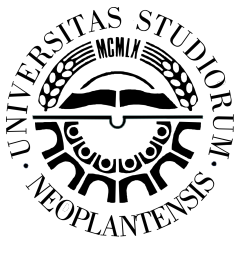
University of Novi Sad
Faculty of Sciences
Department of mathematics and informatics



Predrag Matavulj

Fusion of Heterogeneous Data in Convolutional Networks for Real-Time Pollen Particle Identification

Master thesis

Supervisor:

dr Sanja Brdar

2018, Novi Sad

Abstract

The aim of this thesis is to present a part of a solution for automated pollen particle identification. Pollen identification is an important problem considering the fact that there is an increasing number of people suffering from allergies and that pollen concentrations impact diverse biosystems including agriculture. Currently, the identification process is manual, takes a lot of time and requires human resources. New devices on the market create data sets relevant for the pollen identification and classification, but require development of predictive models based on machine learning. In this thesis we focused on four types of pollen (Brussonetia, Betula, Picea and Juglans) different in morphology and size and utilized convolutional neural network to learn classification model. The network learned features form scattering images (24 pixels x number of acquisitions), fluorescence spectrum (32 x 8 acquisitions separated by 500 ns) and fluorescence lifetime (64 x 4 acquisitions). After the last convolutional layer these features were concatenated into one feature vector representing all three sources of information, which allowed the gradient to flow through the whole network. The result of the classification is measured in accuracy, giving the best result of 84.9% for classification when all three sources are used. Furthermore, introducing probability threshold improves the accuracy score while discarding a number of samples.

Contents

LIST OF FIGURES

LIST OF TABLES

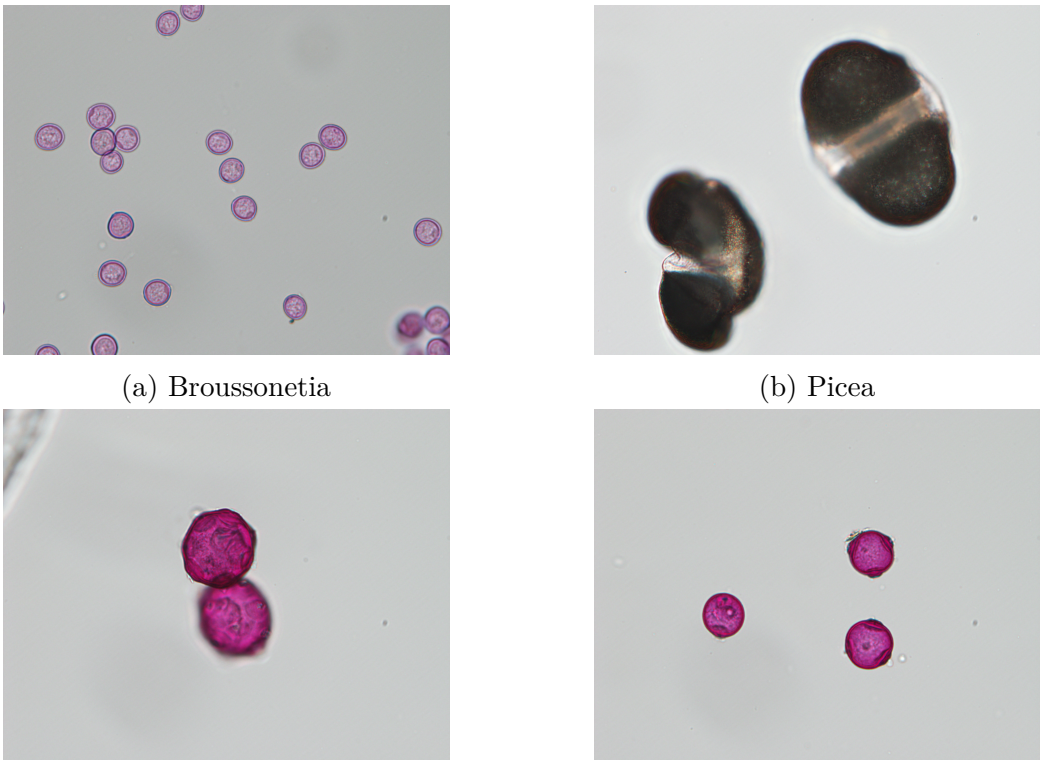
1 Introduction

An aerosol is a collection of liquid or solid particles suspended in a gas. There are many kinds of aerosols: atmospheric clouds of ice particles or water droplets, resuspended soil particles, photochemically formed particles, salt particles formed from ocean spray, etc. They include a wide range of phenomena such as mist, clouds, smog, dust and fume. They vary greatly in their ability to affect not only visibility and climate, but also our health and quality of life. An understanding of the properties of aerosols is of great practical importance. Particle size ranges from about 0.002 to more than 100 um. Aerosols of biological origin are called bioaerosols. They include viruses, viable organisms, such as bacteria and fungi, and products of organisms, such as fungal spores and pollen. The health effects caused by bioaerosols include infectious disease, sensitization reactions, such as asthma, and reactions to toxins or irritants [1].

Pollen grains are relatively large, near-spherical particles produced by plants to transfer genetic material to the flower of other plants of the same species. The grains range in size from 10 to 100 um, with most between 25 and 50 um. Wind-pollinated plants produce and release a large amount of bioaerosol pollen seasonally, controlled by wind and weather [1]. Pollen is one of the most common triggers of seasonal allergies. Many people know pollen allergy as “hay fever”. Most of the pollens that cause allergic reactions come from trees, weeds and grasses. These plants make small, light and dry pollen grains that travel by the wind. Grasses, mainly ragweed, are the most common cause of allergy. Other common sources of weed pollen include sagebrush, pigweed and tumbleweed. Certain species of trees, including birch, cedar and oak, also produce highly allergenic pollen. Plants fertilized by insects, like roses and some flowering trees, like cherry and pear trees, usually do not cause allergies. [2].

Since the counting of these particles is still done mostly manually under microscope, there is exigency for fast, reliable and cost-effective aerosol detectors. New laser-based technology - Rapid-E - promises to deliver real time information by recording scattered light and laser-induced fluorescence patterns, representing morphological and chemical fingerprints of airborne particles.

The main goal of this thesis is to classify four types of pollen, particularly 1500 samples of Broussonetia, 548 samples of Picea, 2033 of Juglans and 3257 samples of Betula, whose morphology can be seen in Figure 1.



(a) Broussonetia (b) Picea

(c) Juglans (d) Betula

Figure 1: Grains of different types of pollen under microscope

Rapid-E includes three different datasets: scattered light image, fluorescence spectrum and fluorescence lifetime. In order to get the best classification results, some kind of data fusion was needed, so that the classifier can see all data before making a decision.

The classification algorithm is based on convolutional neural networks, which have so far shown better performance on similar problems in image processing and object classification compared to other machine learning classifiers [10]. The algorithm is implemented in an integrated development environment Jupyter Notebook, designed for people with knowledge of Python's language. PyTorch library makes it possible to efficiently define, evaluate and calculate, necessary for the training of neural networks. In addition to the above mentioned advantages, this library was chosen for deployment purposes, since through CUDA (Compute Unified Device Architecture) is possible to use the Graphics Processing Unit (GPU), which further provides unavoidably faster computing compared to the Central Processing Unit (CPU) [11].

1 Introduction

Algorithms and data used in this thesis are linked to "Real-time measurements and forecasting for successful prevention and management os seasonal alergies in Croatia-Serbia cross-border region" - RealForAll (2017HR-RS151) project, co-financed by the Interreg IPA Cross-border Cooperation, Provincial secretariat for finances and institutions implementing the project (BioSense Institute - Research Institute for Information Technologies in Biosystems (Serbia), J. J. Strossmayer Universityof Osijek, Department of Mathematics (Croatia), Univeristy of Novi Sad, Faculty of Sciences (Serbia) and City of Osijek (Croatia)), in order to increase the current performance of atmospheric models [3].

2 Data and Technology

2.1 Rapid-E

Real-Time Airborne Particle Identifier (Rapid-E) is a fully automated instrument that can accurately analyze aerosol particles in real time - it characterizes any airborne particle in the range of 0.5-100 micrometers. The idea is to find technology which will count and characterize different airborne particles and ultimately provide a good understanding of atmospheric aerosol concentrations and their influence on the environment and public health [5].

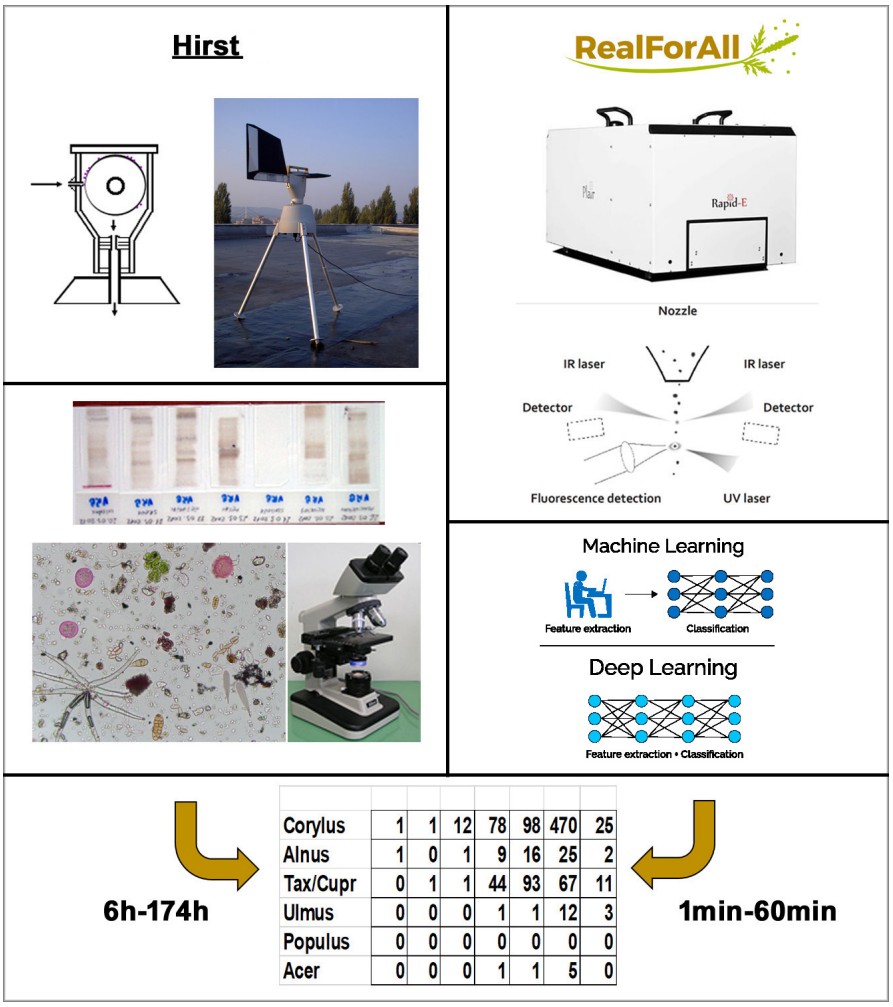


Figure 2: Hirst vs. RealForAll

Source: Hirst vs. RealForAll [3]

The counting of these particles is still done mostly manually under microscope with Hirst, which is very demanding job that can take up to few days [4]. In contrary, classification with Rapid-E only takes few minutes (Figure 2).

Rapid-E works on two physical principles: morphological analysis through scattered light, and chemical analysis through high-resolution - laser-induced fluorescence spectrum and fluorescence lifetime [5].

Continuously pulling in ambient air with its particles, Rapid-E captures the particles, which then enter the nozzle and that creates a narrow laminar flow in the measurement chamber. Time-resolved and multi-pixel scattered light enables morphological analysis. The technology includes a controlled laser, collection lens, linear polarizer and 24-pixel light detectors, placed at different scattering angles of 45 to 135 degrees, which are uninterruptedly gathering the intensity of light on the wavelength of the red laser. According to the Mie theory, every particle crossing the laser diffuses laser light and the pixels record the light's intensity [6], [7].

The system offers an additional degree of discrimination of aerial particles with a new method for chemical analysis. The UV laser interacts twice with the example particle, first contact is carried out with a UV laser polarization and the second follows with a time delay of a few nanoseconds, after rotating the laser polarization by 90 degrees. The fluorescence spectrum is measured with eight consecutive acquisitions distanced by 500 nanoseconds (ns) over 32 channels of the spectrometer. By simultaneous acquisition over four separated bands the fluorescence lifetime is measured [6], [7].

2.2 Data description and preprocessing

As mentioned before, the raw data consists of three measures:

- scattered light image
- fluorescence spectrum
- fluorescence lifetime

Scattered light

Scattering image is of dimensions 24xn, where n is the number of acquisitions, so the width of the image is not fixed. Figure 3 shows the distribution of

the second scattered image dimension of four types of pollen: Broussonetia, Picea, Juglans and Betula.

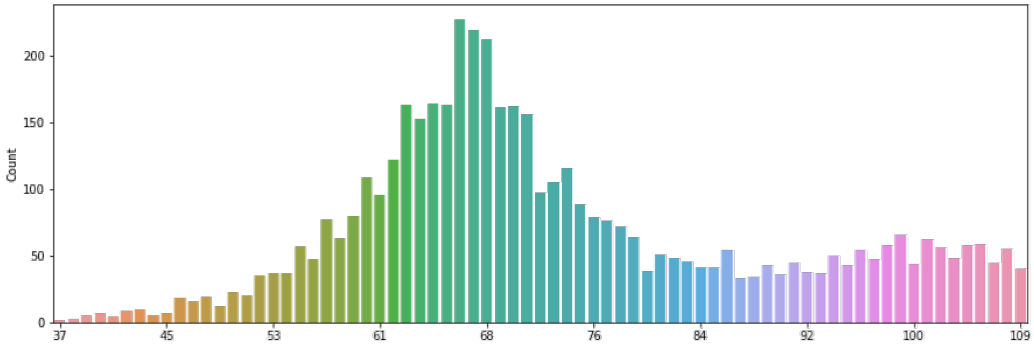


Figure 3: Scattering image width distribution of four types of pollen

Since our classification architecture considers the same input dimensions of every image, it was necessary to define how to equalize each image's width. The first option in resolving this problem was to pad every image up to the size of the biggest image in the dataset. There are few possible ways to execute padding: most popular are zero padding and mirror padding. The first one will add zeros from left and right, column by column, until we get the wanted width. The second one will mirror the existing end regions of the image to get the size that we want [13]. The first problem with this approach is that we do not delete the noise which we believe that is in the both ends of the image. The second problem is that for some other classes of airborne particles, scattering image width can be in the range from 50 up to 200 or more pixels. So if we would have to pad image of width e.g. 50 pixels to be 200 pixels, that would impose some critical problems, both from classification and computer memory point of view, since there would be large amount of zeros in almost every image.

The second method involves finding the center of mass of the image. The idea is inspired from [8], [9] and further expanded with the combination of more connected components. It is performed as follows:

1. Create binary image from the input image.
2. Go trough the binary image with 3x3 median filter.

3. Find the biggest connected component and take its center. Likewise, if there are other connected components that are at least $\frac{1}{3}$ of the size of the biggest connected component, take their center as well.

4. Average all the centers to get the center of mass.

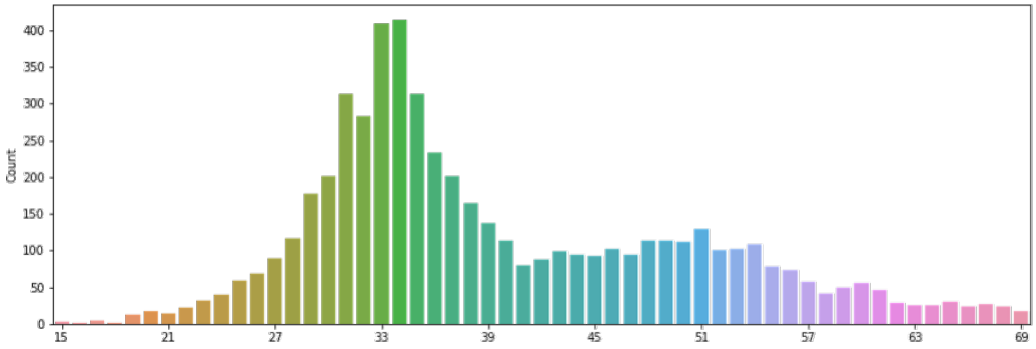


Figure 4: Center of mass distribution for four types of pollen

Most scattering images of Broussonetia, Picea, Juglans and Betula have the center of mass at around 35th pixel (Figure 4). Now that we have found center of mass for each image, we want to either cut or zero-pad each image according to its size. The agreed size for the convolutional neural network was 24x70 pixels. Finally, Figure 5 represents an example of scattering image which went through the explained procedure. Blue color represents low pixel intensities, while yellow color represents high pixel values.

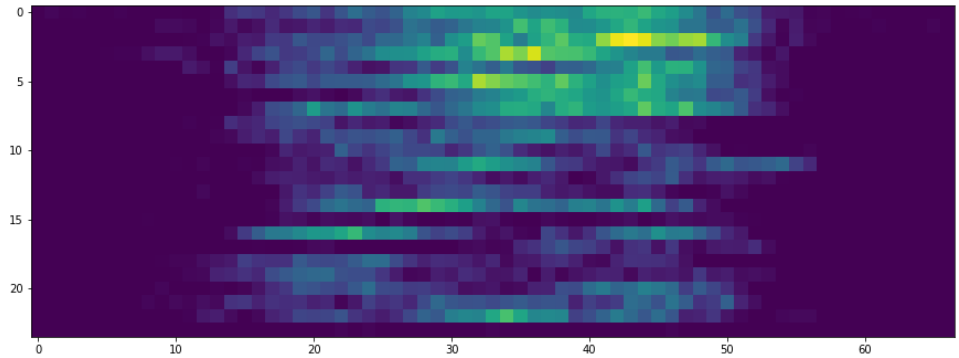


Figure 5: Example of the scatter image

At the end we compress the dynamic range of an image by replacing each pixel value with its logarithm. This has the effect that low intensity pixel values are enhanced [12], [13].

2.2 Data description and preprocessing

Fluorescence spectrum

Fluorescence spectrum has eight consecutive acquisitions distanced by 500 nanoseconds (ns) over 32 channels of the spectrometer unit. Channels correspond to following values in nm: 350.0, 364.5, 379.0, 393.5, 408.0, 422.5, 437.0, 451.6, 466.1, 480.6, 495.1, 509.6, 524.1, 538.7, 553.2, 567.7, 582.2, 596.7, 611.2, 625.8, 640.3, 654.8, 669.3, 683.8, 698.3, 712.9, 727.4, 741.9, 756.4, 770.9, 785.4, 800.0.

1. Spectrum at T_1 – indicates values acquired by the spectrometer during the UV laser shot.

2. Spectrum at T_2 – indicates values acquired by the spectrometer 500 ns after the UV laser shot.

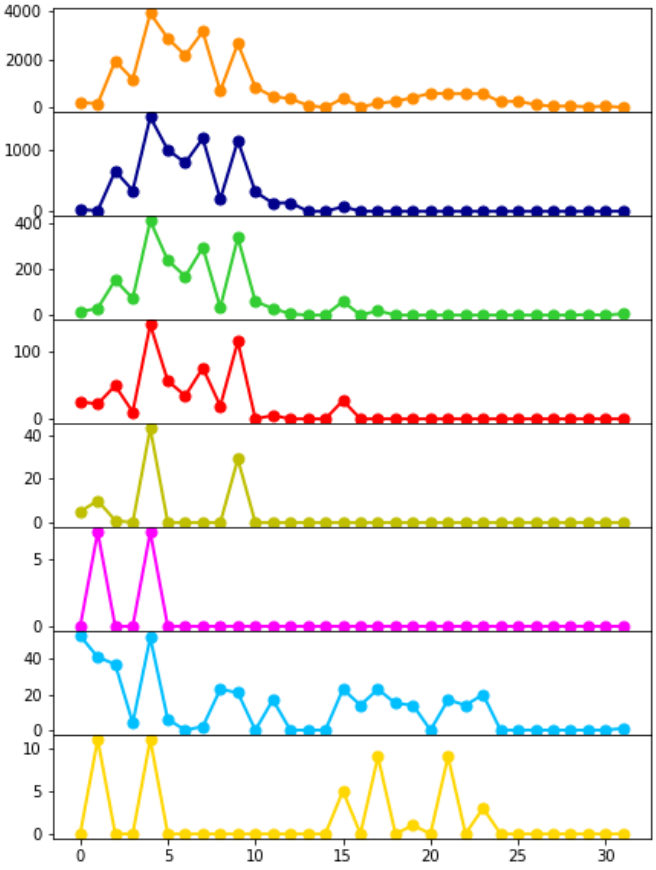


Figure 6: Example of fluorescence spectrum

3. Spectrum at T_3 – indicates values acquired by the spectrometer 1000 ns after the UV laser shot.
4. Spectrum at T_4 – indicates values acquired by the spectrometer 1500 ns after the UV laser shot.
5. Spectrum at T_5 – indicates values acquired by the spectrometer 2000 ns after the UV laser shot.
6. Spectrum at T_6 – indicates values acquired by the spectrometer 2500 ns after the UV laser shot.
7. Spectrum at T_7 – indicates values acquired by the spectrometer 3000 ns after the UV laser shot.
8. Spectrum at T_8 – indicates values acquired by the spectrometer 3500 ns after the UV laser shot.

Figure 6 represents an example of fluorescence spectrum. Each subplot represents one of eight acquisitions (bands) $T_1...T_8$. On x-axis are the 32 channels of the spectrometer unit.

In order to utilize fluorescence spectrum in our convolutional neural network, instead of using it in a given form, we created an image, stacking bands $T_1...T_8$ on top of each other, which gave us an image of dimensions 8x32 pixels.

The big challenge is filtering the data. Currently, filtering is done in a way that maximal spectral peak must exceed a threshold of value 1500. The value was selected by placing a certain amount of pollen in Rapid-E, which then determines the number of florescent particles. That number was closest to the amount of pollen with threshold value of 1500. Decreased threshold value of 1000 and increased value of 2000 were also considered, but were not optimal considering the trade-off between the number of particles for training and their quality. Also, maximal spectral peak must not exceed a threshold of value 10000.

Fluorescence lifetime

By simultaneous acquisition over four separated bands the fluorescence lifetime is measured. Figure 7 shows an example of fluorescence lifetime. Regarding the convolutional neural network, the concept of stacking bands on top of each other is used here also, creating an image of dimensions 4x64

pixels. Each band of the fluorescence lifetime corresponds with the following ranges (in nm), respectively: "350-400", "420-460", "511-572", "672-800".

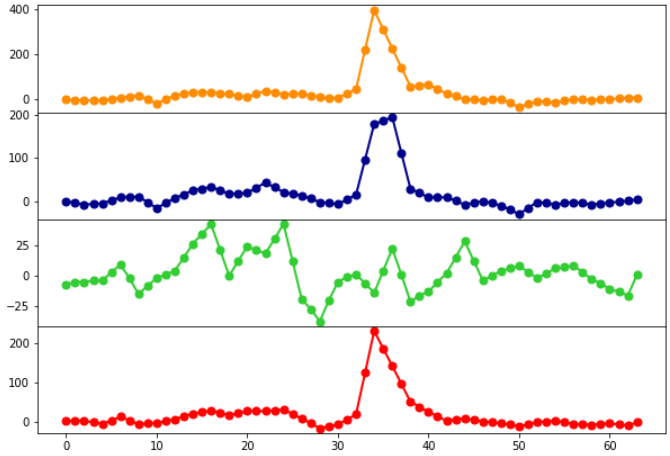
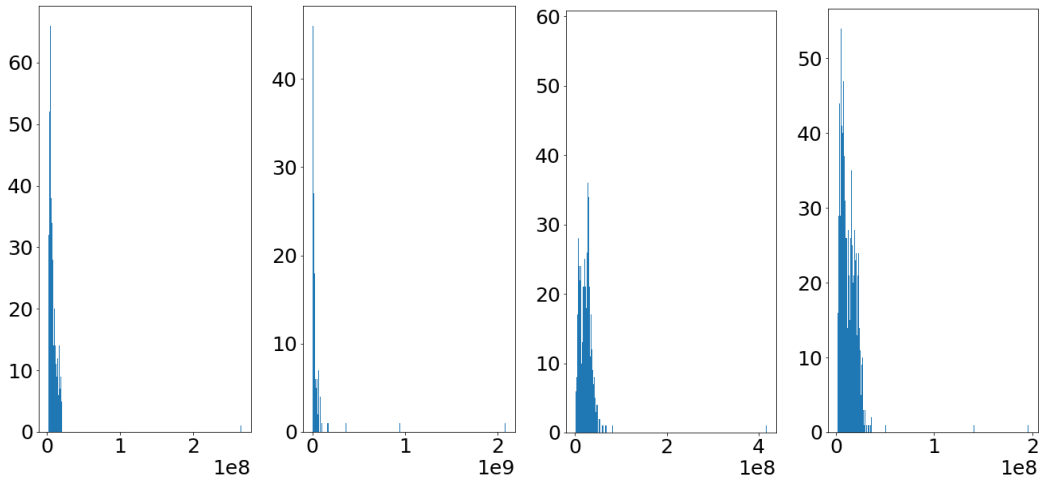


Figure 7: Example of fluorescence lifetime

2.3 Statistics

Beside the variability of the scattering image sizes that was previously described in subsection 2.3, there is high variability in maximum values in



(a) Broussonetia (b) Picea (c) Juglans (d) Betula

Figure 8: Histograms of maximum values in scattering images for four types of pollen

images which is related to the size of particles. To observe the maximum values across four pollen types that we consider in this master thesis, we created histograms, presented in Figure 8.

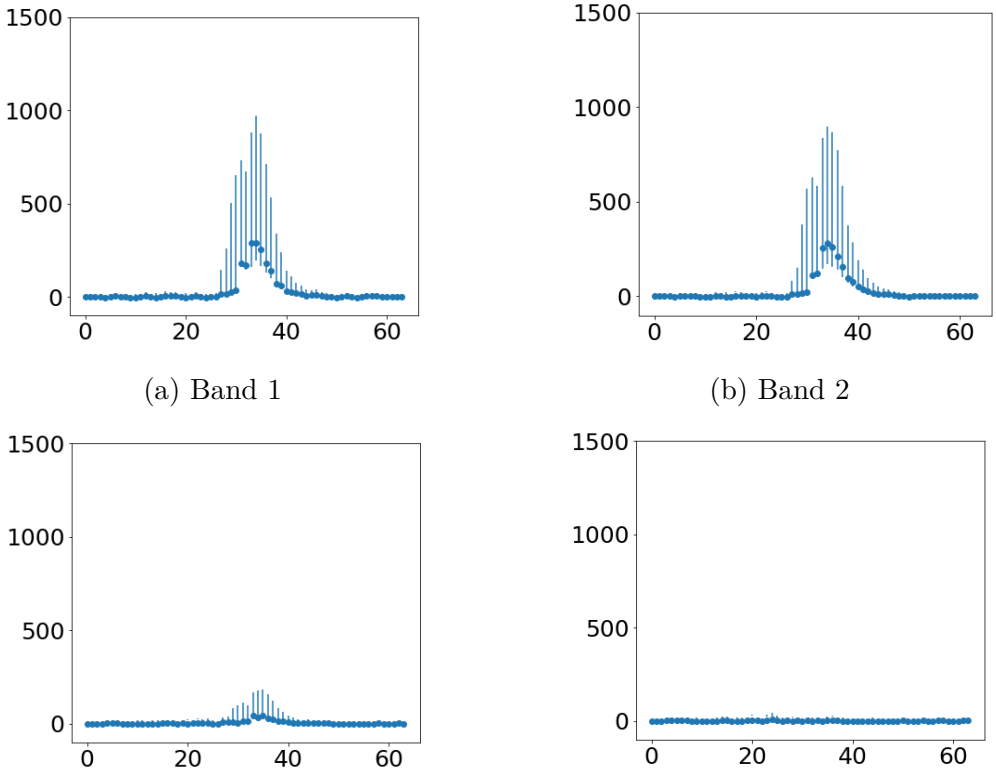
To observe the in-class variability of lifetime and spectral signals of the pollen samples, we have visualized class median values along with the error bars that denote 25th and 75th percentiles. Figure 9 shows the in-class variability of spectrum data for classes Broussonetia, Picea, Juglans and Betula.

(a) Broussonetia (b) Picea (c) Juglans (d) Betula

Figure 9: Median plots with error bars of spectral data for four types of pollen

2.3 Statistics

Figures 10, 11, 12 and 13 show the in-class variability of lifetime data for classes Broussonetia, Picea, Juglans and Betula, respectively.



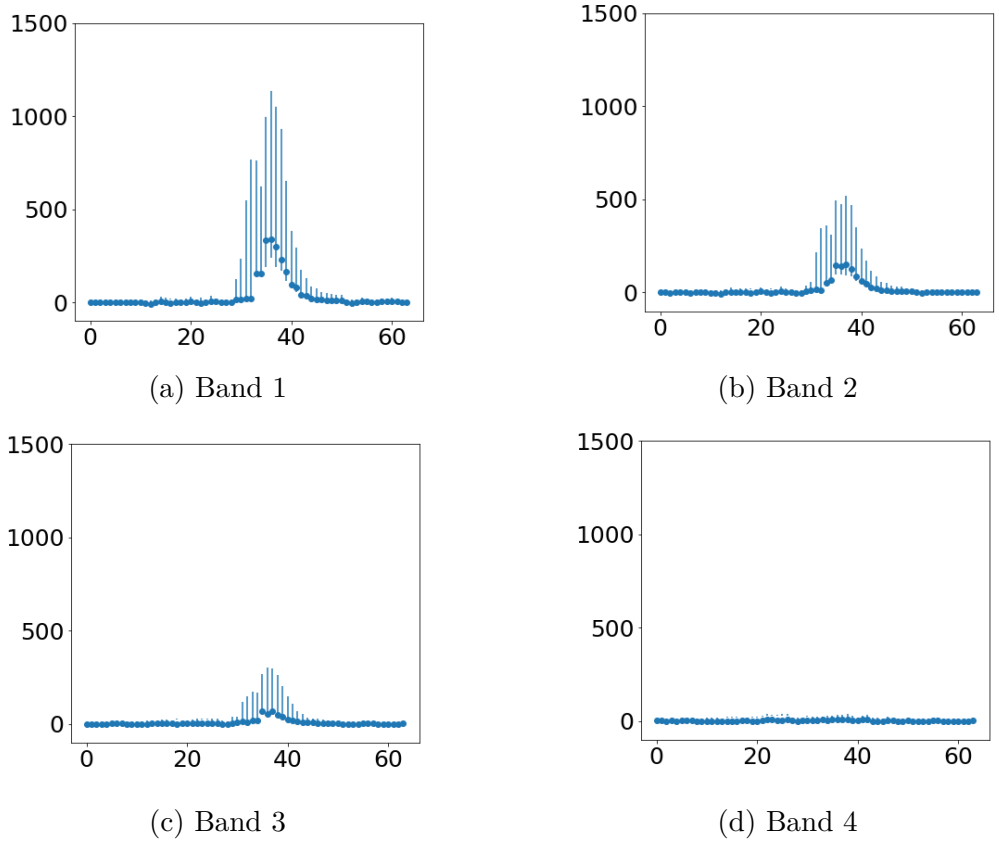
(a) Band 1

(b) Band 2

(c) Band 3

(d) Band 4

Figure 10: Median plots with error bars of spectral data for Broussonetia



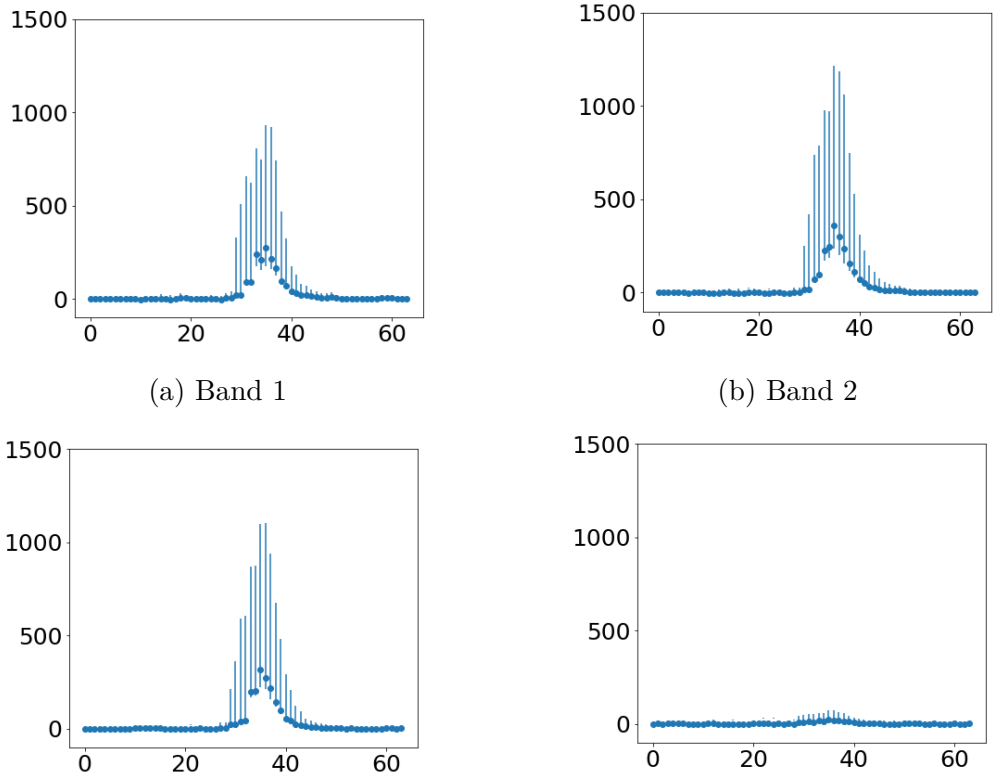
(a) Band 1

(b) Band 2

(c) Band 3

(d) Band 4

Figure 11: Median plots with error bars of spectral data for Picea



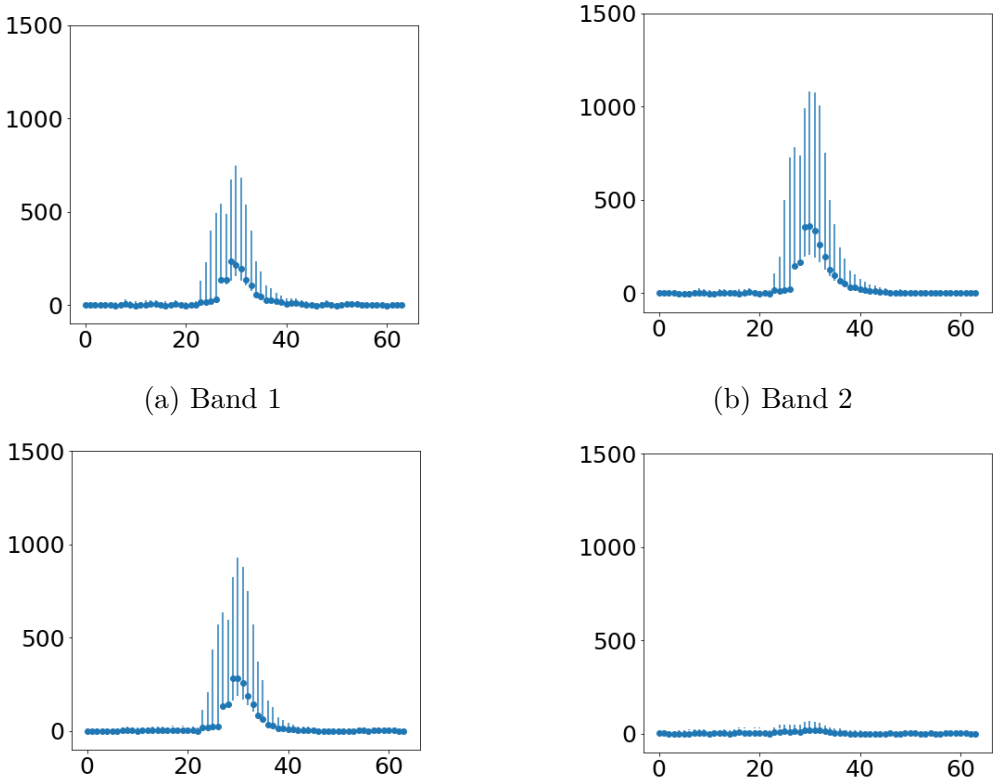
(a) Band 1

(b) Band 2

(c) Band 3

(d) Band 4

Figure 12: Median plots with error bars of spectral data for Juglans



(a) Band 1

(b) Band 2

(c) Band 3

(d) Band 4

Figure 13: Median plots with error bars of spectral data for Betula

High in-class variability demonstrates how challenging is the problem of classifying pollen and that there is need for advanced machine learning methods.

3 Neural Networks

3.1 Basic concepts

In 1943 Warren S. McCulloch and Walter Pitts introduced the idea of neural networks. Since at that time developers manually determined the weight matrices, which was very extensive and demanding job, the development of neural networks has had several ups and downs. In 1958, Frank Rosenblatt develops a well-known backpropagation algorithm that automatically generates weight matrices of neural networks. Since it was proved to be very slow, in the case of a greater number of layers, the further development continued in the 80's and 90's when computer speed and capacity were increased. In need for better hardware to train deep neural networks (DNN), in 2006 Geoffrey Hinton discovered a completely different approach. Graphics Processing Units (GPU) allow training of deeper neural networks with three or more layers [14].

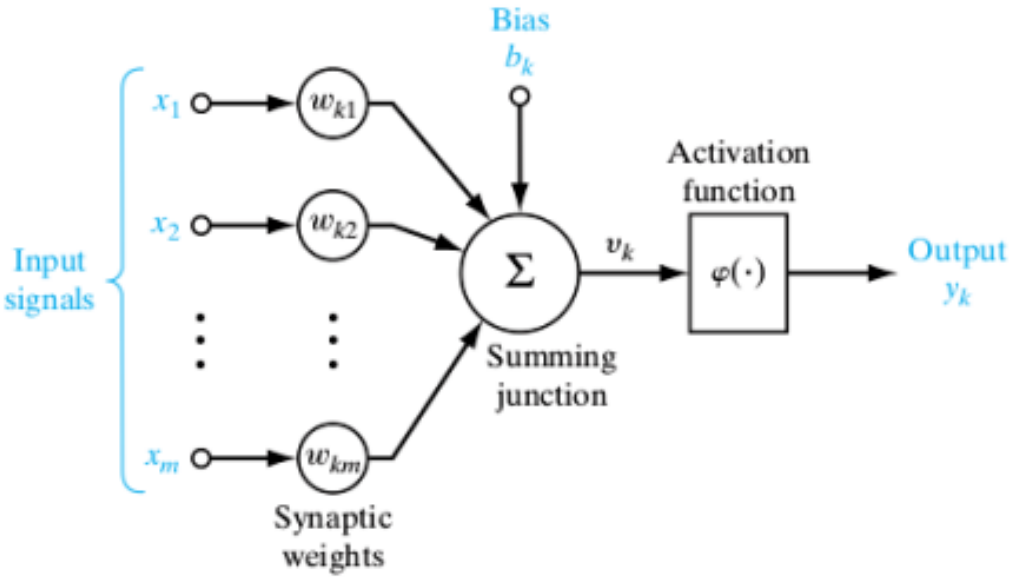


Figure 14: Architecture of an individual neuron

Source: Neural networks in a nutshell [34]

The idea of a neural network is to extract upon the training data the necessary rules on which to classify the unknown samples. It consists of individual, interrelated units called neurons (Figure 14). The input layer consists

of many neurons and receives input features, while each following layer receives as inputs, exit of the previous layer. The output layer gives the output features. Layers between the input and output layers are called hidden layers. Each neuron in the same layer has the same activation function, while network layers may have different activation functions. Each neuron is associated with each neuron from the previous layer [15].

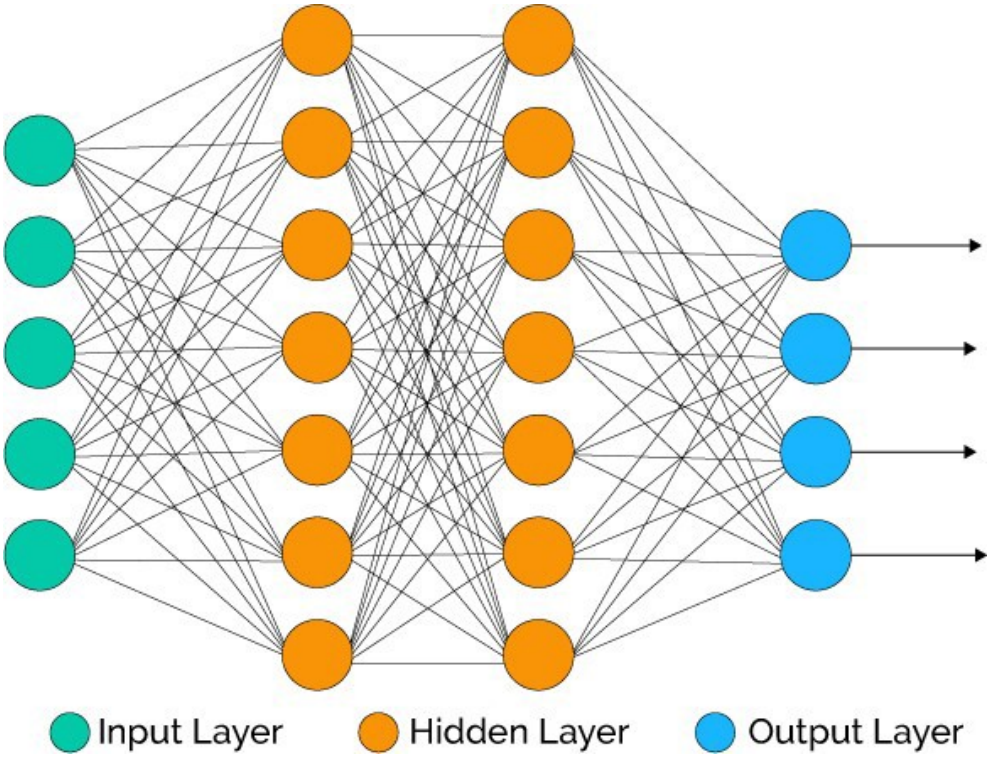


Figure 15: Fully connected neural network with two hidden layers

Source: Machine learning fundamentals: Neural networks [35]

The number of weights and inputs must be the same. Each weight is multiplied by the corresponding input. The product of these multiplications represents the entry into the activation function and then exits the neurons.

$$f(x_i, w_i) = \phi(\sum_i (x_i w_i))$$

In the formula above, the variables x and w represent the inputs and weights of the neurons, respectively. Hidden neurons are connected to each other. They only receive exits from other hidden neurons and serve to understand inputs and generate outputs. They are grouped into completely connected

hidden layers (Figure 15) [14].

There are different neural network architectures. A network where information never passes through the same neuron more than once is referred to as a feedforward network. In these networks, one input influences the activation of all neurons in the remaining layers. [14]

Convolutional neural networks (CNN) are a group of feedforward neural networks that require minimal transformation. They are characterized by their common weights, and they achieve very high accuracy in the application of image classifications, compared to other machine learning algorithms [15]. The goal of a feedforward network is to approximate some function f^*. For example, for a classifier, $y = f^*(x)$ maps an input x to a category y. A feedforward network defines a mapping $y = f(x; \theta)$ and learns the value of the parameters θ which results in the best function approximation. These models are feedforward because information flows trough the function being evaluated from x, trough the intermediate computations used to define f, and finally to the output y. There are no feedback connections in which outputs of the model are fed back into itself [17].

Networks with loops, or with feedback connections, are called Recurrent Neural Networks (RNN). In such networks, neural behavior is not only defined by the activations in the previous layers, but also by activations in previous timeframes. The idea of such neural network is that there are neurons that are active only for a certain period of time. Different neurons are active at different moments or different time periods and thus the cascading response of neurons occurs. Such networks are suitable for applications in data analysis and processes that change over time, such as speech [15].

3.2 Convolutional Neural Networks

In 1980, Fukushima introduced the original concept of a convolutional neural network. Yann LeCun, Lee Bottou, Yoshua Bengio and Patrick Haffner significantly improved the original idea from 1998. From this research has emerged the popular LeNet convolutional neural network with 5 hidden layers. In convolutional neural networks, the order of elements in the vector is of crucial importance for the training of the network. By contrast, most other neural networks treat input data as a vector of values, and the order in which the coming features in the vector are rendered is irrelevant. For convolutional networks, the ordering is very important and cannot be changed after the training of the network. Convolutional neural network has had a

major impact on the development of a Computational Vision (CV). Convolution is an important technique that is often used in deep learning. Hinton has introduced convolution in neural networks in 2014, which is why CNNs achieve very good results because it takes into account the context of pixels and uses overlapping of input data to simulate the functioning of the human eye [16].

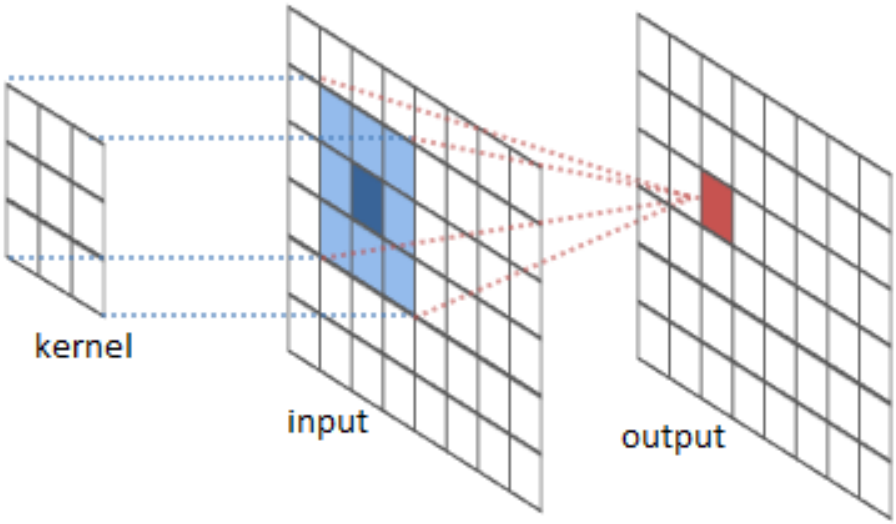


Figure 16: Convolution

Source: Understanding Convolutions [36]

Deep convolutional neuronal networks contain convolutional layers that make very good results in most of the computational nature of the calculus. In complex computational calculations, the dimensionality of parameters is a major problem. Such network is slowly being trained and can easily overfit. The advantage of convolutional layers is that they have far fewer parameters. For computational vision it is important to detect forms, regardless of their location in the image. Fully connected layers have special weights for each pixel, which means that a portion of the network will be trained to recognize the object in the image only if the object is located in the location where it was in the training images. This problem imposed the solution in the form of a filter sliding on the input vector and with every move produces the output (Figure 16). In the case of one-dimensional input, the filter is a vector, in the case of a two-dimensional input it is a matrix. Such filters have relatively small dimensions, mostly 3x3, 5x5 or 7x7 pixels, which significantly reduces the number of parameters in deep neural networks. In this context,

convolution is used to apply a filter to an image at all possible alterations, taking into account the pixel context and imitating human eye sensory cells. The filter makes a set of weights connected to the previous layer, where the previous layer is actually a small part of the input image and the output is a neuron. Such filter is applied by swapping through the full picture, where the sensor fields overlap, thus creating a new matrix that is called a feature map. One convolutional layer contains more than one of these filters, resulting in multiple feature maps. Regular feedforward neural networks generate all the possible weight relations between the two layers. In the terminology of deep learning, such layers are called dense layers [16].

The input to the convolutional layer has dimensions wxhxc, where w is the image width, h height, and c number of channels (in the case of RGB images 3). The dimensionality of the convolutional layer does not depend directly on the input layer, but on the number of channels: wxhxcxk, where w and h is the width and height of the convolution filter, c the number of channels of the previous layer, and k the number of the feature maps of the current layer (equivalent to the channel number at the input). An example of such layer has a filter of size 7x7 and 20 feature maps, which gives 7x7x3x20=2940 weights, which is far less than the number of weights in fully connected layers. That is why people today are using mostly convolutional layers and are avoiding fully connected layers. The feature map, or the output of the convolutional layer, is obtained by convolving the input image with the linear

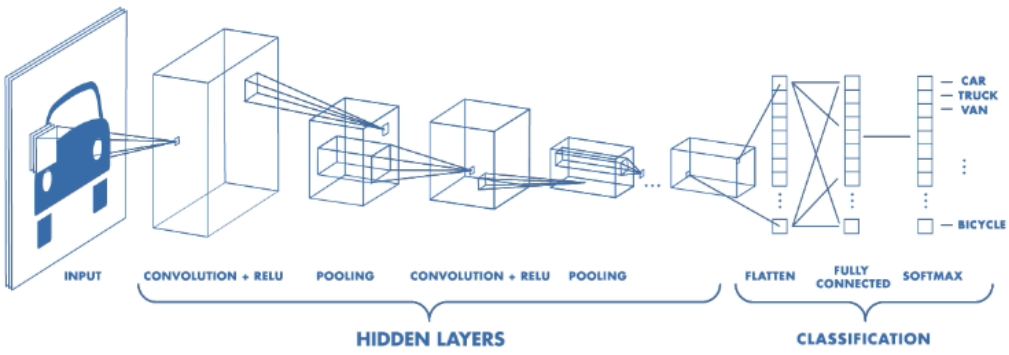


Figure 17: Convolutional neural network

Source: An intuitive guide to Convolutional Neural Networks [37]

filter, adding the bias layer and applying the activation function. If k is the feature map of layer h_k, whose filters are defined by the w_k weights and the bias b_k, then the output of the feature map can be defined by the following expression, where i and j are indices of the input, i.e. the output feature map:

$$h_{ij}^k = h((w^k * x))_{ij} + b_k$$

In practice, each convolutional layer consists of multiple feature maps. In designing neural network architecture, weight and bias values in the network are initially assigned, or network pre-training is performed [16]. Figure 17 shows the concept of convolutional neural network.

3.3 Activation functions

There are several types of activation functions that can be used in neural networks. Properties that are desirable for activation functions are non-linearity and differentiability. Non-linearity provides the possibility of training and learning any nonlinear function. It is desirable that the neural network has at least one neuron with nonlinear activation function that makes the neural network nonlinear. Differentiability is important because of the use of the gradient method while training neural networks. The general form of the activation function is:

$$G(wx^T + b)$$

Usually, weight and bias are initially set to values near the zero gradient. The conclusion that follows is that the expression $wx + b$ also has a value of approximately zero. If the activation function approximates the identity function close to zero, its gradient will be roughly equal to the input [16].

$$\delta G \approx wx^T + b \iff wx^T + b \approx 0$$

A sufficiently strong gradient allows the training algorithm to converge faster. Selecting the right activation function is a very important when creating neural network architecture because, in addition to the above requirement, they learn non-linear complex functional mappings between the inputs and response variables and make sense of something really complicated [16].

Following is the explanation of two activation functions which were used in this paper, ReLU and Softmax. There are many more activation functions.

ReLU (Rectified Linear Unit)

Many researchers point to the benefits of this function in the training process [14], [15], [16]. The use of some other activation functions was mainly for shallow neural networks. The vanishing of the gradient is the main reason why an interruption occurs in training deep networks. ReLU is an activation function that is very efficient computationally. It is defined by the following expression (Figure 18):

$$G_{ReLU}(x) = max(0, x)$$

This activation function is very simple and non-linear and has proved to be very effective in practice. Its derivative in the $\Re^+$ domain is always 1 and does not create saturation. In other words, the codomain of this function is from 0 to infinity, which allows very quick convergence of the gradient. Also, there is no vanishing gradient, which makes it a suitable choice for deep neural networks. This feature has the property to cause "dead" neurons. Such neurons return zero for each sample in the training set. This happens because the weight of such a neuron is adjusted so that the wx product is always negative, so the output from the ReLU activation function is always 0. The advantage of this property is that the resulting zeros can be eliminated from the network and thus achieve computer efficiency. Calculations are significantly simplified and reduced to comparison, addition and multiplication and propagation of a gradient is much more effective if it has a constant value or equal to 0. The disadvantage of this property is that a "dead" neuron can affect overall network accuracy. A good practice is to check the network during the training process [16].

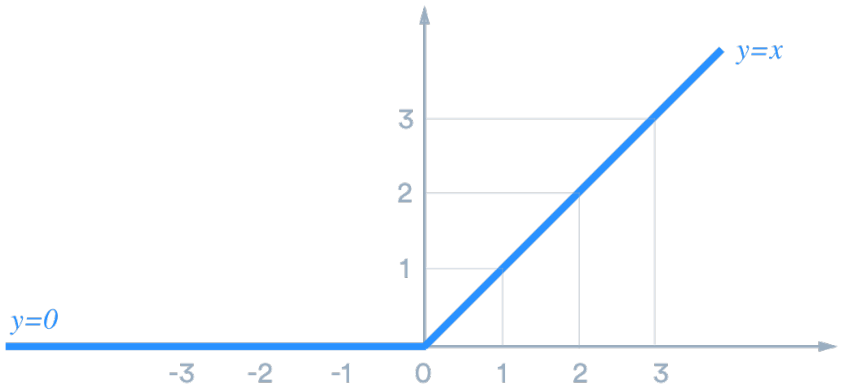


Figure 18: ReLU

Source: Understanding ReLU [38]

Softmax

In this thesis, softmax activation function is used at the output layer of the neural network for classification. The neuron having the highest value declares the input as a member of a particular class, and thus gives an output in the form of probability to which class the input belongs. Without this function, the output represents the numeric value with the highest indication for the class to which the input belongs. This can be easily explained in the case of a flower classification Iris. The database contains 4 measurements for 150 different flowers. Each of the flowers belongs to one of three classes (types). The softmax function allows the neural network to determine the probability that the measurements belong to the classes of species. For example, based on measurements, the output of a neural network will estimate that the probability of 80% is that the input data, i.e. the flower, belongs to the first type, 15% to the other, and 5% to the third. To classify such input data, it is necessary to have an output neuron for each of the classes. Also, the sum of all probabilities should be equal to 100% [16].

$$G_{softmax_i} = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

In the formula above, i represents the output neuron index that counts, and j the index of all neurons in the layer. The variable z represents a set of the output neurons. Output of the individual neuron depends on the output of other neurons [14].

3.4 Cross entropy cost function and methods for regularization

To determine the cost function for the softmax activation function, we start from the likelihood function, which says that by selecting a certain set of model parameters one can predict the belonging to the class of each input sample. The maximized value of this function is:

$$argmax_\theta L(\theta|t,z)$$

Maximizing this function can also be determined by minimizing the negative logarithmic likelihood function, where ξ is a function of the cross-entropy error:

$$-L(\theta|t,z) = \xi(t,z) = -log \prod_{i=c}^{C} y_c^{t_c} = -\sum_{n=c}^{C} t_c log(y_c)$$

It should be noted that in the case of two classes, the output is $t_2 = 1 - t_1$, which means that the expression of the function of the cross-entropy error for this case is equal to the expression for logistic regression error:

$$\xi(t, y) = -t_c log(y_c) - (1 - t_c) log(1 - y_c)$$

Following is the cross entropy error function over a larger series of samples of size n, where $t_{ic} = 1$ if and only if the sample i belongs to the class ci, and y_{ic} represents the probability that the sample i belongs to the class c:

$$\xi(T, Y) = \sum_{i=1}^{n} \xi(t_i, y_i) = -\sum_{i=1}^{n} \sum_{c=1}^{C} t_{ic} log(y_{ic})$$

The derivative of the cross-entropy error function for the softmax activation function is calculated as follows:

$$\frac{\partial \xi}{\partial z_i} = -\sum_{j=1}^{C} \frac{\partial t_j log(y_j}{\partial z_i} = -\frac{t_i}{y_i} \frac{\partial y_i}{\partial z_i} - \sum_{j \neq 1}^{C} \frac{t_j}{y_j} \frac{\partial y_j}{\partial z_i} = -t_i + t_i y_i + \sum_{j \neq 1}^{C} t_j y_i$$

$$= -t_i + \sum_{j=1}^{C} t_j y_i = -t_i + y_i \sum_{j=1}^{C} t_j = y_i - t_i$$

The final result of the error function for the softmax activation function is the same as for logical regression with one output node and it is given by [18]:

$$\frac{\partial \xi}{\partial z_i} = y_i - t_i$$

In addition to the notion of neurons, deep and shallow neural networks, activation functions and cost functions, the problem of overfitting and regularization are also terms that should be explained in more detail. In order to avoid overfitting, it is necessary to divide the database into three parts: a set for training, validation and testing. The training kit is used only in the training phase of the network, a test set for assessing how well the network is trained, and a validation set within the training, but with the purpose of assessing the error for the appropriate choice of parameters, assisting in model selection and so on. When one is determined that the network is trained, for a satisfactory local minimum over a training set, it is necessary to determine the parameters in which the network has yielded good results and how much iteration is needed to find the accuracy in a certain tolerant band. These

parameters are very difficult to determine because the results of the network can stagnate in several iterations before the training process continues. There is a possibility to specify the number of iterations in advance. This practical approach saves and reduces the training time, although it is not easy to determine in advance the number of necessary iterations for the most accurate network training. Other parameters, such as the number of epochs, the number of neurons in the hidden layer, the number of layers, the size of the filters in the case of convolutional neural networks, the learning rate, the regularization parameter, in other words those that the network cannot correct itself in the training process, are called hyperparameters. Validation set is used to determine these parameters. When training networks with large datasets, there is less overfitting. Collecting, classifying and pre-processing data is the most expensive job and is very important because it affects the final outcome. In order to prevent the overfitting, some regularization methods are used. The most commonly used methods are L1, L2 and dropout, and now more modern techniques are used such as batch normalization [15], [19].

In 2012, Hinton introduced a dropout as a simple and efficient algorithm for suppressing overfitting, which works by removing certain neurons in the dropout layer. This method does not change the cost function, but in every epoch it eliminates a predetermined percentage of the hidden neurons, and thus trains the network, and then returns the absent neurons. In this way,

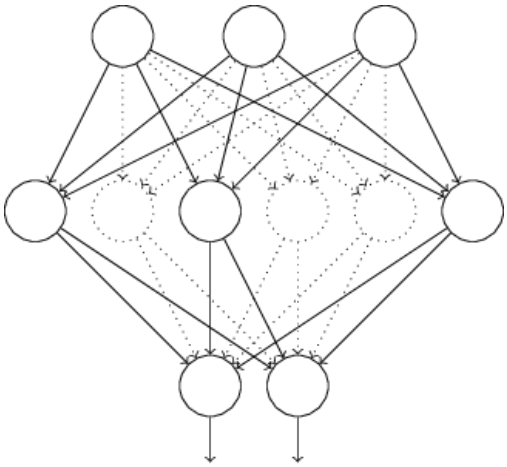


Figure 19: Dropout layer

Source: Explaining dense and dropout layers [39]

the network learns not to depend on few neurons and disregard the rest [14], [16], [21].

In Figure 19, the neural network contains the input, dropout and output layer. Dropout layer removes several neurons. Circles represented by a broken line are neurons eliminated by this layer, and dashed arrows represent the weights that this method removes [14], [16].

When training data is split into small batches, each batch is jargoned as a mini-batch. Using mini-batches of samples is much more effective than m computations for individual examples, or using the whole training set as one "batch", due to the parallelism that comes with the modern computing platforms. Batch normalization reduces internal covariate shift and dramatically accelerates the training of neural networks. The main idea is to fix the means and variances of layer inputs. Normalization is performed on each scalar feature independently, by making it have zero mean and unit variance. Normalization of each dimension for a layer of d-dimensional input $x = (x^{(1)}, \ldots, x^{(d)})$ is performed:

$$\hat{x}^{(k)} = \frac{x^{(k)} - E[x^{(k)}]}{\sqrt{Var[x^{(k)}]}},$$

where the expectation and variance are computed over the mini-batch. Even when the features are not decorrelated, such normalization speeds up convergence [19], [20].

Batch normalization prevents the network from getting trapped in the saturated modes and therefore allows us to use saturating nonlinearities. It permits us to use higher learning rates, since it has a beneficial effect on the gradient flow through the network. Finally, batch normalization reduces the need for dropout since it, in a way, regularizes the model [19], [21].

One important concept of CNNs is pooling - a form of nonlinear downsampling. A pooling function replaces the output of the net at a certain location with a summary statistic of the nearby outputs. There are several functions to implement pooling among which max pooling is the most common (Figure 20). Pooling helps to make the representation become approximately invariant to small translations of the input. Invariance to translation means that if we translate the input by a small amount, the values of most of the pooled outputs do not change. The use of pooling can be viewed as adding an infinitely strong prior that the function the layer learns must

be invariant to small translations. When this assumption is correct, it can greatly improve the statistical efficiency of the network [17].

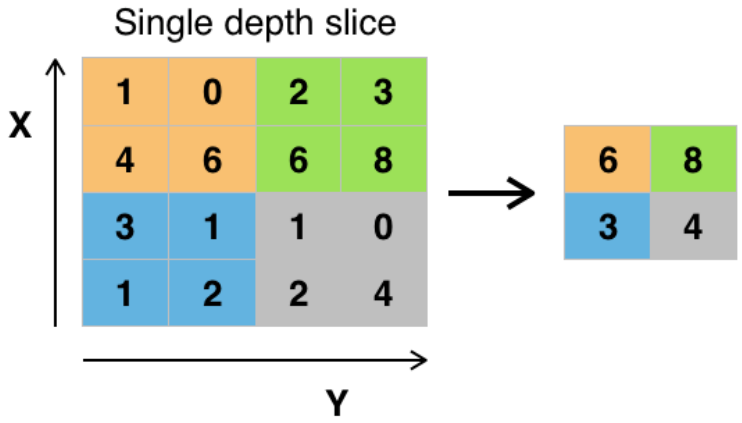


Figure 20: MaxPool layer

Source: A Guide To Understanding Convolutional Neural Networks [42]

3.5 Stochastic Gradient Descent (SGD)

In order to find the minimum of our cost function, an optimizer must be chosen. One of the most popular algorithms is the stochastic gradient descent algorithm. In the total gradient descent algorithm, after each sweep over the training set the weights are updated. Appropriate learning rates ensure convergence of this algorithm to a local minimum of the cost function. The stochastic gradient descent algorithm however has been shown to be more reliable, faster and less prone to reach bad local minima than standard gradient descent. Here, after the presentation of each example, the weights are updated according to the gradient of the loss function, i.e. the value of the cost for this example only [22].

Consider an example $z = (x, y)$, where x is an arbitrary input and y scalar output, and a loss function $l(\hat{y}, y)$. Choose a family F of functions $f_w(x)$ parameterized by a weight vector w. We seek the function $f \in F$ that minimizes the loss $Q(z, w) = l(f_w(x), y)$ averaged on the sample $z_1, \ldots, z_n$

$$E_n(f) = \frac{1}{n} \sum_{i=1}^{n} l(f(x_i), y_i)$$

The empirical risk $E_n(f)$ measures the training set performance. The statistical learning theory (Vapnik and Chervonenkis (1971)) justifies minimizing

the empirical risk instead of the expected risk when the chosen family F is sufficiently restrictive. [23]

Minimization of the empirical risk $E_n(f_w)$ using gradient descent (GD) is done in a way that each iteration updates the weights w on the basis of the gradient of $E_n(f_w)$,

$$w_{t+1} = w_t - \gamma \frac{1}{n} \sum_{i=1}^{n} \nabla_w Q(z_i, w_t)$$

Here γ is an adequately chosen learning rate. The learning rate should be chosen such that it is not too small, so that the execution time of the algorithm will not take too long, but also it's not too big, because it can skip and not found the global minimum [15]. Much better optimization algorithms can be designed by replacing the scalar gain γ by a positive definite matrix Γ_t that approaches the inverse of the Hessian of the cost at the optimum:

$$w_{t+1} = w_t - \Gamma_t \frac{1}{n} \sum_{i=1}^{n} \nabla_w Q(z_i, w_t)$$

The stochastic gradient descent (SGD) algorithm is a drastic simplification. Instead of computing the gradient of $E_n(f_w)$ exactly, each iteration estimates this gradient on the basis of a single randomly picked example z_t:

$$w_{t+1} = w_t - \gamma \nabla_w Q(z_t, w_t)$$

The stochastic process $w_t, t = 1, ...$ depends on the examples randomly picked at each iteration. Since the stochastic algorithm does not need to remember which examples were visited during the previous iterations, it can process examples on the fly in a deployed system. In such a situation, the stochastic gradient descent directly optimizes the expected risk, since the examples are randomly drawn from the ground truth distribution. The convergence of stochastic gradient descent usually requires decreasing gains satisfying the conditions $\sum_t \gamma_t^2 < \infty$ and $\sum_t \gamma_t = \infty$ [23].

The momentum method, originally developed by Polyak [25], is a technique for accelerating gradient descent that accumulates a velocity vector in directions of persistent reduction in the objective across iterations. Given an objective function $f(\theta)$ to be minimized, classical momentum is given by [24]:

$$v_{t+1} = \mu v_t - \epsilon \nabla f(\theta_t)$$
$$\theta_{t+1} = \theta_t + v_{t+1}$$

As we can see from the Figure 21, we refer to momentum as the heavy ball technique to accelerate the convergence rate of gradient-type methods.

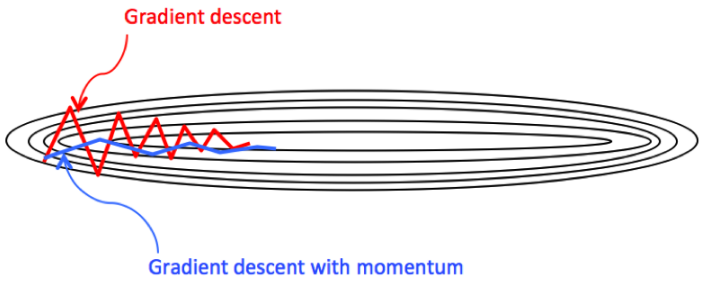


Figure 21: SGD with momentum

Source: Deep Neural Networks tuning and optimization [40]

3.6 Back-propagation

When training a neural network, there is a criterion for how good the final result is. A measure of quality is a cost function and the goal is to minimize this function by choosing the right values of weight and bias. The cost function is a function of a large number of variables and it is not easy to determine its global minimum mathematically. For the minimization of the cost function, we used stochastic gradient descent algorithm [15].

The backpropagation algorithm proved to be very efficient for calculating the gradient of the cost function. It calculates the partial derivatives of the cost function for each weight and each bias. Error calculation is performed from the last to the first layer, by which the algorithm was named [16]. Backpropagation algorithm steps (Figure 22)[15]:

1. Input x: Set appropriate activation a^1 for input layer.
2. Feedforward: For every $l = 2, 3, ..., L$ calculate $z^l = \omega^l a^{l-1} + b^l$ and $a^l = \sigma(z^l)$.
3. Output error δ^l: Calculate vector $\delta^l = \nabla_a C \odot \sigma'(z^L)$.
4. Backpropagation: For every $l = L-1, L-1, ..., 2$ calculate $\delta^l = ((\omega^{l+1})^T \delta^{l+1}) \odot \omega'(z^l)$
5. Output: Gradient of cost function is given with $\frac{\partial C}{\partial \omega^l_{jk}} = a^{l-1}_k b^l_j$ and $\frac{\partial C}{\partial b^l_j} = \delta^l_j$

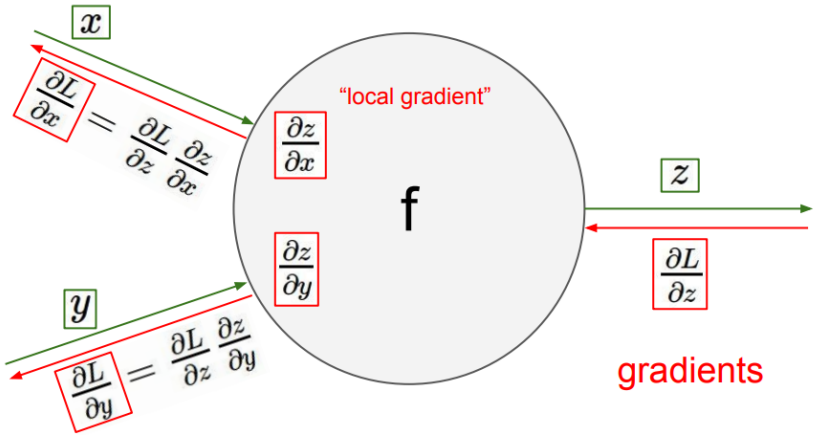


Figure 22: Backpropagation

Source: Back Propagation in Convolutional Neural Networks [41]

Variable x represents one sample from the data set for training, L is the number of layers of the neural network, a_j^l is the activation (input) of j-neuron from the l-layer, b_j^l represents bias of j-neuron from the l-layer, δ_j^l error of j-neuron in l-layer, ω_{jk}^l the weight of the relationship between the k-neuron from the $(l-1)$-layer and the j-neuron from the l-layer. The symbol $\odot$ represents the vector multiplication element per element. Considering that in this paper a convolutional neural network is used, following is an example of a gradient calculation based on the backpropagation algorithm in the convolution layer:

$$\delta^{(i)}{}_n = \frac{\partial J}{\partial x_n} = \frac{\partial J}{\partial y}\frac{\partial y}{\partial x_n} = \sum_{i=1}^{|n|}\frac{\partial J}{\partial y_{n-i+1}}\frac{\partial y_{n-i+1}}{\partial x_n} = \sum_{i=1}^{|n|}\delta_{n-i+1}\,w_i = \left(\delta^{(v)} * flip(w)\right)[n],$$

$$\delta^i = \left[\delta^{(i)}{}_n\right] = \delta^{(v)} * flip(w)$$

$$\frac{\partial J}{\partial w_i} = \frac{\partial J}{\partial y}\frac{\partial y}{\partial w_i} = \sum_{n=1}^{|N|-|n|+1}\frac{\partial J}{\partial y_n}\frac{\partial y_n}{\partial w_i} = \sum_{n=1}^{|N|-|n|+1}\delta_n{}^{(v)}x_{n+i-1} = \left(\delta^{(v)} * x\right)[i],$$

$$\frac{\partial J}{\partial w} = \left[\frac{\partial J}{\partial w_i}\right] = \delta^{(v)} * x = x * \delta^{(v)}$$

The formulas above represent a backpropagation algorithm in a convolution layer where x represents the input to the convolution layer, y output, w the feature vector and δ the gradient of the cost function [26].

Different layers of the network are characterized by different learning speeds. Higher layers learn at moderate, normal speeds. It happens that the lower layers are clogged during the training phase, i.e. that the learning process slows down or completely stops. The reason for this phenomenon is the declining gradient algorithm. In case of blocking the learning of higher layers, the problem of the vanishing gradient is created. Apart from these two, there can also be an exploding gradient. Based on this, it can be concluded that the gradient in deep neural networks is unstable. The vanishing gradient may be prevented or diminished by using some of the ReLU activation functions or by changing the neural network architecture. The cost function can also have an impact on the speed of learning the neural network [16].

3.7 Data fusion

Standard use of the convolutional neural networks allow us to use one source of information to classify samples. Classification from each separate source, then merging the end results is possible, but it is not a good approach. We wanted to create an architecture that would allow the gradient to flow through the whole network, so that the back-propagation can be done updating the weights for each distinct source.

Merging networks can be done in a few ways [27], [28]. Our approach was to concatenate the features after all convolutional layers. After the final convolutional layer, given features are of form nxm, where n is the number of samples going through the network and m is the number of features that comes out after the final convolutional layer. Note that m is different for each source of information.

Since we had three sources of information, scattering image, fluorescence spectrum and lifetime, suppose that after all convolutions, the features that appear are of form nxm_1, nxm_2 and nxm_3. This can be done for arbitrarily many sources. Now, we can concatenate those features to get the form nx$(m_1 + m_2 + m_3)$ which will be given to the fully connected layer to get the end result.

This approach works well if m_1, m_2 and m_3 are numbers close to each other, which usually is not then case. Better method is to pass the features from the last convolutional layer to one fully connected layer, transforming them into more meaningful features for concatenation. For example, suppose that $m_1 = 200$ $m_2 = 2000$ and $m_3 = 5000$. After concatenation, only $\frac{200}{7200} \approx 2.7\%$ of the feature vector will represent the source 1, $\frac{2000}{7200} \approx 27.7\%$ of the feature

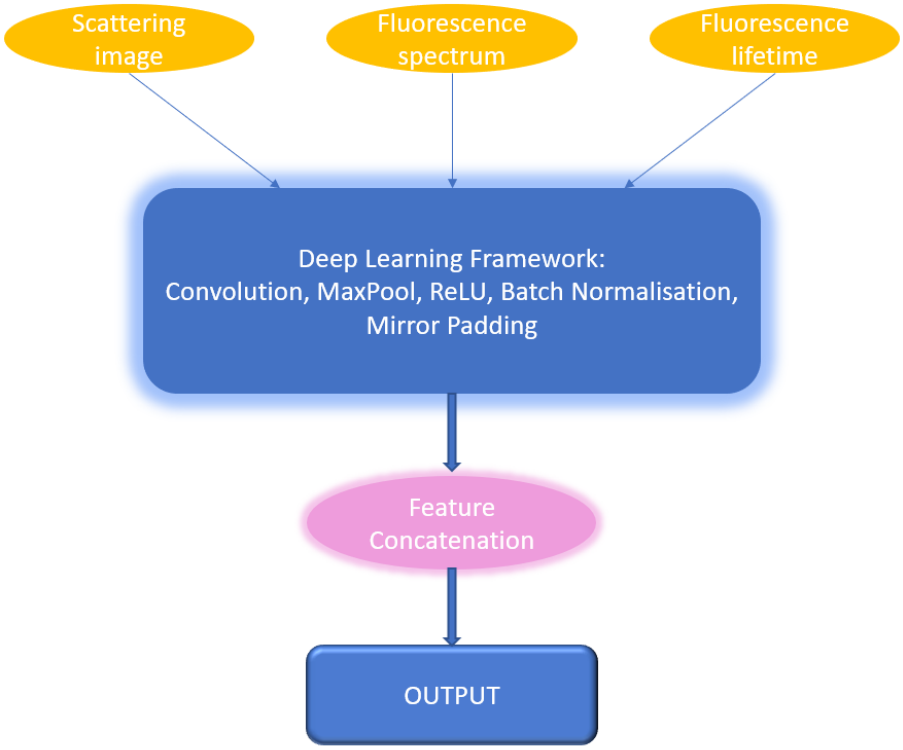


Figure 23: Convolutional Neural Network Architecture

vector will represent the source 2 and $\frac{5000}{7200} \approx 69.4\%$ of the feature vector will represent the source 3. Now, depending on how much information the sources are carrying, this is not very good if your most valuable source is source 1. Therefore, our approach is to transform through one fully connected layer $m_1 \longmapsto m$, $m_2 \longmapsto m$ and $m_3 \longmapsto m$, so that all sources are equally represented in the feature vector. Those transformations do not have to be equal, if you know which source carries more or less information. Figure 23 shows the idea of data fusion in CNNs.

4 Results

Data used in this master thesis is created in calibration events, where domain expert is exposing device with collected pollen samples. Calibration for each class takes up to few minutes. To make the data invariant on time of the calibration, we took random permutation of the data for each class. The data was split before training. The validation set consists of 200 random samples from each class, and the rest of the data is used for training. In order for each of the following experiments to be comparable, the batches used for training were created manually. Therefore, all following experiments used the same data points in each iteration. Also, batches were created to represent all classes equally, having 100 samples of each class.

4.1 Multi-class classification

We wanted to test how our network would perform on just one, two, or on all three different data sources (scattering images, lifetime and spectrum). The performance of networks with one, or combination of two sources, measured in accuracy, is given in Figure 24. The combination of all three sources gives the best accuracy - 84.7%, while the network considering just scattering images yields the lowest accuracy - 67.2%. The highest accuracy considering just one or two sources is 80.9%, obtained with the combination of fluorescence spectrum and lifetime. Figure 25 shows the confusion matrix for classification with all three sources. Here, accuracy is the number of correct predictions divided by the total number of predictions, multiplied by 100 to transform it into a percentage.

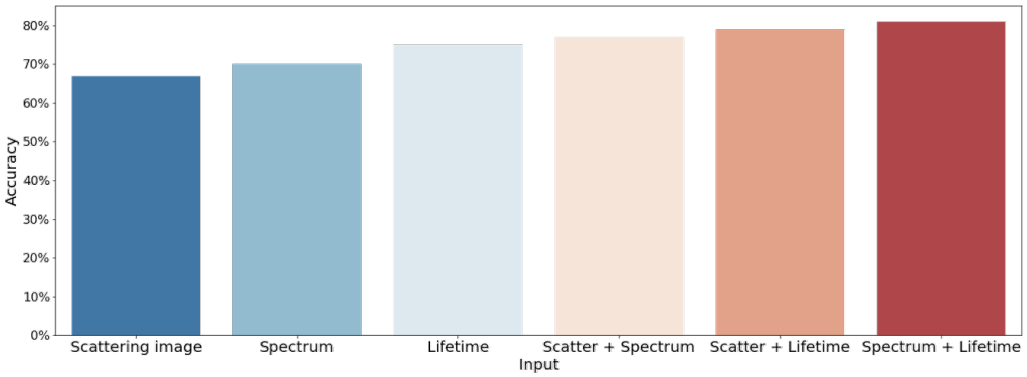


Figure 24: Classification accuracy with different inputs

The first 15 epochs are trained using stochastic gradient descent with 0.1 learning rate, next 15 epochs are trained with 0.01 learning rate and final

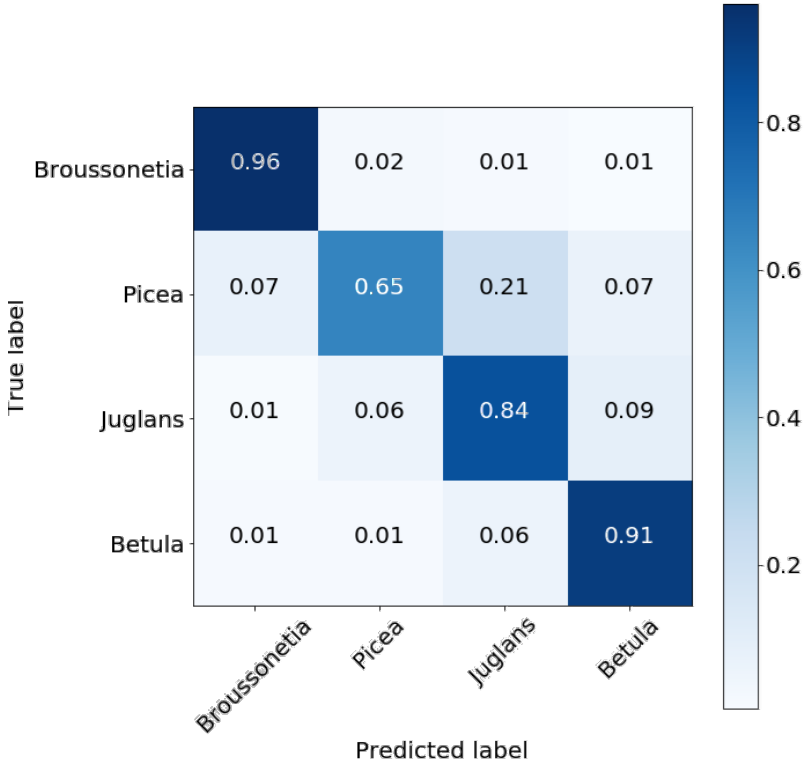
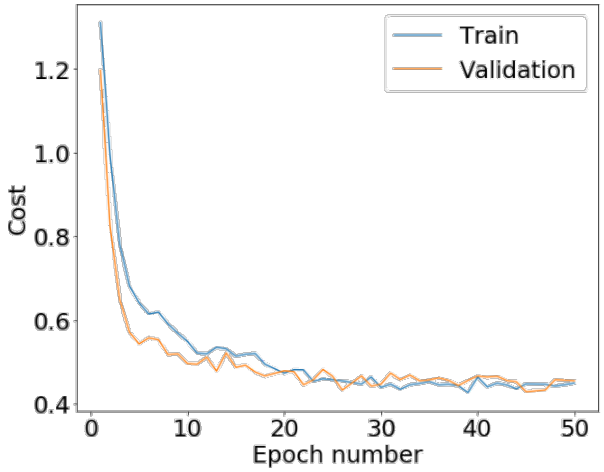


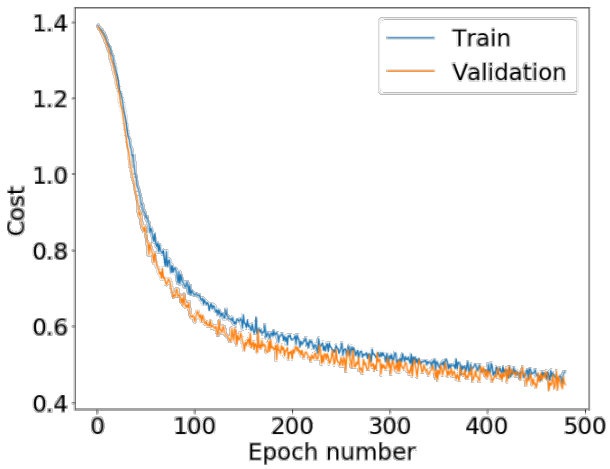
Figure 25: Confusion matrix for classification with all three sources of knowledge

epochs are trained with 0.001 learning rate. All epoch are trained with 0.9 momentum. While minimizing the cost function, we started with high learning rate, so that we could converge to a local minimum more quickly. But, with a high learning rate, the system contains too much kinetic energy and the parameter vector bounces around chaotically, unable to settle down into deeper, but narrower parts of the loss function. Knowing when to decay the learning rate can be tricky. If you decay it slowly, you could waste computation bouncing around chaotically with little improvement for a long time. If you decay it too aggressively, the system could cool too quickly, unable to reach the best position it can. There are few types of implementing the learning rate decay [29]. Since this task did not demand much training time, we decided to change the learning rate manually, whenever the validation error started to saturate. The training took only about 50 epochs, while reducing the validation error rate from 1.3 to around 0.43 (Figure 26(a)).

The experiment with training the network with constant learning rate - 0.001 - showed that we could get the same result, only with ten times more batches (Figure 26(b)), which suggests that the learning rate decay should be implemented in order to decrease training time. Also, while training, the validation error tends to be less than the training error, at least at the beginning (Figure 26). The reason for that is the use of dropout. When the network is in training mode, half of the nodes cannot be seen, therefore the training error is greater than the validation error, since in validation mode the dropout layer is ignored.



(a) Learning rate decay



(b) Constant learning rate: 0.001

Figure 26: Cost vs. number of epochs

		Predicted	
		Negative	**Positive**
Actual	**Negative**	True Negative	False Positive
	Positive	False Negative	True Positive

Figure 27: Classic form of confusion matrix

Consider that we have a confusion matrix as in Figure 27. We introduce three metrics, besides accuracy, for evaluating our classifier:

$$Precision : P = \frac{True\,Positive}{True\,Positive + False\,Positive}$$

$$Recall : R = \frac{True\,Positive}{True\,Positive + False\,Negative}$$

The first one answers answers the question: How many selected items are relevant? The second one answers: How many relevant items are selected? The two measures are sometimes used together in the F1 score to provide a single measurement for a system, where F1 score is defined as:

$$F1 = \frac{P * R}{P + R}$$

The precision, recall and F1 score of our classifier for each class can be seen in Table 1.

Table 1: Precision, recall and F1 score for multi-class classifier

Class	**Precision**	**Recall**	**F1 score**
Broussonetia	0.91	0.95	0.94
Picea	0.86	0.68	0.76
Juglans	0.76	0.84	0.8
Betula	0.88	0.93	0.9
Average	0.85	0.85	0.85

Since the output of the network gives a vector of probabilities, where each element i of the vector represents probability that the sample belongs to class

c_i, we wanted to find out what would happen if we demanded that the classification occurs only if the highest probability in that vector is greater than some probability threshold. We found out that the classification accuracy can be much improved, while discarding a number of samples (Table 2). The problem here is that we need to define only one probability threshold for all classes, which is not good since for some classes the network is more 'certain' - has higher probabilities - then for others.

Table 2: Accuracy vs. probability threshold for multi-class classifier

Probability threshold	**Accuracy**	**Number of samples**
0.5	86%	760
0.6	89%	709
0.7	91%	641
0.8	94%	556
0.9	96%	432

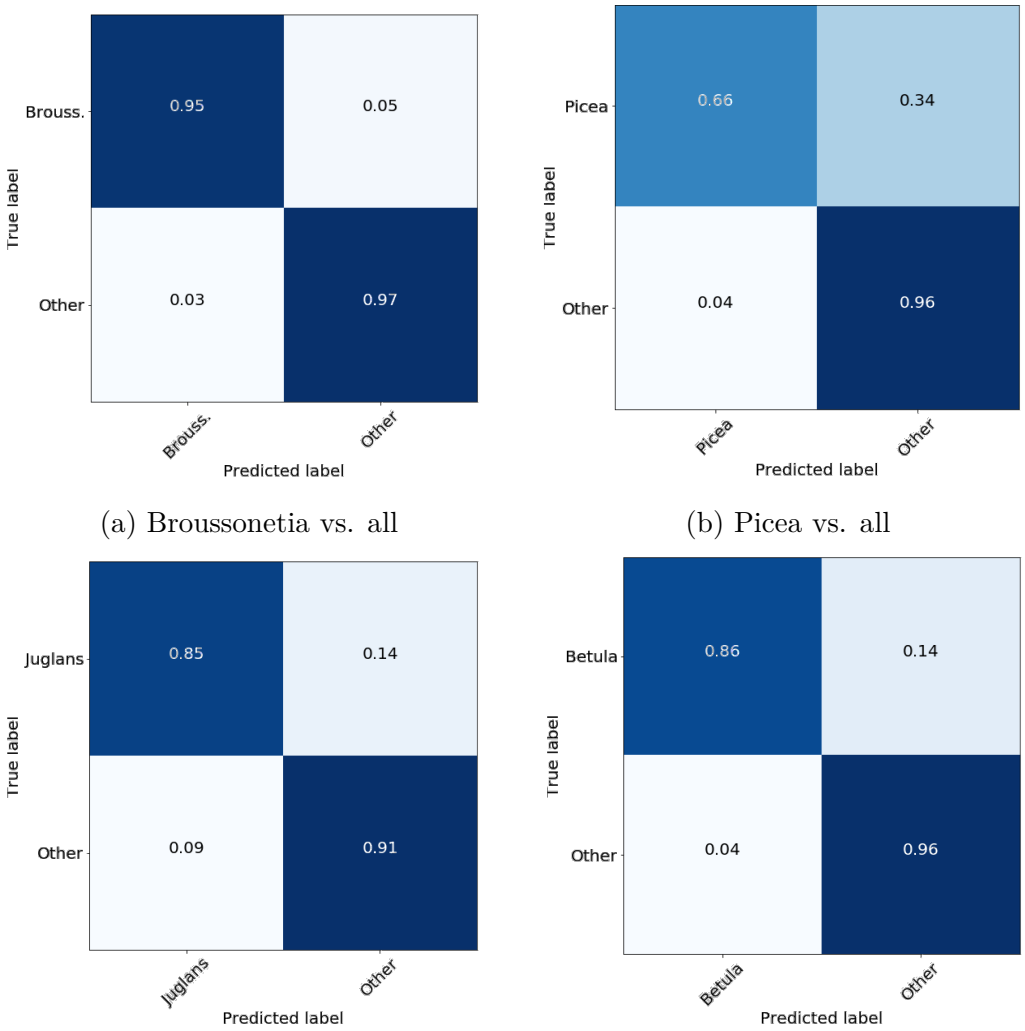
4.2 Binary classification

In order to use different probability thresholds for each class separately, the "one vs. all" approach was used. We trained four different networks, one for each class (e.g. Brousonettia vs. the other three) (Figure 28). The networks yielded:

- Broussonetia: 97% accuracy
- Picea: 89.9% accuracy
- Juglans: 90% accuracy
- Betula: 93.5% accuracy

Table 3: Precision, recall and F1 score for binary classifiers

Class	**Precision**	**Recall**	**F1 score**
Boussonetia	0.92	0.97	0.94
Picea	0.86	0.66	0.74
Juglans	0.76	0.86	0.80
Betula	0.87	0.87	0.88
Average	0.85	0.84	0.84



(a) Broussonetia vs. all

(b) Picea vs. all

(c) Juglans vs. all

(d) Betula vs. all

Figure 28: Confusion matrices for binary classifiers

But here, accuracy measure is not of much value because we have class imbalanced problem, since we have 200 samples from each class for validation (e.g. 200 broussonetia and 600 other). In this case, more meaningful measures would be precision, recall and F1 score (Table 3). Table 4 shows accuracy vs. probability threshold for all four networks, where # samples means number of samples. Overall accuracy is $83.6\% = \frac{\sum_{i=1}^{4} TP(c_i)}{800}$, where $TP(C_i)$ is the number of true positive for class c_i.

Table 4: Accuracy vs. probability threshold for binary classifiers

Probability threshold	Broussonetia		Picea	
	Accuracy	# samples	Accuracy	# samples
0.6	97.5%	785	91.1%	741
0.7	98%	765	93.7%	682
0.8	98.3%	752	94.1%	623
0.9	98.7%	700	97%	526
	Juglans		**Betula**	
	Accuracy	**# samples**	**Accuracy**	**# samples**
0.6	92.9%	741	96%	758
0.7	94.8%	669	96.7%	731
0.8	95.9%	557	98.4%	671
0.9	97.5%	435	99.3%	615

4.3 Fusion of binary classifiers

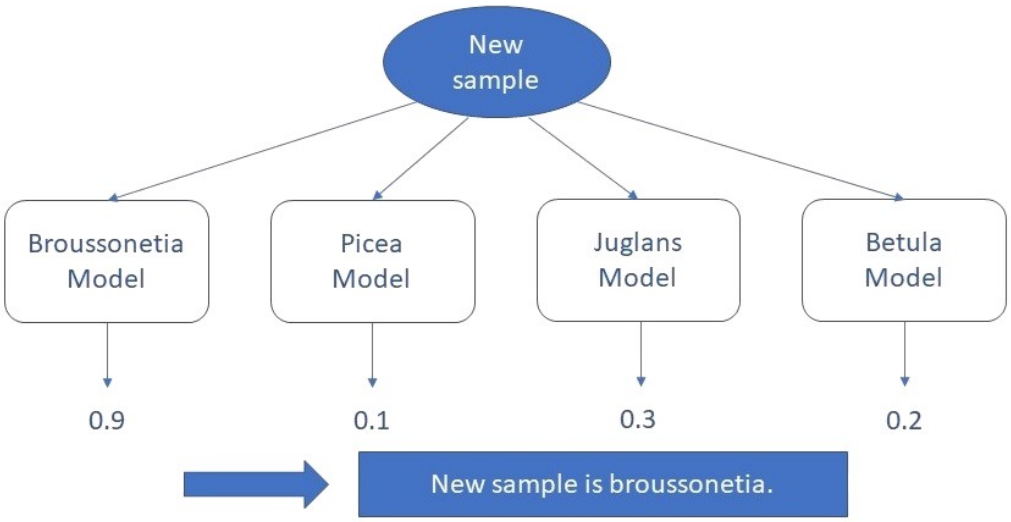


Figure 29: Example of combining binary classifiers for decision making

Finally, we wanted to try and classify samples with each of the four networks. After the classification process, we classified each sample according to which network gave highest probability (Figure 29). This is a very naive approach with which we have obtained accuracy of 84.9%, without further training. Further research should improve the usage of the combination of classifiers as

well as the usage of the probability thresholds. We also imposed probability thresholds and obtained an increase in accuracy (Table 5).

Table 5: Accuracy vs. probability threshold for combination of binary classifiers

Probability threshold	**Accuracy**	**# samples**	**# samples of Broussonetia**
without	84.9%	800	200
0.5	87.3%	748	197
0.6	89.6%	685	194
0.7	92.2%	593	190
0.8	94%	498	181
0.9	95.6%	385	157
	# samples of Picea	**# samples of Juglans**	**# samples of Betula**
without	200	200	200
0.5	175	188	188
0.6	157	173	161
0.7	132	137	134
0.8	115	99	103
0.9	84	50	94

Conclusion

The "old" way of pollen counting includes at least 6 hours up to 170 hours of hard, demanding work by a domain expert, not counting the non-pollen particles. Thus there is a great need for automated particle classifiers, which could also broaden our knowledge about the emergence of new particles and their impact on people and the environment, as well as about the existing ones.

Rapid-E is the first automated particle counter based on laser induced fluorescence technology, which opens new challenges in research and data analysis. Convolutional neural networks were used in these experiments, although they require caution in order to avoid overtraining [30]. The reason for using convolutional neural networks, besides the one that they have proven to work better than other machine learning algorithms in problems like image classification and object segmentation [10], [31], is that the CNNs provide us a way to fuse different data sources representing the same data points, and to incorporate as many aspects and views of each sample as we can.

The results show us, intuitively, that the best classifier is the one which incorporates all three sources, giving 84.7% accuracy. This outcome exceeds the best result of a classifier trained on the combination of two sources by 3.1%, or the best result of a classifier trained on just one source by 9.9%. The classification yields the average precision, recall and F1 score of 85% for all three measures. Furthermore, by introducing a probability threshold, these results can be improved to 96%, but with the price of discarding the samples below the defined threshold. Since we used the same probability threshold for each of the classes, we discarded the remaining samples unevenly.

In order to use different probability thresholds for each of the classes, we trained four "one vs. all" networks. The networks gave the overall accuracy of 83.6%, which can be improved, without further training, to 84.9% by combining all four networks for classification. The classifier yielded the average precision of 85% and 84% for recall, which is almost the same as for multi-class classifier. The benefit of the second approach is in easier operational organization (extensions to other pollen types, model updates, incorporating domain knowledge).

In the experiments presented in this master thesis we tried to classify four types of pollen, different in morphology, to see whether there is a way to distinguish certain kinds of pollen. The results imply that the four pollen

types can be classified with good accuracy by using convolutional neural networks. Remaining challenge is the initial data filtering which is of particular importance and should be explored in future experiments. The next studies will encompass larger number of pollen types, as well as different neural network architectures and comparison of the results obtained with Hirst device in operational mode.

References

[1] William C. Hinds, Aerosol Technology: Properties, Behavior, and Measurement of Airborne Particles (2nd Edition), 1998.

[2] Asthma and Allergy Foundation of America
http://www.aafa.org/page/pollen-allergy.aspx
Accessed at 07.09.2018.

[3] Interreg - IPA CBC Croatia - Serbia RealForAll
https://www.realforall.com/language/en/about-project/
Accessed at 25.09.2018.

[4] J. M. Hirst, An automatic volumetric spore trap. Ann. Appl. Biol. 39, 257–265., 1952.

[5] Solutions for Automatic Pollen Counting
http://www.plair.ch/
Accessed at 03.09.2018.

[6] D. Kiselev, "Method and device for detection and/or morphologic analysis of individual fluid-borne particles", WO/2017/129390, 10.01.2017.

[7] D. Kiselev, "Device and method for detecting and/or characterizing fluid-borne particles", WO/2017/220249, 09.05.2017.

[8] M. Sonka, V. Hlavac and R. Boyle, Image processing, analysis, and machine vision, 2014.

[9] R. Jain, R. Kasturi and B. G. Schunck, Machine vision (Vol. 5), New York: McGraw-Hill, 1995.

[10] A. Krizhevsky, I. Sutskever and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks", Adv. Neural Inf. Process. Syst., pp. 1–9, 2012.

[11] CUDA Zone
https://developer.nvidia.com/cuda-zone
Accessed at 08.09.2018.

[12] Anil K. Jain, Fundamentals of Digital Image Processing, Prentice Hall, 1989.

[13] R. C. Gonzalez and R. E. Woods, Digital Image Processing (3rd Edition), 2007.

[14] Jeff Heaton, Artificial Intelligence for Humans, Volume 3: Deep Learning and Neural Networks, Heaton Research, Inc, 2015.

[15] M. A. Nielsen, "Neural Networks and Deep Learning", Determination Press, 2015.

[16] Hamed Habibi Aghdam, Elnaz Jahani Heravi "Guide to Convolutional Neural Networks", Springer International Publishing AG 201

[17] Ian Goodfellow, Yoshua Bengio and Aaron Courville, Deep Learning, 2016.

[18] How to implement a neural network http://peterroelants.github.io/posts/neural_network_implementation_intermezzo02/ Accessed at 24.08.2018.

[19] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift", 2015.

[20] Y. LeCun and Y. Bengio, "Pattern recognition and neural networks", Handb. Brain Theory Neural Networks, p. 711, 1995.

[21] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting", 2014.

[22] L. Bottou, "Stochastic Gradient Learning in Neural Networks", Proc. Neuro-Nımes, vol. 91, no. 8, 1991.

[23] L. Bottou, "Large Scale Machine Learning with Stochastic Gradient Descent", Proc. COMPSTAT'2010, pp. 177–186, 2010.

[24] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning", 2013.

[25] B. T. Polyak, "Some methods of speeding up the convergence of iteration methods", USSR Comput. Math. Math. Phys., vol. 4, no. 5, pp. 1–17, 1964.

[26] Hiroshi Kuwajima, "Backpropagation in Convolutional Neural Network", 2014.

[27] X. Song, S. Jiang, and L. Herranz, “Combining models from multiple sources for RGB-D scene recognition”, IJCAI Int. Jt. Conf. Artif. Intell., pp. 4523–4529, 2017.

[28] A. C. B. Eunbyung Park, Xufeng Han, Tamara L.Berg, “Combining Multiple Sources of Knowledege in Deep CNNs for Action Reco”, J. Chem. Inf. Model., vol. 53, no. 9, pp. 1689–1699, 2013.

[29] Convolutional Neural Networks for Visual Recognition
http://cs231n.github.io/neural-networks-3/
Accessed at 10.09.2018.

[30] I. V. Tetko, D. J. Livingstone, A. I. Luik, “Neural Network Studies. 1. Comparison of Overfitting and Overtraining”, J. Chem. Inf. Comput. Sci., vol. 35, no.5, pp. 826-833, 1995.

[31] R. V. Singh, “ImageNet Winning CNN Architectures - A Review”, pp. 3-8, 2015.

[32] Jure Leskovec, Anand Rajaraman, Jeffrey D. Ullman, Mining of Massive Datasets, 2011.

[33] B. Crouzy, M. Stella, T. Konzelmann, B. Calpini, and B. Clot, “All-optical automatic pollen identification: Towards an operational system”, Atmos. Environ., vol. 140, pp. 202-212, 2016.

[34] Neural networks in a nutshell
https://themenwhostareatcodes.wordpress.com/2014/03/02/neural-networks-in-a-nutshell/
Accessed at 29.08.2018.

[35] Machine learning fundamentals (II): Neural networks
https://towardsdatascience.com/machine-learning-fundamentals-ii-neural-networks-f1e7b2cb3eef
Accessed at 29.08.2018.

[36] Understanding Convolutions
http://colah.github.io/posts/2014-07-Understanding-Convolutions/
Accessed at 02.09.2018.

[37] An intuitive guide to Convolutional Neural Networks
https://medium.freecodecamp.org/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050
Accessed at 04.09.2018.

[38] ReLU
https://www.tinymind.com/learn/terms/relu
Accessed at 07.09.2018.

[39] Explaining dense and dropout layers
https://www.quora.com/In-TensorFlow-what-is-a-dense-and-a-dropout-layer
Accessed at 27.09.2018.

[40] Deep Neural Networks tuning and optimization
https://medium.com/machine-learning-bites/deeplearning-series-deep-neural-networks-tuning-and-optimization-39250ff7786d
Accessed at 17.09.2018.

[41] Back Propagation in Convolutional Neural Networks
https://becominghuman.ai/back-propagation-in-convolutional-neural-networks-intuition-and-code-714ef1c38199
Accessed at 22.09.2018.

[42] A Guide To Understanding Convolutional Neural Networks
https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks-Part-2/
Accessed at 01.10.2018.

Biography

Predrag Matavulj was born on the 30th of October 1993 in Vukovar, Republic of Croatia. He received his Bachelor degree in Mathematics in 2017 from the Faculty of Sciences, University of Novi Sad, Serbia and he continued his Master studies in the field of Data Science at the same faculty. During his studies Predrag attended several workshops such as ECMI SIG Workshop on Mathematics for Big Data, Training School on Big Data Processing and Analytics in the Internet of Everything E-ra, etc. He also worked on several projects such as Syngenta Crop Challenge 2017, Classification of Agricultural Crops Based on Satelite Images, etc. Since March 2018, Predrag is working as Junior Researcher at the BioSense Institute, currently involved in projects Data Driven Precision Agriculture Services and Skill Acquisition, H2020-WIDESPREAD-052017, 2018-2021 and RealForAll: Real-time measurements and forecasting for successful prevention and management of seasonal allergies in Croatia-Serbia cross-border region, Interreg IPA Cross-border Cooperation programme, 2017-2019.

UNIVERZITET U NOVOM SADU
PRIRODNO-MATEMATIČKI FAKULTET
KLJUČNA DOKUMENTACIJSKA INFORMACIJA

Redni broj:
RBR
Identifikacioni broj:
IBR
Tip dokumentacije: monografska dokumentacija
BF
Tip zapisa: tekstualni štampani materijal
TZ
Vrsta rada: Master rad
VR
Autor: Predrag Matavulj
AU
Mentor: dr Sanja Brdar
MN
Naslov rada: Fuzija heterogenih podataka u konvolucionim mrežama za identifikaciju čestica polena u realnom vremenu
NR
Jezik publikacije: engleski
JP
Jezik izvoda:s/e
JI
Zemlja publikovanja: Republika Srbija
ZP
Uže geografsko područje:Vojvodina
UGP
Godina: 2018.
GO
Izdavač: autorski reprint
IZ
Mesto i adresa: Novi Sad, Trg Dositeja Obradovića 4
MA
Fizički opis rada: text
FO
Naučna oblast: matematika
NO
Naučna disciplina: primenjena matematika
ND

Ključne reči: konvolucione neuralne mreže, klasifikacija, polen
PO
UDK
Čuva se: u biblioteci Departmana za matematiku i informatiku, Prirodno-matematičkog fakulteta, u Novom Sadu
ČU
Važna napomena:
VN
Izvod: Merenje koncentracije atmosferskog polena u realnom vremenu je od velike važnosti, kako za poboljšanje kvaliteta života populacije osetljive na polen, tako i za poljoprivrednu proizvodnju. Novi aparat, zasnovan na laserskoj tehnologiji, nam može dati morfološke i hemijske uvide o aerosolnim česticama u realnom vremenu snimanjem raspršene svetlosti i lasersko indukovane fluorescencije. Klasifikovali smo četiri tipa polena (Broussonetia, Betula, Picea i Juglans), različitih u morfologiji i veličini, pomoću konvolucione neuronske mreža. Mreža je naučila obeležja iz slike raspršene svetlosti (24 piksela x broj akvizicija), spektra fluorescencije (32 x 8 akvizicija, odvojenih za 500 ns) i fluorescencijskog veka trajanja (64 x 4 akvizicije). Nakon poslednjeg konvolucionog sloja, obeležja su spojena u jedan vektor obeležja koji predstavlja sva tri izvora informacija, što omogućava gradijentu da teče kroz celu mrežu. Rezultat klasifikacije je meren preciznošću, dajući najbolji rezultat od 84.7% za klasifikaciju kada se koriste sva tri izvora. Štaviše, kada uzorak prolazi kroz proces klasifikacije (mrežu), izlaz mreže je vektor verovatnoća, pri čemu svaki element i vektora predstavlja verovatnoću da uzorak pripada klasi c_i. Zahtevajući da se klasifikacija vrši samo ako je najveći element vektora veći od nekog praga verovatnoće poboljšava tačnost rezultata dok odbacuje preostale uzorke. Master teza sumira dobijene rezultate pod različitim postavkama i 'preciznost-osetljivost' kompromisima.
IZ
Datum prihvatanja teme od strane NN veća: 28.09.2018.
DP
Datum odbrane:
DO
Članovi komisije:
KO
Predsednik: dr Dušan Jakovetić, docent
Član: dr Srđan Škrbić, vanredni profesor
Član: dr Sanja Brdar, docent

UNIVERSITY OF NOVI SAD
FACULTY OF SCIENCES
KEY WORD DOCUMENTATION

Accession number:
ANO
Identification number:
INO
Document type: monograph type
DT
Type of record: printed text
TR
Contents code: Master thesis
CC
Author: Predrag Matavulj
AU
Mentor: Sanja Brdar, PhD
MN
Title: Fusion of Heterogeneous Data in Convolutional Networks for Real-Time Pollen Particle Identification
XI
Language of text: English
LT
Language of abstract: s/e
LA
Country of publicatin: Republic of Serbia
CP
Locality of publication: Vojvodina
LP
Publication year: 2018.
PY
Publisher: author's reprint
PU
Publ. place: Novi Sad, Trg Dositeja Obradovića 4
PP
Physical description: text
PD
Scientific field: mathematics
SF
Scientific discipline: applied mathematics
SD

Key words: convolutional neural networks, classification, pollen
UC
Holding data: Department of Mathematics and Informatics' Library, Faculty of Sciences, Novi Sad
HD
Note:
N
Abstract: Real-time measurements of atmospheric pollen concentrations are of paramount importance for the improvement of the quality of life in pollen sensitive population as wells as for agriculture production. New laser-based technology can give us morphological and chemical insights of aerosol particles in real time by recording scattered light and laser-induced fluorescence. We classified four types of pollen (Brussonetia, Betula, Picea and Juglans), different in morphology and size, with convolutional neural network. The network learned features form scattering images (24 pixels x number of acquisitions), fluorescence spectrum (32 x 8 acquisitions separated by 500 ns) and fluorescence lifetime (64 x 4 acquisitions). After the last convolutional layer these features were concatenated into one feature vector representing all three sources of information, which allowed the gradient to flow through the whole network. The result of the classification is measured in accuracy, giving the best result of 84.7% for classification when all three sources are used. Furthermore, when a sample goes through the classification process (network), the output of the network is a vector of probabilities, where each element i of the vector represents probability that the sample belongs to class c_i. Demanding that the classification is performed only if the biggest element in the vector is greater than some probability threshold improves the accuracy score while discarding remaining samples. The master thesis summarizes obtained results under different settings and precision-recall trade-offs.
AB
Accepted by the Scientific Board on: 28.09.2018.
ASB
Defended:
DE
Thesis defend board:
DB
President: Dušan Jakovetić, PhD
Member: Srđan Škrbić, PhD
Member: Sanja Brdar, PhD