) Madder Lake + Lead White 3) Jaoriste + Lead White 4) Red Ochre + Yellow Ochre and 5) Red Ochre + Yellow Ochre + Egyptian Blue

The input to the network is the spectrum and the output is a vector of zeros and ones, \( y \in \{0, 1\}^M \). In this network, in the last dense layer, the sigmoid nonlinear activation function is used and the objective loss function of the network is a \textit{binary cross-entropy} based loss. The details of the network are given in section 4.4.5 which is very similar to the network in Fig. 4.15 (a) (the same number of layers and hidden nodes in each layer with different output).
The detected labels, \( y \), can be used as an extra input to the network, an “auxiliary input” in Fig. 4.15 (a) and can be merged with different layers of the network as shown in Figs. 4.15 (b), (c) and (d). In Fig. 4.15(b), Early fusion the labels (a vector of \( M = 11 \) zeros and ones) are concatenated to the spectrum input (of dimension 56) in the first layer resulting in an input with a dimension of 67 where the first 56 terms represent the spectrum and the last 11 terms the labels. In Fig. 4.15 (c), Intermediate fusion the labels are concatenated to the output of the third layer. Lastly, in Fig. 4.15 (d), Late fusion the output of the fifth layer is multiplied by the labels. Finally, another dense layer with \( M = 11 \) hidden nodes is added to the network in Fig.4.15 (d) to estimate the concentration values. It has been verified by our experiments that the performance of the network enhances when the labels are integrated into the outputs of the higher layers of the network (Fig. 4.16).

4.5.2.1. Training. For training the networks developed for pigment unmixing, we can use the Mean Squared Error (MSE), the cosine error or the Kullback-Leibler Divergence (KLD) as the loss function. Due to constraint on the coefficient values to sum up to one (Eq. 4.15), we consider the true and predicted coefficient values of the pigments as belonging to two probability distributions. Therefore, KLD would be an appropriate loss function for comparing the similarity of these two probability distributions. Based on our simulations, we have observed that the KLD loss function leads to a better performance than the MSE or the cosine loss functions. The KLD based loss function is defined as follows:

\[
L = - \sum_{n=1}^{N} \sum_{i=1}^{M} \alpha_i^n \log(\hat{\alpha}_i^n) \quad (4.27)
\]

where \( \hat{\alpha}_i^n \) and \( \alpha_i^n \) are the predicted and true coefficient values of pigment \( i \) for training sample \( n \), respectively. \( N \) and \( M \) again are the total numbers of the training samples and number
of pure pigments, respectively. We use the Adam [82] optimizer with default configuration parameter values to optimize the objective loss function.

For all our deep learning implementations, we use Python utilizing the Keras library [37] with Tensorflow [20] as the backend. The number of epochs and the batch size are set to 200 and 64, respectively. To avoid overfitting, an Early Stopping callback is used and the number of patience is set to 10.

4.5.3. Data Sets

In this section, the training (simulated) and test (mock-up) sets used in the experiments are described.

4.5.4. Mock-up Dataset

We have prepared a set of mock-up paintings composed of 11 pure colors and 41 mixtures. The average spectra of 11 pure pigments in the reflectance space are shown in Fig. 4.15 (e) top and the visible figure of five exemplar mock-up mixtures is provided in Fig. 4.15 (e) bottom. There are 30 mixtures of two colors composed of white pigment and a non-white pure pigment where three different proportions of white pigment (20%, 50% and 70%) have been used. There are 4 mixtures that are composed of two non-white colors with same proportions ($\frac{1}{2}$ of each color). The remaining 7 mixtures are composed of three colors with the same proportions ($\frac{1}{3}$ of each color). However, since no sophisticated unmixing and layering equipment were used, it is impossible to guarantee that in every position in the mock-up square the same concentration of a specific color is used. Unfortunately, the assumption of specific proportions of the colors represents the ground truth information we will be using. We build our hyperspectral image, which will be used as the test set, by concatenating 11
pure pigments and 41 mixtures in a rectangle. From each pure/mixture box, we select a 20 × 20 square to make all the mixtures have the same number of pixels.

4.5.5. Simulated data

To train the network so that the mixing function is learned we need a dataset with known concentration values. Since for any experimental and real dataset, acquiring exact quantitative information is not possible as mentioned earlier, we resort to simulated data. A nonlinear KM mixing function [21, 22, 87] is used to generate simulated data. We use the 11 pure pigments in our set of mock-up paintings to build the mixtures.

We consider mixtures of two or three colors and build all possible mixtures of two/three pure pigments. For each mixture, 500 random combinations of coefficients/concentration values summing up to one are selected and the KM function [87] is used to model random nonlinear mixtures (refer to Eq. 4.15 and replace mixing function with the KM nonlinear function, as shown in Eq. 4.18 in section 4.4.3). This dataset is used to train networks for the pigment identification and unmixing tasks.

4.5.6. Pre-processing

Our captured spectra consist of 240 bands with 2nm resolution from 383 to 893 nm. We remove the noisy channels and to reduce the noise in the signal further, a spectrally moving average filter is used and the number of channels is decreased by a factor of 4 leading to a smoother signal with 56 bands. Also, all pixel values are normalized by the reflectance scale factor of the hyperspectral camera (4095).
4.5.7. Results and Discussion

As mentioned in section 4.5.3, we use the simulated data as our training dataset to find the optimal hyperparameters of the deep neural networks and mapping function between the spectrum and coefficient values. The set of mock-up paintings is used as the test sets here.

4.5.8. Pigment identification

As mentioned before to detect the constituent pigments in any given spectrum, a multi-label classifier [143] using a DNN is used where each unknown spectrum can have multiple labels.

Here, we train 11 binary classifiers and each i-th classifier predicts if the i-th pure pigment is present in the given spectrum or not. The accuracy of the trained classifier on the mock-up test painting is 98.4%.

4.5.9. Nonlinear unmixing

The network learns the mixing function of the pigments using the training set from the simulated data. Due to the space limitation, the average predicted coefficients by different network architectures (Fig. 4.15 (a)-(d)) for five exemplar mixtures from the mock-up (Fig. 4.15 (e) bottom) are given in Fig. 4.16. The studied mixtures are: Minium + Lead White (50%, 50%), Madder Lake + Lead White (80%, 20%), Jaoriste + Lead White (30%, 70%), Red Ochre + Yellow Ochre (50%, 50%) and Red Ochre + Yellow Ochre + Egyptian Blue (∼33%, 33%, 33%).

In Fig. 4.16, each bar refers to a mixture, for which the dark blue, light blue and green show the average estimated coefficient values of the first, second, third (if present in the mixtures) pigments in the mixtures. In Fig. 4.16 yellow refers to the average estimated
Figure 4.16. Average estimated coefficient values by different networks in Fig. 4.15 (a) network architecture 4.15(a), (b) network architecture 4.15(b), (c) network architecture 4.15(c), (d) network architecture 4.15(d), (e) linear unmixing algorithm (Eq. 4.28), (f) linear unmixing algorithm using labels (applied to the subset of the spectra of the present pigments in any given spectrum).
coefficient values of other pigments (not present in the mixture) that the model has predicted wrongly. In Fig. 4.16 (a), we observe that architecture (a) which does not use the information of the labels has the worst performance and describes the mixtures by other pigments than the real constituent pigments (large yellow proportions in mixtures 2, 4 and 5). The predicted coefficients for mixtures 1 and 3 are fairly good. In Fig. 4.16 (b), we observe that the performance of the architecture (b) is better than architecture (a) and does not pick wrong pigments. However, the predicted coefficients for mixtures 2, 3 and 4 are still far from the expected values. The estimated coefficients of architecture (c) (Fig. 4.16 (c)) are closer to the expected values especially for mixtures 2, 4 and 5 compared to the architecture (b) and it seems that the network is learning the unmixing function better than the architectures (a) and (b). The best results is related to the last network (Fig. 4.16 (d)), where the labels are integrated into network in a higher layer. The estimated coefficients for two colors in mixtures 1 and 4 are close to 50 as expected. In mixture 2, the proportion of pigment 1 is close to 80 and in mixture 3 its proportion is close to 30 as expected. In mixture 5, the coefficients of three colors are similar. It seems that the mixtures with white pigment are easier for the network to decompose. On the other hand, the mixture with three non-white pigments seems to be the most challenging. This may be due to the fact that the nonlinear KM mixing function used in building the non-white mixtures may not be as accurate as for the mixtures with white pigments in our settings.

For the comparison with the algorithms used for pigment unmixing, we also studied the performance of linear unmixing algorithm using a fully constrained least square method according to
\[ x = \alpha E, \quad \text{s.t.} \quad \sum_{i=1}^{M} \alpha_i = 1 \quad \text{and} \quad \alpha_i \geq 0, \quad (4.28) \]

In Fig. 4.16 (e), we did not use the labels information and decomposed the given spectrum to a linear combination of spectra of all \( M = 11 \) pure pigments (Eq. 4.28). Similar to the architecture (a) (Fig. 4.15 (a)), the linear unmixing algorithm’s performance in detecting the correct pigments is poor and we observe that there are large yellow portions in mixtures 2, 3, 4 and 5. In Fig. 4.16 (f), we used the labels information and decomposed the given spectrum to a linear combination of the spectra of a subset of \( E \) containing only the present pigments in a given spectrum. As can be seen from Fig. 4.16 (f), the linear unmixing performance is improved compared to Fig. 4.16 (e). However, it is much worse than architectures (c) and (d) and the estimated coefficients for all mixtures are still far from the expected values. As was expected, based on the results shown in Fig. 4.16, we observe that DNNs model the mixing function more accurately and yield better results than linear unmixing algorithm.

We also applied the trained network to a real Fayum dataset and due to scarce appearance of pigments Egyptian Blue and Vermilion, we used XRF elemental maps to identify the pigments. Here we have four elemental maps for Iron (Fe), Lead (Pb), Copper (Cu) and Mercury (Hg). By using a threshold value on XRF maps, we get binary labels and use architecture (d) to estimate the concentration values. The results of the algorithm have been provided in Fig. 4.17. By comparing the Iron XRF elemental map in Fig. 4.17, we observe that the network has described the mixtures containing iron as a mixture of Red Ochre, Yellow Ochre, Jarosite and Iron Oxide Red. From Lead XRF elemental map, we observe that the concentration values of Minium and Lead White are estimated as different values. By comparing the Egyptian Blue concentration map and Copper elemental map, we observe
there is a high correlation. Mercury XRF elemental map and Vermilion concentration map also have high correlation.

To have an end-to-end deep neural network architecture for pigment identification and unmixing, we combined the DNN trained for pigment identification with DNN trained for pigment unmixing using architecture d. The final architecture is shown in Fig 4.18.
Figure 4.18. End to end DNN for pigment identification and unmixing.

As shown this network has two inputs spectra in R and in K/S spaces and two outputs as the labels $y$ and concentration $\alpha$ vectors. The loss function of this network is defined as the combination of classification and regression losses which are binary cross entropy and kld divergence loss, respectively. We also use another parameter $\lambda$ to find the combination of these losses which leads to best unmixing performance.

$$L = -\lambda \left( \sum_{n=1}^{N} \sum_{i=1}^{M} y_i^n \log \hat{y}_i^n + (1 - y_i)^n \log (1 - \hat{y}_i)^n \right) - (1 - \lambda) \sum_{n=1}^{N} \sum_{i=1}^{M} \alpha_i^n \log \left( \frac{\hat{\alpha}_i^n}{\alpha_i^n} \right)$$

(4.29)

where $M$ and $N$ are the number of pure pigments and number of training samples, respectively. $y_i^n$ and $\hat{y}_i^n$ are the true binary label of and the predicted probability of training
sample \( n \) for pigment \( i \). \( \alpha_i^n \) and \( \hat{\alpha}_i^n \) are the true and predicted concentration values of training sample \( n \) for pigment \( i \). The estimated coefficients for the exemplar five mixtures with \( \lambda = 0.2 \) are given in Fig. 4.19.

From Fig. 4.19, we can observe that the end to end architecture results are the closest to what we expected and we can conclude that the network shown in Fig. 4.18 learns the mixing function the most accurate.

4.6. Conclusions

In this chapter, we introduced the first application of sparse unmixing to the problem of pigment identification. This method addresses several limitation of previous methods applied to the pigment identification problem. We showed that distance based methods such as SAM and SCM do not work well in scenarios with highly mixed regions and that our sparse unmixing algorithm can estimate the pigment composition more accurately than a linear unmixing method.
This study presents a new technique for the visualization of similarity between hyperspectral data that is capable of retaining the local structure while also revealing some important global structure of the datacube. Our experiments on mock-up paint samples show that t-SNE outperforms existing state-of-the-art techniques for visualizing and segmenting pigment fingerprints in a reduced data space, in particular in the case of highly absorbing pigments.

Three main advantages of the technique can be mentioned to clarify this result. First, instead of preserving the distances between widely separated datapoints, t-SNE preserves the distances between nearby datapoints and thus the similarities between datapoints rather than dissimilarities. Moreover, whereas PCA and MNF are hampered by non-linear feature relations, the non-linear approach proposed by t-SNE allows for the modeling of any curved manifolds within the dataset. Finally, for visualizing the structure of very large datasets, t-SNE uses random walks on neighborhood graphs to allow the implicit structure of all the data to influence the way in which a subset of the data is displayed. Consequently, the final results provide a straightforward visualization of the clustering of the dataset. This explains the fact that in this study t-SNE has been used alone without preprocessing, or further segmenting strategies – the clustering being selected manually based on the cluster information visually observed in the reduced dataset created. This approach was successfully applied to a historical artifact to segment and identify paint composition. In particular, for the illuminated page studied, four red pigments and dyes have been identified as pure material: cinnabar, vermilion, red madder, pink Brazil wood. For the blue pigment, both pure ultramarine and mixture of ultramarine and azurite have been found. Finally two types of green were differentiated, both Cu-based pigments from which no further information can be extracted without further analytical techniques. These results provided pigment/paint mixture maps that have been correlated to XRF results, confirming the information extracted
using t-SNE approach. This technique represents a new opportunity for hyperspectral data reduction and visualization in the cultural heritage. Whereas it has been combined with manual clustering, some further development in terms of data segmentation could be proposed, and an approach similar to k-means clustering and PPI could benefit the results. Whereas more powerful than classical approaches for data reduction and classification, the drawbacks of computation time and memory allocation were faced when t-SNE was run using standard computer performances. That led to a loss of resolution of the original images (often by a factor of 4). Strategies to speed up the calculation process using more powerful processors, or performing the calculation on previously clustered dataset can be envisaged for future works.

In this chapter, we studied the automatic pigment identification/unmixing in hyperspectral paintings. Using deep neural networks and supervised classification approaches, the pigments present in any given spectrum is found. Using nonlinear dictionary or gradient descent approaches, the concentrations of the pigments are approximated. We studied a set of mock-up paintings and showed deep neural networks identify the pigments for any given spectrum with a high accuracy. Using nonlinear unmixing approach, the pigment ratios were estimated and proven to be very close to the ground-truth information. We also applied the developed method to a HSI dataset acquired on a historic manuscript. Our results were consistent with elemental information obtained using XRF scanning system, and provided a more detailed identification and quantification of the pigments entering into the composition of the paint layers. It is a key aspect for medieval manuscripts, as the palette and pigments used by illuminators are often very similar making distinction between hands or workshops difficult. By studying the way pigments are mixed and applied new clues about the paint techniques can be unraveled. Whereas the approach has proven to be successful for most of the red, blue and green paint layers, a few area where poorly fitted due to a lack of spectra
references. Thus, further work to expand the current pigment database is necessary in order to provide a tool that can be daily used in museum and elsewhere to semi-quantitatively study surface painted materials.

In this section, we studied the automatic pigment identification/unmixing in hyperspectral paintings. Using deep neural networks and a supervised multi-label classification approach, the pigments present in any given spectrum are found. Using another deep neural network, the concentrations of the pigments are approximated. We studied a set of mock-up paintings and showed that the DNNs identify the pigments for any given spectrum with high accuracy. We integrated the label information to different layers of the network and showed that the pigment unmixing performance of the networks is better when late fusion is used and the estimated coefficient values are closer to what is expected. Finally, we developed an end-to-end architecture which consists of two sub-networks. The two parts relate to the multi-label classification and regression parts. It has been shown that this architecture estimated the concentration values more accurately than other developed models.
CHAPTER 5

Convergence of Weights in Deep Neural Networks

We show that for many datasets it may be possible to compute weights of layers in a neural network directly from training data, without stochastic gradient descent (SGD). We find that with ReLU activation and Glorot initialization, training dynamics of layers approximately decouple, with layers closer to input showing faster training dynamics than layers above. We exploit this to analytically calculate the approximate outcome of SGD for each layer and find that it should converge to a class-based PCA, with some weights in every layer dedicated to principal components of each label class. The class-based PCA allows us to train layers directly, without SGD, often leading to a dramatic decrease in training complexity. We demonstrate the effectiveness of this by using our results to replace one and two convolutional layers in networks trained on MNIST, CIFAR10 and CIFAR100 datasets, showing that our method achieves performance superior or comparable to ConvNet. We also show that in our method often a fraction of the training data is sufficient to train layers.

5.1. Introduction

Deep neural networks (DNN) are useful for tasks which can be broken down into a hierarchy smaller tasks, somtimes requiring exponentially less parameters than their shallow counterparts [141]. But training DNN via SGD involves coupling of weight matrices of many layers through error backpropagation. Naively, it seems that the landscape of the cost function should include a multitude of local minima, similar to spin glasses [55, 83], making discovery
of “good” solutions extremely difficult. On the other hand, problems with a clear hierarchy of features can become simpler if the hierarchy is exploited, as is the case for Quicksort and the Barnes-Hut algorithm [18]. Similarly, learning to classify natural scenes becomes more efficient if they are broken down into a low and high-level features [86]. For example faces become easier to classify if the system starts by learning line and curve segments, combining them to make eyes, nose and mouth shapes, and so on. The key point here is that, learning low-level features (e.g. eye shape) should not require learning of high-level features (e.g. face composition) at the same time. Since lower layers (closer to input) in DNN seem to learn low-level features [104], there must exist a way to setup the training dynamics of DNN in which training of lower layers is virtually independent of the accuracy of higher layers. In terms of the learning dynamics, this would mean that the dynamics of weights of higher layers would play little role in training of lower layers. If this effective “decoupling” of learning dynamics of layers occurs, SGD should be able to train the network in a hierarchical fashion, starting from low-level features. Thus, one of the goals of the present work is to find settings under which training a deep neural network simplifies. In particular, we wish to know

1. Under what conditions would SGD equations exhibit decoupling in learning dynamics for low and high-level features, learning them in hierarchical fashion?

2. What low and high-level features are learned when this decoupling occurs?

Below we derive the conditions for such decoupling of learning dynamics and find that they are indeed satisfied in some of the most successful setups and initialization of DNN. This decoupling greatly simplifies the SGD equations, allowing us to estimate the outcome of the training.
5.2. Stochasticity and the learning dynamics

The problem that we will focus on is classification: We wish to map a set of input vectors $X = (x_1, \ldots, x_N)$ to output label vectors $Y = (y_1, \ldots, y_N)$ (assumed to be “one-hot” vectors) using a nonlinear function $f(\theta; X) = Y$, where $\theta$ are the parameters (i.e. weights and biases). The function $f(\theta; X)$ summarizes the action of a neural network with parameters $\theta$ on the data $X$. We discuss the structure of the neural network below. To assess the goodness of fit, we have a cost function $g(\theta, X, Y)$, which we write as $g[\theta]$. The $\theta$ are assumed to be bounded. The goal is to find $\theta$ such that the cost function $g[\theta]$ is minimized. $g[\theta]$ is also assumed to be smooth, except on a set of measure zero. The standard method for training DNNs is SGD. In SGD, data is processed gradually in mini-batches. In early steps, the $\theta$ will have stochastic fluctuations of order $\sigma_\theta/\sqrt{N}$, with $N$ being the number of data inputs processed. Given a cost function $g[\theta]$ with a set of parameters $\theta$ (weights $w$ and biases $b$), the standard error defines a resolution limit for SGD at step $N$: local minima in the landscape of $g[\theta]$ whose widths are smaller than $\sigma_\theta/\sqrt{N}$, ($\sigma_\theta$ is the standard deviation of $\theta$) are invisible to SGD because stochastic fluctuations are large enough to make $\theta$ escape such local minima (SI Fig. 5.1). An important consequence of this is that, much like temperature in simulated annealing [83], standard error results in an effective “smoothing” of the energy landscape in SGD in the usual sense of “Kernel Density Estimation” [48,146], forcing it to find a low energy (cost) local minimum in the smoothened landscape of $g[\theta]$. As SGD progresses, the standard error diminishes and narrow local minima become resolvable.

Near the bottom of the smoothened minimum stochastic fluctuations dominate and the gradient becomes negligible. Thus, SGD will have two phases: 1) A “fast drift phase” driven by strong gradients, with a consistent direction; 2) A “relaxation phase” near the bottom
Figure 5.1. **Schematic of the smoothing effect of standard error on the landscape of the cost function** $g[\theta]$. At step $N$ the standard error in mean of parameters $\theta$ is $a\sigma = \sigma_\theta/\sqrt{N}$. Thus, stochastic fluctuations can move the $\theta$ by a normal distribution of width $a\sigma$. The effective energy landscape is then the convolution of the standard error distribution with the original $g[\theta]$ (dashed blue curve), resulting in a smoothing of the landscape at low $N$ (orange, $N = 100$). At high $N$, the standard error is negligible and the original $g[\theta]$ is recovered (green $N = 4000$).

of a local minimum\(^1\). These two phases were observed and utilized by [136] where the fast drift phase is described as a “representation compression” phase. Fig. 5.2 shows an actual SGD for a convex $g[\theta]$ to illustrate the two phases. Note that these two behaviors are a quite general feature of any stochastic process, such as diffusion, happening on an energy landscape with local minima. We show below that the fast drift phase can lead to a “separation of time scales”: Layers closer to data have much faster dynamics than layers above them. The condition for this to happen is having an activation function which is unbounded\(^1\).

\(^1\)Note that these phases may occur multiple times as different layers may be entering this phase at different times and the landscape becomes less smooth during SGD.
The two phase of Stochastic Gradient Descent (SGD): The dynamics consists of two distinct phases: a fast drift phase, when the gradient of the cost function is large; a relaxation phase, where the gradient is negligible and stochastic fluctuations dominate the dynamics. In SGD, the spread of the fluctuations is due to statistical fluctuations in the input data used to estimate the cost function. As the number $N$ of training data increases, the spread of the fluctuations decreases like $\sigma_\theta/\sqrt{N}$ and the weight $\theta$ settles at the local minimum.

from above and a suitable initialization, e.g. Rectified Linear Units (ReLU) with Glorot initialization [58, 66].

When the system enters the relaxation phase, this separation of time scales still exists and we can utilize it to considerably simplify the SGD equations. The relaxation phase is a period where the gradients mostly vanish and the dynamics is dominated by stochastic fluctuations. The system will be fine-tuning to find the exact position of the local minimum, as the details of landscape of $g[\theta]$ become resolvable. It is important to note that each layer may enter the relaxation phase at a different time step. We will exploit this to analytically derive the distribution of weights. We will now delve into the details of the setup of the problem and the SGD equations.
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n$</td>
<td># of layers</td>
</tr>
<tr>
<td>$N$</td>
<td># of training data</td>
</tr>
<tr>
<td>$C$</td>
<td># of label classes</td>
</tr>
<tr>
<td>$d^{(k)}$</td>
<td>dimension of layer $k$ output</td>
</tr>
<tr>
<td>$x_i, y_i$</td>
<td>$i$th input and label</td>
</tr>
<tr>
<td>$w^{(k)}, b^{(k)}$</td>
<td>weights and biases of layer $k$</td>
</tr>
<tr>
<td>$h_{ia}^{(k)}$</td>
<td>output of layer $k$, channel $a$, image $i$</td>
</tr>
</tbody>
</table>

Table 5.1. Notations

5.3. Setup and notation

Consider the problem of classifying $N$ input vectors $X = (x_1, ..., x_N)$ into $C$ classes by assigning $N$ one-hot vectors $Y = (y_1, ..., y_N)$ as output labels. We want to achieve this task using an $n$ layer neural network defined below. The dimension of the output, or number of output “channels”, of layer $k$ is denoted by $d^{(k)}$. The dimensions of the output of the classification layer is $d^{(n)} = C$, the dimension of the label vectors $y_i$. The output of layer $k$ of the network is

$$h^{(k)} = f\left(\bar{h}^{(k)}\right), \quad \bar{h}^{(k)} = w^{(k)} h^{(k-1)} + b^{(k)}$$

(5.1)

where $f(\cdot)$ is the activation function and $\bar{h}^{(k)}$ is the “raw output”. $w^{(k)}$ and $b^{(k)}$ are the weights and biases of layer $k$, respectively. $h_{ia}^{(k)}$ refers to channel $a$ of the output of layer $k$, for the $i$th input. Thus, $h_i^{(0)} = x_i$ is the input and $h_i^{(n)}$ is the corresponding output of the network. The last layer is the classification layer with number of hidden nodes equal to the number of classes (i.e. $d^{(n)} = C$).

Table 5.1 summarizes our notation.

In most of what follows, we will only show the layer index $k$ and sometimes the input index $i$. Matrix multiplication is implied unless stated otherwise. For the classification layer,
denoted as layer \( n \), we choose a softmax activation function, defined as \( h_{ia}^{(n)} = Z_i^{-1} \exp[\tilde{h}_{ia}^{(n)}] \), with the normalization factor being the sum over classes \( Z_i = \sum_{a=1}^{C} \exp[\tilde{h}_{ia}^{(n)}] \), and use categorical cross-entropy as the cost function

\[
g(h^{(n)}, y) = -\frac{1}{N} \sum_{i,a} y_{ia} \log h_{ia}^{(n)} = -\frac{1}{N} \left( \sum_{i,a} y_{ia} \tilde{h}_{ia}^{(n)} - \sum_{i} \log Z_i \right) \tag{5.2}
\]

Gradient descent consists of changing the weights and biases opposite to the gradient of the cost function to find minima. Generally the network is trained using stochastic gradient descent (SGD), using the number of processed data points \( N \) to define time steps\(^2\), and updating weights and biases opposite to the gradient of the cost function. When a new mini-batch of size \( \delta N \) in each step is processed, the weights and biases will be changed according to

\[
\frac{\delta b^{(k)}}{\delta N} = -\varepsilon \frac{\delta g}{\delta b^{(k)}}, \quad \frac{\delta w^{(k)}}{\delta N} = -\varepsilon \frac{\delta g}{\delta w^{(k)}T} \tag{5.3}
\]

where \( \varepsilon \) is the learning rate, which can be dynamically adjusted, and \( \delta w \equiv w(N + \delta N) - w(N) \).

The components of gradient of the cost function are

\[
\frac{\delta g}{\delta b^{(k)}} = \left( \frac{\partial g}{\partial h^{(n)}} \right)^T A^{(k+1)T}, \quad \frac{\delta g}{\delta w^{(k)}T} = h^{(k-1)} \frac{\delta g}{\delta b^{(k)}} \tag{5.4}
\]

where \( \delta w \equiv w(N + \delta N) - w(N) \), and \( \varepsilon \) is the learning rate, which can be dynamically adjusted.

\(^2\)During training, data points are also reused. After one “epoch”, i.e. after processing all training data once, the mini-batches are randomly sampled from the data set again. For our purpose, we may treat reused data after the first epoch as new data. We will not make a distinction between the different epochs.
\[ \delta g = \sum_{k=1}^{n-1} \left( \frac{\partial g}{\partial w^{(k)}} \delta w^{(k)} + \frac{\partial g}{\partial h^{(k)}} \delta h^{(k)} \right). \] (5.5)

where \( k \) runs over the layers.

The matrix \( A^{(k+1)} \) follows from backpropagation (i.e. chain rule)

\[ A^{(k)} = \prod_{m=k}^{n-1} \left( \frac{\partial \tilde{h}^{(m+1)}}{\partial \tilde{h}^{(m)}} \right)^T = \prod_{m=k}^{n} \tilde{w}^{(m)}, \quad \tilde{w}^{(m)} = \text{diag}(f'(h^{(m-1)}))^T w^{(m)}. \] (5.6)

with \( \text{diag}(f') \) representing a diagonal matrix with the derivative of the activation function of the hidden layers on the diagonal. A technical difficulty arises from \( f' \) becoming vanishingly small or exponentially large, resulting in numerical overflow or bad convergence [19]. While in the past tanh and sigmoid were commonly used as activation functions, most recent DNN architectures favor ReLU activation, defined as \( f(x) = \max\{0, x\} \), due to a number of reasons including lower computational complexity, constant gradient, faster learning, and a reduced likelihood of vanishing gradient problem [19,102]. We will consider a feed-forward network, with the first \( n - 1 \) layers having ReLU activation function\(^3\). With ReLU, the output \( h^{(k)} \) of layer \( k < n \) can be written as

\[ h^{(k)} = \text{diag} \left( \theta \left( \tilde{h}^{(k)} \right) \right) \tilde{h}^{(k)}, \quad A^{(k)} = \prod_{m=k}^{n} \text{diag} \left( \theta \left( \tilde{h}^{(m-1)} \right) \right) w^{(m)}. \] (5.7)

where \( \theta(h) \) is the Heaviside step function and we have used the fact that \( h \partial_h \theta(h) = h \delta(h) = 0 \).

We wish to understand the dynamics of Eq. (5.4). The factors that depend on \( k \) in Eq. (5.4) are the weights of higher layers through \( A^{(k+1)} \) and the output of the previous layer

\(^3\)Note that, although ReLU is not a smooth function, it won’t cause any problems in SGD because the non-smoothness is on a set of measure zero and discrete methods such as SGD will never discover the non-smooth part of the domain.
\( h^{(k-1)} \). We will show below that with ReLU the Frobenius norm of \( A^{(k+1)} \) will be larger for smaller \( k \). For \( h^{(k-1)} \), we will show that it will contain competing terms and together with the nonlinearity from ReLU, \( h^{(k-1)} \) will not have a clear factor of weights and biases like \( A^{(k+1)} \).

5.4. Fast drift phase and singular values of \( w^{(k)} \)

It is known that initial values of the weights affect the accuracy of DNNs significantly [66]. For ReLU, the best performing initialization scheme is found to be the Glorot method [58, 66] which limits the variance of weights so that the initial singular values of initial weights are equal to two. We will now show that Glorot initialization leads to a separation of time scales because it will result in \( A^{(k)} \) acquiring singular values (SV) which are greater than 1 and larger for smaller \( k \). Fig. 5.3 shows experimental evidence supporting this claim in a test network consisting of four dense layers\(^4\) trained on the MNIST dataset. It shows that \( w^{(k)} \) of all layers quickly acquire \( SV > 1 \) and that the product \( \prod_{m=k}^{n} w^{(m)} \), as a proxy for \( A^{(k)} \), has larger maximum SV for smaller \( k \). In Glorot Initialization, all weights \( w^{(k)} \) are initialized as random Gaussian (normal) distributions with variance \( \sigma^2_{(k)} = 2/d^{(k-1)} \). This sets all SVs of \( w^{(k)} \) to \( \sqrt{2} \) as we discuss below (see also [58]).

5.4.1. Singular Values of weights during SGD

To quantify the effect of \( A^{(k+1)} \), we must examine its singular values and find out how they evolve with training steps, i.e. increasing \( N \). Define \( \tilde{w}^{(k)} = \text{diag} \left( \theta \left( \tilde{h}^{(k-1)} \right) \right) w^{(k)} \). At step zero, weights and biases of all layers are random and so we can assume that half of the entries of \( \theta \left( \tilde{h}^{(k-1)} \right) \) will be zero, the other half one. Thus, in \( \tilde{w}^{(k)} \) half of the rows of \( w^{(k)} \)

\(^4\)The exact architecture is: Maxpool (3,3), Dense(30), Dense(100), Dense(30), Classification(10). The dense layers have about 3000 trainable parameters, while the classification layer has 310.
will be replaced by zeros. Consider the symmetric positive semi-definite \( d^{(k)} \times d^{(k)} \) matrix 
\[ M^{(k)} \equiv \tilde{w}^{(k)T} \tilde{w}^{(k)} \]. The eigenvalues of \( M^{(k)} \) are squares of singular values (SV) of \( \tilde{w}^{(k)} \). The \( w^{(k)} \) are generally initialized randomly with zero mean. As columns of \( w^{(k)} \) are uncorrelated at the start, \( M \) will be approximately diagonal and all diagonal entries will be similar because

\[
M^{(k)}_{ab} = \sum_c \tilde{w}^{(k)}_{ca} \tilde{w}^{(k)}_{cb} \approx \frac{d^{(k-1)} \sigma^2_{(k)a}}{2} \delta_{ab}, \tag{5.8}
\]

where \( \sigma^2_{(k)a} = \sum_b \left( w^{(k)}_{ba} \right)^2 / d^{(k-1)} \) is the variance of the weights over the input dimension and the \( d^{(k-1)}/2 \) is because of \( \theta \left( \tilde{h}^{(k-1)} \right) \) eliminating half of the rows. The variance of all rows is chosen to be the same \( \sigma^2_{(k)a} = \sigma^2_{(k)} \) initially. Eq. (5.8) implies that most eigenvalues of \( M^{(k)} \) are initially close to \( d^{(k-1)} \sigma^2_{(k)} / 2 \). Now consider \( B^{(k)} \equiv A^{(k)T} A^{(k)} \) whose eigenvalues are squares of SV of \( A^{(k)} \). Using (5.8) we can progressively simplify \( B^{(k)} \) and get

\[
B^{(k)} \approx \frac{d^{(k-1)} \sigma^2_{(k)}}{2} A^{(k+1)T} A^{(k+1)} \approx \prod_{m=k}^{n} \frac{d^{(m-1)} \sigma^2_{(m)}}{2} I \tag{5.9}
\]

where \( I \) is the \( C \times C \) identity matrix. Experiments have shown [66] that with ReLU, the Glorot initialization [58] (a variant of the Xavier method) which sets \( \sigma^2_{(k)x} = 2 / d^{(k-1)} \) yields better performance than other commonly used initialization methods. Glorot initialization was designed specifically to set the maximum SV of all \( w^{(k)} \) around one to avoid explosion of gradients, while choosing a smaller initialization makes the gradients too small and leads to worse results. However, this initialization does not guarantee that the maximum SV will remain below one. In fact, we argue that the reason this choice works better than a smaller initialization is precisely because the SV become larger than one early in SGD.
Figure 5.3. **Singular Values and Growth of Gradients for Layers:** Using MNIST and a network with four fully connected layers (3 ReLU, one softmax classification, all using Glorot initialization). **A)** Although the initialization starts the networks with all singular values (SV) of all weights $w^{(m)}$ smaller than 2, the maximum SV quickly grows above 2. **B)** The gradient for weights of layer $k$ is proportional to the product $\prod_{m=k}^{n} w^{(k)}$ of weights of all layers above it. As we predicted, the largest SV of the product of weights is greater for lower layers, thus resulting in faster dynamics of lower layers.

Now consider the combination $\tilde{w}^{(k)} \equiv \text{diag}\left(\theta\left(\tilde{h}^{(k-1)}\right)\right) w^{(k)}$ which appears in $A^{(k)} = \prod_{m=k}^{n} \tilde{w}^{(m)}$. Because the initialization is random Gaussian, the rows of the raw outputs $\tilde{h}^{(k)}$ are equally likely to be positive and negative, meaning half of the rows of $\theta\left(\tilde{h}^{(k-1)}\right)$ are zero. Therefore, the SVs of $\tilde{w}^{(k)}$ are initialized to 1 and so all SVs of $A^{(k)}$ are also 1 initially (5.8). The SVs of $A^{(k)}$ are the square-root of eigenvalues of $B^{(k)} \equiv A^{(k)^{T}} A^{(k)}$. In early steps, rows of $w^{(k)}$ are uncorrelated and so $B^{(k)} \approx 2^{k-n} \prod_{m=k}^{n} \|w^{(m)}\|^2 \mathbf{I}$ Eq. (5.9).

As discussed earlier, because of standard error, which for $w^{(k)}$ at step $N$ becomes $\sigma_{(k)}/\sqrt{N}$, SGD cannot resolve sharp minima in early stages (small $N$) and the dynamics effectively occurs on a smoothened landscape (Fig 5.1) of $g[w, b]$ with local minima that are wider and far less abundant than in $g[w, b]$ without this smoothing (i.e. at larger $N$). In early
The square of mean times the input dimensions, which becomes \((\sum_i w_{ij}^{(k)})^2 / d^{(k-1)}\), clearly shows that immediately after the training starts, the mean moves significantly away from zero. This confirms our claim and explains why the SV’s grow larger than 2.

stages, \(N \sim O(1)\) and, therefore, the smoothened local minima have width \(\Delta w^{(k)} \sim O(\sigma_k)\).

The exact location of all local minima is a set of measure zero in the space of \(w^{(k)}, b^{(k)}\) and therefore the initialization has vanishing probability to be precisely on a local minimum, where the gradient would vanish. Therefore, in early stages, the gradient is non-vanishing and is in the direction of the closest local minimum. Thus, the gradient has a consistent direction, in contrast to stochastic fluctuations, and maintains a consistent direction in the \(w^{(k)}\) space. The gradient will, therefore, move the mean \(\bar{w}_a^{(k)} = \frac{1}{d^{(k-1)}} \sum_b w_{ba}^{(k)}\) of columns\(^5\) the away from zero to \(\bar{w}_a^{(k)} \sim O(\sigma_k)\). The largest SV of \(\bar{w}^{(k)}\) is the square-root of the largest eigenvalue of \(\bar{w}^{(k)T} \bar{w}^{(k)}\) which in early stages with \(\sigma_k^2 \approx 2/d^{(k-1)}\) yields

\(^5w_{ba}^{(k)}\) is entry \(a, b\) of the weight matrix \(w^{(k)}\) and \(\bar{w}_a^{(k)}\) is the mean of column \(a\) over the rows
In other words, while the Glorot initialization is exactly tuned to set the initial SVs of $\tilde{w}^{(k)}$ equal to one, it also guarantees that the SVs become larger than one in early stages of SGD. We will argue that this is essential in allowing SGD to solve layers one by one.

The immediate consequence of this is that $||A^{(k+1)}||$ will be larger for smaller $k$, as it contains $n - k$ factors of weights, each with some SVs larger than one.

Because of the multiplication of weights in $A^{(k)}$, the gradients of biases in Eq. (5.4) are much larger for lower layers. The weight gradients Eq. (5.4) also contain another factor $h^{(k-1)}$, which we will discuss next.

### 5.4.2. Magnitude of $h^{(k-1)}$

Note that we cannot yet conclude that the norm of the gradient of weights is larger for lower layers because of the $h^{(k-1)}$ factor in Eq. (5.4). One may argue that $h^{(k)}$ contains the product of the weights of the first $k$ layers and therefore the gradients of $w^{(k)}$ for all $k$ are of the same magnitude.

But it is easy to see that $h^{(k)}$ cannot remain $h^{(k)} \sim \prod_{m=1}^{k} w^{(m)} h^{(0)}$. To see this, we examine the change in $\delta h^{(k)}$ after processing a $\delta N$ minibatch during SGD. Using Eqs. (5.3)–(5.4), we have

$$
\frac{\delta h^{(k)}}{\delta N} = \frac{\delta w^{(k)^T}}{\delta N} h^{(k-1)} + \frac{\delta b^{(k)}}{\delta N} + \frac{w^{(k)^T} \delta h^{(k-1)}}{\delta N}
$$

$$
= \left[ \left(1 + \|h^{(k-1)}\|^2\right) \frac{\delta b^{(k)}}{\delta N} + w^{(k)^T} \frac{\delta h^{(k-1)}}{\delta N} \right]
$$

(5.11)
Figure 5.5. **Comparison of magnitude of output $h^{(k)}$ with linear output $\prod_{m=1}^{k} w^{(m)} h^{(0)}$** (Left) Average output of the first layer of a ConvNet with 32 filters and ReLU divided by norm of $3 \times 3$ filter weights, divided by norm of input image in each $3 \times 3$ block. The plot shows this ratio for different training steps. The solid line is mean over all filters and the shaded area is one sigma above and below the mean (a small amount of random noise was added to make norms nonzero everywhere). The output is much smaller than the magnitude of the weights, which suggests that the bias and ReLU diluted the effect of the weights significantly. (Right) For the fully connected network used in 5.6 the ratio of the actual layer output to the linear approximation is not that striking and may be as high as 0.8 for some layers, but it may also be as low as 0.2.

we can use the fact that $h \partial_h \theta(h) = h \delta(h) = 0$ to use $\tilde{w}^{(k)} = \text{diag} \left( \theta \left( h^{(k-1)} \right) \right) w^{(k)}$ and write

$$w^{(k)} \frac{\delta h^{(k-1)}}{\delta N} = \tilde{w}^{(k)} \frac{\delta \tilde{h}^{(k-1)}}{\delta N}.$$
This yields

\[ W^{(m)} \equiv \prod_{p=k-m}^{k} \tilde{w}^{(p)} \]

\[
\delta \tilde{h}^{(k)} = \sum_{m=0}^{k-1} \left( 1 + \| h^{(k-1-m)} \|^2 \right) W^{(m)^T} \delta b^{(k-m)} \delta N
\]

\[
= -\varepsilon \sum_{m=0}^{k-1} \left( 1 + \| h^{(k-1-m)} \|^2 \right) W^{(m)^T} A^{(k-m+1)} \frac{\partial g}{\partial \tilde{h}^{(n)}}
\]

\[
= -\varepsilon \sum_{m=0}^{k-1} \left( 1 + \| h^{(k-1-m)} \|^2 \right) W^{(m)^T} W^{(m)} A^{(k+1)} \frac{\partial g}{\partial \tilde{h}^{(n)}}
\] (5.12)

Fig. 5.5 shows the magnitude of \( h^{(1)} \) divided by the magnitude of the weights \( w^{(1)} \) and the input \( h^{(0)} \) in a network consisting of a ConvNet with 3 x 3 kernels followed by a dense classification layer being trained on MNIST. As we see the magnitude of the output does not become comparable to \( \| w^{(1)} \| \| h^{(0)} \| \) which itself is larger than \( \| w^{(1)} h^{(0)} \| \) and is, in fact, orders of magnitude smaller. Thus, SGD will change \( \tilde{h}^{(k)} \) by terms proportional to \( A^{(k+1)} \). Therefore \( h^{(k-1)} \) in Eq. (5.4) will not have a well-defined factor of weights and, in particular, \( h^{(k)} \) cannot consistently remain close to \( \prod_{m=1}^{k} w^{(m)} h^{(0)} \).

In conclusion, the only factor of weights that is guaranteed to be present in the evolution of weights (and biases) is the \( A^{(k+1)} \) in Eq. (5.4). As we showed above, \( A^{(k)} \) has larger maximum SV for smaller \( k \), resulting in lower layers having gradients which are larger than the layer above them by a factor larger than one. The significance of this is that it signals the existence of a “separation of time scales”: the dynamics of lower layers is much faster than the layers above them. This is a point worth deliberating because it means that the dynamics of layers approximately decouple and SGD in layer \( k \) may effectively ignore dynamics of layer
\( k + 1 \), allowing SGD to solve lower layers without much disruption from higher layers. We will exploit this observation to solve the distribution of weights analytically.

In the end, note that both ReLU and the initialization of weights played a role in this result. Sigmoid or tanh activation would not have resulted in growing SVs and the time scale of evolution of all layers would have been similar. Additionally, the Glorot initialization seems to be the only initialization for ReLU which both makes sure the output of the layer is not exploding initially, while also being guaranteed to lead to SVs larger than one.

### 5.5. Conditions for Decoupling of Layer Dynamics during SGD

As discussed earlier, training a deep feedforward neural network is nontrivial partly because in Eqs. (5.3) and (5.4) all weights from layers above \( k \) are present, resulting in strongly coupled nonlinear differential equations. The factors in Eq. (5.4) that change from one layer to the next are \( h^{(k-1)} \) and \( A^{(k+1)} \). Our goal is to find out whether the magnitude of the gradients \( \| \delta w^{(k)}/\delta N \| \) can differ considerably for different \( k \). We will now show that the choice of activation function and the initialization of weights can, indeed, lead to an effective decoupling of learning dynamics of lower layers.

#### 5.5.1. Effect of Initialization

Our goal is to find settings in which lower layers learn earlier, so that higher layers can combine the output of lower layers, possibly learning higher-level features. This requires the time scale of training lower layers to be much shorter than higher ones, which means we need gradients of lower layers to be steeper, i.e. \( \| \delta w^{(k)}/\delta N \| \gg \| \delta w^{(k+1)}/\delta N \| \). In Eq. (5.4) this would be achieved if \( \| A^{(k)} \| \gg \| A^{(k+1)} \| \), or if all singular values (SV) of \( \tilde{w}^{(k)} = \text{diag}(f')w^{(k)} \) are greater than one. We also don’t want the SV to be too large, to avoid overflow. With ReLU,
the Glorot initialization scheme [58], which limits the variance of weights so that the initial
singular values of initial weights are equal to two, seems to yield the best performance [58,66].
The effect of Glorot initialization is that at the beginning of the training all SV of $w^{(k)}$ are set
to $\sqrt{2}$, which leads to all SV of $\tilde{w}^{(k)}$ being equal to 1 (section 5.4.1). However, immediately
after the training starts, all SV of $\tilde{w}^{(k)}$ become larger than one (Fig. 5.3) and change to
$$\text{SV} [\tilde{w}^{(k)}] \approx \sqrt{\frac{d^{(k-1)}}{2} \bar{w}^{(k)2} + 1} > 1$$
(5.13)
where $\bar{w}^{(k)}$ is the mean of $w^{(k)}$ over its input dimension. The Glorot initialization starts from
mean zero normal distributions for $w^{(k)}$, but the mean is guaranteed to move away from zero
(Fig. 5.6) during training. The reason for this (section 5.4.1) is that in order for the mean to
stay precisely zero the starting state of the network must sit precisely on a local minimum
of the cost function. Since the set of local minima and saddle points of the cost function is
measure zero, it is impossible for a randomly initialized network to start exactly at a local
minimum. Therefore, the weights will drift along a persistent gradient direction toward the
nearest local minimum (section 5.4.1). Thus, the mean of the gradient cannot be zero in
the first few step, which in turn forces the mean of $w^{(k)}$ to become non-zero (section 5.4.1).
The mean will make all SV of $\tilde{w}^{(m)}$ larger than 1, resulting in $\|A^{(k)}\| > \|A^{(k+1)}\|$. Moreover,
this yields a positive feedback through Eq. (5.21) resulting in near exponential growth of
maximum SV of $w^{(k)}$ in early stages of the training (Fig. 5.3). during can quickly grow much
larger than $\|A^{(k+1)}\|$ (Fig. 5.3) and lead to lower layers getting trained at time scales much
shorter than higher layers. Note that, Glorot initialization is tuned precisely such that any
nonzero $\bar{w}^{(k)}$ results in SV that are greater than 1. If we initialize the weight with a smaller
variance the SV of $\tilde{w}^{(k)}$, and thus $A^{(k)}$, won’t be guaranteed to become larger than 1. Also,
Figure 5.6. **Singular Values and Growth of Gradients for Layers**: Using MNIST and a network with four fully connected layers (3 ReLU, one softmax classification, all using Glorot initialization). Evolution of the variance (A) and mean (B) (calculated over input dimensions) of weights. The square of mean times the input dimensions, which becomes \( \left( \sum_i w_{ij}^{(k)} \right)^2 / d^{(k-1)} \), clearly shows that immediately after the training starts, the mean moves significantly away from zero. This confirms our claim and explains why the SV’s grow larger than 2. (C) Although the initialization starts the networks with all singular values (SV) of all weights \( w^{(m)} \) smaller than 2, the maximum SV quickly grows above 2. (D) The gradient for weights of layer \( k \) is proportional to \( A^{(k)} \). As \( A^{(k)} \) is input-dependent, we plot the maximum SV of \( A^{(k)} \) evaluated over 500 MNIST images. As we predicted, the largest SV of the product of weights is greater for lower layers, thus resulting in faster dynamics of lower layers (The last layer has no layers above it, thus, no \( A^{(n)} \) exists).

Note that, tanh and sigmoid activation have \( f' \approx 0 \) for inputs that strongly activate the layer. This is undesirable, because it makes the entries of \( \tilde{w}^{(k)} \) small and diminishes the magnitude of the SV, meaning that SV of \( A^{(k)} \) won’t be guaranteed to become larger than one.

### 5.5.2. Effect of Layer Output

In Eq. (5.4), beside \( A^{(k+1)} \), the output of the previous layer \( h^{(k-1)} \) also depends on \( k \). If \( h^{(k)} \sim \prod_{m=1}^{k} w^{(m)} h^{(0)} \) gradients of all layers will have a total of \( n - 1 \) factors of \( w^{(m)} \) and thus \( \|\delta w^{(k)}/\delta N\| \) will be similar for all layers, meaning high and low-level features are trained at
Figure 5.7. Convergence of weights in different layers on 3 datasets. The thicker the lines, the higher the layer. yaxis shows the $L_2$ norm of difference between the weight $w(t)$ at step $t$ with the final weight at step $t = 100$. As expected, the first layer converges more rapidly in all three datasets. The second layer in MNIST is slightly slower than the dense layer, but it also has more parameters.

(For clarity, only the first 65 epochs are plotted. The architecture were all 1conv, maxpool, 1conv, maxpool, 1dense. The maxpooling are is over $2 \times 2$, and conv over $3 \times 3$ and had 32 convolutional filters).

the same time. However, writing out the dynamics of $h^{(k)}$ explicitly (section 5.4.2) reveals that, regardless of the initialization of $h^{(k)}$, during training it acquires contributions from weights of all layers in the form

$$\frac{\delta h^{(k)}}{\delta N} = -\varepsilon \sum_{m=0}^{k-1} \left( 1 + \|h^{(k-1-m)}\|^2 \right) W^{(m)} T W^{(m)} A^{(k+1)} \frac{\partial g}{\partial h^{(m)}} , \quad W^{(m)} \equiv \prod_{p=k-m}^{k} \tilde{w}^{(p)}$$

(5.14)

which shows that $h^{(k)}$ for all layers $k$ changes by a sum of terms with products of different number of $w^{(m)}$. There is no guarantee that $\|h^{(k)}\|$ would be comparable to $\left\| \prod_{m=1}^{k} w^{(m)} h^{(0)} \right\|$ and, in fact, a simple check on MNIST reveals that $\|h^{(k)}\|$ becomes smaller than this (Fig. 5.5). Although this drop may not be very dramatic (Fig. 5.5, right) the ratio $\|h^{(k)}\| / \left\| \prod_{m=1}^{k} w^{(m)} h^{(0)} \right\|$ varied from 0.8 to 0.2.

Thus, unlike $A^{(k+1)}$, $h^{(k)}$ does not determine the hierarchy of $\|\delta w^{(k)}/\delta N\|$ for different $k$ as strongly as $A^{(k+1)}$ and in a clear or consistent way.
In conclusion, in Eqs. (5.3) and (5.4) \( A^{(k+1)} \) is the only term that distinguishes the rate of evolution of parameters in different layers. \( A^{(k)} \) has larger SV for smaller \( k \), signalling the existence of a “separation of time scales”: the dynamics of lower layers is much faster than the layers above them. This means that the dynamics of layers approximately decouple and SGD in layer \( k \) may effectively ignore dynamics of layer \( k + 1 \), allowing SGD to solve lower layers without much disruption from higher layers. Thus, we have found a set of conditions that would allow low-level features to be learned earlier, which turned out to be just ReLU activation and Glorot initialization for hidden layers. Now we will derive what solutions the low-level features converge into.

5.6. Relaxation phase

For each layer, when the parameters are in the vicinity of a local minimum, the fast drift phase ends and the layer enters the stochastic relaxation phase. This phase sets in when weights are trained to a good degree and so the gradients \( \delta g / \delta b^{(k)} \) and \( \delta g / \delta w^{(k)} \) become very small. From Eq. (5.4), this means that \( \left\| \delta g / \partial \bar{h}^{(n)} \right\| \) must be small. Using Eq. (5.2), for input \( i \)

\[
\left\| \frac{\partial g}{\partial \bar{h}^{(n)}_i} \right\| = \frac{1}{N} \left\| h^{(n)}_i - y_i \right\| \ll 1 \quad (5.15)
\]

Since the gradient is very small, we can expand the exponential inside \( h^{(n)} \) as a Taylor series\(^6\). First, we make the following \( h^{(n)} \)-dependent variable redefinition

\[
\tilde{y}_i \equiv \log \left( y_i Z_i \right). \quad (5.16)
\]

\(^6\)Note, \( N \) can be large, but it’s always finite.
where we replace the zeros in \( y_i \) with a small positive \( \varepsilon \sim 1/Z_i \) to make \( \tilde{y}_i \) well-defined. Define the projection matrix onto class of \( y_i \) as
\[
P_i \equiv \text{diag}(y_i).
\]
Expanding Eq. (5.15), we have
\[
\frac{\partial g}{\partial \tilde{h}_i^{(n)}} = \frac{P_i}{N} \left( \frac{h_i^{(n)}}{y_i} - 1 \right) = \frac{P_i}{N} \left( \exp \left[ \tilde{h}_i^{(n)} - \tilde{y}_i \right] - 1 \right) \approx \frac{P_i}{N} \left( \tilde{h}_i^{(n)} - \tilde{y}_i \right).
\]
(5.17)

To further simplify this, we define the “optimal input” \( \bar{h}_i^{(0)} \) as the input that would produce exactly the output vector \( y_i \). In practice, \( \bar{h}_i^{(0)} \) can be constructed via activation maximization \([104]\). To be precise
\[
\bar{h}_i^{(n)} = F[\bar{h}_i^{(0)}], \quad \bar{y}_i = F[\bar{h}_i^{(0)}]
\]
(5.18)
where \( F[\cdot] \) summarizes forward propagation through the network. Note that \( \bar{h}_i^{(0)} \) is not unique for many reasons including nonlinearity of the network, as well as weights not being full-rank. Let \( \bar{h}_i^{(k)} \) denote the raw output of layer \( k \) after propagating \( \bar{h}_i^{(0)} \) through the network. We can always find \( \bar{h}_i^{(0)} \) such that the activation patterns of \( \bar{h}_i^{(k)} \) and \( \tilde{h}_i^{(k)} \) are similar, meaning \( \theta(\bar{h}_i^{(k)}) \sim \theta(\tilde{h}_i^{(k)}) \). This means that we choose \( \bar{h}_i^{(k)} \) that uses features similar to \( \tilde{h}_i^{(k)} \) in each layer. This ensures that \( A^{(k)} \) will be the same for \( \bar{h}_i^{(k)} \) and \( \tilde{h}_i^{(k)} \).\footnote{This is a reasonable assumption as we are close to convergence and we can always find an \( \bar{h}_i^{(k)} \) which is close enough to \( \tilde{h}_i^{(k)} \) so that this is satisfied.}
Defining \( \Delta h_i^{(k)} \equiv \text{diag}(\theta(\bar{h}_i^{(k)})) \left( \bar{h}_i^{(k)} - \tilde{h}_i^{(k)} \right) \), we get
\[
\bar{h}_i^{(n)} - \bar{y}_i = w^{(n)T} \Delta h_i^{(n-1)} \approx A^{(k)T} \Delta h_i^{(k-1)}, \quad \forall k\]
(5.19)
where bias-dependent terms exactly cancel. Plugging Eq. (5.19) into Eq. (5.17) and substituting in Eq. (5.4), the weight gradients become

\[
\frac{\delta g}{\delta w^{(k)}} \approx \frac{1}{N} \sum_{i=1}^{N} h_i^{(k-1)} \Delta h_i^{(k-1)} w^{(k)} K_i^{(k+1)},
\]

\[
K_i^{(k)} \equiv A^{(k)} P_i A^{(k)T}.
\] (5.20)

Note that, while \(h_N^{(0)}\) is the actual \(N\)-th input, \(\bar{h}_N^{(0)}\) is the best guess for what input would yield output \(y_N\) based on information in the \(N - 1\) previous data points. Since \(\bar{h}_i^{(k)}\) produces exactly the same redefined label \(\bar{y}_i\) for all \(i < N\), we have \(A^{(k+1)T} \bar{h}_i^{(k)} = A^{(k+1)T} \tilde{h}_i^{(k)}\), which results in the first \(N - 1\) points canceling in Eq. (5.20)

\[
\frac{\delta g}{\delta w^{(k)}} \approx \frac{1}{N} h_N^{(k-1)} \Delta h_N^{(k-1)} w^{(k)} K_N^{(k+1)}. \quad (5.21)
\]

which also makes use of the fact that in the relaxation phase the weights are close to their locally optimal values. Eq. (5.21) is a stochastic equation, but the distribution of its solutions can be calculated.

5.6.1. Estimating distribution of weights for low layers

\(K_i^{(m)}\) is similar to a projection onto label \(y_i\), which is a one-hot vector that is nonzero only for some class \(c\). But different inputs \(h_i^{(0)}\) belonging to the same class \(c\) may still have very different activation patterns \(\theta(\tilde{h}_i^{(k)})\) in different layers because they may contain different low and high level features. As a result, \(A_i^{(k)}\) can be different for inputs belonging to the same class. But it is also likely that there are groups in inputs for each class such that within each group the activation patterns are the same (i.e. all inputs in one group use approximately the same low and high level features) and so have similar \(A_i^{(k)}\). For such a group of inputs within
a class $c$, $K_i^{(k)}$ will be similar, allowing further simplification of Eq. (5.20). However, since we do not know a priori what groups of inputs use the same features, even though these groups exist, we lack the information needed to perform such grouping. So, in the most general case, there does not seem to be any option other than methods like SGD to solve Eq. (5.20).

Despite the above-mentioned problem, in many datasets with refined labels most inputs of the same class share characteristic features. For instance, in the MNIST dataset of handwritten digits with 10 labels all inputs of the class of number 1 contain vertical lines which are less common in other numbers. Therefore, the average of $K_i^{(k)}$ over class 1 will be a good approximation of the characteristic $K_i^{(k)}$ of this class, and it will be different from $K_i^{(k)}$ of other classes. Thus, if for most inputs within class $c$, we could ignore the variability of the activation pattern $\theta(\tilde{h}_i^{(k)})$, we could define a single, averaged $K_i^{(k)}$, denoted by $K_c^{(k)}$, for the whole class $c$. In Eq. (5.20), since all inputs of the same class $c$ have the same $K_c^{(k)}$, we can break the sum over $i$ down to summations over classes.

First, define the “density matrix”\footnote{When $h_i^{(k)}$ are mean zero, the density matrix is just the covariance matrix.} $\rho_c^{(k)}$ for each class $c$ as

$$\rho_c^{(k)}(N) \equiv \frac{1}{N} \sum_{i \in c} h_i^{(k)} h_i^{(k)T}. \quad (5.22)$$

In Eq. (5.21) $\tilde{h}_N^{(k)}$ is the guessed input based on the $N - \delta N$ previous inputs and it can be expressed as a linear combination\footnote{Note, $h_i^{(k)}$ may be overcomplete and not be linearly independent, but $\tilde{S}$ only needs to map onto a linearly independent subset of them.} of previous inputs which belonged to the same class $c$ as the last label $y_N$. The last input $h_N^{(k)}$, on the other hand, contains new information and cannot be an orthogonal transformation of the previous data. Using this, we can show
explicitly (Eq. (5.33)) that
\[
\frac{1}{N} h_{N}^{(k)} \Delta h_{N}^{(k)^T} \approx \frac{1}{2} \frac{\delta \rho_{c}^{(k)}}{\delta N}
\]  
(5.23)

And so using Eq. (5.21) the SGD equations for the weights become
\[
\frac{\delta w^{(k)}}{\delta N} \approx -\epsilon \sum_{c=1}^{C} \frac{\delta \rho_{c}^{(k-1)}}{\delta N} w^{(k)}(k+1). 
\]  
(5.24)

In the relaxation phase, \( \rho_{c}^{(k)} \) fluctuates mostly due to statistical fluctuations in the data. As every input is an independent drawing from the dataset, \( \rho_{c}^{(k)} \) is the sum of \( N \) observations \( h_{i}^{(k)} h_{i}^{(k)^T}/N \). Since mean and variance of the input do not diverge, the Central Limit Theorem implies that \( \rho_{c}^{(k)} \) will have a multivariate Gaussian distribution and it is straightforward to show that (Eq. (5.34)) the variance is
\[
\text{Var} \left[ \delta \rho_{c}^{(k)} \right] = 2N \rho_{c}^{(k)^2}/N^2
\]  
(5.25)

Therefore \( \delta \rho_{c}^{(k)}/\delta N \) is also a Gaussian with mean zero and the above variance and we can write \( \delta \rho_{c}^{(k)}/\delta N = N(0,1) \frac{2}{\sqrt{\delta N}} \rho_{c}^{(k)}/N \). This behavior can be verified directly from data (see Fig. 5.8 for MNIST).

### 5.6.2. Solving the weight distribution

Eq. (5.24) is still hard to solve because \( w^{(k)} \) is sandwiched between two other matrices. The presence of \( K_{c}^{(k+1)} \) is generally what makes SGD highly nonlinear, coupling weights of all layers. This is precisely where the separation of time scales for different \( k \) comes to our rescue.
Putting all the above together, the weight SGD equation becomes

\[
\frac{\delta w^{(k)}_c}{\delta N} = -\varepsilon(N) \sum_c \rho_c^{(k-1)} w^{(k)}_c K_c^{(k+1)}.
\] (5.26)

where \(\varepsilon(N) \equiv \frac{\varepsilon N(0,1)}{N\sqrt{\delta N}}\), is a random Gaussian noise with correlation function \(\langle \varepsilon(N) \varepsilon(N') \rangle = \frac{\varepsilon^2}{N^2 \delta N} \delta(N - N')\). Note that this is a stochastic, “Langevin equation” [38]. While there are no solutions to it, we can find the probability distribution for different solutions using the corresponding “Fokker-Planck equation” [78]. We do so by first moving the weights \(w^{(k)}_c\) to the right hand side. We notice that, since higher layers \(m > k\) evolve much more slowly than layer \(k\), when \(w^{(k)}_c\) in Eq. (5.26) has not fully converged, in \(A^{(k+1)}\) the \(w^{(m)}_c\) \((m > k)\) must be farther from convergence than \(w^{(k)}_c\), and must be still fairly random. Thus rows of \(w^{(m)}_c\) are uncorrelated, resulting in its transpose being approximately proportional to its pseudo-inverse and so \(A^{(k)}_T \approx a_k A^{(k)}_{(k+1)} \) with \(a_k \equiv ||A^{(k)}||^2/d(k-1)\). As a result, \(K_c^{(k)} / a_k\) is a projection matrix onto class \(c\).

Using SVD, we can define a right-pseudo-inverse \(\bar{w}^{(k)}_c^{-1}\) such that \(w^{(k)}_c \bar{w}^{(k)}_c^{-1} = I\). Define the “class-restricted weights,” \(w^{(k)}_c\).

It follows that

\[
w_c^{(k)} \equiv w^{(k)}_c \frac{K_c^{(k+1)}}{a_k}, \quad \bar{w}_c^{(k)} \approx \frac{K_c^{(k+1)}}{a_k} \bar{w}^{(k)}_c^{-1},
\] (5.27)

which satisfy \(w^{(k)}_c \bar{w}^{(k)}_c^{-1} \approx \delta c_c I\). Multiplying both sides of Eq. (5.26) by \(\bar{w}^{(k)}_c^{-1}\), we get

\[
\frac{\delta w^{(k)}_c}{\delta N} \bar{w}^{(k)}_c^{-1} \approx -\varepsilon(N) \rho^{(k-1)}_c.
\] (5.28)
Since gradients for lower layers are much larger than higher ones, $\delta w^{(k)} \gg \delta w^{(m)}$ for $m > k$, we can neglect $\delta (K_c^{(k+1)}/a_k)/\delta N$ terms and write\(^\text{10}\)

$$
\frac{\delta w^{(k)}}{\delta N} = \sum_c \frac{\delta w_c^{(k)}}{\delta N} \approx \sum_c \frac{\delta w^{(k)}}{\delta N} \frac{K_c^{(k+1)}}{a_k},
$$

$$
\frac{\delta w^{(k)}}{\delta N} \bar{w}_c^{(k)-1} \approx \frac{\delta w_c^{(k)}}{\delta N} \bar{w}_c^{(k)-1} \equiv \frac{\delta \log R w^{(k)}_c}{\delta N} \tag{5.29}
$$

where the “right logarithm,” is formally defined using the pseudo-inverse for the largest non-degenerate submatrix such that\(^\text{11}\) $\delta \log R w^{(k)}_c = (\delta w^{(k)}_c) \bar{w}_c^{(k)-1}$.

This suggests that Eq. (5.26) could be solved for the class-restricted weights separately, and that for each class we have a stochastic equation given by

$$
\frac{\delta \log R w^{(k)}_c}{\delta N} = -\varepsilon(N) \rho^{(k-1)}_c \tag{5.30}
$$

This is a Langevin equation with a multivariate noise distributed according to $\rho^{(k-1)}_c$ whose magnitude decreases with increasing $N$. Similar to a random walk, we can find the probability distribution of $w^{(k)}_c$.

The corresponding Fokker-Planck equation has solution (for simplicity $v \equiv \log R w^{(k)}_c$ )

$$
\Pi(v) = Z^{-1} \exp \left[ -\frac{1}{\langle \varepsilon(N)^2 \rangle} v^T \rho^{(k-1)-2} v \right] \tag{5.31}
$$

which is a multivariate Gaussian distribution with spread $\sigma = \rho^{(k-1)}/\sqrt{\langle \varepsilon(N)^2 \rangle}$ proportional to the class density matrix for inputs of layer each $k$. $w^{(k)}_c$ spreads along directions of

\(^\text{10}\)Also, note that because of the $K_c^{(k+1)}$ on the r.h.s. of Eq. (5.26) dimensions that do not get mapped through $K_c^{(k+1)}$ also do not appear in the SGD equations.

\(^\text{11}\)Note that $\log R w^{(k)}_c$ is a $d^{(k-1)} \times d^{(k-1)}$ dimensional matrix, but the its largest non-degenerate submatrix is one dimensional, because of the projection via $K_c^{(k+1)}$. 
eigenvectors $\psi_c^c$ of $\rho_c^{(k-1)}$, spreading wider along eigenvectors with larger eigenvalues. This means that for each class, just as in PCA, there is a subset of weights $w_c^{(k)}$ which are likely to be along principal components (PC) of $h_i^{(k)}$. This suggests that Eq. (5.30) can be approximately solved for each of the class-restricted weights $w_c^{(k)}$ separately using a class-based PCA. Since $wK = wSS^TK$ for any orthogonal $S$, we are free to choose the basis of the output of the weights for each layer (i.e. the choice of basis is part of the design of the network architecture). We will therefore choose distinct rows of $w^{(k)}$ to be dedicated to each class, trivially satisfying $w_c^{(k)}w_c^{(k)^T} \propto \delta_{cc'}$. Then, for each class the most likely outcomes of SGD will be those rows of $w_c^{(k)}$ which are PCs of $h_i^{(k)}$. If a PC of one class is highly correlated with a PC of another class, we can keep one of them. The biases $b^{(k)}$ canceled in Eq. (5.19) and so in the late stages of training they do not seem to play a significant role. We may choose $b^{(k)} = 0$. Thus, we can construct pretrained neural networks with weights found using the class-based PCA described here and with biases set to zero. We will call such a network a “Density Matrix Network” (DMN).

Note that the assumption that the projections $K_i^{(k)}$ can be averaged over one class relied on the assumption that different inputs in the same class use similar features. While this may be generally true of low-level features, it is more likely to fail for high-level features. For instance, all eyes may be broken down into the same low-level curves, but the eye shape, as a high-level feature, can have many varieties. Thus, images of eyes may use the same low-level features, but differ more noticeably in the way they use high-level features. For lower layers (smaller $k$) the similarity of features in lower layers may mask this problem and result in similar $K_i^{(k)}$ for most class inputs, but in higher layers the problem should become more evident. Because of this, we expect our result to perform worse in higher layers, unless
class inputs are further refined using auto-encoders or an information theoretic measure that can quantify these variations within each class.

5.7. Evolution of covariance and density matrix

In Eq. (5.21) $\bar{h}^{(k)}_N$ is the guessed input based on the $N - \delta N$ previous inputs and it can be expressed as a linear combination\(^{12}\) of previous inputs which belonged to the same class $c$ as $y_N$

$$\bar{h}^{(k)}_N = \sum_{i \in c} h_i^{(k)} \tilde{s}_i^T,$$

and consequently $\bar{h}^{(k)}_N \bar{h}^{(k)T}_N = \frac{1}{N} \rho_c^{(k)}$. The last input $h_N^{(k)}$, on the other hand, contains new information and it cannot be an orthogonal transformation on them. Thus, we write $h_N^{(k)} =\sum_{i \in c} h_i^{(k)} (S + \delta S)_i^T$ where $SS^T = I$, but $S + \delta S$ is not orthogonal. Since we have freedom in choosing $\bar{h}^{(k)}_N$, we can choose $\tilde{S} = S$ and so we have

$$\rho_c^{(k)}(N) = \frac{N - 1}{N} \rho_c^{(k)}(N - 1) + \frac{1}{N} h_N^{(k)} h_N^{(k)T}$$

$$= \rho_c^{(k)}(N - 1) + \frac{1}{N} \left( h_N^{(k)} \Delta h_N^{(k)T} + \Delta h_N^{(k)} h_N^{(k)T} \right) + O(\delta S^2) \quad (5.32)$$

Thus

$$\frac{1}{N} h_N^{(k)} \Delta h_N^{(k)T} \approx \frac{1}{2} \frac{\delta \rho_c^{(k)}}{\delta N} \quad (5.33)$$

5.7.1. Dynamics of Relaxation of the Density Matrix

In the relaxation phase, $\rho_c^{(k)}$ fluctuates mostly due to statistical fluctuations in the data. As every input is an independent drawing from the dataset, $\rho_c^{(k)}$ is the sum of $N$ observations $r_i = h_i^{(k)} h_i^{(k)T} / N$ of a random variable $R$. Thus, $\text{Var}[\delta \rho_c^{(k)}] = \delta N \text{Var}[R]$ as it contains $\delta N$

\(^{12}\)Note, $h_i^{(k)}$ may be overcomplete and not be linearly independent, but $\tilde{S}$ only needs to map onto a linearly independent subset of them.
samples. Using the Bienaymé formula [99] and the fact that \( R \) is quadratic in \( H = \{ h_i^{(k)} \} \), we have \( \text{Var}[R] = \frac{\text{Var}[H^2]}{N^2} \). Since mean and variance of the input do not diverge, Central Limit Theorem implies that \( \rho_{c}^{(k)} \) will have a multivariate Gaussian distribution. Using Gaussianity \( H \) we can directly calculate

\[
\text{Var}[H^2] = E[H^4] - E[H^2]^2 = 2\text{Var}[H]^2 = 2\rho_{c}^{(k)^2}
\]

\[
\Rightarrow \text{Var} [\delta\rho_{c}^{(k)}] = 2\delta N \frac{\rho_{c}^{(k)^2}}{N^2} \tag{5.34}
\]

Therefore, \( \delta\rho_{c}^{(k)}/\delta N \) is also a Gaussian with mean zero and the above variance and we can write \( \frac{\delta\rho_{c}^{(k)}}{\delta N} = N(0,1)\frac{2}{\sqrt{\delta N}}\rho_{c}^{(k)}/N \). Fig. 5.8 shows the fluctuations of eigenvalues of \( \rho^{(0)} \) for MNIST, where the input is broken into \( 5 \times 5 \) windows convolved over the images (i.e. input for a convolutional layer). \( \rho^{(0)} \) is 25 dimensional. When scaled by our prediction of the behavior Eq. (5.25) of the fluctuations, the distribution of the fluctuations of all 25 eigenvalues collapse to a single Gaussian with small error bars, confirming our prediction.

### 5.8. Training Layers Using Class-based PCA

Our results above state that for each class, just as in PCA, there is a subset of weights \( w_{c}^{(k)} \) which are likely to be along principal components (PC) of \( h_i^{(k)} \). Since \( wK = wSS^TK \) for any orthogonal \( S \), we are free to choose the basis of the output of the weights for each layer (i.e. the choice of basis is part of the design of the network architecture). We will therefore choose distinct rows of \( w^{(k)} \) to be dedicated to each class, trivially satisfying \( w_{c}^{(k)} w_{c}^{(k)T} \propto \delta_{cc'} \). Then, for each class the most likely outcomes of SGD will be those rows of \( w_{c}^{(k)} \) which are PCs of \( h_i^{(k)} \). If a PC of one class is highly correlated with a PC of another class, we can keep one of them. The biases \( b^{(k)} \) canceled in Eq. (5.20) and so in the late stages of training
Figure 5.8. MNIST showing \( \text{Var}\left[ \frac{\delta \rho}{\delta N} \right] \propto \frac{\rho^2}{N^2} \). The fluctuations in the eigenvalues \( \frac{\delta \lambda}{\delta N} \) of the covariance matrix takes a random Gaussian distribution with zero mean and constant variance over added samples \( N \) when scaled by \( N/\lambda \) (inset). Averaging this distribution over all eigenvalues confirms that they all have the same \( \lambda^2/N^2 \) variance pattern.

they do not seem to play a significant role. We may choose \( b^{(k)} = 0 \). Thus, we can construct pretrained neural networks with weights found using the class-based PCA described here and with biases set to zero. We will call such a network a “Density Matrix Network” (DMN). We will first discuss how such direct calculation of weights can reduce the training time complexity and then show results of simulations.

5.8.1. Disentangling the Weights using Pooling

For images, convolutional layers work because they exploit the translational symmetry, or the fact that features can be anywhere in the image. However, when doing PCA, we rely on the input data to have correlations only due to features intrinsic to the data. Overlapping
Figure 5.9. **DMN with one or two layers versus ConvNet**: The dashed horizontal line indicates the performance of a network consisting of a single fully connected classification layer, showing the baseline performance. In our simulations we are adding layers of DMN or ConvNet before this classification layer. The architectures for the DMN and the ConvNet are chosen to be exactly the same with the same number of filter and layers. The y-axis shows the validation accuracy (percentage) and the x-axis labels show the number of filter (“4” means a single convolutional layer with 4 filters; “4,15” means two layers, first with 4 and second with 15 filters). On MNIST (left) DMN (blue) performs slightly (about 1 percent accuracy) worse than ConvNet (orange) almost in all tested cases. Yet, the mere fact that a pre-computed network is performing comparable to to a trained network is noteworthy. On CIFAR10 (center), DMNs consistently outperform the ConvNets, with a very impressive margin of between 5 – 10% in the two layer setting. On CIFAR100 (right), the single layer DMN outperforms ConvNet, while in two layers the performance is equal. All experiments have 1 dense classification layer with softmax activation. Each DMN and ConvNet layer is ReLU activated and is followed by a $2 \times 2$ maxpooling layer. All filters in DMN and ConvNet have $3 \times 3$ receptive fields. In all simulations the correlation threshold to remove duplicate PC was $T_{PC} = 0.9$. The fraction of training data used for DMN was $T_{frac} = 0.3 – 0.5$ for MNIST, $T_{frac} = 0.7$ for CIFAR10, and $T_{frac} = 1.0$ for CIFAR100.

domains in convolution introduce spurious correlations, and hence spurious PCs, which become the most prominent PCs in PCA. This dramatically reduces the performance of PCA, as actual features intrinsic to the data become much less significant than the spurious ones. Thus, to correctly employ our result and use PCA to extract higher level features, the spurious correlations (“entanglements”) should be removed. While finding the optimal way to do so may be elaborate, a simple solution is to use maxpooling. This will only keep one
out of a few overlapping outputs and greatly reduce the entanglement. Therefore, we use maxpooling after every layer of DMN in all our simulations.

5.8.2. Convolutional DMN

DMN can be made with virtually any architecture, including convolutional. For the first layer, this is related to the Karhunen-Loeve transform [99] of images, where one breaks an image down into blocks and PCA is performed on the block images. However, in higher layers, since DMN relies on the covariance matrix of the input, overlapping receptive fields result in spurious PC’s not arising from covariance of input data (section 5.8.1). Therefore we need to disentangle the outputs of each layer to get rid of the spurious PC. One simple way to do so is to use a maxpooling layer right after a DMN. The architectures we used for our experiments consist of one or two convolutional layers with ReLU activation functions, each followed by a maxpooling layer and ending with a classification layer with softmax activation function. We run the experiments once with DMNs for the convolutional layers and once with regular ConvNet (More experiments with hybrids of DMN and ConvNet, as well Batch Normalization [73] are shown in section 5.9).

5.8.3. Simulations

Training deep neural networks (DNN) requires large amounts of training data and is very computationally expensive. Part of the reason for this is a lack a good mathematical understanding of solutions a DNN converges into. We have shown that it can be possible to derive optimal weights for low-lying layers of a DNN directly from data using a class-based PCA. This relies deeply on a “separation of time scales” which occurs in the highest performing setups for DNNs. We found that when the activation function is similar to ReLU
(unbounded from above and asymptotically linear) and the weights are initialized with a high enough variance (allowing the weights to acquire singular values larger than one) lower layers (closer to input) evolve much faster than layers above them. This effectively decouples the SGD dynamics of lower layer from layers above and so SGD will be able to find optimal weights layer-by-layer, starting from the closest to the input data. The solutions found by SGD in these cases will be a class-based PCA.

Based on this finding, we can construct pretrained layers using class-based PCA for the weights and setting biases to zero. We will call such a pretrained layer a “Density Matrix Network” (DMN). The class-based PCA consists of calculating the eigenvectors of the covariance matrix $\rho_c$ of input for each class $c$ and keeping the eigenvectors with the largest eigenvalues. The number of eigenvectors to keep depends on how much of the variance of the input data we wish to keep. Additionally, $\rho_c$ converges very quickly and using only a fraction of the training data (Fig. 5.10) used for SGD will yield DMNs with performance comparable to SGD in many datasets. We have implemented DMN and tested it on three image datasets.

In MNIST and CIFAR10 datasets we have 10 classes. Therefore in the baseline model (i.e. with no conv. layers), the network consists of a single dense layer with 10 perceptrons and softmax activation function. For CIFAR100 model, the dense layer has 100 perceptrons. Next, we make the network deeper by adding one or two DMN or Conv. layers. We test the effect of the number of filters on the performance of DMN and ConvNet.

The tested DMNs perform on par, or far superior to conventional networks with the same architecture. DMN is within one percent of the performance of a convolutional layers (ConvNet) on MNIST, and significantly superior to ConvNets on CIFAR10. A single layer DMN also performs much better than a similar ConvNet on CIFAR100, and a two layer DMN is equally good as a 2 layer ConvNet. The results are summarized in Fig. 5.9. Most of
Figure 5.10. Convergence of eigenvalues of $\rho$, which determine the likelihood of features (lower means more likely). Using MNIST, we construct the covariance matrix $\rho$ for different number of images ((10, 100, 1000, 10000) images). The images are broken down into $3 \times 3$ squares and flattened. The 9 eigenvalues of $\rho$ are sorted by value. As we see, the eigenvalues converge very quickly and a small subset of the dataset is enough for extracting these features.

Our tested DMNs were trained using a fraction (between 20-70% for MNIST and CIFAR10, and 100% for CIFAR100) of the training data used for the ConvNet.

Finally, note that in these tests DMNs are constructed as convolutional layers\textsuperscript{13}. We only have to note that overlapping receptive fields will result in spurious PCs and we need to disentangle the outputs to get rid of them. One simple way to do so is to use a maxpooling layer right after a DMN. The architectures we used for our experiments consist of one or two convolutional layers with ReLU activation functions, each followed by a maxpooling layer and ending with a classification layer with softmax activation function. We ran the

\textsuperscript{13}This is related to the Karhunen-Loeve transform of images, where one breaks an image down into blocks and PCA is performed on the block images.
experiments once with DMNs for the convolutional layers and once with regular convolutional layers (ConvNet).

We simulate different scenarios and define different architectures with one or two DMNs / convolutional (Conv.) layers and compare their classification accuracies. We use Keras [37] with Tensorflow [2] back-end for all implementations. To study the effect of DMNs and Conv. layers in the classification task, our architectures will include only one dense layer, which is the classification layer followed by a softmax activation layer. Aside from this dense layer, the basic building block of our architectures consists of a convolutional layer, ReLU activation layer and a maxpooling layer (for disentanglement). We use either one or two conv. layers. Then we replace these conv. layers with DMNs and measure how the performance changes (see Fig. 5.11 for a sketch of the architectures). We will denote a DMN layer with 15 filters by “d15”, a Conv. layer with 15 filters as “c15”, max pooling with pool size of 2 as “m”, and dense layer with 10 nodes as “de10”. For DMN and Conv. layers, the kernel size has been fixed to 3. In some experiments, we have added a batch normalization layer and we denote such layer by “BN”.

Additionally, we will compare the performance of DMN compared to a layer with the same architecture trained using SGD. We will also discuss the meaning and effect of the parameters of DMN. Since DMN is based on PCA, we need to choose a threshold $0 < T_v \leq 1$ for how much of the variance of the data we wish to explain. We choose one $T_v$ for each layer, using the same $T_v$ for all classes. Note that $T_v = 1$ means all eigenvectors of $\rho_c^{(k)}$ are kept, making DMN a linear transformation. This yields no improvement to the classification compared to when the layer is absent. On the other hand, very small $T_v$ decreases the performance. We find that there is generally an optimal $T_v$ yielding the best performance. $T_v$ determines the number of filters (or hidden neurons) in the layer and has an effect similar to layer size. It
Figure 5.11. An example of one and two layer networks used in the experiment with MNIST dataset. A sketch of the networks used in the experiments on MNIST, consisting of one (a) or two (b) convolutional layers with ReLU activation, each followed by a maxpooling layer and ending with a classification layer with softmax activation. In our experiments, we start from these architectures and then replace the convolutional layers with DMNs to compare their performances. The maxpooling layer is essential for the DMN as a disentangling step, making sure that the density matrix of the output does not contain spurious features arising from the overlap of the convolutional domains.

would be useful to have a direct, information theoretic measure to fix $T_v$ as it provides an informed way to tune hyperparameters of the architecture.

As we mentioned above, when some PC $\psi_c$ in class $c$ has overlaps significantly with a PC $\psi_{c'}$ of another class $c'$, we will keep either $\psi_{c'}$ or $\psi_c$, not both. To do so, we set a threshold $T_{PC}$ and if $\psi_{c'}^T \psi_c \geq T_{PC}$ we drop one of them. We have found that this exclusion of similar filters actually improves the performance. This may be because including similar PCs would make the repeated PC more prominent than it actually is in the data, reducing the performance of PCA in higher layers.

Lastly, DMN can be constructed with a fraction of the training data, since the density (and covariance) matrix can converge with a small amount of data. The fraction $T_{frac}$ of data used to make the DMN is another parameter whose effect we examined.
Figure 5.12. **Performance as a function of three DMN parameters:** All three parameters affect the performance of the DMN, as well as the size of the layer (or number of filters), and for all there seems to exist a nontrivial optimal value. The threshold of similarity of PC beyond which we drop one of the two similar PCS ($T_{PC}$, left) shows a prominent peak near 80% similarity. Notably, if we do not drop highly similar PC across classes (i.e. $T_{PC} \rightarrow 1$) the performance goes down. We suspect that keeping similar PC skews the frequency of PCs towards the similar PCs, thereby changing the statistics of the data and resulting in worse classification. As DMN is performing PCA on classes, the amount of variance explained by PCA for each class ($T_v$, middle) is another parameter, which also shows a peak around 98% when keeping the other two parameters fixed. As we see keeping all of the variance does *not* result in the best classification accuracy. The fraction of data used to train a DMN ($T_{frac}$, right) is an important factor in determining training time. For MNIST we see that for the first layer as low as 30-40% of the training data yields similar to using all of the data. A reasonable fraction may also be determined prior to training the classification layer by checking if the $\rho_c$ have converged, meaning additional data doesn’t change it’s PCs within a desired tolerance. This method can also be used in higher layers, potentially decreasing training times substantially.

### 5.9. Simple PCA layers

Even without the supervised PCA, regular PCA can yield fairly well performing first layers. In subsequent layers, however, regular PCA performs poorly, mainly because the selected filters are not conditioned on the labels and thus may emphasize features that do not help with the classification, but rather are prominent in the data.

To train a DMN, we simply need to find the eigenvectors of the density matrices $\rho_c$. We will test a naive version of DMN, assuming that $\rho_c$ is is roughly independent of $c$ in the first and second layer. This is equivalent to assuming that the low-level features contribute
Figure 5.13. The comparison of test classification accuracies obtained by one dmn layer network (blue) vs one Conv layer network (green) based on the number of filters applied to MNIST, CIFAR10 and CIFAR100 datasets. The baseline models with one classification layer have the accuracies 92.9, 40.04 and 15.96 for MNIST, CIFAR10 and CIFAR100, respectively.

with similar proportions to all label classes. This way, training a DMN is the same as doing PCA on the full $\rho = N^{-1}XX^T$. When the the dataset is randomly sampled, $\rho$ converges very quickly (Fig. 5.10). Thus, using only a fraction of the data can give us a very good estimate for training a DMN, which can reduce the training time significantly. Since a DMN is pretrained, the removal of one layer from backpropagation may also result in a boost in training time, especially in very deep networks.

To examine the applicability of DMN layers in neural network architectures for image classification, we performed a number of tests on three image datasets: MNIST [93], CIFAR10 [85] and CIFAR100 [85].

For the first set of the experiments, which have a single DMN or Conv. layer, With one DMN, we observe that more filters does not necessarily yield better accuracy (maximum possible filters = (# input channels) $\times$ (kernel size)$^2$) (Fig. 5.13). However, for a network
with one Conv. layer, the classification accuracy increases by the number of layers. The key point to note here is that, DMN with maximum possible filters becomes a linear layer, whereas having less filters means we are only keeping the most prominent PCs, discarding less prominent features. For every dataset, there exists an optimal number of filters that leads to best performance of a DMN (e.g. 6 for MNIST, 12 for CIFAR10, and 15 for CIFAR100). The results of this experiment are shown in Fig. 5.13. Comparing the blue “DMN” line with green “Conv” line, we observe that the performance of 1 DMN layer is comparable with 1 Conv. layer. For CIFAR10 dataset, we see that one DMN layer outperforms one Conv. layer. In conclusion, we have shown that we can replace a Conv. layer with a DMN layer without degrading the classification accuracy.

In the second set of experiments we use two-layer network with different settings: 1. two DMNs, 2. one DMN and one Conv. layer and 3. two Conv. layers. Fig. 5.14 reports the results of two layer networks applied to three datasets. For MNIST dataset, we can observe that adding the second DMN layer increases the classification accuracy slightly. The performance of networks with one DMN and one Conv. layer is very close to two Conv. layer networks. For CIFAR10 dataset, without using BN layer, the networks with two DMN layers have lower classification accuracy compared to one DMN layer network. However, when we add a BN layer, the classification accuracy increases slightly. The best performance achieves by networks with one DMN and one Conv. layer and the worst performance is obtained by two Conv. layer networks. For CIFAR100 dataset, adding the second DMN layer results in higher accuracy as can be observed from Fig. 5.14. The highest accuracy is achieved by a network composed of one DMN layer and one Conv. layer. The first layer provides a better base filters and the input data is transformed to a more meaningful feature space. Adding a Conv. layer, back-propagation and training the network increases the classification accuracy.
Figure 5.14. Test classification accuracies of different two layer networks applied to three datasets. The labels on the x-axis denote the architectures. All have 1 dense classification layer. “d” stands for our DMN, and “c” is a ConvNet. All have $3 \times 3$ receptive fields and after each layer is a $2 \times 2$ maxpool. “d15, c80” means 1DMN layer and one ConvNet with with 15 and 80 filters, respectively.

Here, we plot the effect of number of filters used in DMN layer in one layer DMN network on the classification accuracy and compare it with one Conv. layer network with the same number of filters.

5.10. Conclusion

We have shown that the presence of hierarchical structure in data can result in a separation time scales in the convergence of various layers. Lower layers, learning low-level features, will converge at earlier stages. Focusing on the time scale at which a particular layer $k$ is converging, while higher layers have not converged, we found that the layer is likely to converge to a class-based PCA, with some weights (i.e. units in a dense layer or the filters in a convolutional layer) learning PCs of each class. This observations allows us to train networks by performing PCA on inputs for each class, resulting in a fast and efficient training at least for lower layers. Using this method, we were able to produce pretrained convolutional layers for MNIST, CIFAR10 and CIFAR100 which perform on par or superior to convolutional layers trained via SGD. The hyperparameters of this method are related to the variance explained in the PCA and how informative and discriminative the weights are. Tuning hyperparameters
of a specific layer can also be done very efficiently, as the PCA needs to be performed only once. Other recent work [69] using information theory has also suggested that SVD and PCA can be used as universally optimal guesses for weights in deep neural networks, supporting our results.
CHAPTER 6

Conclusions

In this dissertation, we started with the problems in multi-sensory data. We introduced a probabilistic classification model based on Gaussian Process and Variational Bayesian Inference. We evaluated the performance of the developed algorithm by applying it to several datasets. We then proceeded to a multi-class image classification problem in astrophysics area. We applied several traditional and advanced machine learning techniques to tackle this problem. We developed multi-view deep neural networks. We also presented a probabilistic framework to combine the output of machine learning algorithm and labels of multiple citizens to finalize the output for a given test image. We studied two main problems in hyperspectral paintings which can be used for different application. First, we introduced a sparse model to identify the pigments in a hyperspectral dataset. Then, we applied tsne data reduction approach and clustering algorithms to find interesting regions inside the painting. The last problem that was studied in this area was nonlinear unmixing which was solved by combining the results of a deep learning framework which classified the constituent pigments and applying Kubelka-Munk nonlinear function. Improvement on the objective and subjective quality assessment has also been presented, showing the effectiveness of our developed algorithm. We also studied the applicability of DNNs to pigment unmixing problem. We showed that the unmixing results improves when the information of identified pigments is added to higher layers of the unmixing DNN. At the end, we showed that we can replace the trainable layers in a DNN with pre-trained layers where the parameters can be estimated...
based on PCA for each class. The developed method can decrease the time complexity significantly.
References


This dissertation was typeset with \TeX\ 2\varepsilon by the author.

\footnote{The macros used in formatting this dissertation are based on those written by Miguel A. Lerma, (Mathematics, Northwestern University) which have been further modified by Debjit Sinha (EECS, Northwestern University) to accommodate electronic dissertation formatting guidelines.}