AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE
WYDZIAŁ ELEKTROTECHNIKI, AUTOMATYKI, INFORMATYKI I INŻYNIERII BIOMEDYCZNEJ

Praca dyplomowa

An intelligent system of the cognitive association of recognized and well-established 3D objects with the possibility of finding them by a mobile robot.

Inteligentny system kognitywnego kojarzenia rozpoznanych i dobrze ugruntowanych obiektów 3D z możliwością znalezienia ich przez robota mobilnego.

Autor: *Dominik Stachura*
Kierunek studiów: Automatyka i Robotyka
Opiekun pracy: dr hab. *Adrian Horzyk, prof. AGH*

Kraków, 2019/2020

Pragnę złożyć najszczersze podziękowania promotorowi niniejszej pracy, Panu Profesorowi Adrianowi Horzykowi za pomoc w jej realizacji, dzielenie się swoją wiedzą oraz udzielanie cennych rad.

Table of Contents

Introduction

Machine Learning and Deep Learning are concepts that we hear more and more often. Applications using these technologies surround us from all sides, often we are not aware of it. Many devices or techniques that sometimes make life easier for us, and sometimes make life easier for corporations that are trying to get to know us better, use solutions based on neural networks. Social networking sites learn how to personalize ads, they know how to automatically tag photos we upload. Online stores can automatically recommend products similar to those we have bought or viewed. Banks are increasingly automating the processes of detecting financial crime based on the processing of the data they collect with machine learning algorithms. Voice control of various types of devices, such as mobile phones, computers, TV sets, and household appliances, is also due to algorithms taught how to recognize the commands we issue correctly.

These algorithms are also used in more important aspects of our lives. In medicine, image processing with the use of algorithms that are able to detect or classify medical images is often used. In industry, many processes can also be optimized based on neural networks. This allows you to minimize energy losses, optimize various types of routes, or automatically detect objects of interest through camera image.

This work will place the greatest emphasis on topics related to image processing and classification of objects. These are topics that often appear in existing solutions on the market. You don't have to look far, the topic of autonomous vehicles, very popular lately, is based mainly on the analysis of the image received by cameras. Such a vehicle must be able to recognize objects in front of them and classify them into appropriate groups, such as pedestrians, passenger vehicles, trucks, cyclists. Must be able to recognize road lines or signs. Industrial and mobile robots also use such solutions. Of course, other solutions such as radars or lidars are also helpful, but implementing neural networks into camera images is an increasingly popular solution, especially at times when the computing power of computers is so high.

Machine learning is not a topic recently learned. The first attempt to present a nerve cell model took place as early as 1943. This is the McCulloch-Pitts neuron model. It worked similarly to a simple logic gate, which, after exceeding a certain value at the output, generated a binary output signal. A few years later, the first idea was presented on how to teach such a neuron model of certain behaviors by updating the values assigned to it, weights. The author of this idea was Frank Rosenblatt [1].

The most modern neural networks are nothing but improved and connected in huge clusters, neuron models invented by these scientists. The use of these models to solve specific problems, however, was not so popular at that time. Their use on a global scale began to return with the increase in computing power of modern computers. The possibility of parallel processing based on graphics cards had a great impact.

The essence of the problem

The basic division of machine learning algorithms is the division into supervised and unsupervised (they will be described in more detail later). Algorithms based on the classification of objects in the image are, in most cases, supervised algorithms. This means that they require you to provide a certain amount of tagged data (this number may be higher or lower depending on the problem). Algorithms that are based on previously provided data and associated labels, saying what is in a given image and in what place, can achieve really good results. The problem is the data labeling process itself. It is a tedious process, requiring large human resources, and thus it is a very expensive process. Whole groups of people, separate companies work on data labeling. The cost of tagging one object in the photo by manually determining the bounding box is about $ 0.05. It is estimated that the costs of data labeling in 2018 amounted to USD 316 million, and may reach as much as USD 1.6 billion in 2025 [2].

Purpose and scope of work

The main purpose of this work is to attempt to develop an algorithm that eliminates to some extent, the problem of labeling data that is a training set of neural networks that classify objects seen. The work will be of a research nature. It will be associated with an attempt to develop a new algorithm. The algorithm that will be implemented and described was an idea of the supervisor of this thesis, AGH prof. Adrian Horzyk, who is an expert in the field of research on neural networks, cognitive systems, and self-developing semantic associative memories. The work will describe the experiments carried out, those unsuccessful, with an attempt to explain what has failed, and the successful ones with a description of their accuracy. The developed algorithm should learn to separate the supplied dataset in the form of photos, without labels assigned to appropriate groups. Such a network should be able to recognize certain patterns, shapes, solidify certain objects in the 3D scene, without specific knowledge about what exactly is being taught. Several versions of the algorithm will be presented, technologies used, and scripts are written will be discussed.

1. Convolutional Neural Network

1.1. General information and parameters

Convolutional neural networks are the most important type from the point of view of this work. It is this type of network that will be used to teach patterns and to ground them on the 3D scene. This type of network is mainly used in image analysis and computer vision.

The use of several layers that extrude the characteristics of a given image has its justification in biology. By analyzing the image, the human mind also performs this process in a hierarchical manner, on several layers. One of them is responsible for the perception of simple shapes, such as lines, others discover the colors we see. At the end, the layers at the highest level are responsible for the recognition itself. Generally, deep nets operate on this principle, and image classification is also possible using classic deep nets. However, it should be remembered that in classical architecture, the image fed into the input had to be reduced to a one-dimensional pixel data vector. Such an operation disturbed information on the relative position of individual pixels, and thus we could lose information about specific shapes. Convolutional networks do not require input flattening [3].

Convolutional layers are nothing but filters used in other areas of computer vision. Many useful filter types have been discovered so far, e.g., Sobel filter for detecting edges in the image. Filters are matrixes of numbers, and such a filter and image perform the convolution operation (hence the name of this subcategory of neural networks). The difference is that the filters used in CNN are adaptively adjusted, the values and structures of these filters are not set to constant, the network is taught to adapt in order to extract the most important features. For this reason, earlier layers of such networks are often used as feature extractors, which allows, for example, to transfer the style of one image to another.

The way convolution works in the simplest way can be described as the elementwise multiplication of individual fragments of the input matrix with the filter used. Filters in a convolutional layer can be defined by several parameters:

- Size - it defines how many pixels of the input matrix will participate in the convolution operation, this is the width and height of the filter

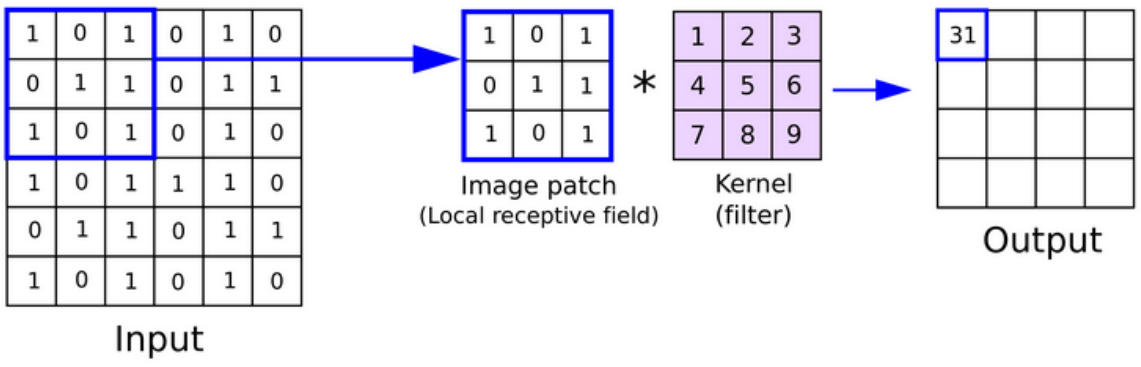

Figure 1: Convolution operation with the filter of size 3x3

https://anhreynolds.com/blogs/cnn.html

- Depth - the number of filters that make up the given layer. Each of the filters can be used to extrude a different feature, so it should be noted that the number of filters must not be too small, because such a network may not have sufficient resources to properly analyze the output image. Speaking of the input image, most often, its depth is '1' - for images in grayscale, or '3' - RGB Images. Therefore, depth can be described as the number of channels needed to properly characterize the image.

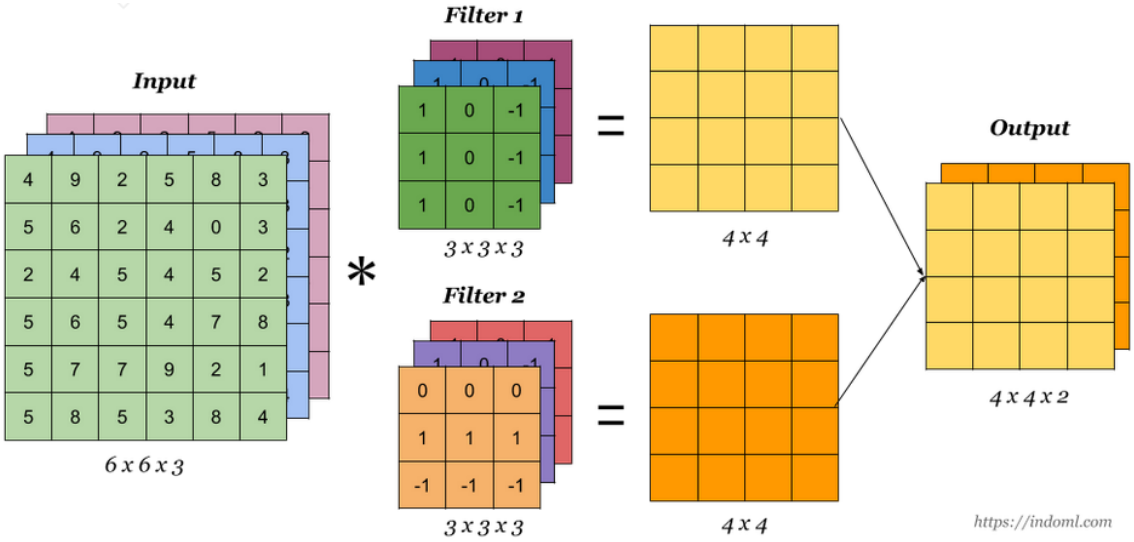

Figure 2: Output of the convolution with the filter with depth 2

https://indoml.com/2018/03/07/student-notes-convolutional-neural-networks-cnn-introduction/

- Stride - This parameter describes how many pixels the filter moves along the input matrix. Thus, the higher its value, the fewer convolution operations will be carried out, and thus the resulting matrix will be smaller.

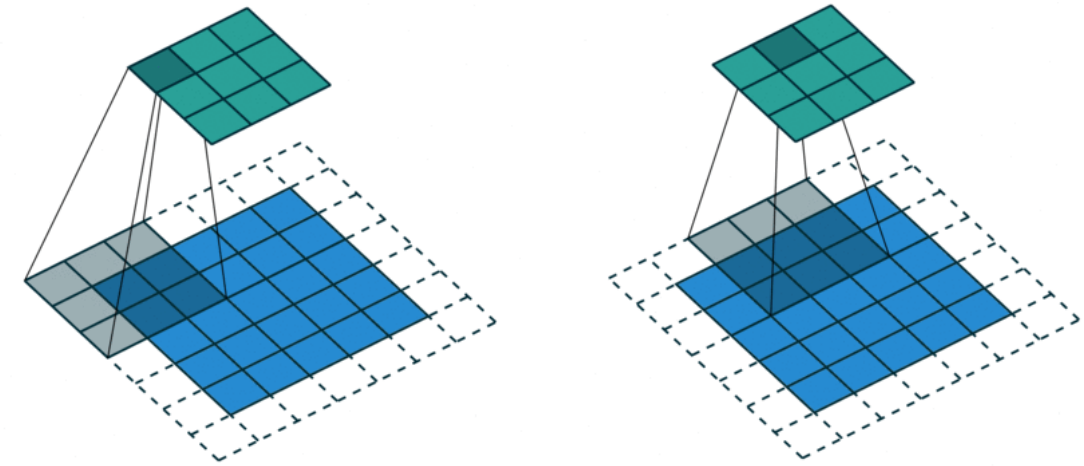

Figure 3: Illustration of convolving with filter and stride set to 2

https://www.quora.com/What-does-stride-mean-in-the-context-of-convolutional-neural-networks

- Padding - completing the input matrix with zeros on the edges. The number of zeros can be different. Padding is mainly used to manipulate the size of the output matrix. It also allows for more accurate data analysis on the edge of the image or input matrix, because by using padding, edge pixels are involved in more convolution operations [4].

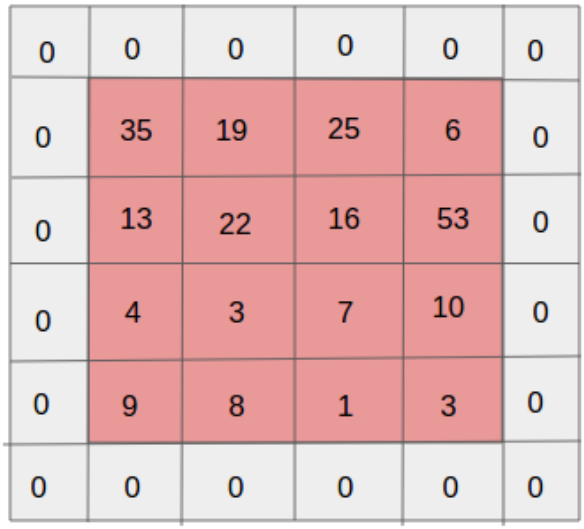

Figure 4: Example of padding of size 1

https://blog.xrds.acm.org/2016/06/convolutional-neural-networks-cnns-illustrated-explanation/

All of the above parameters should be taken into account when calculating the output size of the processed data. To determine this size, it is best to use the formula:

$$output\ size = \frac{W-F+2P}{S+1} \quad (1)$$

Where:

W – input matrix size (width or height)

F - filter size (width or height)

P – Padding value

S – Stride value

Pooling is another important operation used in the convolutional neural network. There are two main types: max-pooling and average-pooling. Max-pooling selects the most

relevant pixels (most enhanced) from a given neighborhood. The pooling operation reduces the size of the input matrix. Max-pooling selects the largest value from the selected pixel window of the input matrix. This allows reducing the impact of small changes, noise, in the vicinity of an important element. This operation also allows entering a small resistance to rotation or size changes of classified objects

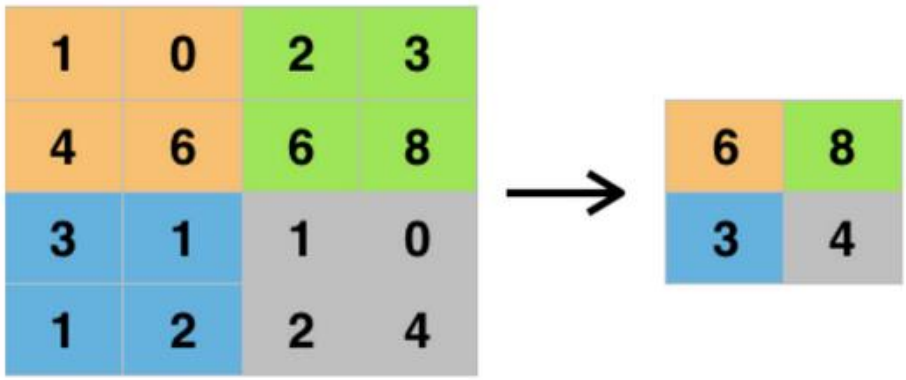

Figure 5: Max-pooling with the window of size 2 and stride 2

https://deepai.org/machine-learning-glossary-and-terms/max-pooling

The image given to the convolution network goes through a certain number of convolution layers and pooling layers. The output of the convolution layers is affected by a corresponding activation function (whose more detailed description will appear at a later stage of the work). The location of these layers, the size of the filters, their number, and other parameters are selected at the stage of architecture modeling. The stage when the image goes through the convolution layers is sometimes called feature extraction, as mentioned above. Then, after passing through the last convolution layer, the output is flattened to a 2D vector, and a classification stage based on the classic deep neural network follows. The most strengthen neuron at the output of the convolution network indicates the belonging of the input image to a given class.

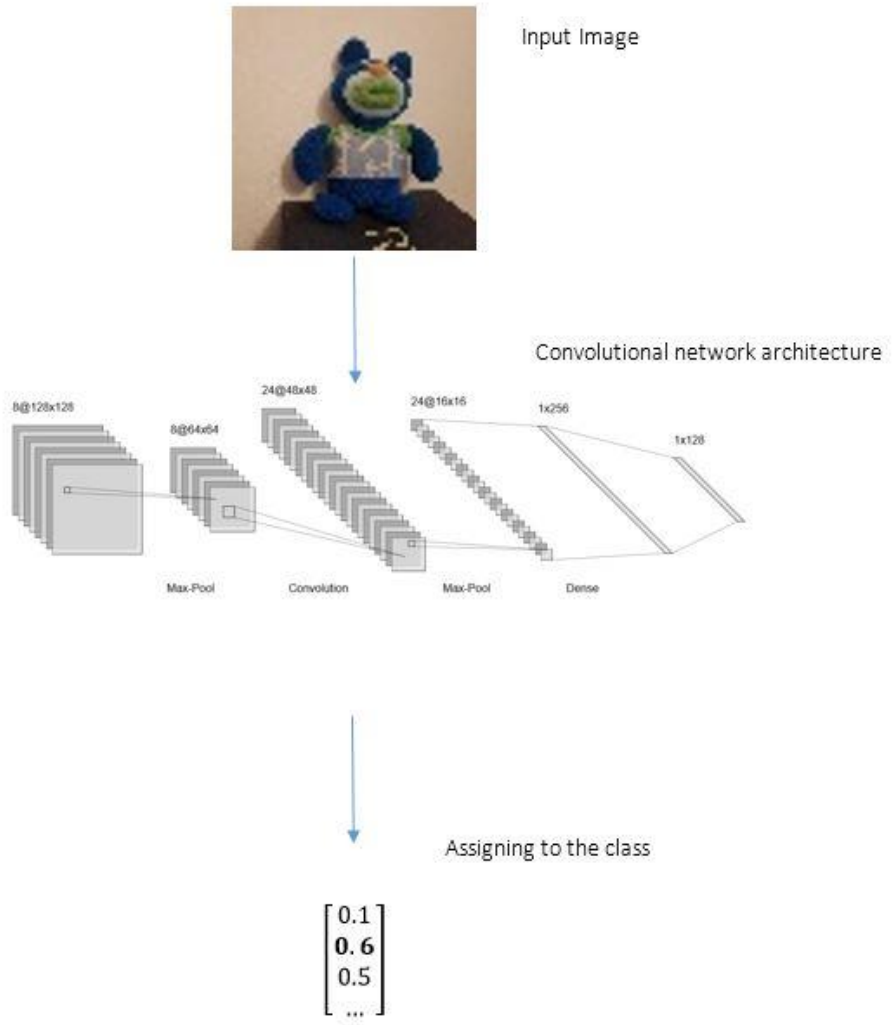

Figure 6: Simplified process of classification in CNN

1.2. Popular Convolutional Neural Network Architectures

The above description presents the basic elements that are part of the convolutional neural networks. However, the way they are combined can be very different, and the efficiency of detection or classification of objects in the image will depend very much on how we will arrange these basic elements. Below is a brief description of the example architectures.

1.2.1. VGG-16

The beginnings of creating convolutional networks were based on increasing the number of successive layers. Increasing computing power allowed this. Previous architectures such as LeNet-5 and AlexNet did not differ much from VGG-16. One change that was introduced at the stage of development of early convolution network architectures is the use of the ReLU activation function (Rectified Linear Units) - will be described later. VGG-16

consists of 13 convolutional layers. The classifier at the output of the last convolutional layer consists of 3 deep layers.

1.2.2. Inception

Inception networks have introduced quite a few new solutions. There are many versions of this architecture, but they are based on similar solutions. One of the novelties was the introduction of additional, smaller networks inside the main network. They occur on the basis of branches. This allows for the parallelization of architecture. In addition, 1x1 convolutions have been introduced. Such filters are mainly used to reduce the depth of the layer, i.e., reduce the number of channels.

1.2.3. ResNet

At some stage, scientists thought that thanks to the high computing power available, it would be possible to increase the number of convolution layers to very large amounts. Unfortunately, at some stage, there is a problem called 'vanishing gradient'. The problem is that derivatives of cost functions can achieve small values. These small values in deep architectures are multiplied many times. At the backpropagation stage, the multiplication of derivatives results from the so-called chain rule. This multiplication may lead to the gradient reaching very low values, and thus the weights will cease to be updated. To prevent this, something called 'skip connections' was invented (Fig. 7). They have been popularized in the ResNet network. These additional connections mean that even if the derivative reaches a small value, it is added to the value before it went through the convolution layers, which prevents it from completely zeroing. This solution allowed the use of even larger architectures (even over 100 layers). What's more, batch normalization has also been introduced, i.e., normalizations before entering each layer, to prevent situations where some activations reach very high values in relation to the others, which also had a positive effect on the learning process [5].

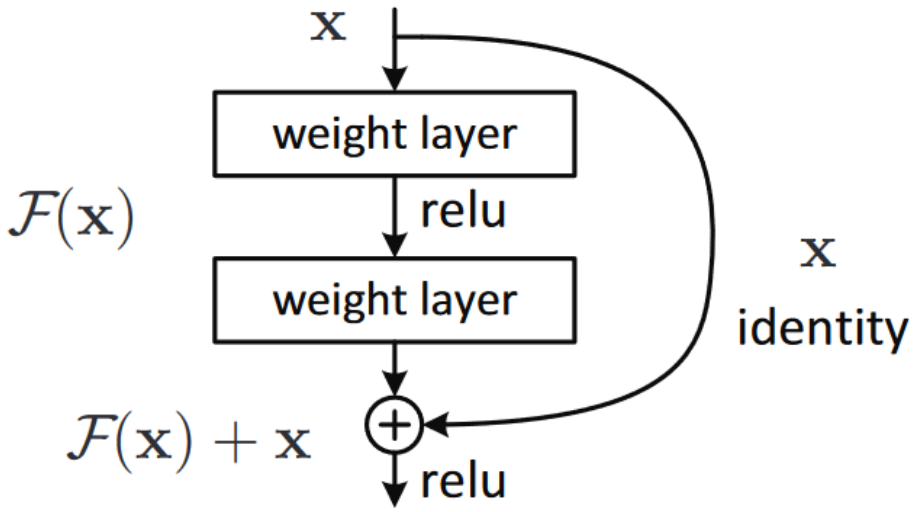

Figure 7: ResNet skip connection [6]

1.3. Summary

Neural networks can be used for many tasks. One of the main goals is image analysis in a broad sense. Thanks to the use of neural networks, we can classify objects on the image into various categories, we can detect these objects and segment the image into parts. All of these tasks can be performed using classic, deep neural networks; however, convolutional neural networks have greater accuracy, speed of learning, and real mapping in the biology of human image perception. This chapter presents the basic elements that make up such networks, explains how they work, and introduces some of the key architectures. The topic of convolutional neural networks is still widely developed, there are many aspects that have not been addressed, but for the purpose of understanding this work, the information provided should be sufficient.

2. Basic mechanisms used in the network learning process

This work will present a new approach to learning convolutional neural networks. In order to understand this algorithm, you need to know the basic mechanisms that allow you to train your neural networks. Most of the issues discussed in this chapter have references not only to CNN but also to classic neural networks.

2.1. Activation functions

Activation functions allow us to determine the activation of a given layer of the neural network. Let's assume the theoretical situation of a simple neural network with one hidden layer. The values in subsequent layers are calculated on the basis of the matrix product of the output of the previous layer, with a weight matrix, i.e., the values of connections between one and the other layer.

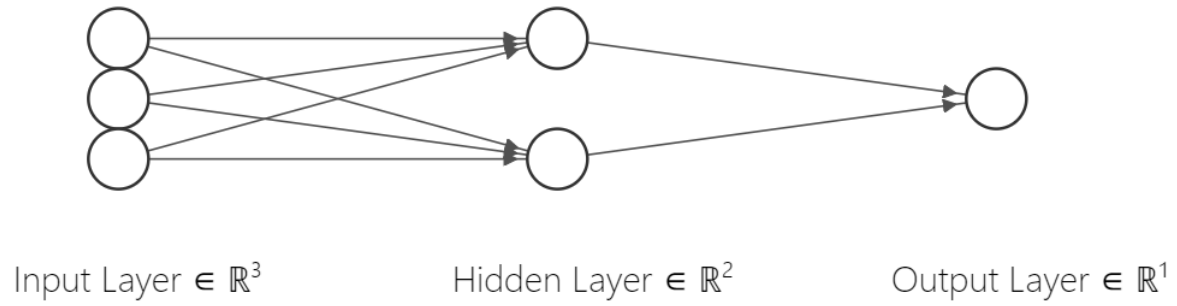

Figure 8: Example of a neural network with one hidden layer

I- Input layer – 3x1,

H- Hidden layer – 2x1,

O- Output layer – 1x1,

A_h-Activation of hidden layer – 2x1,

A_o-Activation of output layer – 1x1,

$W_{i\to h}$-Weights of connections between input and hidden layer – 2x3,

$W_{h\to o}$-Weights of connections between hidden and output layer – 1x2

To define what the activation function is, one must think about where we can get the value of the vector A_h.

The first step that should be performed is the multiplication of the input layer with weights matrix:

$$W_{i\to h} I = H \tag{2}$$

The result of this multiplication is vector H. To calculate the activation of this vector, we can use the activation function.

$$A_h = \emptyset(H) \tag{3}$$

$\emptyset(\cdot)$ is the activation function. In order to use the gradient method to update the weight values, it should be taken into account that the activation function should be differentiable. The most popular activation functions will be presented later in this chapter. One may be surprised by the fact that one of the most popular activation functions, ReLu, is not differentiable for z = 0. Theoretically, when our function hits the point z = 0, sub-derivative is used. In practice, however, calculations performed on a computer very rarely guarantee that the zero value will be represented by a machine exactly by this particular value. In addition, the algorithm implemented in software in such a situation will return a value that is a one-sided derivative of this function.

The activation function is applied to each element of the vector, in this way, we get the activation of a given layer, which can be used to calculate the values of the next layers or to return the network output.

2.1.1. Sigmoid Activation Function

Sigmoid is an S-shaped function. Its popularity results from the fact that the range of values it achieves is in the range (0, 1). This range is suitable, e.g., for classification problems, as it well reflects the probability values of belonging to a given class. So, by using this activation function, we can map each value of the vector of a given layer to a value in the range (0, 1).

Sigmoid Function is given by the following equation:

$$\emptyset(z) = \frac{1}{1+e^{-z}} \tag{4}$$

Advantage of the sigmoid function is the fact that its derivative is calculated in a very simple way:

$$\emptyset'^{(z)} = \emptyset(z)(1 - \emptyset(z)) \tag{5}$$

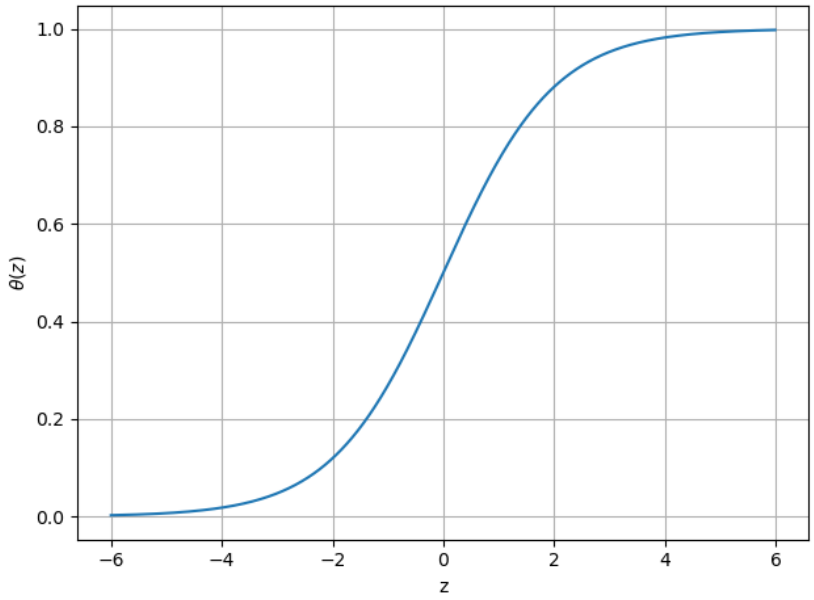

Figure 9: Sigmoid Function

Speaking of Sigmoid Function as an activation function, it is worth mentioning the problem of vanishing gradient. This problem occurs at the stage of backpropagation. By using gradient algorithms that minimize the cost function, we calculate partial derivatives of the cost function and the activation function of previous layers. Then, according to the chain rule, these derivatives are multiplied by themselves. The graph of the first derivative of the sigmoid function (Fig. 10) shows that it takes very small values some distance from the point z = 0. This causes a problem with the gradient disappearing to values close to zero, and hence the weights are not updated.

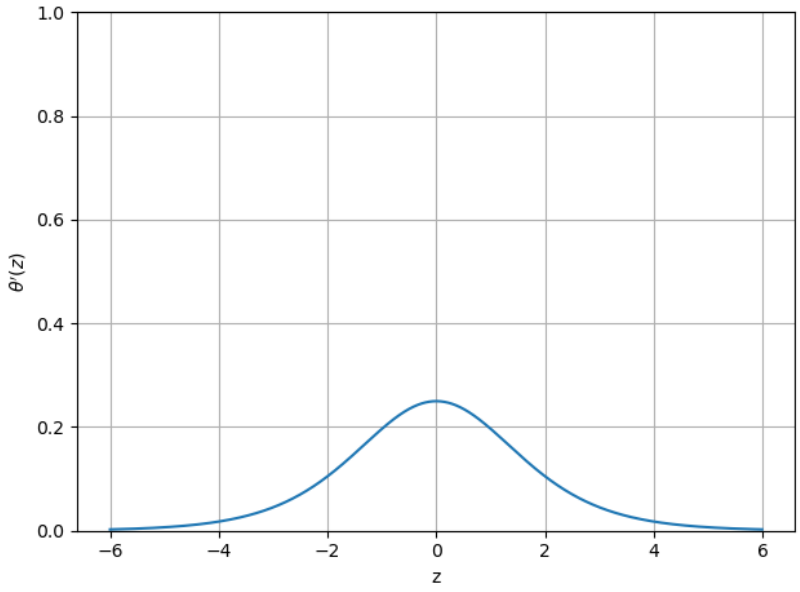

Figure 10: Derivative of a sigmoid function

2.1.2. Hyperbolic Tangent Activation Function

The main difference between sigmoid function and hyperbolic tangent function is that hyperbolic tangent has values in range (-1, 1). It is also s-shaped, so we can assume that this function is ,,zero-oriented". It maps negative values to the value near -1 and positive values to the value near 1.

$$\emptyset(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{6}$$

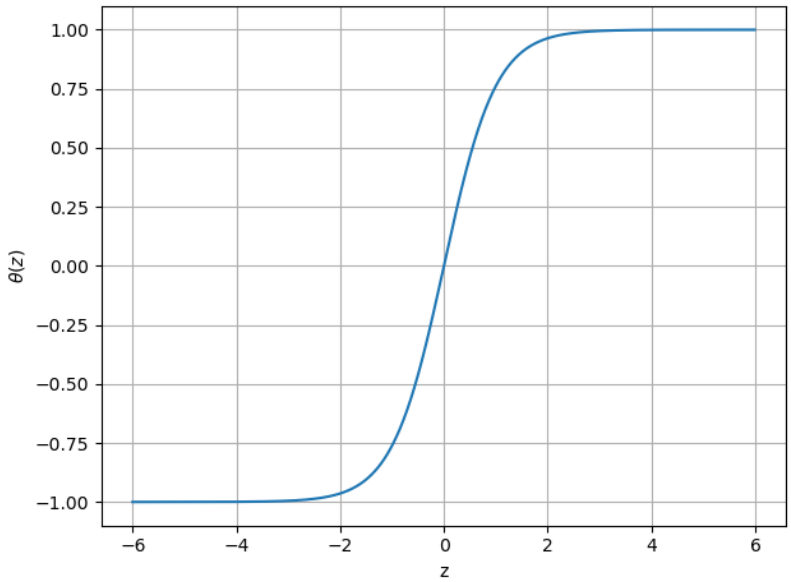

Figure 11: Hyperbolic Tangent Function

2.1.3. ReLu Activation Function

The ReLu activation function is a seemingly simple function. Despite its simplicity, it is one of the most-used activation functions, especially on CNN. The use of this function is computationally effective and allows for fast convergence.

$$\emptyset(z) = \max(0, z) \tag{7}$$

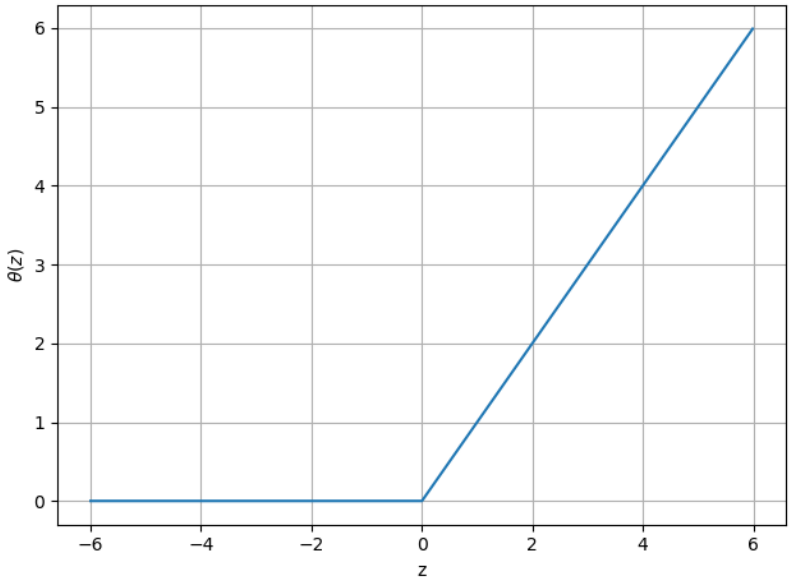

Figure 12: ReLu Activation Function

It is worth mentioning that from the point of view of the convolutional network, the use of this activation function has another advantage. It introduces nonlinearity. The convolution operation itself, as the matrix multiplication operation, is a linear operation. The ReLU function equates all values less than zero to a value equal to zero.

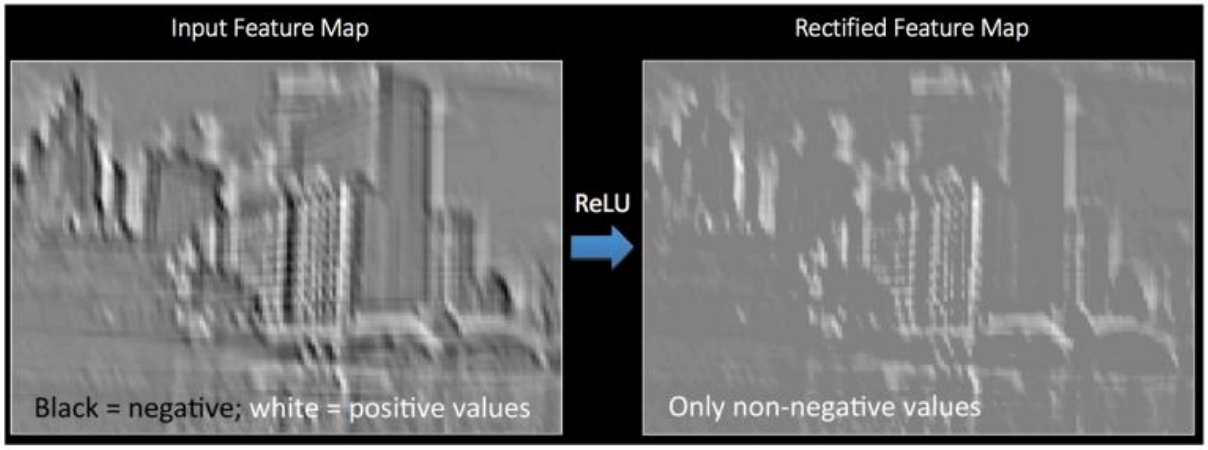

Figure 13: ReLU applied to an image

https://pydeeplearning.weebly.com/blog/basics-of-convolutional-neural-networks

2.1.4. Softmax

The last-mentioned function is important from the point of view of classification problems for more than two classes. It is most often used as the activation function of the last classifier layer. Its basic advantage is that it maps the exit from the network to the vector of

probabilities of belonging to a given class. Thus, each value in the output layer vector is in the range (0, 1), and all values add up to 1. For the output layer, which has to classify input data to K classes, we can calculate Softmax activation of *j*th element in this layer by using the following equation:

$$\emptyset(z_j) = \frac{e^{z_j}}{\sum_{k=0}^{K} e^{z_k}} \tag{8}$$

2.2. Loss Functions

In the process of learning the supervised algorithm model, we get some output. This output may be more or less similar to the value we expect. Assuming that we know the desired value, i.e., the label assigned to the training data, we can assess how close the output of the network was to the true value. We can calculate this difference using the loss function. It takes the output and the exact value as arguments and calculates the error on this basis. The whole learning process boils down to obtaining such parameters that will make this error as small as possible. Therefore, we are dealing with minimizing the loss function. Depending on the problem, this function can be selected differently; however, its selection can significantly affect the accuracy of our algorithm. Gradient methods are used to look for such minima. The gradients of the cost function are giving us the direction of decline in these functions, i.e., the direction that should be followed to find the minimum. It is important to remember about the selection of appropriate parameters, such as the learning rate, in other words, how big a step will be taken towards the minimum.

2.2.1. Mean Square Error

We will often meet cases when we want to predict the value of some continuous variables. It can be a change in temperature, the price of shares on the stock exchange, etc. Then our network or other model returns to us values that are predicted values in the future. Assuming that we know the exact values for a given time interval and the predicted values for the same interval, we can calculate our prediction error using, e.g., mean square error loss function. Calculation of this error is simple and is based on a comparison of predicted value and real value in some specific periods of time.

Assuming we have *n* to compare:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}({y_{real}}^{(i)} - {y_{pred}}^{(i)}) \tag{9}$$

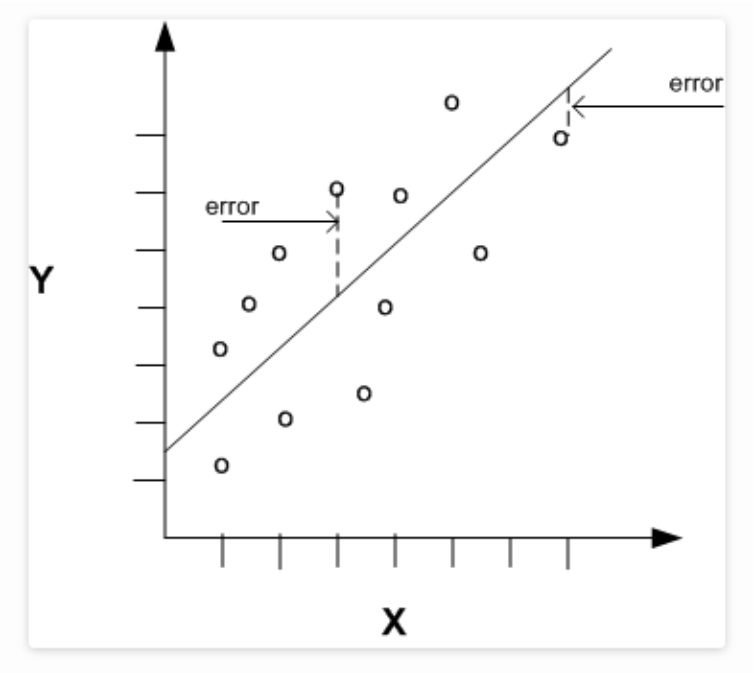

Figure 14: Mean Square Error

https://educationalresearchtechniques.com/2015/11/

2.2.2. Cross-Entropy Loss

This loss function will be used in our model. It is useful in situations where the output of our model has a vector of probabilities of belonging to a given class. For this reason, before using this loss function, it is best to treat the output vector first with the Softmax function described earlier.

For *n* classes we have:

$$entropy = -\sum_{i=i}^{n} prob_{real} \cdot \ln(prob_{out}) \tag{10}$$

For example:

Predicted: $\begin{bmatrix} 0.7 \\ 0.1 \\ 0.4 \end{bmatrix}$ Real: $\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$

$$entropy = -1 \cdot \ln(0.7) \approx 0.357$$

2.3. Optimizers

The previous subsection mentioned loss functions. In order to teach our network to react appropriately to the input provided, we must somehow minimize the loss function so that the

error is as low as possible. Backpropagation provides us with information on the gradients of the functions used. Optimization allows us to update the weights of our network based on these gradients. Various combinations of mini-batch stochastic gradient descent algorithms will be used in this work. This means that at the learning stage, certain subsets, batches of the entire training set will be used, and based on them, the weights will be updated. The entire set will be used only in situations where it was possible due to the RAM used. The batch stochastic gradient optimization methods allow for training more efficiently than training on single samples and give better results than training on the whole set. They also allow for greater disregard of local loss functions. The most basic version of the gradient descent method is to take a step in the opposite direction to the loss function gradient. This way, we approach the function minima by making a step proportional to the learning rate value. It is worth noting that a good technique is to reduce the learning rate parameter at the learning stage, which ensures better algorithm convergence. Simplified version can be described by the given equation:

$$w_t = w_{t-1} - lr\nabla J(w_{t-1}) \tag{11}$$

w – weight

lr – learning rate

J – loss function

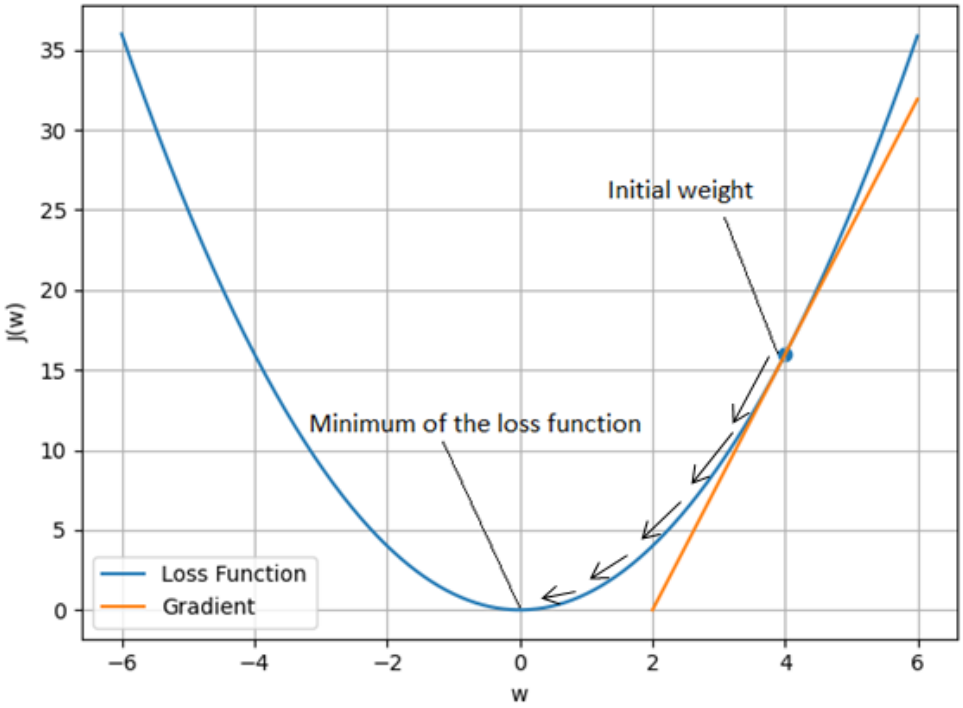

Figure 15: Gradient Descent Algorithm Visualization

2.3.1. Adagrad

The algorithms that optimize the cost function can be modified to make them work more efficiently. Improved performance can mean accelerating the learning process or slowing convergence near local minima.

The main differences in the operation of the discussed algorithms will concern the updating part $-lr\nabla J(w_{t-1})$ from the equation quoted above. For the sake of description, let's write this part as $\Delta(t)$.

Adagrad is one of the optimizations of the SGD algorithm. The learning rate of each parameter is adjusted based on the L2 norm of the previously calculated gradients.

$$\Delta(t) = -\frac{lr}{\sqrt{G_t+\varepsilon}}\nabla J(w_{t-1}) \tag{12}$$

From the formula, we can see that the initiation value of the learning rate is being changed.

Matrix G is a diagonal matrix with dimensions dxd and element i is the sum of squared gradients for iteration $t-1$.

$$G_{t,i} = \sum_{k=1}^{t-1} \nabla J(w_{k,i})^2 \tag{13}$$

The ε parameter is a very low-value factor that is added to avoid division by zero [3].

2.3.2. Adadelta

One of the problems with using the Adagrad algorithm is the fact that the denominator Δ(t) accumulates gradients and increases with each iteration. This large denominator means that the learning rate value tends to very low values. Low learning rates mean that the learning speed is very low and can slow down almost completely.

The Adadelt algorithm tries to deal with this problem. Rather than accumulating gradients from all previous iterations, only a certain number of them are considered. They are not remembered; the incremental degradation is recursively determined using the mean value at time t. The matrix G is replaced with the mean degradation from past gradients.

$$E[\nabla J(w)^2]_t = \rho E[\nabla J(w)^2]_{t-1} + (1-\rho)\nabla J(w_{t-1})^2 \tag{14}$$

ρ is a degradation constant, and its value is usually between 0.9 and 0.999.

We can now set a new way to calculate the learning rate [3]:

$$\Delta(t) = -\frac{lr}{\sqrt{E[\nabla J(w)^2]_t+\varepsilon}}\nabla J(w_{t-1}) \tag{15}$$

2.3.3. Adam

Adam differs from Adadelta in that, in addition to taking into account the previously squared gradients, it also takes into account the gradients themselves, not squared (which is similar to some other algorithms like Momentum).

To use Adam optimizer, we need two parameters:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)\nabla J(w) \tag{16}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2)\nabla J(w)^2 \tag{17}$$

m_t parameter is called the first moment, and the m_t parameter is the second moment.

β value is usually between 0.9 and 0.999.

The authors of the algorithm have corrected the parameters, and their final form is as follows:

$$\widehat{m}_t = \frac{m_t}{1-\beta_1^t} \tag{18}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t} \tag{19}$$

After these parameters are calculated, the weights are updated by calculating the term presented in the previous algorithms [7]:

$$\Delta(t) = -\frac{lr}{\sqrt{\hat{v}_t + \varepsilon}}\widehat{m}_t \tag{20}$$

2.4. Summary

In this chapter, we learn more about the mechanisms used in training deep neural networks and CNN. Various activation functions are discussed. Some of them will be used in models related to this work. These are ReLU for the intermediate layers and Softmax for the classifying layer. We also got to know two of the basic loss functions. One of them, MSE, is used more often for regression tasks. The second, Cross-Entropy Loss, will be used in this work as it is better suited to classification issues. Finally, the more mathematical aspects of training, i.e., optimizers, are discussed. Several formulas for the modification of the SGD algorithm were given. At the experimental stage, their effectiveness will be compared.

3. Basic Division of Machine Learning Algorithms

From the point of view of this work, this chapter raises a very important issue. It will list two basic groups of machine learning algorithms, Supervised and Unsupervised, and another group that, in a way, combines the features of the first two, i.e., semi-supervised. Examples for individual groups and their characteristics will be given

3.1. Supervised Learning

Supervised learning is based on one main premise. The algorithm at the learning stage accepts data that is labeled in some way. These can be, for example, photos with an assigned class, i.e., information that a given photo presents. Supervised learning is based on the fact that the algorithm learns on the labeled data and then can make assumptions about unlabeled data. Unlabeled data is data that has not been classified in any way. The basic, often cited example taken from everyday life is the situation when we have a collection of many messages in our electronic mailbox. Assuming that for a certain period of time, we marked each message as spam or as a useful message, supervised algorithms would be able to learn with some efficiency to classify whether a given message is spam or not.

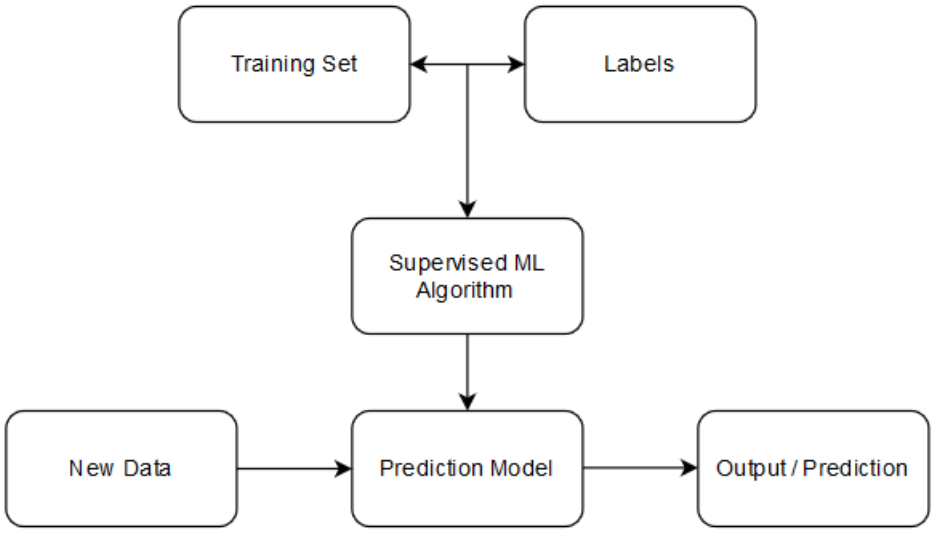

Figure 16: Supervised Algorithm Flowchart

Supervised learning can be used both for classification tasks (such as classifying an image into a given category or anti-spam filter) and for regression issues when we want to predict certain continuous values (such as house prices over a period of time or share prices). In the first case, the labels are specific values assigned to the training data. In the case of regression, a set of continuous values over a given time interval can serve as input. Knowing the values for a certain time, the algorithm should learn to predict trends and the further time course. We can also use supervised algorithms to predict numerical values for a given set of input data, such as a person's life expectancy based on their illnesses and lifestyle.

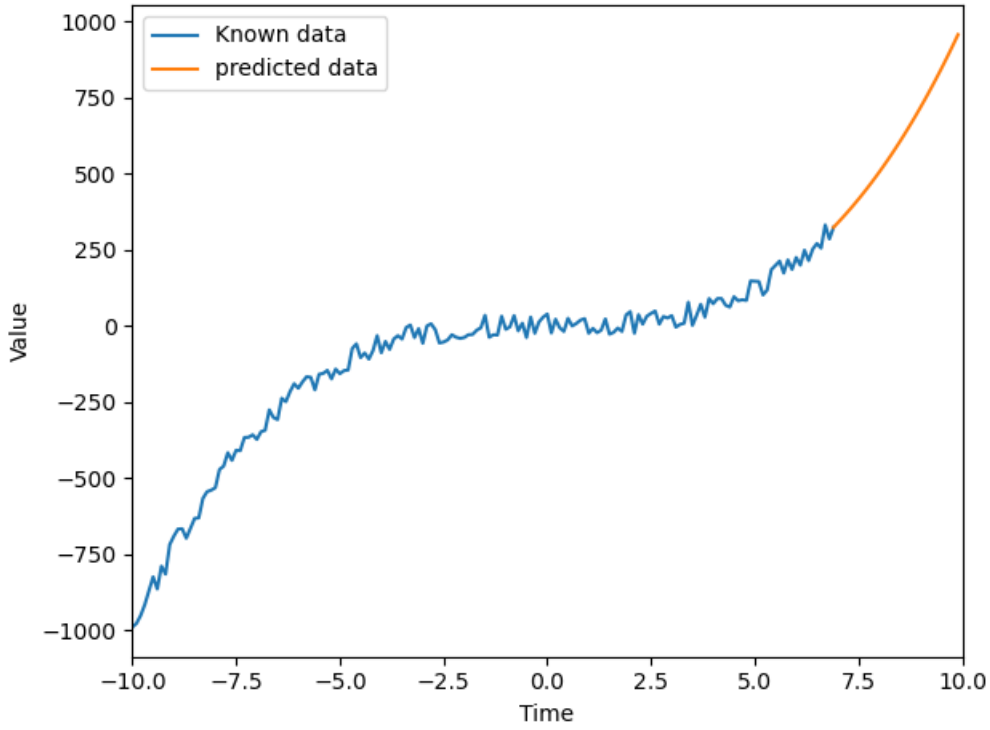

Figure 17: Supervised Algorithm for regression task

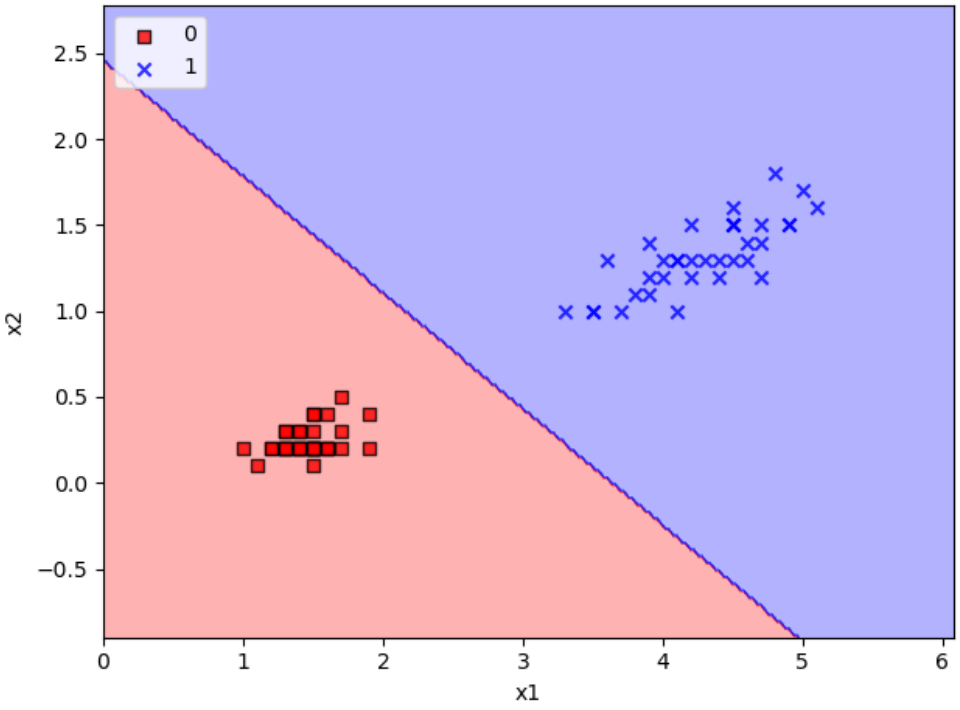

Figure 18: Supervised Algorithm for a classification task

In regression models, we can distinguish several basic algorithms. One of them is linear regression. A linear regression model fits a straight line to the provided data and predicts unknown data. Mean-Square errors discussed earlier may be a frequently used cost function for this type of algorithm. We can introduce regularizations to the regression model to avoid

overtraining the model, i.e. when the model provides a very small error for the training data but does not generalize well for the unknown data. We are then dealing with the Ridge model in the case of the L2 norm and the Lasso model in the case of the L1 norm. Neural networks, such as Recurrent Neural Network or their modifications, such as Long Short-Term Memory, also work well for time series problems.

When it comes to supervised classification algorithms, we can indicate many solutions that work better in various situations. The logistic regression model allows you to assess the probability of belonging to a given group. The k-nearest neighbor algorithm classifies new data based on a selected distance metric to labeled data supplied to the algorithm. The KNN algorithm does not learn but performs a complete check for each new sample. The Support Vector Machine algorithm, although first used in the classification of linearly separable data, thanks to the introduction of regularization and the kernel method, i.e., transforming a data set from *n*-dimensional space to *n+1*-dimensional space, works very well in the classification of linearly non-separable data. The majority of neural networks also belong to supervised algorithms. The weights are tuned in learning and inferring from labeled data. The most important algorithm from the point of view of this work, Convolutional Neural Networks, in its basic version, also belongs to this group of algorithms.

Supervised learning usually produces the best results. It allows for very accurate clustering, regressions, or analysis of new data. However, as mentioned before, it is often very problematic to obtain a large amount of labeled data. Data labeling is a time-consuming and costly process, so sometimes it is worth looking at other available techniques or trying to come up with new algorithms, as will be presented in this paper.

3.2. Unsupervised Learning

Unlike supervised learning, unsupervised learning does not deal with data about which we have specific information. We do not have labels assigned, and we do not know the exact structure of these data. Unsupervised learning aims to recognize a certain structure of the provided data, group them into clusters, and extract some information from them.

Unsupervised algorithms can also help detect specific features in an image and classify those images according to the features detected. Boltzmann machines, for example, operate on a similar principle. One of the simplest and most popular unsupervised algorithms is the K-means algorithm. It uses the selected distance metric to determine the centroids of the subsets of a given set. Then, new data is assigned to a specific group based on the distance to the nearest centroid.

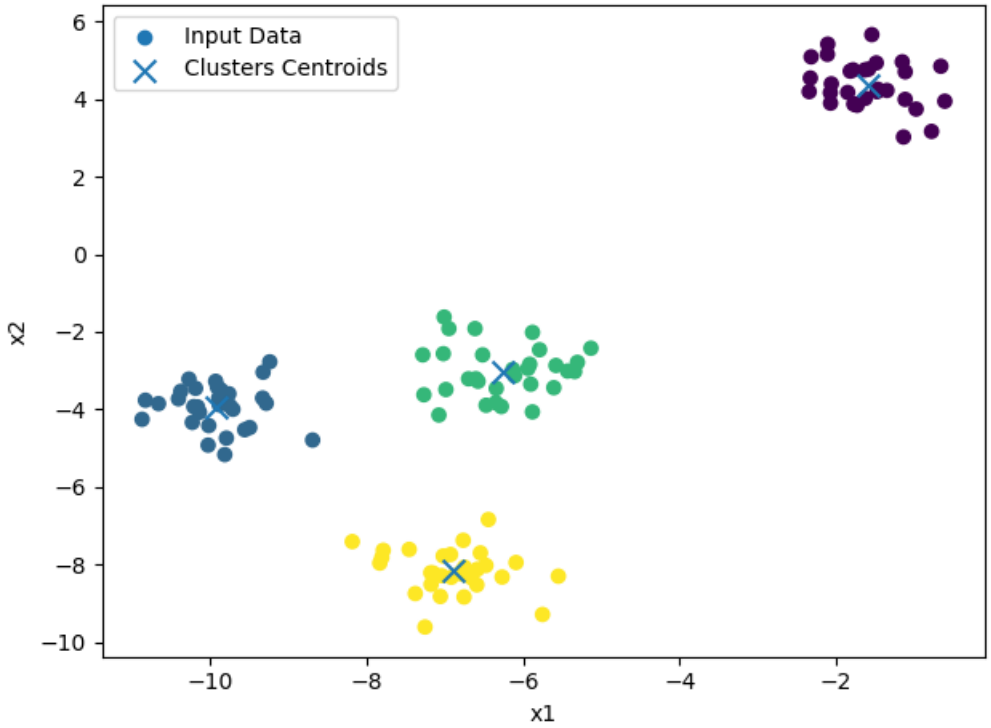

Figure 19: Data clustered with K-means algorithm

When we are dealing with more noisy data, or we do not want to assume in advance the number of clusters we are looking for, we can use the Density-Based Spatial Clustering of Applications with Noise (DBSCAN) algorithm [8]. It allows you to avoid outliers when making a classification. It is also a good idea to visualize the data as much as possible before choosing an algorithm, as not everyone will be good for every data type. An example can be seen in the figure, where the K-Means algorithm does not work as a classifier of a given set.

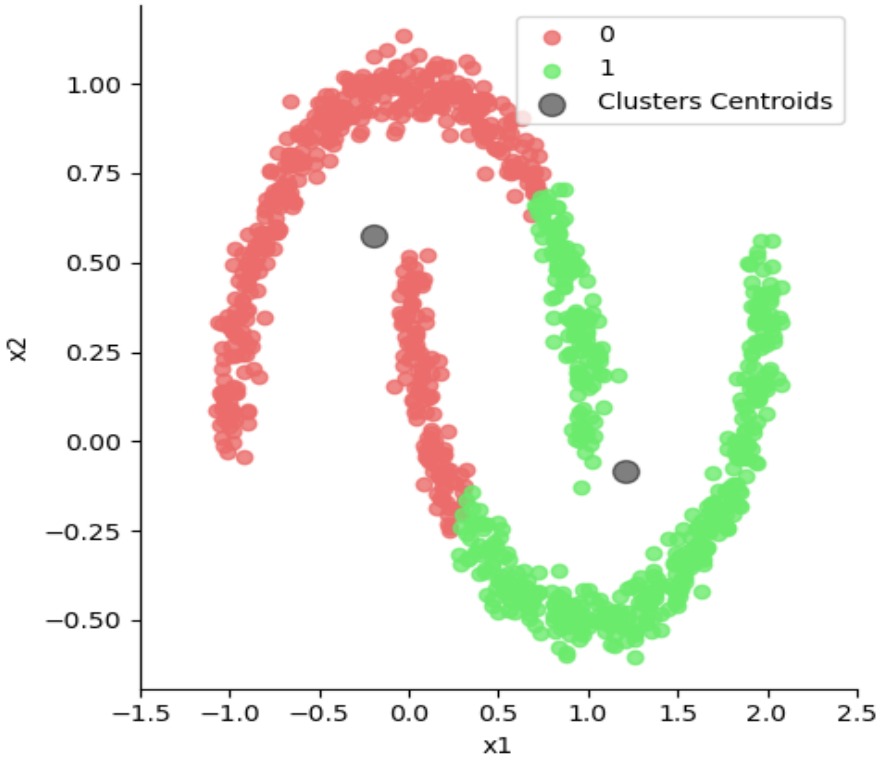

Figure 20: Badly classified data by K-means

Another interesting application of unsupervised learning may be the use of autoencoders. This type of neural network can also be used in computer vision. One application may be when we teach the autoencoder by inputting images that are encoded and then decoded to the original form. An autoencoder trained in this way can be used, for example, in the detection of anomalies. For example, when giving a certain input image that was used during training but with some change (e.g., additional character, potential thief), the network will return a big error as the input image will be very different from the image at the output.

3.3. Semi-supervised learning

In everyday life, the data provided very rarely have labels prepared that we could use for training. There are, of course, the unsupervised algorithms described in the previous section, but they are not ideal for all problems. For example, the classification of photos based on the presented algorithms can be complicated, as most of these algorithms are based on Euclidean distance measurement. One of the unsupervised learning methods is the automatic generation of labels for later use in the training phase. There are different ways to achieve this goal. They can be based on adding labels based on the shape of the data distribution [4]. Some algorithms are based on delivering some labeled and some unlabeled data to the algorithm. The method called self-training is based on the use of one of the classic machine learning algorithms, except that for unlabeled data, semi-labels are assigned. Semi-labels are assigned based on the similarity of the unlabeled data to labeled. Based on this similarity, a whole set

of labeled data is created, which is then used to train the base model [4], [9]. Sometimes combining several unsupervised techniques can produce good results in terms of image classification. One example would be the use of multi-autoencoder and k-means++ to cluster images of datasets [10]. Another way might be to try to generate new labels for unlabeled data using Generative Adversarial Networks [11]. A common algorithm that appears in scientific papers researching ways to unsupervised image segmentation or classification is the fuzzy c-means algorithm [12] - [13]. This algorithm, allowing for the clustering of objects into more than one group, is often used in the analysis of medical images. It can also be combined with other techniques, such as convolutional neural networks, using partially labeled data. The class distribution methods can work as follows: we feed an image to the input of the network, and the network returns us several probability values for belonging to a class. Then this probability distribution becomes something like a label. Noise is added to the image (adversarial method), but the label in the form of the class distribution is still assigned to this new image with noise [14].

An algorithm that is created for this work can also be assigned to this category. For simple datasets, you may not supply any data with labels at all. They are created during the training phase and then used as normal labeled data. For more complex datasets, we can create a network on a small set of labeled data and then use the automatic label generation, which will be described later in this work.

3.4. Summary

The chapter presents the basic division of machine learning algorithms. Supervised algorithms usually perform better, however, they require better-prepared data, labeled in some way. Unsupervised algorithms are often used for classification tasks. They do not require labels but work by looking for similarities between individual samples. The similarity may be assessed, for example, by calculating different distance metrics. A method that combines both of these methods, i.e., semi-supervised learning, can work well in situations where we plan to use algorithms similar to the supervised ones, but we have a limited amount of data from labels. Then we can infer the labels somehow for the unlabeled data based on the labeled data, or generate new labeled data.

4. New unsupervised learning approach of CNN for clustering of images

Along with the proposal of the new approach and the idea presented by the promotor of this work, the object clustering and classification problems can be solved in an unsupervised manner, i.e., without given labels, only based on similarities of the objects in many images. This new approach and the goal of further described experiments were to train a convolutional neural network (CNN) in a supervised manner but without labels. The target of training will be established based on the largest output value, instead of the given labels. The reinforcement of such largest outputs for training data should lead to the development of the representations of similar features in the CNN and the representation of the similar images by the output neurons. These research ideas were implemented in this work, and the experiments were prepared to check this hypothesis and the value of this approach.

The starting point for writing this work was also the created mobile robot. This robot, which is the subject of another diploma thesis, is fully functional and controllable thanks to the transfer of commands to it from the laptop. However, the control based on the commands 'turn left' or 'go straight' does not seem to be an ambitious solution. It would be a good idea to implement an algorithm that would allow, for example, to search for an object passed in the form of instruction. Such an algorithm would not expect an exact instruction of in what direction to move, but only information about to which object should it go or what obstacles to avoid. The robot was not based on LIDAR or radar solutions, but on the vision transmitted from the camera. In such situations, the first solution that may come to mind is Convolutional Neural Networks. To detect a certain group of objects, you can, of course, use a pretrained classifier, available in many basic machine learning libraries. However, when we are dealing with specific objects, the classifier should be trained from scratch. At this point, there is a problem where to get the appropriate amount of data with labels, which will be enough to train such a classifier. Unsupervised or semi-supervised learning may be a good solution. The idea was to provide the algorithm with photos of the objects of interest, and then let the algorithm learn to recognize certain patterns on its own and assign them to appropriate classes at the output of the network. A detailed description of the algorithm prepared in this way will be presented later.

4.1. Camera

The robot was equipped with an Intel Euclid camera. This camera is a very complex device that allows, among other things, to assess the depth or detect motion. It has high-class equipment for such a device. Among others, Intel Atom x7-Z8700 Processor, 4GB RAM, and 32GB storage. It has a built-in Wi-Fi module that facilitates communication. It is also equipped with pressure, distance, and GPS sensors. The big advantage of the device is the installed Ubuntu operating system. Working with a camera requires more programming knowledge than a typical device for taking pictures, but thanks to this, it gives much more possibilities. In order to work with the camera, it must be in the same subnet as the device on

which we want to operate it. In the case of the experiments for writing this work, it was a laptop. When the laptop and the camera are on the same subnet, you can enter the IP address given to the camera by DHCP in the browser window. Then we get access to the basic functionalities of the camera, view preview, and built-in scenarios based on pre-installed Robotics Operating System (ROS). To be able to create scripts and operate them from the camera level, you need to connect to it remotely. This can be done, e.g., with VNC. When we receive the image from the camera on the laptop, it can be used as an ordinary computer. By default, it supports scripts written in Python 2.

Figure 21: Camera used to create a dataset

4.2. Dataset

For the purposes of the work, a script for automating photo taking has been written. This script used the Rospy library, which was used as a communication interface with camera components. In the main part of the program, the subscription of the appropriate node is established. For the needs of the dataset being created, it was a connection with an RGB camera. The OpenCV library was used to save images in the form of a file. It is a very powerful tool for the broadly understood computer vision. The written script has been equipped with parameters handling. The user calling the script in the console can give arguments such as the number of photos we want to take and the name of these photos:
python make_photos.py –name photo –num_photos 30
Photos are then automatically taken, numbered, and saved.

Preparing the dataset assumed taking a certain number of photos of objects belonging to three categories. The following elements were selected for the experiment: teddy bear, mug, and bottle. After taking pictures in different positions and on different backgrounds, the pictures from the camera were transferred to the laptop. Changing the position of the objects

to take each photo was a tedious task, so the data augmentation technique was used. Neural networks, in most cases, recognize the same object in a different position on the scene or a rotated object as something else. Therefore, in a situation where we have little data, it is worth trying to add synthetic data that will be created thanks to the data already possessed. This technique is also often used in situations where our data sets for individual classes differ significantly in the number of elements. Often, better results are achieved for equilibrium sets, so it is worth augmenting the data so that there is the same amount of data for each class. In the case of the conducted experiment, the aim was to increase the number of training examples. At the stage of taking photos from the camera, 50 photos from each class were taken. Then a script was written to augment this data. This script took as a parameter the folder where the images to be augmented are. It also took the number of photos to be generated. Arguments for generating photos were rotation range of 9 degrees and zoomed in / out by a factor of 0.15. Script then read data from a given folder, generated new data using the ImageDataGenerator method from the Keras library, and saved the newly generated data. This way, the size of the training set was increased.

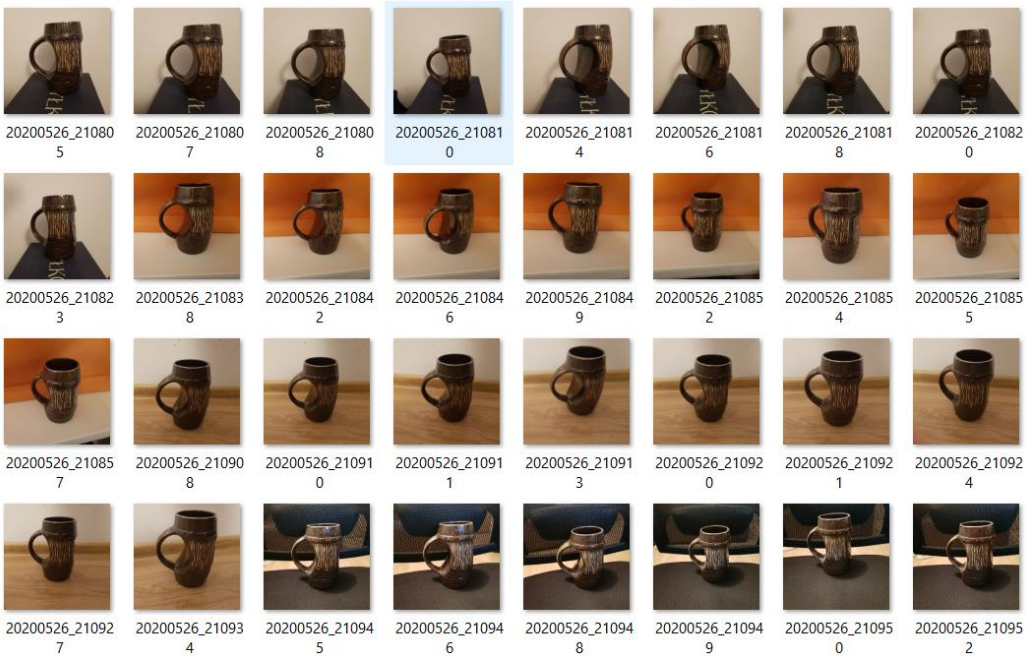

Figure 22: Created dataset - mug photos

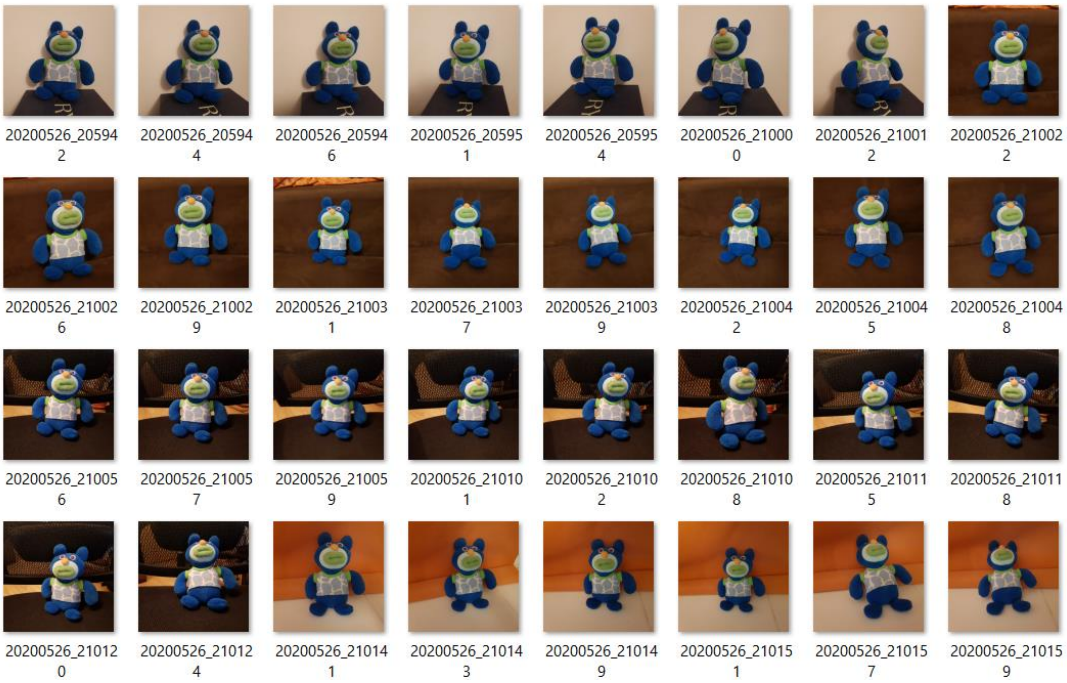

Figure 23: Created dataset - teddy bear photos

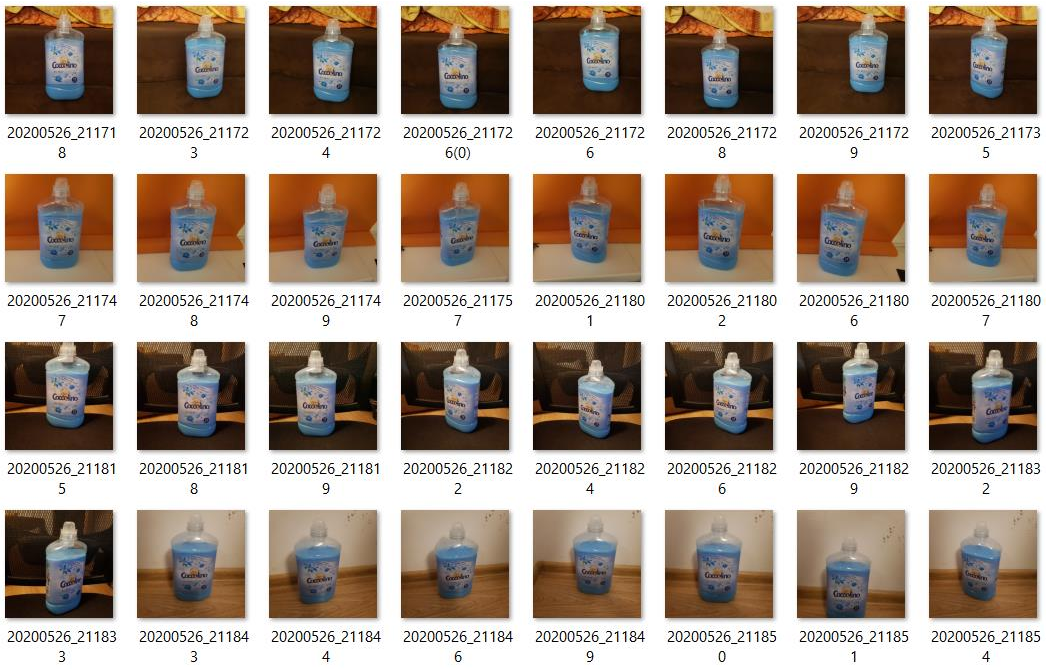

Figure 24: Created dataset - bottle photos

4.3. Summary

This chapter briefly describes the specifications of the camera used and how to create a dataset for the experiment. In fact, the data sets used in practice consist of a much larger number of samples, but such a set is sufficient for the purposes of proving the thesis. The data augmentation method is described, which allows you to generate new training samples based on those already existing. It is important to remember that data usually requires some

preprocessing before use. In order to prepare the data for the appropriate form, the methods from *Torchvision* packages (a component of the *Pytorch* library) were used at the learning stage. The Transforms method allows for appropriate data processing before feeding it to the model. We can automatically convert the training samples to the form of tensors acceptable by the models created in *Pytorch*. We can also automatically resize photos and perform many other operations, such as increasing contrast, rotation, mirroring, etc.

5. Description of the algorithm

5.1. Algorithm classification

The theoretical part of this work discusses the basic aspects of convolutional neural networks. The differences between supervised and unsupervised learning are cited. Methods with features from both of these network training styles, i.e., semi-supervised learning, were also mentioned. The algorithm proposed in this work may be classified into the last group. The initial assumption was to create a neural network in the shape of a classic supervised, convolutional neural network, which will be able to recognize certain patterns and clustered objects without the information about labels. This approach has many advantages. One of them is the simplicity with which the network, thus trained, can be used for further training or classification.

At the beginning of the CNN supervised training process, the parameters of the network are usually randomly initialized. Therefore, we can expect output values based on such random parameters. However, similar input images can stimulate similar neurons in the network the most and can produce similar output stimulations as well. On this basis, we can define a new unsupervised training approach that will reinforce the largest output calculated for every input image. This idea allows us to train CNN networks without labels of classes because we do not train a certain class at a certain output position (neuron) but let the network decide which neuron will represent a group of similar input images. The rule of this approach is straightforward. If a neuron in the output layer achieves the maximum output value, then this neuron is reinforced by the backpropagation algorithm (classically used for supervised training of CNNs).

5.2. Algorithm workflow

If the unlabeled image produces values 0.2, 0.5, and 0.1 on the three output neurons (Fig. 25), the second output neuron has achieved the highest output value 0.5, then this neuron will be reinforced to represent this input image. In such a way, the presented algorithm dynamically sets the desired label for this image as [0, 1, 0]. It turns out that the similar inputs during such a training process start to stimulate the same output neurons still more and more frequently, creating representations of clusters of similar inputs. In such a way, the input images will be clustered by the CNN network using a supervised training algorithm applied to the unlabeled training data.

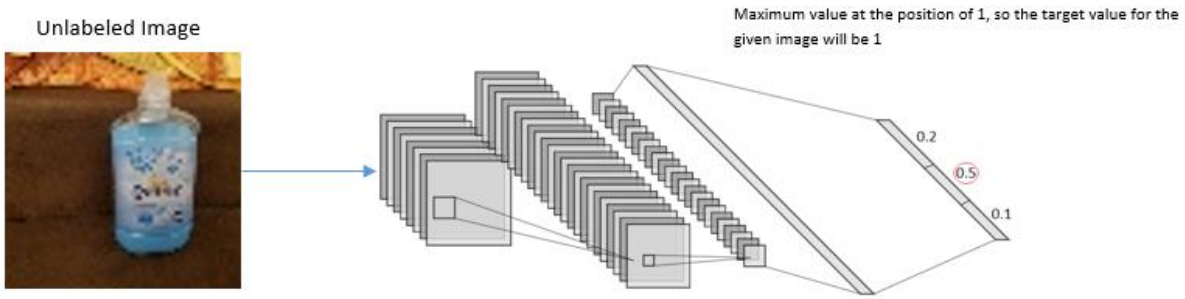

Figure 25: Process of selecting a target value for the given image

The proposed algorithm does not learn any labels but recognizes certain subpatterns and parts of the shapes, representing them by the same neurons. In the learning process, such an algorithm should learn to reinforce a specific output in response to the previously learned input shape. There is no label assigned to this particular output neuron, but the algorithm assigns this output neuron to the input pattern, which reacts the most to it and reinforces this output neuron to learn this input better.

Classic CNNs are trained using the supervised training principle that assigns the desired output neuron and labels to a certain number of input patterns. For each input pattern, the outputs of the network are calculated and compared with the desired outputs (labels), and then, the parameters of the network are updated in the backpropagation process. In our algorithm, the procedure is similar, except that no desired labels are provided during training, and the representation of the groups of input patterns is not fixed and assigned to the certain output neurons. The entire data set or large batches are passed through the network during each epoch and based on the predicted outputs computed in this way; there is a stage of dynamic construction of desired outputs on the basis of the maximum output response which will be used during the learning backpropagation pass and update of the parameters.

Table 1: Network Structure

Layers	Shape
Conv2d-1with ReLu	128, 64, 64
MaxPool2d-2	128, 32, 32
Conv2d-3 with ReLu	128, 32, 32
MaxPool2d-4	128, 16, 16
Conv2d-5 with ReLu	64, 16, 16
MaxPool2d-6	64, 8, 8
Conv2d-7 with ReLu	64, 8, 8
MaxPool2d-8	64, 4, 4
Conv2d-9 with ReLu	64, 4, 4
Linear-10 with ReLu	512
Linear-11 with ReLu	128
Linear-12 with Softmax	Number of classes

The model was created in Python with the use of PyTorch library, which is a very powerful tool for developing deep learning algorithms. The easy-to-use interface makes it really straightforward to model the architecture that we want to use. PyTorch offers many built-in activation functions, optimizers, and utility methods.

At the beginning of each epoch, the entire collection or a large batch passes through the CNN network. In this way, we get predicted outputs for each training example. These predicted outputs are then passed to a function to determine a new subset of data that will be used at the training stage. Suppose we have three neurons at the output of our network. Therefore, we sort our set of predicted outputs 3 times, taking values on all three positions as the sorting key. So first we sort all predicted outputs from the highest value looking at the values at position 0 in the vector. Then we sort by the values at position 1 etc. After each such sorting, we choose *n* training examples, which will be assigned a given label at the learning stage. These training examples, called by us ‘’winners’’, are those that most influenced the strengthening of a given neuron, i.e., what they presented allowed to strengthen this particular neuron at the output. Let's assume that the value of *n*, passed to the function determining the data set is 5, and then after each sorting, we select 5 training examples with a given label, e.g., when we sorted by the values at position 0 in predicted outputs, these 5 selected training examples will have label 0 during training. After this process, 15 training examples are returned, along with labels that can be used for training. Thanks to such a system, we strengthen given patterns at a particular position on the network output, since some training example has strengthened the neuron at the 0 positions the most, we want to strengthen this neuron even more in response to this type of training example, giving a label vector [1, 0, 0] which will be used during training. Thanks to this process, we allow the network to ground further the shape, color, and location of the object in one specific neuron. At the learning stage, the number of ‘’winners’’ assigned to a given neuron increases. We can specify how many times we want to increase the number of ‘’winners’’ by giving an appropriate parameter at the beginning of learning. Thus, the size of the subset is increasing at the learning stage, but it is done in such a way that each neuron always has the same number of assigned learning patterns. At the stage of calculating the cost function and further backpropagation, the number of the neuron at the output to which the given pattern has been assigned are given as valid values, i.e., labels (Fig. 25).

Suppose a simplified situation that the entire set consists of eight learning patterns shown in Fig. 26. We are at the learning stage, where the algorithm has to assign two photos for each output neuron, basing on the maximum values. The diagram in Fig. 26 shows how the patterns that will be used when training in the current epoch are selected on the basis of the maximum output values computed by the CNN network.

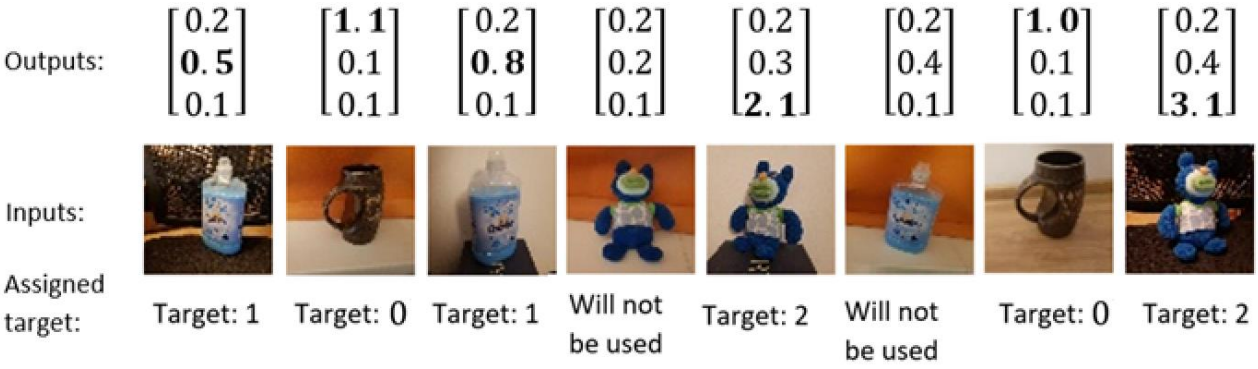

Figure 26: Process of selecting labels for a batch of images

```
train_network(epochs)
{
    for epoch in epochs{
        for batch in batches{
            network_output = model(batch)
            if epoch in increase_sample_number{
                num_of_best++
            }
            dataset_to_use, labels = find_n_best(network_output, num_of_best)
            compute_loss(dataset_to_use, labels)
            backprop_step()
        }
    }
}

find_n_best(network_output, num_of_best)
{
    for class=0;class<num_of_classes;class++{
        best = sort(network_output, key= max(vector [class])
        for best_output in best{
        append (best, class) -> dataset
}
}
return dataset
```

Code 1: Pseudocode of the main loop of the program

5.3. Summary

This chapter describes the workflow of the algorithm created for this work. The algorithm is classified as semi-supervised as it uses a network architecture and approach taken directly from supervised learning, but tries to minimize the need to provide pre-labeled data. The algorithm uses dynamic label creation during the learning stage. This chapter presented how this is done and how the 'winners' are selected and used to properly train the network.

6. Experiments Conducted

6.1. Results of successful experiments

The results carried out on the dataset created for the purpose of the experiment were successful. Our experimental dataset consisted of 300 photos belonging to three classes. The network has learned with an accuracy of about 80 percent to assign a given shape to a given neuron at the output, which is quite a good result, considering that the network did not have any labels or information about the pictures.

The algorithm prepared in this way gives a lot of space for conducting experiments. In our approach, we used the modified supervised learning approach to cluster input training examples without the requirement or use of output labels. Next, these clusters represented by the neurons of the CNN (as a preprocessing module) can be transferred and used to train the classifier. The granularity of the clusters is based on the number of output neurons of the CNN network that can be freely chosen.

When creating a certain number of output neurons of the CNN, we define the granularity of clusters that may be smaller or bigger dependently on future use. It is usually bigger than the number of classes we want to finally identify or train when using these clusters for classification, where we add classifying layer(s) connected to the clustering neurons that helps do discriminate a final set of classes trained in the supervised or semi-supervised manner. The point is that our network can learn to differentiate the data more than comes from the number of demanded classes. One class of objects, e.g., teddy bears used in the previously presented dataset, has a label in the classic supervised algorithm and strengthens one specific neuron at the output during the training. In our solution, we allow the network to train more clusters (later defining classes) on the basis of the presented patterns. The main idea is that every group of similar input patterns strengthens a different neuron at the network output, depending on various features of input patterns, e.g., shapes, color, background color, light intensity, or a position. According to the number of output neurons, the training process allows us to find a certain number of clusters that group together training examples, e.g., images. This technique may serve in the future as a solution to the problem associated with classic convolution networks and supervised training. Such networks often have problems with recognizing the same object, but, e.g., after turning a certain angle or moving away from a given object. It is important to choose the number of neurons at the output because they determine the granularity of clusters that will be created during the training process. Just because we think a learning set should consist of three classes does not mean that the network will not learn to allocate objects to more categories. For example, the same object, but on a different background, can activate a different neuron. Such experiments regarding granulation of the output of the neural network showed that, in most cases, a better effect could be obtained by increasing the number of neurons at the output in relation to the number of classes. Experiments showed that the best results might be achieved for increasing by about two times, but it also depends on the complexity of the dataset. For the used dataset consisting of 3 classes of objects, good results were obtained experimentally for the number

of neurons equal to 5 or 7. It should be borne in mind that the beginning of learning is a crucial moment. The network can strengthen some weights too much, which means that most of the learning samples might be "pulled" by a single neuron. Here, it is wise to experiment with the initialization of weights at the beginning of learning.

In most cases, we proposed initialization drawn from uniform distribution $uniform(-stdv, stdv)$. $Stdv$ value depends on the number of input channels to the current layer. $stdv = \frac{1}{\sqrt{n}}$ where n is a number of input channels multiplied by kernel size. This weight initialization, which is the default one for many architectures, gave us the best results. Another one that we used was Xavier initialization given by equation $G\sqrt{\frac{6}{f_{in}+f_{out}}}$. f_{in} and f_{out} are cumbers of input and output connections, and G is gain. Weights, which were too close to zero, for example, when we set the gain in Xavier initialization to the value of 0.01, meant that the network did not learn anything specific. It should be remembered that for different datasets, different weights initialization may perform better, and there is no one solution to that problem. Some results may be seen in table 2.

Fig. 27 shows the average value of photos assigned to three output neurons at the learning stage. The images present the desired three shapes (classes), which means that the network has learned to distinguish between objects of three learned classes and strengthen the appropriate neurons at the output.

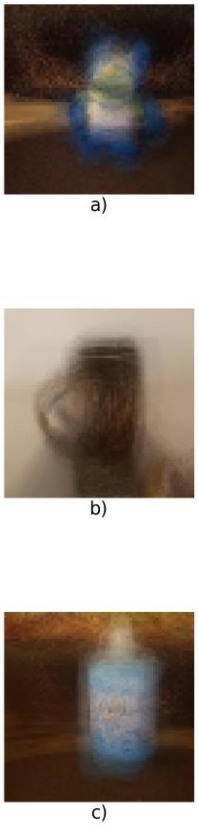

Figure 27: Mean images calculated during the process of learning

A network taught in this way must be prepared for recognizing previously unseen pictures. It must be done because at this stage, and we do not know how the network learned to assign patterns to the output neurons. We need to prepare some mapping to be able to make predictions with the network taught in this way.

For example, if we consider 5 neurons on the output, this process can take place as follows: Check the average image generated at the learning stage. For 5 neurons, but 3 classes, we are considering that the network has learned to categorize those three types of objects to more than 3 groups, which we called granulation of the classification process. If we use more output neurons than the predicted number of classes in the images, the network will increase the granularity of the clusters automatically to distinguish between potential subclasses of the images that humans cannot take into account. This approach to the topic of clustering allows us more flexibility and does not assume a fixed number of clusters for a given data set. By increasing the number of neurons at the output in relation to the potential number of classified objects, we allow the network to draw conclusions not only about the object itself but also about the background or the size and rotation of the objects. The same object on one background can be assigned to one neuron, while on another background to another.

Table 2: Experiments results

Labels	Optimizer	Lr.	Time per epoch [s]	Out. classes	bear acc.	pint acc.	bottle acc.	total acc.
True	Adam	0.001	0.114	3	100%	100%	100%	100%
False	Adam	0.001	0.231	5	64.00%	86.00%	68.00%	72.67%
False	Adam	0.001	0.239	7	48.00%	94.00%	94.00%	78.67%
False	Adam	0.001	0.221	3	14.00%	80.00%	88.00%	60.67%
False	Adadelta	1	0.228	3	24.00%	80.00%	36.00%	46.67%
False	Adagrad	0.01	0.230	5	78.00%	76.00%	68.00%	74.00%

Best results were achieved for Adam optimizer and 7 output classes. The accuracy of almost 80% is still less than 100% for labeled data, but it is result satisfactory enough, taking into account that no labels were provided.

Mean images for 5 classes are presented in Fig. 4. And we can see that "bottle" and "mug" were divided into two additional subclasses. Neurons 0, 1 at the output absorbed the most images depicting a bottle, neuron 2 learned to represent a teddy bear, and neurons 3 and 4 represent a mug. We can see that the same object on two other neurons is presented on a different background, in a different position. The network has created additional subclasses for a theoretical three-class data set.

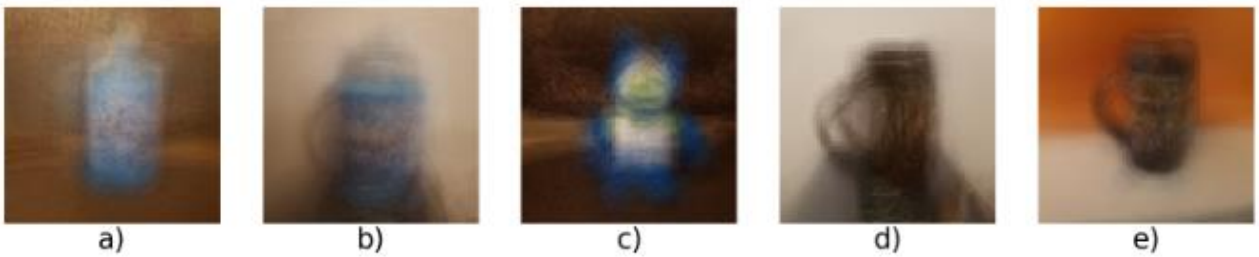

Figure 28: Mean images generated to prepare mapping for the learned model with 5 output neurons

6.2. Mapping the output for further usage

At this stage, you need to create a mapping to know which neuron corresponds to a given class; you can do it based on the image generated in this way; however, the more accurate method is to generate histograms showing how many photos have been assigned to a given neuron, and prepare mapping on this basis.

For each class, I choose the neurons that best classify it.

In this way, I get the following mapping:

‘mapping’: {

‘teddy bear’: [2],

‘mug’: [3, 4],

‘bottle’: [0, 1]

}

For the purpose of this example, we generated histograms showing how many photos have been assigned to every output neuron to ensure that the mapping was created properly.

Histograms show how many images strengthened each particular neuron. In Figure 28, we can notice that for teddy bear neurons at position 2 absorbed most images. Other neurons also absorbed many of teddy bear images, but not as many as other object types.

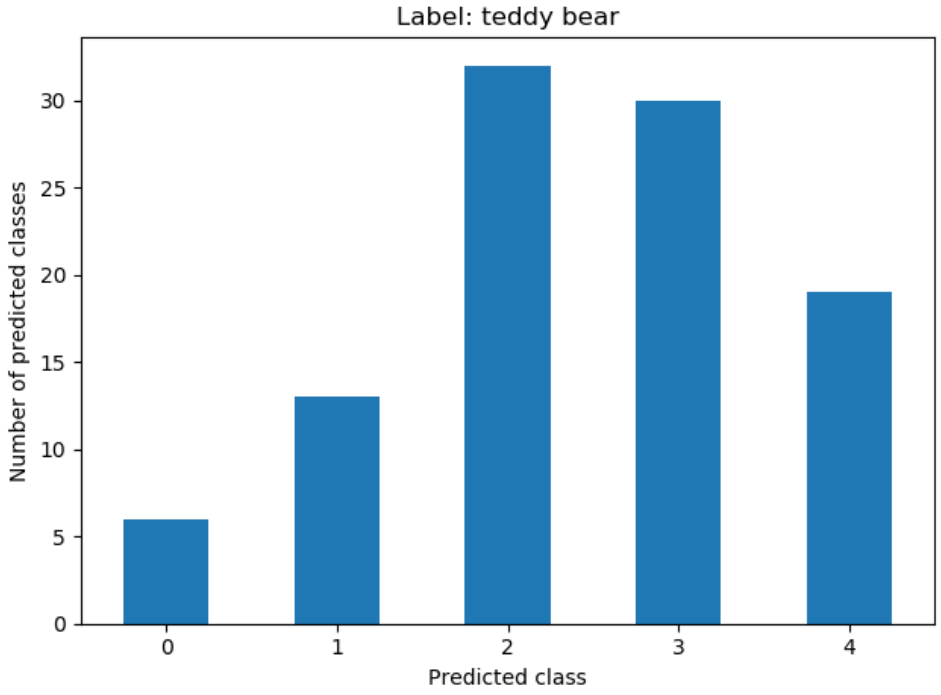

Figure 29: Histogram for teddy bear object type

Fig. 29 presents a similar situation but for the mugs. Neuron 3 is absorbing almost all mug images, but neuron 4 also absorbed some of them, more than any other image type, so we would map neurons 3 and 4 as mug representations.

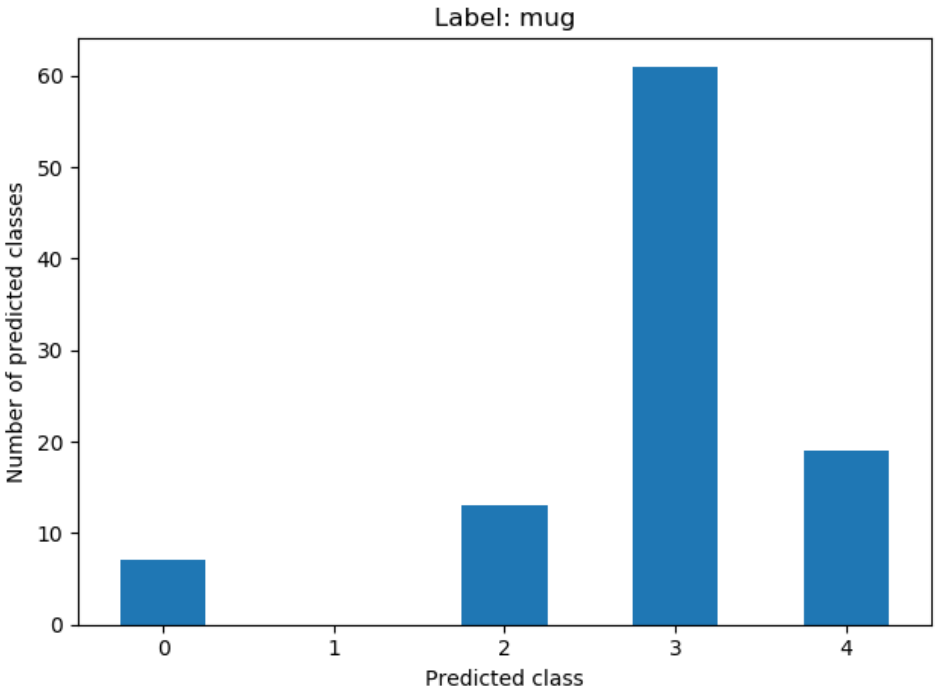

Figure 30: Histogram for a mug object type

The last object which needs to be considered is a bottle. Figure 30 presents a histogram for images presenting a bottle. Neurons 0 and 1 will definitely represent this object type well. Neuron 2 should also be taken under consideration, but it was previously assigned to teddy bear objects.

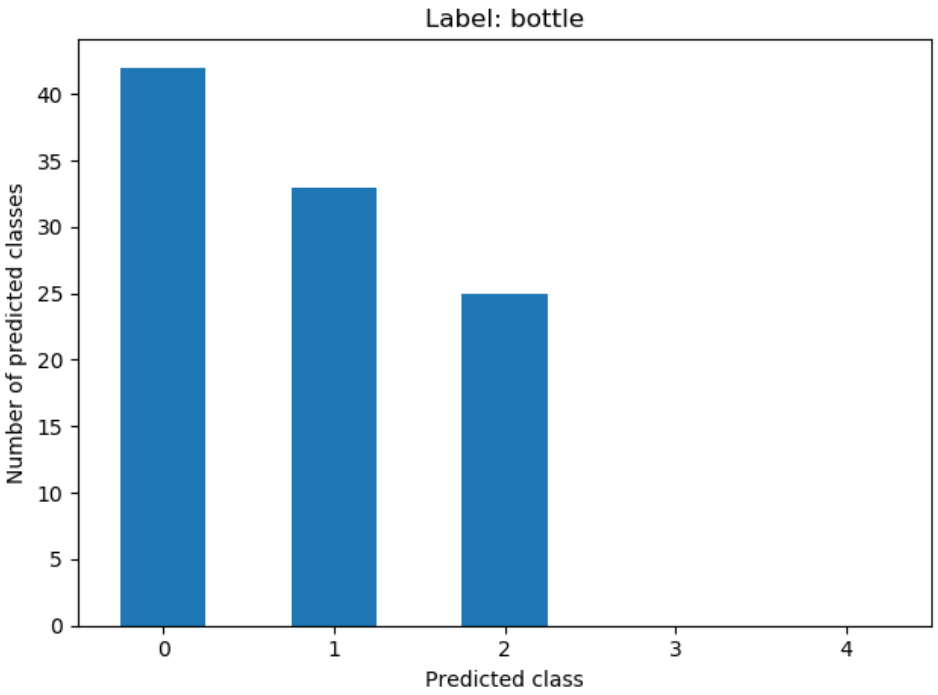

Figure 31: Histogram for bottle object type

Several experiments were carried out. The results were based on a test set not previously seen by the network. Classification of these three objects was a simple task that the supervised network achieves 100% efficiency; however, the results of almost 80% for unsupervised networks would also be optimistic. For some models, although the efficiency was low for one class, it gave very good results for the other two. Such models can also serve as a kind of pre-trained model to recognize these two classes, and then we can retrain the model into more classes. For not very complicated data sets, this method can be used to train a fully functioning classifier. More complex sets, with more classes and more data, can use this method as a kind of network pretraining. We can ground certain patterns in specific neurons on the output, and then train the network using classic, labeled data. We can add layers to the algorithm trained by us. Then only the weights of the added layers can be trained. We can explore the diversity of data sets, i.e., how the network divides specific examples into subcategories.

The experiment carried out on the MNIST data set allowed us to observe several features of the proposed algorithm. The network for this data set was trained for 150 epochs. The batches used were 2000. The problem that arose was forcing the network to detect all occurring patterns. Often, the network learned to recognize only a few of the numbers found in the set. To solve the problem, learning on a small subset of the labeled data has been added to the algorithm. In this way, by providing some labeled data (in our case, 4000 out of 60,000 photos) at the very beginning of the learning, we can, in some way, indicate the initial direction, which should be followed at a later stage of learning. Then, for the rest of the

training, we can use the solution proposed by us. Another observed thing is that for the MNIST dataset, the best results were achieved for 10 neurons at the output, i.e., as many as there are categories. This may be due to the fact that due to the lack of background and similarity of samples from the same groups, networks found it difficult to learn how to divide a given type into a subcategory.

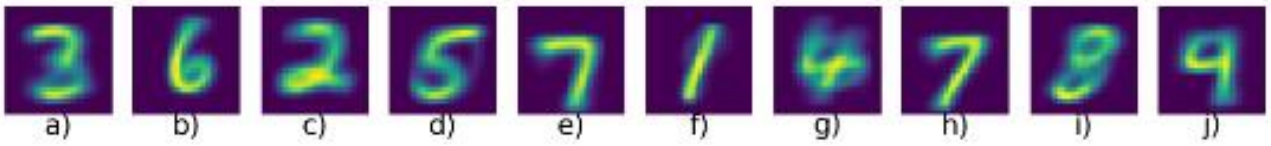

Figure 32: Clustering for MNIST Dataset

6.3. Problem with SGD

As mentioned at the beginning, this work was of a research nature. Many experiments performed at the stage of its creation were unsuccessful. On the basis of these failed experiments, certain conclusions can be drawn which, in the opinion of the authors, are worth quoting as they also have some scientific value.

In the chapter on optimization, different training methods were presented depending on the number of training patterns used in one iteration. In the first version of the algorithm, the Stochastic Gradient Descent method was used, operating on one sample at a time. In the final phase of the experiments, for the prepared data set, the entire set was used in each iteration, large batches were used for the MNIST data, e.g., 2000 samples at a time, and this technique gave good results. Taking a single sample each time meant that at the very beginning of learning, the weight was amplified in such a way that one output neuron absorbed practically all subsequent samples because the weights were set from the beginning to the values strengthening this one neuron. For this reason, the network was unable to solidify any output information. Another reason for using batches is that individual samples may contain some noise. When we consider many samples at once, we average this noise, reducing its impact on the learning process.

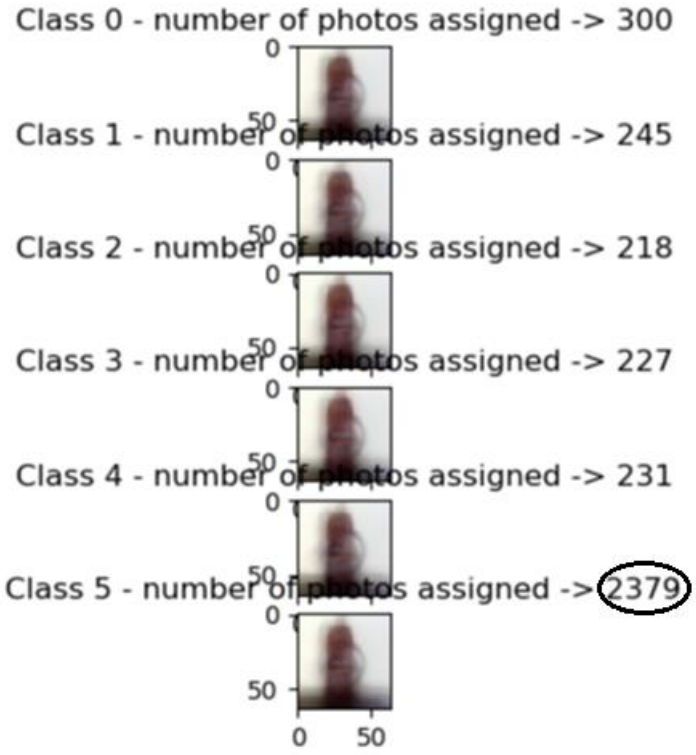

Figure 30: One neuron absorbing almost all samples because of the wrong gradient descent method

6.4. Problem with the backgrounds

Another problem that appeared at the stage of developing the algorithm was an incorrectly prepared dataset. Originally, all photos were taken on the same background. As our collection is not very numerous and the photos of objects belonging to particular groups are varied mainly in terms of location on the stage, rotation or distance from the camera, the network began to learn to recognize the background (as a frequent pattern), instead of individual pieces in the images. This error was hard to detect, only the visualization of what was strengthened by the filters of the first convolutional layer showed that, in most cases, the brightest part, i.e., the most enhanced, was the wall behind the objects (Fig. 34).

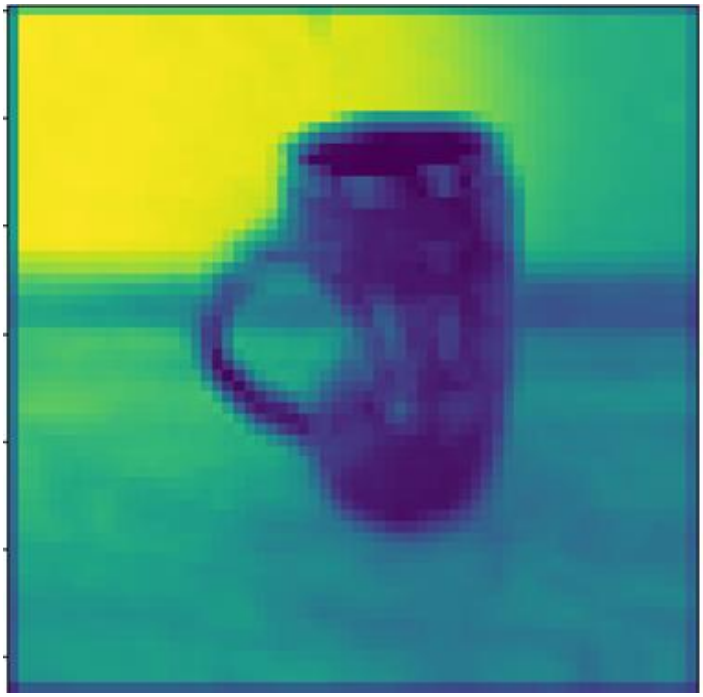

Figure 31: Convolutional Network enhancing wall behind the object instead of the object itself

After changing the dataset with the same background on each of the photos to the one presented in Chapter 4, the network finally learned to recognize certain features on the objects, like it was presented in figure 35.

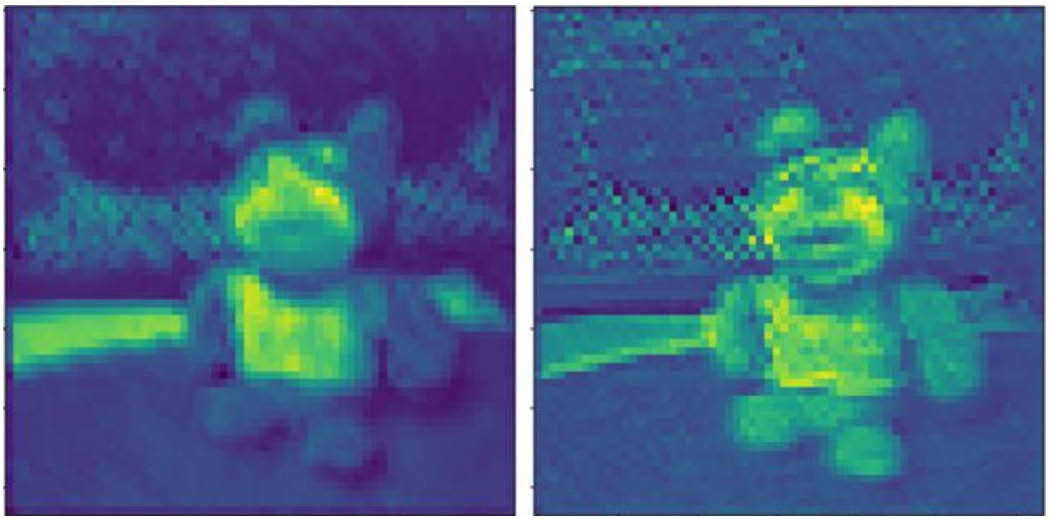

Figure 32: Network enhancing specific parts of the objects

6.5. Summary

The chapter described the results of the conducted experiments. They were successful and left a lot of room for improvement and further experiments. For more complex data sets, at least partially labeled data is needed to indicate the direction of further learning to the network.

7. Conclusion

The main purpose of writing this work was to propose a new algorithm, belonging to the semi-supervised group, that would allow clustering of the objects shown in the photos. This clustering is done without knowing the labels or with the data labeled only to a certain extent. It is an important and popular topic recently. Its popularity results, among other things, from the large resources that should be spent on labeling data. Researchers are constantly trying to invent new algorithms that would allow them to achieve good results with as little effort as possible to prepare the data. The starting point for writing this paper was an attempt to refer to this problem, proposing a new solution and testing it. The first part of the work described theoretical issues, placing more emphasis on convolutional neural networks, as they were the main point of this work. Information presented in the theoretical part is necessary to understand the discussed algorithm. The practical part discusses the issues related to the operating principle of the created algorithm. It shows how data is selected, how labels are created in the learning phase, and it discusses differences between the presented approach and classic networks. The results of the algorithm's operation on a specially prepared dataset and on the MNIST benchmark dataset are also discussed. The conclusions drawn from this work may serve as a starting point for other researchers, and even for the author and the promoter themselves, for further experiments. The efficiency of the algorithm itself can be tested, among other things, in terms of accelerating the entire classification process, in opposition to the data that needs to be labeled. The work was of a research nature, and at the beginning of its writing, the author and the promotor operated based on the promising hypothesis, but they were not fully sure whether the algorithm gives positive results. The fact that the network has been learned to detect simple objects without providing information about them beforehand (labels and classes) gives optimism about the further development of the described approach.

Acknowledgments: This work was supported by the equipment (camera) of the OPUS project No 2016/21/B/ST7/02220, 21st Edition: "*Development of effective mechanisms for robot perception using motivated learning and self-organizing associative memory*." and the promoter research and proposal of the new algorithms and approaches to learning.

Bibliography

[1] V. M. Sebastian Raschka, Python Machine Learning: Machine Learning and Deep Learning with Python, scikit-learn, and Tensorflow, 2nd Edition, Pact, 2017.

[2] "AI Multiple," 15 June 2020. [Online]. Available: https://research.aimultiple.com/data-labeling/.

[3] G. S. D. S. P. R. Valentno Zocca, Python Deep Learning, Packt Publishing 2017, 2017.

[4] Advanced Machine Learning with Python, Pack Publishing 2016, 2016.

[5] R. Karim, "https://towardsdatascience.com," [Online]. Available: https://towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614d#a253.

[6] X. Z. S. R. J. S. Kaiming He, "Deep Residual Learning for Image Recognition".

[7] S. Ruder. [Online]. Available: https://ruder.io/optimizing-gradient-descent/index.html#adam.

[8] *L. Meng'Ao, M. Dongxue, G. Songyuan and L. Shufen, "Research and Improvement of DBSCAN Cluster Algorithm," 2015 7th International Conference on Information Technology in Medicine and Education (ITME), Huangshan, 2015, pp. 537-540, doi: 10.1109/ITME.2015.1.*

[9] T. C. G. Samet Oymak, "Statistical and Algorithmic Insights for Semi-supervised Learning with Self-training," 2020.

[10] *Shingo Mabu, K. K. (2018). Unsupervised Image Classification Using Multi-Autoencoder and K-means++. Journal of Robotics, Networking and Artificial Life, 75-78..*

[11] *Augustus Odena, Semi-Supervised Learning with Generative Adversarial Networks, IEEE Xplore, 2018.*

[12] *M. C. J. Christ and R. M. S. Parvathi, "Fuzzy c-means algorithm for medical image segmentation," 2011 3rd International Conference on Electronics Computer Technology, Kanyakumari, 2011, pp. 33-36, doi: 10.1109/ICECTECH.2011.5941851.*

[13] *S. Riaz, A. Arshad and L. Jiao, "A Semi-Supervised CNN With Fuzzy Rough C-Mean for Image Classification," in IEEE Access, vol. 7, pp. 49641-49652, 2019, doi: 10.1109/ACCESS.2019.2910406..*

[14] *https://medium.com/inside-machine-learning/placeholder-3557ebb3d470.*

[15] K. K. M. O. T. K. Shingo Mabu, "Unsupervised Image Classification Using Multi-Autoencoder and K-means++," *Journal of Robotics, Networking and Artificial Life,* pp. 75-78, 2018.