Artificial Intelligence for Spectral Analysis: a Comprehensive Framework

Zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften (Dr.-Ing.)

von der KIT-Fakultät für
Elektrotechnik und Informationstechnik
des Karlsruher Instituts für Technologie (KIT)

genehmigte

DISSERTATION

von

M.Sc. Xiang Xie
geb. in Anhui, VR China

Tag der mündlichen Prüfung:	31.01.2023
Hauptreferent:	Prof. Dr. rer. nat. Wilhelm Stork
Korreferent:	Prof. Dr. Michael Beigl

Abstract

Spectral analysis is widely utilized in various academic as well as industrial domains to extract relevant element information. Qualitative analysis requires accurate identification of existing elements, whereas in quantitative analysis, the concentrations of all relevant elements should be determined precisely. Although current state-of-the-art approaches deployed in commercial products can achieve excellent results on element quantification tasks, they still face some major limitations: high computational time especially for complex tasks; labor-intensive manual element identification procedure; expensive device calibration costs.

In this dissertation, a comprehensive neural-network-based framework for large-scale spectral analysis is proposed. To set up a new and proper baseline covering the most common elements (up to 28), extensive experiments are conducted to examine the required training data size, select feasible network architecture and discuss problem-specific configurations. On quantification tasks, compared to classic methods, the proposed approach achieves the same level of error rate with a significant speed-up of over 400x. Similarly, for the qualitative analysis, the classification of elements is automated with excellent accuracy of over 99% on real measurements, where the input data dimension is greatly reduced in an explainable way.

Besides, since neural networks usually require huge computational as well as storage resources, the application may face latency, memory footprint, storage size and power consumption issues, especially on low-power edge devices. To address this real-world problem, a hybrid approach is developed to optimize and accelerate the neural network execution while consistently preserving the final performance. Results on various target hardware platforms show that this hybrid

approach can achieve up to 52x mode size compression and 600x speedup with even better performance in most cases, which enables low-lost deployment on edge devices.

Finally, to overcome the last hurdle of the calibration problem towards large-scale deployment on a vast number of devices in industry, a meta-learning-based approach is proposed to achieve excellent calibration results at minimal cost by learning to calibrate. The general spectral analysis problem is formulated as a multi-device multi-configuration task and it achieves the best pre-and post calibration error rate across different unknown devices. Besides, compared to the baseline approaches with calibration, it performs on par even without calibration, which is practical in the real-world scenario where an unknown device needs to be deployed without reference samples available for calibration. Moreover, the resource analysis indicates that the approach requires significantly less expenditure to deploy large-scale devices in industry, which contributes to a huge saving and growth potential.

Zusammenfassung

Die Spektralanalyse wird in diversen akademischen und industriellen Bereichen eingesetzt, um relevante Elementinformationen zu extrahieren. Bei der qualitativen Analyse ist eine genaue Identifizierung der vorhandenen Elemente erforderlich, und bei der quantitativen Analyse, die Konzentrationen aller relevanten Elemente müssen präzis bestimmt werden. Obwohl die aktuellen kommerziellen Ansätze hervorragende Ergebnisse bei der Elementquantifizierung liefern können, stoßen sie immer noch an ihre Grenzen: hohe Rechenzeit (insbesondere bei komplexen Aufgaben), personalintensive manuelle Elementidentifizierung und erhebliche Kosten für die Gerätekalibrierung.

In dieser Dissertation wird ein umfassendes, auf neuronalen Netzen basierendes System für die Spektralanalyse in großem Maßstab entworfen. Um eine neue und angemessene Baseline zu erstellen, wobei die meisten gängigen Elemente (bis zu 28) abdeckt werden können, werden umfangreiche Experimente durchgeführt, um die erforderliche Trainingsdatengröße zu untersuchen, geeignete Netzwerkarchitekturen auszuwählen und problemspezifische Konfigurationen zu analysieren. Bei den Quantifizierungsaufgaben erreicht der vorgestellte Ansatz im Vergleich zu den klassischen Methoden die gleiche Fehlerquote mit einer signifikanten Geschwindigkeitssteigerung um einen Faktor von über 400. Auch für die qualitative Analyse wird die Klassifizierung von Elementen mit einer ausgezeichneten Genauigkeit von über 99% bei realen Messungen automatisiert, wobei die Dimension der Eingabedaten auf einer interpretierbaren Weise stark reduziert wird.

Darüber hinaus erfordern neuronale Netze in der Regel große Rechen- und Speicherressourcen, so dass die Anwendung mit Problemen in Bezug auf Latenzzeiten, Speicherplatzbedarf und Stromverbrauch konfrontiert sein kann, insbesondere

bei Endgeräten mit geringer Leistung. Um dieses Problem zu lösen, wurde ein hybrider Ansatz entwickelt, der die Ausführung neuronaler Netze optimiert, beschleunigt und dennoch die endgültige Leistung beibehält. Die Ergebnisse auf verschiedenen Zielhardwareplattformen zeigen, dass dieser hybride Ansatz in den meisten Fällen eine bis zu 52-fache Komprimierung der Modellgröße und eine 600-fache Beschleunigung mit sogar besserer Performanz erreichen kann, was den Einsatz auf Edge-Geräten mit geringen Kosten ermöglicht.

Um schließlich die letzte Hürde des Kalibrierungsproblems auf dem Weg zu einem großflächigen Einsatz auf einer großen Anzahl von Geräten in der Industrie zu überwinden, wird ein auf Meta-Learning basierender Ansatz entwickelt, um hervorragende Kalibrierungsergebnisse mit minimalen Kosten zu erreichen, indem die neuronale Netze lernen zu kalibrieren. Das allgemeine Spektralanalyseproblem wird als Multi-Geräte-Multi-Konfigurationsaufgabe formuliert und es erreicht die beste Fehlerrate vor und nach der Kalibrierung bei verschiedenen unbekannten Geräten. Im Vergleich zu den Basisansätzen mit Kalibrierung, schneidet es auch ohne Kalibrierung gleich gut ab, was in einem realen Szenario sehr praktisch ist, wo ein unbekanntes Gerät ohne verfügbare Referenzproben für die Kalibrierung eingesetzt werden muss. Darüber hinaus zeigt die Ressourcenanalyse, dass der Ansatz deutlich weniger Ressourcen für den industriellen Einsatz erfordert, was zu einem enormen Einsparungs- und Wachstumspotenzial beiträgt.

Acknowledgements

This work is done during the time I worked as research assistant as well as Ph.D. student at the Institute for Information Processing Technologies (ITIV) of the Karlsruhe Institute of Technology (KIT). It was a wonderful journey through my last few years, even during the unusual pandemic time.

Prof. Dr. rer. nat. Wilhelm Stork, or Willy as we usually call, provides me the great opportunity of doing Ph.D. study in the area which I truly love. Under his thoughtful supervision, I was able to make great progress and explore the broad academic world without any limitations. In the first place, I would like to thank Willy for his guidance, encouragement and inspiration. Besides, I also want to thank Prof. Dr. Michael Beigl for kindly accepting to be the second reviewer of my dissertation. Moreover, a special thank belongs to the Helmut-Fischer-Stiftung for the financial support.

Furthermore, I want to thank my colleges for their unconditional help, generous support and precious discussions. I would like to thank Lars, Malte, Olli and Rainer for helping me understand the physics more deeply and reviewing my manuscript. I also want to thanks Anqi, Con, Marius, Nana, Simon and Toni for their suggestions and inspirations during my Ph.D. study. Besides, I want to thank the students I supervised for the great teamwork and cooperation.

Finally, I want to give a big thanks to my family in China for being with me all the time, even at such a far distance. Spiritually, I am not alone during the long long journey.

Table of Contents

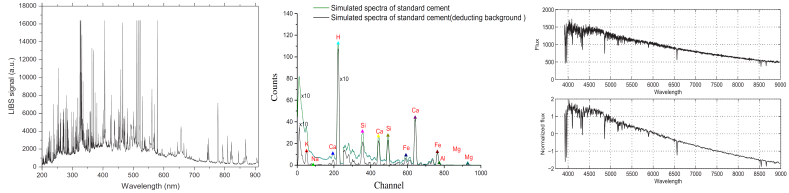
1 Introduction

This chapter briefly introduces the current status, problems and challenges in the domain of spectral analysis. Besides, it gives the motivation and the overview of this dissertation to set up a comprehensive framework based on artificial intelligence.

1.1 Overview of spectral analysis

In general, spectral analysis refers to the analysis of the properties of a spectrum measured with a spectrometer. Since the physical characteristics of different samples are diverse and distinct, various spectrometers are needed to meet the measurement objectives and eventually get the feasible spectra. These spectrometers usually vary from the light sources, the optical systems and the detectors. However, their principles are similar and can thus be unified to a certain extent, which will be further discussed in Chapter 2. Representative spectrometers are e.g. fluorescence spectroscopy [1, 2], gamma-ray spectrometer [3] and laser-induced breakdown spectroscopy [4,5] to name a few. In addition, in Fig. 1.1, the spectra of various samples measured with different spectrometers are illustrated. It demonstrates that across the spectral analysis domains, the spectra, which are one-dimensional data, share similar properties, even though the samples and the measurement systems differ a lot.

Due to the broad data source of spectra, spectral analysis is therefore by nature a widely utilized analytical technology in various academic as well as industrial domains, where the information of relevant elements needs to be extracted and



(a) A bronze alloy measured by laser-induced breakdown spectroscopy [6]

(b) A cement sample measured with prompt gamma neutron activation analysis [7]

(c) Stellar spectrum measured by LAMOST telescope [8], the bottom one is pre-processed.

Figure 1.1: Examples of spectra of various samples measured on different spectrometers

processed. Commonly, the scope of work can be divided into two subgroups: qualitative and quantitative analysis. Qualitative analysis usually refers to the identification of present elements given a measured sample spectrum, whereas in quantitative analysis, the concentration of each element needs to be determined precisely.

An example of pure qualitative analysis is the stellar spectral classification, where the stars need to be classified into different classes. In the meanwhile, drinking water analysis might prefer a quantitative result to examine whether all the indicators are within the given thresholds. As for many applications in the real world, both analysis tasks are often conducted together to obtain a more comprehensive outcome. To demonstrate the need for spectral analysis in different communities, a short list of related applications is shown as follows:

- trace biometal analysis [1, 4, 9]
- metal element analysis [2, 3, 5–7, 10–12]
- drinking water analysis [13, 14]
- blood serum analysis for cancer diagnosis [15]
- soil and rock quality assessment [16–18]
- gas analysis [19]
- bone properties analysis [20]
- stellar spectral classification [8, 21]

- isotope identification [22]
- solar radio spectrum classification [23]
- mineral classification [24]

It is clear, not only from the literature mentioned above but also from the observation in the industrial sections, that metal element analysis is quite an active application field. In addition, it also covers a wide range of task settings, where very complex samples with a high number of elements are within the scope of work. Therefore, this dissertation focuses on the unified task formulation based on metal element analysis, aiming to provide a generalized framework for most spectral analysis applications, which are more domain-specific with yet a narrower scope of work.

1.2 Current problems and limitations

In the first place, the preliminary input of spectral analysis is the captured spectra of high quality, which can be improved with the appropriate choice of spectrometers and proper measurement conditions. However, when it comes to the analysis part, difficulties appear in both sub-task domains.

1.2.1 Problems in qualitative analysis

In qualitative spectral analysis, the element composition should be determined. However, current state-of-the-art approaches deployed in the industry (i.e. the analysis tools in the commercial products) do not provide a one-click solution for element classification. It is required that the elements need to be identified manually by an expert who has domain-specific knowledge, therefore, it is labor-intensive, costly and not error-free.

Besides, since qualitative analysis is usually a preliminary step for the following quantitative analysis, it may lead to unexpected biased results if, e.g., the element classification is not correct. In such cases, the quantitative analysis will fail

to obtain the results on these elements that are not identified in the first step. Moreover, if the qualitative result is not available, which often occurs in the real world when no experts are on site, the results of quantitative analysis are also negatively affected due to their own limitations.

1.2.2 Problems in quantitative analysis

As for quantitative analysis, limitations are also obvious. The construction of spectra can be, in general, precisely formulated by a complex physical model (e.g., the use of fundamental methods [25] and Monte Carlo simulations [26]), given all relevant parameters known. An abstract formulation of the model can be described by Eq. 1.1,

$$\mathbf{S} = P(\mathbf{\Phi}) \tag{1.1}$$

where the spectrum $\mathbf{S}$ is a nonlinear function $P(.)$ of $\mathbf{\Phi}$, and $\mathbf{\Phi}$ denote all the variables that are to be determined, i.e., the concentration of all relevant elements and the thickness of the layers (if applicable). The ultimate goal of quantitative analysis is thus to determine $\mathbf{\Phi}$ from a measurement spectrum given specific measurement conditions with a known device, for which the inverse function $P^{-1}(\mathbf{S})$, denoted in Eq. 1.2, is needed.

$$\mathbf{\Phi} = P^{-1}(\mathbf{S}) \tag{1.2}$$

However, an analytical solution to Eq. 1.2 does not exist and it is not possible to retrieve the relevant parameters directly from a spectrum. To solve such problems, classic methods try to adopt certain optimization methods and utilize an iterative solver to fit a feasible set of $\mathbf{\Phi}^*$ as the final solution. The process starts with a randomized set of parameters $\mathbf{\Phi}^0$, and the physical model $P(.)$ will generate a simulated spectrum with respect to $\mathbf{\Phi}^0$. The difference between the simulation and the measurement will be fed into the iterative solver, which determines the

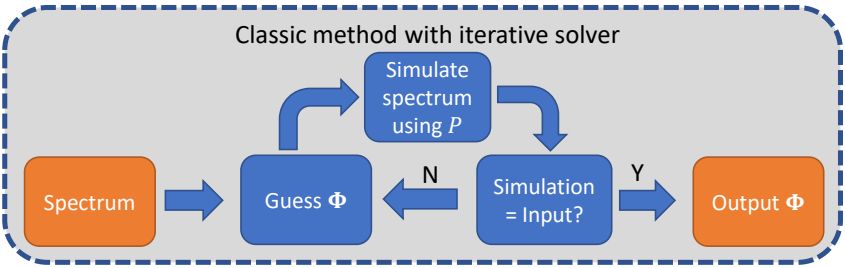


Figure 1.2: The workflow of classic method

update of Φ for the next iteration. The whole process stops when it converges to a stable stage or meets the stop criteria. This procedure is illustrated in Fig. 1.2 for a better understanding. Although this approach can obtain a very accurate result and has been successfully deployed in the industry, it still faces the following limitations:

- Since there is no closed-form solution and an optimization process is involved, this approach can not obtain a deterministic result. Instead, local optima with certain error rates are expected.
- The processing time is high and not stable, due to the fact that a varying number of iterations are required for the iterative solver to converge.
- The performance of this approach depends on the task complexity. Although for simple tasks with only several elements involved, it can get an accurate result within a reasonable time, as the number of elements increases, it faces the curse of dimensionality and the search space may get too large for the solver to converge to a local optimum. Therefore, the computational time of the method increases exponentially and it eventually fails.
- Besides, to obtain a valid result, the classic method requires further information regarding the measurement condition, e.g., the measurement voltage, the type of filter in the device, etc. Therefore, it is not possible to analyze

a sample with unknown measurement conditions, which is a noticeable disadvantage in the real world.

1.2.3 Additional challenges in spectral analysis

The problems and limitations discussed above hold implicitly the assumption that the spectral analysis is conducted on a single measurement device that never degrades. However, in the real world, especially in industrial applications, typically numerous devices are to be deployed in different environments, which introduces two additional factors: calibration and re-calibration.

The physical characteristics of the measurement differ a lot if they are built in different product lines, mainly because the components of the devices may be totally different. However, even in the same product line and the devices consist of the exactly same components, they will also show small but diverse characteristics, which result from the difference among the hardware components due to the production tolerance. Therefore, an initial calibration is needed before the products are shipped to the customers to ensure performance. Later, as the device is in operation for a while, the hardware degrades physically and the analysis results will shift from the actual ones, which leads to performance degradation. Thus, re-calibration is needed at regular time intervals. The calibration costs are enormous in terms of labor, hardware and time costs, which has a negative effect both for the manufacturers and the end customers.

1.3 Motivation of deep-neural-network based framework for spectral analysis

To address all the problems discussed in Section 1.2, this dissertation proposes a deep-neural network-based comprehensive framework for large-scale spectral analysis. The motivation for the utilization of deep neural networks (DNN) is intuitive since by design, DNN is a data-driven approach that can represent any

complicated non-linear functions precisely. The *universal approximation theorem* shows that if the weights of a neural network are properly set, it can approximate the target inverse function of the physical model $P^{-1}(\mathbf{S})$ defined in Eq. 1.2. To obtain the proper weights, the network needs to be trained on a sufficiently large dataset, which can be, satisfyingly, generated by the physical model.

Besides, DNNs also perform well on classification tasks across many domains, e.g., image classification, object detection and so on. It also results from the characteristics of DNNs that they can learn the implicit relationship between the data and its high-level information. In the context of spectral analysis, it refers to the ability to identify the elements when the spectra data is given. Thus, in principle, both quantitative and qualitative spectral analysis tasks can be solved by the DNNs.

Moreover, this dissertation focuses on not only a general solution for spectral analysis that is academically sound, but it also pays attention to the deployment of the whole framework in the real world, especially in large-scale industrial scenarios. As a full replacement of the existing classic approaches, the framework needs to be comprehensive in many perspectives, e.g., the coverage of the whole procedure from data generation to the final results, the coverage of user cases, the efficiency of the analysis and the deployment in the real world. Hence, advanced optimization techniques for DNNs are also within the scope of work of this dissertation.

1.4 Contributions and outline

The outline of the dissertation reflects the development phases of the whole framework. After this introduction Chapter 1, in Chapter 2, the preliminaries in the spectral analysis as well as in the domain of neural networks are presented. The concept of the framework is given in Chapter 3 to provide a clear overview of all contributions of the dissertation.

The contributions and the corresponding chapters are summarized as follows:

- Element quantification

  The most significant problem of the current SOTA solution in the industry is the high processing time in quantitative analysis, as stated in Section 1.2.2.

  As a starting point of this dissertation, a general deep-neural network-based fundamental framework for large-scale quantitative spectral analysis is proposed. **As a first comprehensive approach is this domain**, extensive experiments to examine the required data size, select feasible network architecture and discuss problem-specific configurations. The approach covers all the relevant elements (a total number of 28) that are involved in the spectral analysis, and it delivers a constant time performance regardless of the task complexity. Compared to classic methods, the approach performs on par and achieves a speedup of 400x under a normal task complexity setting. Besides, additional information on measurement conditions is no longer needed, which simplifies the analysis process.

  This part is presented in Chapter 4, where a more detailed discussion of related work, the design of the framework and the evaluation of the approach on simulation and measurement data are given.

- Element classification

  The qualitative task of element classification (i.e., the identification of present elements) is so far not yet addressed by the classic methods and manual labor-intensive work is needed, as discussed in Section 1.2.1.

  Therefore, the framework proposed for element quantification is adopted and the whole design procedure is performed to identify the most common elements precisely and efficiently. The approach achieves up to more than 99% accuracy on simulation datasets and real measurements from the industry. Besides, to reduce the computational and data storage cost under big data industrial settings, the proposed approach utilizes feature selection to select important features to reduce the data dimension while maintaining

the prediction performance and interpretability. Compared to other dimension reduction baseline methods, this approach outperforms by achieving the best prediction accuracy and providing the most intuitive data reduction result. The application of the feature selection method can reduce 80% parameters and 96.9% FLOPs (floating point operations, a measurement of computational cost) of neural networks with even better test accuracy on real-world data.

The work on element classification is presented in Chapter 6 with all relevant details and evaluation results. Besides, the feature selection method utilized in this part is proposed separately in Chapter 5 as a general approach in the domain of machine learning. The effectiveness of this method is demonstrated by the extensive evaluation with other competitors on various task fields, e.g., text classification, and image classification to name a few.

- Accelerated neural network execution

  After solving the two major tasks in spectral analysis, the deployment of DNNs on the products is expected. Since DNNs require huge computational as well as storage resources, their application may face latency, memory footprint, storage size and power consumption issues, especially on low-power edge devices. To address this problem, this dissertation proposes a first hybrid approach to optimize and accelerate the neural network execution while consistently preserving the final performance. First, feature selection is performed to reduce the data dimension and guarantee an efficient input throughput. Then, after the neural network training, the DNNs are further pruned to obtain a more compact network architecture. Finally, the networks will be quantized to reach a higher compression ratio with low-cost operations for edge devices. Extensive experiments on various target hardware platforms are conducted to demonstrate the effectiveness of this approach, and results show that, compared to the original framework developed in Chapter 6, it can achieve up to 52x mode size compression and 600x speedup with even better performance in most cases. As a representative example, it successfully deploys a DenseNet with only a 0.1 MB model

size and 0.9 ms inference time on a Raspberry Pi, which enables real-time on-site spectral analysis for industrial and commercial applications.

This part of the work focuses on the optimization of the DNN utilization in real-world settings. The overall hybrid approach is presented in Chapter 8. Among the techniques adopted in the approach, the feature selection method is already discussed in Chapter 5. As for the network pruning method, it is thoroughly presented in Chapter 7. Independent of the task definitions in spectral analysis, the method is compared with other SOTA network pruning algorithms on famous large-scale image dataset benchmarks to ensure a fair comparison.

- Efficient calibration with meta learning

  As mentioned in Section 1.2.3, another critical issue that appears in the production is the high calibration costs, which also applies to the proposed DNNs framework. In the previous chapters, DNNs achieve excellent and efficient performance, but one DNN only works on one specific device that it was trained on. Thus, to deploy the models in industry to achieve a commercial-level performance on a vast number of devices, enormous DNNs have to be trained for every device.

  To avoid such costs, this dissertation proposes a meta-learning-based approach to achieve excellent calibration results at minimal cost by learning to calibrate. First, the general spectral analysis problem is formulated as a multi-device multi-configuration task that consists of various basic tasks. Then, a meta network is trained based on large-scale datasets with a basic task-aware design. Finally, the network is calibrated with a few measurements (few-shot) on an unknown device to optimize the device-specific performance. Extensive experiments show the effectiveness and efficiency of this approach over baseline methods by achieving the best pre- and after-calibration error rates across different unknown devices. Besides, compared to baselines after calibration, the approach performs on par even without calibration. This makes the zero-shot setting feasible, which is practical in the real-world scenario where an unknown device needs to be deployed

without reference samples available for calibration. Moreover, the resource analysis shows that this approach requires significantly less expenditure to deploy large-scale devices in industry, which contributes to a huge saving and growth potential.

This part serves in this dissertation as the last step towards the efficient deployment of DNNs in industry via minimal cost device calibration. The details along with the real-world comparison are presented in Chapter 9.

After proposing the comprehensive framework covering the most important aspects in spectral analysis, Chapter 10 concludes this dissertation with a short summary and an outlook addressing the current limitations of the dissertation.

2 Preliminaries

In this chapter, the basic principles in the domain of spectral analysis and artificial intelligence (to be specific, DNNs) are introduced to give a brief overview of the fundamentals of this dissertation.

2.1 Physics in spectral analysis

In this section, the basic physics of spectral analysis is briefly introduced. Since the systems themselves are very complex and are not the focus of this dissertation, the introduction will not go too deep into details. Instead, two representative techniques, laser-induced breakdown spectroscopy (LIBS) and X-ray fluorescence (XRF) spectroscopy, are discussed in the following two sections and a generic physical model is derived in Section 2.1.3.

2.1.1 Laser-induced breakdown spectroscopy

The LIBS is atomic emission spectroscopy that utilizes a pulsed laser beam as its energy source. As shown in Figure 2.1, the pulsed laser with high energy will go through an optical system consisting of mirrors and lenses, so that it can be transmitted to the sample in a desired angle and direction. Then, a small portion of the sample will be removed (known as laser ablation) and a high-temperature plasma is initiated. Shortly after that, during the cooling process, the excited electrons of the atoms fall to the ground states, thus emitting light with unique characteristics (i.e. wavelengths). The light will be captured by

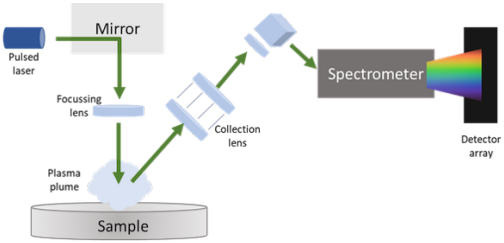


Figure 2.1: The schema of LIBS [27]

the detector through another part of the optical system, resulting in the spectrum measurement.

$$I_{ij}(\lambda) = \frac{1}{4\pi} n_0 A_{ij} \frac{g_i exp^{-E_i/k_B T}}{U(T)} I(\lambda) \tag{2.1}$$

The intensity of the spectrum can be described by Eq. 2.1[28], where I_{ij} is the emission rate density of photons (in $m^{-3}sr^{-1}s^{-1}$), n_0 is the number of neutral atoms in the plasma (in m^{-3}), A_{ij} is the transition probability between level i and level j (in s^{-1}), g_i is the degeneracy of the upper level i ($2J+1$), $U(T)$ is the partition function (in s^{-1}), E_i is the energy level of the upper level i (in eV), k_B is the *Boltzmann constant* (in eV/K), and T is the temperature (in K).

$I(\lambda)$ is the line profile such that Eq. 2.2 is fulfilled, with λ being the wavelength (in nm).

$$\int_{-\infty}^{\infty} I(\lambda) d\lambda = 1 \tag{2.2}$$

Besides, the partition function $U(T)$ is the statistical occupation fraction of every level k of the atomic species as shown in Eq. 2.3.

$$U(T) = \sum_j g_j \exp^{-E_j/k_B T} \tag{2.3}$$

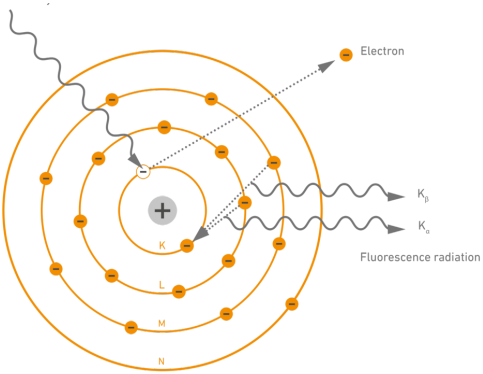


Figure 2.2: The work principle of XRF [29]

2.1.2 X-ray fluorescence spectroscopy

The whole system of XRF spectroscopy works similarly to LIBS, where the major difference is that the source used here is an X-ray instead of a pulsed laser.

In an XRF spectroscopy, the X-ray radiation beam focuses on a spot on the sample surface, and the electrons from the inner shell will be knocked out. Afterward, electrons from outer shells fall naturally into the inner shells, emitting therefore the so-called fluorescence radiation. Similar to LIBS, the fluorescence spectrum is also atom-related due to the unique energy difference of electrons between shells. Therefore, conducting the spectral analysis can find the composition of the sample qualitatively or quantitatively.

$$
\begin{aligned}
I_{E_{i,fl}}(x) = \int_0^x dx' I_{E_0} \cdot exp[-\frac{\mu_{i,E_0}}{\sin\psi_{in}}\rho_i x'] \\
\cdot (\tau_{i,E_0}\omega_{X_i} g_{l,X_i} \frac{j_{X_i}-1}{jX_i}\rho_i \frac{1}{\sin\psi_{in}}) \\
\cdot exp[-\frac{\mu_{i,E_{i,fl}}}{\sin\psi_{out}}\rho_i x'] \cdot \frac{\Omega_{Det}}{4\pi} \cdot \varepsilon_{Det,E_f l}
\end{aligned}
\tag{2.4}
$$

Table 2.1: Overview of the parameters for quantitative analysis with XRF

Parameter	Explanation
$I_{E_{i,\mathrm{fl}}}$	intensity of the measurable fluorescence radiation of element i with photon energy $E_{i,\mathrm{fl}}$
I_{E_0}	intensity of the incident radiation with photon energy E_0
μ_{i,E_0}	absorption coefficient of the element i at photon energy E_0
ψ_{in}	angle of incidence of the exciting radiation
ρ_i	density of the element i
τ_{i,E_0}	photoelectric absorption cross-section of the element i for a photon of energy E_0
ω_{X_i}	fluorescence yield of the absorption edge X_i
g_{l,X_i}	transition probability of the fluorescence line l belonging to the absorption edge X_i
j_{X_i}	jump ratio of the absorption edge X_i
$\mu_{i,E_{\mathrm{fl}}}$	absorption coefficient of element i at fluorescence energy E_{fl}
ψ_{out}	angle of observation/angle in which direction the detector is located
Ω_{Det}	effective solid angle of the detector
$\varepsilon_{\mathrm{Det},E_{\mathrm{fl}}}$	efficiency of the detector at the photon energy E_{fl}

One possible solution for the quantitative analysis is given in Eq. 2.4[30], whereas the overview of all the parameters is given in Table 2.1. It is clear to see that on one hand, both LIBS and XRF are similar in terms of building the spectrometers and measuring the spectra. However, the underlining physics is quite different, complicated, and yet not perfectly described.

Therefore, this requires the classic methods to make great efforts to make the physical modeling precise for each type of spectrometer, which is resource-demanding. However, for DNNs, such processes can be easily worked around by implicitly learning the modeling with data. As shown in the next section as well as the rest of the dissertation, a general solution can be built on a generic physical model, which is not bound by a specific spectrometer type.

2.1.3 Generic abstract physical model in spectral analysis

The previous sections demonstrate that the physics behind different spectrometers is complex and distinct. However, from a high-level perspective, they share a

similar philosophy: a spectrum is one-dimensional data that represents the observation (intensity of the light) of the target object (the sample). The observation is related to the unique structure of the target (the composition and the concentrations) in a nonlinear way. Apart from the sample itself, as described in Chapter 1, the process of measurement (e.g., measurement conditions) also contributes to the final observation. Therefore, in this dissertation, a generic abstract physical model for spectral analysis is given in Eq. 2.5, which is appropriate to represent most spectrometers without mentioning spectrometer-specific configurations.

$$\begin{aligned} \mathbf{S} &= P(\boldsymbol{\theta}, \mathbf{T}, \boldsymbol{\lambda}, \mathbf{K}) \\ \boldsymbol{\theta} &= [\theta_{11}, \theta_{12}, ...\theta_{1j}; \theta_{21}, \theta_{22}, ...\theta_{2j}; ...; \theta_{i1}, \theta_{i2}, ...\theta_{ij}] \\ \mathbf{T} &= [T_1, T_2, ..., T_j] \end{aligned} \tag{2.5}$$

here, $\mathbf{S}$ corresponds to the spectrum, θ_{ij} denotes the concentration of ith element in the jth layer (if applicable, e.g. for coating layers), T_j is the thickness of the jth layer, $\boldsymbol{\lambda}$ is a set of parameters with respect to measurement conditions and $\mathbf{K}$ represent the characteristics of the measurement device. The ultimate goal of spectral analysis is thus to determine $\boldsymbol{\theta}$ and $\mathbf{T}$ from a measurement spectrum given $\boldsymbol{\lambda}$ and $\mathbf{K}$, for which the inverse function $P^{-1}(\mathbf{S})$ is needed.

2.2 Fundamentals in artificial intelligence

To provide a better understanding of the dissertation, in this section, the basic concepts regarding artificial intelligence, particularly neural networks, are introduced.

2.2.1 Definition of neural networks

An (artificial) neural network, as the name suggests, is a network consisting of (artificial) neurons. In artificial intelligence, the idea of neural networks comes

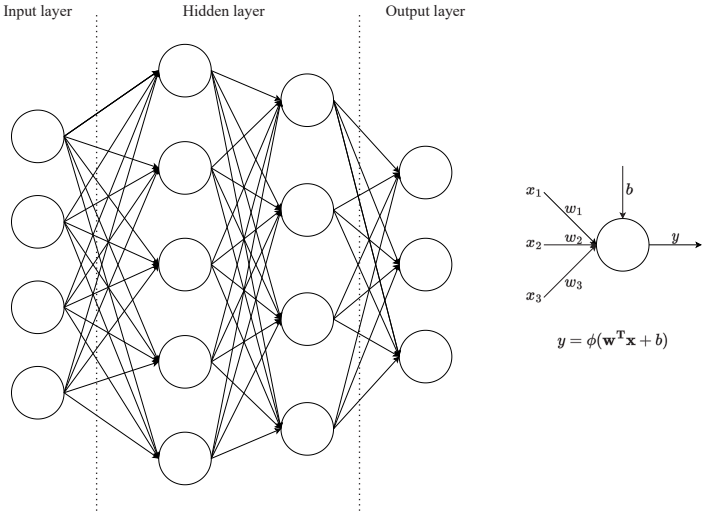


Figure 2.3: The basic architecture of neural networks

from biology and mimics the function of the human brain: the biological neurons connect with others to pass signals, process signals and make decisions.

2.2.1.1 Architecture of Multilayer Perceptrons

For simplicity, the definitions related to the neural networks are given based on the most vanilla neural network architecture: fully-connected neural network, which is also known as Multilayer Perceptrons (MLP). In general, an MLP consists of an input layer, several hidden layers and an output layer, as visualized in Fig. 2.3. Each layer then consists of many neurons, denoted as circles in the figure. The data are passed and processed in the network in a feed-forward fashion, i.e., the output of the previous neurons is the input of the next neurons.

2.2.1.2 Activation function

The output of each neuron is a non-linear function of its input. As shown in Fig. 2.3, suppose a neuron has three inputs from the neurons of the previous layer, denoted as x_1, x_2, x_3, the neuron will assign three different *weights* w_1, w_2, w_3 to each input along with a shared *bias* b. To put it in matrix form, a linear output for linear regression problems will be $\mathbf{w^T x} + b$. However, in this way, the stack of numerous layers, i.e., the stack of linear operations, does not bring any benefit and is equivalent to the result of a network with only one layer. To avoid such problems, non-linearity is introduced so that a deep neural network can represent a very complex non-linear function. The *universal approximation theorem* shows that if the weights are properly given, a neural network can approximate any interesting function within arbitrary errors. This is the motivation for utilizing neural networks in many domains where an implicit relationship between observations and target outcomes exists, but it is not mathematically, or analytically solvable.

Therefore, the output of a neuron is defined in Eq. 2.6 and this non-linear function ϕ is called *activation function*.

$$y = \phi(\mathbf{w^T x} + b) \tag{2.6}$$

There are many choices of activation functions and neither of them outperforms the others in all scenarios. It depends on the applications and the performance is also related to other components used in the neural networks. Fig. 2.4 visualizes several popular activation functions that are defined in Eq. 2.7. In this dissertation, the *ReLU* is the default choice if not otherwise stated.

$$\phi(.) = \begin{cases} \text{Sigmoid: } \dfrac{1}{1+e^{-x}} \\ \text{tanh: } tanh(x) \\ \text{ReLU: } max(0, x) \\ \text{Leaky ReLU: } max(0.1x, x) \end{cases} \tag{2.7}$$

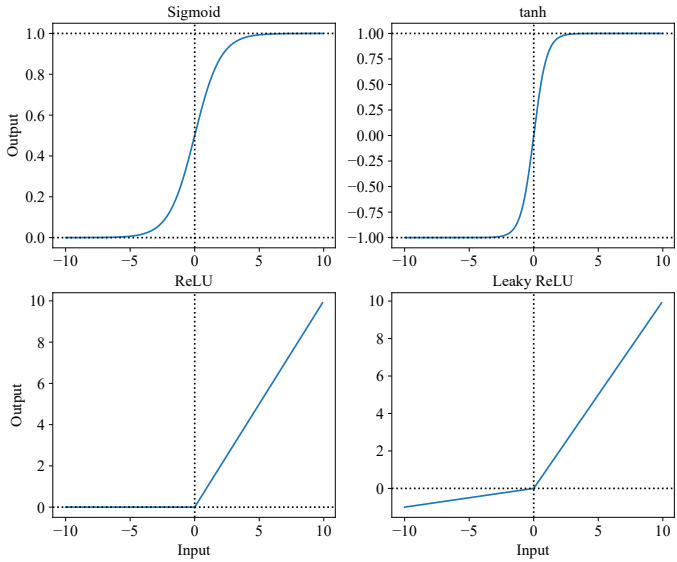


Figure 2.4: Some popular activation functions

For simplicity, the function of the Lth layer is denoted as ϕ^L. Thus, the output (forward pass) of a neural network $\mathbf{y}$ with respect to the input $\mathbf{x}$ is given in Eq. 2.8.

$$\mathbf{y} = \phi^L \cdot \phi^{L-1} \cdot ... \phi^1(\mathbf{x}) \tag{2.8}$$

2.2.1.3 Loss function and evaluation metrics

A neural network is utilized to perform certain tasks, such as *regression* or *classification* tasks. In a regression problem, the network makes predictions for continuous values, such as the temperature of a city, the stock prices and so on. As for classification problems, the target values are discrete and the data need to be classified into different categories, e.g., classification of objects in images (cat, dog, people ...), spam emails detection (spam or not spam).

To evaluate the performance of a neural network on such tasks, a proper evaluation method should be chosen to compare the predictions and the ground truth values (often called *labels* in the machine learning domain), which is also essential to train the neural network. In general, for regression problems, usually the *mean squared error (MSE)*, a popular statistical measurement, is used to assess the goodness of the prediction. Given the labels y and the predictions $\hat{y}$, the MSE is defined as Eq. 2.9. A lower MSE indicates then a better prediction performance.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{2.9}$$

For classification problems, the labels are binary values of either 0 (the object does not belong to the class) or 1 (the object belongs to the class) for a given class. The *binary cross-entropy (BCE)* is often used to find out whether the predictions fit the true labels.

$$BCE = \frac{1}{n}\sum_{i=1}^{n} y_i \cdot log(\hat{y}_i) + (1 - y_i) \cdot log(1 - \hat{y}_i) \tag{2.10}$$

These estimators calculate the error rates that a neural network makes, given the current parameters $\boldsymbol{\theta}$ of the network. Therefore, they are typically called the *loss function* $l(\mathbf{x}, \boldsymbol{\theta})$. The general concept of neural network training is thus to adapt the parameters of the network given training data so that the values of the loss function can be minimized with better parameter sets $\boldsymbol{\theta}^*$, which is formally described by Eq. 2.11.

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \sum_{i=1}^{n} l(x_i, \boldsymbol{\theta}) \tag{2.11}$$

To evaluate the performance of a neural network under different task settings, there are also various metrics that monitor the results with their own focuses. For instance, the *mean absolute error (MAE)*, defined in Eq. 2.12, has the same unit

as the labels and provides thus an intuitive description of the error rate, which is often used in the real world regression problems.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{2.12}$$

For classification problems with multiple classes, the *F1 score* examines the balanced prediction performance. The F1 score is defined as follows, which is the harmonic mean of the *precision* and *recall*:

$$\begin{aligned} F1Score &= 2 \cdot \frac{Precision \times Recall}{Precision + Recall} \\ Recall &= \frac{TP}{TP + FN} \\ Precision &= \frac{TP}{TP + FP} \end{aligned} \tag{2.13}$$

To explain the notation using the element identification problem as an example, TP denotes True Positive and stands for a prediction being positive (e.g., the element Au is present) and true. Similarly, FN and FP stand for False Negative and False Positive. Recall assesses effectively how many "should be found" elements are "found", and the precision indicates how many "found" elements are correct. In the multi-class and multi-label case, where there are more than multiple classes (number of elements) and multiple labels (one sample contains more than one element), the average F1 score across all classes is used in this dissertation.

Besides, the *accuracy* metric, defined in Eq. 2.14, is also widely applied in many classification problems, although it may lead to biased results when the class distribution is unbalanced. For instance, assume that 99% of people do not have a specific disease. In this case, a simple model that always outputs 0 (i.e., not ill) can achieve an accuracy of 99% without any true prediction power.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.14}$$

2.2.1.4 Back propagation and optimizer

The neural networks only work when the right weights are assigned. Therefore, they need to be trained with respect to the problems, or more precisely, to the training data. Since the loss functions are designed for neural networks, they can be theoretically trained using a classic optimization method such as gradient descent. However, unlike many classic optimization problems, neural networks often have significantly bigger parameter space (e.g., billions of parameters) and they are also non-convex and have thus many local minimums. Therefore, the technique of *back propagation (BP)* is adopted to compute the gradient of the network.

Due to the lengthy mathematical description, this section does not cover the whole process of BP, which can be found in many famous textbooks/courses about neural networks. In short, the unique idea of BP is that the gradient is calculated layer-wise, instead of globally, from the last layer to the first layer using the chain rule. In this way, an efficient calculation is possible with powerful modern graphics cards (GPU).

After obtaining the gradient with respect to the loss function, an update of the network parameters is needed. There are many optimizers that are widely used in this domain, such as *stochastic gradient descent (SGD)* and Adam[31].

SGD is a modification of the vanilla gradient descent method that updates only with a subset (called *batch*) of the whole training data. The major reason for the modification is that the training data, especially for the large-scale image datasets, are often too large even to fit in the memory for computation. Therefore, the parameters $\boldsymbol{\theta}$ are updated at a slower but feasible convergence speed to the local optimum. In Eq. 2.15, each update step t is referred to as one *iteration*, and the α is the update step size.

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha \frac{1}{n} \sum_{i=1}^{n} \nabla_{\boldsymbol{\theta}} l(x_i, \boldsymbol{\theta}_t) \tag{2.15}$$

Adam is an advanced optimization method involving two additional moment parameters in the update loop. The update procedure is defined in Eq. 2.16 and it outperforms SGD in most cases with default parameter settings, although at a higher computational cost.

$$
\begin{aligned}
g_t &= \nabla_{\boldsymbol{\theta}} l(x_i, \boldsymbol{\theta}_{t-1}) \\
m_t &= \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t \\
v_t &= \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2 \\
\hat{m}_t &= m_t / (1 - \beta_1^t) \\
\hat{v}_t &= v_t / (1 - \beta_2^t) \\
\boldsymbol{\theta}_t &= \boldsymbol{\theta}_{t-1} - \alpha \cdot \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)
\end{aligned}
\tag{2.16}
$$

For these optimizers, the update step size α is called the *learning rate*, and it represents how fast a neural network should update its weights towards the desired direction in the search space.

2.2.2 Common neural network architectures

In addition to MLP, this section introduces several other popular neural network architectures, which are also evaluated in this dissertation. An overview illustration of the architectures is shown in Fig. 2.5.

2.2.2.1 Convolutional neural networks

Convolutional neural networks (CNN) have been widely utilized in many application domains such as image classification [32] and object detection [33].

The architecture of a classic CNN *AlexNet* [34] is visualized in Fig. 2.6. The input data are $227 \times 227 \times 3$ images with the same pixel width W and height H of 227, and since there are 3 color channels for R, G, and B values, the data has a depth D of 3. To exploit the spatial information in the input data, one *convolutional layer*

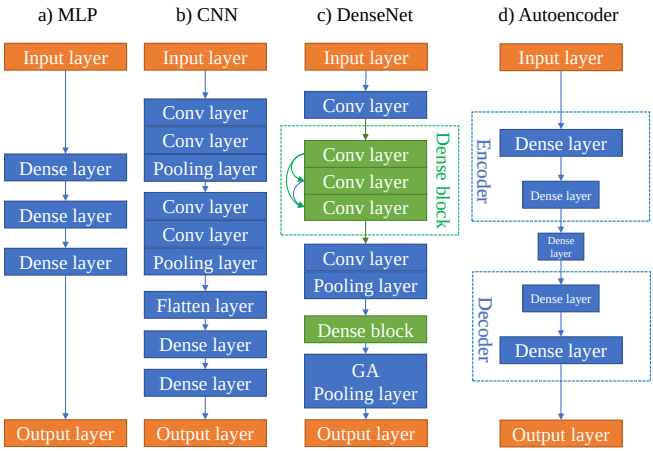


Figure 2.5: Overview of neural network architectures

(Conv layer) consists of several *filters*, each of them is a 3-dimensional tensor, e.g., $11 \times 11 \times 3$ with learnable weights. During the forward pass, the filters are slid across the weight and width axis of the data, outputting the dot product of the filters and the local input data, which is the reason for the name convolutional layer.

The output dimension of a Conv layer depends on the configuration of how the convolutional operations are executed. With a given input size of $W_1 \times H_1 \times D_1$, number of filters K, kernel size of F, the stride (step size of the convolution) S and the zero padding size (zero-filling of the image borders) P, the output size $W_2 \times H_2 \times D_2$ is thus given in Eq. 2.17. For instance, given input data size of $227 \times 227 \times 3, K = 96, F = 11, P = 0$, the output size is then $55 \times 55 \times 96$.

$$\begin{aligned} W_2 &= (W_1 - F + 2P)/S + 1 \\ H_2 &= (H_1 - F + 2P)/S + 1 \\ D_2 &= K \end{aligned} \tag{2.17}$$

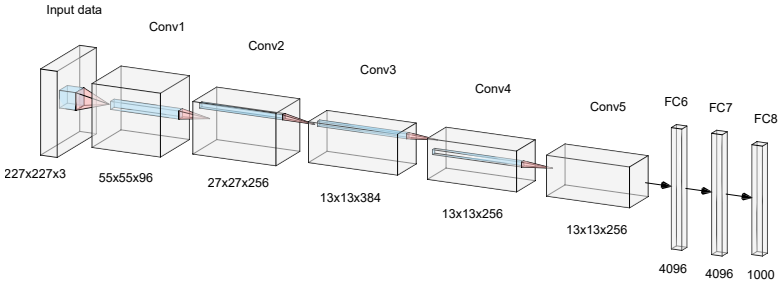


Figure 2.6: AlexNet architecture, visualization based on [35]

Other layers that are designed for CNNs are e.g. *Pooling layer* and *Flatten layer*. The Pooling layer does not contain any parameters but simply calculates and outputs the max (Maxpooling) or mean (Averagepooling) values of a local region. The Pooling layers are utilized to reduce the data volume without losing too much information. The Flatten layer is used to, as the name suggests, flatten the output of Conv layers to 1-d data so that the fully-connected layer (Dense layer) can be connected to the Conv layers. Therefore, it is placed between the last Conv layer and the first Dense layer.

The Conv layers are expected to function as feature extractors, the first Conv layers extract some low-level features from the input data, and the deeper layers build high-level features based on the low-level features. Followed by several fully connected layers, the CNNs perform significantly better than MLPs on classification/regression tasks where spatial information is important.

2.2.2.2 Densely connected convolutional networks

Densely connected convolutional network (DenseNet) [36] is a recently proposed architecture that outperforms most other convolutional networks on image classification benchmark tasks. The unique feature of DenseNet is the introduction of *Dense block*, as visualized in Fig. 2.5 (c). Within a Dense block, all Conv

layers are directly connected and one layer outputs effectively the aggregation of all previous Conv layers. This feature allows a deeper layer to get access to a much shallower layer, which makes the network more compact and efficient. Compared to classic CNNs, DenseNets have significantly fewer parameters and require thus fewer storage resources.

2.2.2.3 Autoencoder

Autoencoder is another popular neural network architecture that is often used for feature transformation, anomaly detection and denoising. The architecture of a plain vanilla Autoencoder is described in Fig. 2.5(d) with a similar structure as an MLP. The primary feature of an Autoencoder is that the number of neurons in each layer decreases in the beginning and then increases, aiming to reconstruct the original input data without loss of information (i.e., $output = input$). This design of a bottleneck in the middle of the neural network allows it to learn low-dimensional yet high-level features, which makes it feasible for feature reduction. Therefore, the output of the bottleneck layer given input spectra is then the transformed features with reduced dimension.

2.2.3 Hyperparameter, overfitting and underfitting

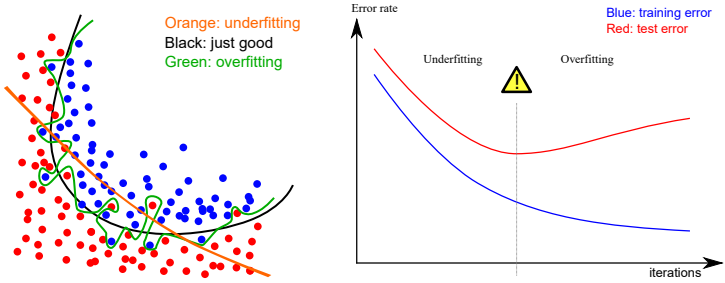
In the domain of artificial intelligence, the term *hyperparameter* is used nearly everywhere. The hyperparameters refer to the parameters related to the model (e.g., a neural network) and the training process that can not be determined with a clear, deterministic procedure.

For instance, when designing the neural network architecture, it is necessary to choose the number of layers, the number of neurons in each layer, the type of activation function, the size of a filter, and so on. Similarly, during training, the learning rate (and how to decrease the learning rate, if any), the number of the batch size and training iterations are also crucial for the final performance. However, there are no such methods that choose the suitable values of these

parameters to achieve the best result. Therefore, they are called hyperparameters and can only be determined empirically. Moreover, depending on task settings, one set of hyperparameters may work well on one task but not for another task.

Like the optimization problem of the neural network training itself, the search for appropriate hyperparameters is time-consuming and usually ends up with a local optimum. For example, the *grid search* evaluates the hyperparameters from an exhaustive combination of finite candidate sets; a *random search* tests the hyperparameters with randomly generated values; *evolutionary search* utilizes evolutionary algorithms to search for a better hyperparameter set.

During the hyperparameter search, it is necessary to assess the performance of the neural networks to make a solid choice. The values of the loss functions and other metrics are good indicators to monitor performance. Note that the assessment should be unbiased, otherwise a choice might be less feasible. Here, the concept of *underfitting* and *overfitting* play an important role.



(a) Goodness of the model, based on [37] **(b)** Error curve, based on [38]

Figure 2.7: Visualization of overfitting and underfitting

Assume a neural network is trained to classify the objects into two classes (red and blue) and it uses a nonlinear line to separate different classes, as shown in Fig. 2.7a. At the beginning, the network does not perform well and the classification line (orange) can not properly separate the two classes. Later, as the training goes on, the black classification line can correctly classify most objects, which is superior. Although some outlines do exist and are not correctly classified, it is

expected not to. However, if the network is further trained to develop the green classification line, which perfectly classifies all the objects it has observed (i.e., the *training data*). But if it is tested on an unknown, yet the data from the same distribution (the *test data*), the error rate will tend to be larger than that on the training data.

This phenomenon is described as the transition from underfitting, where the model still needs to improve its performance, to overfitting, where the model is skewed to the training data and can not perform well on other data. Fig. 2.7b depicts the error curves of a model on training data and test data, respectively.

2.2.4 Tensorflow: a machine learning platform

The implementation, training and deployment of neural networks require high efficiency, robustness, usability and portability. In this dissertation, the *Tensorflow* [39] framework is utilized as the platform to build, train and test the neural networks.

Tensorflow is free and open-source, it covers most aspects that are required to develop and deploy DNN applications. It provides both *Python* and *C++* APIs and GPU support for the acceleration of intensive matrix operations. Besides, the models can be easily deployed on Web (*Tensorflow.js*), mobile and edge devices (*TensorLite*), this multi-platform functionality is a huge plus for large-scale industrial applications where both the development and the deployment are crucial and need to be consistent.

3 Concept and design of the framework

Before diving into the details of all the contributions of this dissertation, this chapter gives a high-level overview of the architecture of the dissertation. First, the requirements for the framework are briefly defined, from which the right solutions should be derived. Afterward, the concept of the framework design is visualized and each module of the framework, which fulfills the corresponding requirement, will be addressed in the following chapters 4 to 9, respectively.

3.1 Requirements for the framework

In this section, the requirements are divided into two subgroups: functional and non-functional requirements. Functional requirements describe **what** the framework should achieve to successfully accomplish the tasks, whereas non-functional requirements guide **how** the framework should do to solve the problem in a better way.

For a better understanding of the requirements addressing the existing problems, in this section, a thorough description of the overall spectral analysis procedure with classic methods is visualized in Fig. 3.1.

The workflow begins with a sample pool and the samples (*A, B, C...*) in the pool need to be analyzed. Then, a suitable device (*1, 2, 3...*) is selected to measure the sample with a feasible measurement condition (*a, b, c...*). Afterward, the classic

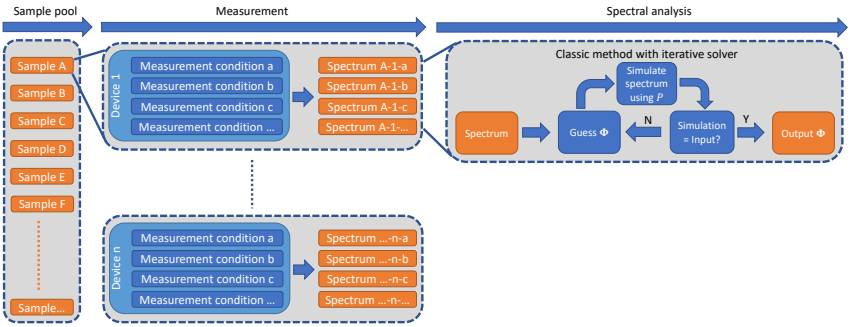


Figure 3.1: The workflow of classic method

method begins to analyze the sample (e.g., *Spectrum-A-1-a*) with all available information (sample composition, device information and measurement condition), where the iterative solver adapts the element parameters Φ until convergence.

The limitations of the procedure are obvious: the sample composition is needed to constrain the parameter space Φ; the device information and measurement condition are needed to obtain a precise physical model; the solver takes many iterations to converge, which is dependent on the complexity of the tasks.

3.1.1 Functional requirements

In the domain of spectral analysis, The most essential functional requirements focus on two major problems: qualitative and quantitative analysis. Since in current classic solutions, qualitative analysis is conducted manually in an inefficient manner due to the absence of feasible algorithms, the requirement for the qualitative analysis is then a first solution towards automatic element identification with high accuracy.

- *FR1.1*: automatic identification of element composition
- *FR1.2*: high accuracy for element identification

As for quantitative analysis, the major limitations of classic methods with iterative solvers are the long and unstable computational time, which is correlated to the number of relevant elements. To demonstrate this issue, assume that a solver needs 10 iterations to search in the space of 0% to 100% for a pure element sample to achieve an accuracy of 1%. Accordingly, the number of iterations needed for a sample with 10 elements will be 10^{10} to obtain similar performance. All else being equal, the search time increases exponentially and it will eventually collapse and can not converge. Therefore, the requirement for quantitative analysis is straightforward: high precision and low computational time.

- *FR2.1*: low error rate for element quantification
- *FR2.2*: low, robust computational cost

3.1.2 Non-functional requirements

Based on functional requirements, the functionality of the framework is concretely described. As for non-functional requirements, they essentially define the optimizations that have to be conducted to make the framework efficient, practical and user-friendly for the final deployment.

The first non-functional requirement regarding real-world applications is therefore the related hardware cost. The neural networks are powerful, yet consume enormous hardware resources. Hence, minimizing the hardware demand plays an important role in cost reduction. Since the cost of neural networks is dependent on the input data dimension as well as its own architecture, it is thus desirable to reduce the data throughput and the network size without losing performance on the tasks.

- *NFR1.1*: low data dimensionality without performance degradation
- *NFR1.2*: compact neural network architecture without performance degradation

Besides, in the actual deployment of the framework, different target platforms may be involved, including professional workstations, mobile devices and low-power edge devices. Budget platforms possess less computational power and thus require further execution optimization. To ensure the robust performance of the proposed framework under various scenarios, considerations regarding cross-platform functionality are crucial.

- *NFR2.1*: consistent performance with respect to different target hardware platforms

Apart from cross-platform ability, cross-device performance is also essential in real-world applications where numerous devices are in use. By default, calibration costs increase linearly with the number of devices in deployment, which contributes to a significant variable cost. Therefore, it is beneficial if such costs can be reduced by a large ratio.

- *NFR3.1*: minimal cost of device calibration

3.2 Concept, design of the comprehensive framework

3.2.1 Overview

Based on the consideration mentioned in Section 3.1, the framework is proposed in a way that all the requirements can be fulfilled. With preliminaries related to neural networks being introduced in Chapter 2, in this section we explain the general idea of the DNN-based framework and demonstrate how the whole framework is built step by step until it is mature to be deployed in the industry.

The backbone of the framework focuses on the functional requirements and serves as the minimum viable product. The modules of the backbone realize the functionality of element quantification and identification to prove the concept of the

Table 3.1: Overview of the requirements on the framework addressed in the dissertation

	Requirements	Chapter 4	Chapter 5	Chapter 6	Chapter 7	Chapter 8	Chapter 9
Functional requirements	FR1.1			x			
	FR1.2			x			
	FR2.1	x					
	FR2.2	x					
Non-functional requirements	NFR1.1		x			x	
	NFR1.2				x	x	
	NFR2.1					x	
	NFR3.1						x

framework. Afterward, further development and optimizations complete the framework with a generalized solution at lower costs. The details of the modules are visualized in Section 3.2.2 and an overview regarding which chapters cover which part of the framework, is given in Table 3.1.

3.2.2 Details of the framework components

3.2.2.1 Element quantification and identification

First, the functional requirements regarding element identification and quantification should be met. Therefore, for each task, we conduct thorough research with extensive experiments in two separate chapters.

In this dissertation, the framework starts with a solution to element quantification problems. As shown in Fig. 3.2, to avoid using iterative solvers that take a long time to compute, we propose a DNN-based approach to simplify the analysis by directly estimating the inverse system function $P^{-1}(\mathbf{S})$ with a prediction network. Besides, to eliminate the effects of various measurement conditions, a scale network is introduced for compensation. Therefore, compared to classic methods, the framework directly feeds the spectrum to the neural networks and can get the results at once.

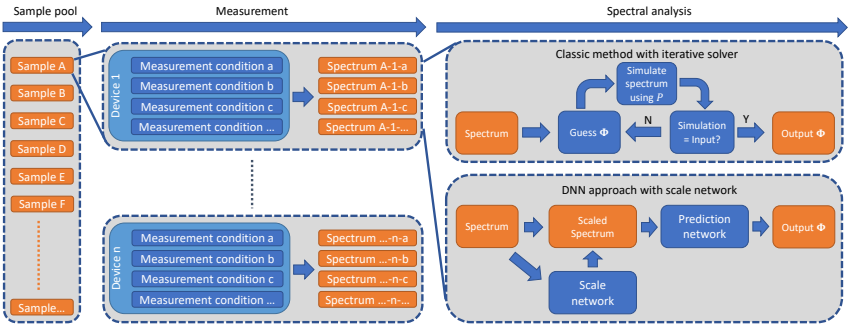


Figure 3.2: The workflow of element quantification

Similarly, the element identification can be solved in the same way, whereas the differences and modifications compared to element quantification are discussed in the corresponding chapter in detail. As for now, the functional requirements *FR1.1*, *FR1.2*, *FR2.1* and *FR2.2* can be fulfilled with the current framework shown in Fig. 3.3.

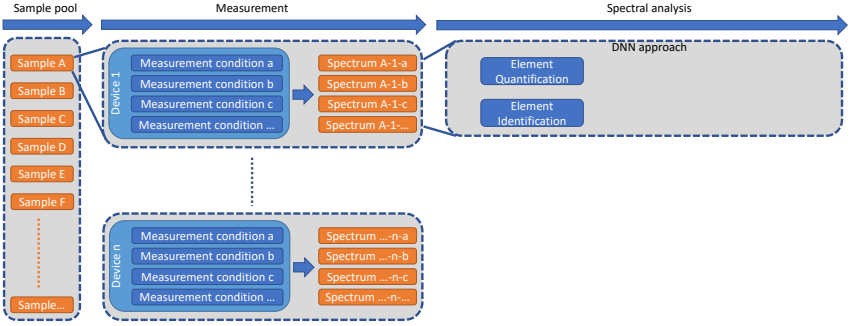


Figure 3.3: Framework with element quantification and identification

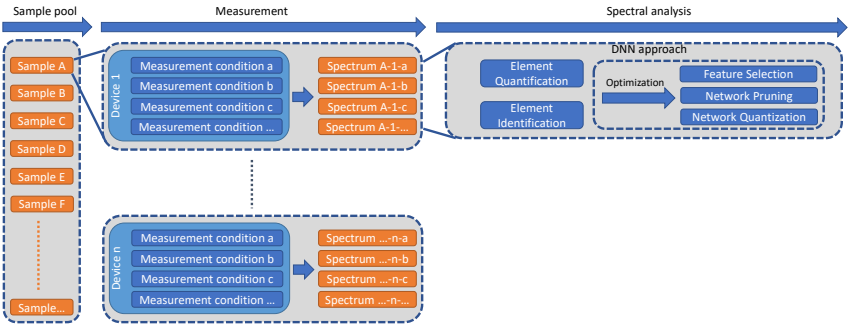
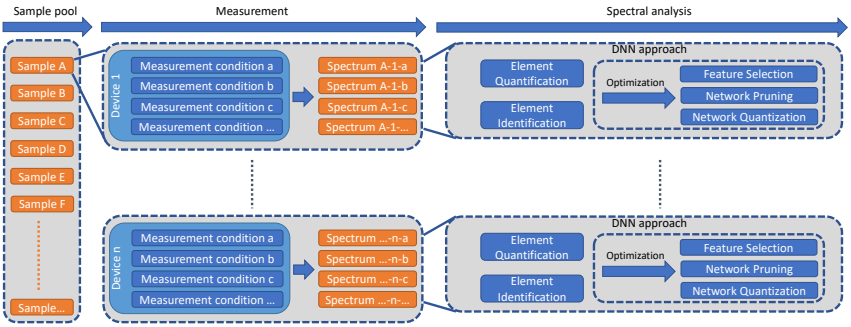


Figure 3.4: Framework with optimization

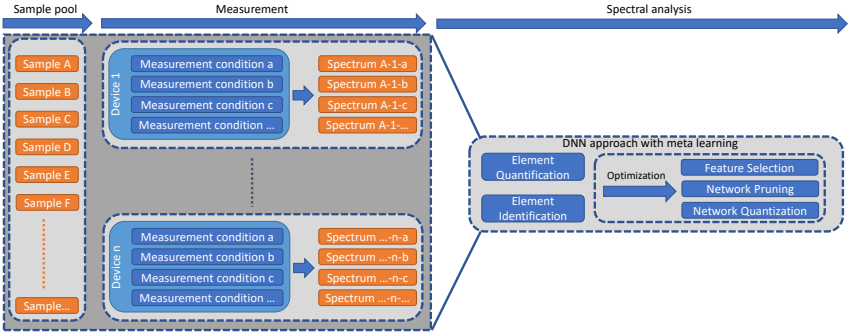
3.2.2.2 Optimization of the framework

After the functional requirements are all met, the framework will be further optimized to fulfill the non-functional requirements. To reduce the hardware as well as the computational costs, the following aspects are taken into consideration.

- The spectral data, as demonstrated in Chapter 1, have plenty of redundant information that is less relevant to the problems. Thus, if only important spectral features are selected as the final input data, the cost of DNNs will significantly decrease because they are input dependent. Therefore, we introduce a feature selection method to accomplish this task.
- Obviously, the cost of a DNN also depends on its own architecture and size. Hence, if the inverse system function can be represented by a smaller neural network without performance degradation, it can then greatly reduce the hardware demand that the DNNs may require. To address this aspect, we propose a network pruning method to slim the DNNs while maintaining their performance on the given problems.
- When it comes to real-world applications, different platforms, especially low-power edge devices, are involved in the deployment. The execution of a DNN may suffer from huge latency issues or may not be possible at all due to limited hardware resources. Therefore, specific configurations of



(a) Framework in real-world applications



(b) Generalized framework in real-world applications

Figure 3.5: Framework overview before and after cross-device consideration

DNNs with light operations are introduced to solve such problems. In this dissertation, we utilized the network quantization technique to decrease the requirements on edge devices.

With all the optimizations integrated into the framework, the framework fulfills *NFR1.1*, *NFR1.2* and *NFR2.1* to provide a highly efficient solution that is feasible in the industrial context.

3.2.2.3 Cross-device calibration

Note that so far, the proposed framework focuses on the solution on a specific device, whereas in reality, the framework should work on enormous measurement devices with various characteristics, as visualized in Fig. 3.5a. It essentially means that DNNs must be re-trained for each device, leading to a significant calibration cost.

To address this problem, the dissertation formulates the spectral analysis problem in a more general form, and the DNNs in the framework are trained in a generalized manner by explicitly taking the difference between devices into consideration. As demonstrated in Fig. 3.5b, the sample pool and device pool are treated as a unified task pool, and the generalized DNNs are trained based on this formulation. This approach enables a fast calibration of the DNNs on an unknown device, and thus meets the last non-functional requirement *NFR3.1*.

4 Element quantification

As stated in Section 1.4, in this dissertation, the first step is to set up a proper baseline framework for quantitative spectral analysis. This chapter begins with the formulation of the element quantification task, then related work is discussed to demonstrate the limitations of previous work. Afterwards, the framework is developed and evaluated on both simulation data and measurement data.

4.1 Current status and related work

One major challenge of quantitative spectral analysis is how to retrieve information from measurements in a robust, fast and accurate manner. Recall that in Section. 2.1.3, the generic abstract physical model that describes a spectrum is defined as Eq. 4.1:

$$\mathbf{S} = P(\boldsymbol{\theta}, \mathbf{T}, \boldsymbol{\lambda}, \mathbf{K}) \tag{4.1}$$

where $\mathbf{S}$ corresponds to the spectrum, θ denotes the concentration of relevant elements, T is the thickness of the layers, $\boldsymbol{\lambda}$ is a set of parameters with respect to measurement conditions and $\mathbf{K}$ represent the characteristics of the measurement device. The ultimate goal of spectral analysis is thus to determine $\boldsymbol{\theta}$ (and $\mathbf{T}$) from a measurement spectrum given $\boldsymbol{\lambda}$ and $\mathbf{K}$, for which the inverse function $P^{-1}(\mathbf{S})$ is needed.

Classic analytical methods utilize fundamental methods [25] and/or Monte Carlo simulations [26] to build the physical modeling. By doing so, one can generate simulation spectra with given input parameters $\boldsymbol{\theta}$, $\mathbf{T}$, $\boldsymbol{\lambda}$ and $\mathbf{K}$. Optimization

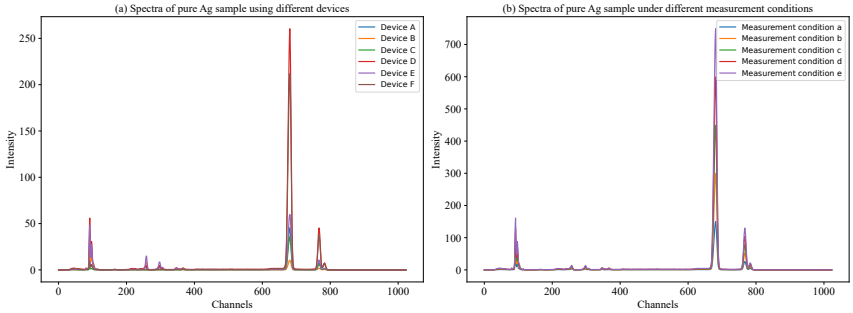


Figure 4.1: Measured spectra of one pure Ag sample using (a) different devices under the same measurement condition or (b) different measurement conditions on the same device. The spectra are easily distinguishable by their intensity and characteristics of peaks. For instance, in (a), device B has the largest peaks in the range of 200-400 but much lower peaks in the range of 600-800, respectively.

methods can be used to find a suitable set of parameters so that the simulation spectrum and measurement spectrum will eventually fit. As a result, the quantitative results are obtained. Still, there are critical limitations of such methods: 1) the simulation is not perfectly precise due to the complexity of physical modeling, which leads to systematic errors during the fitting process; 2) the computational time on these methods, particularly the Monte Carlo simulation, is high. On the one hand, the improvement in the quality of the simulated spectrum leads to an increase in the computation time; on the other hand, usually many iterations are required for an optimization method to find the proper parameter set. Therefore, real-time analysis is often not feasible, especially for complex spectra.

Recently, neural-network-based approaches[2, 4–7, 17, 19] are proposed for spectral analysis and are claimed to be effective due to the ability of neural networks to approximate a nonlinear function very precisely. Compared to classic analytical methods, neural networks can directly serve as the desired inverse function $P^{-1}(\mathbf{S})$ when properly trained. However, related works [1, 2, 6, 17] show quantification results in different application domains with up to 20% test error rate, which is only considered to be "relatively accurate".

The major cause of not satisfying results is that the data for network training is not sufficiently large. Typically a few tens of or hundreds of, at maximum several thousands of spectra are used for training, whereas one single spectrum can contain more than 6000 features (or referred to as channels, or data points). These high-dimensional data with yet few instances available are difficult to train, and overfitting problems often occur. To optimize the training process, some previous works utilize feature selection to reduce the input dimension. For instance, [1,2,5] use empirical strategy to select regions of element peaks manually as input, [4] use a wrapper algorithm to select features. However, the overall goodness of performance is still limited, particularly for empirical strategy. As illustrated in Fig. 4.1, for different devices and/or measurement conditions, the spectra of one identical sample can be significantly different regarding the overall amplitude and existence/absence of certain peaks. Therefore, the manual selection of certain regions of interest is not applicable in terms of generalization. Moreover, some previous works use normalization during data preprocessing, which is in most cases not feasible to train a more general neural network for mainly two reasons: 1) Unlike image data which have a specific value range (typically $0 - 255$) and can thus easily be normalized (e.g., to $0 - 1$), spectra data have unlimited range, and it is hence only applicable under constant measurement device settings. 2) The utilization of normalization compensates for the effect of $\boldsymbol{\lambda}$, $\mathbf{K}$ and $\mathbf{T}$, which makes the quantification of layer thickness an unsolvable task. These reasons also explain why so far, the application of neural networks in spectral analysis is limited to a single device, single measurement condition and element quantification only.

The main limitations of previous works are summarized as follows:

- There is usually only a limited number of spectra available for neural network training, which leads to performance drop and overfitting problems.
- The scope of application is narrow. Besides the constraint of a single device, single measurement condition and element quantification only, the neural network is designed and trained for a specific task, in which the quantitative results of a few known elements are to be determined.

- Potential for improvement regarding the usage of neural network technique is huge. Misuse and/or vanilla use of preprocessing (feature selection, normalization), network design (shallow network architecture) and evaluation (evaluation metrics) can be optimized.

Table 4.1: Summary of all relevant elements

Atomic Number	Name	Atomic Number	Name
14	Si	45	Rh
15	P	46	Pd
16	S	47	Ag
24	Cr	48	Cd
25	Mn	49	In
26	Fe	50	Sn
27	Co	51	Sb
28	Ni	73	Ta
29	Cu	74	W
30	Zn	77	Ir
31	Ga	78	Pt
33	As	79	Au
42	Mo	82	Pb
44	Ru	83	Bi

In this chapter, a large-scale spectral analysis framework using deep neural networks is proposed. We generate a huge number of simulation spectra based on an existing analytical model (which is currently deployed in commercial products) and exploit the power of neural networks. We show that the performance of neural networks is robust, accurate and significantly faster compared to state-of-the-art analytical methods. Besides, we illustrate that the data needed for training is related to the complexity of the systems (i.e., the number of elements), and to obtain a well-generalized neural network, a certain reasonable number of spectra is required. In short, we set up a baseline covering most use cases as an all-in-one solution in the broad domain of quantitative spectral analysis.

The rest of the chapter is organized as follows: Section 4.2 formally defines the task of spectral analysis and gives a short introduction to relevant neural network architectures. In Section 4.3 we conduct comprehensive experiments to construct our baseline framework.

4.2 Task definition and network architecture

As discussed in Section 4.1, element concentration $\boldsymbol{\theta}$ and layer thickness $\mathbf{T}$ need to be determined given measurement spectrum $\mathbf{S}$, measurement condition $\boldsymbol{\lambda}$ and device characteristics $\mathbf{K}$. In this chapter, we hold $\mathbf{T}$ constant because the inclusion of layer systems requires a more complicated discussion and is out of the scope of work. Spectrum $\mathbf{S}$ is in our case a one-dimensional vector of 1024 channels and we define 28 most relevant elements (see Table 4.1) as the size of $\boldsymbol{\theta}$. Thus, we have $\mathbf{S} \in R^{1\times 1024}$ and $\boldsymbol{\theta} \in R^{1\times 28}$.

Both $\mathbf{T}$ and $\boldsymbol{\lambda}$ contribute approximately as nonlinear multipliers of the spectrum. Therefore, we can rewrite Eq. 4.1 as:

$$\mathbf{S} \approx M^t(\mathbf{T})M^\lambda(\boldsymbol{\lambda})P'(\boldsymbol{\theta}|\mathbf{K}) \tag{4.2}$$

where $M^t(\mathbf{T})$ and $M^\lambda(\boldsymbol{\lambda})$ represent the nonlinear contribution of $\mathbf{T}$ and $\boldsymbol{\lambda}$, respectively.

The tasks of this chapter are hence considered as follows:

1. Since there has been no comprehensive, comparable work before, we focus on the fundamental problems of using neural networks in spectral analysis and set up a general baseline covering the most relevant elements.
2. Due to the complexity and limited pages, we hold firstly $\mathbf{T}$, $\boldsymbol{\lambda}$ and $\mathbf{K}$ constant and search for the most feasible network architecture for complex spectral analysis.
3. Then, we extend to variable $\boldsymbol{\lambda}$ to construct the whole baseline framework and discuss the effect of $\mathbf{K}$.

For our baseline framework, we examine the MLP, CNN and DenseNet as a representation of the most popular DNN architectures, which are visualized in Fig. 4.2(a), (b) and (c).

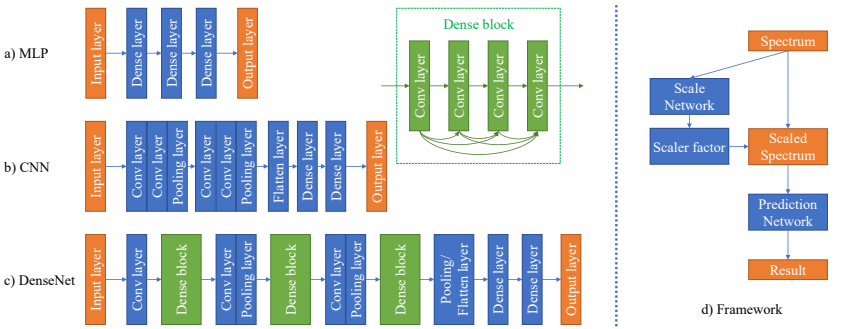


Figure 4.2: Network architecture and overall workflow

4.3 Experiments and results

4.3.1 Appropriate size of training dataset

Before going through a comprehensive comparison of different network architectures with various hyperparameters, we use an MLP with a reasonable hidden layer size of $\{512, 512, 512\}$ to determine the number of spectra that are needed to train a neural network properly. The spectra are generated based on a random approach, in which the number of elements (varying from 2 to 28) and the concentration of each element (from 0% to 100%) are freely chosen. For training, we use SGD with a weight decay of 1e-4 and a Nesterov momentum of 0.9. The initial learning rate is set to 1e-5 and it is divided by 0.1 after 80 and 120 epochs (160 epochs in total). We train the MLP from scratch with varying sizes of training datasets and test the performance on a separate test dataset consisting of 40k spectra.

The results are illustrated in Fig. 4.3. In (a) and (b), with the increased number of spectra, the overall test loss (mean squared error (MSE)) reduces gradually and stays stable when approaching 160k spectra. Furthermore, as we discussed before, overfitting occurs when the size of the training dataset is relatively small and test loss is much higher than training loss. To address this point, we plot the

final *test loss - train loss* after 160 epochs in (b). One can see that the phenomenon of overfitting is significant with a low number of spectra and it disappears when the data get sufficient. Hence, it indicates that for a complex spectral analysis task containing 28 elements, 100k to 150k spectra are a reasonable starting point for good training. We use 140k spectra in the rest of the chapter, if not otherwise mentioned.

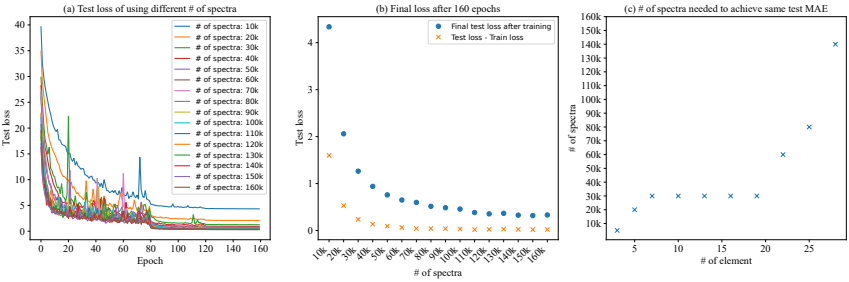


Figure 4.3: Performance of MLP using different data sizes, all else being equal. In (a), the overall test loss at all training steps reduces as the data size increases. Here, the learning rate reduces at epoch 80 and 120, respectively. In (b), the final test losses after 160 epochs using varying data sizes are plotted in blue circles. To illustrate the overfitting effect, the differences between test loss and train loss are plotted in orange x. In (c), with an increased number of involved elements, the required data size to train a neural network also rises.

4.3.2 Selection of model topology

In this section, we evaluate the performance of MLP, CNN and DenseNet with varying topologies and sizes. The neural networks are trained and tested on the same training/test dataset. From an application perspective, we use mean absolute error (MAE) as our evaluation metric since the element concentrations are the final results that need to be determined. Besides, inference time is also an important criterion, especially for real-time applications.

For MLP, we use vanilla grid search to construct MLPs with $\{2, 3, 4, 5, 6, 7, 8\}$ hidden layers and all layers consist of $\{32, 64, 128, 256, 512\}$ neurons, which

results in a total number of 35 MLP candidates. Table 4.2 gives a brief summary of the networks. For instance, MLP_1 has $\{32, 32\}$ neurons in the two hidden layers.

Table 4.2: Summary of MLP candidates

#layers \ #neurons	32	64	128	256	512
2	MLP_1	MLP_2	MLP_3	MLP_4	MLP_5
3	MLP_6	MLP_7	MLP_8	MLP_9	MLP_10
4	MLP_11	MLP_12	MLP_13	MLP_14	MLP_15
5	MLP_16	MLP_17	MLP_18	MLP_19	MLP_20
6	MLP_21	MLP_22	MLP_23	MLP_24	MLP_25
7	MLP_26	MLP_27	MLP_28	MLP_29	MLP_30
8	MLP_31	MLP_32	MLP_33	MLP_34	MLP_35

For CNN, we adopt the classic VGGNet [32] and vary the size of networks accordingly. The candidates are described in Table 4.3, where each *Conv* block represents two *Conv* layers followed by a *Maxpooling* layer. The architecture of CNN_1 is then {2x(Conv layer with 32 filters), Maxpooling layer, 2x(Conv layer with 64 filters), Maxpooling layer, Flatten layer, 2x(Dense layer with 128 neurons)}.

Table 4.3: Summary of CNN candidates

CNN candidate	Conv_1	Conv_2	Conv_3	Dense_1	Dense_2	Dense_3
CNN_1	32	64	N.A.	128	128	N.A.
CNN_2	64	128	N.A.	256	256	N.A.
CNN_3	128	256	N.A.	512	512	N.A.
CNN_4	32	64	64	128	128	128
CNN_5	64	128	128	256	256	256
CNN_6	64	128	128	512	512	512
CNN_7	128	256	256	512	512	512

For DenseNet, we make some major adaptions based on DenseNet-40 to obtain good performance due to the application in quantitative problems (i.e., element quantification in our case) rather than qualitative problems (e.g., classification of images). We increase the number of filters in the first conv layer, use different

growth rates (i.e., how many filters each layer contains) and switch the last Pooling layer to Flatten layer while increasing the size of the final fully connected layers. The configurations are listed in Table 4.4 where only DenseNet_1 uses the Pooling layer for comparison purposes. The number of *DBlock* represents the number of layers in each *Dense* block.

Table 4.4: Summary of DenseNet candidates

DenseNet candidate	Conv_1	DBlock_1	DBlock_2	DBlock_3	DBlock_4	growth_rate	pooling/flatten	Dense_1	Dense_2
DenseNet_1	24	12	12	N.A.	N.A.	12	pooling	256	256
DenseNet_2	96	12	N.A.	N.A.	N.A.	12	flatten	512	512
DenseNet_3	128	6	6	6	6	12	flatten	256	256
DenseNet_4	128	6	6	6	6	18	flatten	256	256
DenseNet_5	128	6	6	6	6	24	flatten	256	256
DenseNet_6	128	6	6	6	6	12	flatten	512	512
DenseNet_7	128	6	6	6	6	18	flatten	512	512
DenseNet_8	128	6	6	6	6	24	flatten	512	512

After training, we evaluate the networks on the test dataset and the results are illustrated in Fig. 4.4. To get a better overview of our evaluation criteria, we plot the test MAE against the number of parameters, the number of floating-point operations (FLOPs) and the inference time of neural networks in (a), (b) and (c), respectively. Note that for all *X-axes*, logarithmic scales are applied.

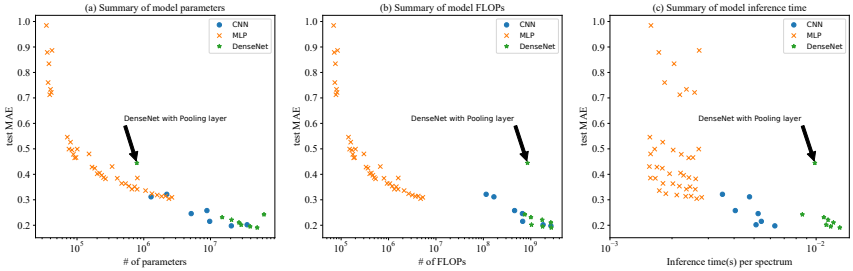


Figure 4.4: Comparison of different neural network architecture and sizes. All x-axes use logarithmic scales. In general, enlarging the network can improve the performance, although the marginal gain decreases. Note that DenseNet with the Pooling layer does not perform well while requiring the same level of storage and computational resources as others in which Flatten layers are applied.

The number of parameters of one network determines the model's size, which needs to be stored in a data storage device. In (a), it is clear that MLPs require fewer parameters in most cases because of the plain architecture, whereas CNNs and DenseNets (using Flatten layer) contain a much larger number of parameters since the Flatten layer is the main cause of parameter growth. Note that DenseNet_1 has a comparable parameter size to MLPs as it utilizes a Pooling layer instead of Flatten layer. However, its test MAE is much higher than other DenseNets, while the FLOPs and inference time are in the same order of magnitude. We observe a general trend that regardless of network architecture, using larger networks usually improves the performance until it saturates at some point, which is intuitive.

The number of FLOPs indicates the required operations for a neural network to execute and it is directly related to the computational time of a neural network. As shown in (b) and (c), compared to parameter size, there is a much larger FLOPs and inference time gap between MLPs and CNNs & DenseNets since the convolutional layers have a higher computational cost than fully-connected layers. In (c), we calculate the inference time using a batch size of 1 to simulate real-world scenarios where each spectrum is separately analyzed. We observe slight differences between theoretical (FLOPs) and real computational time:

1. The execution time of MLPs is not perfectly correlated to their FLOPs. We believe the reason is that there are other overheads (e.g., I/O, matrix computation for evaluation, etc.) that lead to a similar time performance among MLPs ($1 - 2ms$ per spectrum).

2. DenseNets need more computational time than CNNs, even with similar FLOPs. One reason is that for DenseNets, the output of layers needs to be concatenated for further computation, which contributes to a higher total time.

It is clear that CNNs and DenseNets can achieve better test MAE than MLPs and all of them have excellent time performance (up to $20ms$ per spectrum). Nevertheless, some MLPs also have good performance and require less storage and computational resources, which is practical for low-power devices. Therefore, from each category, we choose one network with the least test MAE (which is

also the largest network) as the final candidates. We denote them as MLP, CNN and DenseNet for simplicity.

4.3.3 Selection of hyperparameter

Aside from the neural network architecture, there are still many other hyperparameters that can be carefully tuned to obtain better performance. In this section, we discuss several major aspects regarding hyperparameter selection, namely the choice of optimizers, batch sizes and learning rates.

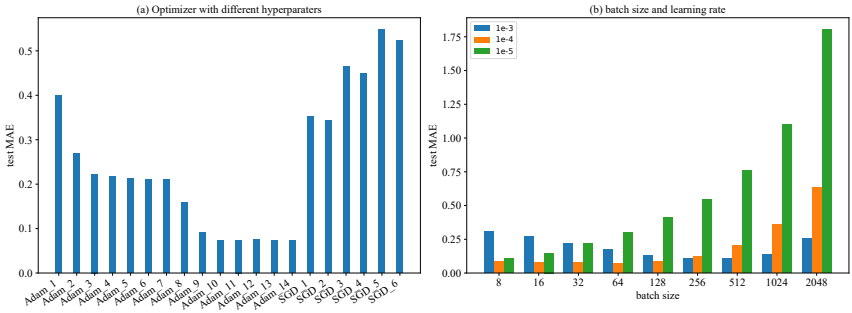


Figure 4.5: Comparable results of hyperparameters. In (a), Adam outperforms SGD in most cases. In (b), we find that using a learning rate of 1e-4 and batch size of 64 can achieve good results.

4.3.3.1 Choice of optimizer

The choice of a proper optimizer is a crucial part for neural network training because the optimizer determines the update steps of network parameters. SGD and Adam [31] are the most popular optimizers among others, and there has been plenty of discussion on the choice of a good optimizer. Some [40, 41] claim that SGD has better stability and less generalization error over Adam, whereas there is also argument [42] that Adam could outperform when properly tuned. In this section, we evaluate both SGD and Adam with various hyperparameters. For Adam,

we use two learning rates (1e-4 and 1e-5) and vary the value of hyperparameter epsilon from 1e-1,1e-2 to 1e-7, which leads to 14 distinct configurations. Since SGD can not converge with a learning rate of 1e-4, we use only 1e-5 and choose a momentum of 0.5, 0.7 and 0.9 with Nesterov set to *True* and *False*, respectively. Thus, we evaluate 6 SGD settings in total.

Fig. 4.5 illustrates the results of optimizer comparison. In (a), we notice that for our specific context, Adam outperforms SGD by a large margin concerning test MAE. The best test MAE of SGD is 0.34, whereas Adam achieves a better value of 0.21 with the same learning rate and 0.07 with a higher learning rate. Therefore, we choose to use Adam as our default optimizer.

4.3.3.2 Batch size and learning rate

Batch size and learning rate usually have a joint effect on network training, i.e., when batch size is changed, the learning rate should also be adjusted accordingly. In this section, we test all the combinations of batch size {8, 16, 32, 64, 128, 256, 512, 1024, 2048} and learning rate {1e-3, 1e-4, 1e-5}. In Fig. 4.5 (b), the comparable results are presented. When using a small learning rate (1e-5), enlarging the batch size leads to worse performance due to reduced parameter update steps within the same training epochs. For the large learning rate (1e-3), increasing the batch size improves test MAE in the beginning since a large batch size provides better stability for the parameter update with a large learning rate. Later, the test MAE gradually rises with the batch size. For our task, 1e-4 is a feasible learning rate that delivers a better and more robust performance within a large range of batch sizes. Thus, we choose the combination of 1e-4 learning rate and 64 batch size for training.

4.3.4 Evaluation on real measurements

To make an extensive evaluation of our approach in the real-world scenario, we collect various samples measured using three different measurement devices

Table 4.5: Summary of measurement spectra and evaluation results

Device	Sample	# repeat	Element concentration in %														Reference		Ours	
			Au	Ag	Pt	Cu	Zn	Sn	Pb	Fe	Ni	Mn	Pd	In	Ir	Ta	MAE (%)	Time (s)	MAE (%)	Time (s)
device 1	sample_1	10				89.75	8.70		1.47	0.02	0.01						**0.06±0.01**	21.00±4.15	0.12±0.01	
	sample_2	10				61.42	35.63	0.13	2.51	0.15	0.06						**0.14±0.00**	13.09±7.12	**0.14±0.00**	
	sample_3	10				60.00	38.80	0.65	0.50	0.02	0.01						**0.13±0.01**	17.13±5.75	0.22±0.01	
	sample_4	10				88.40	3.42	4.42	4.16	0.09	0.16						**0.14±0.00**	14.77±7.35	0.15±0.01	
	sample_5	10				69.00	0.10	0.01	0.01	0.61	30.22	0.82					**0.60±0.03**	2.46±0.40	0.70±0.01	
	sample_6	10				85.30	0.01	9.07	1.98		3.37						0.15±0.01	19.88±5.86	**0.08±0.01**	
	sample_7	10				77.10	0.26	7.16	14.80	0.02	0.49						**0.87±0.01**	12.96±9.87	0.90±0.01	
device 2	sample_8	200	57.83	29.98		12.19											**0.11±0.02**	0.85±0.20	0.17±0.02	**0.049±0.002**
	sample_9	200	74.83	4.84		20.33											0.13±0.02	0.90±0.17	**0.07±0.01**	
	sample_10	200	33.09	12.44		38.66	15.81										**0.20±0.02**	0.97±0.18	0.25±0.02	
	sample_11	200	58.59	4.49		36.92											**0.15±0.01**	0.94±0.18	0.16±0.01	
	sample_12	200	95.01	2.48		2.51											**0.04±0.01**	0.83±0.22	**0.04±0.02**	
device 3	sample_13	10	2.00	58.00			3.35	2.00					32.90	1.50			**0.24±0.01**	17.48±2.15	**0.24±0.00**	
	sample_14	10	55.60	24.40	1.00	14.00	1.00						3.70	0.20			**0.50±0.00**	16.63±7.68	0.51±0.01	
	sample_15	10	74.00	14.50	1.50		3.30						5.50	1.00	0.10	0.10	**0.10±0.01**	11.22±9.01	0.15±0.00	

(Device_1, Device_2 and Device_3) under different measurement conditions. A summary of these samples is given in Table 4.5. The exact element concentrations are shown in % and each sample is measured several times (see column "# repeat") to avoid possible bias caused by the random measurement noises. The average performance of the reference analytical method and our approach is listed in the table, which will be discussed in detail later.

Until this section, this work focuses on training neural networks using simulation spectra where $\boldsymbol{\lambda}$ and $\mathbf{K}$ are constant, which is only applicable when the measurement condition and measurement device hold constant. However, as mentioned in section 4.1 and 4.2, when evaluating a real measurement under different measurement conditions and/or measurement devices, the spectrum will differ from the training dataset and thus lead to inaccurate results.

According to Eq. 4.2, the effect of $\boldsymbol{\lambda}$ can be roughly treated as a linear scale factor $M^{\lambda}(\boldsymbol{\lambda})$ of the original spectrum. Therefore, if the measurement spectrum is properly re-scaled, this effect can be compensated and the neural network can still achieve good performance.

The determination of $M^{\lambda}(\boldsymbol{\lambda})$ can be calculated mathematically as long as the element concentrations are known, which is however not our case. Instead, we introduce one additional neural network to predict the scale factor of a given spectrum. The original training data will be scaled with a randomly generated

Table 4.6: Performance comparison on measurement spectra measured with different devices. The MAE values are shown in %.

		D1 data			D2 data			D3 data		
		MLP	CNN	DenseNet	MLP	CNN	DenseNet	MLP	CNN	DenseNet
D1 model	No scale	20.86	12.11	21.14	41.45	27.44	42.79	69.09	49.62	71.11
	MLP_scale	0.39	0.36	**0.33**	1.31	1.26	1.35	0.63	0.52	0.68
	CNN_scale	0.39	0.34	**0.33**	1.31	1.26	1.35	0.63	0.52	0.68
	Reference		0.30			-			-	
D2 model	No scale	7.02	3.60	7.34	11.91	5.29	11.88	20.55	12.11	20.62
	MLP_scale	0.96	1.03	1.03	0.29	0.15	0.19	0.53	0.38	0.47
	CNN_scale	0.95	1.00	1.03	0.29	**0.14**	0.19	0.53	0.42	0.47
	Reference		-			0.13			-	
D3 model	No scale	6.04	5.91	6.10	4.65	6.00	4.60	1.16	1.25	1.11
	MLP_scale	0.89	0.85	0.87	0.41	0.20	0.27	0.40	0.35	**0.30**
	CNN_scale	0.89	0.81	0.87	0.41	0.39	0.27	0.40	0.35	**0.30**
	Reference		-			-			0.28	

scale, and we train an MLP and a CNN (denoted as MLP_scale and CNN_scale) to predict the corresponding scales using the same architecture as selected before. We train the networks with the same approach as given in the previous section and compare the results with mathematical solutions. Omitting some details, we find that both MLP and CNN networks perform well and have a test MAE of scale factor at about 0.03, which is very precise.

Therefore, our whole proposed framework (as shown in Fig. 4.2 (d)) consists of two neural networks: one scale network and one prediction network. The spectrum will be fed into a scale network to obtain the corresponding scale, then it will be re-scaled and the prediction network gives the final result.

We evaluate the prediction performance of final candidates MLP, CNN and DenseNet on measurement data from all three devices, with and without scale network. Besides, we also compare the results of the classic analytical method, which is denoted as the reference. To illustrate the effect of parameter $\mathbf{K}$ which is determined by individual measurement device characteristics, we generate three simulation datasets based on Device_1, Device_2 and Device_3, respectively. Then, we train the neural networks using each simulation data and denote the networks as D1 model, D2 model and D3 model. Finally, the networks are tested on

real measurements from Device_1 (D1 data), Device_2 (D2 data) and Device_3 (D3 data).

Table 4.6 summarizes the comparable results. Examining the performance of the D1 model on D1 data, the D2 model on D2 data and the D3 model on D3 data, it is noticeable that without a scale network, the test MAE of the prediction network is high due to different measurement conditions $\boldsymbol{\lambda}$ between simulation and measurement. In comparison, the introduction of the scale network dramatically improves the final performance, regardless of which combination of the prediction network and scale network is tested. Moreover, CNN and DenseNet outperform MLP constantly.

Comparing the results of neural networks across different devices, we notice that D1 models trained on D1 data perform the best on real measurements from the same device (D1 data). The same applies to the D2 model on D2 data and D3 model on D3 data. In other cases, the performance of neural networks is worse due to the difference in datasets with unique device characteristics $\mathbf{K}$. Besides, we observe that CNN and DenseNet often have better generalization ability and better test MAE on unknown datasets than MLP.

The results of the reference method are also listed in Table 4.6. Since it can only evaluate data from the same device, only three test MAEs are given. The performance of the analytical method is negligibly better than our methods by 0.03%, 0.01% and 0.02%, respectively. However, as shown in Table 4.5, its computational time is up to 400x slower than our approach (21.00s compared to 0.049s). Nevertheless, the unstable time performance (different evaluation times on different samples) is still one big concern. For instance, the reference method needs on average 21 seconds to obtain the quantification results on sample_1, whereas sample_8 requires only about 1 second. This is primarily because the reference method needs different iterations to achieve good local minima for different spectra.

Table 4.5 also gives more detailed comparable results on each sample. It is noticeable that neither the reference method nor our method can outperform the other in terms of test MAE. However, our approach achieves a much better

and more robust inference time performance and is thus feasible for real-time applications.

5 General feature selection

Chapter 4 proposes a baseline framework for quantitative spectral analysis and outperforms the commercial solutions with the same level of precision with a significant speedup. Before applying the framework to the qualitative spectral analysis, we will discuss in this chapter the concept of feature selection as an intermediate step because in the next chapter, we want to not only extend our framework for element identification but also come to a solution that is efficient in terms of the computational cost. As we will see later in Chapter 6, spectral data do have redundant information that can be safely removed without performance degradation.

Therefore, we propose in this chapter a feature selection method *watermelon* that outperforms most SOTA competitors. Note that this method is a general approach and it is not limited to the domain of spectral analysis. Based on the evaluation of text data, face image data, biological data, and so on, this method proves to be powerful and efficient, which eliminates the potential bias if it is only examined on spectral data.

5.1 Introduction

In the era of big data, classification techniques are often faced with challenges from the enormously growing amount of high-dimensional data in different domains. Thus, selecting a small subset of features while minimizing the generalization error has become a focus in such scenarios. Feature selection methods can be categorized according to different perspectives [43], one common approach

is to partition them into three groups concerning different selection strategies: *wrapper*, *filter* and *embedded* methods [44]. Wrapper methods evaluate the feature subsets by directly training them with a predefined learning algorithm, thus, they are in general very computationally expensive for high-dimensional data and rarely applied in real-world problems. Meanwhile, embedded methods interact with the learning algorithm during the training process to select a subset of features, which are usually more time efficient than wrapper methods. Nevertheless, they are still slower than filter methods and the performance depends highly on the learning algorithm. In contrast, filter methods are independent of any learning algorithm and score the features by analyzing the properties of data. They consume in general less time and can avoid overfitting problems in most cases, however, may fail to select the best subset of features [45].

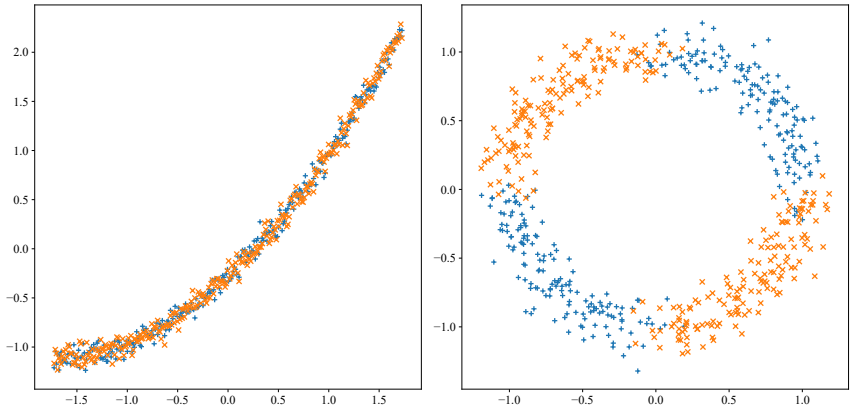
In the past several decades, more than one hundred different feature selection methods have been proposed and most of them are filter methods due to their time efficiency [43]. Due to the fact that finding the global best feature subset is generally NP-hard, most algorithms apply therefore sub-optimal approaches based on forward/backward sequential search strategies by adding/removing features one by one. Regarding the evaluation criteria, these algorithms can be roughly grouped into the following subgroups: *similarity-based*, *sparse-learning-based*, *statistical-based* and *information-theoretical-based* methods [43].

Similarity-based methods assign more importance to the features which can better preserve data similarity than others. *Fischer Score* [46] and *Trace Ratio Criterion* [47] select those features in which the values are similar within the same class but dissimilar across different classes. *ReliefF* [48] selects features that can separate instances from different classes. As the name indicates, sparse-learning-based methods use sparse regularization terms to remove task-irrelevant features. Liu et al. [49] use $l_{2,1}$-norm regularization to obtain a subset of features and *Efficient and Robust Feature Selection* [50] employs a joint $l_{2,1}$-norm minimization on both the loss function and the regularization. Meanwhile, statistical-based methods such as *f_score* [51] and *Gini Index* [52] use different statistical measures to determine whether a feature can separate instances from different classes properly.

As for information-theoretical-based methods, the general approach is to maximize feature-class relevance while minimizing feature-feature relevance, which is actually thought of as redundancy, based on information-theoretic concepts, e.g. *mutual information (MI)*. Mutual information can measure any kind of linear or nonlinear relationship between different variables and is invariant under transformations in the feature space [45]. Hence it has been widely used in this family of methods to score the features. Concretely, they use feature-class relevance as the reward term and feature-feature relevance as the penalty term. In the last few decades, many algorithms have been proposed, they are for instance *Mutual Information Maximization (MIM)* [53], *Mutual Information Feature Selection (MIFS)* [54], *Minimum Redundancy Maximum Relevance (MRMR)* [55], *Conditional Infomax Feature Extraction (CIFE)* [56], *Joint Mutual Information (JMI)* [57, 58], *Conditional Mutual Information Maximization (CMIM)* [59, 60], *Double Input Symmetrical Relevance (DISR)* [61] and *Fast Correlation-Based Filter (FCBF)* [62]. One motivation for using mutual information in these algorithms is that the *Bayes error rate*, which indicates the lowest possible error rate for any classifier, can be bounded by the mutual information [45] (more details in Section 5.2).

This kind of approach has been proven to be powerful in many real-world problems. However, there are still some limitations. One major problem is that the score term(s) and the penalty term(s) are usually not comparable and one term often becomes negligible w.r.t. other term(s). Addressing this problem, some approaches (e.g. Hall and Smith [63], Estévez et al. [64]) suggest the use of *normalized mutual information (NMI)* to quantitatively scale the mutual information to the range of zero to one, where zero means no mutual information and one means perfect correlation.

A further problem is, as Guyon And Elisseeff [44] suggest, that a significant relevance between features does not necessarily impact the performance of a classifier and may, on the contrary, provide more information and reduce the noise. Fig. 5.1 illustrates that while features with perfect monotonic correlation are truly redundant, using highly relevant features may still bring more gain due to their great complementarity.



(a) monotonic correlation **(b)** non-monotonic relevance

Figure 5.1: Two examples of different kinds of relevance. In (a), there is no improvement by using two features instead of only one. However, in (b), a quasi-perfect separation can be achieved in the two-dimensional space.

Inspired by the discussions above, in this chapter we propose a filter method *Watermelon* in a wrapper manner, which scores individual features by directly estimating the corresponding Bayes error rate based on *kernel density estimation* and generates the feature subset using forward search. This idea is effectively applying a simple wrapper method that selects features greedily based on the performance of an ideal classifier. Besides, compared to other algorithms, our approach interprets the effects of relevance quantitatively in a different way. It rewards significant feature-feature relevance, which is usually considered as redundancy by other methods and penalizes only significant monotonic correlation between features. For simplicity, in this chapter, we use the word correlation for monotonic relationships and relevance for non-monotonic relationships if not specified.

To evaluate our method, we compare our approach with other state-of-the-art feature selection algorithms (including all the methods mentioned above) on various classification benchmarks [43]. Overall our approach outperforms other competitors and achieves top performance across different tasks.

The rest of the chapter is organized as follows: Section 5.2 introduces some preliminary concepts and related literature. In Section 5.3 our approach will be presented and Section 5.4 illustrates the performance of the proposed method through experiments.

5.2 Background and related work

5.2.1 Bayes error rate

Consider a classification problem where a data vector x needs to be classified into one of L classes. Let $P(c_i)$ denote the *a priori* class probability of class i, $i = 1, 2, ...L$, $p(x|c_i)$ denote the class *likelihood*, which is the conditional probability density of x given that it belongs to class i. And the *a posteriori* probability $P(c_i|x)$, which is the probability that a data vector belongs to class i, is:

$$P(c_i|x) = \frac{p(x|c_i)P(c_i)}{\sum_{i=1}^{L} p(x|c_i)P(c_i)} \tag{5.1}$$

A *Bayes classifier* assigns then a data vector x to class i if class i has the highest posterior. The error rate of such classifiers is Bayes error rate (BER), which is defined as [65]:

$$BER(x) = 1 - \sum_{i=1}^{L} \int_{C_i} P(c_i)p(x|c_i)\mathrm{d}x \tag{5.2}$$

where C_i is the region where class i has the highest posterior.

5.2.2 Kernel density estimation

Kernel density estimation (KDE) is a non-parametric approach to estimating the probability density function and the general form is:

$$\hat{p}(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n} K_{\mathbf{H}}(\mathbf{x} - \mathbf{x}_i) \tag{5.3}$$

where $K_{\mathbf{H}}(.)$ denotes the kernel function and $\mathbf{H}$ is a $d \times d$ bandwidth matrix for the d-dimensional data $\mathbf{x}$ with n instances.

Then, the class probability $\hat{P}(c_i)$ and the conditional probability $\hat{p}(\mathbf{x}|c_i)$ can be defined as [66]:

$$\hat{P}(c_i) = \frac{n_{c_i}}{n} \tag{5.4}$$

$$\hat{p}(\mathbf{x}|c_i) = \frac{1}{n_{c_i}}\sum_{i \in I_{c_i}} K_{\mathbf{H}}(\mathbf{x} - \mathbf{x}_i) \tag{5.5}$$

where n_{c_i} is the number of instances that belong to class i and I_{c_i} is the indices of these instances.

5.2.3 Spearman's rank correlation coefficient

Slightly different from the most well-known *Pearson coefficient*, which measures the linear relationship of two variables, Spearman's rank correlation coefficient (Cor) assesses monotonic relationships and is formally defined as:

$$Cor(x,y) = \rho = \frac{S_{xy}}{S_x S_y} = \frac{\sum_{i=1}^{n}(R(x_i) - \overline{R(x)})(R(y_i) - \overline{R(y)})}{\sqrt{\sum_{i=1}^{n}(R(x_i) - \overline{R(x)})^2 \sum_{i=1}^{n}(R(y_i) - \overline{R(y)})^2}} \tag{5.6}$$

where $R(x)$ and $R(y)$ are the ranks of variables, $\overline{R(x)}$ and $\overline{R(y)}$ are the corresponding mean ranks. Then we have $Cor(x,y) \in [-1,1]$ and there is a perfect

positive (negative) monotonic correlation if $Cor(x, y)$ is 1(-1). A zero means the absence of correlation.

5.2.4 Information-theoretic concepts

Entropy gives the uncertainty of a random discrete variable X with the mass probability $P(x_i), x_i \in X$. It is mathematically defined as:

$$H(X) = -\sum_{x_i \in X} P(x_i) \log(P(x_i)) \tag{5.7}$$

Given another discrete random variable Y, the *conditional entropy* which describes the uncertainty of X given Y is:

$$H(X|Y) = -\sum_{y_j \in Y} P(y_j) \sum_{x_i \in X} P(x_i|y_j) \log(P(x_i|y_j)) \tag{5.8}$$

where $P(y_i)$ denotes the prior probability of y_i and $P(x_i|y_i)$ denotes the conditional probability of x_i given y_i.

Mutual information, which measures the amount of information that two discrete variables X and Y share, can be defined as:

$$\begin{aligned} I(X;Y) &= H(X) - H(X|Y) \\ &= \sum_{x_i \in X} \sum_{y_j \in Y} P(x_i, y_j) \log \frac{P(x_i, y_j)}{P(x_i)P(y_j)} \end{aligned} \tag{5.9}$$

Mutual information is symmetric and is zero when X and Y are statistically independent.

As mentioned in Section 5.1, the Bayes error rate regarding X is bounded above and below by MI as Eq. 5.10, and the Bayes error rate is minimized when $I(X;Y)$ is maximized [45].

$$1 - \frac{I(X;Y)+\log 2}{\log |Y|} \leq BER(X) \leq \frac{1}{2}(H(Y) - I(X;Y)) \tag{5.10}$$

The term normalized mutual information (NMI) is not standardized and there are several possibilities to normalize mutual information [64, 67, 68]:

$$NMI(X;Y) = \begin{cases} \frac{I(X;Y)}{min(H(X),H(Y))} \\ \frac{I(X;Y)}{\sqrt{H(X)H(Y)}} \\ \frac{I(X;Y)}{max(H(X),H(Y))} \\ \frac{2I(X;Y)}{H(X)+H(Y)} \\ \frac{I(X;Y)}{H(X,Y)} \end{cases} \tag{5.11}$$

Then, *conditional mutual information* gives the mutual information shared by discrete variables X and Y given discrete variable Z:

$$\begin{aligned} &I(X;Y|Z) = H(X|Z) - H(X|Y,Z) = \\ &\sum_{z_k \in Z} P(z_k) \sum_{x_i \in X} \sum_{y_j \in Y} P(x_i, y_j|z_k) \log \frac{(P(x_i, y_j|z_k)}{P(x_i|z_k)P(y_j|z_k)} \end{aligned} \tag{5.12}$$

As shown in [43, 69], most of the information-theoretical-based feature selection methods, including MIM, MIFS, MRMR, CIFE, JMI, CMIM and DISR, can be unified by a conditional likelihood maximization framework:

$$J(X_k) = I(X_k;Y) + \sum_{X_j \in S} g[I(X_j;X_k), I(X_j;X_k|Y))] \tag{5.13}$$

where S denotes the current selected feature set, $X_j \in S$ is a feature in the current S and $g(.)$ is a linear or nonlinear function w.r.t. $I(X_j;X_k)$ and $I(X_j;X_k|Y)$. $J(.)$ represents a feature selection criterion for a unselected feature X_k. The

higher (or the less) the value of $J(X_k)$ is, the more important the feature X_k is. As we explained in Section 5.1, the feature-feature relevance $I(X_j; X_k)$ is used as the penalty term and the feature-class relevance $I(X_j; X_k|Y)$ is used as score term.

5.3 Watermelon feature selection

We pose the feature selection problem as follows: Given a dataset $X = \{X_1, X_2, ...X_d\} \in R^{n\times d}$ with n instances and d features and the corresponding class vector $Y \in R^n$ with L classes, find subset $S \subset F$ with m features that maximizes the performance of a classifier, where $F = \{f_1, f_2..., f_d\}$ is the original feature set.

For binary classification, we estimate the $BER(f_k)$ for each individual feature f_k based on KDE. As for multi-class problem, instead of directly estimating the $BER(f_k)$, we divide the classification problem into $\binom{L}{2}$ binary classification problems and estimate $BER(c_i, c_j|f_k)$ of feature f_k for each class-pair $(i, j) \in L, i \neq j$. In this way, our approach gets more information about the performance of feature f_k regarding each class, hence it is able to select a new feature that improves the classification error of a specific class. Moreover, it also circumvents the issues of efficiency and overfitting brought by multinomial estimations [65,70]. Using Eq. 5.2 and 5.5, the estimated pairwise BER can be defined as:

$$\begin{aligned} BER(c_i, c_j|f_k) = 1 &- \int_{C_i} \hat{P}(c_i)\hat{p}(X_k|c_i)\mathrm{d}x \\ &- \int_{C_j} \hat{P}(c_j)\hat{p}(X_k|c_j)\mathrm{d}x \end{aligned} \tag{5.14}$$

where $BER(c_i, c_j|f_k) \in [0, 1]$. We use the sum of BERs as the score function $Q(f_k)$, thus, a lower score indicates a better feature.

$$Q(f_k) = \sum_{i,j\in L, i\neq j} BER(c_i, c_j|f_k) \tag{5.15}$$

Due to the difficulty of the multi-dimensional integral of multivariate density functions [65], our approach simply estimates the BERs for each individual feature as a base score and, as we explained in Section 5.1, assesses the relevance and correlation between features and updates their scores dynamically. The criteria are described as follows:

- if a feature is significantly correlated with any selected feature, it should be penalized.
- if a feature is significantly relevant (but not correlated) to any selected feature, it should be rewarded.

The adaption will be applied to both the selected features and the new feature. For simplicity of notation, we only show the update of the new feature and the update of selected features follows the same schema.

Consider the situation that we have already selected one feature f' and now evaluate a new feature f_k. If f_k is highly correlated with f', i.e., $Cor(f', f_k)$ is significant, the score of f_k will be penalized. And if $|Cor(f', f_k)| = 1$, which means a perfect correlation and f_k can not bring any improvement, we have effectively $BER(c_i, c_j|f_k) = 1$ for every class-pair.

For general case, the penalty function of $BER(c_i, c_j|f_k)$ is defined as Eq. 5.17. The $BER(c_i, c_j|f_k)$ of feature f_k increases with the highest absolute correlation coefficient between f_k and any feature f' in S, and it will be penalized to one (i.e. $upper\_bound = 1$) if there is a perfect correlation. We apply an activation function $AF(x, th)$ to the absolute value of the correlation coefficient to suppress the effect of weak correlation because in this situation, we consider it as not correlated. $AF(x, th)$ is zero if x is under the threshold and grows to one when x is also one.

$$AF(x, th) = \begin{cases} 0 & \text{if } x \leq th \\ \frac{1}{1-th}(x - th) & \text{if } x > th \end{cases} \tag{5.16}$$

$$
\begin{aligned}
&penalty(BER(c_i, c_j|f_k)) = BER(c_i, c_j|f_k)+ \\
&(upper\_bound- \\
&BER(c_i, c_j|f_k))AF(\max_{f' \in S} |Cor(f', f_k)|, th_{cor})
\end{aligned}
\tag{5.17}
$$

Besides, high relevance should be rewarded. We use NMI to measure the relevance and in this chapter, it is defined in Eq. 5.18. How to quantify the improvement brought by significant relevance is still an open question. The effects can be very task-specific and in this chapter we assume that if two features are perfectly relevant (i.e. $NMI(f'_k, f_k) = 1$), one feature should perform at least as well as the other one because one feature is a non-redundant representation of another feature and this representation brings extra information. Based on this assumption we can make a worst-case update defined in Eq. 5.19.

$$
NMI(X;Y) = \frac{2I(X;Y)}{H(X) + H(Y)} \tag{5.18}
$$

$$
\begin{aligned}
&reward(BER(c_i, c_j|f_k)) = BER(c_i, c_j|f_k)- \\
&max(0, (BER(c_i, c_j|f_k)- \\
&BER(c_i, c_j|f'_k))AF(NMI(f'_k, f_k), th_{nmi}))
\end{aligned}
\tag{5.19}
$$

We notice that NMI also captures correlation and we only want to reward relevance. Although the values of NMI and Cor are in the same range, they are still not directly comparable. Since high Cor leads to high NMI but not vice versa, we consider Cor as a dominant term and update the $BER(c_i, c_j|f_k)$ in two steps: reward relevance and then penalize correlation. For a new feature f_k and each feature $f' \in S$, the $BER(c_i, c_j|f)$ will be updated by Eq. 5.20. Note that for $f' \in S$, the $upper\_bound$ in Eq. 5.17 should be the original value of itself.

$$
BER^*(c_i, c_j|f_k) = penalty(reward(BER(c_i, c_j|f_k))) \tag{5.20}
$$

This update schema realizes effectively the following procedure: in the first step, if features are relevant, all the BERs will reduce to the same class-pair-specific

minimum. And in the second step, if features are furthermore correlated, the BERs of selected features increase back to their original values and the BERs of the new feature will be all penalized to one.

Algorithm 1 Watermelon feature selection

Require: $X \in R^{n \times d}$, $Y \in R^n$, L, m
1: init $S \leftarrow \emptyset$
2: **for** $k = 1 : d$ **do**
3: **for** $i, j = 1 : L, i \neq j$ **do**
4: calculate $BER(c_i, c_j | f_k)$
5: **end for**
6: calculate score function $Q(f_k)$
7: **end for**
8: $f_k^* = \underset{f_k \in F}{\operatorname{argmin}} Q(f_k), S \leftarrow S \cup f_k^*$
9: **while** $|S| < m$ **do**
10: **for all** $f_k \in F \setminus S$ **do**
11: calculate $Q(S; f_k)$
12: **end for**
13: $f_k^* = \underset{f_k \in F \setminus S}{\operatorname{argmin}} Q(S; f_k), S \leftarrow S \cup f_k^*$
14: update all score functions
15: **end while**
16: **return** S

Note that we only actually apply the update after a new feature is selected. To determine which feature should be selected, we evaluate the sum of all the updated scores $Q(S; f_k)$, which is defined in Eq. 5.21, if a specific feature f_k is selected. Then the f_k, which leads to the lowest $Q(S; f_k)$, will be selected and the update will be applied. The whole procedure of our algorithm is shown in Algorithm 1.

$$Q(S; f_k) = \sum_{f \in S \cup f_k} \sum_{i,j \in L, i \neq j} BER^*(c_i, c_j | f) \tag{5.21}$$

Table 5.1: Summary of feature selection methods

Category	Method
similarity-based	fischer_score[46]
	reliefF[48]
	trace_ratio[47]
sparse-learning-based	ll_l21[49]
	ls_l21[49]
	RFS[50]
statistical-based	f_score[51]
	gini_index[52]
information-theoretical-based	CIFE[56]
	CMIM[59, 60]
	DISR[61]
	FCBF[62]
	ICAP[71]
	JMI[57][58]
	MIFS[54]
	MIM[53]
	MRMR[55]

5.4 Experiments

In this section, we compare our approach with seventeen popular feature selection algorithms, which are listed in Table 5.1, on seventeen classification benchmarks. To fairly evaluate all the algorithms, we choose various datasets (collected by Li et al. [43]) with continuous or discrete values from different domains. They are text data, face image data, handwritten image data, spoken letter recognition data and biological data with up to 40 classes and 11340 features. Note that some datasets contain only very few instances, which is very challenging for the algorithms. Table 5.2 shows the summary of the datasets.

Table 5.2: Summary of experiment datasets

Data Type	Dataset	#instances	#features	#classes
text data	PCMAC	1943	3289	2
face image data	COIL20	1440	1024	20
	ORL	400	1024	40
	orlraws10P	100	10304	10
	warpAR10P	130	2400	10
	warpPIE10P	210	2420	10
	Yale	165	1024	15
handwritten image data	USPS	9298	256	10
	Gisette	7000	5000	2
spoken letter recognition data	Isolet	1560	617	26
biological data	CLL_SUB_111	111	11340	3
	Colon	62	2000	2
	GLIOMA	50	4434	4
	Lung	203	3312	5
	Lymphoma	96	4026	9
	nci9	60	9712	9
	TOX_171	171	5748	4

5.4.1 Experiment setup

All the datasets will be preprocessed and standardized to zero mean and unit variance. For the comparative algorithms, we use the implementation by Li et al. [43]. In order to quantitatively assess the algorithms' ability to choose essential features, we consider the protocol of [72, 73] and let all the algorithms use the preprocessed data to select the first 200 most important features for each dataset. Then we train a *linear Support Vector Machine (SVM)* (implemented by *scikit-learn* [74]) with different cardinalities of features to evaluate their performance. To avoid overfitting and ensure reproducibility, we split the datasets into 10 different train sets (80%) and test sets (20%) using 10 fixed seeds. Finally, we use the average accuracy of the 10 splits as the performance metric of the SVM on the selected feature subsets. For a fair comparison, we do not tune any hyperparameters of the algorithms (if any) or SVM and keep them as they are. For our approach, we keep $th_{cor} = 0.5, th_{nmi} = 0.3$.

Table 5.3: Average accuracy results of dataset COIL20

	#Features								
Method	10	25	50	75	100	125	150	175	200
fischer _score	21.9	52.5	86.2	90.2	90.6	92.1	93.0	93.1	94.9
gini_index	56.0	77.0	86.8	86.4	88.5	89.0	93.3	94.4	94.1
ICAP	67.9	78.1	78.9	78.9	78.2	79.4	83.2	85.4	89.6
JMI	65.1	73.6	75.6	80.3	79.9	80.3	82.3	86.1	88.4
l1_l21	35.0	60.0	73.2	78.1	81.5	83.2	89.2	90.2	91.1
MIFS	26.2	39.2	46.3	54.1	59.1	65.1	67.2	69.7	73.8
MIM	49.1	72.5	83.6	84.6	86.2	87.6	88.4	89.0	89.3
MRMR	67.8	85.1	88.6	92.5	93.7	93.6	94.2	94.3	94.9
reliefF	38.5	71.6	80.7	83.1	86.0	89.1	91.5	94.0	94.8
RFS	61.8	74.7	81.2	81.9	83.6	84.9	87.5	87.8	89.4
CIFE	38.7	38.7	38.7	38.7	38.7	38.7	38.7	38.7	38.7
CMIM	67.9	78.1	78.9	78.8	78.1	79.5	83.2	85.4	89.5
DISR	54.8	75.2	87.5	91.1	92.8	95.0	95.5	95.9	96.2
f_score	21.9	52.5	86.2	90.3	90.6	92.2	93.0	93.0	94.8
trace_ratio	21.9	52.4	86.1	90.2	90.6	92.1	93.0	93.1	94.9
ls_l21	26.2	68.8	85.9	90.1	93.0	94.2	95.1	95.0	95.7
FCBF	19.4	19.4	19.4	19.4	19.4	19.3	19.4	19.4	19.4
Watermelon	**83.6***	**94.2***	**96.4***	**97.5***	**97.8***	**98.3***	**98.3***	**98.4***	**98.4***

5.4.2 Results and analysis of experiments

As an example, table 5.3 presents the evaluation results on the dataset COIL20. The bold values with an asterisk indicate the best performance achieved among all the competitors. Our approach clearly outperforms all the other algorithms, regardless of how many features are used by the classifier. Besides, our approach shows its great efficiency by achieving an average accuracy of 83.6% with only 10 features while the best performance of other algorithms is only 67.9%. This phenomenon applies also to other benchmarks in most cases. In table 5.4 we present a summary of all the results by averaging the average accuracy obtained by SVM with the first 10, 25, 50, 75, 100, 125, 150, 175 and 200 features, which are selected and ordered by the corresponding feature selection methods. In the last column, we show the average ranks of the algorithms regarding the accuracy obtained on each benchmark. Our approach is noticeably better than any other method by obtaining an average rank of 1.9, while the average rank of the next

Table 5.4: Summary of evaluation results. Accuracy is the average accuracy obtained while varying the cardinality ($[10, 25, 50, 75, 100, 125, 150, 175, 200]$) of selected features. The last column lists the average ranks of the methods on all the benchmarks.

Method	CLL_SUB_111	COIL20	Colon	GLIOMA	Isolet	Lung	Lymphoma	nci9	ORL	orlraws10P	PCMAC	TOX_171	USPS	warpAR10P	warpPIE10P	Yale	Gisette	Avg. Rank
Watermelon	76.4	**95.9***	**84.9***	**78.0***	**86.4***	91.9	87.0	**76.4***	**90.6***	**95.8***	89.5	**85.3***	**92.1***	**93.5***	**98.9***	**74.9***	**93.7***	1.9
DISR	64.6	87.1	83.6	65.2	72.1	89.5	90.3	71.9	79.5	75.1	89.9	73.0	82.2	86.1	96.2	65.7	93.2	6.2
ICAP	63.8	80.0	81.1	63.8	70.1	89.1	**91.3***	71.9	83.7	67.8	89.7	83.8	87.0	82.3	96.6	60.8	92.5	7.2
fischer_score	58.6	79.4	79.0	77.6	74.0	88.9	86.3	73.9	81.4	80.2	88.5	79.7	86.6	84.2	97.6	66.2	92.8	7.6
JMI	62.3	79.1	83.5	64.0	69.3	90.5	91.2	70.5	78.6	91.4	89.6	67.8	88.2	84.5	97.5	61.9	92.7	7.41
f_score	58.6	79.4	79.0	77.6	74.0	88.9	86.3	74.6	81.4	80.2	88.5	79.7	86.6	84.2	97.6	66.2	82.3	7.8
trace_ratio	58.6	79.4	79.0	77.6	74.0	88.9	86.3	74.5	81.4	80.2	88.5	79.7	86.5	84.2	97.6	66.3	92.4	7.8
MRMR	55.1	89.4	79.0	68.2	77.3	86.6	91.3	70.6	82.7	72.4	**90.5***	62.2	73.3	85.0	98.3	59.3	92.2	8.0
CMIM	63.8	79.9	81.3	63.8	70.0	89.1	**91.3***	69.9	83.7	67.8	89.7	71.5	87.0	82.3	96.6	60.8	91.9	8.2
MIM	63.9	81.1	78.5	71.6	59.0	87.6	87.3	67.8	60.9	85.3	89.7	74.3	86.9	83.1	94.8	59.3	92.9	9.1
reliefF	62.5	81.0	83.2	71.4	63.2	90.4	80.2	60.1	76.8	77.8	73.4	76.8	88.0	85.8	95.5	55.9	92.8	9.1
gini_index	**79.0***	85.1	79.7	76.4	60.2	89.9	66.5	39.4	77.9	65.2	89.9	69.2	79.4	72.4	94.8	45.9	92.8	10.3
RFS	74.4	81.4	58.2	32.0	73.0	**92.7***	79.6	34.4	51.4	51.6	86.6	85.1	89.3	68.5	95.4	34.6	91.0	11.2
l1_l21	58.4	75.7	80.7	68.8	68.6	90.7	82.4	45.2	56.0	49.3	84.5	81.5	83.5	80.5	95.8	42.7	83.4	11.9
ls_l21	42.1	82.7	60.9	48.7	79.5	69.0	47.5	26.0	84.6	61.9	70.7	57.6	90.4	73.8	95.0	59.1	78.3	12.7
MIFS	54.6	55.6	78.5	48.3	64.9	84.1	85.1	46.1	79.1	77.6	86.3	55.7	72.1	63.0	95.6	46.9	84.7	13.8
FCBF	50.9	19.4	83.1	37.0	21.8	81.0	86.7	70.6	9.3	19.5	87.3	22.3	30.8	19.6	26.2	12.1	84.7	15.1
CIFE	41.7	38.7	79.8	48.0	61.4	70.7	63.3	24.3	25.4	63.5	82.0	30.0	32.7	26.2	92.8	20.3	87.6	16.0

best method DISR is only 6.2. It proves the effectiveness and robustness of our approach by outperforming other competitors with a significant leading rank and achieving the highest accuracy in 13 of 17 benchmarks.

Subsequently, following Brown et al. [69], we use *Friedman test* and then *Nemenyi post-hoc test* to validate whether our approach is statistically significantly better than other algorithms based on the ranks. The analysis results are presented in Fig. 5.2 as a *Significant Dominance Partial Order Diagram*. Every bold line connecting two algorithms indicates a significant difference (In our case, since we order the algorithms from top to bottom with ascending values of ranks, this difference means that the upper one is better than the lower one) at the 99% confidence level, every dashed line at the 95% level and every dotted line at the 90% level. This diagram illustrates that in our experiments, our approach is significantly superior to MIM, reliefF, gini_index, RFS, l1_l21, ls_l21, MIFS, FCBF and CIFE with 99% confidence, to CMIM with 95% confidence and to f_score, trace_ratio and MRMR with 90% confidence, which agrees with the observation on the average ranks. Moreover, we find that MIFS, FCBF, and CIFE are in general less powerful compared to other methods. Note that the absence of a connection does not mean that there is no significant difference, but that the number of datasets is not enough for the Nemenyi test to make a decision [69,75].

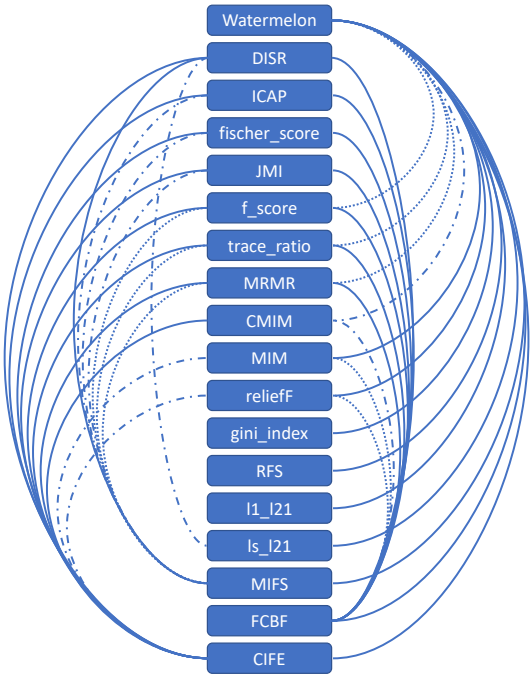


Figure 5.2: Significant Dominance Partial Order diagram. Methods are placed top to bottom regarding their ranks, and a connection between any two methods indicates a statistically significant difference using the Nemenyi post-hoc test at the corresponding confidence level. For instance, watermelon is superior to MIM with 99% confidence.

5.4.3 Discussion

The results of the experiments show that our method outperforms others by a large margin. To better explain what is the key to achieving the advantage, we re-run the experiments with two modified versions of the proposed method: *watermelon-B* and *watermelon-B-S*. Watermelon-B scores and selects a feature subset purely based on the BERs and does not adjust the scores according to feature redundancy or relevance. Watermelon-B-S uses the BERs as base scores and takes the feature redundancy into consideration, however, feature relevance is ignored.

Table 5.5: Comparison of different versions of watermelon. The first two columns show the average accuracy and rank achieved by the methods over all the benchmarks, and the last column shows the overall ranks among all the competitors. The comparison is based on the two runs of the whole experiment using a linear SVM and random forest.

Classifier	Method	Avg. Acc	Avg. Rank	Overall Rank
SVM	watermelon-B	79.4	7.4	7
	watermelon-B-S	86.3	2.8	1
	watermelon	87.7	1.9	1
random forest	watermelon-B	82.0	8.8	8
	watermelon-B-S	87.3	3.7	1
	watermelon	89.7	2.7	1

We keep other experiment setups constant and repeat the same procedure as before. The corresponding results are summarized in the first three rows of table 5.5. In the first two columns, we list the average accuracy and rank achieved by the methods over all the benchmarks. Besides, the overall ranks of the methods among all the competitors are shown in the last column.

The watermelon-B, which only uses the BERs as feature scores, obtains an average rank of 7.4 with 79.4% average accuracy over all the datasets and takes the 7th place out of 18 competitors. It shows that scoring and selecting the feature subset by purely estimating the Bayes error rate is already a very competitive approach, although it does not achieve absolute state-of-the-art performance. One major problem of the simple approach is the ignorance of feature interaction.

Watermelon-B-S takes the feature redundancy into consideration and thus gets significant improvements by achieving an average accuracy and rank of 86.3(+6.9)% and 2.8(-4.6), respectively. Besides, it already outperforms other competitors with the 1st place overall rank. Hence it is clear that removing redundant features is a crucial part of feature selection, as many algorithms claimed. In this paper we further constrain the definition of redundancy: only monotonic relationships should be penalized. Besides, we state that feature relevance, which is defined as a non-monotonic relationship, brings extra benefits due to the complementarity of features. This statement is also proved by the experiments. Compared to watermelon-B-S, the average accuracy and rank of the original watermelon are further improved by 1.4% and 1.1 through rewarding relevant features.

In addition, before making a final conclusion, it is essential to check whether the evaluation is skewed by the specific choice of the classifier (linear SVM). Thus, we repeat the whole experiment again with a *random forest* while holding all other settings constant for a fair comparison. We use the implementation of the random forest also from scikit-learn with default hyperparameters and show the final results in the last three rows of table 5.5. The performance of random forest is overall better than SVM on all the benchmarks in most cases, which is also as expected since the random forest is most likely to be the best compared to other classifiers [76]. As Table 5.5 shows, the original version of watermelon and watermelon-B-S still outperform others regarding their overall ranks, which again proves the effectiveness and robustness of our approach. Note that the average ranks using random forest are slightly lower than SVM due to the fact that random forest is not only a powerful classifier but also an embedded feature selection method. Thus, the proposed performance is actually the result of "feature selection of feature selection", which makes up the performance margin between watermelon and others. That is also the main reason why we do not use random forest in our main experiment: it introduces extra factors and thus disrupts the evaluation.

6 Element identification with interpretable dimension reduction

This chapter, based on the framework proposed in Chapter 4, focuses on the qualitative spectral analysis. This chapter begins with the same approach as in the quantitative part to extend the baseline framework for element identification. Besides, to further improve the non-functional performance, i.e., the costs of the framework, this chapter applies the watermelon feature selection method proposed in Chapter 5, and compares it with several dimension reduction methods to demonstrate its superiority. This is the initial motivation for the development of the hybrid approach for real-time low-cost spectral analysis in Chapter 8, which builds the linkage between academic research (make it work) and industrial applications (make the work efficient) that are all within the scope of work of this dissertation.

6.1 Introduction

Fast, automatic and accurate qualitative spectral analysis is an essential part in many application domains where the presence of certain elements needs to be determined given measured spectral data. The identification task traditionally requires a well-trained expert to conduct a labor-intensive workflow manually, which is very time-consuming.

To accomplish such objectives automatically, neural network-based approaches for element identification have been proposed since the late last century [3] starting with the usage of shallow MLPs. Following work applies further techniques to improve the performance of neural networks, such as a much deeper network, the utilization of CNNs [9, 77, 78], feature reduction [12, 79], batch normalization and dropout [22]. These methods are proven to be effective in terms of prediction accuracy [77].

Despite the progress achieved previously, there are still certain limitations and potential for improvement:

- The scope of work is usually limited to identifying only a small number of elements (typically 3-7 [12, 22, 77]) with one neural network being responsible for the prediction of each element, which is not feasible when a more generalized solution is required, especially for large-scale industrial applications with big data involved. As the task complexity increases, the cost of data preparation, network training, storage, transmission and deployment also raises exponentially.
- In many cases, the training dataset is not sufficient to train a neural network with good performance [9, 79] and there are fewer real-world measurements for a solid evaluation, which is also a major reason for the limited scope of work.
- The application of feature reduction often requires prior knowledge. For instance, one needs to manually choose features where spectral peaks are present to select regions of interest (ROI) [79], which is not feasible across different application types or even different measurement conditions. Therefore, it is not applicable in real-world industrial applications.

To address the problems mentioned above, in this chapter, the framework for element quantification is extended for element identification. First, we compare different neural network architectures and train the networks using large-scale simulation data, where most of the relevant elements (the complete list of 28 elements is summarized in Table 4.1 in Chapter 4) are covered by a single network.

Results on measurement data show that CNN-based networks outperform MLPs in most cases, whereas MLPs require fewer computational resources and are also competitive in certain cases. Besides, to enable an efficient workflow and minimize the computational cost with high-dimensional spectral data, the feature selection method *watermelon* is adopted to perform feature selection to reduce the data dimension. Comparable results with other data reduction methods show that this approach achieves better accuracy and provides more intuitive interpretable results, and tests on measurement data illustrate that neural networks trained on a small feature subset (32 from 1024 original features) can achieve even better prediction accuracy. Moreover, due to the reduced input data dimension, the network size can be reduced to 19.82% and 3.17% of the original size regarding the number of parameters and floating-point operations (FLOPs), respectively.

The main contributions of this chapter can be summarized as follows:

- The extended framework significantly improves the performance of DNN-based approaches on large-scale spectral identification with a thorough evaluation of various network architectures and sizes.
- By introducing feature selection for high-dimensional spectral data, the proposed approach reduces the computational demand by a large ratio with even better prediction performance. Compared to other dimension reduction techniques, watermelon outperforms on real measurement data and provides better interpretability for explainable AI (XAI), which is crucial for industrial applications.
- Besides, the test on simulation data and measurement data suggests that dimension reduction provides in general a better real-world performance against noises and overfitting for high-dimensional data, which is an important best practice for other relevant domains.

The rest of the chapter is organized as follows: In Section 6.2 we give a brief task definition and an introduction to related work. Then, the extended framework with interpretable dimension reduction is proposed in Section 6.3, and Section 6.4 presents the experiment results with further discussion.

6.2 Related work and preliminary

6.2.1 Task definition

Recall that the generic physical model for spectral analysis is formulated as follows:

$$\mathbf{S} = P(\boldsymbol{\theta}, \mathbf{T}, \boldsymbol{\lambda}, \mathbf{K}) \tag{6.1}$$

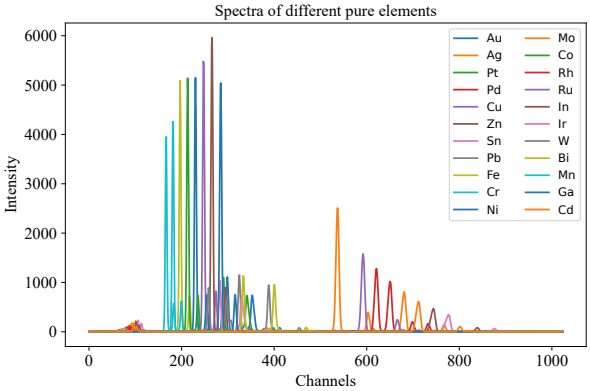


Figure 6.1: Measured spectra of different pure elements. Although each element has unique peaks, they often overlap with the peaks of other elements, which makes it hard to identify a wide range of elements without error.

similar to Chapter 4, we have $\mathbf{S} \in R^{1\times 1024}$ in our case and $\boldsymbol{\theta} \in R^{1\times 28}$ determine the presence and relative amplitudes of peaks in a spectrum (see Fig. 6.1). For a spectral identification task, the target is then the qualitative analysis of the element concentrations (i.e., $\boldsymbol{\theta} \in \{0, 1\}$), as shown in Eq. 6.2:

$$\boldsymbol{\theta} = P^{-1}(\mathbf{S}|\mathbf{T}, \boldsymbol{\lambda}, \mathbf{K}) \tag{6.2}$$

6.2.2 Related work

Early neural-network-based works mainly adopt shallow MLPs to perform spectral identification tasks with few spectra at hand. For instance, [21] trains an MLP with one hidden layer to classify stellar spectra, and the dataset is relatively small, with only 213 spectra available. In order to reduce the data dimension, principal component analysis (PCA) is applied. As for gamma-ray spectra analysis, [79] collects 409 spectra and also trains an MLP with one hidden layer. To better train the network with few samples, feature reduction is performed. It selects element peaks based on the second-order derivative of the spectra, where the thresholds for peak identification are manually set. In addition, this approach requires manually determined reference peaks to match. [22] trains an MLP on simulation data and utilizes the dropout method to achieve better performance while minimizing the overfitting effects. The simulated spectra contain up to 5 elements and the MLP is trained to identify only one element. [12] applies MLPs with linear activation functions to identify up to 7 elements, and the optimal linear associative memory (OLAM) approach is applied to train the network. To reduce the feature space, one spectrum is divided into subparts and the area under the curve of each part is used as the transformed feature.

Recently, CNN models have been introduced in the domain of spectral analysis. [77] applies CNNs to the whole spectra to identify different radionuclides. One major disadvantage of this work is that a dedicated CNN is trained for each element, which leads to significant time and resource demands. To train the CNNs, they generate synthetic data based on the Monte Carlo simulation. Still, this approach is slow, and the spectra' quality is not ideal. [9] uses single-layer multiple-kernel-based CNN to classify different biological sample groups. However, only 360 spectra are available for training.

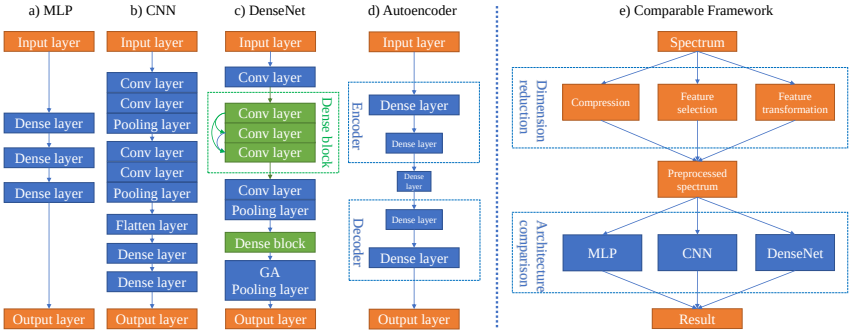


Figure 6.2: Illustration of different network architectures and the framework of our approach. To set up a new proper baseline for large-scale element identification, we conduct extensive tests on MLPs, CNNs and DenseNets with varying hyperparameters. Besides, compared to other dimension reduction methods (compression and feature transformation), our feature selection-based approach outperforms with better performance and interpretability.

6.3 Comprehensive element identification with interpretable dimension reduction

In this section, the overall framework for an efficient comprehensive element identification with interpretable dimension reduction is proposed. As illustrated in Fig. 6.2 (e), following the same approach of Chapter 4, we first propose a baseline for large-scale spectral identification by evaluating different neural network architectures. Then, we adopt feature selection to reduce the data dimension for an efficient workflow. To demonstrate the effectiveness of this procedure, we compare our approach with other dimension reduction methods on both simulation data and measurement data from the industry.

6.3.1 Data preprocessing

Since we focus on the identification of elements but not the exact concentrations of elements, the absolute amplitude of spectra is irrelevant to our scope of work. Instead, information needs to be extracted to predict the presence/absence of

elements. Thus, we apply *l2-norm* to preprocess the spectral data $\mathbf{S}$ for better training performance.

$$X_i = \frac{S_i}{\sqrt{\sum_i S_i^2}} \tag{6.3}$$

6.3.2 Network architecture and evaluation metrics

In this chapter, we also evaluate MLP, CNN and DenseNet with varying architecture hyperparameters (e.g., number of layers, filters) on a large-scale dataset. As for performance evaluation, there are various metrics to evaluate neural networks on classification problems, as stated in Section 2.2.1.3 of Chapter 2. In this chapter, we use *F1 score* to examine the balanced prediction performance because the spectral analysis task is a multi-label multi-class problem.

6.3.3 Feature selection for dimension reduction

In real-world applications, high-dimensional data is common but hard to handle. For our scope of work, the spectral data also contain much redundant information as shown in Fig. 6.1. To achieve an interpretable result for dimension reduction, previous work usually manually selects ROIs, which is not feasible for large-scale applications. To automate this process, we choose our work *watermelon* (proposed in Chapter 5) to select the most important features.

6.3.4 Other dimension reduction approaches

To obtain a solid evaluation result of dimension reduction on spectral data, we also consider the usage of other feature reduction techniques to reduce the input data dimension: compression and feature transformation.

6.3.4.1 Compression

As mentioned in Sec. 6.1, related work [12] in this domain adopts simple data reduction techniques to reduce the input size. Since our work is already beyond the scope of work of previous works, a direct comparison with them under our task setting is not possible. Therefore, we test a similar method to serve as the baseline for a fair comparison. Here we utilize a straightforward approach to reduce the input dimension by averaging consequent k data points evenly, which is defined by Eq. 6.4.

$$X'_j = \frac{\sum_{j=ki}^{ki+k-1} X_j}{k}, i \in 0, 1, 2, ...[\frac{1024}{k}] \tag{6.4}$$

In this way, we can compress the spectrum by a factor of k without the need for prior knowledge, which is superior compared to previous work.

6.3.4.2 Feature transformation

Another class of approaches is transforming the original high-dimensional features into low-dimensional latent ones while preserving as much information as possible. Previous works utilize a well-known classic method of principal component analysis (PCA)[1, 18]. Similarly, in this chapter, we adopt a more popular neural network-based method, i.e., Autoencoder, due to its excellent performance on feature transformation. The architecture of a plain vanilla Autoencoder is described in Fig. 6.1(d) with a similar structure as an MLP.

6.4 Experiments and results

6.4.1 Experiment setup

In order to identify up to 28 elements under a large-scale task setting, we generate 140k simulation spectra to train our neural networks and 40k spectra as the test dataset. The number of elements and the concentrations of elements are randomly chosen for each spectrum.

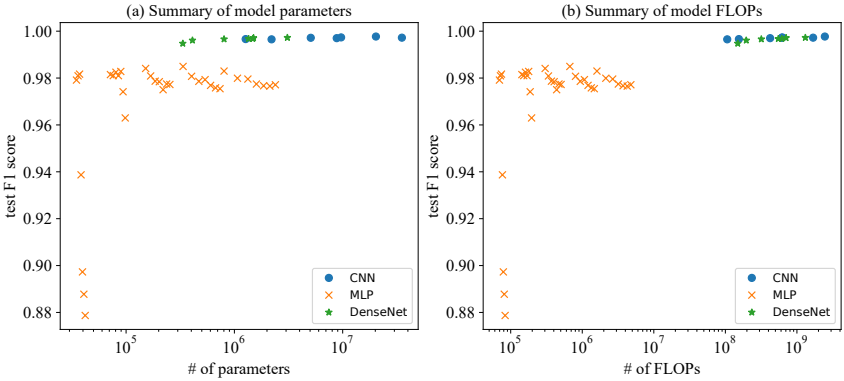


Figure 6.3: Performance of MLP, CNN and DenseNet with different network sizes, the x-axis is logarithmic. CNNs and DenseNets outperform MLPs by a noticeable margin.

Besides, to determine the feasible network size, we test each network architecture with various configurations. The MLPs consist of 2 to 8 layers, while all the layers in one network have the same neuron size, varying from {32, 64, 128, 256, 512}. Therefore, there are 35 MLP candidates in total. For CNNs, following the architecture shown in Fig. 6.2 (b), seven networks with different filter sizes (from 32 to 128) and neuron sizes (from 128 to 512) are evaluated. As for DenseNet, based on the original DenseNet-40 architecture, we also train seven networks with varying filter sizes and growth rates.

Table 6.1: Summary of network performance

Network	F1 Score	#parameter	#FLOPs
MLP	0.9828	1.60M	3.19M
CNN	0.9976	20.27M	2456.00M
DenseNet	0.9967	1.35M	613.60M

The networks are all trained using an Adam optimizer with a learning rate of 1e-4 for 120 epochs, where the learning rate decreases by a factor of 0.1 after 60 and 90 epochs. Besides, a weight decay of 1e-4 is applied and *binary cross-entropy* is used as the loss function. Since this is a multi-label multi-class problem, the *sigmoid* activation function is applied at the output layer.

6.4.2 Classification results on simulation data

The comparable results of the network candidates on the test dataset are visualized in Fig. 6.3. The test F1 scores of all the networks (Y-axis) are plotted against the parameter sizes and FLOPs sizes (X-axis) in (a) and (b), respectively. It shows that as the network size increases, the performance improves until a later saturation stage. Besides, CNNs and DenseNets have generally achieved better test F1 scores than MLPs, even with similar parameter sizes.

For the following part of the work, we choose one network for each network architecture considering its performance and resource demand. An overview of the networks is summarized in Table 6.1. The MLP consists of 8 hidden layers with 512 neurons in each layer. The CNN has a structure of {2x(Conv layer with 128 filters), Maxpooling layer, 2x(Conv layer with 256 filters), Maxpooling layer, 2x(Conv layer with 256 filters), Maxpooling layer, Flatten layer, 3x(Dense layer with 512 neurons)}. As for DenseNet, the DenseNet-40 with a growth rate of 12 is chosen.

Regarding the network size, we note that MLP and DenseNet have a similar level of parameter number at about 1.5 million. At the same time, CNN is significantly

larger in terms of both the number of parameters and FLOPs (20M and 2456M, respectively).

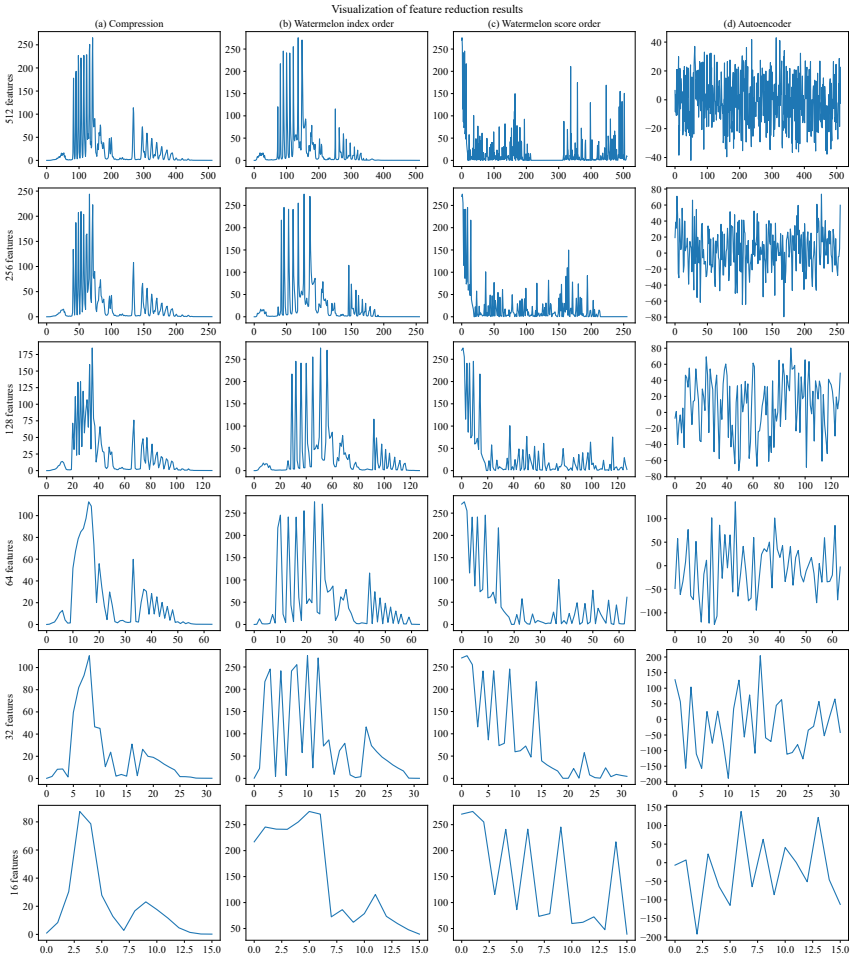


Figure 6.4: Visualization of reduced features by (a) compression, (b) and (c) watermelon, (d) Autoencoder. In (b), the features are ordered by their original indices and in (c), the most important features are the most left ones. Both compression and watermelon preserve the shape of spectra, and watermelon performs better, especially when the number of features is limited. In contrast, Autoencoder produces latent features that are hard to interpret.

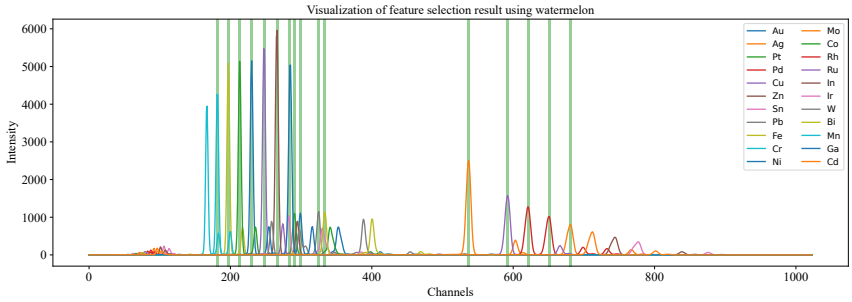


Figure 6.5: First 16 selected features by watermelon, the major peaks are automatically detected.

6.4.3 Comparison of dimension reduction methods

To obtain less demanding networks and better interpret the results, we further examine the effects of different feature reduction methods. For compression, we use $k \in \{2, 4, 8, 16, 32, 64\}$ which leads to a final input spectrum size of $\{512, 256, 128, 64, 32, 16\}$. This approach is straightforward, intuitive and can preserve the shape of spectra with a reasonable choice of k. To illustrate the results, the compressed spectra of the mixture of all relevant elements are displayed in Fig. 6.4(a). Note that the most significant peaks are still visually distinguishable until $k = 8$, i.e., with 128 features remaining. However, as k grows, a large portion of information gets lost due to the fusion of major peaks.

Similarly, for feature selection, we use watermelon to select the most important $\{512, 256, 128, 64, 32, 16\}$ features using the original spectra dataset. To show the efficiency of this method, Fig. 6.5 depicts the 16 most important features in green lines. Note that most major peaks are selected as ROI automatically and precisely, which outperforms the traditional ROI selection approaches that require prior knowledge and cost of time. The results of all selected feature subsets are visualized in Fig. 6.4(b) and (c). In (b), the selected features are ordered by their original indices and in (c), the feature selection scores are used where the most left features are the most important features.

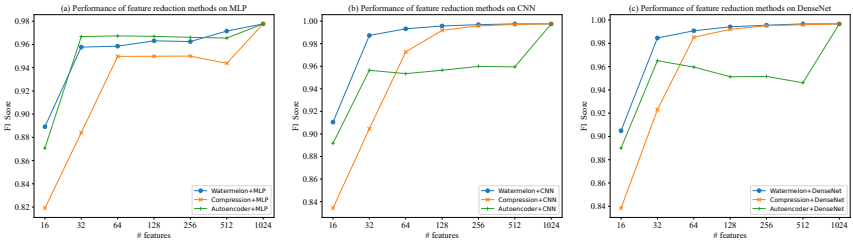


Figure 6.6: Performance of MLP, CNN and DenseNet using different feature reduction methods. Watermelon achieves overall best performance and Autoencoder is feasible at a low number of features, whereas compression is more suitable for higher feature sizes.

In (b), we notice that compared to compression, watermelon can preserve more information with a much sharper spectrum structure, especially as the number of remaining features decreases. For the target final feature numbers of 32 and 16, watermelon keeps twice the peaks as compression does. In (c), the peaks with the highest intensity are firstly selected, which agrees with Fig. 6.5.

As for feature transformation using Autoencoder, the number of neurons in the bottleneck layer is also $\{512, 256, 128, 64, 32, 16\}$, respectively. The latent features are shown in (d) and in contrast to the other two approaches, it is hard to interpret the output of the Autoencoder.

We re-train the networks with the same configuration based on the reduced features. The comparable results are illustrated in Fig. 6.6. In each subplot, the performance of three feature reduction methods is tested with MLP, CNN and DenseNet, respectively.

For CNN and DenseNet, we find that watermelon outperforms the other two methods in all cases, especially when the number of features is low. This agrees with the visualization of reduced features where watermelon effectively preserves the most important features. Similarly, the difference between watermelon and compression convergences as the number of features grows to 128 since starting from 128 features, compression can also output spectra with distinguishable peaks. In contrast, the performance of Autoencoder saturates at an early stage, leaving a large F1 score gap with the other two methods. When examining the results

with MLP, Autoencoder outperforms others in most cases, while watermelon is still superior with the least and most features (16 and 512). Besides, compression always leads to the worst F1 score.

We consider the main cause of different evaluation results as the existence/absence of spatial information. Compression and watermelon maintain the spectrum structure while Autoencoder does not. Moreover, CNN and DenseNet can take advantage of spatial information of input data and build high-level features that are utilized to make the final prediction, whereas MLP does not have such ability. Therefore, the combination of spatial information capable networks (CNN&DenseNet) and methods (compression&watermelon) deliver the best overall performance. In comparison, although Autoencoder is preferred for MLP, the F1 scores are generally lower than others.

6.4.4 Evaluation on real measurements

In the previous sections, we evaluate the networks on simulation data. To assess the performance of our approach in real-world scenarios, we collect 15 complex samples measured on different measurement devices that are summarized in Table 6.2, where each sample is measured for $\#repeat$ times. Theoretically, elements with lower concentrations are harder to identify. Thus, to provide a better overview of the distribution of concentrations, the elements are marked as *trace* ($<$1%), *very low* (1-10%), *low* (10-30%), *medium* (30-70%), *high* (70-90%) and *very high* ($>$90%), respectively. Since the neural networks predict the presence of all 28 elements, whereas there are only 13 elements involved in the measurement data, it will lead to an F1 score of zero for the elements that are not present at all. To properly assess the performance of the models on real data, we calculate the accuracy instead. Then, the accuracies of MLP, CNN and DenseNet using varying numbers of features are presented in Table 6.3. Overall, our approach performs also very well on measurement data with an accuracy of up to 0.9911.

Across different combinations of neural networks and feature reduction techniques, we find that similar to the results on simulation data, CNN-based models

Table 6.2: Summary of measurement spectra. The elements are marked as *trace* (<1%), *very low* (1-10%), *low* (10-30%), *medium* (30-70%), *high* (70-90%) and *very high* (>90%)

Sample	# repeat	Element												
		Au	Ag	Pt	Cu	Zn	Sn	Pb	Fe	Ni	Mn	Pd	In	Ir
sample_1	10				high	very low		very low	trace	trace				
sample_2	10				medium	medium	trace	very low	trace	trace				
sample_3	10				medium	medium	trace	trace	trace	trace				
sample_4	10				high	very low	very low	very low	trace	trace				
sample_5	10				medium	trace	trace	trace	trace	medium	trace			
sample_6	10				high	trace	very low	very low		very low				
sample_7	10				high	trace	very low	low	trace	trace				
sample_8	200	medium	low		low									
sample_9	200	high	very low		low									
sample_10	200	medium	low		medium	low								
sample_11	200	medium	very low		medium									
sample_12	200	very high	very low		very low									
sample_13	10	very low	medium			very low	very low					medium	very low	
sample_14	10	medium	low	trace	low	trace						very low	trace	
sample_15	10	high	low	very low		very low						very low	trace	trace

(CNN and DenseNet) outperform MLP on real data by a large margin in most cases.

A major difference between simulation and measurement data is that on measurement data, the accuracy does not decrease consistently as the number of features decreases. It forms a quasi U-shape curve and the best accuracy is gained with only 32 features selected by watermelon (also see Fig. 6.7 for illustration). To explain such phenomena, we consider the following possible causes:

- Different from simulation data, random noise will be captured during the measurement and the signal-to-noise ratio (SNR) is relatively high at low-intensity channels. Therefore, we observe the performance difference between simulation and measurement data. Until the number of features decreases to a certain degree, the useful information and the noises are proportionally reduced. Since there are fewer features available, the performance drops as well.
- As the compression rate gets higher, most low-intensity features are filtered as shown in Fig. 6.4 and the overall SNR is thus improved. Therefore, neural networks will not learn from noises that may lead to overfitting problems. As a result, performance is improved.

Table 6.3: Summary of comparable results using different dimension reduction techniques, the accuracy on real measurement data using different feature sizes is presented.

Method	Compression			Watermelon			Autoencoder		
# feature	MLP	CNN	DenseNet	MLP	CNN	DenseNet	MLP	CNN	DenseNet
1024	0.9394	0.9813	0.9770	0.9394	0.9813	0.9770	0.9394	0.9813	0.9770
512	0.9438	0.9748	0.9717	0.9629	0.9755	0.9661	0.9816	0.9772	0.9777
256	0.9484	0.9810	0.9881	0.9417	0.9618	0.9673	0.9806	0.9767	0.9862
128	0.9664	0.9893	0.9842	0.9520	0.9717	0.9861	0.9762	0.9841	0.9880
64	0.9644	0.9892	0.9884	0.9443	0.9886	0.9884	0.9874	0.9878	0.9873
32	0.9724	0.9591	0.9666	0.9592	**0.9911**	0.9867	0.9882	0.9856	0.9874
16	0.7687	0.8435	0.8265	0.9569	0.9561	0.9398	0.9748	0.9711	0.9769

- For watermelon, the accuracy is lower than others at the middle point. It is mainly because its algorithm tries to gather as many good but also uncorrelated features as possible. Therefore, it is less robust than others where useful, but correlated features contribute to a stable result. Still, it achieves the best performance with only 32 features.
- Last but not least, since the measurements only contain a subset of all the elements that neural networks can identify, the performance might be different on more complex spectra, where the number of different samples is higher and all possible elements are covered.

6.4.5 Comparison of model sizes

In Section 6.4.3, different feature reduction techniques are introduced to reduce the overall resources needed. In Fig. 6.7, we compare the model sizes concerning their performance using CNNs as an example. The accuracies (Y-axis) of models using different approaches against the number of model parameters (X-axis, logarithmic scale) are plotted in (a), while the number of FLOPs is shown in (b). The number of parameters indicates the required physical size to store the network, and FLOPs represent the computational cost needed for the network execution.

From the figure, we note that dimension reduction methods can significantly reduce the model sizes with even better identification results. Using 32 features

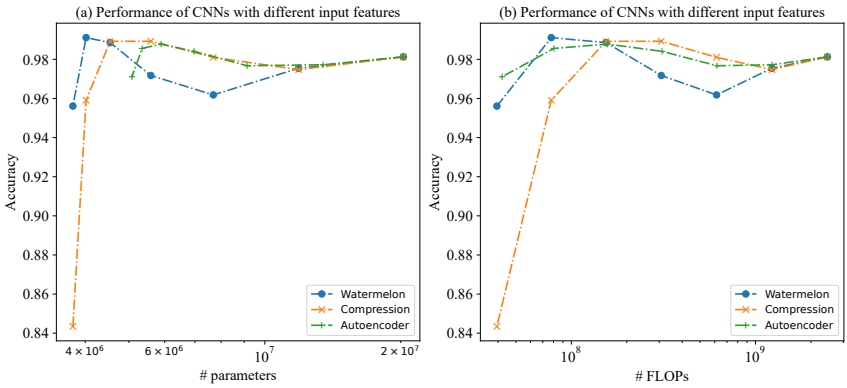


Figure 6.7: Performance and size comparison of CNNs on real measurements, x-axis uses the logarithmic scale. Similar to simulation results, watermelon still obtains the best accuracy with a lower model size.

selected by watermelon, the number of parameters is only 19.82% of the original model and the number of FLOPs is reduced to 3.17%. Besides, the parameter and FLOPs size of the Autoencoder approach is always larger than compression and watermelon because except for the prediction network, Autoencoder itself also introduces extra parameters and FLOPs since it is also a neural network.

Overall, our approach with watermelon achieves the best accuracy on real-world measurement data with significantly reduced data dimension, computational, and storage costs.

7 Efficient network pruning via feature selection

The findings in Chapter 6 show that for the spectral analysis tasks, the DNNs can achieve excellent performance with only a small portion of the input data, leading to an attractive solution for the practical deployment in industrial applications, where the computational resources are expensive and not unlimited.

This inspires ongoing thinking of what else could be done to further optimize the whole framework. Note that in this workflow (input-model-output), the feature selection focuses on the input part and greatly reduces the data volume that has to be processed by the model. A further step, after the data are given, is to investigate whether the cost of the model itself can be improved. DNNs are known to be powerful but also computationally intensive, therefore, an enormous saving may be achieved if the size of DNNs can be slimmed without significant performance degradation.

This chapter, similar to Chapter 5 where the watermelon feature selection is introduced, proposes a network pruning method that compresses the network size for efficient execution. Based on the watermelon method, this innovative approach prunes a network from a feature selection perspective. Besides, to demonstrate its generalization ability, the approach is proposed as a general method for DNNs and is not bounded by the spectral analysis domain. Experiment results on large-scale image classification tasks, which are the golden benchmarks in the domain of machine learning, validate its efficiency, and in Chapter 8, it will be examined on the spectral data to focus on the specific topic of spectral analysis.

7.1 Introduction

In the past few decades, deep CNNs have been successfully utilized in many application fields, such as image classification [34], semantic segmentation [80], object detection [33] and natural language processing [81]. Large-scale datasets [82] promote the development of deeper and larger CNN networks from classic AlexNet [34] and VGGNet [32] to ResNet [83] and DenseNet [36] with up to billions of parameters and hundreds of layers.

Although these networks are powerful, they demand a large memory footprint, considerable hardware storage and huge computational resources. Thus, the acceleration of CNNs using network pruning [84, 85] has become a focus in the past years. Network pruning approaches can be empirically divided into two groups: *unstructured* and *structural* pruning. Unstructured methods prune the weight parameters in the neural network and can achieve a very high compression rate [86–89]. However, since the network structure does not change and the sparse weights have to be stored with extra overhead, which often requires specific libraries and/or hardware, the actual computational cost is not significantly reduced compared to the impressive compression rate of parameters.

In contrast, structural methods directly remove neurons, filters and/or layers to gain a more compact network. As the network gets slimmed, fewer FLOPs are required for the network training and inference, which is superior to unstructured methods. These approaches typically use certain criteria to select less important parts of the network and among them, the utilization of regularization terms is an active sub-domain. Different norms ($l_{1/2}$-norm, l_1-norm and l_2-norm) are applied directly or indirectly to the layer weights to train a sparse network and redundant filters are pruned [90–93].

Besides, some approaches assess the importance of filters based on statistical information, such as l_1-norm of weights [94], average feature map ranks [95], expected output values [96] and gradient-based scores [89, 97]. Another group of network pruning methods try to minimize the reconstruction error after pruning to obtain a good sub-network based on various concepts, including l_1-norm and

LASSO [92], second-order weights information [98,99] and greedy strategy [100]. Moreover, techniques from other domains are also introduced to prune deep CNNs: meta learning [93, 101], reinforcement learning [102] and generative adversarial learning [103] to name a few. Note that some methods do not belong to only one category, for instance, FilterSketch [98] uses second-order information and aims to minimize the reconstruction error.

Many of these algorithms need to train a network from scratch or retrain the network during pruning, which is time-consuming, especially for large networks on large datasets. Additionally, apart from reducing the reconstruction error, numerous approaches [93, 101, 104–107] select sub-networks based on their performance (e.g. accuracy, loss) after pruning. However, we find that pruning with huge reconstruction errors or a zero post-pruning prediction ability is feasible. Moreover, although various methods assess the importance of filters based on certain rules, few of them explicitly utilize feature selection methods. To our best knowledge, only NISP [108] adopts a feature selection method but it also minimizes the reconstruction error.

Therefore, in this chapter, we propose a network pruning method from a pure feature selection perspective based on *watermelon* (introduced in Chapter 5). We show that ignoring the huge reconstruction error can also lead to excellent pruning results by preserving important internal high-level features. Inspired by the common brief that the Conv layers serve as feature extractors of input data, we evaluate the outputs of the last Conv layer as latent features and perform feature selection on it. Then, backward from the last Conv layer to the first Conv layer, each layer determines which filters of the previous layer will be removed based on their redundancy. Afterward, if there are other following dense layers except for the last prediction layer, feature selection will also be applied to select the neurons to be pruned. Finally, we retrain the network to retain the prediction performance.

We evaluate our method on CIFAR10 [109] and ImageNet [82] using VGGNet (VGG-16) and DenseNet (DenseNet40, DenseNet121). We achieve state-of-the-art performance without the need for training networks from scratch or training to prune. We show that our method is intuitive, straightforward, and effective on

different CNN architectures. Also, we provide a new view of network pruning: find important features while ignoring the reconstruction error.

In summary, the main contributions of this chapter are twofold:

1. We propose a network pruning method from a pure feature selection perspective, and our approach is competitive with other state-of-the-art methods;

2. Different from many approaches that minimize the reconstruction error, we show that it is not an essential criterion by achieving excellent results with significant reconstruction errors.

7.2 Related works and preliminary

Early works on network pruning are mainly unstructured methods. A representative work [86] zeros out weights with very small magnitudes, which leads to a very sparse weights matrix. Although it achieves an impressive parameter compression rate, the actual computational cost is not significantly reduced due to the constant network topology. [87] prunes parameters based on [86] and retrain the network with the corresponding initial weights (called winning ticket). A following work [96] of structural pruning prunes filters/neurons with the least expected post-activation values and retrains the networks with reinitialization.

As for other structural pruning methods, network slimming [90] applies scaling factors to filters in Conv layers and uses l_1-norm regularization to train the network. Afterward, filters with small scaling factor values are removed. Similarly, [91] prunes an untrained network with a custom regularization term. SCOP [110] generates the knockoff data from training data, which have a similar distribution to training data but no ground-truth information. After training, it prunes filters with high scaling factors on the knockoff data. CCPrune [111] uses l_1 regularization to both the Conv layer and batch-normalization (BN) layers. Besides, it jointly evaluates the importance of the filters based on the weights of Conv layers and the

scaling factors of BN layers. FETS [112] measures the influence of a filter based on the average output. It applies group Lasso to force the FLOP-relevant sparsity regularization terms to be small during training, thus removing less important filters.

Similarly, SSL [113] utilizes group Lasso regularization to prune network structures. [92] uses l_1-norm and solves Lasso to minimize the reconstruction error and prune unimportant filters. ThiNet [100] iteratively prunes filters by minimizing the reconstruction error of layer output based on a greedy strategy. NISP [108] performs feature selection on the final response layer and propagates the "importance score" to each layer, aiming to minimize the reconstruction error of the whole network. [114] leverages the correlation between filters in the same Conv layer as regularization terms and trains to push strongly correlated filters to highly correlated. Then, redundant filters are removed.

Different from training to prune, some approaches prune a network before training. SynFlow [115], SNIP [89] and GraSP [97] use gradient-based scores to prune networks before training. Similar to these property-based methods, OBD [116], OBS [117] and L-OBS [99] prune networks based on the second-order derivatives. HRank [95] assesses the quality of filters based on the average rank of feature maps of the filters. [118] uses Fischer information to evaluate the importance of filters and prune the least important ones. [119] estimates the distribution of channel saliency in the BN layers using variational technique and thus prunes redundant filters.

Combined with techniques from other domains, EPruner [120] applies affinity propagation to select essential filters. [121] builds graphs to assess the redundancy of layers and prunes filters with the smallest absolute weights. FilterSketch [98] uses second-order information of layer weights and prunes the network with the frequent direction method.

A meta-learning-based approach MetaPruning [101] trains a meta-network to predict the weights of given network architecture. To search for the best sub-network, an evolutionary procedure is applied. One following work DHP [93] develops a hypernetwork with latent vectors to generate sub-network candidates

and uses proximal gradient with l_1 regularization for the network search to get rid of the usage of evolutionary algorithms. [107] introduces $l_{1/2}$ norm to the BN layers during training for sparse learning and applies a genetic algorithm to select the best sub-network according to their fitness scores.

Parallel to evolutionary algorithms, LWP [102] utilizes a reinforcement learning-based approach to select good sub-networks with high rewards. GAL [103] applies generative adversarial learning to learn a new network with soft masks that can generate similar output to the original network. l_1 regularization is used to make the new network sparse and ready to be pruned. [105] directly trains a neural network to predict the accuracy of sub-networks and find an optimal sub-network. EagleEye [104] adapts the BN layers to estimate the post-pruning accuracy of sub-network candidates by removing filters with small l_1-norm. [106] uses a polynomial to estimate the accuracy of network candidates. Besides, it prunes a network from three dimensions (filter, layer and input).

[108], [114] and [121] are the most similar works to our approach. However, our method is different and new for the following reasons:

1. Although [108] also utilizes a feature selection method on the response layer, our concept differs. We show that keeping important and less redundant latent features leads to good pruning results even with a huge reconstruction error, which is the main objective function to be minimized in [108] as well as many other methods mentioned above.

2. While [114] assess the redundancy of filters in the **same** layer (so does [121]) using Pearson correlation coefficient, we use Spearman's rank correlation coefficient to find redundant filters in the **previous** layer based on the filters in the current layer. In short, we find redundancy from a feature selection perspective but not in a statistical information-based way. Besides, Spearman's coefficient is preferred in terms of finding monotonic relationships, which can better determine the feature redundancy. Also, [114] needs to retrain the network during pruning, whereas ours does not.

7.3 Model pruning via feature selection

Our approach performs filter pruning on conv layers and neuron pruning on dense layers from a feature selection perspective. Given a CNN network, for the cth conv layer l_c, $c \in$ [1,# conv layers], we denote $W_c \in \mathbb{R}^{k \times k \times f_{c-1} \times f_c}$ as the kernel weights and $X_c \in \mathbb{R}^{w_c \times h_c \times f_c}$ as the output feature maps. Here, $k \times k$ is the kernel size, f_c is the number of filters in cth conv layer, w_c and h_c are the weight and height of the feature map respectively. For simplification of notation, we ignore the bias weights.

7.3.1 Feature selection score

7.3.1.1 Neuron score

For all dense layers and the last conv layer (or the following flatten layer), their output can be observed as features F given training dataset D with L classes. We use watermelon as our feature selection criterion to evaluate these features, which estimates the Bayes error rate on each feature and penalizes feature redundancy while rewarding feature relevance. It follows a greedy search to add the next best feature into the selected feature subset N incrementally. The corresponding *Neuron score* $Q(N; F_n)$ (lower is better) given a new neuron n is defined as Eq. 7.1, where the estimated Bayes error rate $BER(c_i, c_j|F_n)$ with respect to class i and j is dynamically adjusted according to the redundancy and relevance among features (defined as $BER^*(c_i, c_j|F_n)$ in Eq. 7.2). The Spearman's rank correlation coefficient and normalized mutual information are applied to calculate such relationships, and an activation function AF is introduced to compensate for the effect of weak relationships. For more details, please refer to Chapter 5.

$$Q(N; F_n) = \sum_{F_k \in N \cup F_n} \sum_{i,j \in L, i \neq j} BER^*(c_i, c_j|F_k) \tag{7.1}$$

$$BER^*(c_i, c_j|F_n) = penalty(reward(BER(c_i, c_j|F_n))) \tag{7.2}$$

7.3.1.2 Filter score

Because of the massive number of conv layers in deep CNNs, performing feature selection on each conv layer is time-consuming. Thus, we follow the philosophy of watermelon to detect redundancy using Spearman's rank correlation coefficient in a data-free way.

For each filter in cth conv layer, its weights have a size of (k, k, f_{c-1}) and we calculate the Spearman's rank correlation coefficient (defined as Cor in Eq. 7.4) of weights corresponding to every two filters f_i, f_j in (c-1)th conv layer. For the convenience of math, we reshape the weights into $(k * k, f_{c-1})$. Then we set a threshold thr_{cor} and use a binary step activation function BAF, as defined in Eq. 7.3, to indicate whether two parameter sets are significantly correlated. A significant correlation means that one filter in the current conv layer uses highly correlated parameters to process the output of two previous filters. Therefore, these filters are considered redundant.

$$BAF(x, thr) = \begin{cases} 1 & \text{if } x \geq thr \\ 0 & \text{if } x < thr \end{cases} \tag{7.3}$$

$$Q_{ic} = BAF(\max_{i,j \in f_{c-1}, i \neq j} |Cor(F_i, F_j)|, thr_{cor}) \tag{7.4}$$

$$Q_i = \sum_{c=1}^{f_c} Q_{ic} \tag{7.5}$$

The *Filter score* Q_i of ith filter in previous conv layer l_{c-1} is thus defined as the accumulation of individual score Q_{ic} of all the filters in current conv layer l_c.

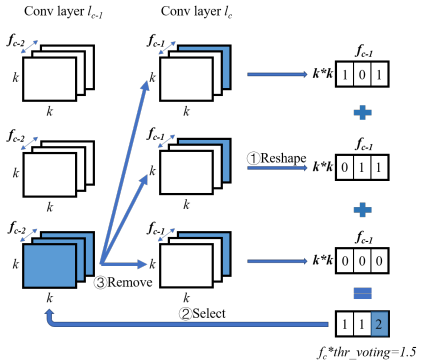


Figure 7.1: Pruning strategy of conv layers using filter scores. For illustration purpose, filter size $f_c, f_{c-1}, f_{c-2} = 3$ and $thr_{voting} = 0.5, thr_{cor} = 0.7$. 1) For conv layer l_c, the weights of each filter are reshaped from (k, k, f_{c-1}) to $(k * k, f_{c-1})$ and Q_{ic} is calculated. 2) The filters in l_{c-1} will be marked (as blue) if $Q_i > f_c \cdot thr_{voting}$. 3) Marked filters in l_{c-1} and corresponding weights in l_c will be pruned.

7.3.2 Network pruning and fine-tuning

Based on the scores introduced in Section 7.3.1, we describe the general pruning process in this section. Since the conv layers function as high-level feature extractors, we begin the feature selection and network pruning from the last conv layer, which is usually connected to a flatten layer (VGGNet) or a pooling layer (DenseNet). For simplicity, we use the flatten layer for description.

The size of the flatten layer is $w_c * h_c * f_c$ since the last conv layer l_c it connects to has a size of (w_c, h_c, f_c). We define a global ratio threshold $thr_{sparsity}$ across all layers to determine a certain percentile of neurons or filters to be removed $(\#neurons \cdot thr_{sparsity})$. The neuron scores are calculated based on the flatten layer output given the training dataset to find and mask out redundant or irrelevant "features of features". Then, the filters in conv layer l_c will be removed as long as their mapped outputs in the flatten layer are all removed.

After that, based on the remaining filters in the last conv layer l_c, we calculate the filter scores to prune filters of the previous conv layer l_{c-1} given $thr_{sparsity}$. To exploit the advantage of feature selection and prevent filters with positive

contributions from being over-pruned due to the fixed threshold, we further use a threshold thr_{voting} to determine whether a filter is to be pruned or not. A filter in conv layer l_{c-1} will be pruned only if 1) its filter score Q_i is greater than $thr_{voting} \cdot f_c$ and 2) ratio threshold $thr_{sparsity}$ is not reached. By repeating these steps as shown in Fig. 7.1, we can prune all conv layers backward.

Moreover, the flatten layer is also the starting point of the following dense layers. First, because the flatten layer is partially masked out, the weights of the next dense layer should also be pruned. Then, similar to the flatten layer, the outputs of each dense layer are evaluated based on neuron scores, and neurons with the highest scores will be removed entirely. Finally, the network will be retrained to retain its performance. The whole procedure is summarized in Algorithm 1.

7.3.3 Handling cross-layer connections structure

Till now, we can prune classic plain CNN architectures like AlexNet and VGGNet, where pruning of one layer only affects its neighborhood layer. However, DenseNet connects each layer to every other layer by concatenating the outputs of conv layers within a dense block. Besides, it replaces the flatten layer with a global average pooling layer, and there is only one dense layer at the end of the network. To adapt for DenseNet, we modify our method as follows:

first, we also start from the last conv layer and similar to the flatten layer, feature selection will be performed on the pooling layer. Since the output of the pooling layer is the concatenation of all filters of all conv layers in the last dense block, we can easily apply a direct mapping from the pooling layer output to each filter. In this way, we can prune a whole dense block at once. For the conv layer in transition blocks, it is not possible to calculate the redundancy because 1×1 kernel is used, and the weights of the conv layer have the shape of $(1, 1, f_{c-1}, f_c)$. Instead, we evaluate the l_1-norm of weights of the current conv layer l_c and prune the corresponding filters of the previous conv layer l_{c-1} with the smallest values, which means that they are not effectively used by the current conv layer and therefore are considered as redundancy (see Fig. 7.2). Repeating this, all the

Algorithm 2 Watermelon model pruning

Require: Network M, $D \in R^{n \times d}$ and labels $Y \in R^n$, $thr_{spar}, thr_{cor}, thr_{voting}$

$Q \leftarrow \emptyset$, $N \leftarrow \emptyset$, $F \leftarrow \#neurons$ for flatten layer
$X = D \times M_{\text{till\_flatten\_layer}}$
while $\#N < \#neurons \cdot thr_{spar}$ **do**
  for all $F_n \in F \setminus N$ **do**
    calculate $Q(N; F_n)$ given X and thr_{cor}
  end for
  $Q \leftarrow Q(N; F_n)$, $N \leftarrow N \cup \underset{F_n \in F \setminus N}{\operatorname{argmin}} Q(N; F_n)$
end while
mask out N in flatten layer, update last conv layer l_c
for all $l_i \in [c, c-1, c-2, ...2]$ **do**
  init $C \leftarrow \emptyset, Q \leftarrow \emptyset$
  for all $j \in [1, 2, ..., f_{i-1}]$ **do**
    calculate $Q_j, Q \leftarrow Q \cup Q_j$
  end for
  while $\#C < f_{i-1} \cdot thr_{spar}$ **and** $\max_j Q \geq f_i \cdot thr_{voting}$ **do**
    $C \leftarrow C \cup \arg\max_j Q$, $Q \leftarrow Q \setminus \arg\max_j Q$
  end while
  remove filters C in layer l_{i-1}
end for
for all dense layers **do**
  update weights according to previous layer
  repeat step 1-8 to remove N neurons
end for
retrain the network
return M

conv layers can be pruned backward. As for the dense layers (if applicable), the approach is the same as mentioned in Section 7.3.2.

For DenseNet-B and DenseNet-BC, the additional bottleneck layer with 1×1 convolution is applied before each 3×3 conv layer to cut down the feature maps size, which can be pruned using the same mechanism as for transition layers.

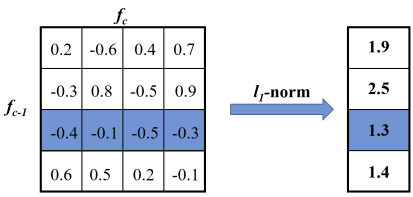


Figure 7.2: Illustration of pruning on conv layer with 1×1 kernel size. The weights have a shape of $(1, 1, f_{c-1}, f_c)$ and we reshape the weights to (f_{c-1}, f_c). l_1-norm will be calculated and filters with the smallest values will be pruned.

7.4 Experiments

7.4.1 Experiment setup

We empirically evaluate our approach on a variant of VGGNet and DenseNet40 with CIFAR10. As for the large-scale dataset ImageNet, we use DenseNet121. We implement our approach in Tensorflow [39] and adopt the original implementation of watermelon feature selection from [122]. For efficient computation, we utilize the simplified version watermelon-B.

To compensate for the effects of different frameworks and make a fair comparison with other methods, we adopt the same data augmentation and hyperparameter configurations if possible. As evaluation metrics, we report the results with relative accuracy drop, compression rate of parameters and FLOPs. Since different methods use different compression schema and target compression rates, it is hard to compare the performance due to different network sizes. To make the results more comparable, we define *Accuracy/Parameter Drop Ratio* and *Accuracy/FLOPs Drop Ratio* as the accuracy drop divided by the parameter drop or FLOPs drop as our additional evaluation criteria, which is helpful to compare different approaches with similar but different compression rates.

7.4.1.0.1 CIFAR10 CIFAR10 dataset consists of 60000 32×32 color images in 10 classes, of which 50000 are training images and 10000 are test images. On

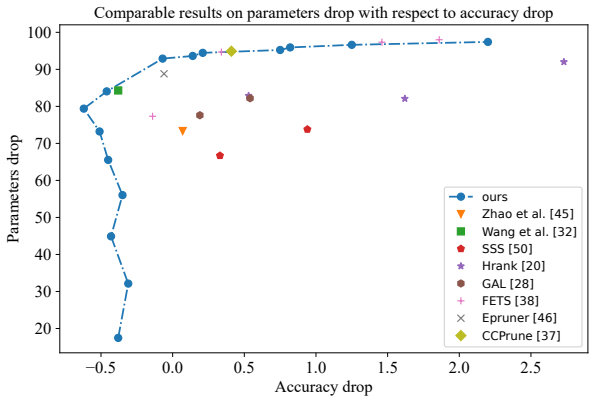


Figure 7.3: Pruning results of our approach at different ratios compared to other SOTA methods using VGG-16 on CIFAR10. Points at top left are better than those at bottom right.

CIFAR10, we test a variant of VGG-16 adopted from [90,94] for a fair comparison. The VGG-16 has only two dense layers (512 and 10 for 10 classes), and we use the same data augmentation scheme by shifting and horizontally mirroring the images. To train the baseline model, we use 160 epochs with a batch size of 64. We set the initial learning rate to 0.1 and it decreases by $1/10$ after 80 and 120 epochs. SGD with 0.0001 weight decay and Nesterov momentum of 0.9 is used.

7.4.1.0.2 ImageNet ImageNet is a large-scale dataset of 1000 classes that contains 1.2 million training images and 50000 validation images. We also adopt the standard data augmentation, which scales the image size so that the small side has a pixel width of 256. Then we apply random crop and horizontal flip for training and only center crop for validation. The validation accuracy of the pre-trained model is 73.30%.

7.4.2 VGGNet on CIFAR10

To better compare to other methods, we prune the network at different ratios. The hyperparameters of our approach are kept constant and we use $thr_{cor} = 0.5$

and $thr_{voting} = 0.4$ for all our experiments. As for fine-tuning, the network is retrained for 40 epochs, where the learning rate reduces from 0.001 to 0.0001 after 20 epochs. We summarize our experiments in Tab. 7.1 and the results are listed in ascending order in terms of the FLOPs pruning ratios (column ΔFLOPs). Results show that our approach outperforms others in most cases with a smaller accuracy drop at different parameters and FLOPs compression ratios. To visualize the efficiency of our method, we plot the comparable results in Fig. 7.3 where the Y-axis and X-axis present the parameters drop ratios (ΔPar.) and accuracy drop ratios (ΔAcc.), respectively. In this figure, points at top left are superior to others, and our method builds a clear borderline that divides most of the other competitors into the bottom right group. Note that up to a parameter compression ratio of 92.9%, our approach always obtains accuracy improvement, which most likely results from removing less critical and/or highly redundant internal features. By doing so, the overfitting phenomenon (training accuracy is above 99.9% whereas the test accuracy is under 95%) is greatly improved and the pruned network can find a better local minimum with higher test accuracy.

Moreover, after the network is pruned, we find that the network has almost no prediction ability without retraining and behaviors like a fresh initialized network (e.g., accuracy is 0.1 for CIFAR10 with 10 classes), especially when pruning the conv layers. This is generally considered a bad signal by other approaches [92, 100, 108] where the reconstruction error is the primary objective function to be minimized. However, the network can recover its full power within only several epochs of retraining. We compare the test accuracy of a pruned network with pre-trained weights and a fresh initialized model with the same network structure. Both of them have 0.1 validation accuracy initially, but the pruned network achieves a good performance (to 91.27% accuracy) after one epoch of training, while the fresh network can only get convergence at a lower accuracy (about 78.69%) after 20 epochs. We believe that it is because the network has already learned internal features that are essential for the classification task. Due to the dramatically changed network topology, the output of the layers is shifted and/or biased and therefore leads to zero prediction power, which can be, however,

recovered in a very short time. From a feature selection perspective, it indicates that keeping essential internal features might be a key for network pruning.

7.4.3 DenseNet on CIFAR10 and ImageNet

To illustrate the efficiency of our method on modern CNNs, we also evaluate the compact yet very effective CNN architecture DenseNet on CIFAR10. We use DenseNet40 and prune it at different target ratios while skipping the first two conv layers. After pruning, we retrain the network for 40 epochs and reduce the learning rate from 0.01 to 0.001 after 20 epochs. The comparable results are shown in Tab. 7.1, and we manage to prune 62% of the parameters and 63% of the FLOPs with only a negligible accuracy loss of 0.54%. Moreover, the comprehensive comparison validates our method's competitive performance compared to other state-of-the-art methods.

Additionally, we also validate our approach on the large-scale dataset ImageNet with DeseNet121. We prune the network with various pruning ratios, and the first two conv layers stay unchanged. Afterward, we retrain the pruned network for 30 epochs with decreased learning rates of 0.01, 0.001 and 0.0001 for 10 epochs each.

The results in Tab. 7.1 show that for the complex classification task, our approach manages to achieve a better Acc./Parameters and Acc./FLOPs drop compared to other competitors in most cases, too. Likewise, we again observe a huge post-pruning reconstruction error (high loss and low accuracy) on the large and compact DenseNet architecture with 121 Conv layers. However, the final performance is still very competitive with other reconstruction error minimization-based approaches, which validates the concept of network pruning from a pure feature selection perspective.

Table 7.1: Comparable results on CIFAR10 (VGG-16, DenseNet40) and ImageNet (DenseNet121) at different pruning ratios. A negative value means an accuracy improvement after pruning. The results are sorted in ascending order w.r.t. FLOPs pruning ratios (ΔFLOPs) and our results are in bold.

Network	Method	Orig. acc	ΔAcc	ΔPar.	ΔFLOPs	ΔAcc./Par.	Δ Acc./FLOPs
VGG16 /CIFAR10	SSS[123]	93.96	0.33	66.70	36.30	0.0049	0.0091
	ours	93.44	-0.62	79.39	36.62	**-0.0078**	**-0.0169**
	Zhao *et al.*[119]	93.25	0.07	73.34	39.10	0.0010	0.0018
	GAL[103]	93.96	0.19	77.60	39.60	0.0024	0.0048
	SSS[123]	93.96	0.94	73.80	41.60	0.0127	0.0226
	Wang *et al.*[107]	93.78	-0.38	84.30	43.40	-0.0045	-0.0088
	GAL[103]	93.96	0.54	82.20	45.20	0.0066	0.0119
	FETS[112]	93.94	-0.14	77.32	50.00	-0.0018	-0.0028
	Wang *et al.*[91]	93.44	-0.19	-	50.00	-	-0.0038
	Hrank[95]	93.96	0.53	82.90	53.50	0.0064	0.0099
	ours	93.44	-0.07	92.90	55.51	**-0.0008**	**-0.0013**
	Hrank[95]	93.96	1.62	82.10	65.30	0.0197	0.0248
	ours	93.44	0.21	94.46	69.44	**0.0022**	**0.0030**
	CCPrune[111]	93.80	0.41	94.90	73.68	0.0043	0.0056
	FETS[112]	93.94	0.34	94.66	75.61	0.0036	0.0045
	EPruner[120]	93.02	-0.06	88.80	76.34	-0.0007	-0.0008
	Hrank[95]	93.96	2.73	92.00	76.50	0.0297	0.0357
	CFP[114]	93.49	0.26	-	80.36	-	0.0032
	ours	93.44	0.82	95.92	81.03	**0.0085**	**0.0101**
	CFP[114]	93.49	0.51	-	81.93	-	0.0062
	FETS[112]	93.94	1.46	97.33	83.33	0.0150	0.0175
	ours	93.44	1.25	96.62	85.75	**0.0129**	**0.0146**
	FETS[112]	93.94	1.86	98.00	87.50	0.0190	0.0213
DenseNet40 /CIFAR10	**ours**	94.44	0.05	22.85	22.20	**0.0022**	**0.0023**
	Liu *et al.*[90]	94.81	0.00	36.50	32.80	0.0000	0.0000
	GAL[103]	94.81	0.20	35.60	35.30	0.0056	0.0057
	ours	94.44	0.06	35.50	35.96	**0.0017**	**0.0017**
	Hrank[95]	94.81	0.57	36.50	40.80	0.0156	0.0140
	Zhao *et al.*[119]	94.11	0.95	59.67	44.78	0.0159	0.0212
	ours	94.44	0.31	47.76	46.81	**0.0065**	**0.0066**
	GAL[103]	94.81	0.31	56.70	54.70	0.0055	0.0057
	Liu *et al.*[90]	94.81	0.46	66.30	57.60	0.0069	0.0080
	FETS[112]	94.31	-0.30	68.25	59.68	-0.0044	-0.0050
	Hrank[95]	94.81	1.13	53.80	61.00	0.0210	0.0185
	ours	94.44	0.54	61.67	62.91	**0.0088**	**0.0086**
	GAL[103]	94.81	1.58	75.00	71.40	0.0211	0.0221
	ours	94.44	1.25	71.46	72.39	**0.0175**	**0.0173**
	FETS[112]	94.31	0.50	79.34	77.22	0.0063	0.0065
	ours	94.44	1.96	80.84	81.16	**0.0242**	**0.0241**
DenseNet121 /ImageNet	**ours**	73.30	0.16	30.43	30.20	**0.0053**	**0.0053**
	DBP[124]	75.01	6.93	66.00	37.00	0.1050	0.1873
	ours	73.30	0.96	44.17	44.35	**0.0217**	**0.0216**
	Wang *et al.*[106]	75.01	1.33	48.00	51.00	0.0277	0.0261
	Liu *et al.*[90]	75.01	1.43	51.00	51.00	0.0280	0.0280
	Zhang *et al.*[102]	74.65	1.84	35.21	54.67	0.0523	0.0337
	ours	73.30	1.71	56.60	56.28	**0.0302**	**0.0304**

8 Real-time low-cost spectral analysis

Chapter 6 demonstrates that the computational cost of DNNs can be reduced significantly via feature selection (proposed in Chapter 5) while providing an interesting insight into what input data are important to DNNs, which makes the DNN-based framework explainable to some extent. In addition, the network pruning method introduced in Chapter 7 shows that the size of DNNs can also be greatly compressed without noticeable performance drop.

Therefore, in the context of this dissertation, especially with a focus on the deployment in the real world, it is desired to apply these optimizations to the proposed DNN-based framework to enhance the performance, so that a spectral analysis task can be executed in real-time and at low cost. This is especially important for edge devices since they usually do not have enough computational and storage resources for large DNNs, yet are largely used in industrial applications due to the cost advantage over expensive GPUs.

This chapter integrates the results of Chapter 4, 5, 6 and 7 and thus makes the proposed framework accurate, efficient and robust. With a specific focus on edge devices, quantization techniques are also discussed to enable the smooth transition of academic results to real-world applications.

8.1 Introduction

Generally, although neural networks are powerful, they demand a considerable amount of memory footprint, computational resources as well as hardware storage. It is less of a concern when such models run on servers or dedicated workstations with powerful graphics processing units (GPUs). However, a huge expenditure on hardware is needed. Therefore, for the end users, the purchase of high-performance computers is an enormous cost in addition to the already expensive measurement device, which makes the DNN-based solution less attractive. Similarly, for industrial applications, the deployment of DNNs in the production lines contributes to a significantly higher fixed cost and a lower profit margin.

Besides, in real-world scenarios, on-site and/or on-device analysis is often required, where only limited computational power is available on edge devices. Meanwhile, real-time execution is also an essential factor for many industrial applications, e.g., quality control in mass production, where the analysis results should be given within a certain time interval. Therefore, a real-time low-cost solution for edge devices is required.

Previous related work in spectral analysis mainly focused on the application of neural networks to solve the identification and quantification tasks. However, to our best knowledge, none of them addressed the time and resources issues in real-world scenarios. Hence, in this chapter, we propose a first hybrid strategy for the comprehensive acceleration of network execution in spectra analysis. Our approach focuses on the following three major points:

1. Input data dimension: the computational cost of a neural network is largely influenced by the input data dimension, especially for the convolutional layers, which also demand the most resources. Besides, the input data size also affects the pre-processing and the data transmission in the whole workflow, which leads to a large latency overhead.

2. Network size: in general, a larger network can achieve better performance on a given task. However, the improvement will eventually saturate as the

size increases and enlarging the network will only result in redundancy. Therefore, a trade-off should be made between the network size and the performance.

3. Network cost: apart from the network size, the computational cost of a network also depends on its operation type and the target platform. By default, neural networks mainly utilize 32-bit floating-point operations for training and inference. Theoretically, the cost is accordingly reduced when using 16-bit floating point, 16-bit integer or even 8-bit integers. However, on the other side, the performance could degrade with lower precision operations. Therefore, an appropriate choice is crucial to the final overall results.

To address the issues mentioned above, our approach consists of three phases across the whole training process. First, before training, we perform feature selection on the spectral data to reduce the input data dimension without losing important information. Then, after training the network on the datasets, we further prune the network to obtain a slimmed network that requires fewer storage and computational resources while still maintaining its performance. Finally, the networks are quantized with low-bit operations and variables to gain extra acceleration. We conduct the feature selection and network pruning based on the methods proposed in Chapter 5 and Chapter 7. To show the feasibility of our approach, we evaluate CNNs on large-scale simulation and real measurement data. Moreover, to simulate real-world applications with edge devices, we conduct extensive experiments and compare the results on different target platforms. Results show that our approach achieves excellent performance on edge devices with significantly reduced network size and inference time. In the meanwhile, the performance is even slightly better on real data in most cases.

The main contributions of this chapter can thus be summarized as follows:

- To achieve a real-time low-cost network execution on edge devices, we are, to the best of our knowledge, the first to introduce network pruning & network quantization and propose such a hybrid approach in the broad

domain of spectral analysis, where the major aspects of the whole DNN-based pipeline are taken into consideration.

- We successfully demonstrate the effectiveness of our approach under real-world settings by innovatively reducing the input data size, network size and network cost. The significant speedup and cost-saving enable the deployment of DNNs on edge devices by achieving accurate analysis results in real time.
- We show that our approach is modular, easy to adapt, and robust in most cases. It thus sets up a new baseline framework and provides the best practices for all related domains of interest in the IoT and big data applications/deployments, where the efficient utilization of DNNs is essential and desired.

In this chapter, we skip the introduction to preliminary and related work since they are already discussed in the previous chapters. Instead, the comprehensive workflow of the approach is proposed directly in Section 8.2. We examine the performance of the approach with extensive experiments and the results and discussions are given in Section 8.3.

8.2 Real-time low-cost spectral analysis via a hybrid approach

In this chapter, we propose a hybrid approach for real-time low-cost spectral analysis to accelerate and compress neural networks from three perspectives: input data dimension, network size and network cost. The whole workflow is visualized in Fig. 8.1.

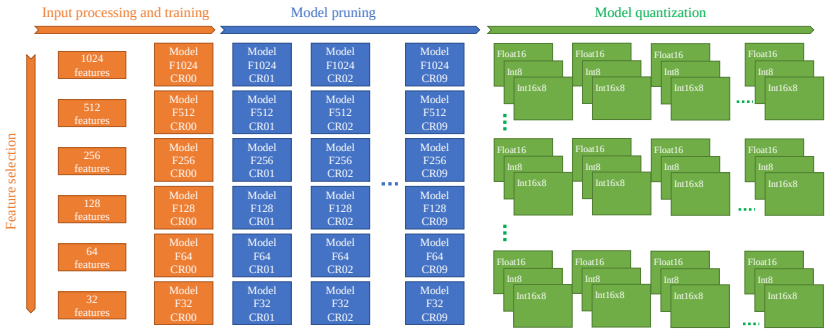


Figure 8.1: The framework of our approach. Feature selection is performed on input data to select desired feature subsets. Then, neural networks will be trained based on the data subsets (F32, F64, ...). Afterward, network pruning with target ratios (CR01, CR02, ...) is executed. Finally, the models will be quantized to various formats (float16, int8, ...).

8.2.1 Data reduction using feature selection

The reduction of input data dimension can contribute to network acceleration since the corresponding operations decrease proportionally, which usually results in a neural network with fewer parameters. Meanwhile, overfitting problems can be suppressed since high-dimensional data tend to cause the curse of dimensionality. Nevertheless, in the context of spectral analysis, we find that typical spectral data contain a substantial amount of redundant information. Hence, the input dimension can be reduced by a large ratio with negligible performance degradation while preserving the primary spatial structure.

Therefore, in the first step, we use watermelon feature selection on a large-scale simulation dataset to select feature subsets with different sizes. Based on these subsets, we train various CNN-based neural networks, which are denoted as *Model F1024, Model F512* and so on.

8.2.2 Network size reduction via efficient network pruning

Network pruning refers to a family of methods that applies certain criteria to given large network architecture to localize such neurons, filters or parameters that contribute the least to the final performance. To reduce the network size in the second step, the trained neural networks are pruned to reduce their parameter sizes and floating-point operations (FLOPs) sizes. The parameter sizes describe the storage demand of a neural network, whereas the FLOPs represent the computational cost for the network execution. To determine to which extent the networks can be pruned without significant performance degradation, various target compression ratios (CR) are tested. As visualized in Fig. 8.1, for all the networks obtained with different feature sizes, we further prune them based on the proposed method in Chapter 7 with the same set of CRs for a fair comparison.

8.2.3 Network quantization via Tensorflow Lite

Network quantization is an efficient approach to reducing the network size and the computational demand. Standard neural networks mainly utilize 32-bit floating-point operations for training and inference. Besides, the weights are also stored in the 32-bit floating-point format. In network quantization, the operations and weights are reduced to lower bit-depth floating and/or integer representations. This is thus a convenient approach for applying neural networks on less powerful edge devices, especially for those where only integer operations are supported.

Quantization can be roughly divided into two subgroups: quantization-aware training and post-training quantization. Quantization-aware training [125, 126] trains quantized neural networks and thus generally achieves better performance. On the contrary, post-training quantization methods [127, 128] quantize already trained neural networks and are, in most cases, a data-free approach.

To construct a smooth workflow from network training to the final deployment on different platforms, in this dissertation, we utilize the Tensorflow [39] framework

to build, train and quantize the neural networks. The built-in Tensorflow Lite module provides various implementations for post-training network quantization.

The original network weights and input/output values are usually 32-bit floats. Once converted to TensorFlow Lite networks, the weights can be further quantized to 16-bit floats, 8-bit ints as well as 8-bit ints with 16-bit activations.

For integer models, the input/output can also be fully converted to the respective integer format to accelerate the execution. In such a case, floating-point values are approximated by the Eq. 8.1. Here, an extra representative dataset is required to calibrate the range for integer representation.

$$real\_value = (int8\_value - zero\_point) \times scale \tag{8.1}$$

In the last step of our approach, all the networks obtained before are quantized to gain further optimization, where different data formats are tested (*Float16, Int8, Int16x8*). To achieve a better trade-off, a comprehensive comparison regarding the model size, overall accuracy and inference time is needed. Since the results may vary across platforms, we conduct our test on diverse hardware.

8.3 Experiments

8.3.1 Experiment setup

8.3.1.1 Dataset

We generate the training dataset based on a random approach, where the presence of elements (up to 28 elements in total, the full list is shown in Table 4.1 of Chapter 4) and the concentrations are independently determined. Therefore, for a given spectra number N (in our case 100k), we have data $\mathbf{D} \in R^{N \times 1024}$ and labels $\boldsymbol{Y} \in R^{N \times 28}$. As for validation, we generate a separate dataset with 20k spectra.

Besides, to validate the final performance of models in the real world, we collect 1100 measurements of 15 various samples and each sample is measured by *#repeat* times. Table 4.5 of Chapter 4 gives the detailed description of each sample. The samples consist of up to 7 elements and they are measured under different measurement conditions and/or on different devices, making the identification a challenging task. Moreover, many element concentrations are very low and the existence of such trace elements makes it more difficult to identify all elements.

8.3.1.2 Neural networks and hyperparameters

Regarding the choice of neural network architectures, we follow Chapter 4 and 6 since different networks and combinations of hyperparameters are already tested. Besides, comparable results can be obtained. In this chapter, we mainly examine two CNN-based neural networks: classic VGG-like CNN and DenseNet. The CNN has six conv layers (one maxpooling layer after every two conv layers) and three fully-connected layers. The sizes of filters are $\{128, 128, 256, 256, 256, 256\}$ and each fully connected layer consists of 512 neurons. For DenseNet, the original configuration of DenseNet-40 is applied. The networks are trained using an Adam optimizer for 120 epochs with a learning rate of 1e-4, which decreases to 1e-5 and 1e-6 after 60 and 90 epochs, respectively.

8.3.1.3 Hardware

To test the overall performance of models in different scenarios, we simulate the following use cases without the need for expensive GPUs:

- High-end workstation with powerful CPU (Desktop with i7-9700k).
- On-site spectral analysis with mobile devices (Microsoft Surface with i5-8350U).
- Industrial application with embedded systems (Raspberry Pi 3B).

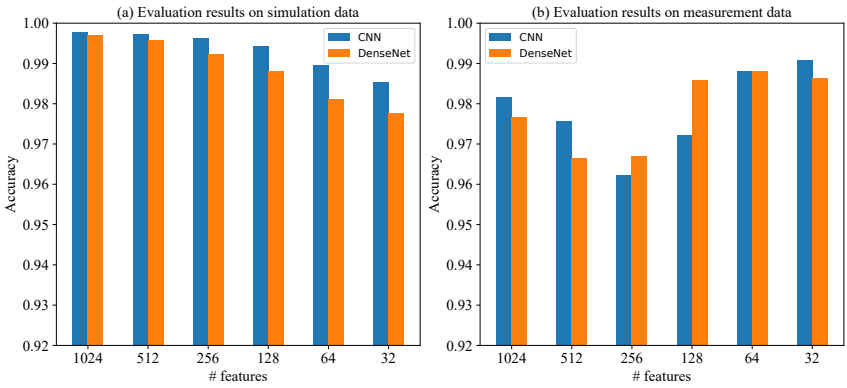


Figure 8.2: Comparison of feature selection results on simulation and measurement data.

8.3.2 Application of feature selection

To perform feature selection on the training dataset, we adopt the original implementation of watermelon to select $\{32, 64, 128, 256, 512\}$ most important features, denoted as $\{F32, F64, F128, F256, F512\}$ in Fig. 8.1. With the feature subsets, we train the neural networks with the same settings as described in the last section.

The validation performance on simulation data (a separate validation dataset containing 20k spectra) and measurement data are visualized in Fig. 8.2. On simulation data, the test accuracies of both CNN and DenseNet decrease simultaneously as the size of feature subsets decreases, which is intuitive. As for measurement data, it is interesting to note that the accuracies reduce at first but then increase to even higher values. Chapter 6 suggested that it is potentially because the overfitting problem usually arises when the feature space is high dimensional. The low-intensity noisy features may mislead the neural network to a better but unpractical local minimum. Besides, as the number of features gets lower, it effectively applies a denoising mechanism for robust performance on measurement data. Hence, in the following sections, we evaluate the models based on their performance on real measurements.

Table 8.1: Parameter compression ratio using feature selection and network pruning

	CNN						DenseNet					
#CR \ #F	1024	512	256	128	64	32	1024	512	256	128	64	32
00	100.00%	58.62%	37.93%	27.58%	22.41%	19.82%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%
01	89.79%	53.37%	34.36%	26.25%	21.57%	19.25%	88.44%	89.31%	88.45%	88.88%	88.52%	88.52%
02	80.40%	48.83%	31.48%	25.02%	20.97%	18.84%	76.45%	76.92%	77.69%	75.85%	76.68%	75.45%
03	71.27%	44.26%	28.66%	23.81%	20.42%	18.54%	65.64%	64.67%	64.38%	64.08%	64.91%	64.67%
04	62.17%	39.95%	26.00%	22.65%	19.80%	17.91%	54.24%	53.32%	53.09%	53.11%	54.43%	54.27%
05	53.51%	35.79%	23.64%	21.43%	19.09%	17.20%	38.38%	38.08%	40.17%	38.09%	40.85%	41.85%
06	44.94%	31.51%	21.46%	20.34%	18.27%	16.49%	26.32%	25.08%	27.18%	28.56%	27.80%	28.43%
07	36.42%	27.29%	19.27%	19.15%	17.50%	15.77%	16.69%	15.72%	16.68%	17.51%	17.38%	17.34%
08	28.06%	22.99%	17.08%	17.92%	16.47%	14.81%	8.34%	8.54%	8.70%	8.83%	9.63%	9.58%
09	19.74%	18.81%	14.74%	16.31%	14.89%	13.68%	3.31%	3.22%	3.10%	3.09%	3.15%	3.06%

Table 8.2: FLOPs compression ratio using feature selection and network pruning

	CNN						DenseNet					
#CR \ #F	1024	512	256	128	64	32	1024	512	256	128	64	32
00	100.00%	50.02%	25.03%	12.54%	6.29%	3.17%	100.00%	50.00%	25.00%	12.50%	6.25%	3.13%
01	84.56%	45.02%	21.87%	12.11%	6.05%	3.06%	89.96%	45.47%	22.55%	11.28%	5.60%	2.80%
02	74.76%	43.38%	20.10%	11.83%	5.98%	2.99%	79.42%	39.97%	20.15%	9.80%	4.93%	2.43%
03	67.03%	41.69%	18.82%	11.60%	5.95%	2.97%	68.84%	34.05%	16.99%	8.48%	4.26%	2.12%
04	60.33%	41.03%	17.84%	11.46%	5.90%	2.89%	57.32%	28.19%	14.15%	7.06%	3.63%	1.79%
05	57.03%	40.85%	17.34%	11.31%	5.79%	2.82%	40.50%	20.32%	10.71%	5.09%	2.75%	1.39%
06	54.77%	40.43%	17.10%	11.26%	5.66%	2.75%	27.91%	13.77%	7.35%	3.82%	1.88%	0.97%
07	52.94%	40.12%	16.89%	11.12%	5.58%	2.68%	17.99%	8.73%	4.62%	2.39%	1.19%	0.60%
08	52.03%	39.59%	16.68%	10.98%	5.41%	2.58%	9.14%	4.74%	2.37%	1.23%	0.64%	0.33%
09	51.63%	39.40%	16.28%	10.65%	5.12%	2.46%	3.42%	1.75%	0.84%	0.42%	0.21%	0.10%

Aside from the identification ability, model size and the computational cost are also major factors in this work. In Table 8.1 and 8.2, the compression ratios of model parameters and FLOPs are presented (the rows with $\#CR = 00$, indicating no network pruning is performed). For CNN, the parameters are significantly reduced to about 20% with a feature size of 32. However, there is no improvement for DenseNet due to its unique structure. In a CNN network, the size of the flatten layer and the fully-connected layers vary with the input data size. Therefore, reducing feature numbers can lead to huge parameter compression results. In contrast, DenseNet utilizes global-average pooling that produces input-independent output, which results in the constant parameter number.

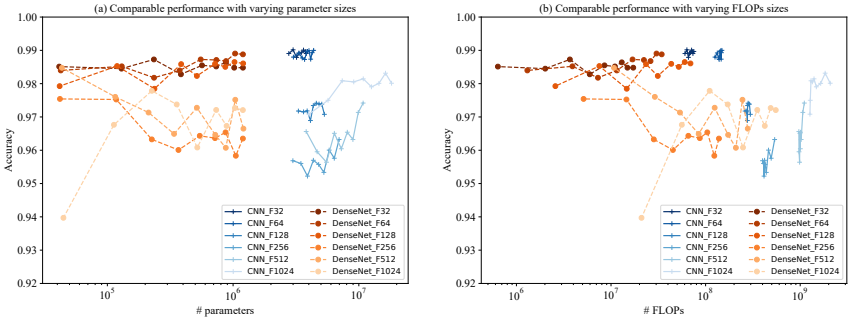


Figure 8.3: Comparison of network pruning results on measurement data. Each line consists of 10 data points and represents the pruning result of one network at different compression ratios (from CR00 (no pruning) to CR09 (pruning at a 90% target ratio)).

Nevertheless, both CNN and DenseNet require fewer FLOPs for compact datasets. With a feature number of 32, only 3% of the original operations are needed for the network inference. This is because the conv layers need fewer iterations to finish the convolution operations on a reduced input space. With the same filter sizes and smaller output shapes, the overall computational costs are minimized by large ratios.

8.3.3 Network pruning

For each neural network in the previous section, we further perform network pruning on all the conv layers with varying target pruning ratios from 10% to 90% (denoted as CR01, 02, CR03, ..., CR09 in Fig. 8.1). The pruning results are presented in Table 8.1 and 8.2. To illustrate the effects of network pruning on the model performance, the *Accuracy - # pamameters* and *Accuracy - # FLOPs* comparisons are visualized in Fig. 8.3. Note that the X-axes use logarithmic scales.

Overall, DenseNets achieve higher final compression ratios across all the configurations. Besides, it shows great efficiency in terms of the absolute parameter number and FLOPs number due to its compact architecture. The network pruning

approach performs better on DenseNets for mainly two reasons: 1) the fully-connected layers in CNN are not pruned due to a high performance drop, which contributes to the large parameter size. 2) There are hyperparameters set by the network pruning method, which determine to which extent the filters are considered redundant and thus removed. We do not finetune these parameters for a fair comparison, and the FLOPs compression ratio is thus not as significant as on DenseNets.

Significant network compression ratios can be obtained by combining feature selection and network pruning methods. Using 32 features and a target pruning ratio of 90%, the final models have only 13.68% and 3.06% of the original parameter sizes for CNN and DenseNet, respectively. Moreover, they require only 2.46% (CNN) and 0.1% (DenseNet) of the original computational resources, which is beneficial for low-power devices. Concerning the identification performance, there is only a negligible accuracy drop of 0.001 and 0.004 compared to the global maximum for CNN and DenseNet.

8.3.4 Network quantization

Finally, we apply the network quantization techniques to the neural networks to gain further optimization. As shown in Fig. 8.1, we quantize each neural network to the following formats:

- 16-bit floats
- 16-bit integer activations with 8-bit integer weights
- 8-bit integers

After quantization, we evaluate the storage sizes, execution time and accuracy scores of all the converted networks. In Fig. 8.4, the comparison of neural networks with 32 features measured on a Raspberry Pi is visualized as representative results. To make a cross-comparison between CNNs and DenseNets, the X- and Y axes are shared.

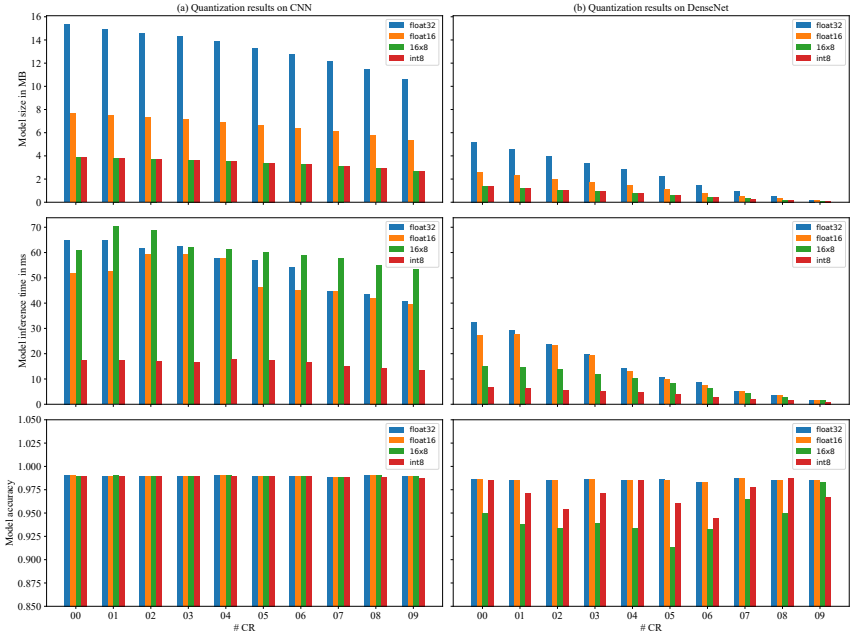


Figure 8.4: Comparison of network quantization results on measurement data and the X- and Y-axes are shared. Networks are trained on the dataset with 32 features. The time performance is measured on a Raspberry Pi 3B.

8.3.4.1 Model size

From 32-bit float (float32) to 16-bit float (float16), all the model sizes are reduced by roughly 50%. Similarly, from 16-bit float down to 16-bit int activations and 8-bit weights (16x8), as well as to fully 8-bit integers (int8), the overall sizes can be further reduced by about another 50%. Here, the difference between 16x8 and int8 is minimal.

8.3.4.2 Inference time

The inference time is calculated based on the average value of 1000 samples. As a general trend, the execution time of neural networks decreases as the bit depth gets smaller. For instance, at $CR = 00$, the inference time of the CNN reduces from 64.95ms (float32) to 17.55ms (int8), which results in a 3.7x speedup. As the model gets smaller, the time improvement of quantization shrinks because of other overheads in the whole computational pipeline that can not be compressed. An anomaly that we notice is that the 16x8 quantization on CNN requires more time than float16 on CNNs, which results from some issues with the experimental quantization mechanism of the framework.

8.3.4.3 Final accuracy

We note that for CNNs, the final performance difference among quantization methods is negligible, which indicates that the CNNs can be quantized and deployed on edge devices without further optimization. Meanwhile, the accuracy of DenseNets is not as stable as CNNs, particularly for 16x8 and int8 quantizations. The performance with int8 is generally better than 16x8 and both methods perform worse than float models in most cases. There are also exceptions at $CR = 04, 08$ where int8 achieves better test accuracy. We believe that the reasons are primarily: 1) DenseNet is more sensitive to quantization because of its compact and interconnected layer structure. 2) Besides, the absolute number of parameters and FLOPs is smaller than the number of CNNs, which makes it harder to quantize. 3) There are still experimental features in Tensorflow Lite and the results may get improved at a later version.

8.3.5 Comparison on different hardware platforms

In this section, we compare the time performance of neural networks with different hardware to simulate various real-world scenarios. Tested hardware is one

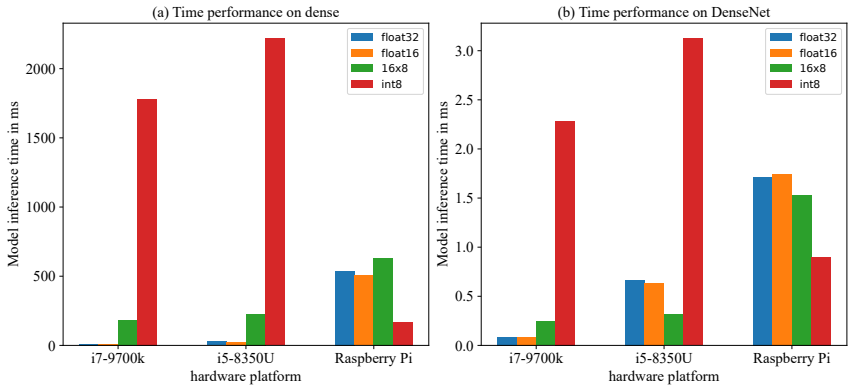


Figure 8.5: Comparison of time performance on different hardware platforms using 32 features.

high-end workstation equipped with i7-9700k, a mobile device Microsoft Surface with i5-8350U, and an embedded system Raspberry Pi 3B. We summarize the comparable results in Fig. 8.5 based on the models with 32 features as input. The difference among hardware platforms is significant due to different computational power. Notably, quantization techniques do not always bring execution acceleration for any given hardware architecture. The models with integer quantization (16x8 and int8) are much slower than the float models on the workstation and the mobile device because the special instructions are not optimized on the x86_64 architecture. On the contrary, Raspberry Pi does gain optimization running quantized models. Therefore, one should selectively choose quantization techniques according to the target hardware.

8.3.6 Overall comparison

Finally, we give an overview of the improvement of our hybrid approach over the previous baseline work. As a representative of edge devices, the results on the Raspberry Pi are shown in Table 8.3. For instance, the original model size of the DenseNet is 5.2 MB and each spectrum needs 540 ms inference time. With a combination of three steps of our approach, we achieve up to 52x model size

Table 8.3: Summary of the final performance of our approach. Time is measured on a Raspberry Pi 3B.

Model / Method	CNN			DenseNet		
	Model size (MB)	Inference time (ms)	Accuracy	Model size (MB)	Inference time (ms)	Accuracy
Baseline	77.32 (1.0x)	886.89 (1.0x)	0.9815	5.20 (1.0x)	540.02 (1.0x)	0.9766
Feature selection	15.33 (5.0x)	64.95 (13.7x)	0.9907	5.20 (1.0x)	32.53 (16.6x)	0.9863
Network pruning	10.58 (1.5x)	40.77 (1.6x)	0.9890	0.20 (26.0x)	1.71 (19.0x)	0.9851
Network quantization	2.69 (3.9x)	13.57 (3.0x)	0.9876	0.10 (2.0x)	0.90 (1.9x)	0.9662
Overall acceleration	28.7x	65.4x	-	52.0x	600.0x	-

compression ratio and 600x network execution speedup on edge devices with even slightly better in many cases. Besides, we find that the utilization of feature selection and network pruning leads to the most significant resource reduction (300x for inference time), which can easily be adopted in other related applications without a domain- or hardware-specific modification, which is a huge advantage. Moreover, the final time performance of 0.9 ms enables real-time analysis even with other processing overheads being taken into consideration.

9 Minimal cost device calibration via meta learning

In the previous chapters, the DNN framework for spectral analysis is proposed, extended and optimized. It achieves SOTA performance with the cost-efficient consideration of the application in industry. However, so far, the focus of previous chapters is on the simplified scenario where only one specific measurement device is involved. Therefore, when it comes to the mass-production manufacturing industry, the framework faces the same challenge as the classic methods do: device calibration. Intensive costs are required to ensure the performance and consistency across a large number of devices, even as the devices begin to age, which contributes to an undesired variable cost from the economic perspective.

This chapter aims to address this problem by providing a meta-learning-based solution for minimal-cost device calibration of the DNN framework. In the rest of the chapter, a more detailed description of the problem is given in 9.1 followed by the related work in Section 9.2. The framework of our approach is presented in Section 9.3. Afterward, we conduct extensive experiments to evaluate the approach and discuss the comparable results with baseline methods in Section 9.4.

9.1 Introduction

As shown in previous chapters, the utilization of DNNs can significantly optimize the spectral analysis procedure and enable real-time analysis at deployment.

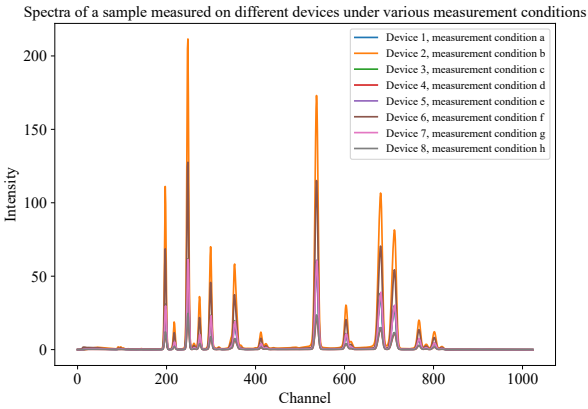


Figure 9.1: Spectra of a reference sample consisting of 10% Au, 20% Ag, 20% Cu, 20% Cd, 20% Fe and 10% Mo. Measured using different devices and under different measurement conditions, the resulting spectra are also different.

However, there is still a major limitation for both classic methods and DNN approaches: device calibration.

$$\mathbf{S} = P(\mathbf{\Phi}, \boldsymbol{\lambda}, \mathbf{K}) \tag{9.1}$$

Note that in Eq. 9.1, a spectrum is not only determined by the measured sample itself, but also by the measurement conditions $\boldsymbol{\lambda}$, which are set during the measurement, and the characteristics of the measurement device $\mathbf{K}$. Thus, the spectra of a sample measured on different devices and/or under different measurement conditions are also different regarding the absolute intensity, peak positions and relative peak intensity ratios, as shown in Fig. 9.1.

For the classic methods, $\boldsymbol{\lambda}$ and $\mathbf{K}$ are considered within the physical model. Nevertheless, to get a feasible and accurate result, calibration is needed for each unique device, and a-prior information of $\boldsymbol{\lambda}$ must be given. Typically, the first calibration will be done when the device is produced in the factory, where a large size of reference samples needed to be measured under various measurement conditions. Later after the initial deployment, it also requires a regular on-site

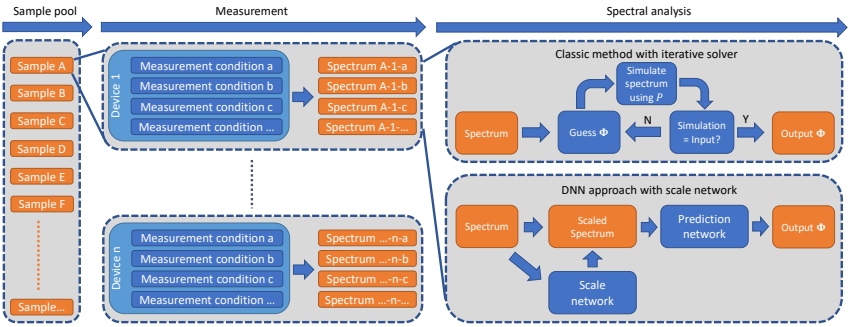


Figure 9.2: Overview of the whole workflow for spectral analysis tasks. A sample will be measured on different devices under various measurement conditions, resulting in different spectra. Spectral analysis is done either using the classic method or the DNN approach. Note that both methods work on one specific device. Therefore, device calibration is required and leads to enormous resource demand, especially in the large-scale manufacturing industry.

calibration with a time interval of six to twelve months to maintain its accuracy, which is time and labor-intensive.

As for the DNN framework, the parameter $\mathbf{K}$ is also not well taken into consideration, instead, it is treated as constant. As a result, its cross-device result (see Table 4.6 in Chapter 4) shows that the DNNs trained on one device have larger error rates on other devices. Therefore, for each device produced, dedicated DNNs have to be trained to reach commercial-level performance, which makes the process costly and resource-demanding for large-scale industrial applications.

An overview of the whole workflow is visualized in Fig. 9.2, which suggests that the device calibration is crucial and costly for both classes of approaches. In this chapter, we focus on the problems from the manufacturing industry perspective, where a big number of product lines and products produced in each product line are involved. More concretely, the challenges can be summarized as follows:

- As commercial products, the results of the spectral analysis must be very precise. To guarantee that, device calibration is needed. However, this leads to a huge cost to (a) the industry because the time and labor invested increase significantly with the number of products manufactured per year,

which limits the growth potential; (b) the customer since regular calibration is needed and to conduct a calibration, the cost of a large set of expensive reference samples and the on-site technical support is huge.

- Therefore, the cost of calibration should be minimized (a) from the industry side to enable a competitive and attractive pricing strategy while maintaining a reasonable profit margin, and (b) from the customer side to reduce the maintenance cost and improve the user experience to keep a long-term growth of customer sizes.

To address the issues mentioned above, in this chapter, we propose the meta-learning-based minimal-cost device calibration for DNN-based spectral analysis. We formulate the spectral analysis with different $\boldsymbol{\lambda}$ and $\mathbf{K}$ as different basic tasks and adopt the model-agnostic meta-learning (MAML) [129] to train a more general meta network for fast calibration. We conduct extensive experiments with large-scale simulation datasets as well as the real measurement of various samples on 20 measurement devices. Results show that the proposed approach outperforms the original framework by achieving consistently excellent performance across various evaluation datasets on unknown devices. Besides, compared to the original framework after calibration, it performs on par without calibration, which enables zero-shot device calibration. Moreover, to obtain better precision on one specific device, our approach requires only a small amount of spectra to calibrate itself, which is a huge improvement over classic methods.

The main contributions can thus be summarized as follows:

- To address the common calibration problems in the broad domain of spectral analysis, we are, to the best of our knowledge, the first to propose meta learning-based approach with domain-specific task formulations, where not only the unique device characteristics but also the changing measurement conditions are taken into consideration.

- We train our models on large-scale simulation datasets and evaluate the performance on real-world measurements across different devices, demonstrating the effectiveness of our approach as a new DNN baseline method regardless of the need for calibration.
- While achieving better performance, we significantly reduce the calibration time and the corresponding costs both for the industry and the customers, which largely simplifies the overall procedure and contributes to the sector growth.

9.2 Related work

9.2.1 Transfer learning and meta learning

The concept of transfer learning [130] has been proposed since the 1970s and is still a hot topic in recent years. The general idea of transfer learning is to pretrain a neural network for certain tasks, and then apply it to another unknown but related task. A common example in the computer vision domain is to train a large DNN on a large dataset (e.g. ImageNet with 1000 classes of objects). Afterward, the main architecture (usually the conv layers) is adopted and kept constant. By adding additional fully-connected layers, the model can be further trained (referred to as finetuning) to classify images of other classes.

One major limitation of transfer learning is the performance degradation on unknown tasks, especially when only a few data are available and overfitting problems are significant. To address such problems, meta learning has become popular in the last few decades with an explicit design of “learning to learn". Although there is no standard definition of meta learning, it typically refers to the process to learn high-level meta information from various tasks, so that a network can adapt quickly to a new unseen task.

The recent representative work model-agnostic meta-learning (MAML) [129] proposes a general meta learning formulation, which is compatible with various

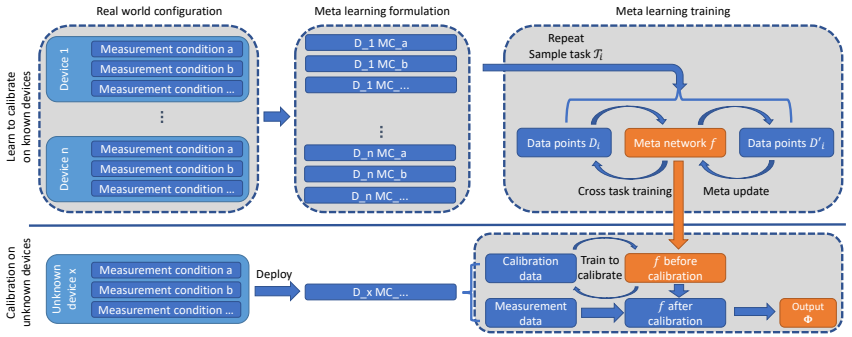


Figure 9.3: Framework of our approach towards minimal cost device calibration. In our meta learning procedure, the general spectral analysis problem is formulated as a collection of enormous basic tasks, where each task corresponds to one specific device (*D_1, D_2, ...*) with one fixed measurement condition (*MC_a, MC_b, ...*). The meta training will be performed on the task pool based on all known devices. Later, at calibration time, the new measurements on an unknown device under an unknown measurement condition will be used for post-calibration training of the meta network and evaluation of the final performance.

neural network architectures and learning problems, as long as they are gradient descent-based approaches. Given the task distribution $p(\mathcal{T})$, the objective of MAML is to train a model f in a way that the parameters θ of which are sensitive to the changes of tasks. To realize that, for each task $\mathcal{T}_i \sim p(\mathcal{T})$, random data points $\mathcal{D}$ will be sampled to train an inner loop using Eq. 9.2, where the α and $\mathcal{L}$ denote the gradient descent step size and the loss function, respectively.

$$\theta' = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta) \tag{9.2}$$

Afterward, the meta update will be performed on separate sampled sets of data points $\mathcal{D}'_i$ using Eq. 9.3, where β is the meta update step size. This training mechanism results in a meta network, which is sensitive to task changes and can thus adapt to an unknown task with less effort.

$$\theta = \theta - \beta \nabla_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta'_i}) \tag{9.3}$$

9.3 DNN-based minimal cost device calibration via meta learning

In this section, we present the framework of our approach to realizing the minimal cost device calibration based on meta learning. We aim to achieve the best pre-calibration performance for zero-shot generalization and the best post-calibration performance for few-shot minimal cost calibration.

9.3.1 Problem definition

Mathematically, the general solution of spectral analysis is $\mathbf{\Phi} = P^{-1}(\mathbf{S}, \boldsymbol{\lambda}, \mathbf{K})$, and current solutions focus on the sub-problem of $\mathbf{\Phi} = P^{-1}(\mathbf{S}|\boldsymbol{\lambda}, \mathbf{K})$ or $\mathbf{\Phi} = P^{-1}(\mathbf{S}, \boldsymbol{\lambda}|\mathbf{K})$ on one specific device. Therefore, to obtain a more generalized meta network that can fast adapt and deploy on different devices under varying measurement conditions, we formulate the meta learning task as a *multi-device multi-configuration* problem.

The real-world problem setting is visualized in Fig. 9.3 in the upper left corner. Here, multi-device refers to the task difference due to the variable $\mathbf{K}$, which mainly comes from a) the difference in overall design between product lines; b) the change of hardware components within the same product line; and c) the difference of the same hardware component between different product batches.

Meanwhile, multi-configuration refers to the different measurement conditions $\boldsymbol{\lambda}$ for the real measurements, which are influenced by e.g. the reference voltage, the controlled current, the applied filter and the measurement time.

9.3.2 Meta learning based calibration

Therefore, in our meta learning formulation, the most basic task $\mathcal{T}_i$ is then defined as the spectral analysis on one device $\mathbf{K}_k$ under one measurement conditionλ_j.

Accordingly, the task distribution $p(\mathcal{T})$ is then defined as a collection of enormous sub-tasks, as shown in Eq. 9.4.

$$P^{-1}(\mathbf{S}|\boldsymbol{\lambda}_j, \mathbf{K}_k) = \mathcal{T}_i \sim p(\mathcal{T}) \tag{9.4}$$

During the training phase, we collect various physical devices and generate large-scale simulation datasets under different measurement conditions, where each dataset represents one basic task. Then, we adopt the MAML method to train our meta network by randomly sampling the tasks from the task pool. This step is denoted as pre-calibration training on known devices and after this step, the meta network can 1) achieve good performance across different tasks and 2) adapt fast to new unknown tasks.

To evaluate the post-calibration performance, at the calibration phase, we collect another set of devices as unknown devices and train the meta network with a few new samples measured on them. To ensure a fair comparison, for each unknown basic task $P^{-1}(\mathbf{S}|\boldsymbol{\lambda}_j, \mathbf{K}_k)$ (i.e., an unknown device with unknown measurement condition), a fixed number of reference samples are measured and the spectra are split into a small calibration dataset and the test measurement dataset with a fixed seed. Then, the meta network will be calibrated (i.e. post-calibration training) using the calibration data and the performance is evaluated on the measurement dataset.

9.3.3 Choice of baseline methods

Since we are the first to introduce meta learning in the spectral analysis domain, there is no previous work to compare to. To demonstrate the effectiveness and efficiency of our approach, we set up two baseline methods for comparison.

9.3.3.1 Vanilla training

First, since the framework proposed in Chapter 4 is the SOTA solution in this domain, we choose it as one baseline work, denoted as the “vanilla training" to evaluate the pre- and post-calibration performance, respectively. Note that for pre-calibration training, we only use the simulation dataset from one device. Besides, the post-calibration training is the same as the configuration in this chapter.

9.3.3.2 Transfer training

Another widely used technique, as suggested in Section 9.2, is the transfer learning. Therefore, we also adopt this approach to train a network on all known datasets for pre-calibration training, and then finetune the network with the calibration data for post-calibration training. In addition, for a fair comparison, the network architecture used by all the methods is the same.

9.4 Experiment

9.4.1 Experiment setup

9.4.1.1 Data preparation

To validate our approach in the industry context, we collect in total 20 different measurement devices for the experiment. Among them, 10 devices are used for pre-calibration training and another 10 devices are for post-calibration training. The measurement conditions are exhaustively selected as long as they are feasible for measurement, and for each basic task $P^{-1}(\mathbf{S}|\boldsymbol{\lambda}_j, \mathbf{K}_k)$, we generate a simulation dataset consisting of 100k spectra. Each spectrum has a length of 1024 and a corresponding label is assigned to the spectrum, where the number of elements varies from 2 to 28 and the corresponding concentrations are randomly chosen between 0% and 100%. Thus, in each dataset, we have the data $D \in R^{100k \times 1024}$

Table 9.1: Details of samples for the real measurements

Sample	# repeat	Element concentration in %					
		Au	Ag	Pd	Cu	Zn	Ni
sample_1	3	95.08	2.46	0	2.46	0	0
sample_2	3	74.99	10.18	0	14.83	0	0
sample_3	3	75.43	5.02	0	9.89	9.66	0
sample_4	3	58.08	4.89	0	37.03	0	0
sample_5	3	58.23	30.02	0	11.75	0	0
sample_6	3	33.46	12.55	0	39.48	14.51	0
sample_7	3	90.06	4.4	0	5.54	0	0
sample_8	3	75.03	14.86	0	10.11	0	0
sample_9	3	75.2	0	9.92	7.98	0	6.9
sample_10	3	58.56	27.68	13.76	0	0	0

and the label $Y \in R^{100k \times 28}$. A full list of all relevant elements is summarized in Tab. 4.1 of Chapter 4.

As for real measurement data regarding the performance evaluation in the real world, we collect 10 different samples, denoted as *sample_1* to *sample_10*, and measure them on all of the 20 devices under randomly chosen measurement conditions, denoted as *Data 1* to *Data 20.* To obtain a reliable result, each sample is measured three times. The details of the samples are shown in Tab. 9.1.

9.4.1.2 Network selection

For all the experiments of the vanilla training, the transfer learning and our approach, we use the same network architecture for a fair comparison. According to the results in Chapter 4, we choose the VGG-like CNN for faster training without performance degradation.

The CNN consists of 6 convolutional layers with a fixed kernel size of 12, while the number of filters grows from 128 to 256. After the Flatten layer, the 3 followed dense layers have a fixed neuron size of 512. The network has in total over 20M parameters and an overview of the architecture is displayed in Tab. 9.2. Besides, for vanilla learning and transfer learning, a scale network is needed. Following

Table 9.2: The network architecture

Layer name	Filter size	Output size	# param.
Conv1	(1x12,128)	(1024,128)	1664
Conv2	(1x12,128)	(1024,128)	196736
MaxPooling		(512,128)	
Conv3	(1x12,256)	(512,256)	393472
Conv4	(1x12,256)	(512,256)	786688
MaxPooling		(256,256)	
Conv5	(1x12,256)	(256,256)	786688
Conv6	(1x12,256)	(256,256)	786688
MaxPooling		(128,256)	
Flatten		(32768,)	
Dense1		(512,)	16777728
Dense2		(512,)	262656
Dense3		(512,)	262656
Dense4		(22,)	11286

Chapter 4, an MLP with 6 layers, each of which consists of 512 neurons is adopted. As for our meta learning approach, no scale network is needed.

9.4.1.3 Training

The training process is divided into two parts: pre-calibration training and post-calibration training. For pre-calibration training, the simulation datasets are used to achieve the desired performance on large-scale spectral analysis tasks. To be more concrete, for vanilla training, the network is trained on the dataset from one specific device. Out of the 100k spectra of a single dataset, 80% are used as training data and 20% are used as test data. As for transfer learning, the network should theoretically be trained on all available datasets. However, heuristics show that as the number of devices increases, the need for the size of each dataset reduces. Approximately 20k spectra per dataset are sufficient, as long as the total number of spectra exceeds 100k. Therefore, we reduce the size of a single dataset during the experiment accordingly. For our meta learning-based approach, we notice that the total data size is not dependent on the number of basic tasks (i.e. the number of devices and measurement conditions) and the volume of the single dataset can be proportionally reduced while the total data size stays at 100k.

The vanilla learning and the transfer learning both utilize the Adam optimizer with an initial learning rate of $1e-4$. The networks are trained on the training data for 120 epochs with a batch size of 64, and the learning rate is divided by 0.1 after 60 and 90 epochs, respectively.

As for meta learning training, an SGD with a learning rate (step size α) of $1e-5$ is used for inner loop training and an Adam with a learning rate (step size β) of $1e-4$ is applied for the meta update. Except for that, all other hyperparameters are kept the same.

Later in post-calibration training, all the methods follow a uniform setting. Since the focus of the work is on real-world deployment in the industry, we choose to evaluate the pre- and post-calibration performance on real measurements. Therefore, we choose 2 out of the 10 samples using a fixed seed as the calibration data and train all the networks on them using an Adam with a learning rate of $1e-5$ for 20 epochs. Finally, we test the performance on the rest measurement data.

9.4.2 Performance evaluation

After the training as described in the last section, the models are evaluated on all measurement data across different devices. Since 10 devices are used for pre-calibration training and we train for each device a vanilla training model, we obtain a total number of 10 models, denoted as *V_1* to *V_10*. As for transfer learning and meta learning, we train a model on all 10 devices, hence we have one transfer learning model and one meta learning model, respectively.

Then, the models are evaluated on the reference samples measured on different devices *Data 1* to *Data 20*. Note that the devices of *Data 11* to *Data 20* are not available during the pre-calibration training. The evaluation metric is chosen to be the *Mean Absolute Error (MAE)*, which is a standard metric in the industry as well as in the deep learning domain. The performance of pre- and post-calibration training is summarized in Tab. 9.3 and 9.4, respectively.

Table 9.3: Test MAE of real measurements from different devices on all the methods before calibration. For vanilla learning, 10 models (*V_1 to V_10*) are trained on 10 known devices.

		Vanilla learning										Transfer learning	Meta learning
		V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8	V_9	V_10		
Known device	Data 1	0.33	0.71	0.35	0.53	0.35	0.74	0.56	0.55	0.32	0.80	0.36	0.27
	Data 2	0.87	0.61	0.71	1.03	0.78	0.83	1.18	1.00	0.83	0.93	0.59	0.29
	Data 3	0.42	0.73	0.34	0.61	0.39	0.74	0.70	0.58	0.35	0.80	0.39	0.29
	Data 4	0.28	0.67	0.31	0.34	0.30	0.59	0.46	0.37	0.28	0.68	0.30	0.27
	Data 5	0.41	0.81	0.38	0.56	0.39	0.79	0.62	0.59	0.36	0.86	0.39	0.26
	Data 6	0.67	0.41	0.50	0.81	0.54	0.43	1.01	0.74	0.68	0.49	0.42	0.29
	Data 7	0.55	1.21	0.68	0.60	0.59	1.02	0.35	0.79	0.52	1.11	0.41	0.28
	Data 8	0.28	0.66	0.30	0.36	0.29	0.57	0.47	0.34	0.28	0.65	0.30	0.27
	Data 9	0.42	0.92	0.41	0.57	0.42	0.84	0.59	0.60	0.35	0.91	0.37	0.26
	Data 10	0.67	0.42	0.51	0.81	0.55	0.41	1.02	0.74	0.69	0.47	0.42	0.26
Unknown device	Data 11	0.65	0.45	0.49	0.78	0.53	0.41	1.00	0.72	0.67	0.45	0.41	0.26
	Data 12	0.67	0.41	0.50	0.81	0.54	0.43	1.01	0.74	0.68	0.49	0.42	0.29
	Data 13	0.53	0.47	0.41	0.63	0.46	0.27	0.86	0.53	0.55	0.31	0.27	0.27
	Data 14	0.62	0.44	0.45	0.74	0.50	0.42	0.95	0.69	0.63	0.47	0.39	0.27
	Data 15	0.63	0.43	0.47	0.77	0.51	0.44	0.97	0.71	0.64	0.50	0.41	0.27
	Data 16	0.67	0.42	0.50	0.80	0.54	0.44	1.00	0.74	0.68	0.50	0.44	0.29
	Data 17	0.66	0.42	0.50	0.79	0.53	0.40	1.00	0.72	0.68	0.46	0.42	0.29
	Data 18	0.54	0.46	0.43	0.65	0.46	0.26	0.88	0.53	0.57	0.28	0.27	0.26
	Data 19	0.62	0.42	0.46	0.75	0.50	0.39	0.96	0.68	0.63	0.46	0.38	0.26
	Data 20	0.57	0.44	0.44	0.67	0.47	0.26	0.91	0.57	0.59	0.29	0.26	0.28

To give a better overview of the performance comparison, we visualize the pre-calibration results in the radar diagrams in Fig. 9.4 and each axis represents the test MAE of measurement data on one specific device. Note that in this setting, a smaller polygon outperforms a large polygon due to a smaller average error rate. On the left side, the test MAEs on known devices (*Data 1* to *Data 10*) and unknown devices (*Data 11* to *Data 20*) are depicted in the upper and lower diagrams, respectively.

Overall, we can see from the diagrams that the transfer learning polygon inscribes the normal learning polygons and the meta learning polygon is noticeably smaller than the transfer learning polygon, which indicates that our approach is superior to the two baseline methods, while the transfer learning is also generally better than the vanilla learning baseline. This is intuitive since the vanilla model only

Table 9.4: Test MAE of real measurements from different devices on all the methods after calibration. Bold values are the best.

		Vanilla learning										Transfer learning	Meta learning
		V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8	V_9	V_10		
Known device	Data 1	0.23	0.32	0.25	0.23	0.24	0.31	0.23	0.22	0.25	0.31	0.22	**0.16**
	Data 2	0.31	0.27	0.29	0.31	0.29	0.33	0.39	0.30	0.34	0.32	0.28	**0.17**
	Data 3	0.23	0.32	0.23	0.23	0.23	0.31	0.25	0.22	0.24	0.32	0.22	**0.17**
	Data 4	0.23	0.33	0.25	0.22	0.24	0.31	0.24	0.23	0.24	0.32	0.23	**0.15**
	Data 5	0.24	0.32	0.25	0.23	0.23	0.33	0.24	0.23	0.25	0.33	0.23	**0.17**
	Data 6	0.26	0.32	0.24	0.23	0.27	0.24	0.32	0.25	0.29	0.21	0.22	**0.16**
	Data 7	0.28	0.38	0.32	0.26	0.29	0.39	0.26	0.29	0.33	0.40	0.23	**0.16**
	Data 8	0.23	0.33	0.25	0.23	0.24	0.31	0.23	0.23	0.24	0.33	0.21	**0.16**
	Data 9	0.25	0.33	0.24	0.23	0.23	0.33	0.24	0.24	0.23	0.35	0.22	**0.16**
	Data 10	0.26	0.32	0.24	0.24	0.28	0.23	0.34	0.25	0.30	0.21	0.22	**0.17**
Unknown device	Data 11	0.26	0.34	0.24	0.23	0.27	0.24	0.33	0.25	0.29	0.22	0.22	**0.16**
	Data 12	0.26	0.32	0.24	0.23	0.27	0.24	0.32	0.25	0.29	0.21	0.22	**0.16**
	Data 13	0.26	0.31	0.24	0.26	0.28	0.24	0.34	0.26	0.31	0.22	0.21	**0.16**
	Data 14	0.25	0.33	0.24	0.23	0.27	0.25	0.33	0.25	0.29	0.22	0.22	**0.16**
	Data 15	0.25	0.32	0.23	0.23	0.27	0.24	0.32	0.24	0.28	0.21	0.22	**0.16**
	Data 16	0.26	0.31	0.24	0.23	0.28	0.25	0.33	0.25	0.30	0.21	0.22	**0.16**
	Data 17	0.25	0.31	0.24	0.24	0.27	0.23	0.33	0.25	0.29	0.22	0.23	**0.16**
	Data 18	0.26	0.31	0.24	0.26	0.27	0.24	0.33	0.25	0.31	0.21	0.22	**0.15**
	Data 19	0.25	0.31	0.24	0.24	0.27	0.25	0.33	0.25	0.28	0.22	0.22	**0.16**
	Data 20	0.26	0.31	0.25	0.26	0.28	0.24	0.33	0.25	0.30	0.22	0.22	**0.15**

trains on its specific device, whereas the other two methods are trained across different devices. It is thus to be expected that the general performance of the vanilla training is the worst. Also, our approach is designed to learn the device differences, which contributes to the lower MAE compared to transfer learning.

Surprisingly, we find the vanilla learning on its specific device also does not perform well on the measurement data compared to our approach and transfer learning. We believe that it is largely due to the overfitting problem that occurred during the training on the simulation datasets, where the model is exposed only to one basic task $P^{-1}(\mathbf{S}|\boldsymbol{\lambda}_j, \mathbf{K}_k)$. At test time, the change of the measurement condition and the noise captured during the measurement leads to a performance shift of the vanilla model, while our approach and the transfer learning are less affected due to the better variety of the training datasets.

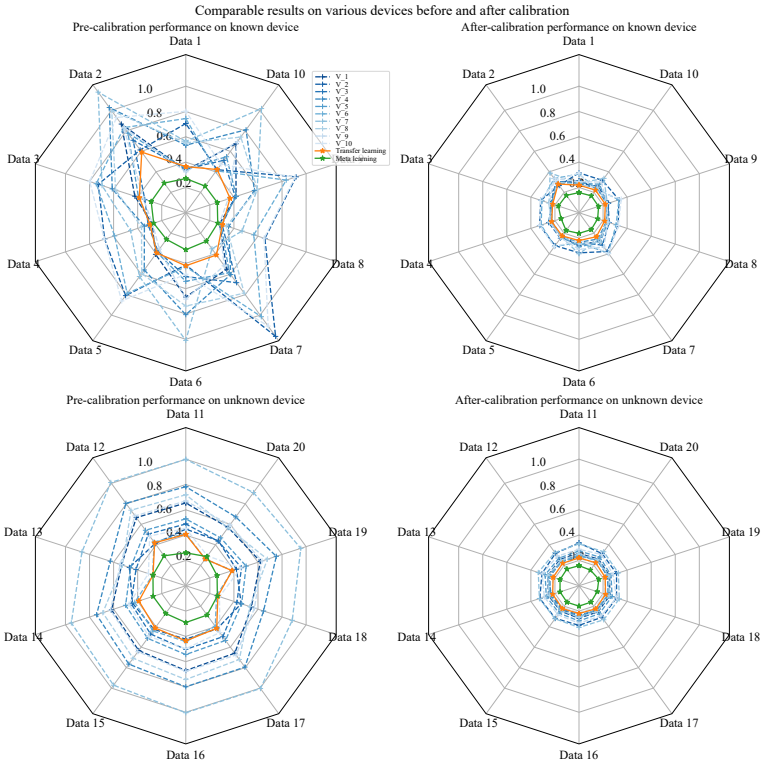


Figure 9.4: Comparable results of different methods on various data. Each axis represents the test MAE of the measurement spectra on different devices and the method with a smaller polygon has thus better performance. Overall, our approach outperforms transfer learning, which again outperforms vanilla learning on both known and unknown devices. Also, it applies to both the pre- and after-calibration performance.

Another interesting observation is that regardless of the methods, the results on known devices and unseen devices are similar. After further analysis, we summarize the explanations as follows:

- For vanilla training, a model is expected to perform well only on its training device. Therefore, for e.g. *V_1*, its test MAE on *Data 2* to *Data 20* should be similar, because they are all unknown devices to the vanilla approach.

To validate that, we conduct a two-tailed t-test and the result indicates that the MAE on *Data 1* is statistically significantly smaller than the MAE on other devices with a p-value$<$0.01.

- For transfer learning and our meta learning approach, since the models are trained on the collection of 10 known devices, we believe that they have learned the general task $P^{-1}(\mathbf{S}, \boldsymbol{\lambda}, \mathbf{K})$ and therefore perform well on the measurement data from both known and unknown devices, which is also agreed by the statistical test. For instance, the null hypothesis on *Data 1* and *Data 11* can not be rejected with a p-value of 0.22 and it indicates that the models do not overfit on known devices. Moreover, to which extent the methods learn the general task leads to the performance difference. As shown in the figure, our approach outperforms transfer learning by learning a more generalized meta network that can quickly adapt across basic tasks.

As for post-calibration performance, as mentioned before, we finetune all the models on 2 of 10 samples and visualized the final performance on the rest 8 samples in Fig. 9.4 on the right side. It shows that after two-shot calibration, all the models achieve a better performance compared to pre-calibration, and our approach consistently outperforms the baseline methods by a clear MAE margin.

Additionally, we compare the overall performance of all the methods on unknown devices in Fig. 9.5 to provide a better overview of the whole calibration process, where the average test MAEs on *Data 11* to *Data 20* are shown in the figure. We find that in general, the few-shot (i.e., two-shot) calibration can significantly improve the performance of all the methods on an unknown device, and the vanilla training is outperformed by transfer learning, which is again outperformed by our approach.

Besides, we notice that the pre-calibration performance of our approach is on par with the post-calibration performance of the vanilla learning, which shows the great effectiveness of our approach. Therefore, our approach is also feasible

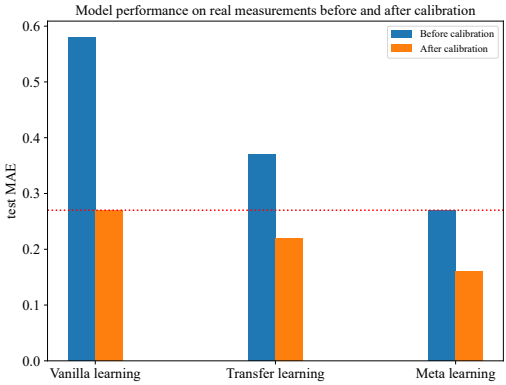


Figure 9.5: Comparable performance of all the methods on measurement spectra before and after calibration. Our meta learning approach outperforms the baseline methods by a large margin. Besides, our method performs before calibration on par with the vanilla learning after calibration, which is advantageous for zero-shot calibration.

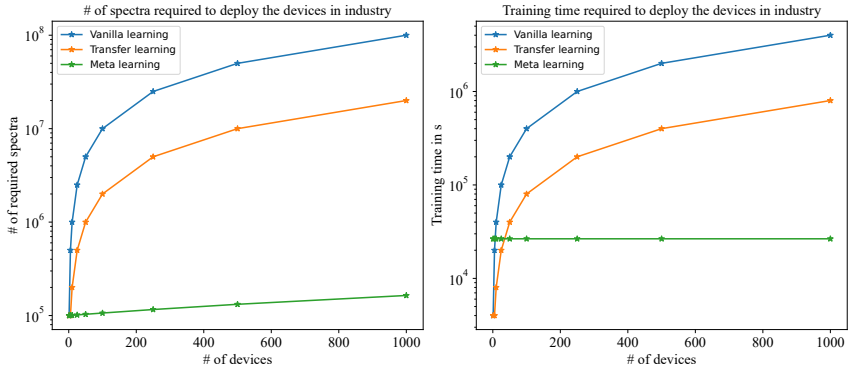


Figure 9.6: The real-world resource analysis of device deployment in industry. The data are the required spectra size and training time to achieve the best performance of each method, respectively. Note that the y-axes are all logarithmic. At a large scale, our approach demands significantly fewer resources compared to the baseline methods, leading to minimal cost device calibration.

in a zero-shot calibration setting and this is very practical in real-world scenarios, where no additional information is available for an unknown device and a measurement spectrum needs to be analyzed immediately.

9.4.3 Resource analysis

So far, we discuss the performance of our approach as well as the baseline methods for the calibration problem. In this section, we conduct a resource analysis in industry to demonstrate the efficiency of our method over the others. In Fig. 9.6 (a), we give the total number of spectra needed to deploy all the approaches on varying numbers of devices at their best performance. And in (b), the total training time including the calibration time is depicted. Note that for both sub-figures, the y-axis is logarithmic.

When a new device is produced, vanilla learning needs a new simulation dataset to train a separate model on the new device. Therefore, both the required number of spectra and training time are huge. Compared to that, transfer learning requires fewer spectra per device, which leads to a lower training time. However, it still needs to re-train the model as the device size changes. As for our approach, the total number of spectra is nearly constant (it is constant for zero-shot calibration if no calibration data is needed) as the number of devices increases. Although at an early stage when only several devices are involved, the training time of our approach is longer than others due to the meta learning training design, it takes less time when the number of devices exceeds 30, which is typically the case in the industry. Moreover, the time demand is constant regardless of the number of devices. Therefore, our approach consumes the fewest resources and provides the best performance on large-scale problems, and the resource analysis indicates the effectiveness and saving potential of our approach, especially in the manufacturing industry.

10 Conclusion and future work

So far, all the aspects of the whole framework are proposed in detail, which is ready to be utilized in production as an AI-based alternative. This chapter concludes the dissertation with a conclusion section and an outlook of future work.

10.1 Conclusion

In this dissertation, a DNN-based comprehensive baseline framework for large-scale spectral analysis is proposed, which outperforms the current classic commercial solution in most aspects. The effectiveness, applicability and usability of the framework are demonstrated by extensive experiments on complex simulation datasets and real measurements from more than 20 measurement devices, which serve well as real-world scenarios.

The design of the whole framework starts with the two fundamental problems in spectral analysis: quantification (quantitative analysis) in Chapter 4 and identification (qualitative analysis) in Chapter 6.

The quantitative problem is first discussed mainly due to the fact that it is the current focus of the market and the requirement on the result accuracy is high. Compared to previous work, the proposed framework enlarges the element coverage from several specific elements to a broad range of 28 most relevant elements as an all-in-one solution. In order to set up the new baseline with ideal performance, the proper size of training data and network architecture are examined and discussed. As a result, this framework extends the application to a more general

scenario and compared to classic analytical methods, it achieves similar quantification performance while requiring much less computational time by a factor of 400. Similarly, the work on qualitative analysis in Chapter 6 also achieves great improvement over classic methods (which is essentially not capable of solving the tasks) as well as previous work by a large margin.

Afterward, optimizations to the DNNs, which are the core of the framework, are introduced to achieve an efficient workflow throughout the analysis pipeline: **data input**, **network execution** and **deployment**.

To reduce the input data volume and release the throughput burden, the feature selection method watermelon is proposed in Chapter 5 as a dedicated module to demonstrate its generalization ability across machine learning domains. It shows that directly using the estimated Bayes error rate as the score of individual features is intuitive, straightforward and effective. We furthermore interpret the interaction between features and illustrate that although monotonic correlation indicates true redundancy, a non-monotonic relationship can bring more information and improve the performance of a classifier, which is against the heuristic used by many other popular algorithms. We quantify and apply the effects of correlation and relevance into our approach based on Spearman's rank correlation coefficient and normalized mutual information. We validate our proposed method on various real-world problems and the results show that our approach is very robust and outperforms other state-of-the-art competitors.

Later in Chapter 6, the method is further examined on qualitative problems with comparison to other dimension reduction techniques. Combined with watermelon, the framework achieves even better accuracy on real measurement data with 19% and 3% of the original storage and computational costs, respectively. Moreover, compared to other dimension reduction approaches, watermelon provides a more interpretable result that is physically explainable, which is essential for industrial applications. In addition, the experiment analysis provides several best practices for related domains toward a better model deployment in real-world applications.

To further exploit the power of DNNs that require a considerable amount of hardware resources, a network pruning method is introduced in Chapter 7 as a general

approach to slim the networks without losing performance. The importance and redundancy of internal features in DNNs are evaluated, and this information is utilized to guide the pruning process. The approach achieves impressive results on different datasets and network architectures compared to other competitors.

Combined with feature selection, network pruning and additionally network quantization, the proposed framework is further optimized for real-time low-cost network execution on edge devices in Chapter 8. We apply feature selection to reduce the input data size, perform network pruning on the trained networks to reduce the network size and finalize the whole pipeline with quantization to optimize the network cost. We evaluate our approach on various hardware platforms and achieve up to 52x model size compression ratio and 600x network execution speedup on edge devices. Moreover, we demonstrate that our approach consistently delivers robust performance at all three steps, with even higher accuracy in most cases. Overall, we show that our approach is effective and easy to adopt, even for complex spectral analysis tasks. Besides, we suggest that care should be taken regarding the application of quantization techniques on different target platforms. A transfer to other IoT/big data domains is also straightforward and does not require further modification.

As the last step, an approach towards minimal cost device calibration is integrated into the framework in Chapter 9, which aims to reduce the calibration costs related to the increasing number of devices in the long-term large-scale deployment. More concretely, we formulate the general spectral analysis problem as a multi-device multi-configuration task, which is a large collection of different basic tasks. Then, we train a meta network based on large-scale datasets simulated with a basic task-aware design. To calibrate on the specific device, the network is further finetuned with a few measurements (few-shot) to achieve better performance. We conduct extensive experiments and demonstrate that our approach outperforms the baseline methods by a noticeable margin. Besides, our approach performs on par before calibration compared to previous work after calibration, which makes the zero-shot calibration feasible. It is thus a huge benefit in real-world scenarios where no calibration data is available for unknown devices. Last but not least, the resource analysis shows that our approach requires significantly less expenditure to deploy

large-scale devices in industry, which contributes to a huge saving potential both for the industry and the customers.

Overall, the proposed framework is comprehensive in the sense that it not only solves the kernel problems that exist in the current status as an all-in-one solution, but it also crosses through the whole product life cycle in the industry, namely from problem to solution, from development to deployment, with economic costs taken into consideration.

10.2 Future work

The future work of this dissertation is considered twofold: theoretical and practical.

10.2.1 Future work in fundamental research

Theoretically, or from the academic perspective, the potential for improvement of the proposed framework is still large. To accomplish the two major tasks in spectral analysis, further advanced neural network architectures can be tested, e.g., ResNet, Transformer, and so on. Besides, the optimizations of the framework are worth deeper digging. For instance, for the watermelon method, we are going to further investigate the influence of feature relevance in high dimensions, and the use of more advanced optimization solutions instead of greedy search is also of interest. And for the network pruning method, since we adopted a simplified version of watermelon because of resource and time constraints, we believe that the usage of the original version may improve the results due to its excellent ability to identify the difference between redundancy and relevance, especially for a high target ratio. Besides, we did not fine-tune the parameters of our approach (e.g., the number of Conv layers to be skipped) and network settings, which is one of our focuses for future work. Moreover, the implementation of the approach on other CNN architectures is also our next step. Moreover, we consider the optimization

of the meta learning training process an important step. Distributed training and continuous training can largely improve the efficiency of the whole workflow.

10.2.2 Future work in practices

Practically, especially for the industrial setting, there are also improvements that are expected to be conducted. First, the analysis of samples with layers is an important following work, which is not covered in this dissertation. For such systems, we need to determine both the element concentrations and the layer thicknesses, which adds up the complexity of the underlining physics model. It leads, therefore, to the next key discussion on the limitations of the DNN framework: can it do anything? Note that in the context of spectral analysis, there are physical limitations due to the optical systems. For example, as the thickness of the sample increases, the captured spectra will gradually saturate until there is no difference between two samples with thicknesses of, e.g., 60 um and 70 um. Another type of problem occurs when it comes to multi-layer settings. A sample with a structure of gold in the first layer and silver in the second layer may, theoretically, have the same spectrum as another sample with silver in the first layer and gold in the second layer, yet with different thicknesses. Hence, if the input data are the same, how can this framework correctly output two different results? Therefore, it is necessary to clarify the limitations of the framework and such cases need to be identified during the analysis so that the results are unbiased and meaningful.

Besides, although this dissertation realizes a cross-platform framework for the deployment on various platforms, efficient maintenance of the framework is of great interest to the industry. As the framework itself evolves, when new DNN models are available and an update is required to be consistent and worldwide, a well-functioning infrastructure plays a crucial role in these real-world scenarios.

Abbreviations and Symbols

Abbreviations

CNN	Convolutional Neural Network
DenseNet	Dense Convolutional Network
DNN	Deep Neural Network
FLOPs	Floating Point Operations
SOTA	State-Of-The-Art
LIBS	Laser-induced breakdown spectroscopy
XRF	X-ray fluorescence
MLP	Multilayer Perceptrons
BCE	Binary Cross-Entropy
MSE	Mean Squared Error
BP	Back Propagation
GPU	Graphics Card
SGD	Stochastic Gradient Descent
ROI	Regions of Interest
PCA	Principal Component Analysis

List of Figures

List of Tables

Publications

Journal paper

[1] W. Chen, W. Tian, X. Xie, and W. Stork, "Rgb image-and lidar-based 3d object detection under multiple lighting scenarios," *Automotive Innovation*, pp. 1–9, 2022.

Conference paper

[1] X. Xie, Y. Gao, and W. Stork, "Pa-dcgan: Efficient spectrum generation using physics-aware deep convolutional generative adversarial network with latent physical characteristics and constraints," in *2022 IEEE Symposium Series on Computational Intelligence (SSCI)*, 2022.

[2] X. Xie and W. Stork, "Large-scale spectral analysis for element quantification using deep neural networks," in *2022 International Joint Conference on Neural Networks (IJCNN)*, 2022, pp. 1–8.

[3] X. Xie, M. Jin, A. Chu, and W. Stork, "Minimal cost device calibration in spectral analysis via meta learning: Towards efficient deployment of deep neural networks in industry," in *2022 IEEE International Conference on Big Data (Big Data)*, 2022.

[4] X. Xie, T. Chen, and W. Stork, "Enabling real-time low-cost spectral analysis on edge devices with deep neural networks: a robust hybrid approach," in *2022 IEEE International Conference on Big Data (Big Data)*, 2022.

[5] X. Xie and W. Stork, “Watermelon: a novel feature selection method based on bayes error rate estimation and a new interpretation of feature relevance and redundancy,” in *2020 25th International Conference on Pattern Recognition (ICPR)*. IEEE, 2021, pp. 1360–1367.

[6] X. Xie, T. Chen, A. Chu, and W. Stork, “Efficient network pruning via feature selection,” in *2022 26th International Conference on Pattern Recognition (ICPR)*, 2022.

[7] X. Xie and W. Stork, “Efficient comprehensive element identification in large scale spectral analysis with interpretable dimension reduction,” in *2022 IEEE International Conference on Big Data (Big Data)*, 2022.

Bibliography

[1] J. Okonda, K. Angeyo, J. Mangala, and S. Kisia, "A nested multivariate chemometrics based calibration strategy for direct trace biometal analysis in soft tissue utilizing energy dispersive x-ray fluorescence (edxrf) and scattering spectrometry," *Applied Radiation and Isotopes*, vol. 129, pp. 49–56, 2017.

[2] F. Li, Q. Hu, X. Xu, L. Ge, X. Tang, and Z. Chen, "Bp neural network based on dropout applied to the edxrf quantitative analysis of heavy metal elements," in *Proceedings of the 3rd International Conference on Computer Science and Application Engineering*, 2019, pp. 1–5.

[3] P. E. Keller and R. T. Kouzes, "Gamma spectral analysis via neural networks," in *Proceedings of 1994 IEEE Nuclear Science Symposium-NSS'94*, vol. 1. IEEE, 1994, pp. 341–345.

[4] E. C. Ferreira, D. M. Milori, E. J. Ferreira, R. M. Da Silva, and L. Martin-Neto, "Artificial neural network for cu quantitative determination in soil using a portable laser induced breakdown spectroscopy system," *Spectrochimica Acta Part B: Atomic Spectroscopy*, vol. 63, no. 10, pp. 1216–1220, 2008.

[5] P. Inakollu, T. Philip, A. K. Rai, F.-Y. Yueh, and J. P. Singh, "A comparative study of laser induced breakdown spectroscopy analysis for element concentrations in aluminum alloy using artificial neural networks and calibration methods," *Spectrochimica Acta Part B: Atomic Spectroscopy*, vol. 64, no. 1, pp. 99–104, 2009.

[6] E. D'Andrea, S. Pagnotta, E. Grifoni, G. Lorenzetti, S. Legnaioli, V. Palleschi, and B. Lazzerini, "An artificial neural network approach to laser-induced

breakdown spectroscopy quantitative analysis," *Spectrochimica Acta Part B: Atomic Spectroscopy*, vol. 99, pp. 52–58, 2014.

[7] K.-X. Peng, J.-B. Yang, X.-G. Tuo, H. Du, and R.-X. Zhang, "Research on pgnaa adaptive analysis method with bp neural network," *Modern Physics Letters B*, vol. 30, no. 32n33, p. 1650386, 2016.

[8] K. Wang, P. Guo, and A.-L. Luo, "A new automated spectral feature extraction method and its application in spectral classification and defective spectra recovery," *Monthly Notices of the Royal Astronomical Society*, vol. 465, no. 4, pp. 4311–4324, 2017.

[9] W. B. Sohn, S. Y. Lee, and S. Kim, "Single-layer multiple-kernel-based convolutional neural network for biological raman spectral analysis," *Journal of Raman Spectroscopy*, vol. 51, no. 3, pp. 414–421, 2020.

[10] J. Vrielink, R. M. Tiggelaar, J. G. Gardeniers, and L. Lefferts, "Applicability of x-ray fluorescence spectroscopy as method to determine thickness and composition of stacks of metal thin films: A comparison with imaging and profilometry," *Thin Solid Films*, vol. 520, no. 6, pp. 1740–1744, 2012.

[11] S. Arzhantsev, X. Li, and J. F. Kauffman, "Rapid limit tests for metal impurities in pharmaceutical materials by x-ray fluorescence spectroscopy using wavelet transform filtering," *Analytical chemistry*, vol. 83, no. 3, pp. 1061–1068, 2011.

[12] M. Saritha and V. Nampoori, "Identification of spectral lines of elements using artificial neural networks," *Microchemical Journal*, vol. 91, no. 2, pp. 170–175, 2009.

[13] M. Bieroza, A. Baker, and J. Bridgeman, "Classification and calibration of organic matter fluorescence data with multiway analysis methods and artificial neural networks: an operational tool for improved drinking water treatment," *Environmetrics*, vol. 22, no. 3, pp. 256–270, 2011.

[14] N. M. Peleato, R. L. Legge, and R. C. Andrews, "Neural networks for dimensionality reduction of fluorescence spectra and prediction of drinking water disinfection by-products," *Water research*, vol. 136, pp. 84–94, 2018.

[15] E. A. Hernández-Caraballo and L. M. Marcó-Parra, "Direct analysis of blood serum by total reflection x-ray fluorescence spectrometry and application of an artificial neural network approach for cancer diagnosis," *Spectrochimica Acta Part B: Atomic Spectroscopy*, vol. 58, no. 12, pp. 2205–2213, 2003.

[16] M. Kaniu, K. Angeyo, A. Mwala, and F. Mwangi, "Energy dispersive x-ray fluorescence and scattering assessment of soil quality via partial least squares and artificial neural networks analytical modeling approaches," *Talanta*, vol. 98, pp. 236–240, 2012.

[17] J. El Haddad, D. Bruyère, A. Ismaël, G. Gallou, V. Laperche, K. Michel, L. Canioni, and B. Bousquet, "Application of a series of artificial neural networks to on-site quantitative analysis of lead into real soil samples by laser induced breakdown spectroscopy," *Spectrochimica Acta Part B: Atomic Spectroscopy*, vol. 97, pp. 57–64, 2014.

[18] T. F. Boucher, M. V. Ozanne, M. L. Carmosino, M. D. Dyar, S. Mahadevan, E. A. Breves, K. H. Lepore, and S. M. Clegg, "A study of machine learning regression methods for major elemental analysis of rocks using laser-induced breakdown spectroscopy," *Spectrochimica Acta Part B: Atomic Spectroscopy*, vol. 107, pp. 1–10, 2015.

[19] Z. Zhong, S. Tang, G. Peng, and Y. Zhang, "A novel quantitative spectral analysis method based on parallel bp neural network for dissolved gas in transformer oil," in *2016 IEEE PES Asia-Pacific Power and Energy Engineering Conference (APPEEC)*. IEEE, 2016, pp. 1979–1983.

[20] M. Unal and O. Akkus, "Raman spectral classification of mineral-and collagen-bound water's associations to elastic and post-yield mechanical properties of cortical bone," *Bone*, vol. 81, pp. 315–326, 2015.

[21] H. P. Singh, R. K. Gulati, and R. Gupta, "Stellar spectral classification using principal component analysis and artificial neural networks," *Monthly Notices of the Royal Astronomical Society*, vol. 295, no. 2, pp. 312–318, 1998.

[22] M. Kamuda, J. Stinnett, and C. Sullivan, "Automated isotope identification algorithm using artificial neural networks," *IEEE Transactions on Nuclear Science*, vol. 64, no. 7, pp. 1858–1864, 2017.

[23] X. Yu, L. Xu, L. Ma, Z. Chen, and Y. Yan, "Solar radio spectrum classification with lstm," in *2017 IEEE international conference on multimedia & expo workshops (ICMEW)*. IEEE, 2017, pp. 519–524.

[24] T. Salge, R. Neumann, C. Andersson, and M. Patzschke, "Advanced mineral classification using feature analysis and spectrum imaging with eds," in *Proceedings: International Mining Congress and Exhibition, 23rd, Turkey, UCTEA Chamber of Mining Engineers of Turkey*, vol. 357, 2013.

[25] T. Schoonjans, A. Brunetti, B. Golosio, M. S. del Rio, V. A. Solé, C. Ferrero, and L. Vincze, "The xraylib library for x-ray–matter interactions. recent developments," *Spectrochimica Acta Part B: Atomic Spectroscopy*, vol. 66, no. 11-12, pp. 776–784, 2011.

[26] R. Sitko, "Quantitative x-ray fluorescence analysis of samples of less than 'infinite thickness': difficulties and possibilities," *Spectrochimica Acta Part B: Atomic Spectroscopy*, vol. 64, no. 11-12, pp. 1161–1172, 2009.

[27] [Online]. Available: https://www.princetoninstruments.com/learn/laser-induced-breakdown-spectroscopy

[28] A. Miliolek, V. Palleschi, and I. Schechter, "Laser-induced breakdown spectroscopy: Fundamentals and applications," 2006.

[29] [Online]. Available: https://www.helmut-fischer.com/techniques/basics-of-xrf-x-ray-fluorescence-analysis

[30] [Online]. Available: https://de.wikipedia.org/wiki/RÃűntgenfluoreszenzanalyse

[31] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[32] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[33] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 580–587.

[34] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, pp. 1097–1105, 2012.

[35] A. LeNail, "Nn-svg: Publication-ready neural network architecture schematics," *Journal of Open Source Software*, vol. 4, no. 33, p. 747, 2019. [Online]. Available: https://doi.org/10.21105/joss.00747

[36] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.

[37] Chabacano. [Online]. Available: https://en.wikipedia.org/wiki/File:Overfitting.svg

[38] Gringer. [Online]. Available: https://en.wikipedia.org/wiki/File:Overfitting_svg.svg

[39] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous

systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[40] M. Hardt, B. Recht, and Y. Singer, "Train faster, generalize better: Stability of stochastic gradient descent," in *International conference on machine learning*. PMLR, 2016, pp. 1225–1234.

[41] A. C. Wilson, R. Roelofs, M. Stern, N. Srebro, and B. Recht, "The marginal value of adaptive gradient methods in machine learning," *Advances in neural information processing systems*, vol. 30, 2017.

[42] D. Choi, C. J. Shallue, Z. Nado, J. Lee, C. J. Maddison, and G. E. Dahl, "On empirical comparisons of optimizers for deep learning," *arXiv preprint arXiv:1910.05446*, 2019.

[43] J. Li, K. Cheng, S. Wang, F. Morstatter, R. P. Trevino, J. Tang, and H. Liu, "Feature selection: A data perspective," *ACM Comput. Surv.*, vol. 50, no. 6, Dec. 2017. [Online]. Available: https://doi.org/10.1145/3136625

[44] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *Journal of machine learning research*, vol. 3, no. Mar, pp. 1157–1182, 2003.

[45] J. R. Vergara and P. A. Estévez, "A review of feature selection methods based on mutual information," *Neural Computing and Applications*, vol. 24, no. 1, pp. 175–186, Jan. 2014. [Online]. Available: https://doi.org/10.1007/s00521-013-1368-0

[46] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern classification*. John Wiley & Sons, 2012.

[47] F. Nie, S. Xiang, Y. Jia, C. Zhang, and S. Yan, "Trace ratio criterion for feature selection." in *AAAI*, vol. 2, 2008, pp. 671–676.

[48] M. Robnik-Šikonja and I. Kononenko, "Theoretical and empirical analysis of ReliefF and RReliefF," *Machine learning*, vol. 53, no. 1-2, pp. 23–69, 2003.

[49] J. Liu, S. Ji, and J. Ye, "Multi-task feature learning via efficient l2, 1-norm minimization," *arXiv preprint arXiv:1205.2631*, 2012.

[50] F. Nie, H. Huang, X. Cai, and C. H. Ding, "Efficient and robust feature selection via joint l2, 1-norms minimization," in *Advances in neural information processing systems*, 2010, pp. 1813–1821.

[51] S. Wright, "The interpretation of population structure by f-statistics with special regard to systems of mating," *Evolution*, vol. 19, no. 3, pp. 395–420, 1965.

[52] L. Ceriani and P. Verme, "The origins of the gini index: extracts from variabilità e mutabilità (1912) by corrado gini," *The Journal of Economic Inequality*, vol. 10, no. 3, pp. 421–443, Sep. 2012. [Online]. Available: https://doi.org/10.1007/s10888-011-9188-x

[53] D. D. Lewis, "Feature selection and feature extraction for text categorization," in *Proceedings of the workshop on Speech and Natural Language*. Association for Computational Linguistics, 1992, pp. 212–217.

[54] R. Battiti, "Using mutual information for selecting features in supervised neural net learning," *IEEE Transactions on neural networks*, vol. 5, no. 4, pp. 537–550, 1994.

[55] H. Peng, F. Long, and C. Ding, "Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 27, no. 8, pp. 1226–1238, 2005.

[56] D. Lin and X. Tang, "Conditional infomax learning: an integrated framework for feature extraction and fusion," in *European conference on computer vision*. Springer, 2006, pp. 68–82.

[57] H. H. Yang and J. Moody, "Data visualization and feature selection: New algorithms for nongaussian data," in *Advances in neural information processing systems*, 2000, pp. 687–693.

[58] P. E. Meyer, C. Schretter, and G. Bontempi, "Information-theoretic feature selection in microarray data using variable complementarity," *IEEE Journal of Selected Topics in Signal Processing*, vol. 2, no. 3, pp. 261–274, 2008.

[59] M. Vidal-Naquet and S. Ullman, "Object recognition with informative features and linear classification." in *ICCV*, vol. 3, 2003, p. 281.

[60] F. Fleuret, "Fast binary feature selection with conditional mutual information," *Journal of Machine learning research*, vol. 5, no. Nov, pp. 1531–1555, 2004.

[61] P. E. Meyer and G. Bontempi, "On the use of variable complementarity for feature selection in cancer classification," in *Workshops on applications of evolutionary computation*. Springer, 2006, pp. 91–102.

[62] L. Yu and H. Liu, "Feature selection for high-dimensional data: A fast correlation-based filter solution," in *Proceedings of the 20th international conference on machine learning (ICML-03)*, 2003, pp. 856–863.

[63] M. A. Hall and L. A. Smith, "Feature selection for machine learning: comparing a correlation-based filter approach to the wrapper." in *FLAIRS conference*, vol. 1999, 1999, pp. 235–239.

[64] P. A. Estévez, M. Tesmer, C. A. Perez, and J. M. Zurada, "Normalized mutual information feature selection," *IEEE Transactions on neural networks*, vol. 20, no. 2, pp. 189–201, 2009.

[65] K. Tumer and J. Ghosh, "Bayes error rate estimation using classifier ensembles," *International Journal of Smart Engineering System Design*, vol. 5, no. 2, pp. 95–109, 2003.

[66] S. Xu, J. Dai *et al.*, "Semi-supervised feature selection by mutual information based on kernel density estimation," in *2018 24th International Conference on Pattern Recognition (ICPR)*. IEEE, 2018, pp. 818–823.

[67] N. X. Vinh, J. Epps, and J. Bailey, "Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction

for chance," *Journal of Machine Learning Research*, vol. 11, no. Oct, pp. 2837–2854, 2010.

[68] A. F. McDaid, D. Greene, and N. Hurley, "Normalized mutual information to evaluate overlapping community finding algorithms," *arXiv preprint arXiv:1110.2515*, 2011.

[69] G. Brown, A. Pocock, M.-J. Zhao, and M. Luján, "Conditional likelihood maximisation: a unifying framework for information theoretic feature selection," *Journal of machine learning research*, vol. 13, no. Jan, pp. 27–66, 2012.

[70] A. Pérez, P. Larrañaga, and I. Inza, "Bayesian classifiers based on kernel density estimation: Flexible classifiers," *International Journal of Approximate Reasoning*, vol. 50, no. 2, pp. 341–362, 2009.

[71] A. Jakulin, "Machine learning based on attribute interactions," Ph.D. dissertation, Univerza v Ljubljani, 2005.

[72] L. Yu, Y. Han, and M. E. Berens, "Stable gene selection from microarray data via sample weighting," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 9, no. 1, pp. 262–272, 2011.

[73] G. Roffo, S. Melzi, and M. Cristani, "Infinite feature selection," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 4202–4210.

[74] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[75] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *Journal of Machine learning research*, vol. 7, no. Jan, pp. 1–30, 2006.

[76] M. Fernández-Delgado, E. Cernadas, S. Barro, and D. Amorim, "Do we need hundreds of classifiers to solve real world classification problems?" *The journal of machine learning research*, vol. 15, no. 1, pp. 3133–3181, 2014.

[77] G. Daniel, F. Ceraudo, O. Limousin, D. Maier, and A. Meuris, "Automatic and real-time identification of radionuclides in gamma-ray spectra: a new method based on convolutional neural network trained with synthetic data set," *IEEE Transactions on Nuclear Science*, vol. 67, no. 4, pp. 644–653, 2020.

[78] M. Kamuda and C. J. Sullivan, "An automated isotope identification and quantification algorithm for isotope mixtures in low-resolution gamma-ray spectra," *Radiation Physics and Chemistry*, vol. 155, pp. 281–286, 2019.

[79] E. Yoshida, K. Shizuma, S. Endo, and T. Oka, "Application of neural networks for the analysis of gamma-ray spectra measured with a ge spectrometer," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 484, no. 1-3, pp. 557–563, 2002.

[80] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3431–3440.

[81] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural language processing (almost) from scratch," *Journal of machine learning research*, vol. 12, no. ARTICLE, pp. 2493–2537, 2011.

[82] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[83] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[84] T. Liang, J. Glossner, L. Wang, S. Shi, and X. Zhang, "Pruning and quantization for deep neural network acceleration: A survey," *Neurocomputing*, vol. 461, pp. 370–403, 2021.

[85] D. Blalock, J. J. Gonzalez Ortiz, J. Frankle, and J. Guttag, "What is the state of neural network pruning?" *Proceedings of machine learning and systems*, vol. 2, pp. 129–146, 2020.

[86] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," *arXiv preprint arXiv:1506.02626*, 2015.

[87] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," *arXiv preprint arXiv:1803.03635*, 2018.

[88] V. Sehwag, S. Wang, P. Mittal, and S. Jana, "Hydra: Pruning adversarially robust neural networks," *Advances in Neural Information Processing Systems*, vol. 33, pp. 19 655–19 666, 2020.

[89] N. Lee, T. Ajanthan, and P. H. Torr, "Snip: Single-shot network pruning based on connection sensitivity," *arXiv preprint arXiv:1810.02340*, 2018.

[90] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, "Learning efficient convolutional networks through network slimming," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 2736–2744.

[91] Y. Wang, X. Zhang, L. Xie, J. Zhou, H. Su, B. Zhang, and X. Hu, "Pruning from scratch," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 07, 2020, pp. 12 273–12 280.

[92] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 1389–1397.

[93] Y. Li, S. Gu, K. Zhang, L. V. Gool, and R. Timofte, "Dhp: Differentiable meta pruning via hypernetworks," in *European Conference on Computer Vision*. Springer, 2020, pp. 608–624.

[94] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," *arXiv preprint arXiv:1608.08710*, 2016.

[95] M. Lin, R. Ji, Y. Wang, Y. Zhang, B. Zhang, Y. Tian, and L. Shao, "Hrank: Filter pruning using high-rank feature map," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 1529–1538.

[96] C. M. J. Tan and M. Motani, "Dropnet: Reducing neural network complexity via iterative pruning," in *International Conference on Machine Learning*. PMLR, 2020, pp. 9356–9366.

[97] C. Wang, G. Zhang, and R. Grosse, "Picking winning tickets before training by preserving gradient flow," *arXiv preprint arXiv:2002.07376*, 2020.

[98] M. Lin, L. Cao, S. Li, Q. Ye, Y. Tian, J. Liu, Q. Tian, and R. Ji, "Filter sketch for network pruning," *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[99] X. Dong, S. Chen, and S. Pan, "Learning to prune deep neural networks via layer-wise optimal brain surgeon," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[100] J.-H. Luo, J. Wu, and W. Lin, "Thinet: A filter level pruning method for deep neural network compression," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 5058–5066.

[101] Z. Liu, H. Mu, X. Zhang, Z. Guo, X. Yang, K.-T. Cheng, and J. Sun, "Metapruning: Meta learning for automatic neural network channel pruning," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

[102] X. Zhang, H. Liu, Z. Zhu, and Z. Xu, "Learning to search efficient densenet with layer-wise pruning," in *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2020, pp. 1–8.

[103] S. Lin, R. Ji, C. Yan, B. Zhang, L. Cao, Q. Ye, F. Huang, and D. Doermann, "Towards optimal structured cnn pruning via generative adversarial learning,"

in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 2790–2799.

[104] B. Li, B. Wu, J. Su, and G. Wang, "Eagleeye: Fast sub-net evaluation for efficient neural network pruning," in *European conference on computer vision*. Springer, 2020, pp. 639–654.

[105] S. Gao, F. Huang, W. Cai, and H. Huang, "Network pruning via performance maximization," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 9270–9280.

[106] W. Wang, M. Chen, S. Zhao, L. Chen, J. Hu, H. Liu, D. Cai, X. He, and W. Liu, "Accelerate cnns from three dimensions: A comprehensive pruning framework," in *International Conference on Machine Learning*. PMLR, 2021, pp. 10 717–10 726.

[107] Z. Wang, F. Li, G. Shi, X. Xie, and F. Wang, "Network pruning using sparse learning and genetic algorithm," *Neurocomputing*, vol. 404, pp. 247–256, 2020.

[108] R. Yu, A. Li, C.-F. Chen, J.-H. Lai, V. I. Morariu, X. Han, M. Gao, C.-Y. Lin, and L. S. Davis, "Nisp: Pruning networks using neuron importance score propagation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 9194–9203.

[109] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

[110] Y. Tang, Y. Wang, Y. Xu, D. Tao, C. Xu, C. Xu, and C. Xu, "Scop: Scientific control for reliable neural network pruning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 10 936–10 947, 2020.

[111] Y. Chen, X. Wen, Y. Zhang, and W. Shi, "Ccprune: Collaborative channel pruning for learning compact convolutional networks," *Neurocomputing*, vol. 451, pp. 35–45, 2021.

[112] Z. Guo, Y. Xiao, W. Liao, P. Veelaert, and W. Philips, "Flops-efficient filter pruning via transfer scale for neural network acceleration," *Journal of Computational Science*, vol. 55, p. 101459, 2021.

[113] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," *Advances in neural information processing systems*, vol. 29, 2016.

[114] P. Singh, V. K. Verma, P. Rai, and V. Namboodiri, "Leveraging filter correlations for deep model compression," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2020, pp. 835–844.

[115] H. Tanaka, D. Kunin, D. L. Yamins, and S. Ganguli, "Pruning neural networks without any data by iteratively conserving synaptic flow," *Advances in Neural Information Processing Systems*, vol. 33, pp. 6377–6389, 2020.

[116] Y. LeCun, J. Denker, and S. Solla, "Optimal brain damage," *Advances in neural information processing systems*, vol. 2, 1989.

[117] B. Hassibi and D. Stork, "Second order derivatives for network pruning: Optimal brain surgeon," *Advances in neural information processing systems*, vol. 5, 1992.

[118] L. Liu, S. Zhang, Z. Kuang, A. Zhou, J.-H. Xue, X. Wang, Y. Chen, W. Yang, Q. Liao, and W. Zhang, "Group fisher pruning for practical network compression," in *International Conference on Machine Learning*. PMLR, 2021, pp. 7021–7032.

[119] C. Zhao, B. Ni, J. Zhang, Q. Zhao, W. Zhang, and Q. Tian, "Variational convolutional neural network pruning," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 2775–2784.

[120] M. Lin, R. Ji, S. Li, Y. Wang, Y. Wu, F. Huang, and Q. Ye, "Network pruning using adaptive exemplar filters," *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[121] Z. Wang, C. Li, and X. Wang, "Convolutional neural network pruning with structural redundancy reduction," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 14 913–14 922.

[122] [Online]. Available: https://github.com/Tzutori/watermelon-feature-selection

[123] Z. Huang and N. Wang, "Data-driven sparse structure selection for deep neural networks," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 304–320.

[124] W. Wang, S. Zhao, M. Chen, J. Hu, D. Cai, and H. Liu, "Dbp: discrimination based block-level pruning for deep model acceleration," *arXiv preprint arXiv:1912.10178*, 2019.

[125] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 6869–6898, 2017.

[126] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 2704–2713.

[127] M. Nagel, R. A. Amjad, M. Van Baalen, C. Louizos, and T. Blankevoort, "Up or down? adaptive rounding for post-training quantization," in *International Conference on Machine Learning*. PMLR, 2020, pp. 7197–7206.

[128] R. Zhao, Y. Hu, J. Dotzel, C. De Sa, and Z. Zhang, "Improving neural network quantization without retraining using outlier channel splitting," in *International conference on machine learning*. PMLR, 2019, pp. 7543–7552.

[129] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *International conference on machine learning*. PMLR, 2017, pp. 1126–1135.

[130] S. Bozinovski, "Reminder of the first paper on transfer learning in neural networks, 1976," *Informatica*, vol. 44, no. 3, 2020.

[131] D. Cai, X. He, J. Han, and T. S. Huang, "Graph regularized nonnegative matrix factorization for data representation," *IEEE transactions on pattern analysis and machine intelligence*, vol. 33, no. 8, pp. 1548–1560, 2010.

[132] D. Cai, X. He, Y. Hu, J. Han, and T. Huang, "Learning a spatially smooth subspace for face recognition," in *2007 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2007, pp. 1–7.

[133] K. Lang, "Newsweeder: Learning to filter netnews," in *Proceedings of the Twelfth International Conference on Machine Learning*, 1995, pp. 331–339.

[134] J. C. Davis and R. J. Sampson, *Statistics and data analysis in geology*. Wiley New York et al., 1986, vol. 646.

[135] M. Lin, R. Ji, Y. Zhang, B. Zhang, Y. Wu, and Y. Tian, "Channel pruning via automatic structure search," *arXiv preprint arXiv:2001.08565*, 2020.

[136] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[137] F. Chollet *et al.*, "Keras," https://keras.io, 2015.

[138] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient inference," *arXiv preprint arXiv:1611.06440*, 2016.

[139] D. Molchanov, A. Ashukha, and D. Vetrov, "Variational dropout sparsifies deep neural networks," in *International Conference on Machine Learning*. PMLR, 2017, pp. 2498–2507.

[140] T. Schoonjans, V. A. Solé, L. Vincze, M. S. del Rio, K. Appel, and C. Ferrero, "A general monte carlo simulation of energy-dispersive x-ray fluorescence spectrometers—part 6. quantification through iterative simulations," *Spectrochimica Acta Part B: Atomic Spectroscopy*, vol. 82, pp. 36–41, 2013.

[141] V. Solé, E. Papillon, M. Cotte, P. Walter, and J. Susini, "A multiplatform code for the analysis of energy-dispersive x-ray fluorescence spectra," *Spectrochimica Acta Part B: Atomic Spectroscopy*, vol. 62, no. 1, pp. 63–68, 2007.

[142] F. Li, L. Ge, Z. Tang, Y. Chen, and J. Wang, "Recent developments on xrf spectra evaluation," *Applied Spectroscopy Reviews*, vol. 55, no. 4, pp. 263–287, 2020.

[143] Y. Shi, K. Davaslioglu, Y. E. Sagduyu, W. C. Headley, M. Fowler, and G. Green, "Deep learning for rf signal classification in unknown and dynamic spectrum environments," in *2019 IEEE International Symposium on Dynamic Spectrum Access Networks (DySPAN)*. IEEE, 2019, pp. 1–10.

[144] Google. [Online]. Available: https://www.tensorflow.org/mobile/tflite

[145] [Online]. Available: https://github.com/Tzutori/watermelon-feature-selection

[146] Y. Choi, J. Choi, M. El-Khamy, and J. Lee, "Data-free network quantization with adversarial knowledge distillation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 2020, pp. 710–711.

[147] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean, "Efficient neural architecture search via parameters sharing," in *International Conference on Machine Learning*. PMLR, 2018, pp. 4095–4104.

[148] Y. Chen, G. Meng, Q. Zhang, X. Zhang, L. Song, S. Xiang, and C. Pan, "Joint neural architecture search and quantization," *arXiv preprint arXiv:1811.09426*, 2018.

[149] S. Jung, C. Son, S. Lee, J. Son, J.-J. Han, Y. Kwak, S. J. Hwang, and C. Choi, "Learning to quantize deep networks by optimizing quantization intervals with task loss," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 4350–4359.

[150] D. Zhang, J. Yang, D. Ye, and G. Hua, "Lq-nets: Learned quantization for highly accurate and compact deep neural networks," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 365–382.

[151] L. Deng, G. Li, S. Han, L. Shi, and Y. Xie, "Model compression and hardware acceleration for neural networks: A comprehensive survey," *Proceedings of the IEEE*, vol. 108, no. 4, pp. 485–532, 2020.

[152] X. Dong, J. Huang, Y. Yang, and S. Yan, "More is less: A more complicated network with less inference complexity," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[153] M. Figurnov, M. D. Collins, Y. Zhu, L. Zhang, J. Huang, D. Vetrov, and R. Salakhutdinov, "Spatially adaptive computation time for residual networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[154] A.-H. Phan, K. Sobolev, K. Sozykin, D. Ermilov, J. Gusak, P. Tichavský, V. Glukhov, I. Oseledets, and A. Cichocki, "Stable low-rank tensor decomposition for compression of convolutional neural network," in *Computer Vision – ECCV 2020*. Cham: Springer International Publishing, 2020, pp. 522–539.

[155] S. Anwar, K. Hwang, and W. Sung, "Structured pruning of deep convolutional neural networks," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 13, no. 3, pp. 1–18, 2017.