AGH
Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie
Faculty of Physics and Applied Computer Science

Bachelor thesis

Eryk Zarębski

field of study: Applied Computer Science

An algorithm for classifying chemical elements using data collected with the X-ray fluorescence technique

Thesis supervisor: prof. dr hab. inż. Tomasz Szumlak

Kraków, January 14, 2024

Abstract

In recent years, the application of deep learning models has demonstrated remarkable efficiency in addressing complex tasks that were beyond the reach of traditional methods. Meanwhile, a novel method for non-invasive X-Ray Fluorescence (XRF) data acquisition has been developed. Despite the existence of algorithms for XRF data analysis, there is still a need for an algorithm that could surpass their limitations. For instance, the widely-used Region of Interest (ROI) technique encounters difficulties in detecting less abundant elements in a spectra, differentiating signals with closely resembling characteristic energies, and providing information about their true distribution within the spectrum.

This thesis presents an effort to develop a Vision Transformer (ViT)-based model for the multi-label classification of chemical elements within XRF spectra. The classification process was supported by clustering spectra in order to mitigate their noisiness, generating artificial data for model training in the absence of real labeled data, and implementing a preprocessing pipeline aimed at ensuring reliable results.

Although the trained model fell short of surpassing established methods, it offers hope that with greater effort, it could become a reliable solution.

Contents

1 Introduction

1.1 Motivation

The work presented is built upon contributions made by dr. inż. Bartłomiej Łach in his doctoral thesis entitled "Rozwój systemu detekcyjnego do obrazowania przestrzennego rozkładu pierwiastków metodą fluorescencji rentgenowskiej" [Łach(2022)] (eng. *Development of a detection system for spatial imaging of element distribution using X-ray fluorescence*).

His dissertation was dedicated to development of system for non-invasive analysis of works of art that uses X-ray radiation to carry out measurements. The system presented was designed to provide a map of element distribution in the surface the object, offering essential information for analyzing the pigments used in the artwork. The results provided by analysis could be source of important knowledge for conservators of monuments and art, allowing them to determine the object's state of preservation, quality of past maintenance processes, the work's authenticity and to expand state of knowledge.

Currently the most popular technique to perform measurements using X-ray - MA-XRF (Macro X-Ray Fluorescence), operate by scanning selected parts of art point by point. This method is characterized by high spatial and energetic resolution. However, due to usage of polycapillary lenses it is limited to performing scans on 2D surfaces. Another drawback this method shows is long measurement time.

An alternative method involves scanning not just a single point but multiple points within a specified area of the surface, by utilizing FF-XRF (Full-Field X-Ray Fluorescence). This exact approach was implemented by researchers at AGH University of Krakow in the DETART (Detector for Art) data acquisition system. DETART not only can scan large portion of the surface (over 10^5 pixels at the same time), but it is also capable of performing scans of 3D objects without losing spatial resolution thanks to (theoretically) near infinite depth of field provided by pinhole camera.

After data acquisition, the analysis can be performed. Łach proposed three different methods for analyzing data: "standard" ROI (Region Of Interest) method and two different machine learning algorithms: PCA (Principal Components Analysis) and NFM (Non-negative Matrix Factorization). Each of them had its advantages and disadvantages. In the end, there may not be a single method that is universally superior. However, employing multiple methods allows us to draw more comprehensive conclusions, leveraging the strengths of each and compensating for their respective limitations.

1.2 Concept

Concept of this thesis is heavily inspired by the work presented in [Jones et al.(2022)]. The authors of this article introduced the idea of a CNN (Convolutional Neural Network) model

tasked with classifying spectra measured with XRF into the one of the pigment classes. Researchers trained the model on synthetic data first, achieving an accuracy of 55% on real XRF spectra, and 96% after applying transfer learning by using portion of real XRF samples. However, there are some potential downsides to consider for this model:

- The model can only classify pigments on which it has been trained.
- It does not provide explicit information about each element.

A potential solution to these problems may be to shift the focus from classifying pigments to classifying elements. A multi-label classification approach might be used to assess existence of each learned element independently.

1.3 Introduction to XRF

1.3.1 Physics Foundations

The XRF method utilizes X-ray radiation to irradiate objects, causing them to emit photons that are specific to each present element. Characteristic X-ray fluorescence radiation is type of secondary radiation generated in matter under the influence of radiation from an external source - X-ray tube in research setting. The method is considered harmless when used over short periods of time. Therefore, it can be used to study delicate objects, such as art, monuments, geological and biological samples.

The fundamental physics behind the whole process is the phenomenon of photoelectric absorption. In this process, one of photons transfers all its energy to one of the electrons bound to the inner electron shell of an atom. Consequently the electron is "knocked out" from the shell, leaving a vacant state. Absorption can only occur if the energetic criterion is satisfied; that is, the energy that photon transfers to the electron must be greater than the binding energy of electron on the specific shell.

After the emission of the electron, the atom is excited, which results in electron jumping from a higher energy shell to the vacancy left by the emitted electron, creating its own vacancy. The entire process continues until the atom reaches an equilibrium state.

Electrons on outer shells have higher energies than electrons on lower shells. According to the law of conservation of energy, something must happen to the excess energy - the emission in the form of characteristic photon. The binding energies on each shell are unique features of every element and are related to the energies of characteristic photons. These energies are also known as spectral lines. For example, when an electron jumps from shell L to K, it releases energy equal to the difference between their binding energies, resulting in emission of the K_α line. Analogically, when it jumps from shell M to K, it produces the K_β emission line, and so on. Therefore, having *a priori* knowledge about energies of specific spectral lines, one should be able to identify elements present in measured spectrum - see Figure [1].

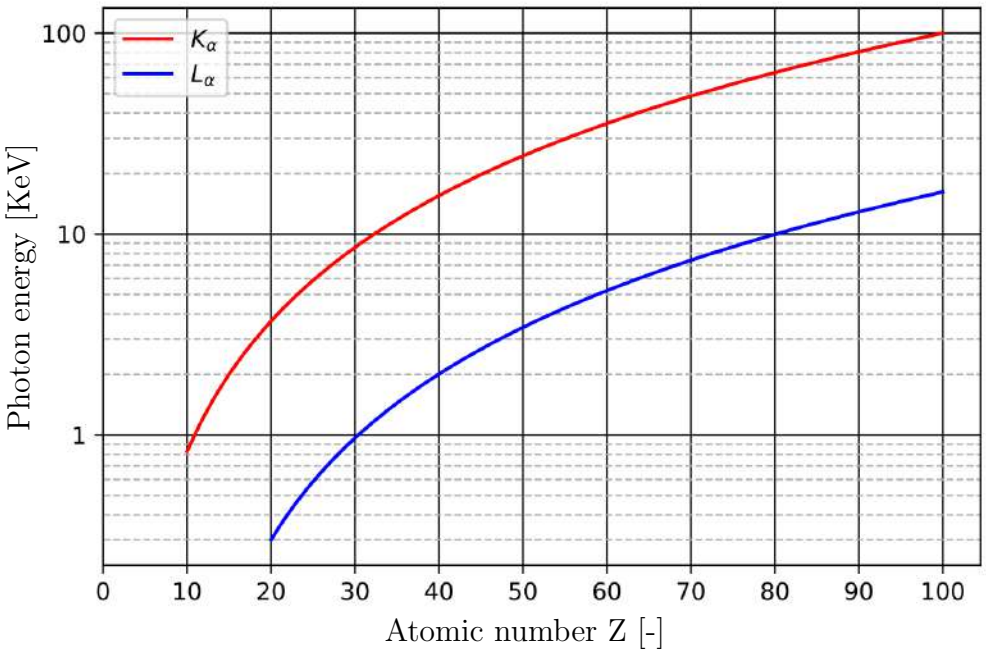


Figure 1: The dependence of the energy of photons emitted due to transitions of electrons from the L to K level (K_α spectral line) and M to L level (L_α spectral line) as a function of the atomic number Z. Source: [Łach(2022)]

At the same time competitive process can occur - Auger electron ejection. This process involves the following events [Pavan M. V. et al.(2023)]:

1. An electron on the inner shell is ejected by a photon, creating a vacancy.

2. A secondary electron drops down to fill the vacancy.

3. If sufficient energy is emitted, a tertiary electron is ejected (Auger electron).

For example, vacancy may be created on shell K, leading to transition from shell L to K and then emission of Auger electron from shell L (or M, or any higher). Emission of these electrons is an undesirable effect in context of XRF, as it is responsible for the presence of background radiation in the measured spectrum.

1.3.2 Imaging Techniques

XRF is primarily employed for quantitative measurements of elements in studied samples. Handheld scanners with high energy resolution are available on the market and suitable for simple measurements of elements present in metal alloys, geodes and other materials. However, if one is interested in spatial imaging, device like that will (probably) not suffice. The demand for spatial imaging led to the evolution of X-ray scanning microscopy - see Figure [2].

(a) Example of handheld XRF scanner. Source: [Bruker(2023)]

(b) MA-XRF spatial imaging concept. Source: [Łach(2022)]

Figure 2: Scanning single spatial position

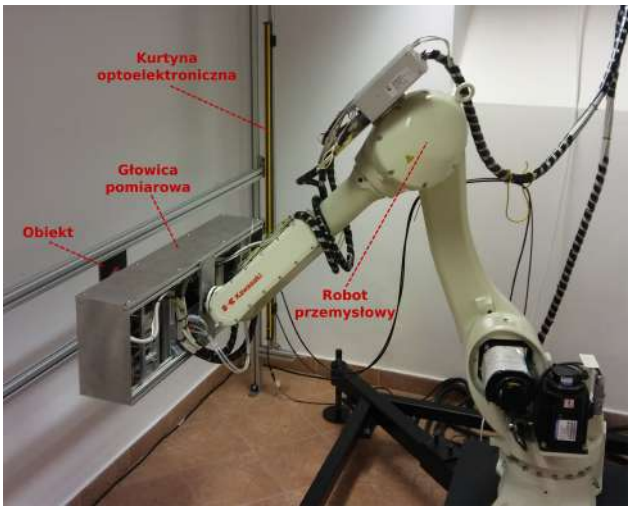
Most popular spatial imaging technique involve scanning the object point by point. It goes under a name of MA-XRF (Macro X-Ray Fluorescence) and/or μ-XRF (Micro X-Ray Fluorescence), depending on the area and resolution of the scanning apparatus.

This method comes with crucial disadvantage, as it is sensitive to changes in distance from the object to the polycapillary lens that focuses X-rays. When the distance changes, the resolution also changes, leading to serious distortions in measurements. Scanning point by point also comes with long measurement time. For example, scanning a painting with an area of 65×45 cm^2, a resolution of 1300×900 pixels (each pixel 500μm in size), and a dwell time of 10 ms per pixel could take about 3.5 hours [Matthias et al.(2013)].

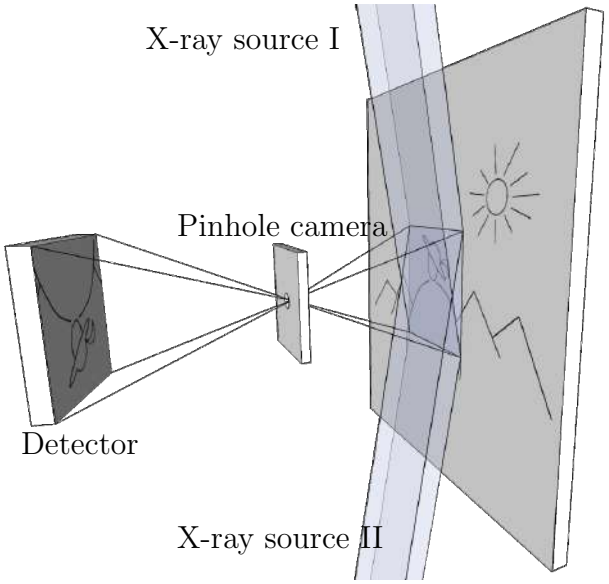
On the brighter side:

- The detector used doesn't have to be position-sensitive.
- The results have very high energetic resolution, allowing trouble-free identification of elements.
- Spectrometers are commonly available.

Due to the demand for eliminating weaknesses in MA-XRF, an alternative method has been developed - FF-XRF (Full-Field X-Ray Fluorescence). It is characterized by replacing polycapillary optics with a wide, homogenous beam of radiation paired with a pinhole camera. This combination guarantees high depth of field, enabling the scanning of uneven surfaces and 3D objects. When combined with good position-sensitive detector, it allows for scanning multiple points of surface simultaneously. However, in contrary to the MA-XRF, choosing the right detector is very tricky. Detectors with good spatial resolution are lacking in terms of energetic resolution. For this specific use case, choosing the best detector was challenging, but in the end, it was decided to use slightly modified GEM (Gas Electron Multiplier) detector. Implementation of FF-XRF can be seen in Figure [3].



(a) Prototype FF-XRF scanner. Robotic arm allows for scanning predefined parts of an object. Source: [Łach(2022)]



(b) FF-XRF spatial imaging concept. Source: [Łach(2022)]

Figure 3: Scanning multiple spatial positions

2 Methodology

2.1 Used Technologies

All code developed during the project was written in the `python3` programming language. Google Colab was utilized as runtime environment, as it provides resources that allows rapid testing of DNN (Deep Neural Network) implementations. Google Colab also enabled easy sharing of interactive notebooks with the thesis supervisor.

The primary technologies and libraries used in the project include:

- `python3` programming language.
- `pytorch` - Chosen as the deep learning framework due to the author's familiarity with it and its widespread adoption among researchers (over 60% of new paper implementations use `pytorch` [Papers with Code(2023)]).
- `hdbscan` - Implementation of the HDBSCAN (Hierarchical Density-Based Spatial Clustering of Applications with Noise) algorithm for data clusterization.
- `umap-learn` - Implementation of the UMAP (Uniform Manifold Approximation and Projection) algorithm for dimensionality reduction.
- `minisom` - Implementation of a SOM (Self-Organizing Map) algorithm for data clusterization.
- `scikit-learn`, `scipy`, `matplotlib`, `pandas`, `numpy` - Common tools used for data analysis in the `python` ecosystem.

2.2 Key Algorithms

2.2.1 DNN Selection

The initial choice for classifying XRF spectra involved utilizing a neural network from the ResNet family, specifically ResNet50. The selection of ResNet was arbitrary but justified by its reputation as a robust CNN architecture. Notably, ResNet architecture won the ImageNet Large Scale Visual Recognition Challenge in 2015 [ILSVRC(2015)].

ResNet architecture is characterized by its ability to support very deep networks. This is attributed to the presence of residual connections, which allow each block of network for the learning of residual mappings $g(x) = f(x) - x$ rather than the usual mapping $f(x)$ [Zhang et al.(2022d)].

If the desired mapping is identity mapping $f(x) = x$, then block must only learn mapping $g(x) = 0$, which is easy to learn. As a result it is hard to degrade performance of this architecture with increasing depth - see Figure [4].

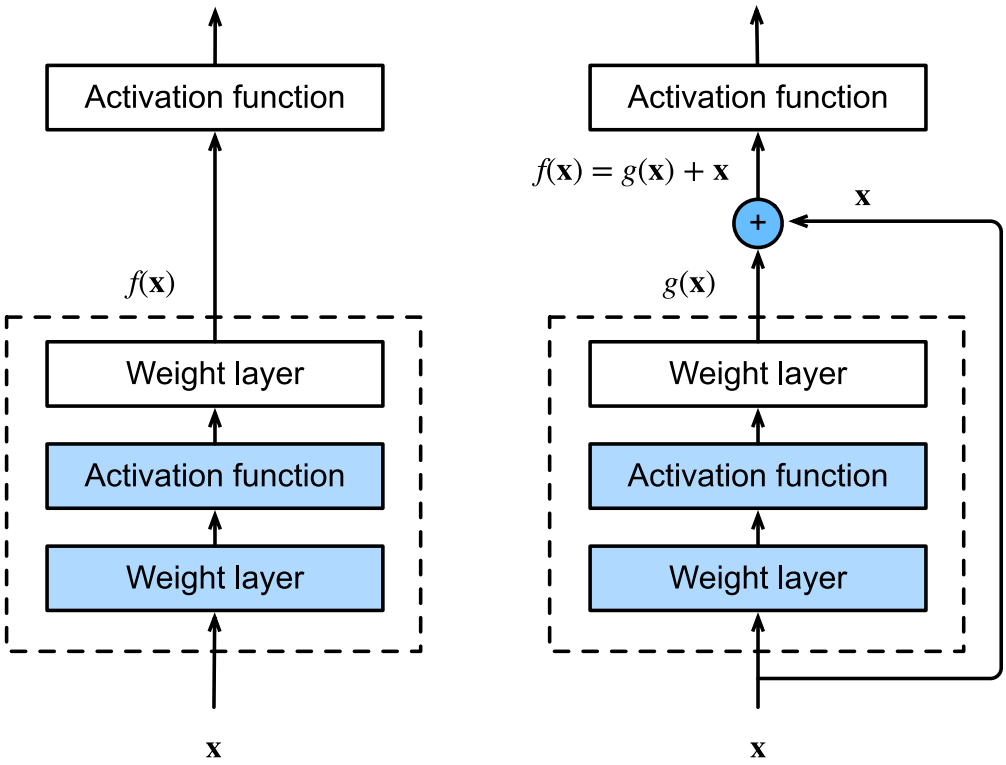


Figure 4: The residual block (right) needs to learn the residual mapping $g(x) = f(x) - x$, wheres regular block (left) must learn direct mapping $f(x)$. Source: [Zhang et al.(2022d)]

However, the original architecture of ResNet incorporates a *global average pooling* operation just before the final fully connected layer. This operation calculates the average value over the spatial dimensions of a single feature map. In the case of adapting ResNet to work with 1D spectra, the input for global average pooling has a shape of (batch_size, channels, features)

and the output has shape (batch_size, channels, 1). It means that due to averaging over features the spatial information is lost!

This resulted in the network not performing as expected, since peak positions in XRF spectra are crucial to identify elements. Furthermore, replacing global average pooling with a flattening operation was not feasible, as it would lead to
channels $\times$ features $\times$ fully_connected_size total connections with the fully connected layer. For example, with an input vector of shape $(\text{batch\_size}, 2048, 128)$ (which was observed during development), this would result in approximately 5×10^8 trainable parameters, while default implementation of ResNet50 have only $\sim 2.6 \times 10^7$ as a whole!

To address this problem, several possibilities were considered:

1. Modifying the architecture of ResNet to further reduce dimensionality further using convolution and pooling operations.

2. Reducing size of fully connected layer.

3. Opting for a completely different architecture.

While the two first options vere feasible, the decision was made to change used architecture completely. As a result, the author chose to use the ViT (Vision Transformer).

2.2.2 Multi-Head Attention

To understand ViT one ought to first understand how transformers work in general. Transformer architecture was originally meant to be replacement for RNNs (Recurrent Neural Networks) [Vaswani et al.(2017)]. Although transformers needs more training data (due to small inductive bias[1]) than recurrent networks to achieve similar results, they have significant advantage in terms of parallelization.

Unlike classic RNNs, which require the use of the hidden state calculated at time step $t-1$ to compute the hidden state at time step t, which makes them non-parallelizable, transformers are highly parallelized, thanks to *multi-head attention*, which makes heavy use of matrix multiplication.

Multi-Head Attention works based on *attention mechanism*, which is somewhat similar to a database query [Zhang et al.(2022a)]. To explain it let's define a key-value database consisting of $(\mathbf{k}, \mathbf{v})$ vector pairs which can be queried using $\mathbf{q}$ vector query:

$$D \stackrel{\text{def}}{=} \{(\mathbf{k_i}, \mathbf{v_i}) \mid i = 1, 2, \ldots, n\}.$$

[1] e.g. "In computer vision related tasks, the great success of convolutional neural networks (CNN) is often attributed to its inductive biases, such as locality and translation equivariance. [Mormille et al.(2023)]". In contrast, transformers exhibit less inductive biases, enabling them to explore a broader hypothesis space. Consequently, they may converge to local optima and generalize poorly on unseen data, when trained on insufficient data.

Then attention over D can be denoted as:

$$\text{Attention}(\mathbf{q}, D) = \sum_{i=1}^{n} \text{a}(\mathbf{q}, \mathbf{k_i})\mathbf{v_i},$$

where $\text{a}(\mathbf{q}, \mathbf{k_i}) \in \mathbb{R}$ are attention weights.

If exactly one of the weights $\text{a}(\mathbf{q}, \mathbf{k_i}) = 1$, while all others are 0, then attention works like normal database query and returns value of $\mathbf{v_i}$ for $\mathbf{k_i}$ that matches $\mathbf{q}$. In case that there are multiple non-zero weights then some linear combination of vectors is retrieved. For deep learning applications the following properties are desirable: $\sum_i \text{a}(\mathbf{q}, \mathbf{k_i}) = 1$ and $\text{a}(\mathbf{q}, \mathbf{k_i}) \geqslant 0$. To guarantee this behaviour, the Softmax function can be applied:

$$\alpha(\mathbf{q}, \mathbf{k_i}) = \text{Softmax}(\text{a}(\mathbf{q}, \mathbf{k_i})) = \frac{\exp(\text{a}(\mathbf{q}, \mathbf{k_i}))}{\sum_j \exp(\text{a}(\mathbf{q}, \mathbf{k_j}))}.$$

The last thing to be defined is the attention scoring function $\text{a}(\mathbf{q}, \mathbf{k_i})$. It is highly unlikely to find any exact match between $\mathbf{q}$ and $\mathbf{k_i}$, so $\text{a}(\mathbf{q}, \mathbf{k_i})$ must be defined as some similarity function between vectors in feature space.

Let's take a look at Gaussian similarity kernel, which is a non-linear function of euclidean distance:

$$\text{K}(\mathbf{x}, \mathbf{x}') = \exp(-\frac{\|\mathbf{x} - \mathbf{x}'\|}{2\sigma}).$$

It has nice property of being bound between zero and one, unlike euclidean distance which could be anything and lead to numerical instabilities. However, it has disadvantage of being computationally costly.

Now lets consider the kernel with substituted $\mathbf{q}$ and $\mathbf{k_i}$, exponentiation skipped, and in expanded form:

$$\text{k}(\mathbf{q}, \mathbf{k_i}) = -\frac{1}{2}\|\mathbf{q} - \mathbf{k_i}\|^2 = \mathbf{q}^\intercal\mathbf{k_i} - \frac{1}{2}\|\mathbf{k_i}\|^2 - \frac{1}{2}\|\mathbf{q}\|^2.$$

The last term is constant across all values of $\mathbf{k_i}$ and due to normalization its presence don't affect the result. Therefore, it can be safely omitted. Similarly, the second term may be disregarded because batch/layer normalization effectively bounds $\|\mathbf{k_i}\|$, ensuring a negligibly small impact on the final result [Zhang et al.(2022b)].

If we assume that $q \in \mathbb{R}^d$ and $k_i \in \mathbb{R}^d$, and that their elements are drawn from distribution $\mathcal{N}(\mu = 0, \sigma^2 = 1)$, then their dot product will have mean zero and variance d. After normalization by factor $\frac{1}{\sqrt{d}}$, the first commonly used attention function - *scaled dot product attention* [Vaswani et al.(2017)] can be written down as:

$$\text{a}(\mathbf{q}, \mathbf{k_i}) = \frac{\mathbf{q}^\intercal\mathbf{k_i}}{\sqrt{d}}.$$

Final attention weights can be calculated by applying Softmax:

$$\alpha(\mathbf{q}, \mathbf{k_i}) = \text{Softmax}(\text{a}(\mathbf{q}, \mathbf{k_i})) = \frac{\exp(\frac{q^\intercal k_i}{\sqrt{d}})}{\sum_j \exp(\frac{q^\intercal k_j}{\sqrt{d}})}.$$

To take advantage of parallelization, vector multiplication can be replaced with matrix multiplication. When computing attention for n queries and m key-value pairs, where both queries and keys have a length of d_k (although it must not necessarily be the case), and values have a length of d_v, the following matrices must be defined: $Q \in \mathbb{R}^{n \times d_k}$, $K \in \mathbb{R}^{m \times d_k}$, and $V \in \mathbb{R}^{m \times d_v}$. These matrices will form a formula analogous to that of vectors:

$$\text{Attention}(Q, K, V) = \text{Softmax}(\frac{QK^\intercal}{\sqrt{d}})V.$$

In practice, it was found that it is advantageous to combine multiple attention pooling outputs computed in parallel. In theory it may lead to capturing different behaviours of attention mechanism, e.g. capturing short-range and long-range dependencies within a sequence [Zhang et al.(2022c)].

A single output of attention pooling was originally referred to by the authors as a *head*. multi-head attention can be calculated in following way:

$$\text{MultiHeadAttention}(Q, K, V) = \text{Concat}(head_1, head_2, \ldots, head_n)W^O,$$

where $head_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \in \mathbb{R}^{p_v}$, where $W_i^Q \in \mathbb{R}^{d_k \times p_k}$, $W_i^K \in \mathbb{R}^{d_k \times p_k}$, $W_i^V \in \mathbb{R}^{d_v \times p_v}$ and $W_i^O \in \mathbb{R}^{(hp_v) \times p_o}$ are learnable parameters. To better manage computational cost, the input sizes for each head are parameterized in following manner: $p_k = p_v = p_o/h$, where h is number of heads and p_o is size of output of last fully connected layer. Thanks to that, computational cost don't increase with higher number of heads.

2.2.3 ViT Architecture

Multi-Head Attention is the most important concept used in transformer architecture, it is no different when it comes to Vision Transformer [Dosovitskiy et al.(2020)]. DNN architecture used to classify XRF spectra is slightly modified version of ViT, which was adapted to work with 1D input.

Most parts of the architecture remain unchanged. Transformer encoder is implemented (Listing [1]) in exactly the same way as in original paper - Figure [5]. In contrast to the original transformer encoder architecture [Vaswani et al.(2017)], layer normalization in ViT is applied before all MHA and MLP (Multi Layer Perceptron) in order to enhance training effectiveness. Moreover, the commonly used non-linear activation function ReLU (Rectified Linear Unit) is replaced with its smoother counterpart, GELU (Gaussian-Error Linear Unit).

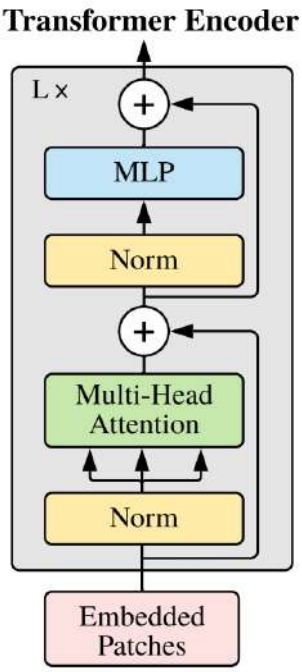


Figure 5: Transformer Encoder architecture. Source: [Dosovitskiy et al.(2020)]

```python
class TransformerBlock(nn.Module):
    def __init__(self, embed_dim, num_heads, hidden_dim, dropout_rate):
        super(TransformerBlock, self).__init__()
        self.norm1 = nn.LayerNorm(embed_dim)
        self.attention = nn.MultiheadAttention(embed_dim, num_heads)
        self.norm2 = nn.LayerNorm(embed_dim)
        self.feedforward = nn.Sequential(
            nn.Linear(embed_dim, hidden_dim),
            nn.GELU(),
            nn.Dropout(dropout_rate),
            nn.Linear(hidden_dim, embed_dim),
            nn.Dropout(dropout_rate)
        )

    def forward(self, x):
        attn_output, _ = self.attention(*([self.norm1(x)] * 3))
        x = x + attn_output
        x = x + self.feedforward(self.norm2(x))
        return x
```

Listing 1: Transformer Encoder block implementation. The implementation details were based on [Zhang et al.(2022e)]

Several modifications were introduced in the remaining part of the original implementation - Figure [7], Figure [6].

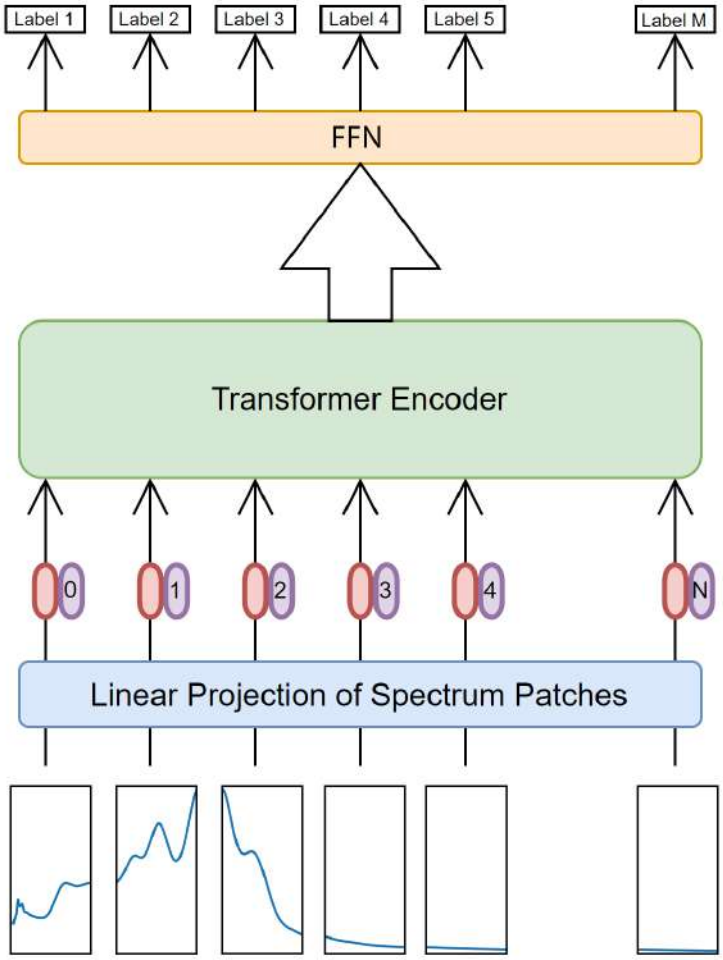


Figure 6: Modified ViT architecture.

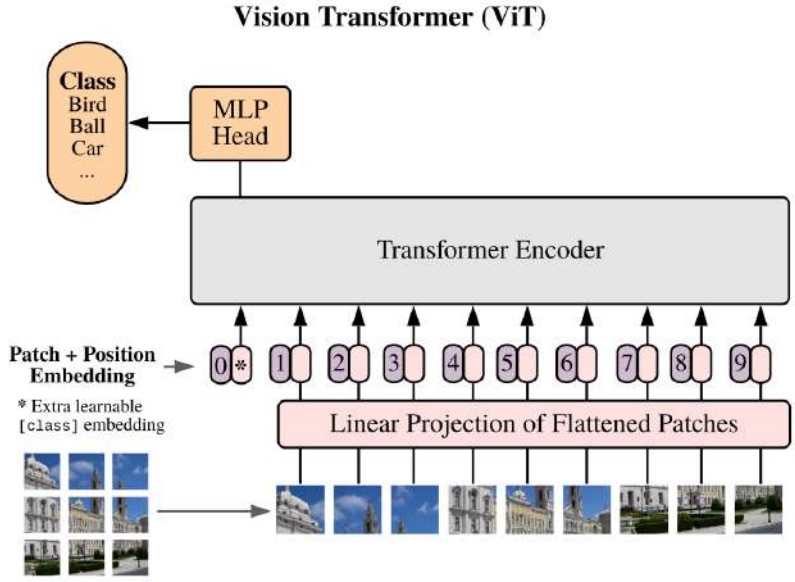


Figure 7: Original ViT architecture. Source: [Dosovitskiy et al.(2020)]

The `[class]` token has been removed. `[class]` token was an idea introduced in model BERT (Bidirectional Encoder Representations from Transformers) [Devlin et al.(2018)] and it was used as an input to classifying MLP. However, due to small embedding and input size it was possible to remove it and replace with fully connected layer. Additionally, the authors discovered that the `[class]` token was not inherently superior to other methods; other methods required just a distinct learning rate [Dosovitskiy et al.(2020)].

Another adjustment was to replace patch embedding using `nn.Conv2d` (2D convolution operation) with `nn.Conv1d`, because the input is not a 2D image but a 1D spectrum.

Last but not least, the softmax function was replaced with the sigmoid function to enable the classification of multiple labels simultaneously. Implementation details are shown in Listing [2].

```
class VisionTransformer(nn.Module):
    def __init__(self, input_size, patch_size, embed_dim, num_heads, num_classes,
    ↪  num_layers, hidden_dim, dropout_rate):
        super(VisionTransformer, self).__init__()
        num_patches = input_size // patch_size
        self.patch_embed = nn.Conv1d(1, embed_dim, kernel_size=patch_size,
        ↪  stride=patch_size, bias=False)
        self.pos_embed = nn.Parameter(torch.zeros(1, num_patches, embed_dim))
        self.dropout = nn.Dropout(dropout_rate)
        self.transformer_blocks = nn.ModuleList([
            TransformerBlock(embed_dim, num_heads, hidden_dim, dropout_rate) for _ in
            ↪  range(num_layers)
        ])
        self.norm = nn.LayerNorm(embed_dim)
        self.fc = nn.Linear(embed_dim * num_patches, num_classes)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        x = self.patch_embed(x)
        x = x.flatten(2).transpose(1, 2)
        x = self.dropout(x + self.pos_embed)
        for i, block in enumerate(self.transformer_blocks):
            x = block(x)
        x = self.norm(x)
        x = x.reshape(x.size(0), -1)
        x = self.fc(x)
        x = self.sigmoid(x)
        return x
```

Listing 2: ViT implementation. The implementation details were based on [Zhang et al.(2022e)]

2.2.4 Self-Organizing Map

Being able to classify elements in the spectrum is only part of the success. Data gathered using FF-XRF has a mean of around 20,000 photon counts per one spatial coordinate - see Figure [8]. When distributed over ∼4000 possible energy levels, it provides a rather noisy spectrum that may not yield correct classification. It would also come with long total inference time.

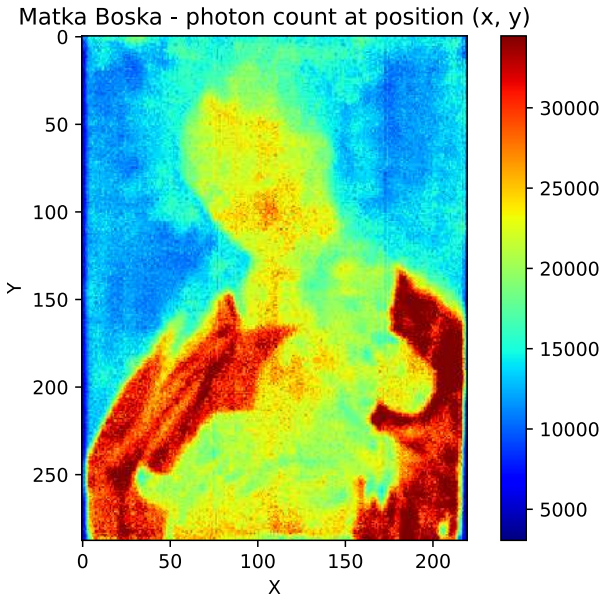


Figure 8: Photon count in data gathered from the painting "Mother of God with the Child Eating an Apple"

To address this issues, one of the following strategies may be involved:

1. Calculating the average of spectra in some small region.

2. Calculating the average of clustered spectra.

The first option is easier because clustering high-dimensional data is a daunting task due to the *curse of dimensionality*.

Available options are highly limited, but they do exist. One algorithm that can tackle high-dimensional clustering is the SOM (Self-Organizing Map) algorithm, which is an unsupervised machine learning algorithm based on a neural network. It is mainly used for visualization of feature-rich data because it reduces its dimensionality to (usually) 2D map, but has been proven to work well as clustering algorithm for XRF spectra [Kogou et al.(2020)].

The algorithm works in the following way [Ahn and Syn(2005)]:

1. n weights $\mathbf{w_i}$ of the same length as feature vectors are initialized.

2. Weights are distributed over a 2D map using a specified topology, such as a square, hexagonal, or random grid.

3. A vector $\mathbf{d}$ is chosen from the training data set.

4. $\mathbf{d}$ is compared to each node $\mathbf{w_i}$ on the map using a distance function, such as the l_2 norm or l_1 norm. The node with the closest distance is designated as the BMU (Best Matching Unit).

5. The neighborhood of the BMU is calculated.

6. The BMU and all nodes in the neighborhood are updated, making them more similar to $\mathbf{d}$.

7. The algorithm is repeated from step 3 for a specified number of iterations.

The weights are updated using following equation [Wikipedia(2023)]:

$$\mathbf{w}_i^{s+1} = \mathbf{w}_i^s + \theta(u, v, s) \cdot \alpha(s) \cdot (\mathbf{d} - \mathbf{w}_i^s).$$

Here, s represents the number of iterations, u the index of the BMU and v the index of the node on the map (may be the same as u). The function $\theta(u, v, s)$ represents the neighborhood function and states that the BMU is updated the most and farther neighbors are updated less. An example of a neighborhood function could be a Gaussian kernel. The function $\alpha(s)$ represents the learning rate schedule. The visualization of weight update step is shown in Figure [9].

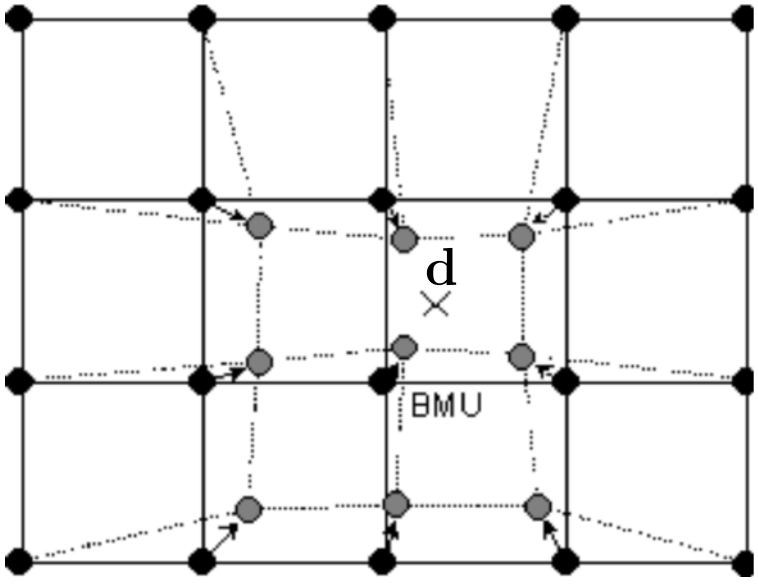


Figure 9: Updating the best matching unit (BMU) and its neighbours towards the input sample $\mathbf{d}$. Source: [Vallés-Pérez(2015)]

Although algorithm is fairly simple, the most popular python packages don't provide its implementation, and the most popular implementation - package `minisom` had too large

RAM memory requirements, which lead to crashing Colab environment with 50GB of memory. Because of that, its basic, slightly rewritten Python implementation was used. Its source can be found in [Ralhan(2018)].

2.2.5 UMAP and HDBSCAN Clustering Pipeline

Unfortunately, clustering the data with SOM showed some weird artifacts in all analyzed samples. It was the reason to find alternative clusterization method to ensure that the artifacts were not caused by the algorithm. After some research it was found that UMAP (Uniform Manifold Approximation and Projection) → HDBSCAN (Hierarchical Density-Based Spatial Clustering of Applications with Noise) clustering pipeline should also work with large datasets of high dimensional data.

UMAP is an algorithm for dimension reduction based on manifold learning techniques. It comes with uniform density assumption, so density is not preserved correctly. However, according to the documentation, UMAP output should be easily clustered by HDBSCAN algorithm [McInnes et al.(2018b)], which is density based algorithm.

The HDBSCAN algorithm is a follow-up to the DBSCAN algorithm. The key distinction between HDBSCAN and DBSCAN lies in HDBSCAN's ability to determine clusters of different densities.

Due to the complexity of these algorithms and the author's lack of familiarity with topology, further explanation of the algorithms will not be provided here. Interested readers can refer to their respective documentation pages [McInnes et al.(2018a)], [McInnes et al.(2016)].

2.3 Artificial Data Generation

To train a DNN in a supervised fashion, one must have sufficient sample of labeled data. Acquisition of such data is by and large beyond the scope of this thesis. The only feasible way to acquire large enough amounts of data is to generate it. Such efforts have already been made in [Jones et al.(2022)], and a similar approach is used here.

To generate training data, a special function has been created - Listing [3].

```
def generate_training_data(element_lines
, mu_max_err=0.002
, sigma_max_err=0.08
, mu_max_err_global=0.05
, samples=10
, elements_per_sample=3
, ...):
    Parameters:
        - element_lines (dict): Dictionary of element lines.
        - mu_max_err (float): Maximum error for mu of single peak.
        - sigma_max_err (float): Maximum error for sigma of single peak.
        - mu_max_err_global (float): Maximum error for mu of all peaks.
        - samples (int): Number of samples to generate.
        - elements_per_sample (int): Number of different elements per sample.

    Returns:
        - X (list): List of input data samples.
        - y (list): List of target data samples.
```

Listing 3: Function for training data generation

The function creates artificial spectra by adding multiple gaussians in form:

$$\mathcal{N}(\mu + \mathcal{U}(-e_{\mu}^{\text{global}}, e_{\mu}^{\text{global}}) + \mathcal{U}(-e_{\mu}^{\text{local}}, e_{\mu}^{\text{local}}), \sigma + \mathcal{U}(-e_{\sigma}^{\text{local}}, e_{\sigma}^{\text{local}})),$$

where:

μ – energy of peak [KeV],
σ – standard deviation of the Gaussian of the peak of energy μ [KeV],
e_{μ}^{global} – maximum absolute energy error of the whole spectrum [KeV],
e_{μ}^{local} – maximum absolute energy error of a single peak [KeV],
$e_{\sigma}^{\text{local}}$ – maximum absolute error of the standard deviation of a single peak [KeV],

Parameters with an `err` in their name shown in Listing [3] are used to define the maximum difference from the theoretical values of μ and σ. The deviations are drawn from a uniform distribution $\mathcal{U}(-err, err)$. This augmentation should account for possible small discrepancies from theory.

There is a local (affecting single peak) parameter defined for μ, which should be small because the relative peak positions should not change much. However, the author observed that the peak position in the provided sample of a copper PCB, after correction, differed by about 0.07 keV from the theoretical value, so it may be wise to augment data for possible larger, global (affecting all peaks) translation error.

σ of a peak is calculated similarly as in [Jones et al.(2022)], using following formula:

$$\sigma = \sqrt{\frac{NOISE}{2.3548} + 0.00358 \times FANO \times E},$$

where:

E – energy of a peak in [KeV],
$FANO$ – fano factor [-],
$NOISE$ – electronic contribution to the peak width [KeV],

Fano factor was calculated using Gaussian fitted on accumulated spectrum of copper plate - Figure [10]. Although peak in Figure [10] seem to be generated correctly, the process is wrong at few points because calculated $FANO$ is an about order of magnitude larger than literature says, $NOISE$ is unknown, and constant 0.00358 is correct only for Silicon Drift Detectors. Further in the work it will be assumed that σ calculation works correctly.

Additionally, the arbitrary exponential is added to the sum of gaussians to account for background radiation - it is however not ideal solution.

Parameter `element_lines` mentioned in Listing [3] needs some further explanation - it is an arbitrary data structure that keeps information about spectral lines for different elements in format shown in Listing [4].

```
element_lines = {
    0: (("k_alpha", 3.314, 1), )
    , 1: (("k_alpha", 3.692, 1), )
    , 2: (("k_alpha", 4.512, 1), )
    , 3: (("k_alpha", 4.953, 1), ("k_alpha_esc", 1.995, 0.2), )
    , ...}
```

Listing 4: Keys are indexes of different elements. Under each index the element spectral lines are listed. First value in each tuple is name of the line, second is its energy, and the last is relative intensity in element spectrum

The line intensities defined in Listing [4] are arbitrarily. Probably, the ideal action would be to not generate spectra but to use measurements of reference materials, which could then be combined in similar fashion.

Example spectra with signals from copper and lead present can be seen in Figure [11], while visualization of data generation pipeline can be found in Figure [12].

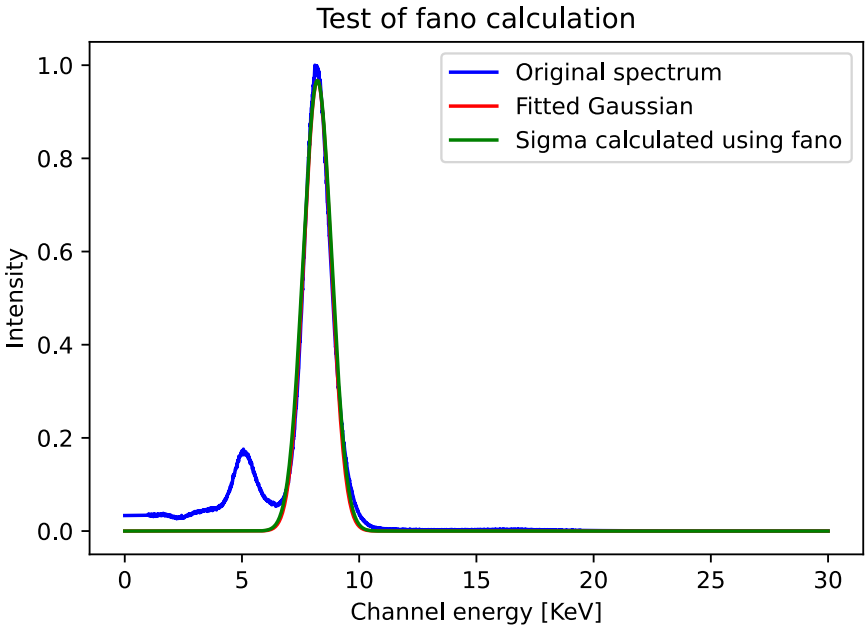


Figure 10: Sigma calculated using fano factor

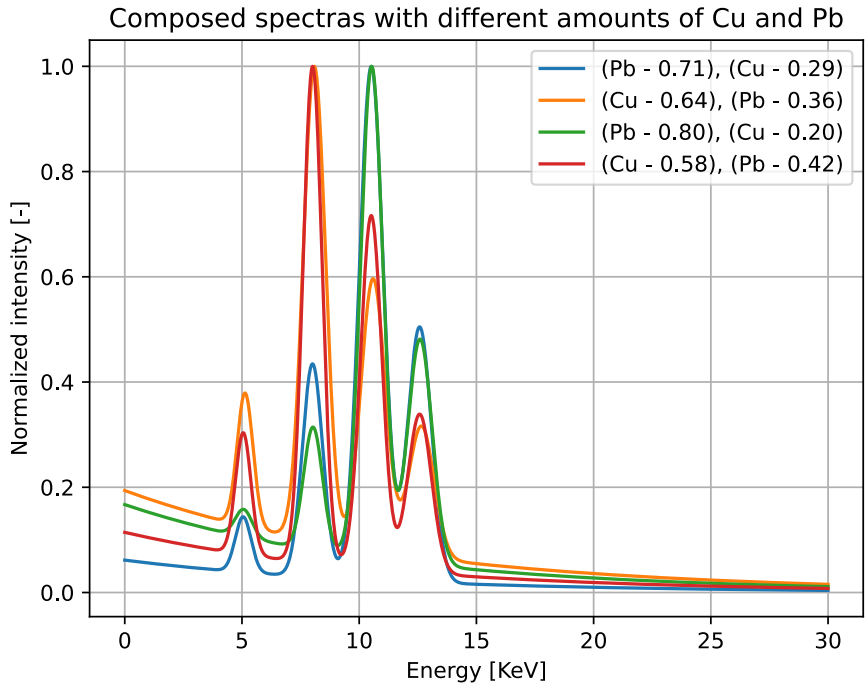


Figure 11: Example spectra with signals from Cu and Pb

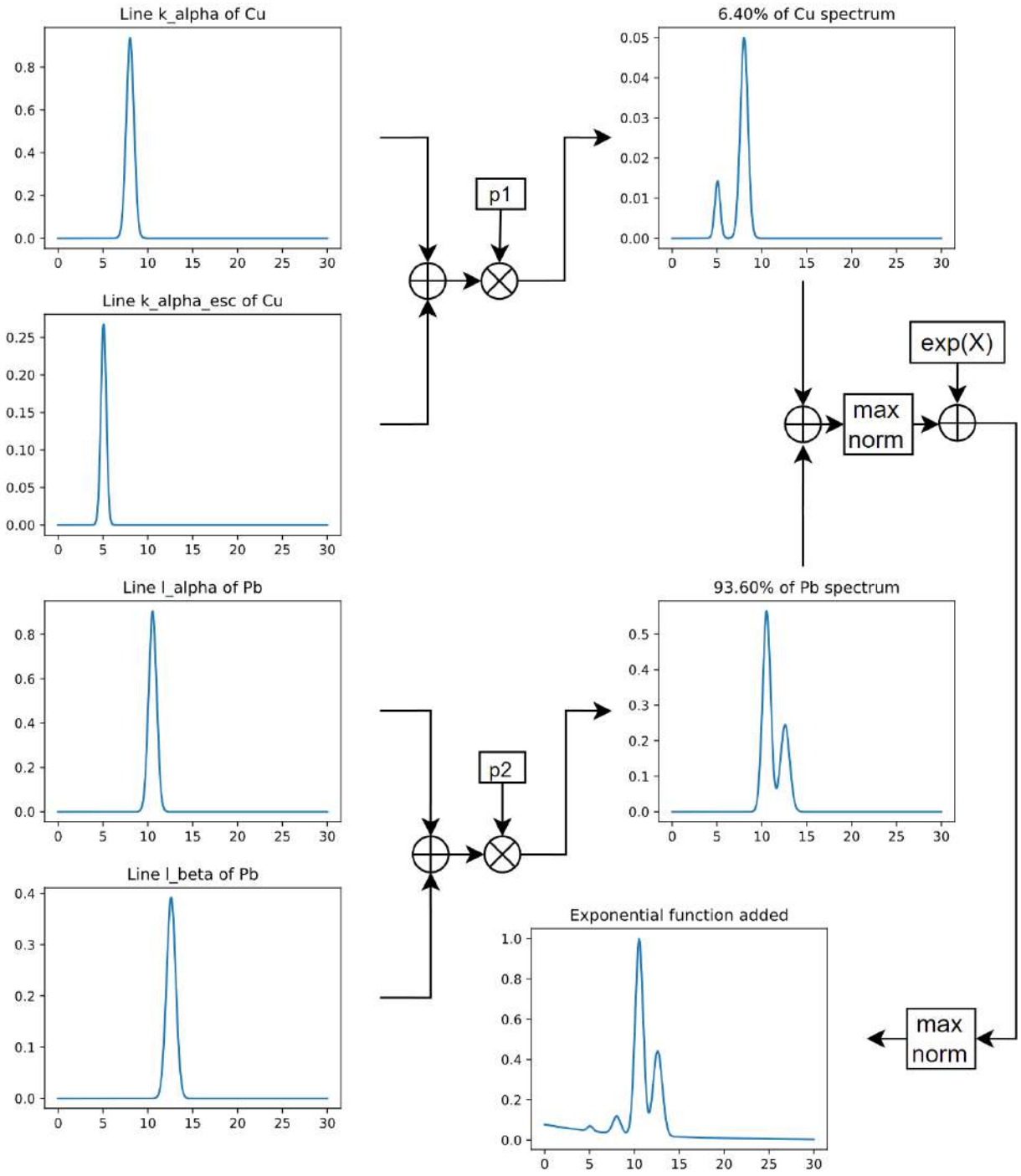


Figure 12: Steps in generation of example training spectrum

2.4 Real Data Preprocessing

Before real data can be passed to classification algorithm it ought to be preprocessed. Peak positions in acquired spectrum differ from measurement to measurement - this problem has been addressed in [Góral(2023)] master thesis. Basing on master thesis achievements a function for data preprocessing have been created - Listing [5].

```
def get_processed_spectrum(e_xy, peaks, energies, min_energy=0, max_energy=30
                           , energy_margin=1, target_length=4096,
                           ↪  points_to_extrapolate_count=100, ...):

    Parameters:
        - e_xy: numpy array, shape (H, W, C), representing the raw data.
        - peaks: array-like, containing peak positions for energy calibration.
        - energies: array-like, containing corresponding energies for calibration
        ↪  peaks.
        - min_energy: float, optional, default: 0, the minimum energy boundry.
        - max_energy: float, optional, default: 30, the maximum energy boundry.
        - energy_margin: float, optional, default: 1, margin of mask at the edges of
        ↪  the spectrum.
        - target_length: int, optional, default: 4096, the desired length of the output
        ↪  spectrum.
        - points_to_extrapolate_count: int, optional, default: 100, number of points
        ↪  used for extrapolation at boundries of spectrum.

    Returns:
        - channel_energies: numpy array, shape (target_length,), the interpolated
        ↪  channel energies.
        - spectrum_acc: numpy array, shape (target_length,), accumulated and processed
        ↪  spectrum.

```

Listing 5

Preprocessing involves following steps:

1. The channels are mapped to energy (work done in [Góral(2023)]) and cropped if they fall outside the defined range [`min_energy`, `max_energy`] [KeV].

2. The left and right boundary, each of size `energy_margin` [KeV], is masked with zeros because boundaries tend to have high amount of noise.

3. The spectrum is padded with zeros until it fills the range [`min_energy`, `max_energy`].

4. The spectrum is extrapolated on the left and right sides using the exponential function `a * np.exp(b * x)`, which is fitted using `points_to_extrapolate_count` non-zero

points at the ends of the spectrum.

5. Finally, the spectrum is interpolated to the defined length `target_length`.

The visualization of these steps is shown in Figure [13].

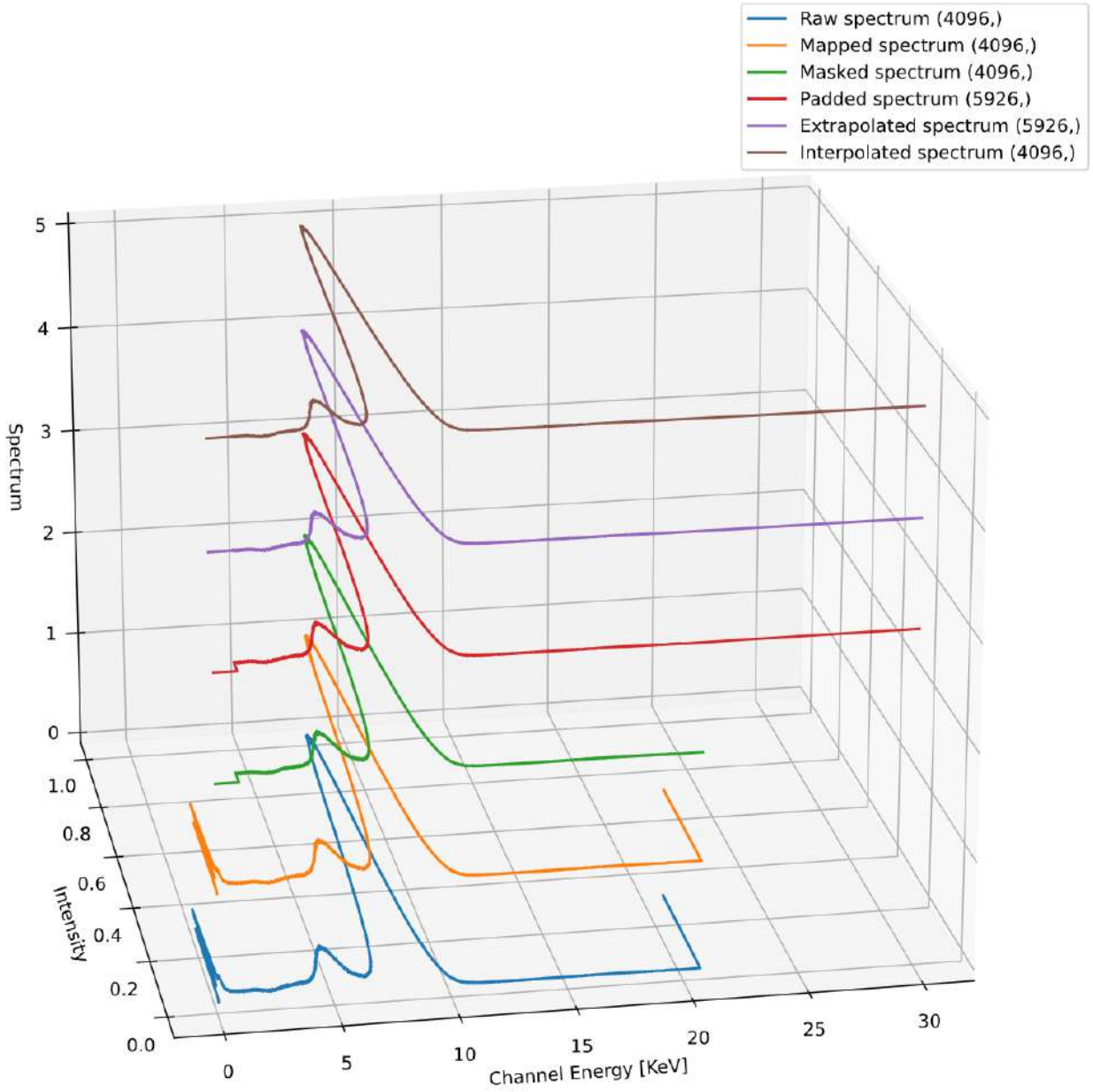


Figure 13: Steps of spectrum preprocessing

The processed spectrum can then be passed to a classifier that was trained using the same energy mapping. Alternatively, the extrapolation step can be omitted; in this case, the classifier should also be trained on masked spectra or a different energy range, for example, [1, 29] [KeV].

3 Results

3.1 Spectra Clusterization

Clusterization of spectra will be presented using data gathered on "Portrait of John III Sobieski in Karacena Scale Armour" (see Figure [14]).

Figure 14: An examined fragment of "Portrait of John III Sobieski in Karacena Scale Armour". Source: [Wikimedia(2023)]

3.1.1 Clusterization With SOM

The first clustering algorithm, SOM, was invoked as presented in Listing [6].

```
1 som = SelfOrganizingMap(input_size=e_xy.shape[-1], map_size=(3, 3), learning_rate=0.1,
  ↪  sigma_neigh=0.5, sigma_decay=0.5)
2 som.train(e_xy_flat_normalized, epochs)
3 bmus = [[som.find_bmu(e_xy_flat_normalized[i * e_xy.shape[1] + j]) for j in
  ↪  range(e_xy.shape[1])] for i in range(e_xy.shape[0])]
4 labels = [[bmu[0] * map_size[1] + bmu[1] for bmu in bmus_row] for bmus_row in bmus]
```

Listing 6: Invocation of SOM algorithm

The arguments used to initialize `SelfOrganizingMap` are:

`input_size`: Configured to the number of channels (4094 in this instance).

`map_size`: Set to (3, 3), indicating a total of $3 \times 3 = 9$ clusters.

`learning_rate`: The learning rate during the first epoch.

`sigma_neigh`: The influence on the neighbors of the BMU. Lower values result in larger influence.

`sigma_decay`: The rate of decay for the learning rate. Lower values result in slower decay.

The performance of the algorithm scales linearly with the number of clusters. With a total of 9 clusters, the clusterization of data with shape of (524, 348, 4094) took approximately 6 minutes. Therefore, it may not be feasible method if many more clusters were needed, at least while using basic python implementation and performing clusterization globally.

The result of applying clusterization on raw spectrum can be seen in Figure [15].

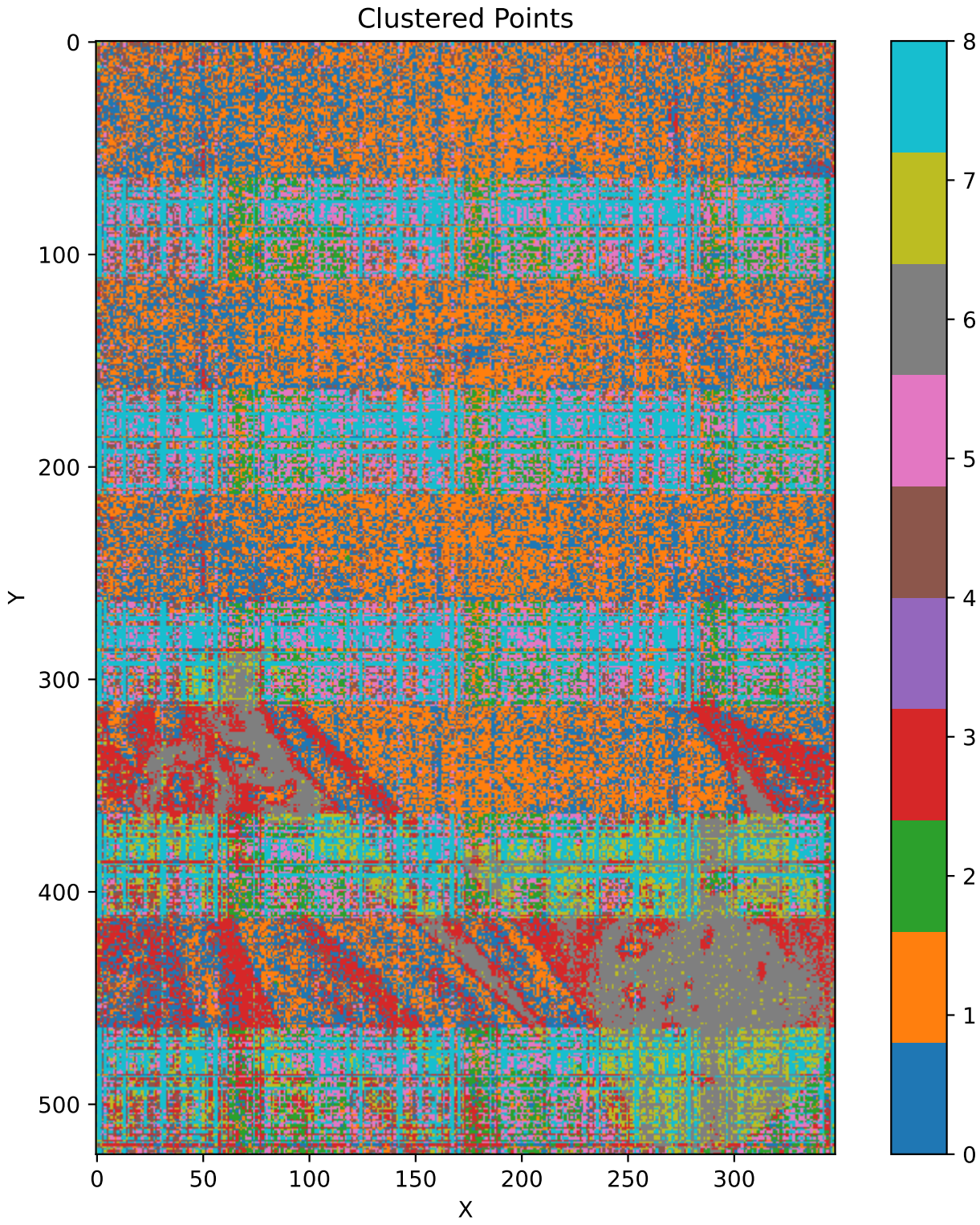


Figure 15: Clusterization of unprocessed spectra of "Portrait of John III Sobieski in Karacena Scale Armour" using SOM

As the reader may notice, certain clusters in Figure [15] are artificial. It was sensible to investigate the possibility of the artifacts arising from a poor implementation of the SOM algorithm. This involved employing a UMAP to HDBSCAN pipeline to validate the clustering results.

3.1.2 Clusterization With UMAP and HDBSCAN

Clusterization with UMAP and HDBSCAN was invoked as presented in Listing [7].

```
clusterable_embedding = umap.UMAP(
    n_neighbors=30,
    min_dist=0.0,
    n_components=1
).fit_transform(e_xy_flat_normalized)

clusterable_embedding = clusterable_embedding.reshape(e_xy_flat.shape[0], 1)
labels = hdbscan.HDBSCAN(
    min_samples=10,
    min_cluster_size=2500,
    metric='l2'
).fit_predict(clusterable_embedding) + 1
```

Listing 7: Invocation of UMAP to HDBSCAN pipeline

The meaning of arguments passed to UMAP is as follows:

`n_neighbors`: The number of neighbors used for manifold approximation. As `n_neighbors` places more focus is placed on the global structure of the data.

`min_dist`: The minimum distance between points in the dimensional embedding. For clusterization, values around 0 are preferred, because it is desirable to transform data into "tightly packed" clumps.

`n_components`: The number of dimensions in the low-dimensional representation. Experimentation showed that reducing dimensions to one gave the best results.

Arguments for HDBSCAN are:

`min_samples`: The number of samples in a neighborhood for a point to be considered a core point.

`min_cluster_size`: The minimum number of points required to form a cluster. It is probably the most important parameter; smaller values will lead to more clusters.

`metric`: The distance metric used for clustering. Euclidean distance works well, although other popular metrics give good results too.

Dimensionality reduction using UMAP took approximately 7 minutes. The computational cost scales with the parameter `n_neighbors`. Further clusterization with HDBSCAN is much faster (due to the really low dimensionality of input) and allows for fast experiments with different parameters, e.g. to achieve the desired granularity of clusters.

It appears that although clusterization worked, it did not prevented artifacts from appearing - see Figure [16].

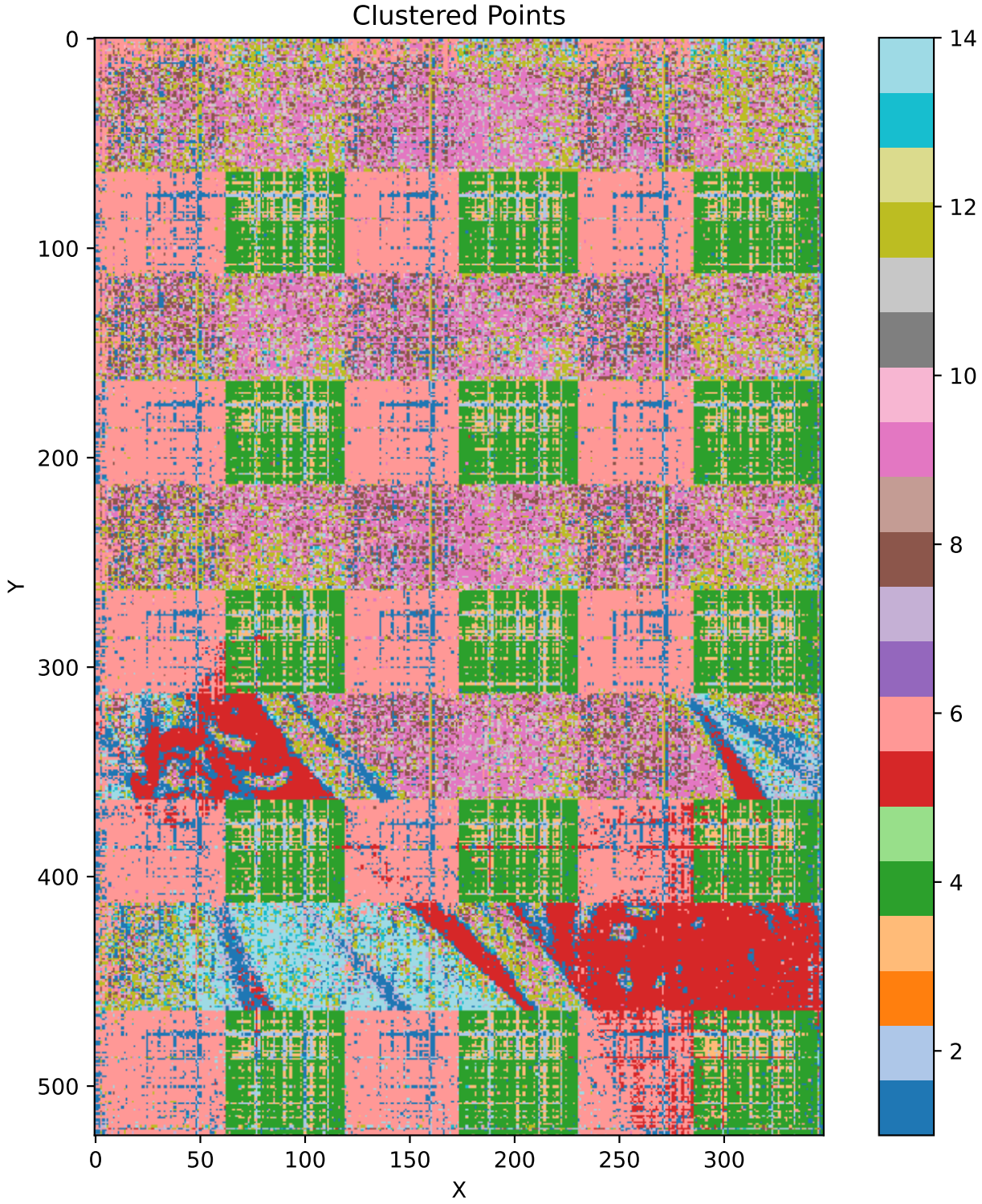


Figure 16: Clusterization of unprocessed spectra of "Portrait of John III Sobieski in Karacena Scale Armour" using UMAP and HDBSCAN

3.1.3 Source of Clusterization Artifacts

The source of clusterization artifacts remains unknown as of today. However, due to the nature of the artifacts, they may be attributed to a poorly implemented pinhole camera. Two possible implementations of a pinhole camera are demonstrated in Figure [17].

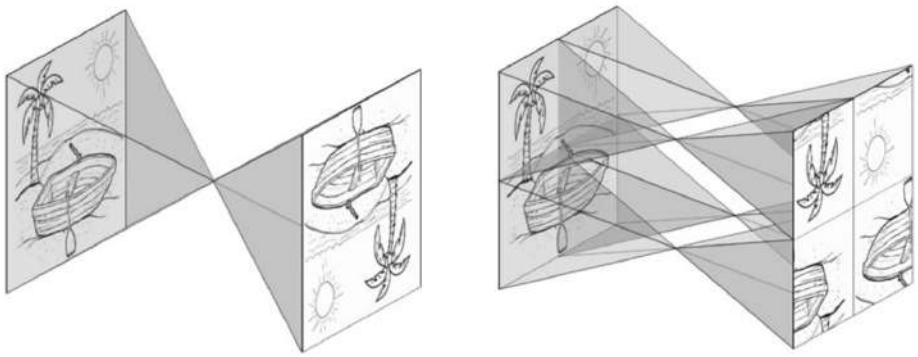


Figure 17: 1-hole pinhole camera (left) and 4-hole pinhole camera (right).
Source: [Łach(2022)]

The details of the pinhole camera implementation used during measurements are unknown, but the artifacts strongly suggest that a 4-hole pinhole camera was used. It seems that differences between pinhole cameras lead to different artifacts showing up in spectra, which lead to artificial clusters being found. Fortunately, the clusterization problem turned out to be easily solvable.

3.1.4 Clusterization Results

By simply removing one hundred points from both ends of the spectrum, the results of clusterization were tremendously improved, as visible in Figure [18] and Figure [19].

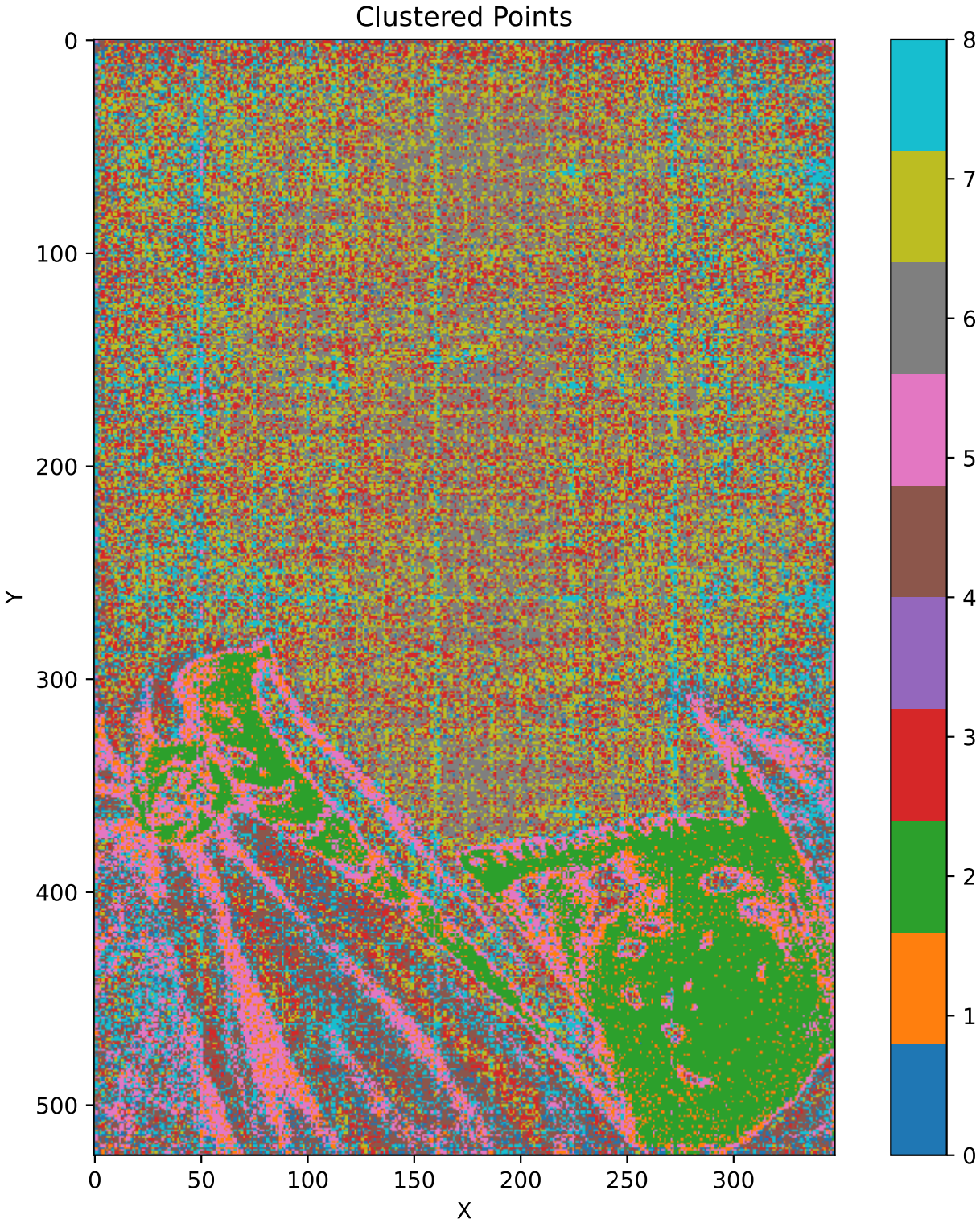


Figure 18: Clusterization of the cropped spectra of "Portrait of John III Sobieski in Karacena Scale Armour" using SOM

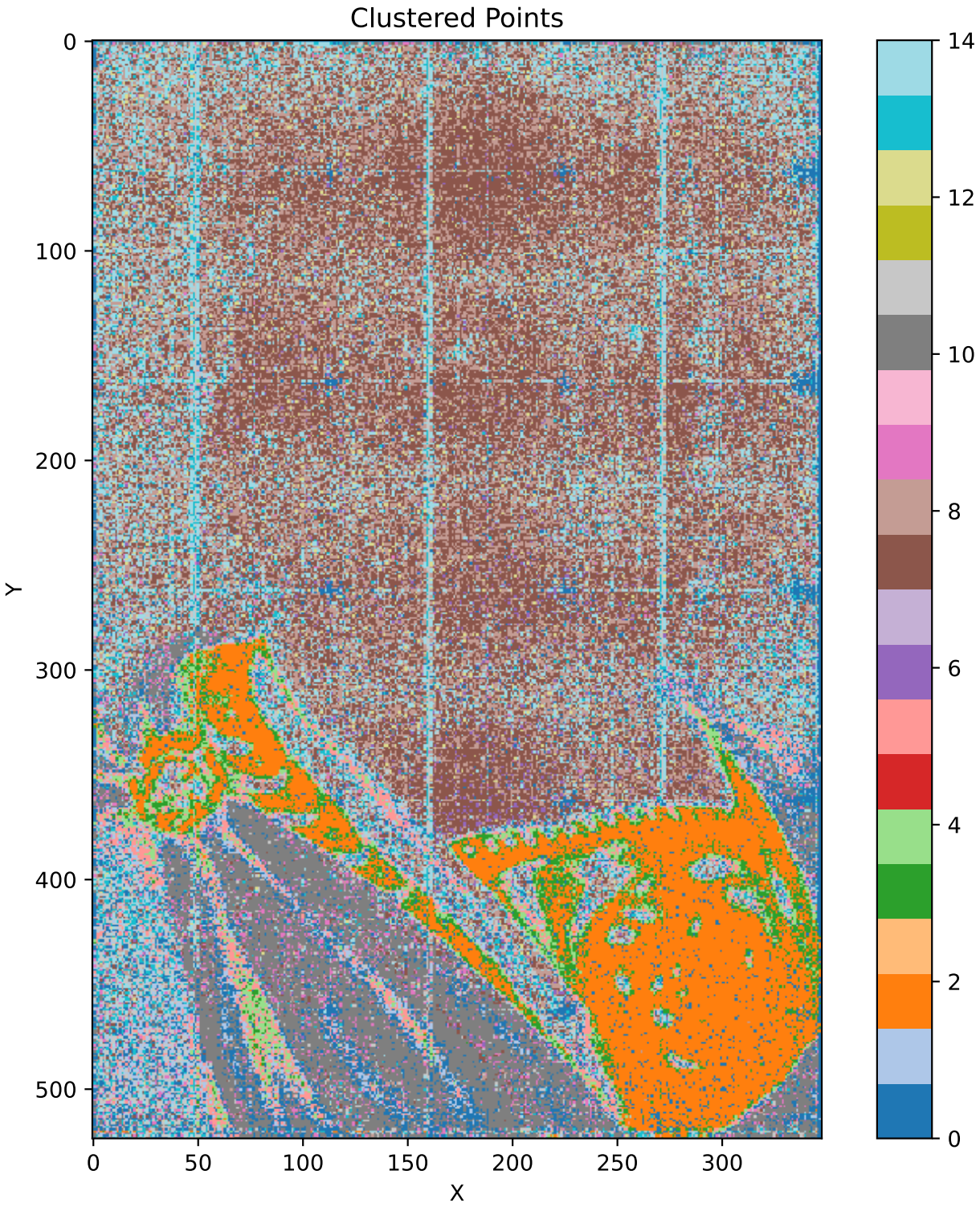


Figure 19: Clusterization of the cropped spectra of "Portrait of John III Sobieski in Karacena Scale Armour" using UMAP and HDBSCAN

To better understand the clusters they can be visualized on separate pictures, as shown in Figure [20] and Figure [21].

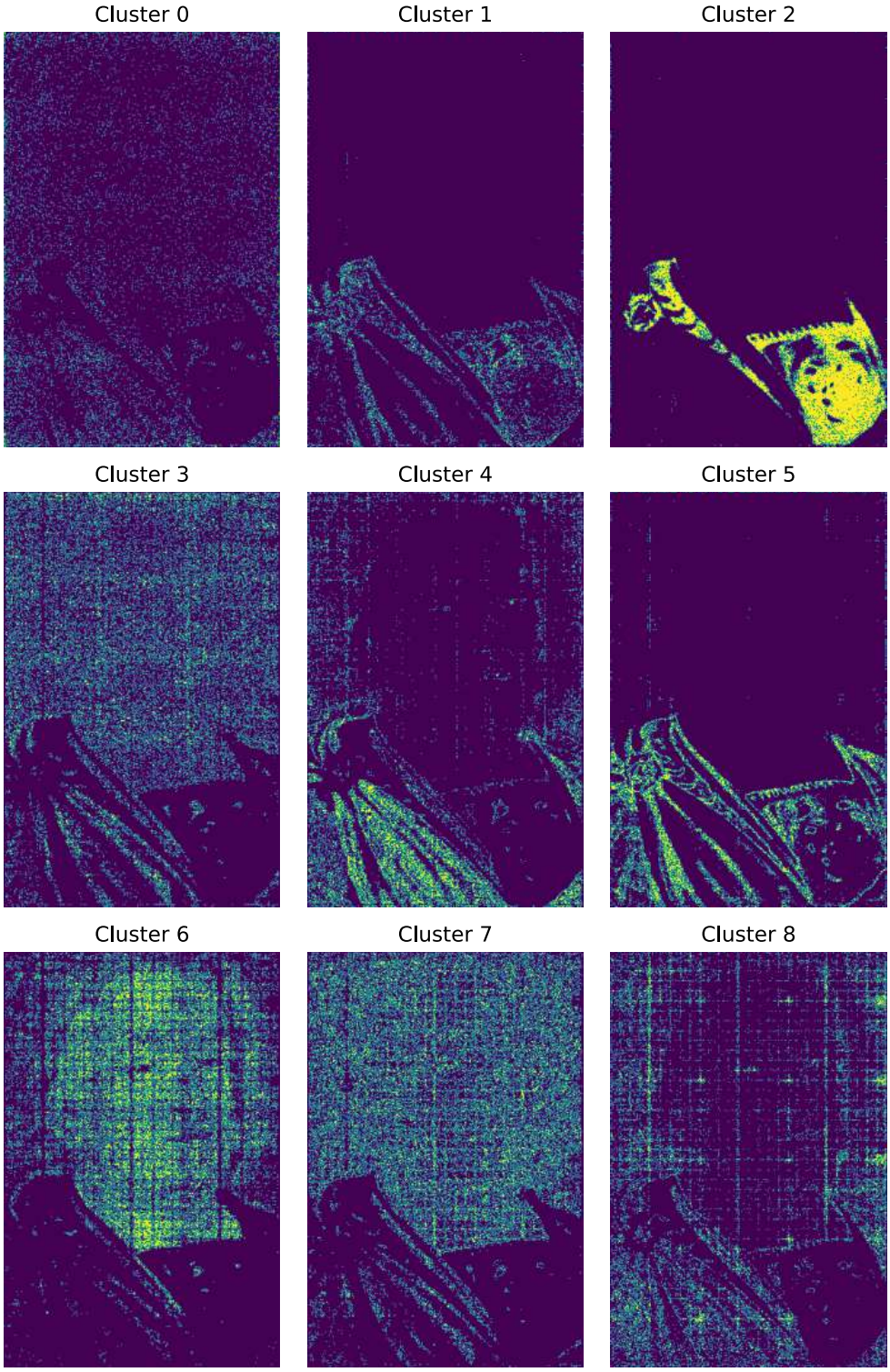


Figure 20: Separate clusters from SOM clusterization

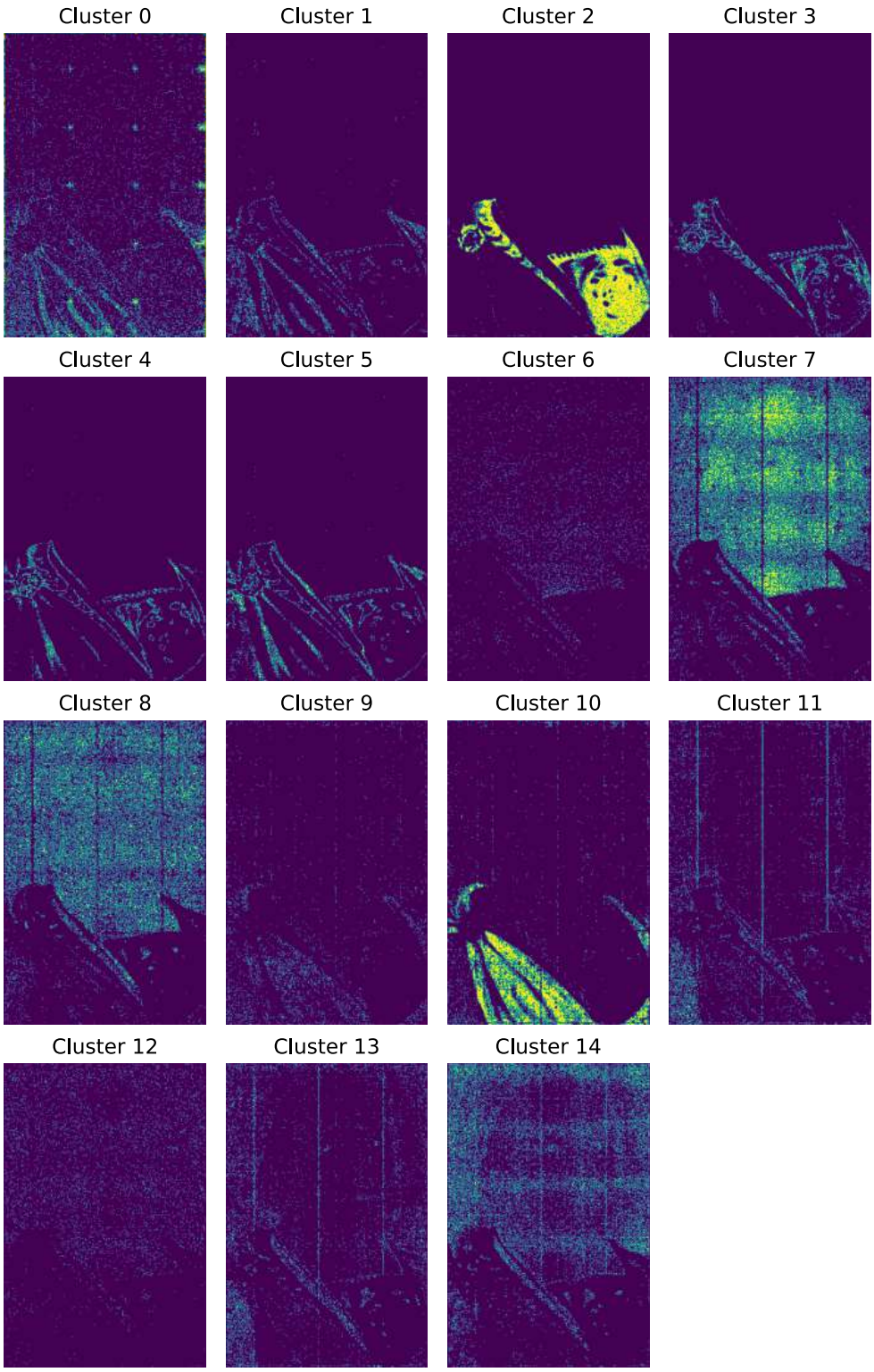


Figure 21: Separate clusters from UMAP and HDBSCAN clusterization

Spectra from each cluster were accumulated and preprocessed (as was discussed in Section [2.4]) before being passed to classifier - see Figure [22] and Figure [23].

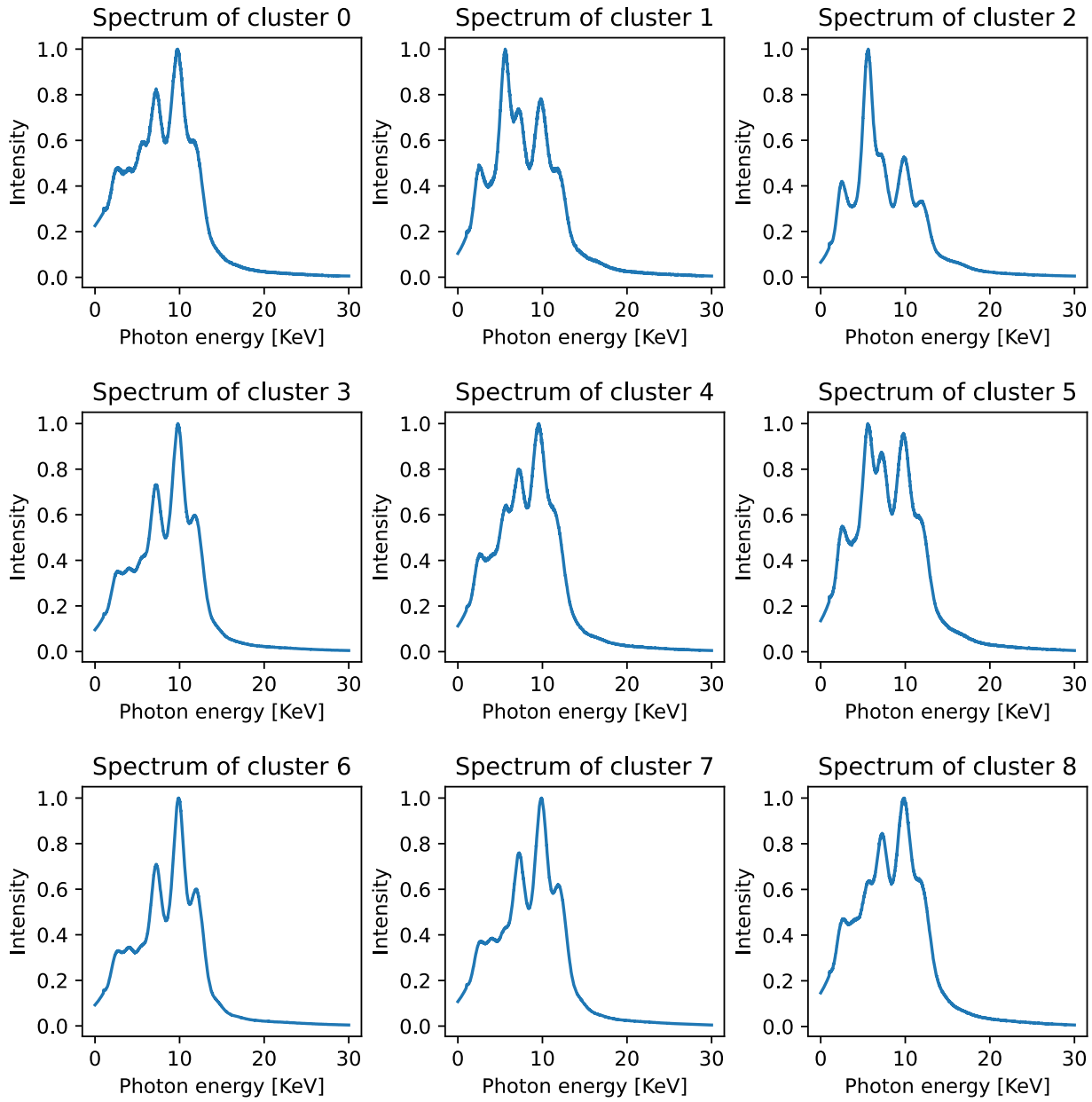


Figure 22: Processed spectra from SOM clusterization

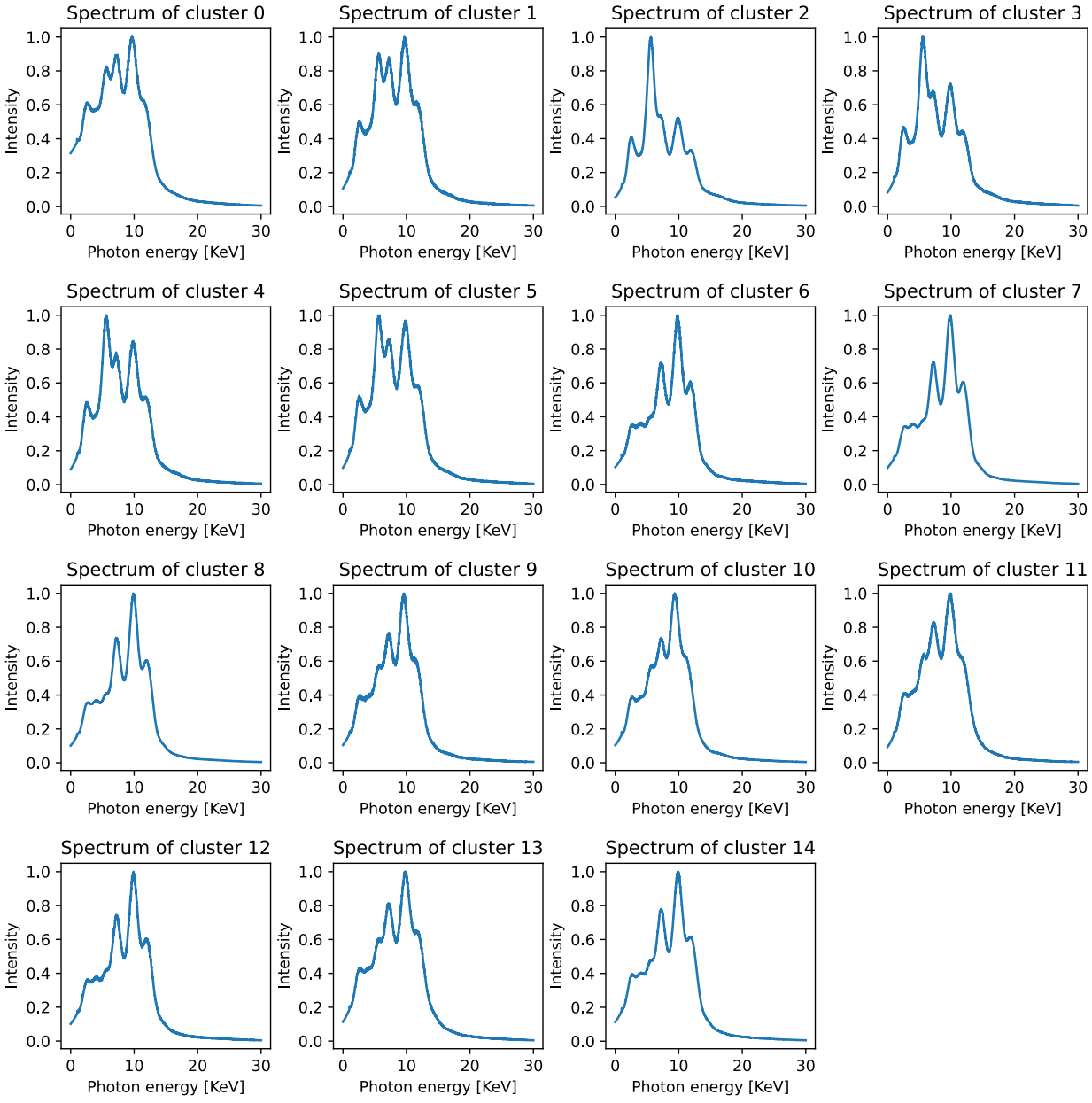


Figure 23: Processed spectra from UMAP and HDBSCAN clusterization

3.2 Training ViT

Results visible in next pages were obtained using ViT model trained with the hyperparameters visible in Listing [8].

```
input_size = 4096  # Size of the input spectrum
patch_size = 16    # Size of each patch. In this case there will be input_size / patch_size = 256 patch embeddings. Smaller values are more computationally costly, as attention calculation scales as O(n^2) with number of patches
embed_dim = 16     # Dimensionality of the embedding that the patch is projected into
num_heads = 8      # Number of attention heads in the model. Each head will have dimension embed_dim / num_heads = 2
num_classes = 20   # Number of output classes
num_layers = 6     # Number of the transformer blocks
hidden_dim = 4 * embed_dim  # Dimensionality of the hidden layer inside transformer block
dropout_rate = 0.1  # Dropout rate used in every dropout operation

model = VisionTransformer(input_size, patch_size, embed_dim, num_heads, num_classes, num_layers, hidden_dim, dropout_rate)
```

Listing 8: ViT model initialization

For the training phase, 90,000 samples were generated, comprising 10,000 samples for different number of elements ranging from 1 to 9. 90% of the samples were utilized in the training step, while the remaining 10% were allocated for the validation. The Adam optimizer was employed with a learning rate of 0.0001, and Binary Cross Entropy loss was used because multi-label classification reduces to multiple binary classification problems.

The model was trained for 10 epochs, with a batch size of 128. Loss through epochs can be seen in Figure [24]. Training loss was averaged over all mini batches in epoch, which is why it is generally larger than validation loss, which was calculated after the epoch.

After training, the ROC (Receiver Operating Characteristics) curve over 5000 random samples was calculated to assess the model's performance. The ROC curve is a plot of recall (sensitivity) against $1-$specificity [Gajowniczek et al.(2014)]. Recall, also known as the TPR (True Positive Rate) measures the ability of a model to capture all the relevant instances and is defined as $\frac{\text{TP}}{\text{TP+FN}}$ (look Table [1] for abbreviations). $1-$specificity, or FPR (False Positive Rate) is defined as $1-\frac{\text{TN}}{\text{TN+FP}}=\frac{\text{FP}}{\text{TN+FP}}$ and says what proportion of actual positive instances were are incorrectly identified as negative. When the threshold for positive classification, e.g., 0.5, is lowered, the number of positive classifications increases, resulting in an increase of both TPR and FPR. However, lowering the threshold will lead to an increase in false positives, possibly greater than true positives, causing the $\frac{\text{TPR}}{\text{FPR}}$ ratio to decrease.

In general, better classifiers should have a greater area under the ROC curve, known as

AUC (Area Under the Curve). The ROCs and AUCs were calculated using the best weights of the trained model and are visible in Figure [25].

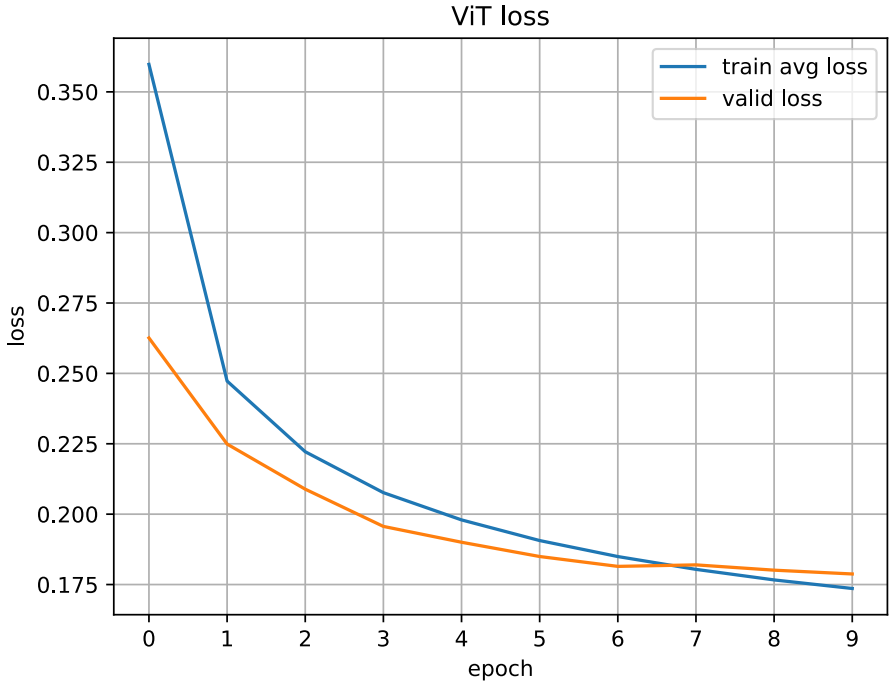


Figure 24: Loss in each training epoch

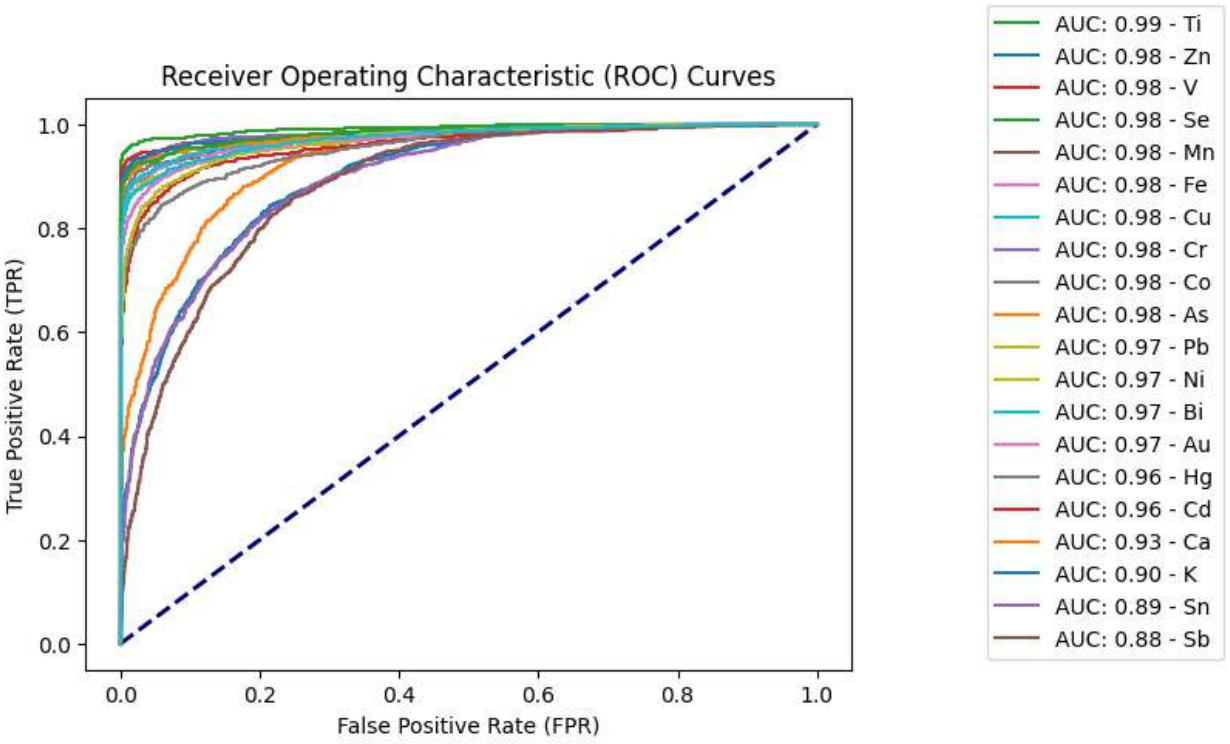


Figure 25: ROC and AUC of each trained element

ROC of a few elements, namely Ca, K, Sn, and Sb, seems to differ significantly from

the rest (see Figure [25]). The energies of their spectral lines are as follows: 3.692, 3.314, 3.444, and 3.604 [keV]. As one can see, these values are quite close to each other, and no other spectral lines were defined in `element_lines` (see Listing [4]) argument that could help differentiate between elements. In that case model could be improved by providing more accurate theoretical spectra.

Besides ROC, a few metrics were evaluated for different number of elements in spectra, namely: accuracy, precision, recall and f1 score. Precision measures the accuracy of the positive predictions and is evaluated as $\frac{TP}{FP+TP}$. f1 score is popular metric that unifies precision and recall under one metric evaluated as a harmonic mean $2 \times \frac{precision \times recall}{precision+recall}$. The evaluation was conducted on 1000 random samples within each category and is visible in Figure [26].

Based on the precision histogram visible in Figure [26], it can be concluded that the count of false positives grows rapidly with the number of elements in the spectrum. It is hard to determine how this issue may be mitigated; perhaps the best course of action is to compare results with another method. Besides attempts to improve the existing model, hyperparameters or training process, it may be worthwhile to try training separate classifiers for each element and assess whether such an approach could yield better results.

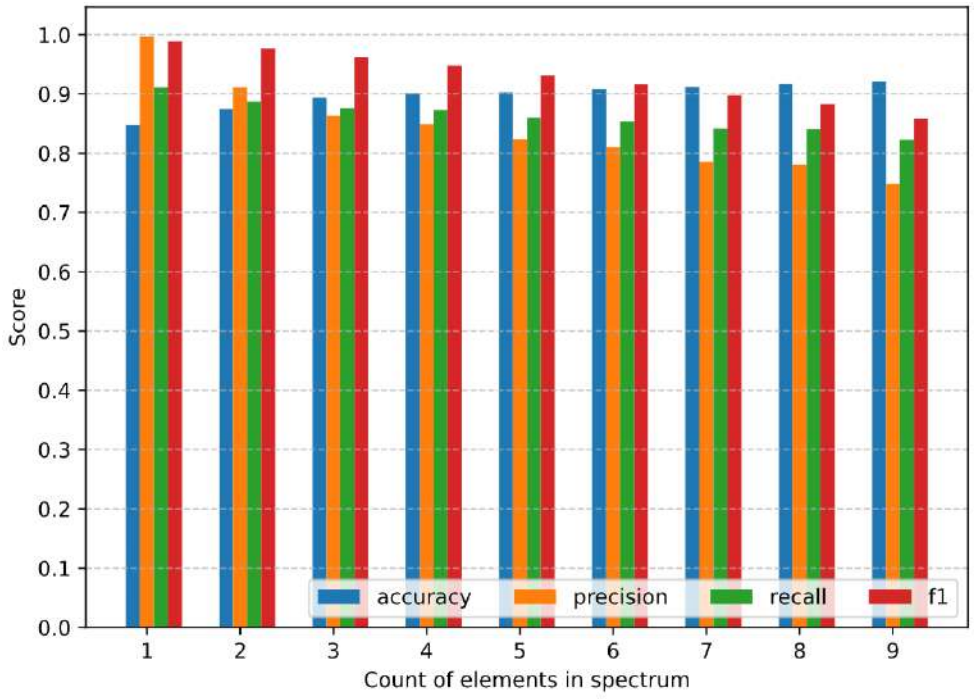


Figure 26: Metrics for different number of elements in spectrum

	Real Positive	**Real Negative**
Predicted Positive	TP (True Positive)	FP (False Positive)
Predicted Negative	FN (False Negative)	TN (True Negative)

Table 1: Classification matrix

3.3 Classifying Spectra

3.3.1 Simple Spectrum Classification

After training the model was tested on a "base case", that is accumulated spectrum of copper PCB shown in Figure [27]. The output probabilities are visible in Figure [28].

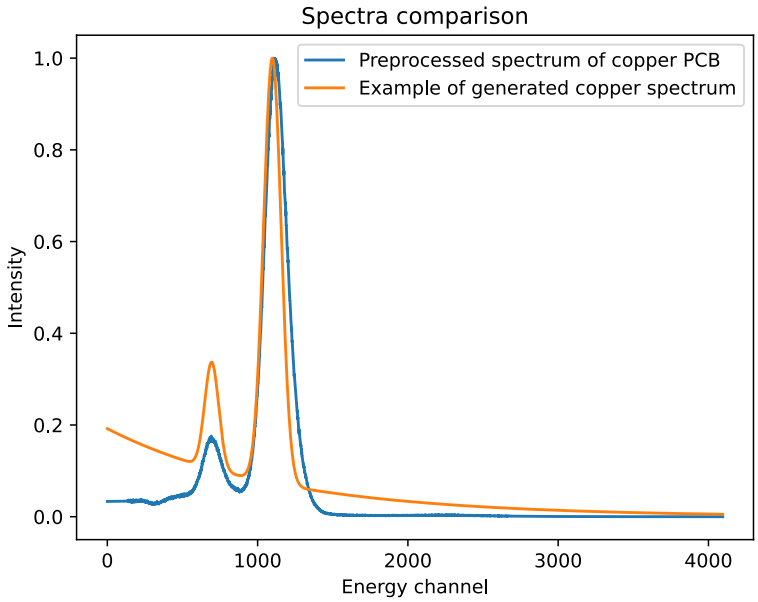


Figure 27: Real and artificial spectrum comparison

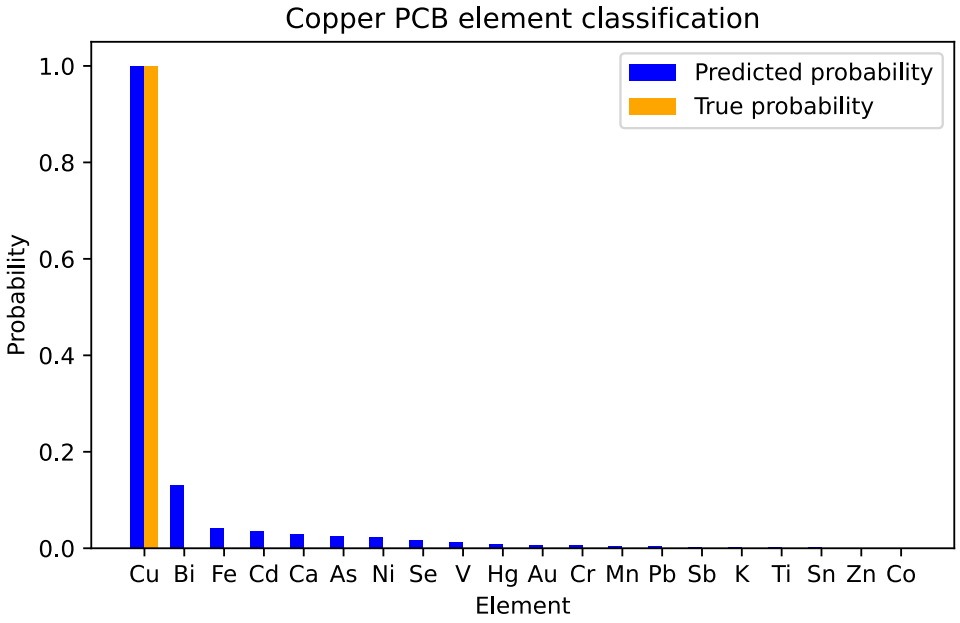


Figure 28: Classification results

For this simple case, the results are satisfactory.

3.3.2 Classification of Clustered Spectra

Now, while having in mind that the model is not ideal, the results of clustered spectra classification can be presented. Spectra were clustered, accumulated and preprocessed as presented in Section [3.1.4]. In the last step they were classified and probability maps for each element were visualized. Due to probabilistic nature the maps they do not provide any quantitative information about elements. Results are shown in Figure [29], Figure [30] and Figure [31].

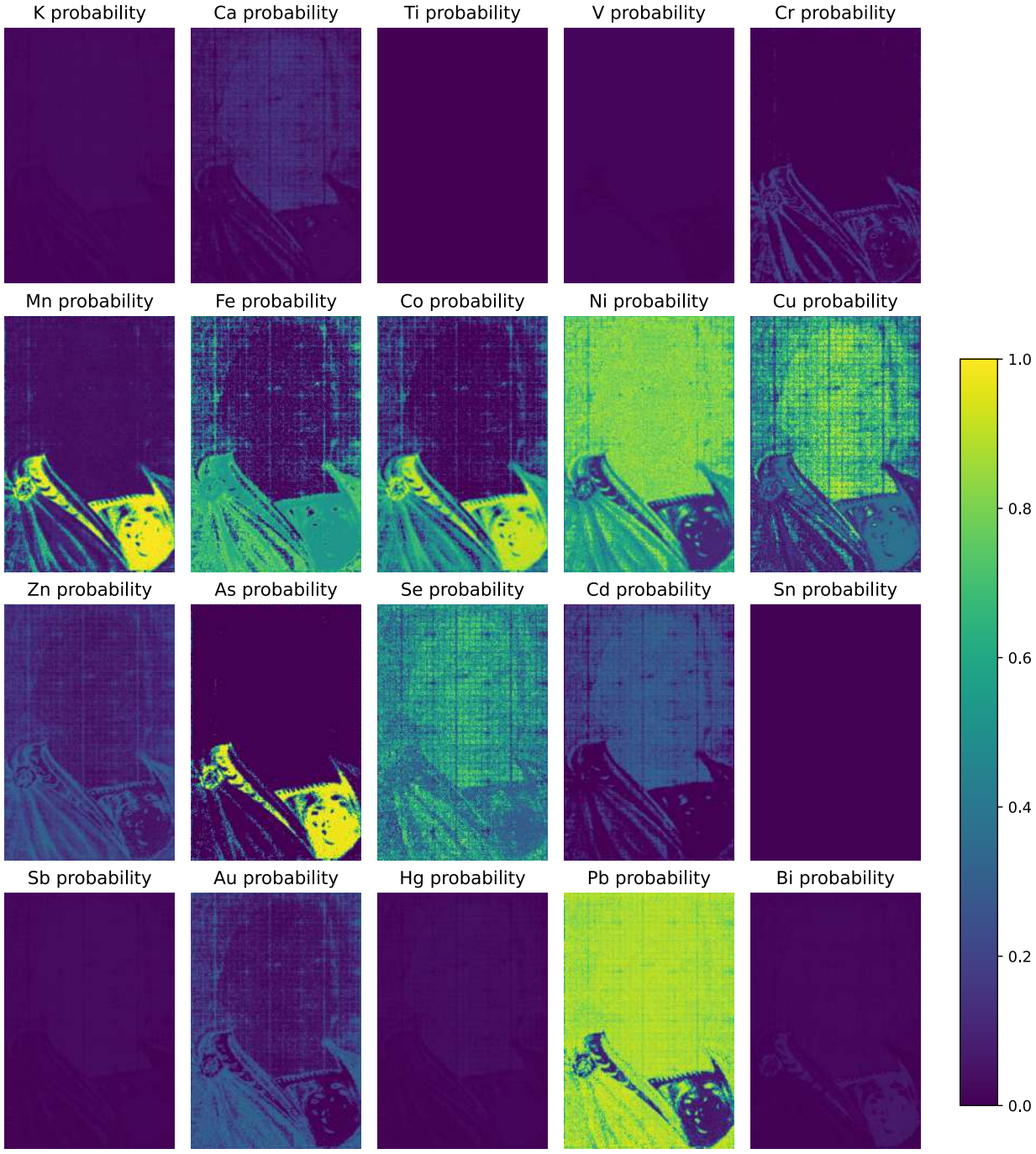


Figure 29: Probability maps of clustered spectra of "Portrait of John III Sobieski in Karacena Scale Armour"

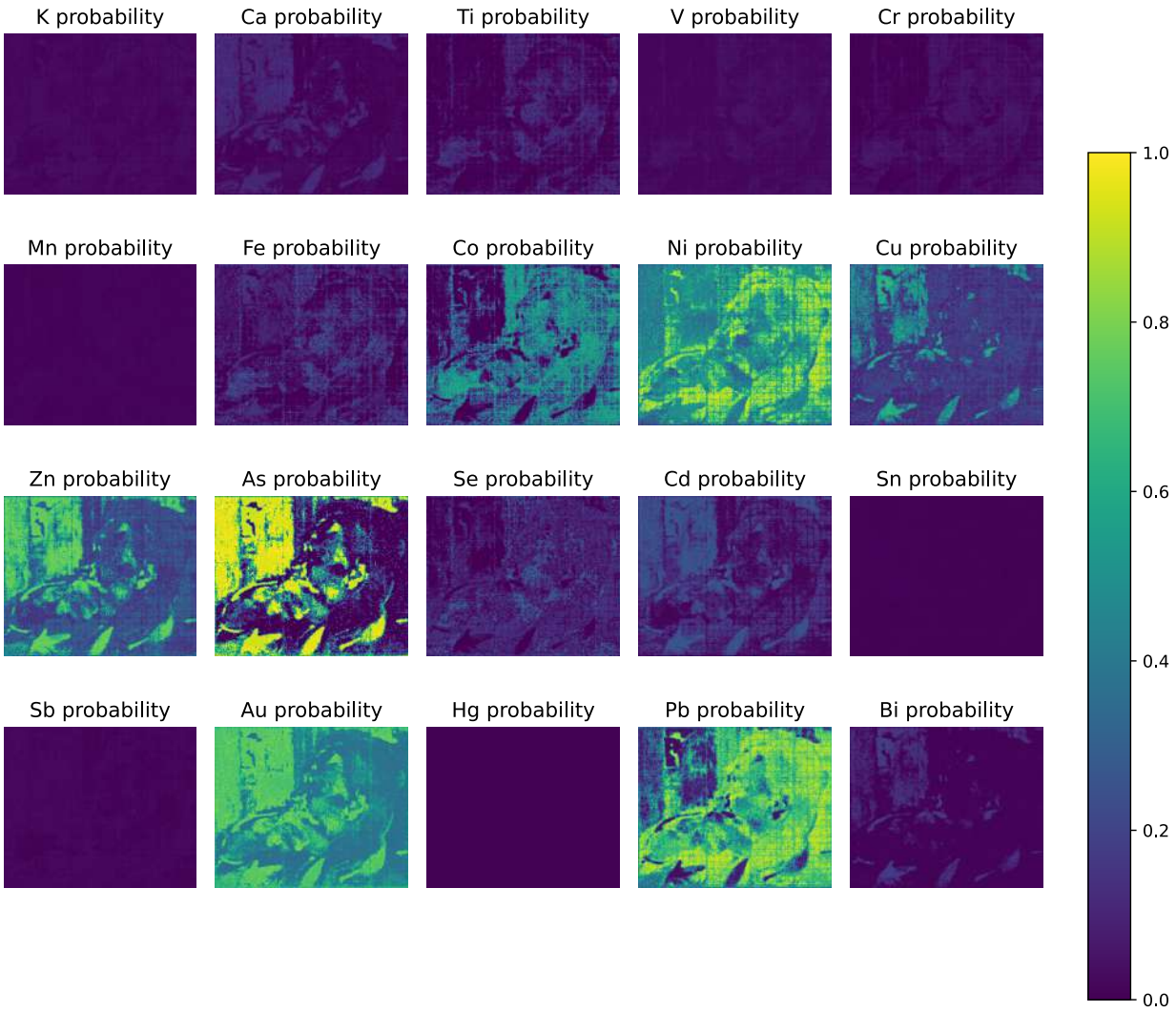


Figure 30: Probability maps of clustered spectra of “Portrait of Mieczysław Gąsecki”

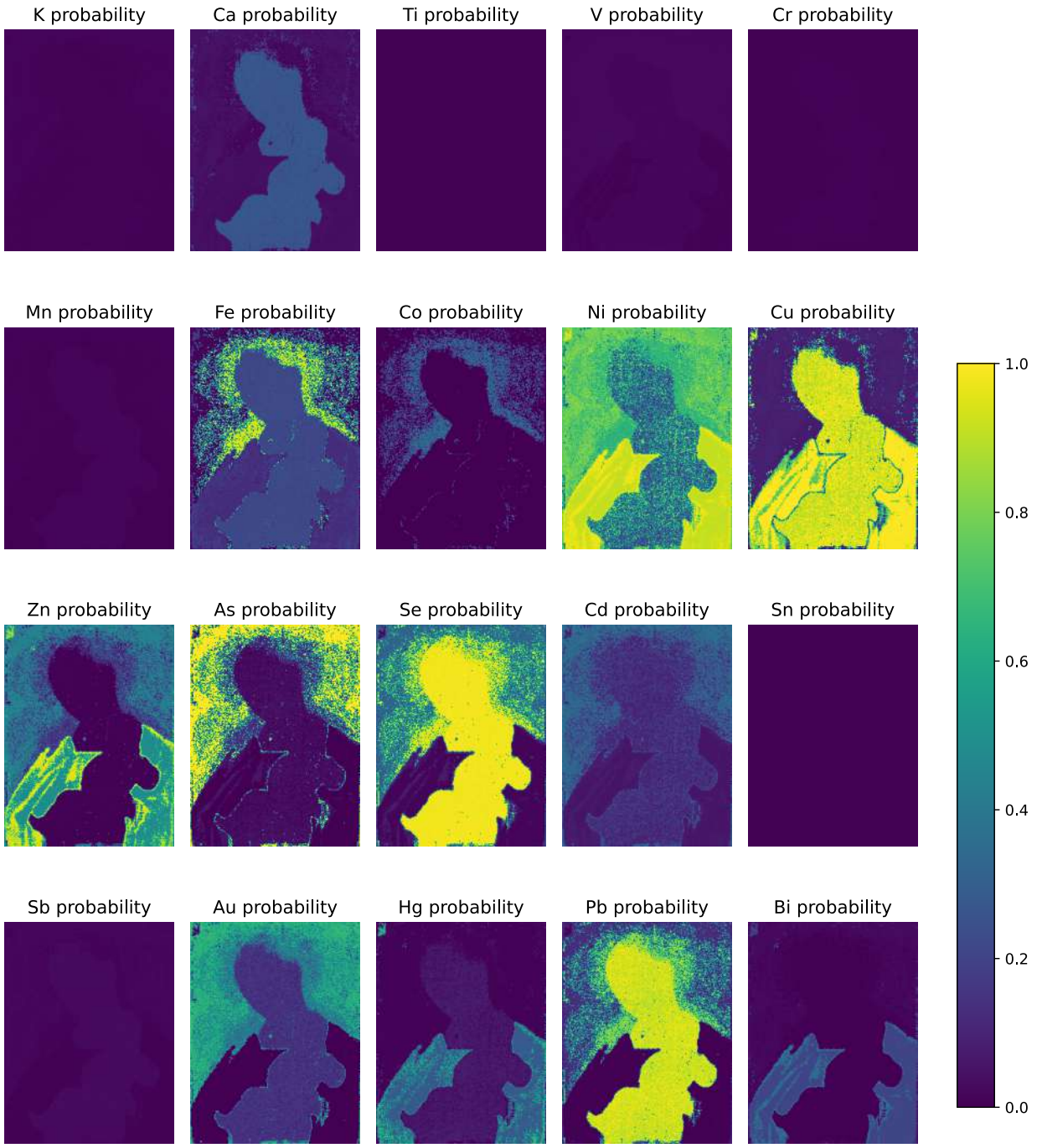


Figure 31: Probability maps of clustered spectra of "Mother of God with the Child Eating an Apple"

As expected, many false positives appeared in probability maps. On the other hand obtained results are still quite decent. As one can see in Figure [32], Pb was detected pretty well in places of higher concentration. Little worse, but Fe was also registered. Cu was mostly registered incorrectly because the escape photons of the L_α spectral line of Pb (7.59 keV) have energy close to the K_α line (8.05 keV) of Cu [Łach(2022)]. This particular line was not included in the training process, so adding it could potentially improve the results.

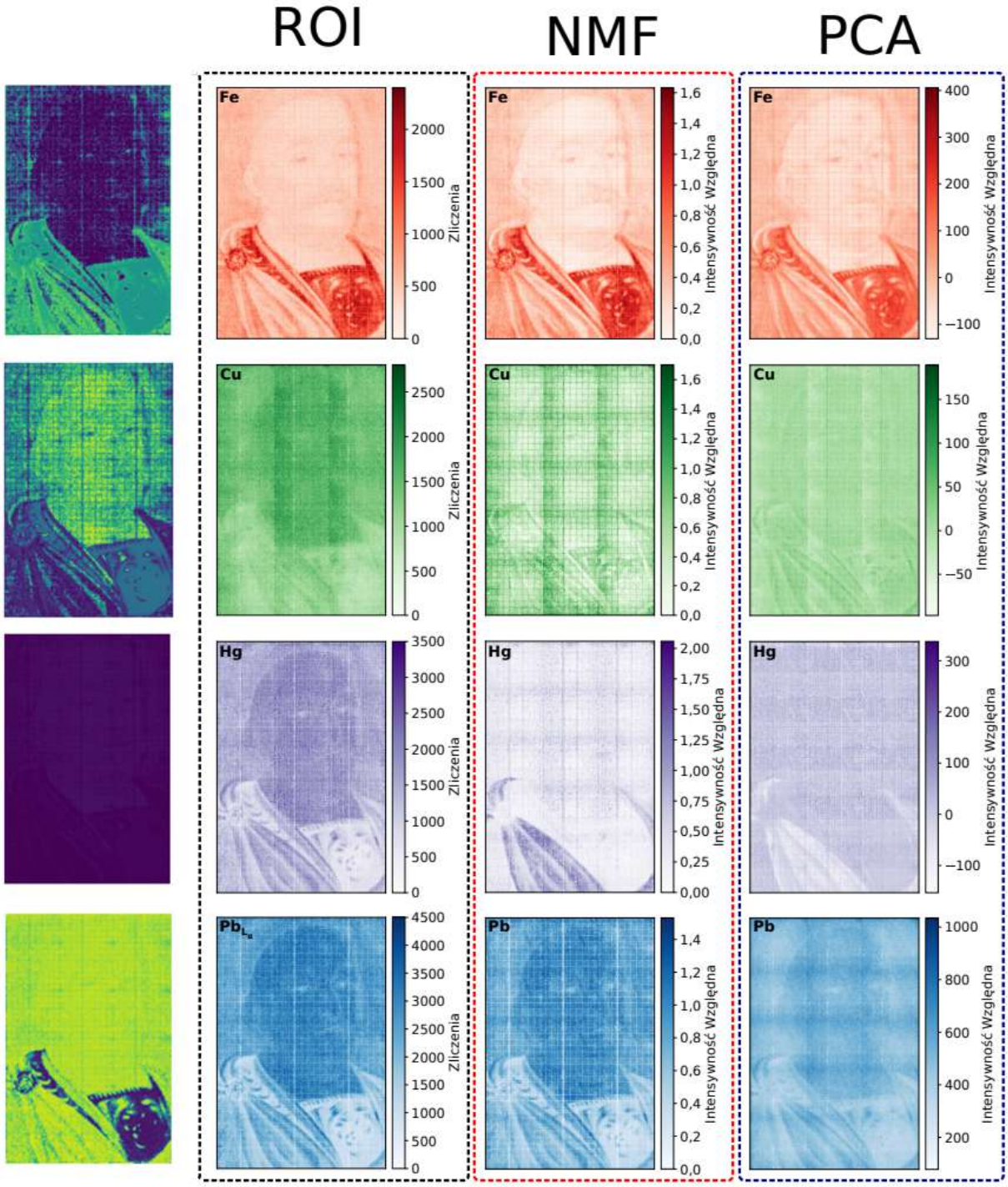
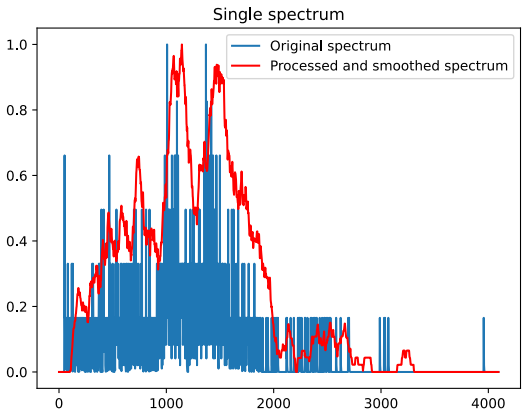


Figure 32: Comparison of probability maps with element distribution maps from [Łach(2022)]

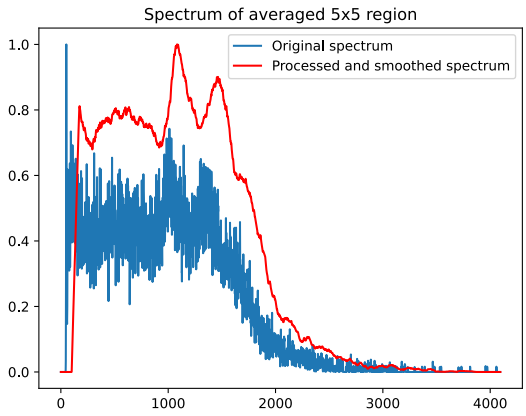
3.3.3 Classification of Small Regions

Another option was to classify single spectra or spectra averaged over small regions, e.g. 10×10 points.

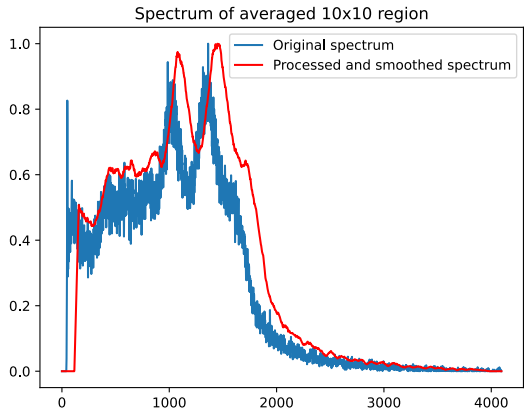
Savitzky–Golay filter implementation from `scipy` package was used to smooth spectra before further processing. Extrapolation step was omitted to reduce computational cost and issues with extrapolation of noisy spectra. Example spectra are shown in Figure [33].



(a) Single spectrum



(b) Spectrum averaged over 5×5 region



(c) Spectrum averaged over 10×10 region

Figure 33: Example spectra

It is clearly visible that averaging over larger regions yield much clearer spectra, but it must be kept in mind that spatial resolution is lost in this way.

Resulting probability maps for different region sizes are shown in Figure [34], Figure [35] and Figure [36].

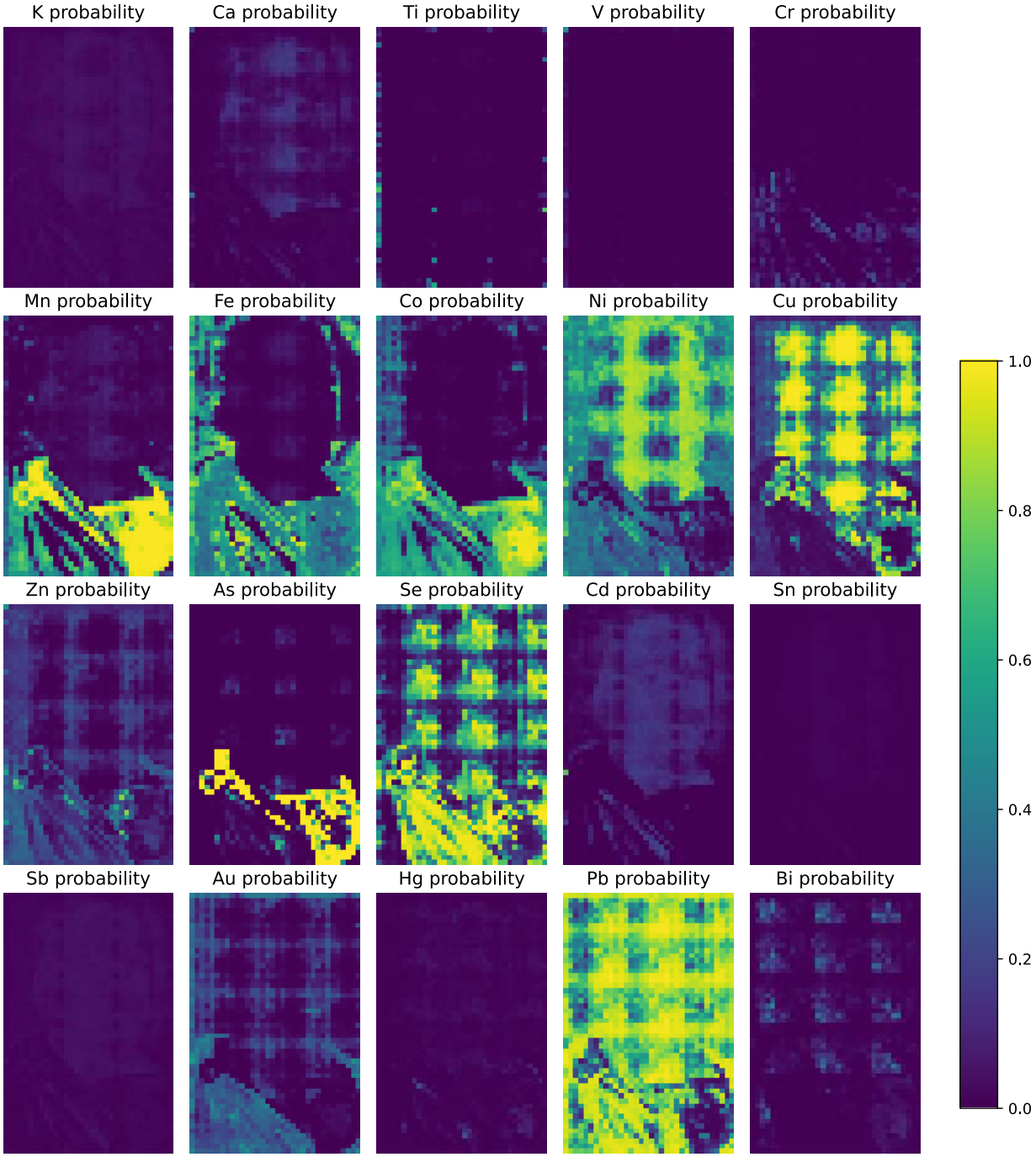


Figure 34: Probability maps calculated over 10×10 segments of "Portrait of John III Sobieski in Karacena Scale Armour"

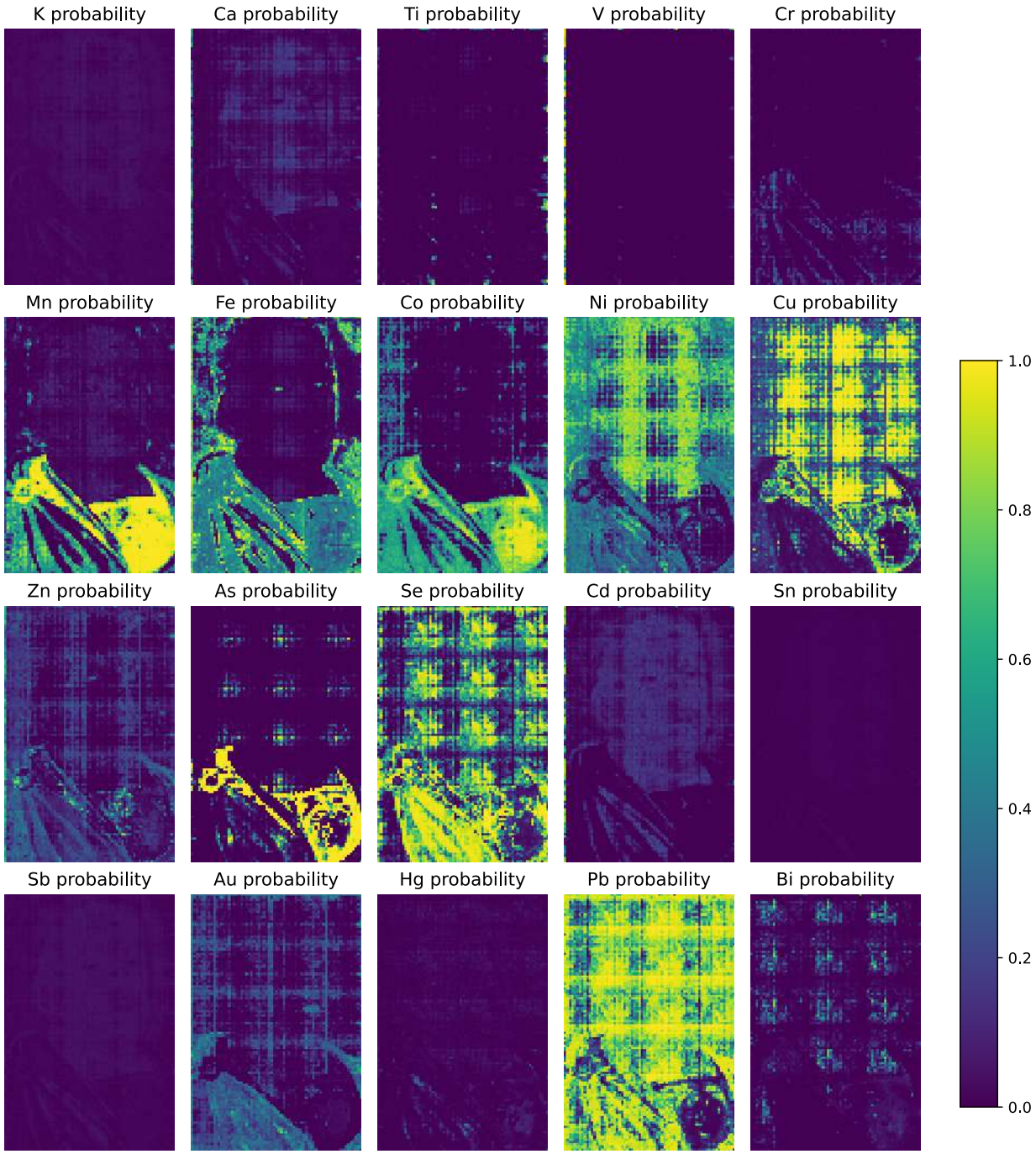


Figure 35: Probability maps calculated over 5×5 segments of "Portrait of John III Sobieski in Karacena Scale Armour"

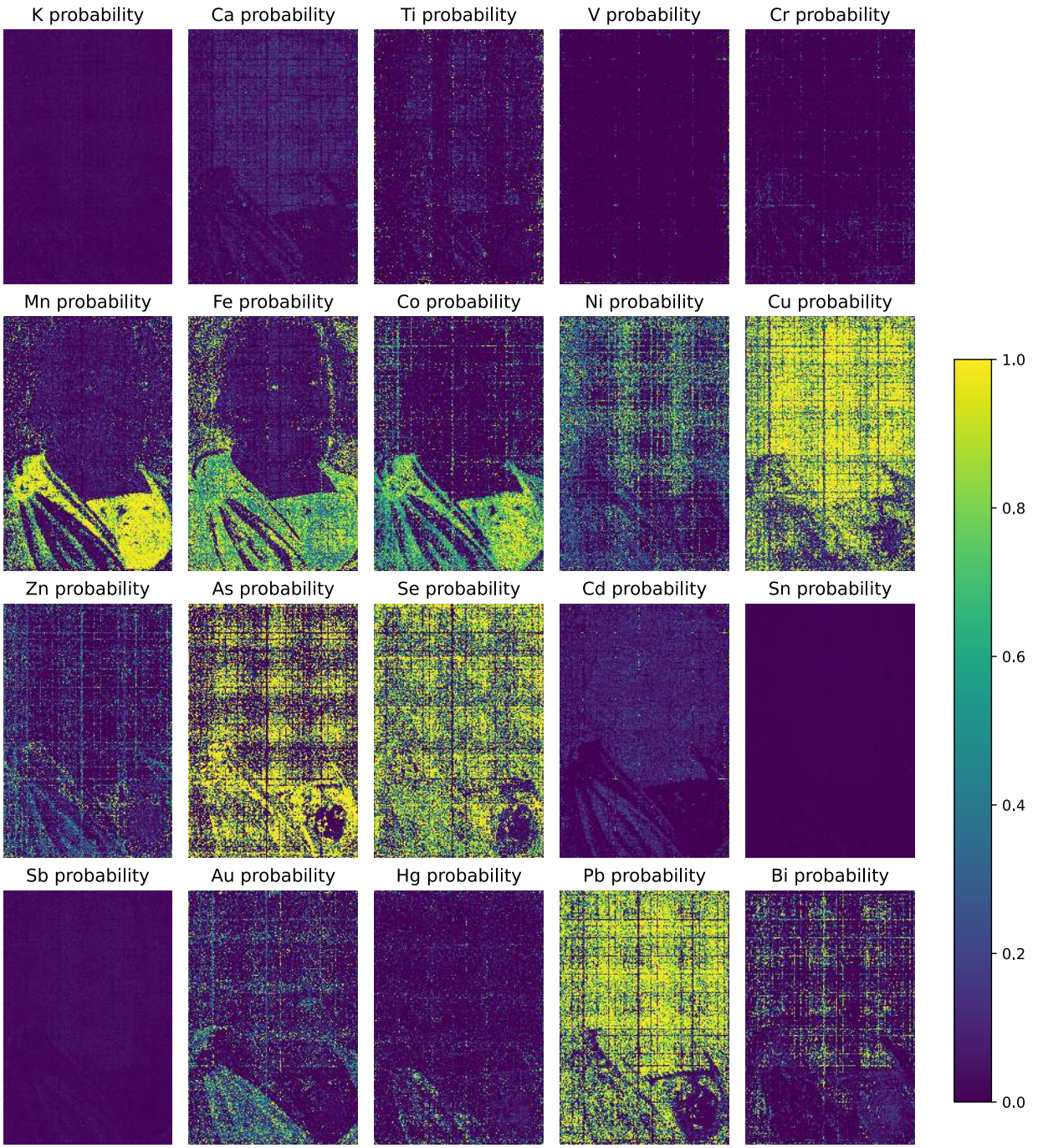


Figure 36: Probability maps calculated over single points of "Portrait of John III Sobieski in Karacena Scale Armour"

It seems that despite the author's concerns classification of single spectra works decent. As Figure [36] shows, it is the only case when the model succeeded to somehow register Hg element, which was invisible in Figure [32]. However, it took a lot of time. Inference of all spectra took approximately 20 minutes on T4 GPU for this fairly small model (10^6 parameters).

Similar probability maps of another two paintings are visible in Figure [37] and Figure [38].

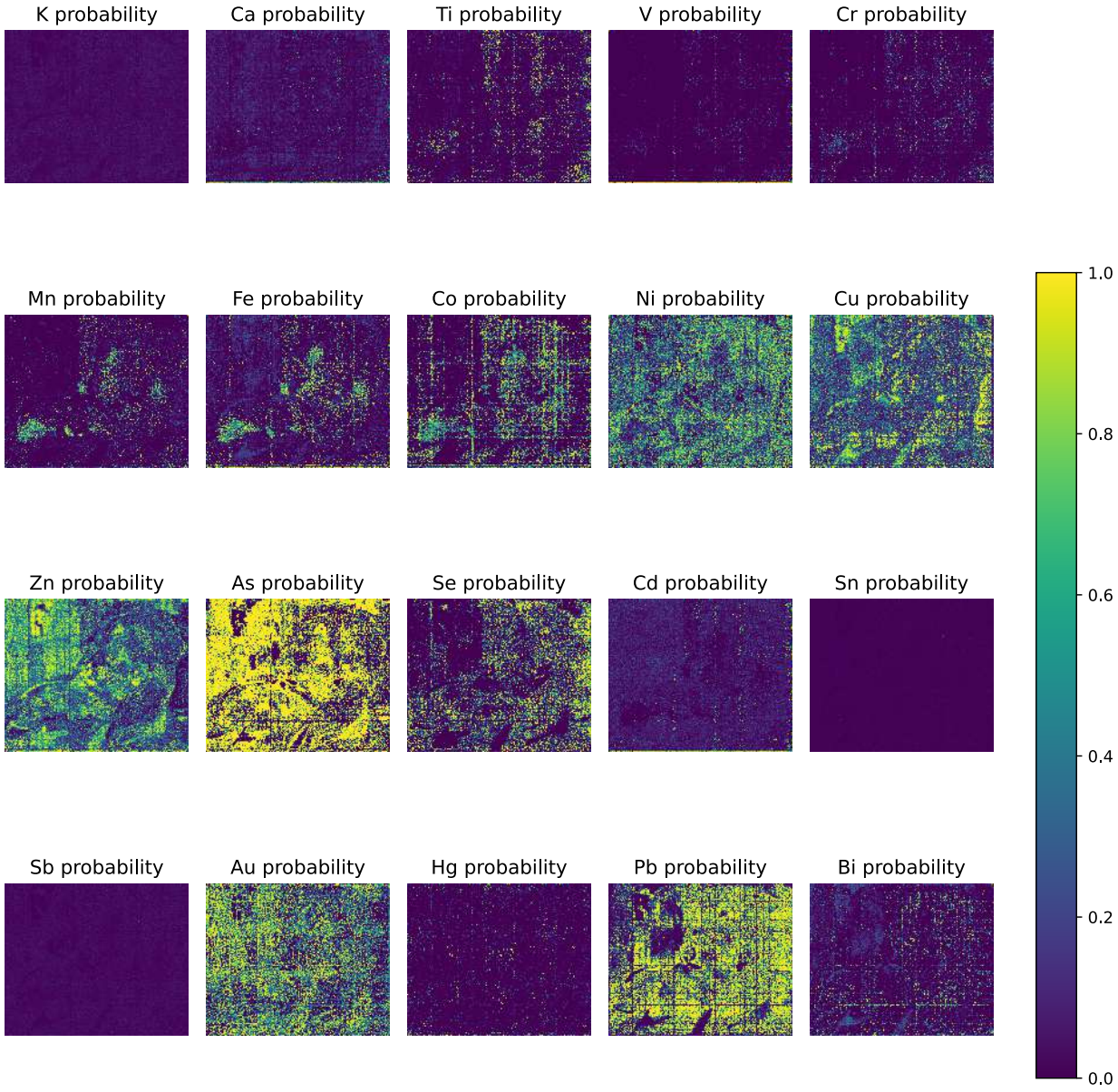


Figure 37: Probability maps calculated over single points of "Portrait of Mieczysław Gąsecki"

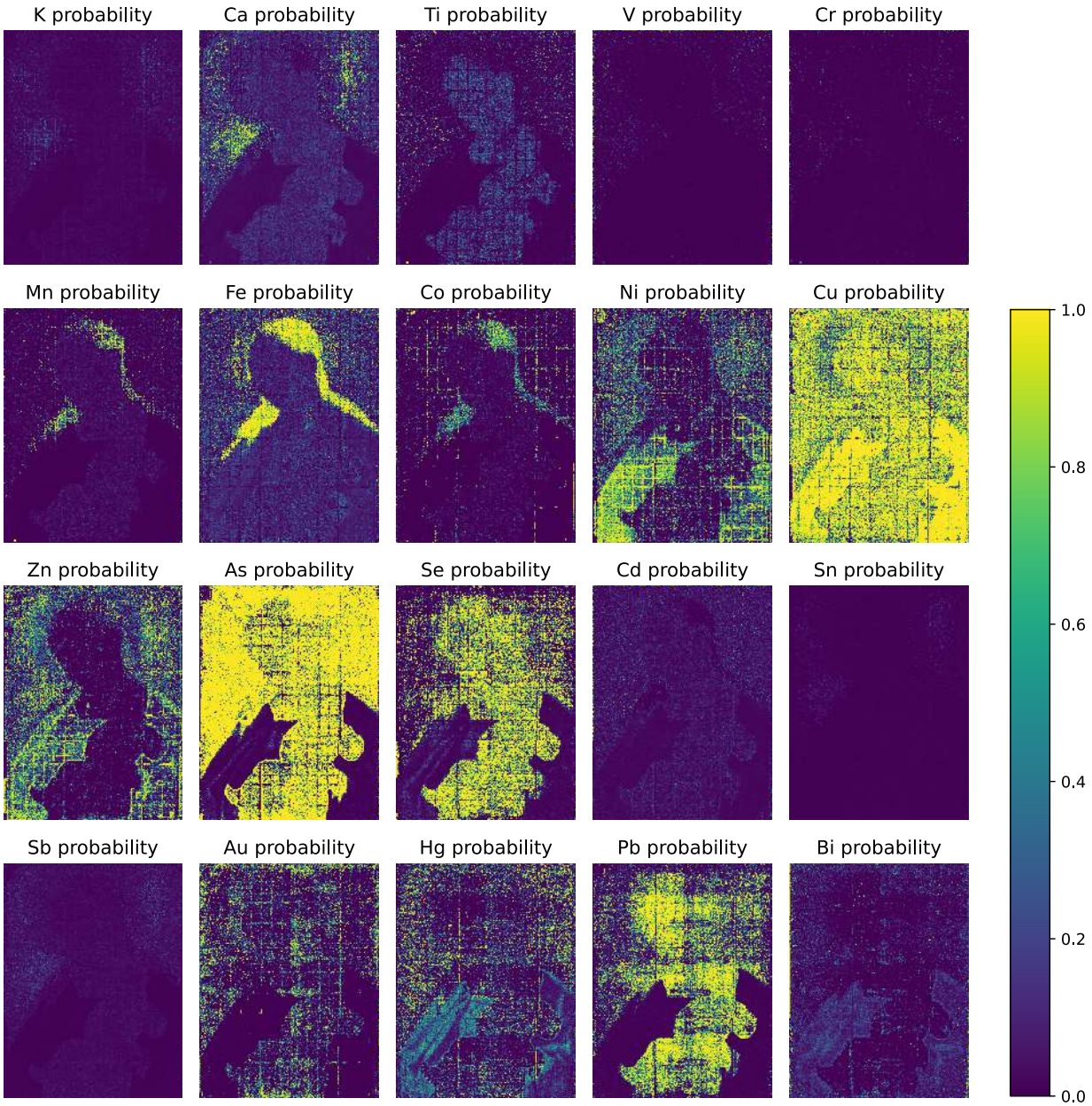


Figure 38: Probability maps calculated over single points of "Mother of God with the Child Eating an Apple"

4 Conclusion

In conclusion, this bachelor thesis aimed at addressing the problem of classifying elements present in XRF spectra, utilizing deep learning methods. Originally, the ResNet architecture was selected for its proven robustness. However, its utilization of global average pooling resulted in the loss of spatial information and poor performance. Consequently, it was replaced with the Visual Transformer-based architecture.

The classification process was supported by preprocessing of real spectra to ensure reliable classification and artificial data generation in order to train the model in the absence of real labeled data. Clusterization of spectra was employed as an attempt to deal with noisy spectra, although its significance proved to be less crucial than author expected.

The trained model demonstrated effectiveness on artificial data and some success on real data. However, issues such as too many false positive cases and overall low quality of classification persist. To address these issues, future efforts could focus on generating more accurate theoretical spectra or, preferably, using real spectra from reference materials to generate training data that could take into account the characteristics of the data acquisition apparatus. Additionally, the model, hyperparameters, and training process still could be improved as the results presented in this work were slightly less effective, than the best trained model. The investigation of potential issues with the pinhole camera, as mentioned in Section Section [3.1.3], is also recommended to understand whether it is causing any other issues.

References

[Ahn and Syn(2005)] Jae-Wook Ahn and Sue Yeon Syn. Som tutorial, 2005. URL https://sites.pitt.edu/~is2470pb/Spring05/FinalProjects/Group1a/tutorial/som.html. Accessed: 2023-12-12.

[Bruker(2023)] Bruker. Bruker. https://www.bruker.com/pl/products-and-solutions/elemental-analyzers/handheld-xrf-spectrometers/S1-TITAN.html, 2023. Accessed: 2023-12-06.

[Devlin et al.(2018)] Devlin et al. Bert: Pre-training of deep bidirectional transformers for language understanding, 2018. URL https://arxiv.org/abs/1810.04805. Accessed: 2024-01-14.

[Dosovitskiy et al.(2020)] Dosovitskiy et al. An image is worth 16x16 words: Transformers for image recognition at scale, 2020. URL https://arxiv.org/abs/2010.11929. Accessed: 2024-01-14.

[Gajowniczek et al.(2014)] Gajowniczek et al. Estimating the roc curve and its significance for classification models' assessment. *QUANTITATIVE METHODS IN ECONOMICS*, XV:382–391, 12 2014.

[Góral(2023)] Adrian Antoni Góral. Analiza inteligentna map przestrzennego rozkładu pierwiastków. Master's thesis, AGH University of Krakow, 2023.

[ILSVRC(2015)] ILSVRC. ImageNet Large Scale Visual Recognition Challenge 2015 results, 2015. URL https://image-net.org/challenges/LSVRC/2015/results.php. Accessed: 2023-12-06.

[Jones et al.(2022)] Cerys Jones et al. Neural network-based classification of x-ray fluorescence spectra of artists' pigments: an approach leveraging a synthetic dataset created using the fundamental parameters method. *Heritage Science*, 10(1), June 2022. ISSN 2050-7445. doi: 10.1186/s40494-022-00716-3. URL http://dx.doi.org/10.1186/s40494-022-00716-3.

[Kogou et al.(2020)] Sotiria Kogou et al. A new approach to the interpretation of xrf spectral imaging data using neural networks. *X-Ray Spectrometry*, 50(4):310–319, August 2020. ISSN 1097-4539. doi: 10.1002/xrs.3188. URL http://dx.doi.org/10.1002/xrs.3188.

[Matthias et al.(2013)] Alfeld Matthias et al. A mobile instrument for in situ scanning macro-xrf investigation of historical paintings. *Journal of Analytical Atomic Spectrometry*, 28 (5):760–767, 2013. doi: 10.1039/c3ja30341a.

[McInnes et al.(2016)] McInnes et al. Hdbscan documentation, 2016. URL https://hdbscan.readthedocs.io/en/latest/index.html. Accessed: 2024-01-04.

[McInnes et al.(2018a)] McInnes et al. Umap documentation, 2018a. URL https://umap-learn.readthedocs.io/en/latest/. Accessed: 2024-01-04.

[McInnes et al.(2018b)] McInnes et al. Umap faq, 2018b. URL https://umap-learn.readthedocs.io/en/latest/faq.html. Accessed: 2023-12-13.

[Mormille et al.(2023)] Luiz H. Mormille et al. Introducing inductive bias on vision transformers through gram matrix similarity based regularization. *Artificial Life and Robotics*, 28(1):106–116, January 2023. ISSN 1614-7456. doi: 10.1007/s10015-022-00845-9. URL http://dx.doi.org/10.1007/s10015-022-00845-9.

[Papers with Code(2023)] Papers with Code. Trends. https://paperswithcode.com/trends, 2023. Accessed: 2023-12-06.

[Pavan M. V. et al.(2023)] Raja Pavan M. V. et al. Auger electron spectroscopy. https://chem.libretexts.org/Bookshelves/Analytical_Chemistry/Physical_Methods_in_Chemistry_and_Nano_Science_(Barron)/01%3A_Elemental_Analysis/1.14%3A_Auger_Electron_Spectroscopy, 2023. Accessed: 2023-11-28.

[Ralhan(2018)] Abhinav Ralhan. kohonen-maps. https://github.com/abhinavralhan/kohonen-maps/blob/master/som-random.ipynb, 2018. Accessed: 2024-01-13.

[Vallés-Pérez(2015)] Iván Vallés-Pérez. Self-organizing map graphic, 2015. URL https://ivape3.blogs.uv.es/2015/03/15/self-organizing-maps-the-kohonens-algorithm-explained/. Accessed: 2023-12-13.

[Vaswani et al.(2017)] Ashish Vaswani et al. Attention is all you need, 2017. URL https://arxiv.org/abs/1706.03762. Accessed: 2024-01-14.

[Wikimedia(2023)] Wikimedia. File:unknown - portrait of john iii sobieski (1624–1696) in karacena scale armour - mnk xii-356 - national museum kraków.jpg — wikimedia commons, the free media repository, 2023. URL https://commons.wikimedia.org/w/index.php?title=File:Unknown_-_Portrait_of_John_III_Sobieski_(1624%E2%80%931696)_in_Karacena_Scale_Armour_-_MNK_XII-356_-_National_Museum_Krak%C3%B3w.jpg&oldid=832190876. Accessed: 2023-12-23.

[Wikipedia(2023)] Wikipedia. Self-organizing map, 2023. URL https://en.wikipedia.org/wiki/Self-organizing_map. Accessed: 2023-12-12.

[Zhang et al.(2022a)] Aston Zhang et al. Queries, keys, and values, 2022a. URL https://d2l.ai/chapter_attention-mechanisms-and-transformers/queries-keys-values.html. Accessed: 2023-12-08.

[Zhang et al.(2022b)] Aston Zhang et al. Attention scoring functions, 2022b. URL https://d2l.ai/chapter_attention-mechanisms-and-transformers/attention-scoring-functions.html. Accessed: 2023-12-09.

[Zhang et al.(2022c)] Aston Zhang et al. Multi-head attention, 2022c. URL https://d2l.ai/chapter_attention-mechanisms-and-transformers/multihead-attention.html. Accessed: 2023-12-09.

[Zhang et al.(2022d)] Aston Zhang et al. Residual networks (resnet), 2022d. URL https://d2l.ai/chapter_convolutional-modern/resnet.html. Accessed: 2023-12-07.

[Zhang et al.(2022e)] Aston Zhang et al. Transformers for vision, 2022e. URL https://d2l.ai/chapter_attention-mechanisms-and-transformers/vision-transformer.html. Accessed: 2023-12-15.

[Łach(2022)] Bartłomiej Łach. *Rozwój systemu detekcyjnego do obrazowania przestrzennego rozkładu pierwiastków metodą fluorescencji rentgenowskiej*. PhD thesis, AGH University of Krakow, 2022.