

TDK-dolgozat

Lieb Hanna-Georgina

Kaszta Tamás

37. Országos Tudományos Diákköri Konferencia

Informatikatudományi szekció

Budapest, 2025. április 23–25.

Wavelet-alapú képosztályozás

Prototípus alapú modellek

Szerzők:

Lieb Hanna-Georgina

Babeş–Bolyai Tudományegyetem, Matematika és Informatika kar, Adatelemzés és modellezés,

I. év

Kaszta Tamás

Babeş–Bolyai Tudományegyetem, Matematika és Informatika kar, Adatelemzés és modellezés,

I. év

Témavezetők:

dr. Csató Lehel

Babeş–Bolyai Tudományegyetem, Matematika és Informatika kar



Abstract

Prototype-based networks make predictions on interpretable representations of the input image, so-called prototypes, a structural part of the decision-making system. The need for transparent decisions in safety-critical domains is becoming a burning question with the widespread adoption of intelligent systems in diverse areas.

We used a prototype-based network, ProtoPNet, as our starting point, considering its strengths, namely the scalability of the prototypes, and its weaknesses, specifically the need for considerable computational resources. We propose a novel architecture, WaveProtoPNet, which is based on prototypes, but the feature extraction is realised through wavelets, a complex and powerful mathematical tool successfully employed in image processing.

Our results are outlined, with an emphasis on the interpretability and performance of the system. Building wavelets into the network results in a dramatic reduction of learnable parameters, leading to lower memory consumption; wavelets also form a bijection between the input and the feature spaces, a simple way to visualise prototypes. Visualised prototypes combined with the weights of the final layer provide powerful insight into the decision-making process. Our model remains highly adaptable, as images with custom sizes could be learnt.

Based on experiments on the NCT-CRC-HE-100K dataset, which encompasses high-quality real-world medical data, we highlight the strengths of WaveProtoPNet with respect to different parameter settings and also discuss its shortcomings.

Contents

Introduction	1
1 Machine Learning and Explainability	2
1.1 Supervised, Unsupervised and Reinforcement Learning	2
1.2 Neural Networks and CNNs	3
1.3 Explainability	5
2 Wavelets	6
2.1 Fourier-transform	6
2.2 Wavelets	7
3 Wavelet-based Prototype Network - waveProtoPNet	11
3.1 Prototype-based architecture	11
3.2 Structural details	11
3.3 Training	13
3.4 Wavelets on GPU	14
3.5 Decomposition level and overlap	15
3.6 Receptive field	17
4 Experiments and results	19
4.1 Dataset	19
4.2 Training parameters	20
4.3 Backbone size	20
4.4 Number of prototypes	21
4.5 Decomposition level and Overlap	22
4.6 Performance with different wavelets	22
4.7 Prototype-check	24
4.8 Misclassification	24
Conclusions and future work	26

Introduction

The adoption of intelligent systems is embraced in various safety-critical domains, although the knowledge to thoroughly grasp the inner mechanisms and the mathematical tools for modeling them is highly deficient. Therefore, an urgent need is presented to get insight into the decision-making process to prevent dysfunctions that could lead to serious shortcomings. In our investigations, we focus on medical data, as this usage domain of image processing is one of the utmost complexity and

To investigate the inner states of a model and obtain some explanation for its actual output, prototypes were introduced. Prototypes are structural parts of a decision-making system through which input data flows. They could be visualised in the input space. Various approaches have been discussed in PrototypeDL, where the authors built a model based on prototypes of input image sizes. Their idea was further refined into ProtoPNet, which uses prototypes smaller than the input size. In such a way, important details about different classes may be distilled.

However, the more classes in the dataset, the more prototypes are necessary for an accurate prediction. This implies the introduction of additional training parameters, resulting in larger computational resources and longer training times.

To address the weaknesses mentioned above, we introduce wavelets into the prototype-based model. Wavelets are a mathematical tool for signal processing. There have been stunning results when wavelets are applied in image processing [11], and are also used together with Neural Networks [5].

In our paper, we present a network specialised for image processing based on prototypes and wavelets, WaveProtoPNet. This new system is based on a fusion of ideas coming from ProtoPNet and wavelet-based signal processing. It addresses the problem of extreme parameter numbers, by replacing the backbone of the system, with feature extraction based on wavelets.

This paper is organised as follows: In the first part, we introduce the prototype-based model, highlighting the features of ProtoPNet on which we build. Then we cover some basic concepts of the wavelet transform we are using to extract meaningful features. In Section 3, the structure of WaveProtoPNet is sketched. Subsequently, come our experiments regarding the efficiency of the system concerning different parameter settings. In the Conclusion, our results are summarised and further research ideas are discussed.

1 Machine Learning and Explainability

Machine learning (ML) is a subfield of artificial intelligence (AI) which aims to create systems and algorithms with the ability to learn certain characteristics from previously presented data. Based on these learnt features, assumptions are made on new data.

In the following section, notions are presented that are used in the context of wavelet-based image classification: unsupervised and supervised learning, convolutional neural networks, and explainability of ML models.

1.1 Supervised, Unsupervised and Reinforcement Learning

Machine learning has three main categories: supervised, unsupervised, and reinforcement learning. When talking about **supervised machine learning**, each training data comes with a **label**, which stands for the class from which the image comes from. In **unsupervised machine learning**, there are no labels associated with the training data. During **reinforcement learning**, the model interacts with the outside world and receives rewards for its actions, with the aim of attaining maximum reward. The aim of supervised learning is to find a function mapping the input space to the labels, while in the case of unsupervised learning we desire to deduce certain patterns inherent in the training data set.

A training data set is used to train the model. A test data set is used to evaluate the model's performance on unseen data. The validation data set is used to adjust the hyper-parameters of the model. Hyper-parameters are altered after the training is finished.

The data used by an ML model could be separated into three groups: **train**, **test** and **validation**. The train and test data is used during training and testing of the model, respectively, while validation is kept for fine-tuning the parameters of the system.

The error function calculates the performance of a ML model. During learning, we aim to minimise this function, usually the average of samples losses:

$$E(\theta) = \frac{1}{N} \sum_{n=1}^N L(f(x_i, \theta), y_i), \quad (1)$$

where N is the number of training examples, L is the loss function, f is the model, x_i - i^{th} input, y_i - i^{th} label, θ - model's parameters.

When training a model, its **parameters** (θ) are fine-tuned according to the loss function.

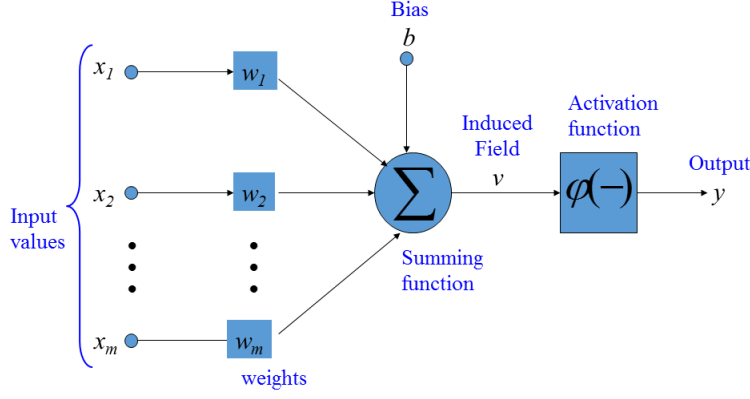


Figure 1: Model of a perceptron

(Source: <https://www.gabormelli.com/RKB/images/3/31/artificial-neuron-model.png>)

In a high-dimensional weight space, this represents taking a step towards a smaller error: $\theta = \theta - \eta \nabla_{\theta} E$. The gradient of the error function is represented by $\nabla_{\theta} E$, which is the derivative, with respect to the parameters. For minimising the error function, we follow the negative gradient direction in adapting the parameters (gradient descent). The function's minimum in the weight space is where $\nabla_{\theta} E = 0$. The **learning rate** η is a hyperparameter that controls the magnitude of a step taken in one iteration.

1.2 Neural Networks and CNNs

The processing and structural units of neural networks are neurons. It is made up of synapses (links). Each synapse has a weight by which the incoming signal is multiplied. The signals coming from all synapses are then added up (linear combination). The soma is where this takes place. The soma also has a weight (bias), which is added to the sum of inputs. This value is then modified by the activation function, which results in the output of the neuron. Activation functions are usually non-linear functions such as sigmoid, tanh, and ReLU.

As in Figure 1 can be seen, with input data $x_1, x_2, \dots, x_m$, weights $w_1, w_2, \dots, w_m$, bias b and the activation function φ , the output y neuron can be calculated:

$$v = \sum_{i=1}^m w_i x_i + b,$$

$$y = \varphi(v).$$

The neurons in neural networks are organised into layers. The first layer is the **input layer**, the last layer is the **output layer**, between the two are the **hidden layers**. The output of a neuron

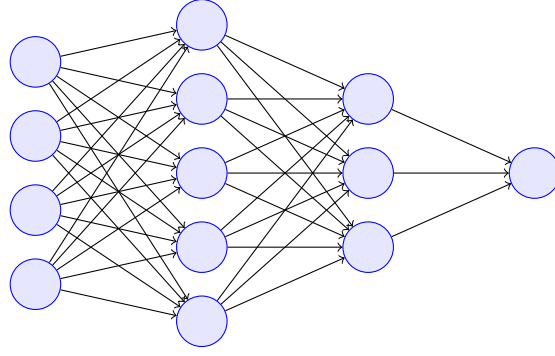


Figure 2: Neural network illustration.

(adapted from: https://tikz.net/neural_networks/)

is dispersed towards all the inputs of the neurons in the following layer. When training the neural network, we calculate the gradient and use stochastic gradient descent (SGD). According to SGD, all weights are updated during **backpropagation**, this being controlled by the η learning rate:

$$\theta_k = \theta_k - \eta \frac{\partial E}{\partial \theta_k}.$$

Starting from the loss function, the partial derivatives are calculated from the output layer towards the input layer.

Convolutional neural networks (CNN) are neural networks [4]. It has fewer parameters, but it is more resilient when faced with distorted data, using locally sensitive, orientation-selective units.

CNNs are built from **convolutional layers**. Convolutional layers planes of neurons. An output of a plane is a **feature map**, thus the output of a convolutional layer is a set of feature maps. The convolutional layers convolved their input. The output of a neuron plane is a **feature map**, the output of a convolutional layer is a set of feature maps. Convolution is the shift of a **kernel** across an image when dealing with 2D data.

$$S(i, j) \doteq (I * K)(i, j) \doteq \sum_m \sum_n I(i, j) K(i - m, j - n), \quad (2)$$

where I - input image, K - kernel, S - output image, and $*$ - convolution operator. The kernel is a small, learnable matrix.

After the convolutional layers, a local subsampling and nonlinearity operation is performed to reduce the resolution. In addition to the layers' weights, there is a bias for controlling the effect of nonlinearity. Layers performing convolution and subsampling result in a hierarchical structure in which, with decreases of resolution, the number of feature maps increases. Most of

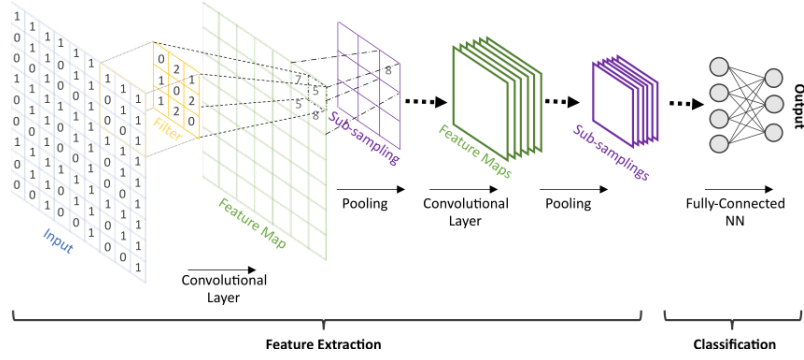


Figure 3: Sketch of a CNN, presenting the convolution using a kernel, the pooling operation and the succession of layers followed by a fully connected layer, providing the probability distribution of the input. (Source: [2])

the time, it ends with a fully connected layer, after which a softmax nonlinearity is placed for classification.

1.3 Explainability

In their paper, Gilpin et al. [3], highlighted that explainability is becoming a crucial concept in the domain of ML. It allows us to get more transparent decisions and to gain insight into the models decision-making process. With it can be observed biases of the system or the dataset, and by it is easier to confirm the correctness of the models working. The authors emphasised the difference between explainability and interpretability, the latter meaning that the model's inner states are comprehensible for humans, and the link between causes and effects. Explainability, on the other hand, is the characteristic of a system that is capable of providing explanations; it is the information about the causalities of a system. Creating an explanation is automatic. In post-hoc models, the reasoning behind an output is independent of the training procedure of the model. We consider models with constituent parts responsible for providing an explanation; based on evidence, this one is less erroneous and more efficient.

Explainable models often use prototypes as structural parts of the ML system. The prototypes could be imagined as the centroid of an information cloud: a sample for a cluster. They are also learnable parameters that are updated during training. The prototypes could be projected back into the input space, thus conveying the explainability to a model. For example, in the case of a system responsible for image classification, prototypes are visualised, so it is interpretable for beholders how the parts of the network look which contributed towards the taken decision. Prototypes ensure a system without any pos-hoc analysis and are also transparent regarding its decisions.

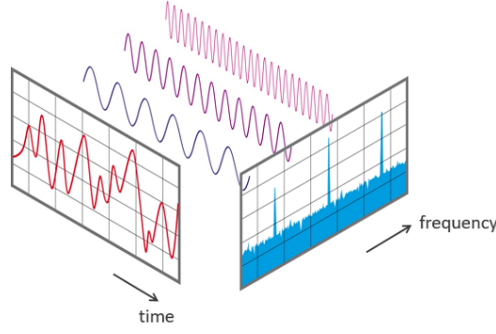


Figure 4: Fourier transform of a function (red, on the left): it is decomposed in the frequency domain to its 3 constituent sinusoidal components (purple)

(Source: <https://i.stack.imgur.com/D88Xo.png>)

2 Wavelets

In the following, we introduce basic concepts that are important to understand the mechanism of how wavelets work and why they are useful for feature extraction from images. What we cover in the subsequent section is based on “*A Wavelet Tour of Signal Processing*” by Mallat [9].

Wavelets have been used successfully in signal processing for a long time. Images could be imagined as 2D signals, that is where the idea of crossing the two comes from. Regarding the processing of signals, it is of great ease to use basis functions that are space- and frequency-localised.

In a study by Olshausen and Field [10], some notable similarities are observed between the feature maps of the convolutional layer and the actual neural maps in the animal cortex. Wavelets also have significantly fewer parameters, thus reducing memory and time consumption.

2.1 Fourier-transform

The wavelet transform is developed using the idea behind the Fourier transform. Fourier transform is used in function analysis for decomposing a function into constituent frequencies, the result being a time-invariant representation of the function in the frequency domain. Fourier transform is a linear operation that decomposes a function $f(x)$ into the sum of sinusoidal waves of the form $e^{i\omega x}$. The Fourier transform of a function $f(x)$ is defined as:

$$\hat{f}(\omega) = \int_{-\infty}^{+\infty} f(x)e^{-i\omega x} dx \quad (3)$$

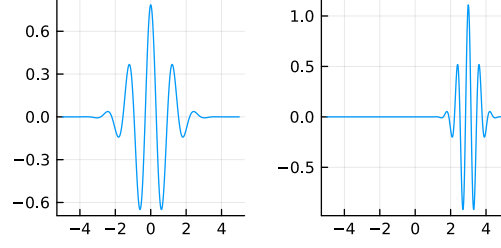


Figure 5: left: Morlet-wavelet; right: translated and scaled Morlet-wavelet

The inverse of Fourier transform is:

$$f(\omega) = \frac{1}{2\pi} \int_{-\infty}^{+\infty} \hat{f}(x) e^{i\omega x} dx \quad (4)$$

The result of Fourier decomposition in the case of discrete signals is in orthogonal Fourier basis. Although the Fourier transform is not time-localised, it is a crucial feature for signal processing. Thus, it is inadequate for feature extraction. It is desirable to have local frequency components of images; that is why we prefer wavelets over Fourier transform.

2.2 Wavelets

The wavelets keep both time and frequency components of the functions. Wavelet decomposition is a linear transformation in time and frequency, which has wavelet coefficients as a result. They could not be changed; wavelet parameters do not need to be learnt.

A collection of wavelets used for filtering is created starting from a mother wavelet ψ :

$$\bar{\psi}_j(u) = \frac{1}{\sqrt{2^j}} \psi\left(\frac{-u}{2^j}\right),$$

for $j, u \in \mathbb{Z}$. Applying scaling and translation of that mother function, we get a wavelet family.

Each wavelet has the following properties:

- wavelets have unit norm,
- centered at 0,
- having a 0 average, $\int_{-\infty}^{+\infty} \psi(t) dt = 0$.

Given a function f , which is a one-dimensional signal, its decomposition into wavelet coefficients is:

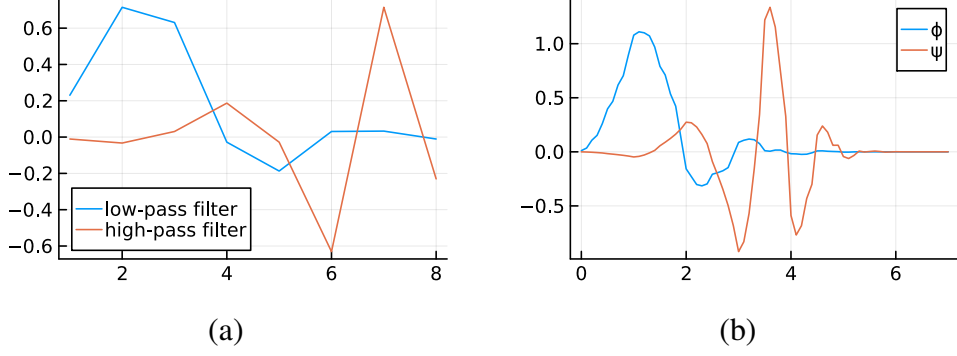


Figure 6: **(a)** Daubechies4-wavelets conjugate mirror filters; **(b)** (ψ) Daubechies4-wavelet and (ϕ) , resulting from 7 and 8

$$Wf(u, s) = \langle f, \psi_{u,s} \rangle = \int_{-\infty}^{+\infty} f(t) \frac{1}{\sqrt{s}} \psi^* \left(\frac{t-u}{s} \right). \quad (5)$$

The equation above is the expression of an inner product signaling how similar the original function and the wavelet are. The wavelet scans the entire span of the function. When working with convolution, we can use the translation-invariant form of it, where the u is fixed - the translation is performed by the convolution.

For a description of wavelets, we cover the notion of conjugate mirror filters. These are obtained from a high-pass filter h , which results from a low-pass filter l .

$$h[n] = (-1)^{1-n} l[1-n]. \quad (6)$$

We introduce a scaling function ϕ with the aim of covering all frequencies. ϕ is defined by the l low-pass filter.

$$\phi_k(t) = \sqrt{2} \sum_{-\infty}^{\infty} l[n] \phi_{k-1}(2t - n). \quad (7)$$

With increasing parameter k , the scaling function becomes more and more accurate.

The final wavelet is the result of a combination of a high-pass filter and a scaling function.

$$\psi(t) = \sqrt{2} \sum_{-\infty}^{\infty} h[n] \phi(2t - n). \quad (8)$$

When a certain function is analysed using a wavelet with the aim of decomposing it, the operation is done using the inner product of the scaling functions and the wavelet. The name of the method we used in implementing our model is called fast wavelet transform, which means

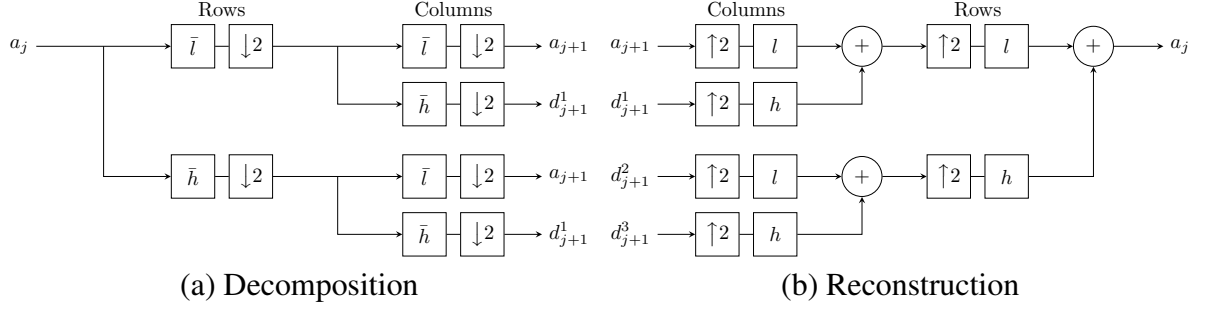


Figure 7: Fast two-dimensional wavelet transform. **(a)** a_j , input 2D signal, is decomposed through 1D convolution with conjugate mirror filters and subsampling. **(b)** Reconstruction based on a_{j+1} and d_{j+1}^k : upsampling by doubling rows and columns and inverse filtering.

decomposition using high- and low-pass filters.

1. Decomposition of function with low- and high-pass filter, with 2 downsampling by each step:

$$a_{j+1}[p] = \sum_{-\infty}^{\infty} l[n - 2p]a_j[n] = a_j \star \bar{l}[2p] \quad (9)$$

$$d_{j+1}[p] = \sum_{-\infty}^{\infty} h[n - 2p]a_j[n] = a_j \star \bar{h}[2p], \quad (10)$$

where for $\bar{l}$ and $\bar{h}$: $\bar{x}[n] = x[-n]$.

2. Further decomposition of the two results until the target level of decomposition.

Reconstruction is done through recursively applying upsampling.

$$a_j[p] = \sum_{-\infty}^{\infty} l[p - 2n]a_{j+1}[n] + \sum_{-\infty}^{\infty} h[p - 2n]d_{j+1}[n]. \quad (11)$$

Images are 2D signals; therefore, to analyse them, we need two-dimensional wavelets. Wavelets in higher dimensions resemble 1D wavelets.

$$\psi^1(x) = \phi(x_1)\psi(x_2), \psi^2(x) = \psi(x_1)\phi(x_2), \psi^3(x) = \psi(x_1)\psi(x_2), \quad (12)$$

where $x = (x_1, x_2)$ is the two-dimensional signal.

Scaling is achieved based on the below formula.

$$\left\{ \phi_{j,n}^2(x) = \phi_{j,n_1}(x_1)\phi_{j,n_2}(x_2) = \frac{1}{2^j} \phi\left(\frac{x_1 - 2^j n_1}{2^j}\right) \phi\left(\frac{x_2 - 2^j n_2}{2^j}\right) \right\}_{n \in \mathbb{Z}^2} \quad (13)$$

Using the above equations, image-decomposition with wavelet transform is done as:

$$a_j[n] = \langle f, \phi_{j,n}^2 \rangle \quad d_j^k[n] = \langle f, \psi_{j,n}^k \rangle, \quad 1 \leq k \leq 3$$

Two-dimensional wavelet filters are the result of a scaling function and wavelet products. The wavelet transform can be described as convolving the signal with low- and high-pass filters recursively.

$$a_{j+1}[n] = a_j \star \bar{l}l[2n] \quad d_{j+1}^2[n] = a_j \star \bar{h}l[2n] \quad d_{j+1}^1[n] = a_j \star \bar{l}h[2n] \quad d_{j+1}^3[n] = a_j \star \bar{h}h[2n] \quad (14)$$

The procedure is as follows: the filters are first applied to each row of the image. After this comes a downsampling. The results of downsampling are convolved column-wise, followed by another downsampling. This is one iteration, which could be repeated until the desired decomposition level is achieved. The reconstruction of an image is done in the same fashion with execution of the above operations in reverse order: upsampling, then convolution.

3 Wavelet-based Prototype Network - waveProtoPNet

In the following section, the novel architecture is presented. We call this network WaveProtoPNet, as it builds on the structural strengths of ProtoPnet [1] and adopts the efficient analysis capabilities of wavelets. We present the implementation details and how the actual decomposition process looks like.

3.1 Prototype-based architecture

The WaveProtoPNet model is based on the ideas presented and tested in the paper that describe ProtoTypeDL [8] and which were further developed into ProtoPnet [1]. However, the former had serious shortcomings, as there was no option for prototypes smaller than the input image size, thus the system remained in most cases agnostic towards small but relevant details. In an attempt to solve this problem, the authors of ProtoPnet introduced a prototype layer consisting of prototype units smaller than the input image size. That adaptation we also use in our network.

3.2 Structural details

With respect to the structure of the network, three constituting parts could be described. The first is a wavelet decomposition (f), which has the duty of extracting features from its input. After this comes the layer of prototypes (g_p). Finally, the data flow through a fully-connected layer ($h - w_h$ weights), which is responsible for the classification probabilities.

Information flows through these structural parts as follows: for an image x from the $D = \{(x_i, y_i)\}$ dataset, where $x_i \in \mathbb{R}^p$, $y_i \in \{1, \dots, K\}$ for each $i \in \{1, \dots, n\}$ we receive from the wavelet transform the $f(x)$ feature maps in the form of $H \times W \times D$. Thus, the wavelet transform represents a function $f : \mathbb{R}^p \rightarrow \mathbb{R}^q$. The output of this function is transferred to the prototype layer being rearranged before: the $H \times W \times D$ data is transformed into $H^* \times W \times D$ shaped matrix because this form enhances the comparison between prototypes and prototype-sized patches (Figure 10). We shall discuss in detail the data transformation that takes place there. The prototype outputs are fed into the fully-connected layer, mapping $w : \mathbb{R}^m \rightarrow \mathbb{R}^K$, and finally the softmax function $s : \mathbb{R}^K \rightarrow \mathbb{R}^K$.

The prototype layer is formed by the prototype vectors m : $P = \{p_j\}_{j=1}^m$. Each of these vectors $j \in \{1, \dots, m\}$, $m \geq K$, p_j is of $H^* \times W \times D$ form. The procedure of learning the prototypes

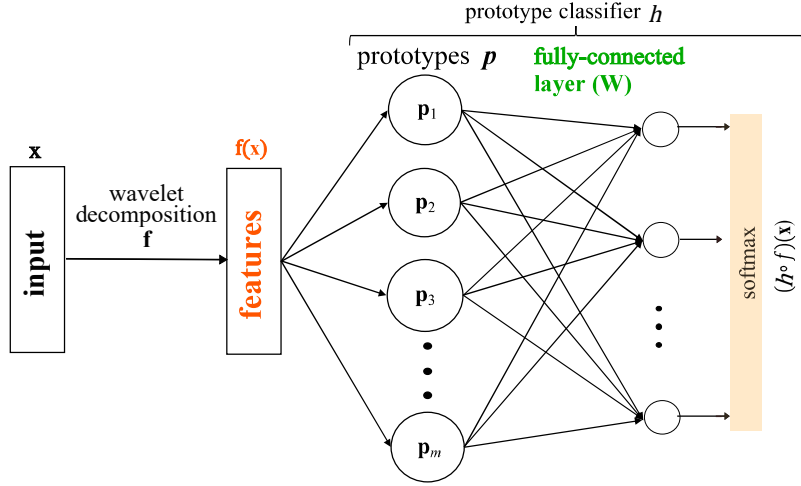


Figure 8: The structure of the WaveProtoPNet.

takes place during the training of the whole network. The prototypes learn information from the feature maps. As a prototype in our case does not represent whole images, but rather just a part of it, we are going to call the region of an image learned by a prototype a patch. In a prototype layer, there are sets of prototypes for each class of the input set, each class having more prototypes to capture the detail complexity of the input data.

Regarding the function of a prototype layer unit, given a feature map $f(x) = z$, the j^{th} prototype computes a squared distance between every patch of the z of equal size to the prototype and the prototype itself. The output of prototypes could be thought of as activation maps, in which each point corresponds to a patch on the original image; the value of a point on this activation map stands for the distance between its corresponding image patch and the prototype. In each prototype output, a global max pooling is applied before being passed onto the fully-connected layer, i.e., the global output of the prototype layer is a vector with length m . The intuition behind this pooling-base reduction is that we focus only on the fact of how close a prototype is to any patch of an image, which means how useful that prototype is for classifying.

For any $f(x) = z$ the output of the j^{th} prototype is

$$g_{p_j}(z) = \max_{\tilde{z} \in \text{patches}(z)} \log \left(\left(\|\tilde{z} - p_j\|_2^2 + 1 \right) / \left(\|\tilde{z} - p_j\|_2^2 + \epsilon \right) \right).$$

This being a monotonically decreasing function with respect to the distance between the prototype and the patch, it is straightforward to invert the results: the larger the value, the closer the patch

is to the prototype.

The role of the softmax layer following the fully-connected layer is to normalize the outputs of the latter. In this way, we get a probability distribution over the dataset classes.

3.3 Training

WaveProtoPNet training is carried out in two steps. First, a prototype SGD is performed, after which comes the training of the fully-connected layer. The aim of the first stage is to have prototypes that learned the most important features from the input images. The prototypes' formation occurs while the fully connected layers' weights are frozen. After this comes the phase where the whole system is trained in an end-to-end fashion assuring a fine-tuning of separate parts. For the SGD a loss function is minimized, consisting of three constituent parts.

The first term is *Cross-entropy loss* ($CrsEnt$). Its role is to measure the accuracy of the classification given the training labels. The next term is responsible for the closeness of the feature vectors to at least one prototype of its own class. It is called *cluster cost* ($Clst$). It is important to have prototypes that represent various parts of the input images. The third part is *separation cost* (Sep), which is responsible for the separation of prototypes of different classes. In this way, prototypes of the same class easily attract images from the corresponding class, and it is less prone for an image to be associated with a prototype that does not represent its own class.

For $D = \{(x_i, y_i)\} = [X, Y]$ being a set of labeled images for training, the final loss function is composed from:

$$Clst = \frac{1}{n} \sum_{i=1}^n \min_{k:p_j \in P_{y_i}} \min_{z \in patches(f(x_i))} \|z - p_k\|_2^2, \quad (15)$$

$$Sep = -\frac{1}{n} \sum_{i=1}^n \min_{k:p_j \notin P_{y_i}} \min_{z \in patches(f(x_i))} \|z - p_k\|_2^2, \quad (16)$$

The combination of which yields the final loss function:

$$E(h \circ f, D) = CrsEnt(h \circ g_p \circ f, D) + \lambda_1 Clst + \lambda_2 Sep. \quad (17)$$

The first stage of training is with the loss function described above. Afterwards, Initialisation of the weights of the fully connected layer is of crucial importance to attain explainability. For

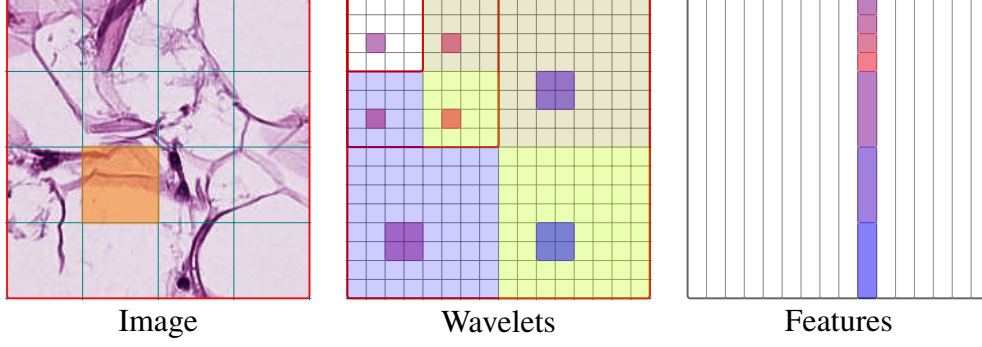


Figure 9: **(left)** The original image with the observed patch. **(mid)** Highlighted: the information belonging to the observed patch on the dwt-decomposed image. **(right)** Information collected in a feature vector.

any edge connecting the prototype j^{th} to the output unit k^{th} , we have a $w^{k,j}$ attributed. The weights are initialised as $w^{k,j} = 1$ if $p_j \in P$, in all other cases -0.5 . The intuition behind this is to fine-tune the system so that if a prototype is close to an image, a higher probability is achieved, a larger distance between prototype and image being associated with lower probability value. The output we get from the fully connected layer is a weighted sum of all prototypes' similarity scores. If an image of class x resembles a prototype of class y , then the similarity score is a highly negative number; on the other hand, the similarity values obtained for the same image with prototypes of class x will add up to a larger positive value. In this way, prototypes tend to learn only the features of their classes. These weights are then optimised using convex optimization. Our target is to bring the initially negative values as close to zero as possible to avoid negative reasoning. This is achieved by altering the loss function with the aim of reducing cross-entropy and reducing the absolute value of negative weights.

$$E(h \circ f, D) = \frac{1}{n} \sum_{i=1}^n CrsEnt(h \circ g_p \circ f, y_i) + \lambda \sum_{k=1}^K \sum_{j: p_j \notin P_k} |w_h^{(k,j)}| \quad (18)$$

3.4 Wavelets on GPU

The language of implementation of the ML network was `Julia`. There is no GPU-compatible wavelet implementation available in `Julia`, therefore we had to write the code ourselves from zero. We created a new package with the necessary functions as well.

As mentioned in Section 2, the discrete wavelet transform can be thought of as a convolution, with the use of four filters. These are transformed into four two-dimensional filters corresponding to the average (low-low) and three detail components (low-high, high-low, and high-high) of the

analyzed image. The gpu-compatible wavelet architecture has two main structural parts:

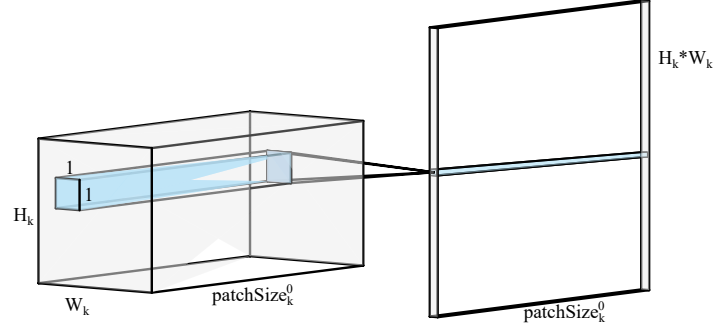
1. *Filtering* – Using the predefined 2-dimensional filters, executes the convolution with stride of 2, which results in downsampling of the image;
2. *Rearrangement* – the four decomposed parts are rearranged. This is done to achieve a form of the result needed at the last level. This is done on each level by convolution. For ease of imagination, given a part of the image $2 \times 2 \times d$ (where d is the depth of the image), this rearrangement operation places its $1 \times 1 \times d$ parts below each other, obtaining the result of a shape $1 \times 1 \times 4 * d$.

If the *overlap* flag is on, the result is rearranged once more. We will cover the details of this case in the following section.

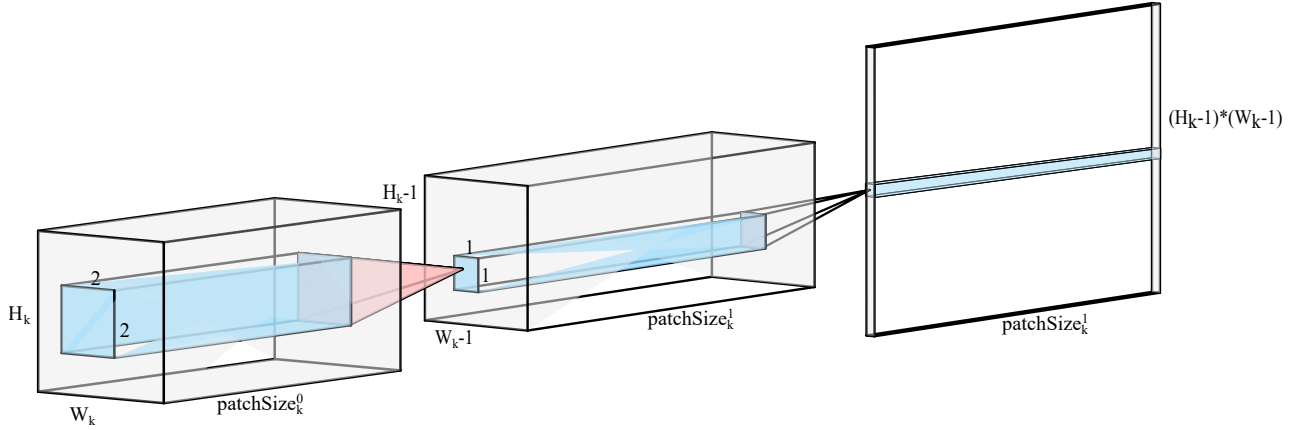
3.5 Decomposition level and overlap

The level of decomposition and overlap are modifiable parameters of the model, which greatly influence the output of the system. The *decomposition level* is determining how many times the decomposition operation is repeated; it functions like a recursion: the image in the first iteration, then the output of previous decomposition is decomposed in each phase until the desired decomposition level is attained. The decomposition level directly influences the size of prototypes and the number of patches in which an image is split. We present this with mathematical notation on which the processing of images is built. Let $k \in \mathbb{N}$ be the number of decomposition levels. If $k = 0$, no decomposition takes place, and each pixel represents a separate patch, the operation resulting in as many patches as there were pixels in the original image. With higher values of k , the level of decomposition deepens and the sizes of the patches will follow; a patch will be four times larger than at the previous level: $patchSize_k = patchSize_{k-1} * 4$. The size of a single patch at the k^{th} level can be calculated as follows: if there is no overlap set, $patchSize_k^0 = (2^k)^2 * C$, where C stands for the number of channels in the input image. Therefore, the decomposition without overlap at the level k results in a $W_k \times H_k \times patchSize_k^0$ tensor, with W_k and H_k defined as $W_k = (W/(2^k))$, $H_k = (H/(2^k))$, where W and H represent the width and height of the original images. The number of resulting patches will be: $nr_{patch} = W_k * H_k$.

In case *overlap* is applied alongside decomposition, the receptive field widens, making it possible to focus on more details of an image. The core of a patch contains the most information;



(a) without overlap;



(b) with overlap

Figure 10: Patch extraction with and without overlap. The decomposition output at the level k has a size of $W_k \times H_k \times patchSize_k$. **(a)** A patch is represented as a $1 \times 1 \times patchSize_k$ part. This results in a $W_k * H_k \times patchSize_k$ -shaped output. **(b)** A patch is a $2 \times 2 \times patchSize_k$ part, extracted with overlap. This creates a $(W_k - 1) \times (H_k - 1) \times patchSize_k * 4$ tensor, from which patches are extracted with the same method as without overlap, obtaining a result with shape $(W_k - 1) * (H_k - 1) \times patchSize_k * 4$.

it is the area in the center that is the most representative of that patch. These center areas are not independent of each other when overlap is applied, as in this case, a patch contains half of the patches from all four neighbors. The actual operation that executes the overlap takes place after decomposition (the same decomposition procedure as without overlap), which is illustrated in Figure 10. On the result of the decomposition (shape: $W_k \times H_k \times patchSize_k^0$), a patch is a region $2 \times 2 \times patchSize_k^0$. The extraction works with $stride = 1$, producing half-size overlap and creating patches four times larger than without overlap: $patchSize_k^1 = patchSize_k^0 * 4$. After this, the resulting $(W_k - 1) \times (H_k - 1) \times patchSize_k^1$ -shaped matrix patches are extracted as $1 \times 1 \times patchSize_k^1$ parts, resulting in $(W_k - 1) * (H_k - 1)$ patches.

Considering overlap, the final formulas are the following:

$$patchSize_k = (2^k)^2 * C * (4^{overlap}), \quad (19)$$

$$nr_{patch} = (W_k - overlap) * (H_k - overlap), \quad (20)$$

where C is the number of channels.

3.6 Receptive field

The core of a patch is a patch's most representative part; that $l \times l$ region of the original image that is processed at each level of decomposition, where l is the length of the low-/high-pass filter of the wavelet.

The *receptive field* of a patch is the region of the original image from which the patch contains information. We start our calculations without overlap, and in the end we will deduce from this the equations that also include overlap too.

- $k = 0$ decomposition level – a pixel's receptive field is itself: a 1×1 region of the image.
- when $k > 0$, considering the filter length l and the downsampling, the $[i, j]$ th patch from the decomposed image and its receptive field from the previous level:

$$[i, j]_k \rightarrow [(2i - 1) : (2i - 2 + l), (2j - 1) : (2j - 2 + l)]_{k-1}$$

Calculating it, level by level, and considering only the first parameter because the second

looks similar, the receptive fields and their sizes in the original image will be the following:

$$\begin{array}{llll}
[i, _]_0^0 \rightarrow & [i : i, _]_0 & \Rightarrow & 1 \times 1 \\
[i, _]_1^0 \rightarrow & [(2i - 1) : (2i - 2 + l), _]_0 & \Rightarrow & l \times l \\
[i, _]_2^0 \rightarrow & [4i - 3 : 4i - 6 + 3 * l), _]_0 & \Rightarrow & (3 * l - 2) \times (3 * l - 2) \\
\vdots \rightarrow & \vdots & \Rightarrow & \vdots \\
[i, _]_k^0 \rightarrow & [2^k(i - 1) + 1 : 2^k(i - 2 + l) + 2 - l, _]_0 & \Rightarrow & (2^k(l - 1) - l + 2) \times (2^k(l - 1) - l + 2)
\end{array}$$

With overlap, a slight modification is required. At the last level of decomposition, the information is gathered in a 2×2 shape. The information belonging to a patch $[i, _]_k^1$ at level k is with overlap and spans from the beginning of the $[i, _]_k^0$ till the end of the $[i + 1, _]_k^0$:

$$\begin{aligned}
[i, _]_k^1 &= [i : i + 1, _]_k^0 \\
&= [2^k(i - 1) + 1 : 2^k(i - 1 + l) + 2 - l, _]_0 \\
&\Rightarrow (2^k l - l + 2) \times (2^k l - l + 2)
\end{aligned}$$

The receptive field, based on decomposition level k , with filter length l , and overlap flag *overlap*, has a shape of:

$$(2^k(l - \neg overlap) - l + 2) \times (2^k(l - \neg overlap) - l + 2). \quad (21)$$

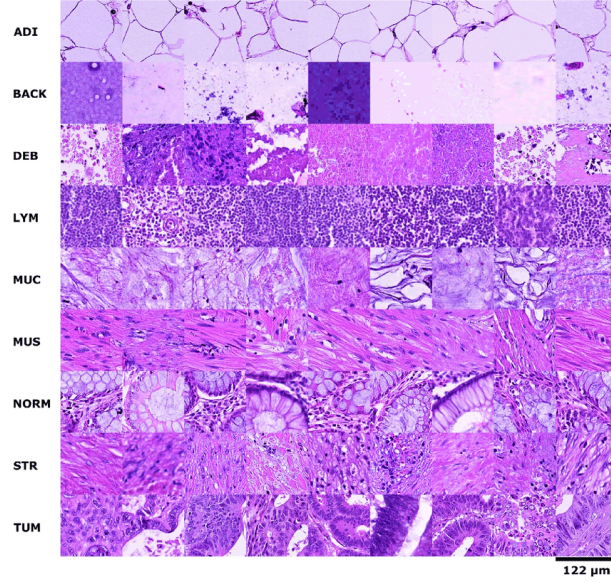


Figure 11: Images from the nine classes of NCT-CRC-HE-100K dataset.

(Source: [7])

4 Experiments and results

In the next section of this paper, our experiments are described in detail. We investigated the performance of waveProtoPNet on a medical database with various tissue samples. We cover the different parameter settings, prototypes, and wavelets used, and the results of these model configurations.

4.1 Dataset

The NCT-CRC-HE-100K dataset [6] contains 100000 images of human colorectal cancer (CRC) and normal tissue. RGB images of size 224×224 come from nine classes, namely: adipose (ADI), background (BACK), debris (DEB), lymphocytes (LYM), mucus (MUC), smooth muscle (MUS), normal colon mucosa (NORM), cancer-associated stroma (STR), colorectal adenocarcinoma epithelium (TUM).

In a study [7], in which different CNNs were tested during training (VGG19, AlexNet, SqueezeNet, GoogLeNet, Resnet50), an accuracy of 98.7% was achieved in the internal test set (NCT-CRC-HE-100K) and 94.3% in the external test set (NCT-CRC-HE-7K). The NCT-CRC-HE-7K dataset contains 7180 image patches, without overlap with the original NCT-CRC-HE-100K dataset. The images are the same size.

Table 1: Number of parameters in wavelet-decomposition layer, by varying decomposition level, overlap, and wavelet.

Level	Overlap	ParamNr	Level	Overlap	ParamNr
1	0	48	1	0	192
1	1	240	1	1	384
2	0	192	2	0	336
2	1	960	2	1	1104
3	0	912	3	0	1056
3	1	3984	3	1	4128

(a) Daubechies 1
(b) Daubechies 2

4.2 Training parameters

Training has two main phases: one with the first type of loss, concentrating on correct formation of prototypes, keeping the fully connected layer frozen, and a following phase with the second type of loss, focusing on the cross-entropy loss and on the fully connected layer’s weights. Parameters such as wavelet, decomposition level, overlap, and prototype number could be modified, depending on the desired test scenarios. During the experiments, Adam optimiser and the following loss parameters were used: $\lambda_1 = 0.9$, $\lambda_2 = 0.4$, $\lambda = 0.001$.

With the first type of loss, we run the system for 4 epochs, with a 0.01 learning rate, then for other 4 epochs, with a smaller 0.005 learning rate. After that follows the thawing of the fully-connected layer, and the training of the full model, with the second type of loss. It starts with a learning rate of 0.01 for 2, then of 0.005 for 3 epochs, and finally switches to a learning rate of 0.001 for the rest of 6 epochs.

4.3 Backbone size

The waveProtoPNet differs from ProtoPNet in the way information processing and the extraction of features are performed. Its strength lies in the adoption of wavelets for replacing the traditional feature extraction based on sequence of convolutional layers. The reduction of trainable parameters is considerable. We got rid of the whole backbone, the layer containing the prototypes and the fully connected layer remain basically the same from the aspect of memory consumption.

Feature extraction in ProtoPNet was performed through well-known architectures, such as VGG, ResNet, or AlexNet. The number of parameters are the following in various backbones used in the original paper: for VGG16 – 138 million parameters, for VGG-19 – 144 million

Table 2: Accuracies with different porotype number per class, working with Daubechies2 wavelet,decomposition level of 2, without overlap.

Proto/class	Train accuracy	Test accuracy
15	88	87
25	88	87
35	90	89
45	91	90
55	91	89
60	91	90
75	91	90

parameters, for ResNet-34 – 63.5 million parameters, for AlexNet and ResNet-152 – 60 million parameters, DenseNet121 – 8 million parameters.

In comparison with the above numbers, the size of the parameters is magnitudes lower in the wavelet-based model, as we summarise our experiments in Table 1. It is obvious that there is no need for more parameters than 5000 which is 1600 times less than what the smallest backbone requires to attain above 90% accuracy.

4.4 Number of prototypes

We examine the optimal number of prototypes to be used for classification. Based on previous experiments, we concluded that the model performs well and is stable, its results are reproducible, when wavelet decomposition is performed using the Daubechies 2 wavelet, with a decomposition level 2, without overlap. This was the setup for our experiments to fine-tune the number of prototypes. From Table 2, it can be observed that with fewer prototypes, the classification accuracy is relatively low. Increasing the number of prototypes, the accuracy of the model also increases to a certain point. When there were 45 prototypes per class, the precision increased above 90%. From then on, with an increasing prototype number, the variation in precision became inconsiderable. It is noticeable that even with a larger prototype number, the accuracy did not decrease, which means that the model is not overfitting. For the rest of the experiments, we worked with 45 prototypes per class, as it is the best trade-off between memory usage and accuracy.

Table 3: Classification accuracy for of different decomposition levels and overlap, using 45 prototype per class and Daubechies2 wavelet. The size of the prototype is calculated from Equation (19).

ProtoSize	Level	Overlap	Train	Test
12	1	0	89	88
48	1	1	93	92
48	2	0	91	89
192	2	1	93	92
192	3	0	91	89
768	3	1	81	80

4.5 Decomposition level and Overlap

In Table 3 can be seen the results of our experiments with different levels of decomposition. Based on the conclusion of the previous subsection, we run the system with the following setup: Daubechies2 wavelet and 45 prototypes per each class of the dataset. At each level of decomposition, settings with and without overlap were tested.

The size of prototypes is directly influenced by the level of decomposition, the overlap flag, and the size of the images. On the first and second levels of decomposition, better results were achieved when overlap was applied. The reason behind this is that with overlap, more patches of the same image are inspected and compared to the prototypes, thus increasing the chance of observing significant features, which help to increase the classification accuracy even by 4%. However, in the case of level 3, the overlap did not imply better results; in fact, it meant a decrease in accuracy of 9%. This is due to the larger size of the prototype – the lack of ability to learn the prototypes with so many parameters.

The best results were achieved on the first and second levels of decomposition, in both cases with overlap applied, the precision being 92% on the test dataset.

4.6 Performance with different wavelets

The extraction of features is performed with the aid of wavelets; therefore, it is of utmost importance to test as many wavelets for image analysis as possible. We conducted our experiments mostly on the Daubechies wavelet family. But we also tried other wavelets as well: Coiflet and Symlet. We summarise the wavelets and their accuracies obtained in Table 4.

Table 4: Daubechies, Coiflet, Symlet wavelets, with decomposition level 2, overlap and 45 prototype per class.

Wavelet	Tran	Test
db1	94	92
db2	94	92
db3	93	92
db4	92	91
coif2	92	91
sym4	90	87

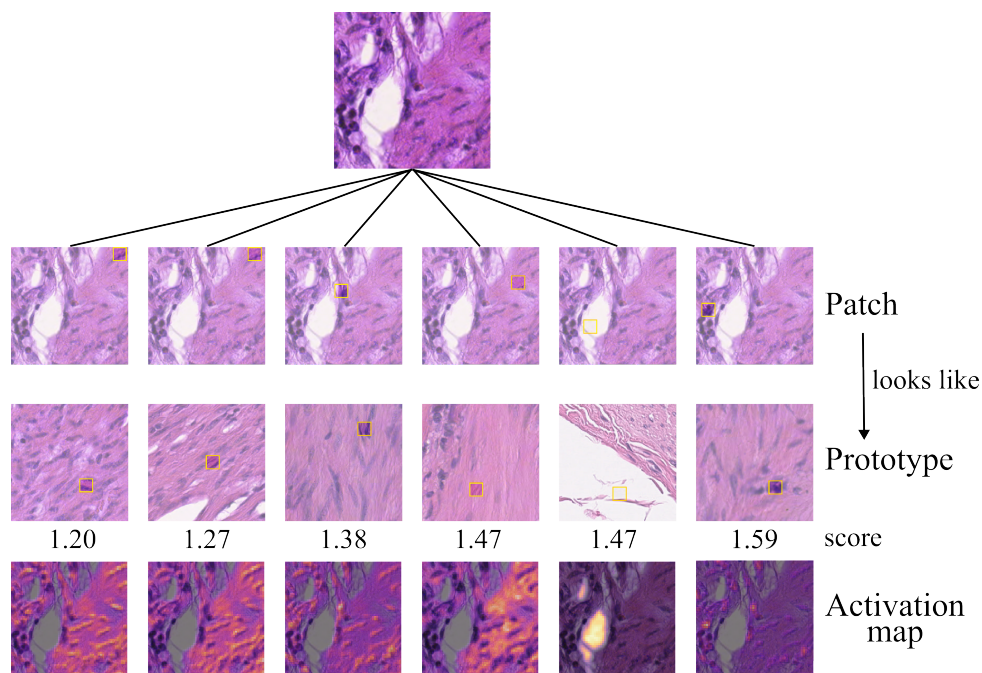


Figure 12: Generated explanation of an image (top of the figure). The first row highlights the patches which were closest to a prototype (second row). In the third row, there are the activation maps of the 6 closest prototypes projected onto the original image from smooth muscle class (MUS). The activation score is placed between the prototype and the activation map.

4.7 Prototype-check

In this subsection, we present the explainability of our model and the actual mechanism by which the predictions are explained. In case of each image from the test dataset, there is a possibility to ask for an explanation from the WaveProtoPNet. Suppose that we have an image that belongs to the mucus class of our tissue samples dataset. If this image is classified as belonging to the mucus class, then the six closest prototypes representing some prototypical parts of the mucus class are selected. For each prototype closest to the image, an activation map is generated. This activation map signifies how similar a region on the original image is to that prototype. These values are then projected back into the input space onto the original image. This is an empirical way of assessing whether the prototype identified as important for classification resembles a truly important region or provides a false explanation. In addition to the activation map, there is an activation value for each prototype. This is calculated by multiplying the similarity score by the appropriate weight in the fully connected layer. The similarity score is obtained as a result of the comparison between the prototype and the feature map of the image during the flow of data through the model. The weight by which this is multiplied is on the edge connecting the actual prototype to the output which is predicted by the model. This weight represents the importance of that peculiar prototype in identifying an image of that certain class. The similarity value with the activation map provides an explanation for the actual prediction.

4.8 Misclassification

In order to examine in which cases the model classifies incorrectly, its predictions were examined for each class. In Figure 13 we can see for each class (vertical axis) what the prediction accuracy was for it. There are some classes for which the prediction is really accurate (adipose, background, lymphocytes), but we can observe regular misclassifications as well. As an example, the normal colon mucosa (NORM) class tends to be classified as mucus or as the colorectal adenocarcinoma epithelium (TUM). It is interesting that it does not necessarily work vice versa: the debris (DEB) class is often classified as cancer-associated stroma (STR), while the cancer-associated stroma is rather misclassified as smooth muscle (MUS).

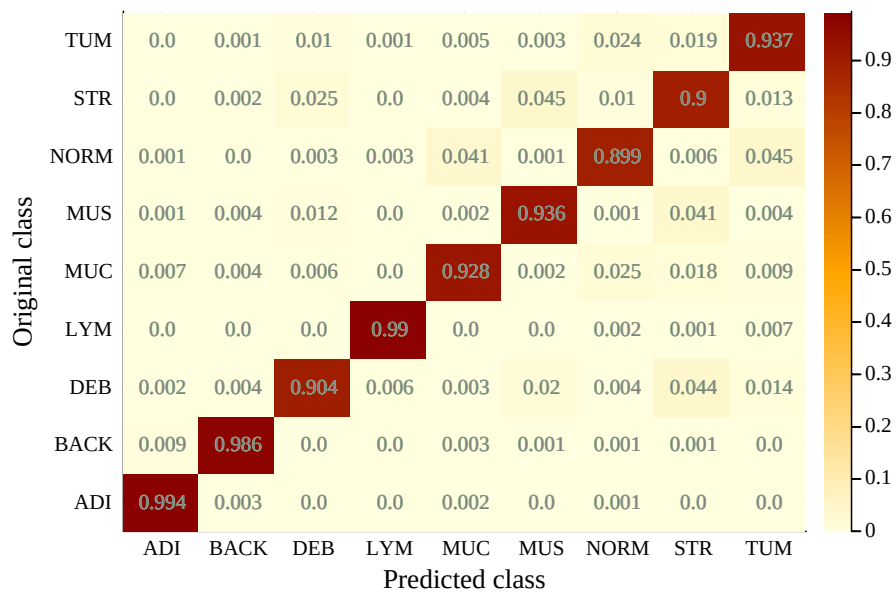


Figure 13: Misclassification over classes. On x-axis there are the predicted classes of pictures belonging to classes specified on y-axis. The values on a row are the accuracies by which images of certain classes were predicted to belong.

Conclusions and future work

Wavelet-based feature extraction ideally substitutes for traditional convolution-based architecture in the case of the image classification model. Combining wavelets with prototypes created a powerful and robust model that produces interpretable predictions.

Significant achievements were obtained with the novel WaveProtoPNet architecture in terms of prediction accuracy, number of trainable prototypes, and explainability. The whole system is scalable and runs on GPUs, which also reduces the time cost of training. With the replacement of the classical backbones with wavelet decomposition, the memory consumption was significantly reduced (min. 1600 times). The precision (94%-train, 92%-test) was on par with cutting-edge technologies when applied to the medical dataset on which we also conducted our research. Our system was able to predict with high confidence and in a reproducible manner the correct classes of tissue samples. It also generated an explanation for each prediction that offers valuable insight into the decision-making process to assess possible shortcomings or eventual biases that otherwise would go unnoticed.

However, there are still aspects of this model that need to be thoroughly researched and refined. The usefulness of the prototype set learnt by the network could be further investigated. The set of prototypes offers a satisfying global result; however, the usefulness of individual prototypes remains unclear. Based on these data, a pruning mechanism could be implemented with the aim of reducing the number of prototypes along with keeping the classification accuracy high. Another idea would be to implement segmentation to easily distinguish between image backgrounds and information-rich areas.

References

- [1] Chaofan Chen et al. “This looks like that: deep learning for interpretable image recognition”. In: *NIPS 33 Proceedings*. Curran Associates Inc., 2018.
- [2] Barbara Cunha et al. “A Review of Machine Learning Methods Applied to Structural Dynamics and Vibroacoustic”. In: *arXiv preprint arXiv:2204.06362* (Feb. 2022).
- [3] Leilani H Gilpin et al. “Explaining explanations: An overview of interpretability of machine learning”. In: *5th IEEE DSAA Conf*. IEEE, 2018, pages 80–89.
- [4] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.
- [5] Tiantian Guo et al. “A review of wavelet analysis and its applications: Challenges and opportunities”. In: *IEEE Access* 10 (2022), pages 58869–58903.
- [6] Jakob Nikolas Kather, Niels Halama, and Alexander Marx. *100,000 histological images of human colorectal cancer and healthy tissue*. Version v0.1. Zenodo, May 2018. DOI: [10.5281/zenodo.1214456](https://doi.org/10.5281/zenodo.1214456). URL: <https://doi.org/10.5281/zenodo.1214456>.
- [7] Jakob Nikolas Kather et al. “Predicting survival from colorectal cancer histology slides using deep learning: A retrospective multicenter study”. In: *PLoS medicine* 16.1 (2019), e1002730. DOI: [10.1371/journal.pmed.1002730](https://doi.org/10.1371/journal.pmed.1002730). URL: <https://doi.org/10.1371/journal.pmed.1002730>.
- [8] Oscar Li et al. “Deep Learning for Case-Based Reasoning Through Prototypes: A Neural Network That Explains Its Predictions”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 32 (2018).
- [9] Stéphane Mallat. *A Wavelet Tour of Signal Processing*. Elsevier, 2009, pages 102–110.
- [10] Bruno Olshausen and David Field. “Emergence of simple-cell receptive field properties by learning a sparse code for natural images”. In: *Nature* 381 (July 1996), pages 607–9.
- [11] Gheyath Othman and Diyar Qader Zeebaree. “The applications of discrete wavelet transform in image processing: A review”. In: *Journal of Soft Computing and Data Mining* 1.2 (2020), pages 31–43.