HIGHER TECHNICAL SCHOOL OF VOCATIONAL STUDIES IN KRAGUJEVAC

RECOGNITION OF HANDWRITTEN DIGITS USING NEURAL NETWORKS

undergraduate thesis

Student
Filip Lazović
094/2014

Mentor
Dr. Vladimir Nedić

Kragujevac
September, 2020

Table of contents

1. Machine learning

Machine learning (hereinafter as ML) is a discipline that studies algorithms that learn based on given data. ML algorithms are used to solve complex problems, which cannot be solved by methods based on manually programmed logic branches. Many problems can be solved in this way, such as a calculator, but there are problems whose solutions do not have clearly defined rules, such as. to recognize the number from the picture. We can make rules based on the pixel values in the image, but that would take a long time. We also cannot guarantee that the program will give an accurate result when the pixel values change by the same number. In contrast to this solution, we will create a large database with examples and a program that will detect which digit is written in the image, automatically.[1]

The first program that was able to learn originated in the 1950s. Arthur Samuel wrote a program to learn the "game of checkers"[2]. He used the Minimax algorithm to calculate which move was next and was the first person to use the term machine learning. In 1952, Frank Rosenblatt made the Perceptron Machine[3], which was tasked with recognizing visual patterns such as human faces. The results were disappointing which led to a stalemate in the development of machine learning until the 1990s. Complex tasks such as recognizing a human face from images became possible after the 1990s for the following reasons:

- The ability of computers to perform more demanding tasks;
- New algorithms;
- Larger databases used for training;

After these changes, ML algorithms show great potential in creating an intelligent machine, especially under a group of algorithms called Deep Learning. With the development of graphics cards, there is a way to speed up the training of these algorithms using parallel computing. With the development of the internet, the amount of training data has increased drastically. 2011 prof. Fei-Fei Li is creating an ImageNet database, inspired by the ILSVRC[4] (ImageNet Large Scale Visual Recognition Challenge, ILSVRC) competition that has revolutionized the field of stroke.

There are two ways of learning: supervised and non-supervised. Supervised learning requires data and labeling. The label is actually the end result we want to predict. Unsupervised learning does not require a label, and the model finds patterns on its own.

ML algorithms are used to solve two types of problems: classifications and regressions. In classification, there are several data labels called classes. In regression, we try to find a trend or relationship between input and output parameters.

Algorithms for classification are Support Vector Machine (SVM*),* K Nearest Neighbor (KNN), Decision Tree.

1.1. Support Vectors Machine

Is also known as the maximum boundary hyperplane method, because a hyperplane is required that separates hyperspace into two parts corresponding to two categories (it can be extended to more categories) so that the distance to the nearest points is maximum. The hyperplane is determined by

the vector w and the equation $w * x + b = 0$. The data must be *linearly separable*, which means that they can be divided by a linear plane. If not, data transformation is performed.[5]

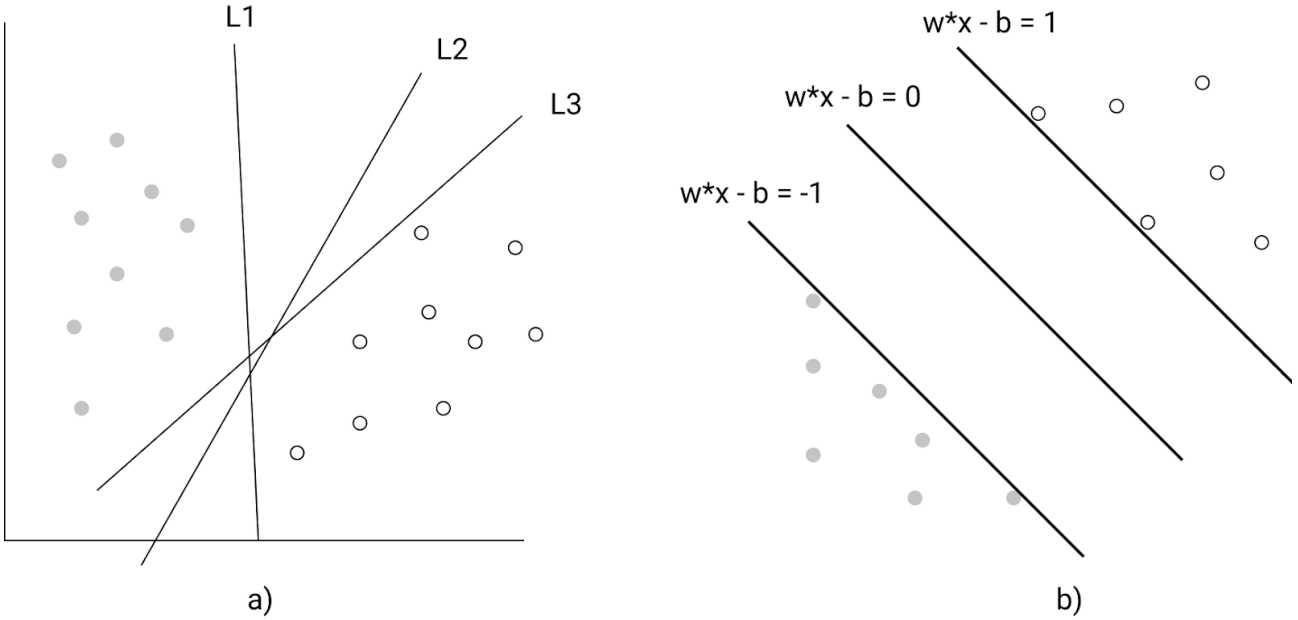


Figure 1. a) Show of several candidates for the line that best separates the data. b) The appearance of the function that best separates the displayed data in two-dimensional space.

If we have a set of examples $\{(x_1 c_1), \ldots, (x_n c_n)\}$ where $c_i \in \{-1, 1\}$, the value of x_i is desirable to be between -1 and 1, and the task is to minimize w, for which a descending gradient algorithm is used.

1.2. K Nearest Neighbors

K nearest neighbors are one of the simplest ML algorithms. Classification can be performed on marked or unmarked data. In the case of unmarked data, clustering using an algorithm such as the k-means method is required. Classification is done by measuring the distance between the new value and all other data.[6] The most popular measure of distance is the Euclidean distance.[7]

If the points x and y with dimensions N are given, the distance d is:

$$d(x, y) = \sqrt{\sum_{i=1}^{N} (x_i - y_i)^2}$$

k is the number of the closest points that we will take into account when determining the class i.e. when $k = 4$ we take the 4 nearest points. The class with the highest number of points is the final result.

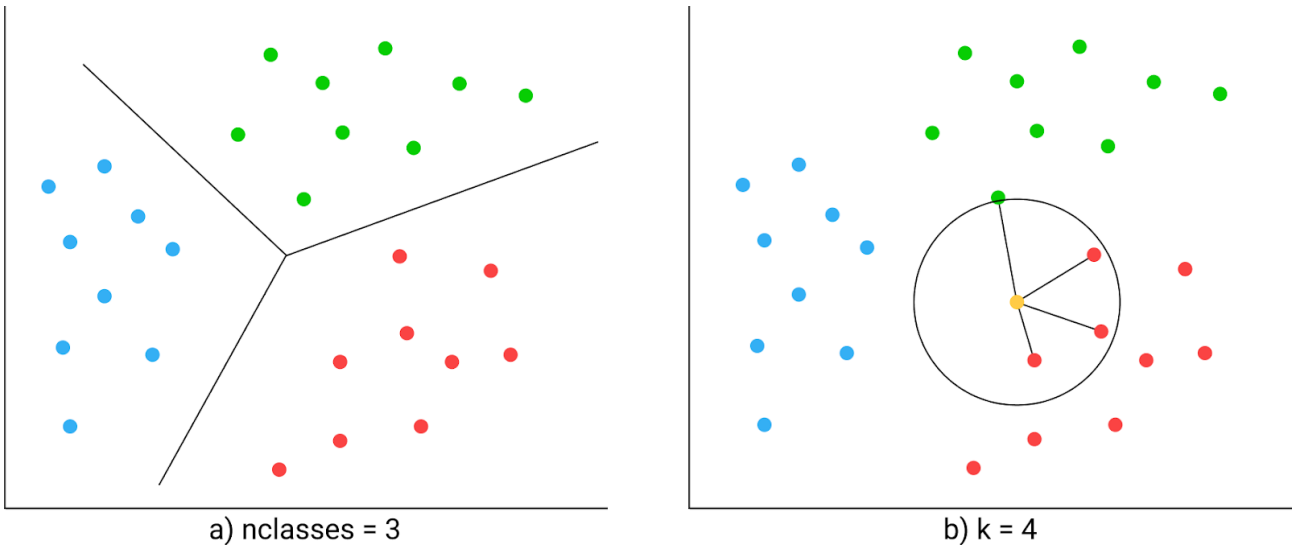
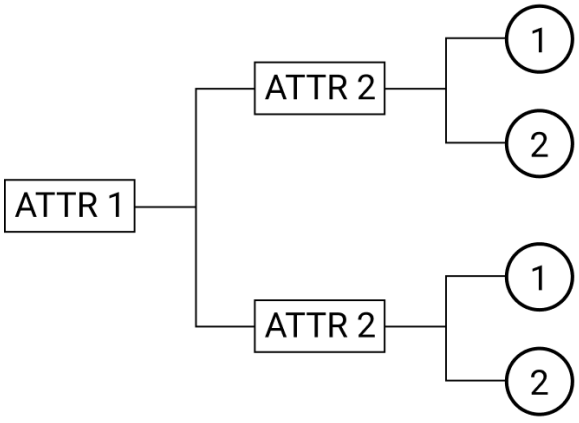


Figure 2. a) Display of data and their classes. b) Example of finding a class for a dot colored yellow.

1.3. Decision Tree

The Decision Tree is a hierarchical structure of data based on decisions.[8] The inner nodes represent the "test" attribute, each branch represents the test result, and each sheet represents a class.



Attribute 1	Attribute 2	Class
T	170	1
⊥	122	1
T	100	2
⊥	70	2
T	99	2

Figure 3. Visual representation of a tree made based on the data in the table.

For binary attributes (true or false), internal nodes are made based on the Gini Impurity[9]:

$$g\,(x_n,\, y_n) \;=\; 1 - \left(\frac{x_n}{x_n+y_n}\right)^2 - \left(\frac{y_n}{x_n+y_n}\right)^2$$

where x_n is the number of correct, and y_n is the number of incorrect attributes. The smaller the Gini value for a given attribute, the better the data division.

For attributes represented by numbers, it is necessary to determine a good value that will divide the data. Suppose that the a_i is at the location i which starts from 1:

$$d_i = \frac{a_i + a_{i-1}}{2}$$

$$x_n = \left|\{n \mid n > d_i\}\right|$$
$$y_n = \left|\{n \mid n \leq d_i\}\right|$$

1.4. Linear and Polynomial Regression

Regression is the process of finding the connection between input and output data. If we have data $\{(x_0,y_0), (x_1,y_1), ..., (x_n,y_n)\}$ the connection between x and y is required. If the connection is linear[10]:

$$y_i = w * x_i + b$$

If the connection cannot be described by a straight line, polynomial regression is used.[11] The non-linearity is introduced by bringing the n-th level of the variable x.

$$y = b + w_1 x + w_2 x^2 + w_n x^n$$

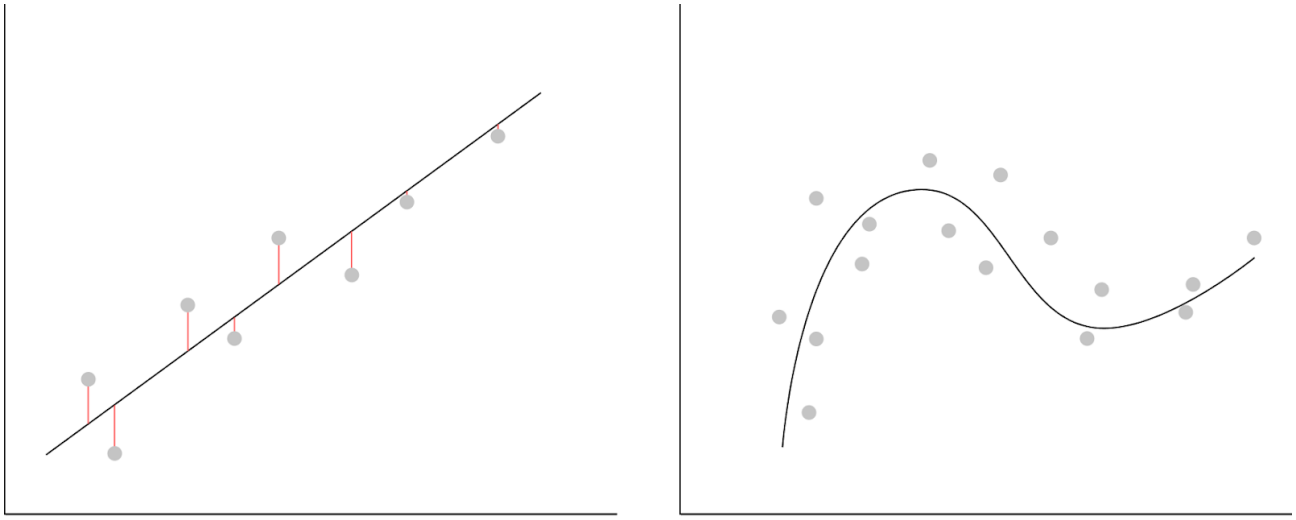


Figure 4. a) Optimal linear function describing the given data. b) A polynomial function that describes the given data.

2. Neural networks

Neural networks are inspired by the work of biological neural networks. They consist of neurons, which retain the biological concept, which means that they receive the input signal, combine that signal with the internal state, and produce an output signal. Input signals are images or documents. The output performs some task (classification or regression).[12]

Since the perceptron did not prove to be a good detector of complex patterns, other methods of ML were used until the advent of the backpropagation method[13] in 1975, which allows the creation of multilayer neural networks. With the development of better and better graphics cards, it was possible to create deep neural networks[14] (with multiple layers), which started the revolution in the field of ML.

The network consists of connections, each connection connecting the output of one neuron with the input of another. Each connection has a weight, which represents its impact on the final result. Neurons can have multiple input/output connections. The sum of the connections calculates the strength of the input signal of the neuron based on the output of the previous output.

2.1. Organization

Neural network models usually have multiple layers. Neurons of one layer can be connected to neurons from the previous or next layer. The layer that receives the data is called the input layer. The layer that produces the output is called the output layer. The layers in between are called Hidden Layers. There are also models without hidden layers. Layers can be linked in several ways. There are fully connected, where the neurons of one layer are connected to all the neurons of the next layer, and the aggregation layer, where a group of neurons from one layer connects to one neuron of the next layer. Networks in which connections between layers go in one direction (from input to output) are called feedforward neural networks. Networks in which neurons are connected to previous layers are called reverse neural networks.[15] A model with only one neuron is called a perceptron.

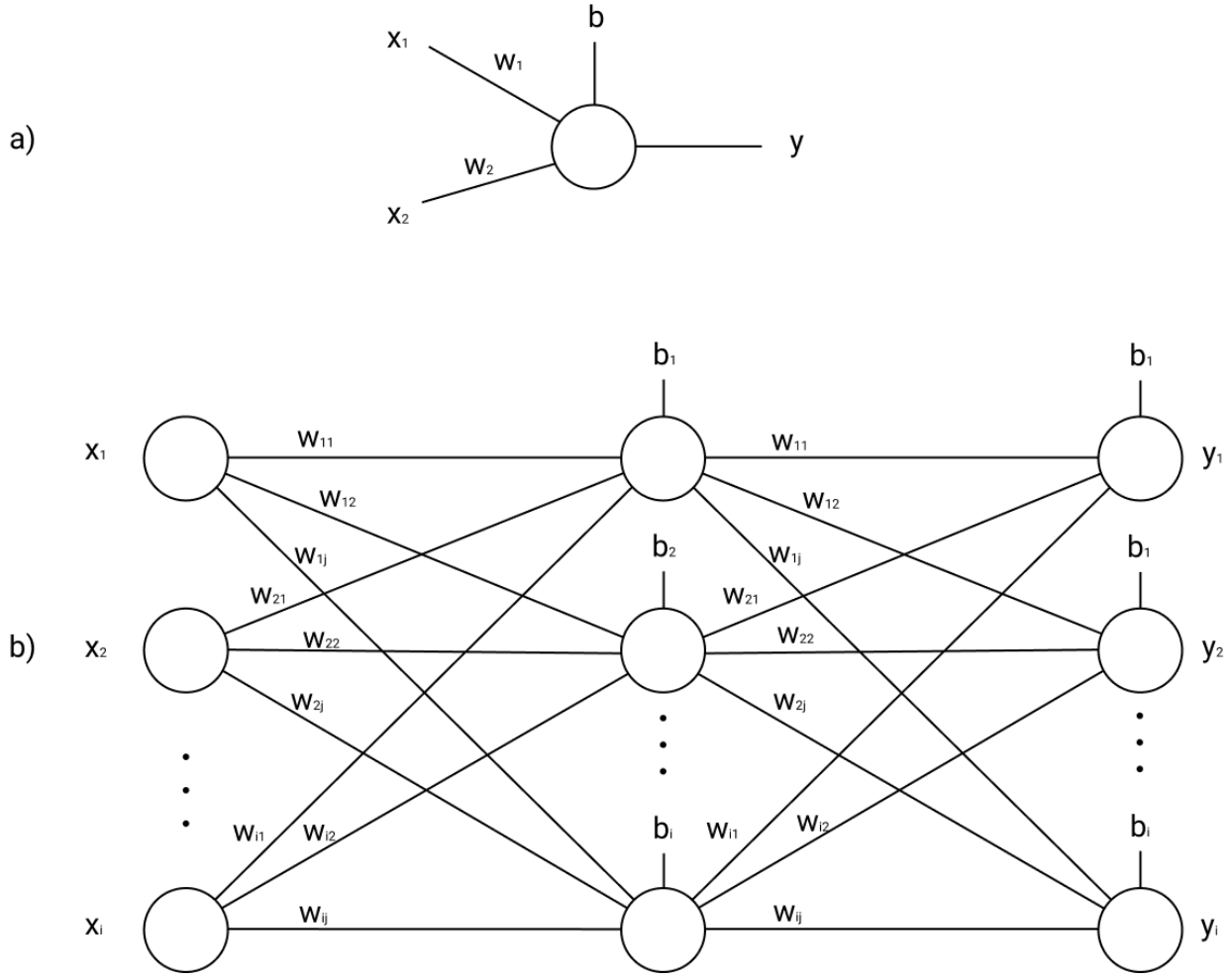


Figure 5. a) Perceptron. b) Neural network.

2.2. Hyperparameters

A hyperparameter is a constant that is initialized before the training process. Examples of hyperparameters are the learning rate[16], the type and the number of hidden on the left, the number of

epochs[17]. It is the programmer who decides the value of the parameters, but some algorithms learn the optimal values for the parameters themselves. This method is called Meta Learning[18].

2.3. Learning

Learning is the adaptation of a network to do a task. It involves adjusting the strength of the connections between neurons to improve the accuracy of the results.[19] This is achieved by minimizing errors. Learning is complete when the error can no longer be significantly reduced. After learning, the error usually does not reach zero. If the error remains large, it means that the model cannot adapt to the input data. This is solved by changing the architecture of our model.

2.4. Learning rate

The learning rate defines the size of the steps that the model takes to reduce the error. A high learning rate reduces training time but also reduces the accuracy of the model, that is, there is a high chance that the model will not reach the state where the error is the smallest. A low learning rate increases the likelihood that the model will come to a state of least error, but training is slower. This is solved by initially setting a high learning rate so that the model skips local minima and finds a global minimum, then gradually decreases until an absolute minimum is reached.

2.5. Weights

The input to each node is the result of the sum of the connections, which is the sum of all the outputs from the previous layer that have been assigned a special weight. The goal of these weights is to find a pattern or feature in the input data. If the sum is large, it means that the certain characteristic is present, otherwise, the input data does not contain the template that this block is looking for. If we have n connections from layer $l-1$ that are inputs to the node j in layer l, and weights are labeled as w and the activation as a then the sum is:

$$z_j^{(l)} = \sum_{k=0}^{n} w_{kj}^{(l)} a_k^{(l-1)}$$

2.6. Activation functions

The output of each node is the result of the activation function[20] (ϕ Function) for which the entry is the result of the sum of connection. From the previous layer, we get the sum of linear functions which is passed to the activation function. To approximate complex functions, nonlinear activation functions are required.

Activation functions give the model two new characteristics: nonlinearity (which allows the model to be a universal approximator of functions) and constraint (limiting the output of a node with a lower or upper limit).

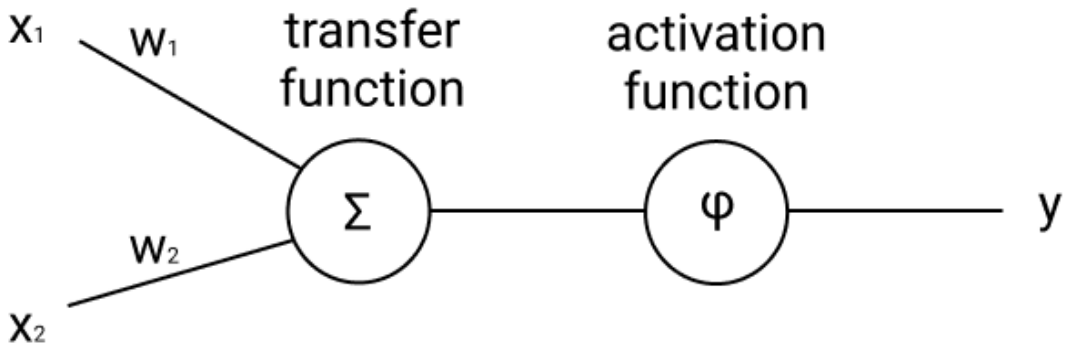


Figure 6. Data flow view. The sum of the connections goes through the activation function.

The most well-known activation functions are:

2.6.1. Sigmoid

Sigmoid belongs to the group of logistics functions. It has the shape of the letter *S*, with the equation:

$$f(x) = \frac{L}{1+e^{-k(x-x_0)}}$$

where:

x_0 = value of x in the middle of the function
L = maximum value of
k = growth rate of the function, or steepness.

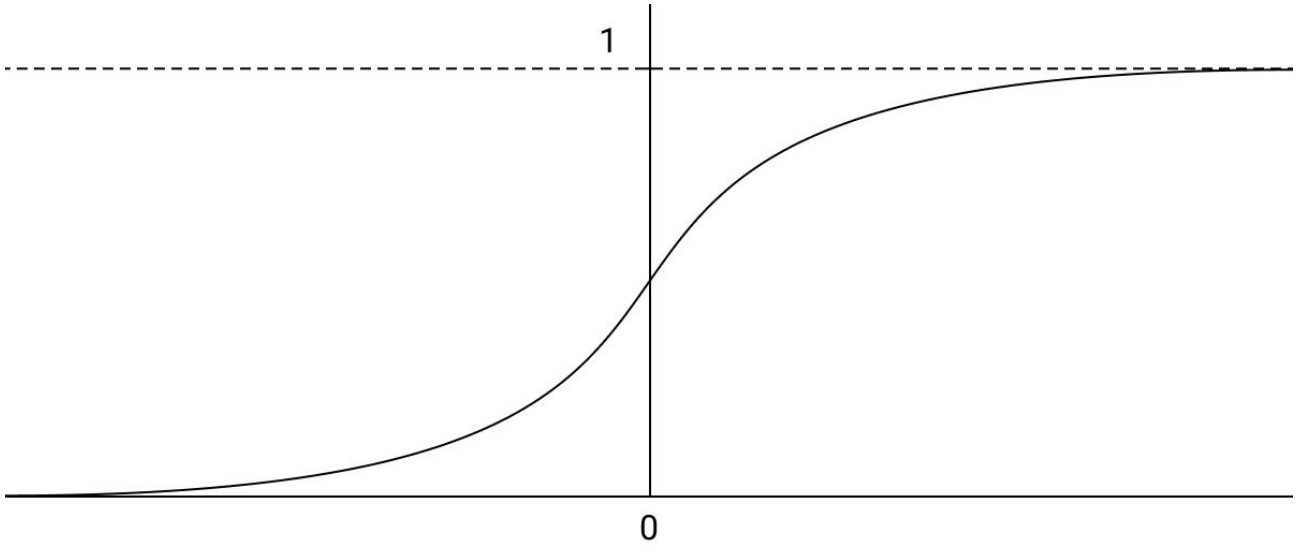


Figure 7. Sigmoid activation function.

2.6.2. Softmax

Softmax function takes a vector with real numbers and normalizes them to the probability distribution. The inputs are numbers from 0 to 1, and the sum of all outputs is 1. It is usually used in the last layer of the neural network to predict the class of the input data. The standard softmax function is defined by the formula:

$$f(x)_{and} = \frac{e^{zi}}{\sum_{j=1}^{K} s^{zj}}$$

if $i = 1, ..., K,$ and $z = (z_{i,}..., z\,k) \in$ R.

2.6.3. RELU

Rectified Linear Unit or RELU. There is no limit, but it always gives 0 if the input is below zero. After zero it acts as a linear function. It is usually used in convolutional layers.

$f(x) = max(0, x)$ where x *is the* input for the neuron.

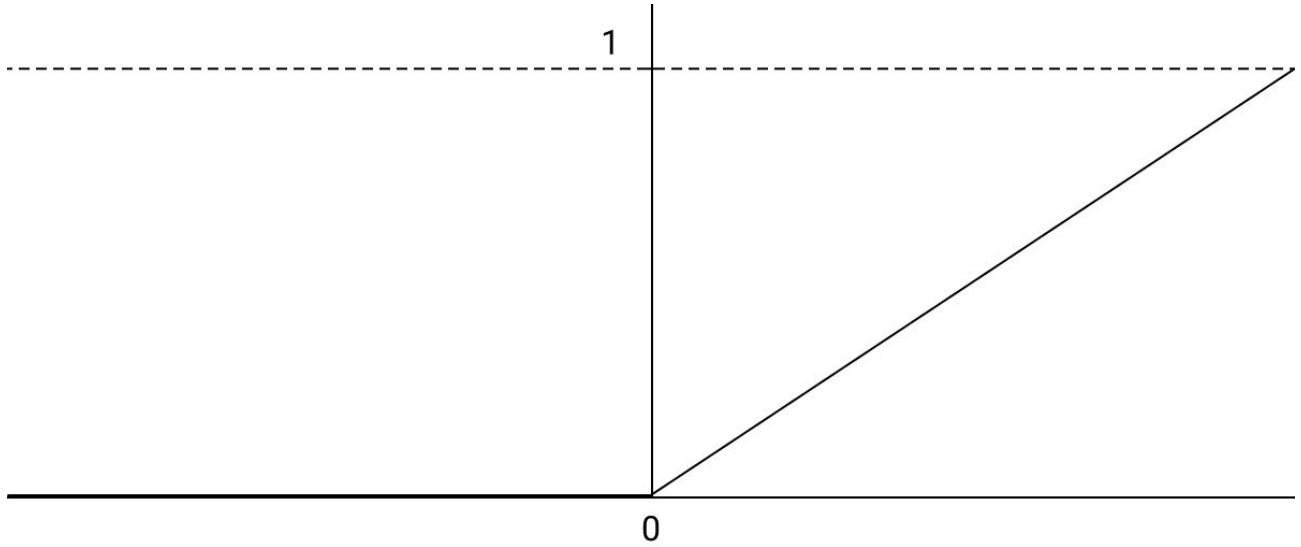


Figure 8. RELU activation function.

2.7. Error

The purpose of the error function is to evaluate the performance of the model, i.e. to give us a numerical value that tells us how close the model is to the correct output. The two most common ways to calculate an error are as the square difference between the correct and the obtained value or the absolute difference between the obtained and the correct value. It is usually denoted as C, and the exact and obtained value as y_i and y, respectively.

$$C = (y_i - y)^2$$

2.8. Regularization

Regularization is a method of generalizing a model if it is too adapted to the data.[21] The goal of the model is to find a function that describes the input data well. If the model describes the data too well, we will not get a good result on data that the model has never seen. To solve this problem, a restriction on parameters in the form of neuronal rejection or normalization is introduced.

2.9. Dropout

Dropout is the random rejection of neurons from a layer with a probability from Bernoulli's distribution.[22] Since neurons are randomly rejected, the model must be adjusted by changing other neurons to recognize the appropriate characteristics.

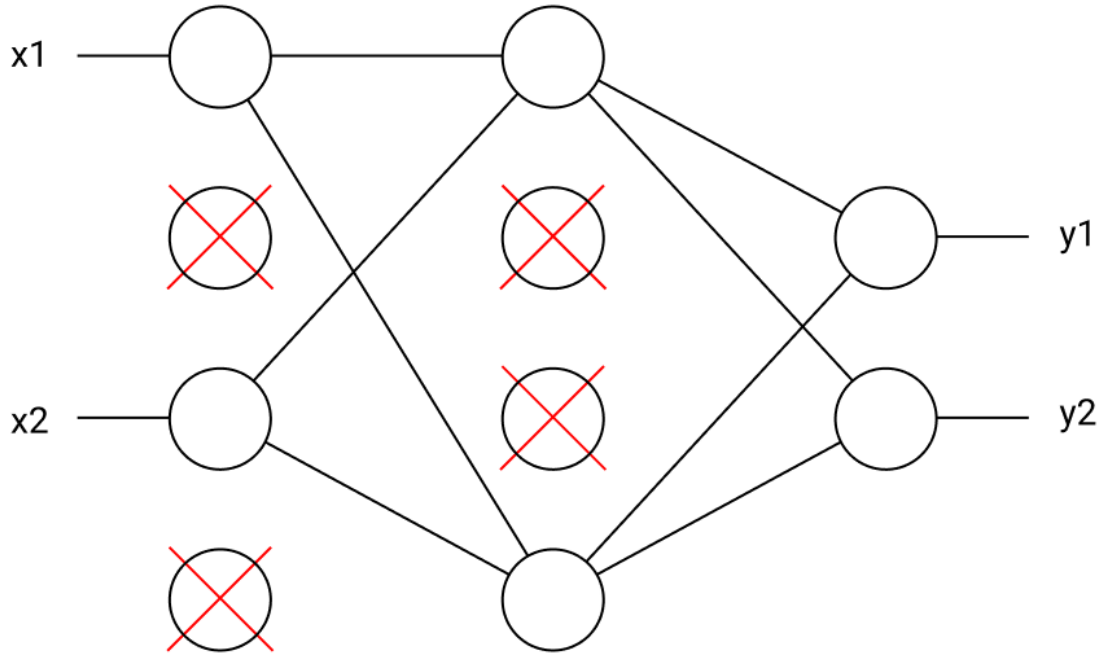


Figure 9. Representation of the network after neuron rejection.

2.10. Normalization

Normalization is a regularization algorithm that enables faster model training because it allows the optimization algorithm to work with a higher learning rate and reach a minimum with fewer epochs. The mean value is defined as m and the deviation from the mean value p so that the data x are within the limits $m - p \leq x \leq m + p$.[23]

2.11. Backpropagation

Backpropagation is a process of changing weights of the model based on the obtained error with the goal of accurately classifying the input data.[24]

The definitions and notations:

L = number of layers in the network
Layers were indexed as $l = 1, 2, ..., L-1, L$.
The nodes in layer l are indexed as $j = 0.1, ..., n-1$.
The nodes in layer $l-1$ are indexed as $k = 0, 1, ..., n\text{-}1$.
y_j = required node value j in output layer L for one training iteration
C = error calculation function for one training iteration

$C_j =$ one node error calculation function for one training iteration
$w_{kj}^{(l)} =$ connection strength between nodes k u layer $l-1$ and node j in node l
$w_j^{(l)} =$ vector containing all connections from layer $l-1$ to node j in layer l
$z_j^{(l)} =$ input for node j in layer l
$g^{(l)} =$ activating the function of a layer l
$a_j^{(l)} =$ the output of the activation function of a node j in layer l

Error is a square difference between the output of the correct activation and output node j in layer L.

$$C_j = (a_j^{(L)} - y_j)^2$$

To get the total error, we calculate the sum of the errors of all nodes on the output layer.

$$C = \sum_{j=0}^{n-1} C_j$$

The activation output of node j in layer l is the result of passing $z_j^{(l)}$ through the activation function $g^{(l)}$.

$$a_j^{(l)} = g^{(l)}(z_j^{(l)})$$

The input to node j in layer l is the weighted sum of activation outputs from the previous layer $l-1$.

$$z_j^{(l)} = \sum_{k=0}^{n-1} w_{kj}^{(l)} a_k^{(l-1)}$$

We conclude that the error calculation is calculated by the composition of the functions:

$$C = \sum_{j=0}^{n-1} C_j(a_j^{(l)}(z_j^{(l)}(w_j^{(l)})))$$

Since we cannot change the error directly, we change the variable $w_j^{(l)}$, in proportion to the size of the error. The task is to calculate how much the error changes with the change of $w_j^{(l)}$, which is achieved by differentiating C_j in relation to $w_j^{(l)}$, using the derivative of a complex function (chain rule).

If we want to find the derivative of the C_1 in relation to the variable w_{11} (L):

$$\frac{\partial C_1}{\partial w_{11}^{(L)}} = \frac{\partial C_1}{\partial a_1^{(L),}} \frac{\partial a_1^{(L)}}{\partial z_1^{(L)}} \frac{\partial z_1^{(L)}}{\partial w_{11}^{(L)}}$$

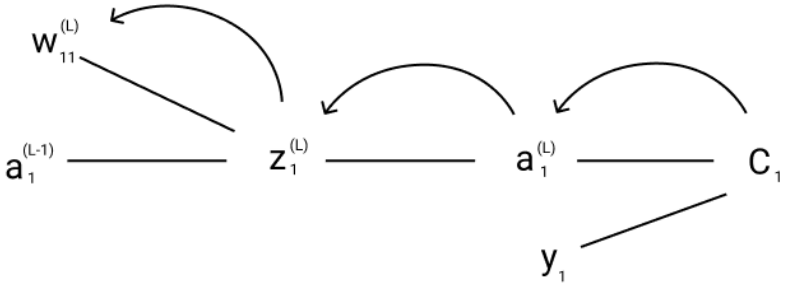


Figure 10. Display of the backpropagation flow.

First item:

$$\frac{\partial C_1}{\partial a_1^{(L)}} = \frac{\partial}{\partial a_1^{(L)}} (a_1^{(L)} - y_1)^2$$

The derivative is taken only from members containing $\partial a_1^{(L)}$, other members are considered as constants, whose derivative is 0. It boils down to:

$$\frac{\partial C_1}{\partial a_1^{(L)}} = 2\,(a_1^{(L)} - y_1)$$

Second item:

$$\frac{\partial a_1^{(L)}}{\partial z_1^{(L)}} = \frac{\partial}{\partial z_1^{(L)}} g^{(L)}(z_1^{(L)}) = g'^{(L)}(z_1^{(L)})$$

Third item:

$$\frac{\partial z_1^{(L)}}{\partial w_{11}^{(L)}} = \frac{\partial}{\partial w_{11}^{(L)}} w_{00}^{(L)} a_0^{(L-1)} + \frac{\partial}{\partial w_{11}^{(L)}} w_{11}^{(L)} a_1^{(L-1)} + \ \ldots\ + \frac{\partial}{\partial w_{11}^{(L)}} w_{kj}^{(L)} a_j^{(L-1)}$$

$$\frac{\partial z_1^{(L)}}{\partial w_{11}^{(L)}} = a_1^{(L-1)}$$

By combining these items we get:

$$\frac{\partial C_1}{\partial w_{11}^{(L)}} = 2\,(a_1^{(L)} - y_1)\, g'^{(L)}(z_1^{(L)})\, a_1^{(L-1)})$$

The previous example shows how the error changes in relation to the connection strength represented by the variable w during one iteration, ie. one input during training. Better results are obtained if we take the mean value of the derivative of several iterations.

$w_{11}^{(L)}$ is then changed by the formula:

$$w_{11}^{'(L)} = w_{11}^{(L)} - r\frac{\partial C_1}{\partial w_{11}^{(L)}}$$

where r *is* the learning rate, a number from 0 to 1. The direction in which $w_{11}^{(L)}$ move depends on the derivation sign.

If we want to change a connection deeper into the network, all we need to do is find out how the final error changes in relation to that connection.

$$\frac{\partial C_1}{\partial w_{11}^{(L-1)}} = \frac{\partial C_1}{\partial a_1^{(L)}}\frac{\partial a_1^{(L)}}{\partial z_1^{(L)}}\frac{\partial z_1^{(L)}}{\partial a_1^{(L-1)}}\frac{\partial a_1^{(L-1)}}{\partial z_1^{(L)-1)}}\frac{\partial z_1^{(L-1)}}{\partial w_{11}^{(L-1)}}$$

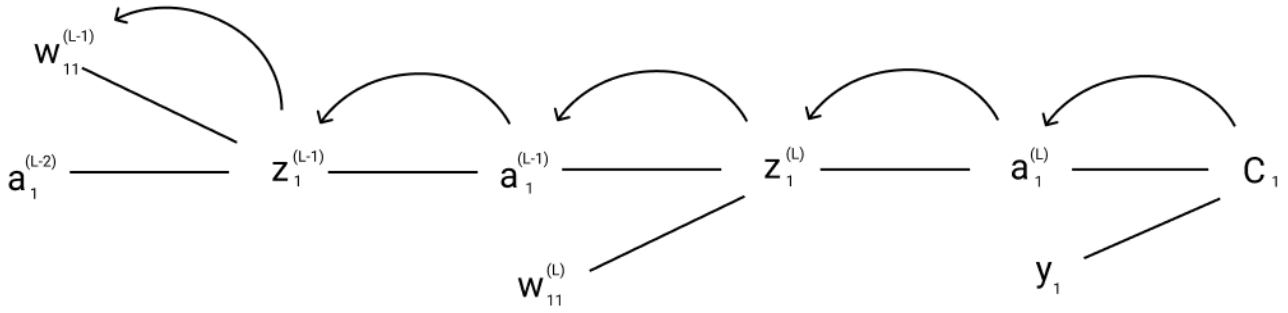


Figure 11. Display of the return propagation flow through a multilayer network.

Bias is also used as an additional input to the node. It is usually denoted by the letter b. In addition to the sum of connections, the activation function of neurons also receives a bias, all other calculations remain the same.

$$z_j^{(l)} = \left(\sum_{k=0}^{n-1} w_{kj}^{(l)} a_k^{(l-1)}\right) + b_k$$

Bias learning is performed in a similar way as for connections:

$$\frac{\partial C_1}{\partial b_1^{(L)}} = \frac{\partial C_1}{\partial a_1^{(L)}}\frac{\partial a_1^{(L)}}{\partial z_1^{(L)}}\frac{\partial z_1^{(L)}}{\partial b_1^{(L)}}$$

The derivative of the input to the respect of activation function is 1:

$$\frac{\partial z_1^{(L)}}{\partial b_1^{(L)}} = 1$$

The final equation is then:

$$\frac{\partial C_1}{\partial b_1^{(L)}} = 2\,(a_1^{(L)} - y_1)\,g'^{(L)}(z_1^{(L)})$$

Changing the bias[25] is done by the same equation as for connections :

$$b_1'^{(L)} = b_1^{(L)} - r\frac{\partial C_1}{\partial b_1^{(L)}}$$

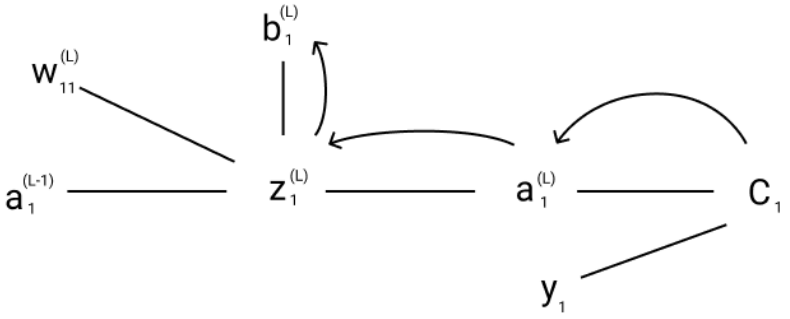


Figure 12. Display of the reverse propagation flow in a displacement network.

As the name suggests, it shifts the activation function in the direction of the x-axis shift sign. Example: If $b = -5$, and we use the Sigmoid activation function the output will be suppressed if the input is less than 5.

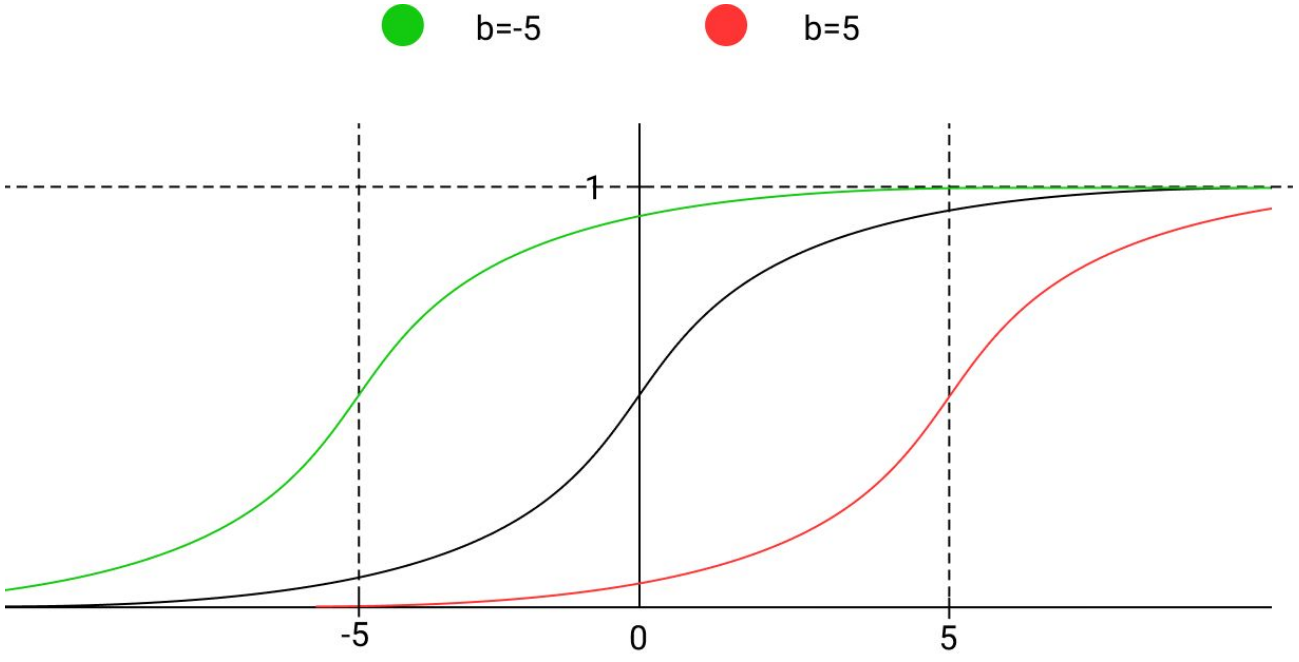


Figure 13. Display of the effect of displacement on the activation function.

2.12. Gradient Descent

We mentioned that the goal of learning is to reduce the error. The process of gradually reducing the error is called gradient descent, which means finding the minimum of the error function.[26] Some functions are simple and the location of the minimum is obvious, but the error function can be much

more complex and can have multiple minima that do not differ too much. Finding the minimum depends on how the network parameters are initialized. During initialization, connection weights are assigned randomly, so we cannot know whether the error will be minimized. Therefore, it would be desirable to repeat the training several times.

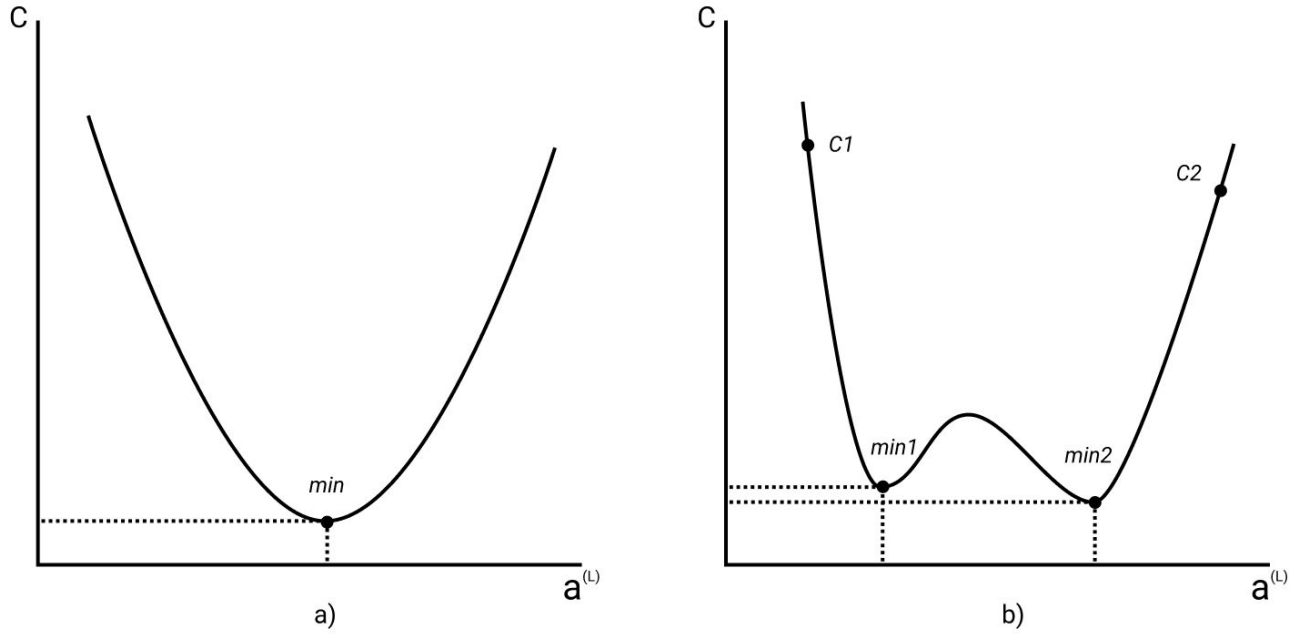


Figure 14. a) Error function with one minimum. b) Error function with two minima.

In the figure, we see that the left error function (a) has an obvious minimum, but the function on the right (b) has two minima. If we get error C_1 during parameter initialization, the minimum will be min_1, otherwise, if the initial error is close to C_2, the minimum will be in min_2. Note that between min_1 and min_2 there is a hill that prevents the error from finding the smallest minimum. Although that hill can be skipped if we use a higher learning rate, we said that with a high learning rate it is harder to find the minimum, so the solution is a declining learning rate.

A declining learning rate implies that the learning rate decreases over time. Initializing a high learning rate would mean that the learning steps are large at the beginning, which would skip the hills that would be in the way, but it would be harder to find the absolute minimum, so gradually reducing the learning rate with each iteration would result in finding a minimum at the end.

2.13. Learning example

id	x_1	x_2	x_3	x_4	y_i
0	0.9	0.9	0.1	0.1	1
1	0.1	0.1	0.9	0.9	0

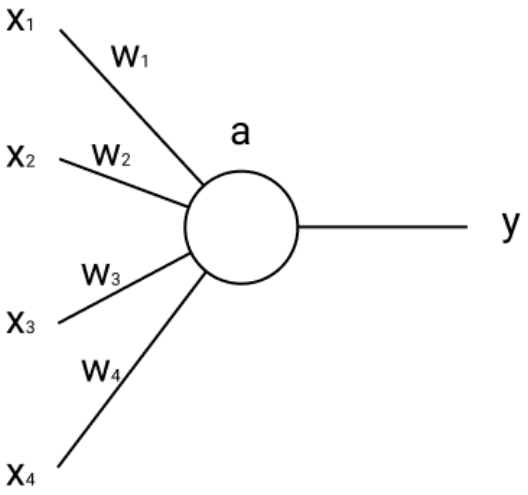


Figure 15. Perceptron with 2 data samples with 4 dimensions.

In the example, we have 4 inputs and one output neuron. We use Sigmoid as an activation function. If we randomly define the values of connections so that they are: $w_1 = 0.2$, $w_2 = 0.5$, $w_3 = 0.3$, $w_4 = 0.6$.

As we have said, the error is calculated by subtracting the output from the activation function and the exact classification value.

$$C_1 = (sig\,(x_1 w_1 + x_2 w_2 + x_3 w_3 + x_4 w_4) - y_i)^2$$

Assigning a variable to the data with identification number 0 we get that the error is 0.107:

$$C_1 = (sig\,(0.9 * 0.2 + 0.9 * 0.5 + 0.1 * 0.3 + 0.1 * 0.6) - 1)^2$$
$$C_1 = (sig\,(0.72) - 1)^2 = (0.672 - 1)^2 = 0.107$$

The derivative of the activation function:

$$sig'(x) = sig\,(x)\,(1 - sig\,(x))$$

Error change in relation to change w_1:

$$\frac{\partial L}{\partial w_1} = 2\,(0.672 - 1) * 0.672 * (1 - 0.672) * 0.9 = -0.129$$

New value w_1:

$$w_1 = 0.2 - (-0.129) = 0.329$$

The same process applies to other connections:

$$w_2 = 0.629$$

$w_3 = 0.314$
$w_4 = 0.614$

Error after modifying input connections with identification number 1:

$$C_1 = (sig\,(0.9) * 0.329 + 0.9 * 0.629 + 0.1 * 0.314 + 0.2 * 0.614) - 1)^2 = 0.07$$

The input error with ID number 2 is then:

$$C_2 = 0.51$$

New connection values:

$w_1 = 0.299$
$w_2 = 0.599$
$w_3 = 0.052$
$w_4 = 0.352$

We see that the new error is smaller than the error before the update:

$$C_2 = 0.373$$

If we ran the algorithm multiple iterations, the connections w_1 and w_2 would increase and w_3 and w_4 would decrease below zero thus the error would be reduced, and r the result of the classification would be correct.

3. Convolutions

The concept of convolution comes from the field of image processing. The idea is to pass an image through a filter that transforms it and expresses some characteristics while suppressing others.[27]

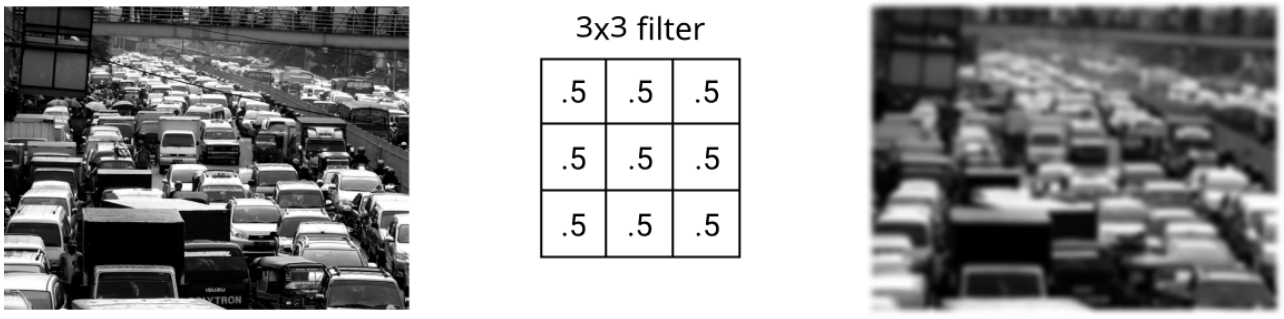


Figure 16. Example of image blur using a filter.

The blurring effect of the image is achieved by a filter. The figure shows an example of a 3x3 filter, which is placed over each pixel of the image. Elemental multiplication is then applied between the filter and the corresponding pixels below the filter. The sum of all elements in the obtained matrix is taken and the obtained value is set instead of the pixel in the center of the filter.

3.1. Filters

In the field of image processing, it is a man who defines certain filters in order to achieve the appropriate effects. On the contrary, in the field of machine learning, the goal of the model is to find the appropriate values that the filter will contain, in order to accurately classify a certain image. Filters can be lines, edges, circles, and even complex shapes such as animal and human body parts, car parts, buildings, and so on.[28]

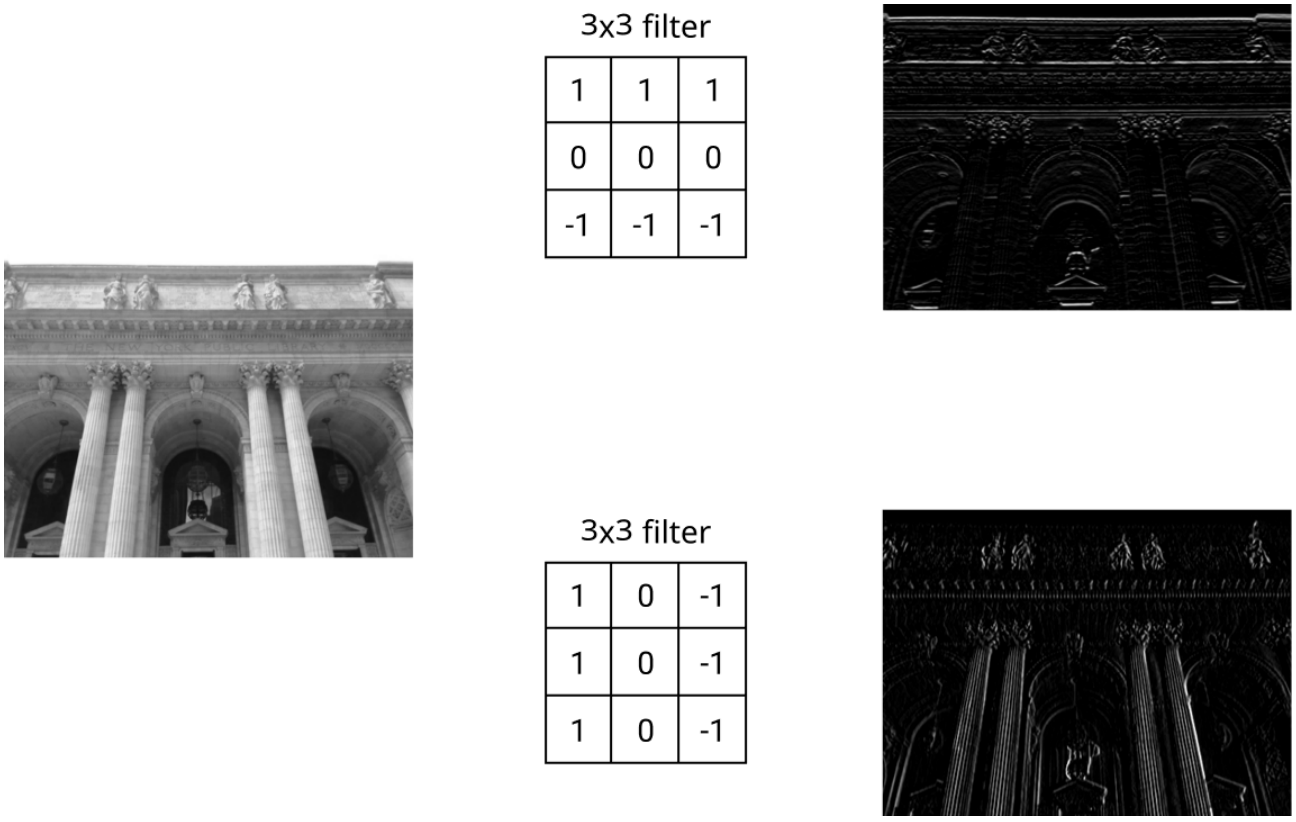


Figure 17. Display of a filter that detects horizontal and vertical lines.

The purpose of the filter is to express the features that a neural network can use to classify an image. They are represented in the form of matrices that pass through each pixel in the image and perform elemental multiplication over the pixels through which they pass.

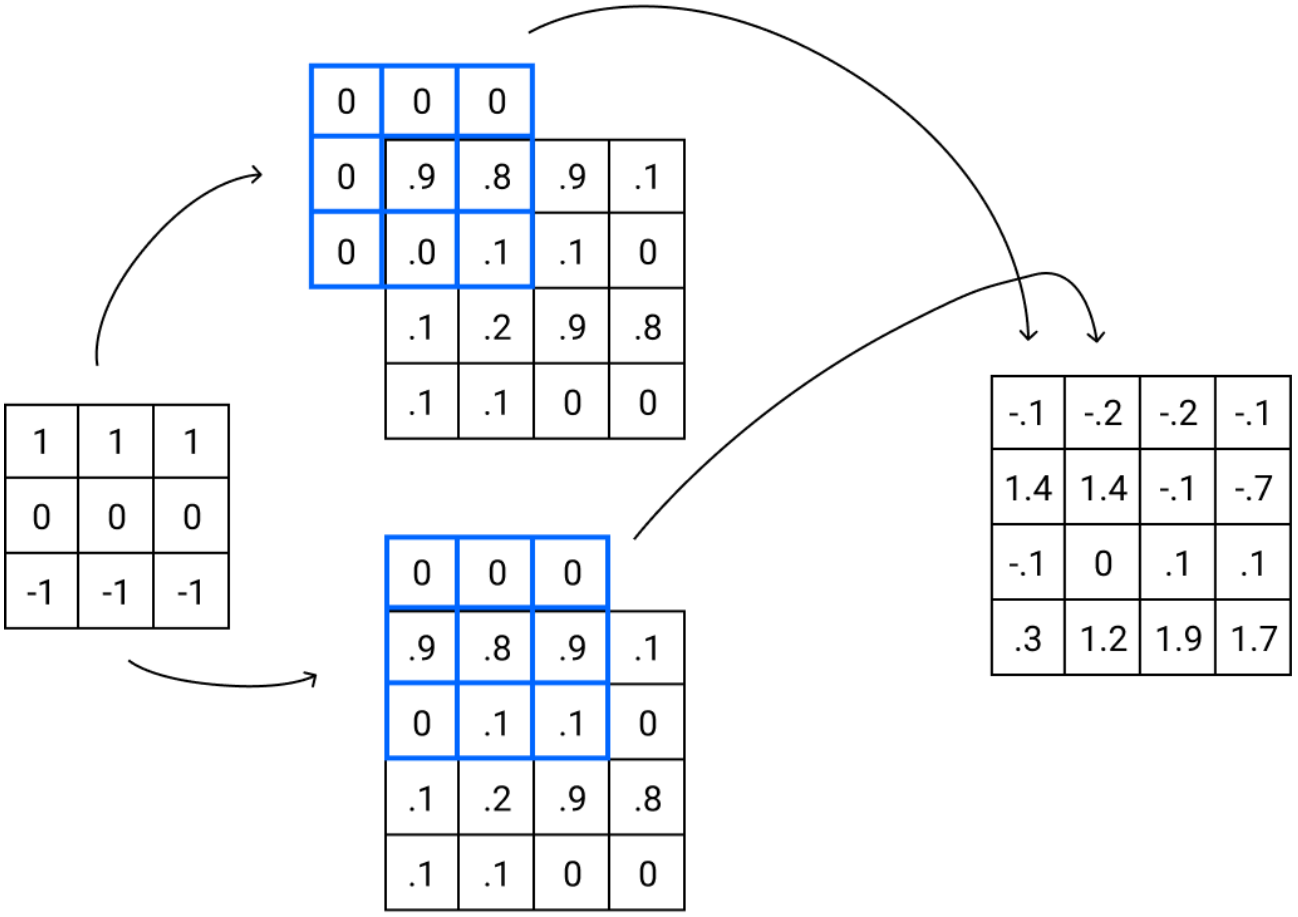


Figure 18. Illustration of the application of 3x3 filters on a 4x4 matrix.

3.2. Pooling

In some cases, reducing the dimensions of the image achieves better expression of the image characteristics, as this eliminates noise and reduces image complexity.
The most well-known method of pooling is MaxPooling. If we define that the filter is 2x2 in size with a displacement of 2 pixels:

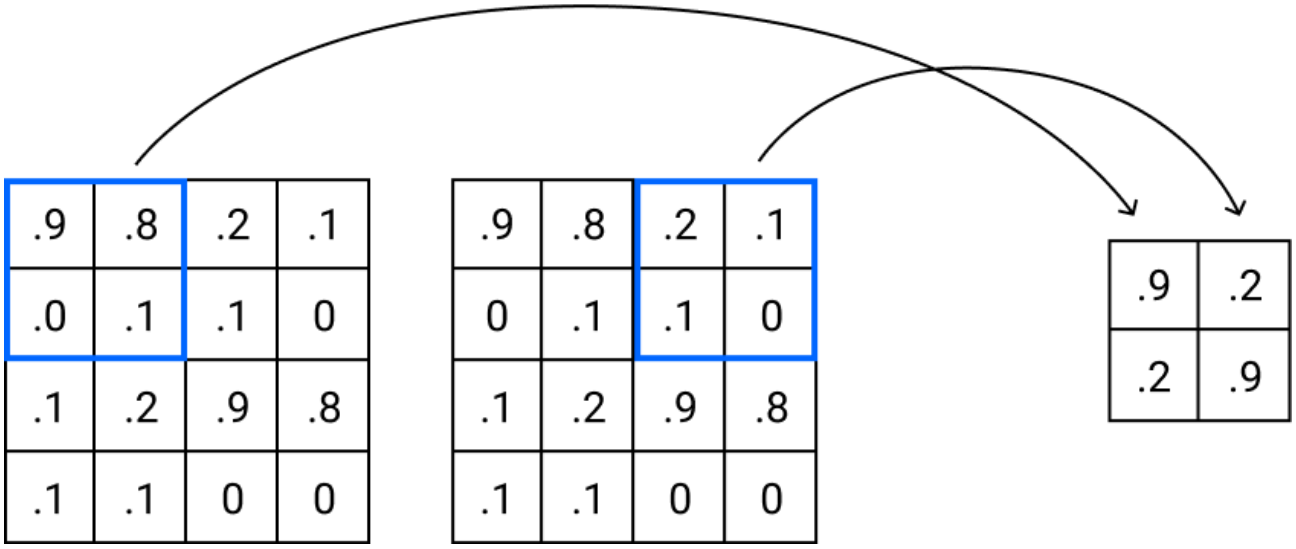


Figure 19. Display of pixel association using 2x2 filters and displacement of 2 pixels.

Convolutional networks can have multiple feature filters and multiple pooling filters before being transferred to a fully connected layer.

3.3. Fully Connected Layer

Although convolutions are good for recognizing different image features, they are not good at distinguishing one class of images from others. That is why traditional neural networks are used. To turn filters into a layer of nodes we need to align them. Flattening is the process of converting multidimensional matrices into a one-dimensional vector.[29] Since each filter will influence the final decision of the model, the aligned filters are merged into one large vector. If we had 2 filters with 2x2 dimensions, the result is one vector with 8 numbers.

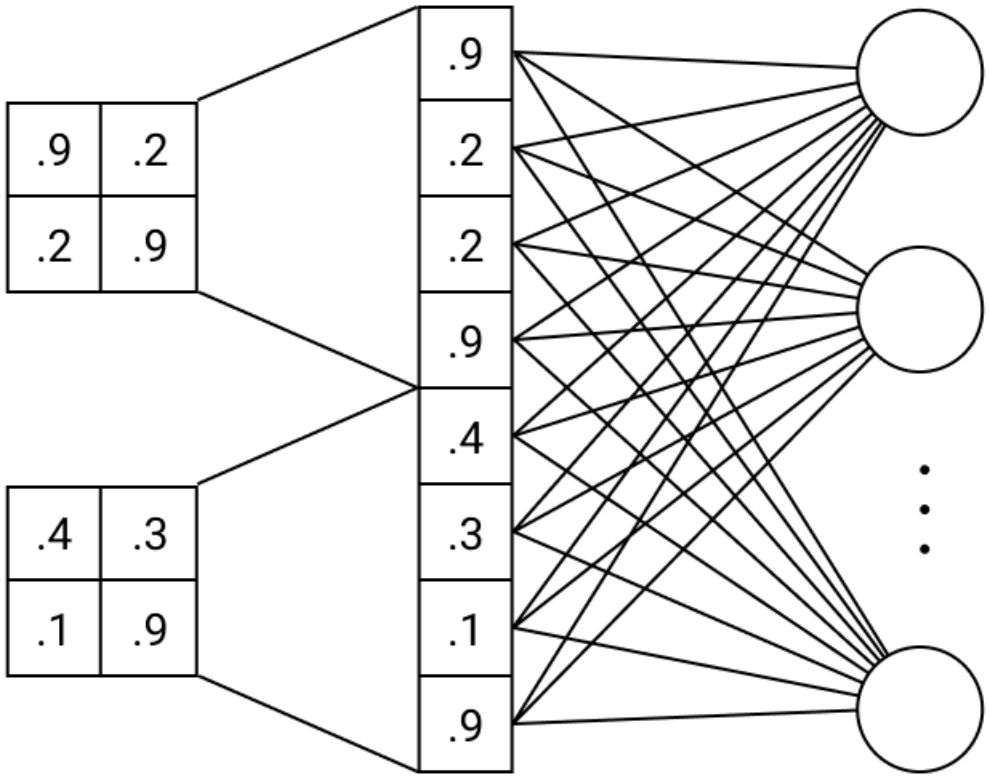


Figure 20. Alignment of filters and forwarding into a fully connected layer.

3.4. Learning Filters

The process of finding or learning filters is done in a similar way as finding the weights in neural networks.
We will consider a simple model that contains one filter, and one node of a fully connected layer.

C = error calculation function
y = correct result
w_i^F = connection strength in a fully connected layer
w_{jk}^C = strength of one pixel in the convolutional layer filter
z^F = input to the node of a fully connected layer
z^C = input to the pixel of the convolutional layer
g^F = activation function for fully connected layer
g^C = activation function for a convolutional layer
a^F = output of activation function of fully connected layer
a_i^C = output of the activation function of convolutional layer filter
r = learning rate

To calculate the error we will use the square difference between the exact value and obtained value.

$$C = (y - a^F)^2$$

Activation output g^F in a fully connected layer:

$$a^F = g^F(z^F)$$

Input to a node in a fully connected layer:

$$z^F = \sum_{i=0}^{I} a_i^C w_i^F$$

Activation output functions g^C in the convolutional layer:

$$a_i^C = g^C(z_i^C)$$

Filter output in the convolutional layer:

$$z_i^C = \sum_{j=0}^{J} \sum_{k=0}^{K} x_{jk} w_{jk}^C$$

If we want to find how the value of w_{jk}^C in the convolutional layer filter changes with respect to the error:

$$\frac{\partial L}{\partial w_{jk}^C} = \frac{\partial L}{\partial a^F} \frac{\partial a^F}{\partial z^F} \frac{\partial z^F}{\partial a_i^C} \frac{\partial a_i^C}{\partial z_C} \frac{\partial z^C}{\partial w_{jk}^C}$$

The following example shows how a pixel w_{11}^C u filter changes in relation to the error L:

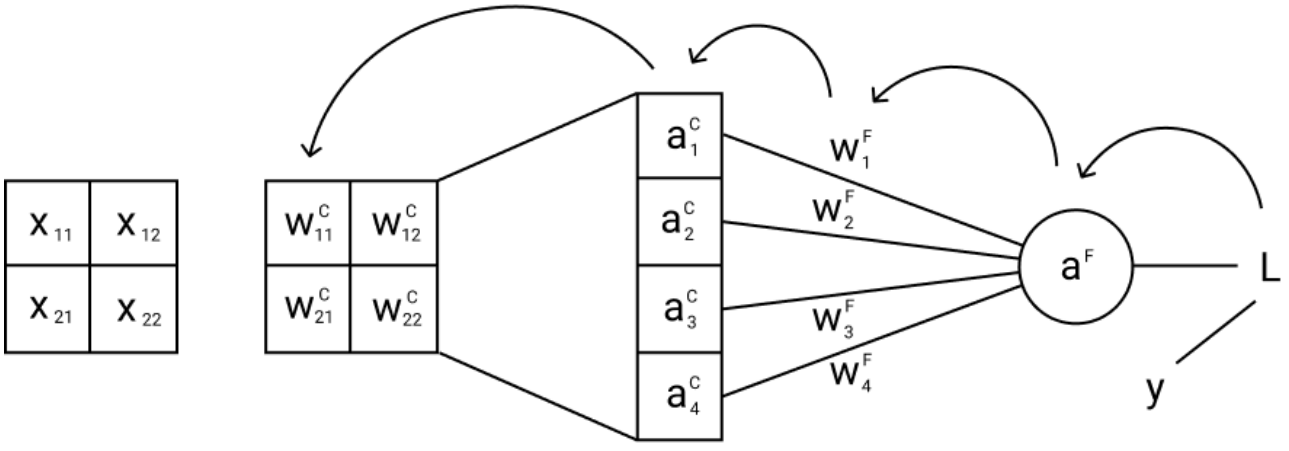


Figure 21. Example of backpropagation in convolutional layers.

$$\frac{\partial L}{\partial w_{11}^C} = 2\,(y - a^F)\,(g'^F(z^F))\,w_1^F\,(g'^F(z_1^C))\,x_{11}$$

And then the value is updated in the direction in which the error is smaller, with the step size r:

$$w'^{C}_{11} = w^{C}_{11} - r\frac{\partial L}{\partial w^{C}_{11}}$$

4. Problem

We solve the classification problem handwritten numbers using neural networks. The database we use is called MNIST (Modified National Institute of Standards and Technology-MNIST) and contains 65,000 copies of digits from 0 to 9. Each example carries a label that tells what number it is.[30]

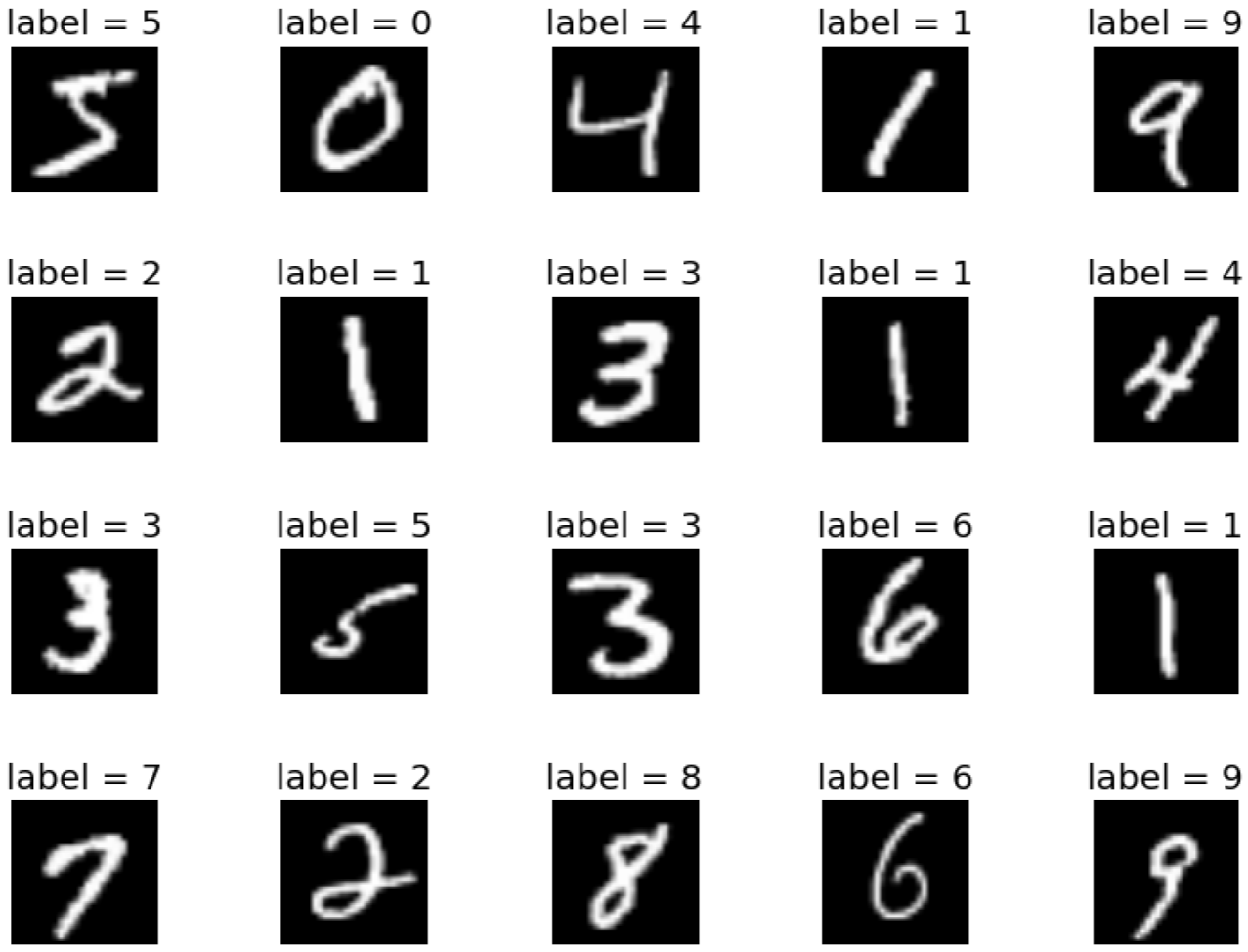


Figure 22. Primers from the MNIST database.

Examples are gray-scale images measuring 28x28. Each pixel is represented by a number from 0 to 255, which in our case will be converted to a value between 0 and 1 and represented as a matrix.

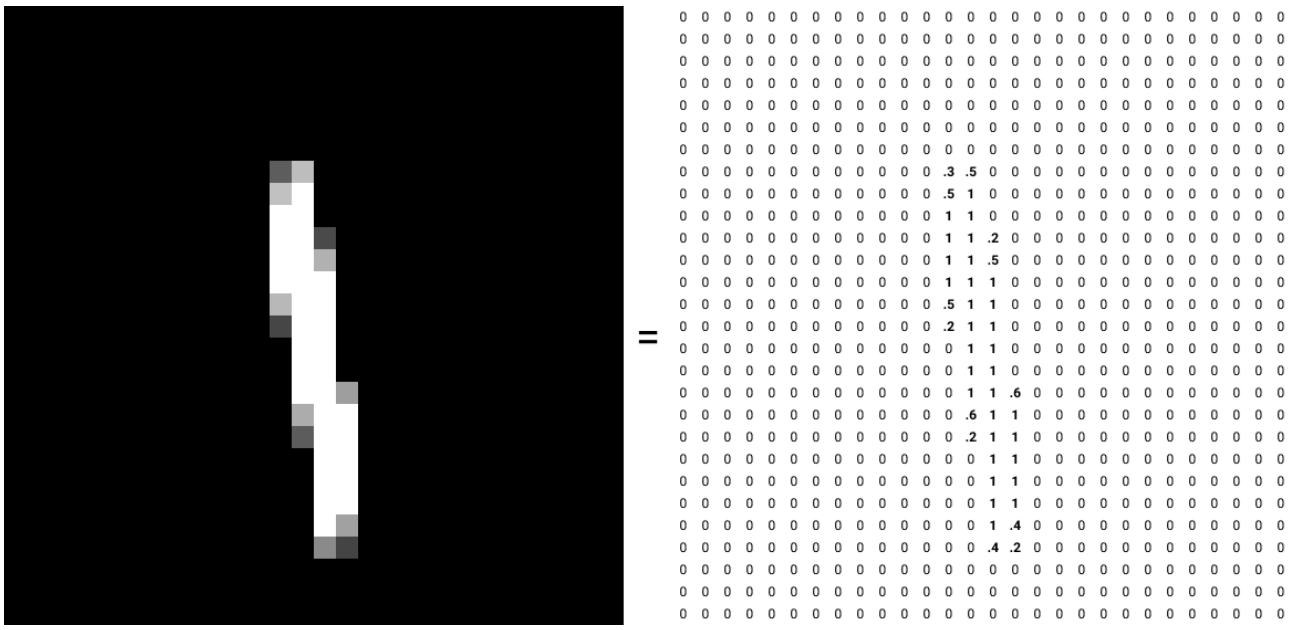


Figure 23. Representation of the image in the form of a matrix.

The data will be divided into two groups. Training group and validation group. This procedure is standard in the field of machine learning because it allows us to better generalize the model. Leaving some of the data for performance testing gives us a clearer picture of how the model will behave over data it has never seen, which can help us improve our model.

The model has 3242 parameters and the architecture consists of:

1) Layer with 32 filters size 5x5. The activation function for this layer is RELU.
2) Layers for merging 2x2 pixels and shifting by 2 pixels. The output is a 14x14 matrix.
3) Another layer with 32 5x5 filters. The activation function for this layer is RELU.
4) A second layer to join a 2x2 filter and move by 2 pixels. The output is a 7x7 matrix.
5) Alignment layer. Since the input of the matrix is 7x7 and the number of filters from the previous layer is 32, the size of the vector is 7x7x32 = 1568.
6) Fully connected layer with 10 neurons. Each neuron represents one class (0 to 9). The activation function is Softmax because the sum of the outputs of all neurons will be 1, which gives us a probability vector which is suitable for classification.

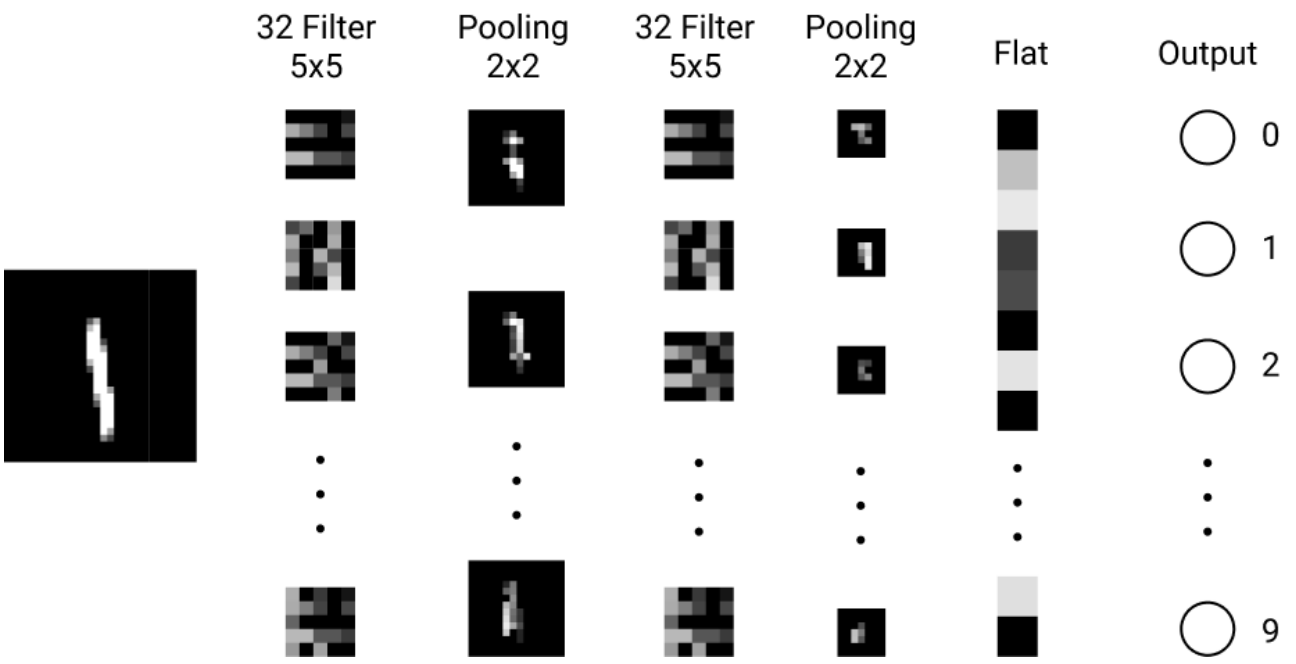


Figure 24. The architecture of our model.

The training lasts for 10 epochs, which means that the entire training set will go through the model, 10 times. The model classifies data for validation in 97.6% of cases. The post-training error was reduced to 0.08.

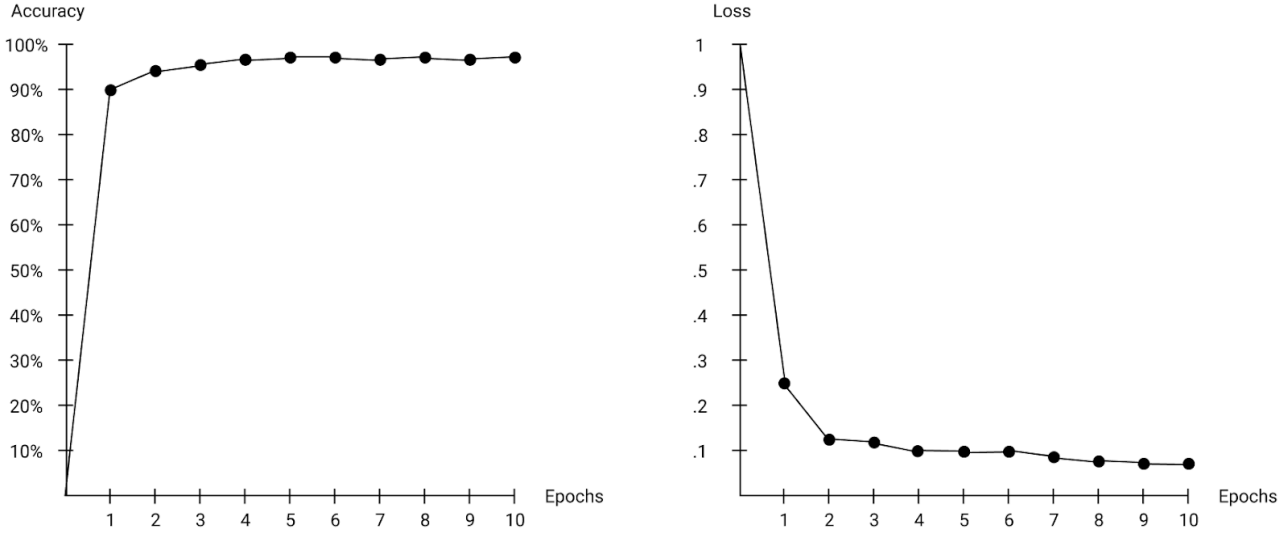


Figure 25. History of accuracy and error of our model during 10 epochs over validation data.
Examples of misclassified images:

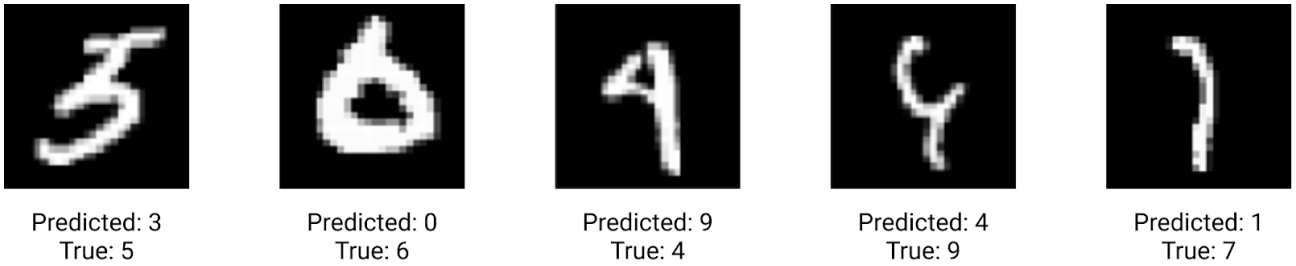


Figure 26. Examples of misclassified images.

We see that a large percentage of digits that are misclassified have the characteristics of multiple digits, which explains why the model is not able to classify a digit with certainty. e.g. the first digit contains the characteristics of digits 3 and 5. Since even a person cannot say with certainty which digit is in question, there is a chance that the examples are not well marked, in which case it would mean that the model recognizes digits better than humans.

One of the biggest drawbacks is that with the addition of noise, the model can misclassify the image.[31]

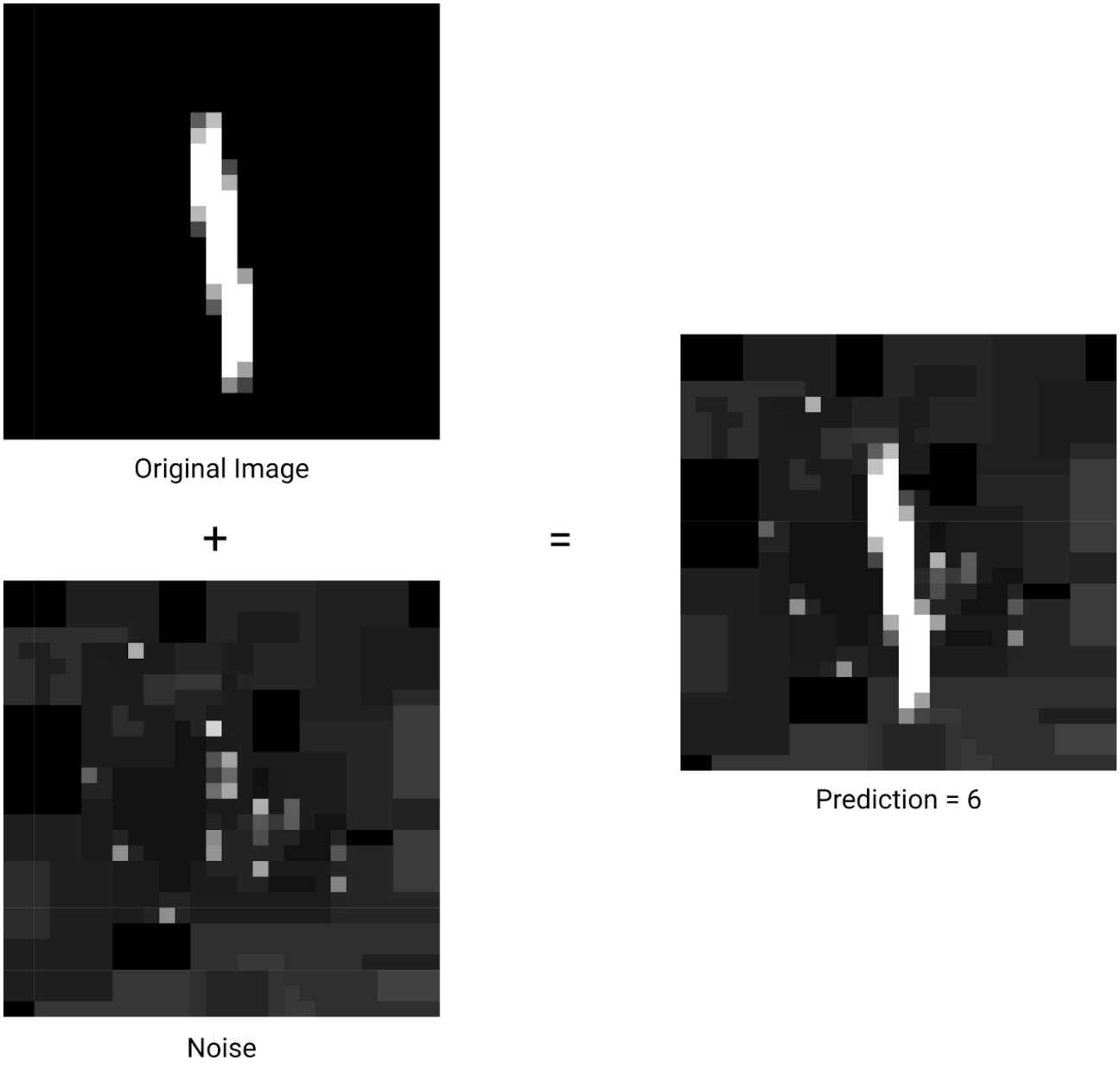


Figure 27. The result of adding noise to one of the examples.

This happens because the model does not know abstract concepts such as line, circle, or any other shapes. In order for a model to know about the concept of a circle or some other shape, it must have many more parameters and a much larger database available that contains many more real-world concepts. All the model sees are numbers, and the characteristics are made based on which numbers give the best result, and sometimes it doesn't have to be something that would be logical for a person. Therefore, by adding a little noise the model sees a completely different figure, although one sees a very small difference.

4.1. Known models

4.1.1. LeNet

This is a popular convolutional model made in the ‘90s. Today it is the standard for beginners who want to enter this field. It consists of 8 layers and 60,000 parameters and can be run on processors. It reaches an accuracy of over 98.5% and an error of 0.06.[32]

Table 1. LeNet model architecture.

Layer	Item	Size layer	The filter size	Move	Trigger function
Input	-	28x28	-	-	-
Convolution	6	28x28	5x5	1	tanh
Pooling	6	14x14	2h2	2	tanh
Convolution	16	10x10	5x5	1	tanh
Pooling	16	5x5	2h2	2	tanh
Convolution	120	1x1	5x5	1	tanh
Fully Connected	-	84	-	-	tanh
Output	-	10	-	-	softmax

4.1.2. AlexNet

Won the 2012 ILSVRC competition. It has 60 million parameters and 650 thousand neurons. Although this model is designed for databases with 1000 classes and a million copies, it achieves over 97% accuracy on MNIST data.[33]

Table 2. AlexNet model architecture.

Layer	Item	Size layer	The filter size	Move	Trigger function
Input	-	227x227	11x11	-	-
Convolution	96	55x55	3x3	4	relu
Pooling	96	27x27	5x5	2	relu
Convolution	256	27x27	3x3	1	relu

Pooling	256	13x13	3x3	2	relu
Convolution	384	13x13	3x3	1	relu
Convolution	384	13x13	3x3	1	relu
Convolution	256	13x13	3x3	1	relu
Pooling	256	6x6	3x3	2	relu
Fully Connected	-	9216	-	-	relu
Fully Connected	-	4096	-	-	relu
Fully Connected	-	4096	-	-	relu
Output	-	1000	-	-	softmax

4.1.3. VGG

Won the second place in the ImageNet competition in 2014. It has 138 million parameters and achieves an accuracy of 98.4% on MNIST data.[34]

Table 3. VGG model architecture.

Layer	Item	Size layer	The filter size	Move	Trigger function
Input	-	224x224	-	-	-
2x Convolution	64	224x224	3x3	1	relu
Pooling	64	112x112	3x3	2	relu
2x Convolution	128	112x112	3x3	1	relu
Pooling	128	56x56	3x3	2	relu
2x Convolution	256	56x56	3x3	1	relu
Pooling	256	28x28	3x3	2	relu
3x Convolution	512	28x28	3x3	1	relu

Pooling	512	14x14	3x3	2	relu
3x Convolution	512	14x14	3x3	1	relu
Pooling	512	7x7	3x3	2	relu
Fully Connected	-	25088	-	-	relu
Fully connected	-	4096	-	-	relu
Fully connected	-	4096	-	-	relu
Output	-	1000	-	-	softmax

4.2. Known Libraries

Since neural networks have not had much change over the years in terms of the math that drives them, it would not be productive for anyone who wants to use neural networks to write the whole code from scratch.

4.2.1. Tensorflow

Developed and used by Google for internal purposes since 2011. In 2015, the source code became available for free. Its popularity has grown from the very beginning and today it is the standard for making neural networks because it is optimized to work on graphics cards, which significantly speeds up training time. It is available in Python, C++, Javascript, Java, and Go programming languages.

4.2.2. PyTorch

The first version came out a year after Tensorflow and soon became its main competitor. It gained popularity when Tesla announced that it was using it for its autonomous vehicles. Companies usually opt for PyTorch because of their dislike of Google. It is only available in the Python programming language and is optimized for graphics cards.

4.3. Use in production

In production, these models are used for the purpose of recognizing numbers from some documents, street signs, license plates, etc. Google uses these models to recognize street numbers on buildings, to make a more accurate map and better address search.[35]

Figure 28. Examples of real-world street numbers.

Although it is possible to make models that can recognize multiple digits in a single image or images with certain noise, which is often the case in real-world data, it would require a model with many parameters, and therefore longer training. If we were to use our model, we would first have to process the images in a certain way before training.

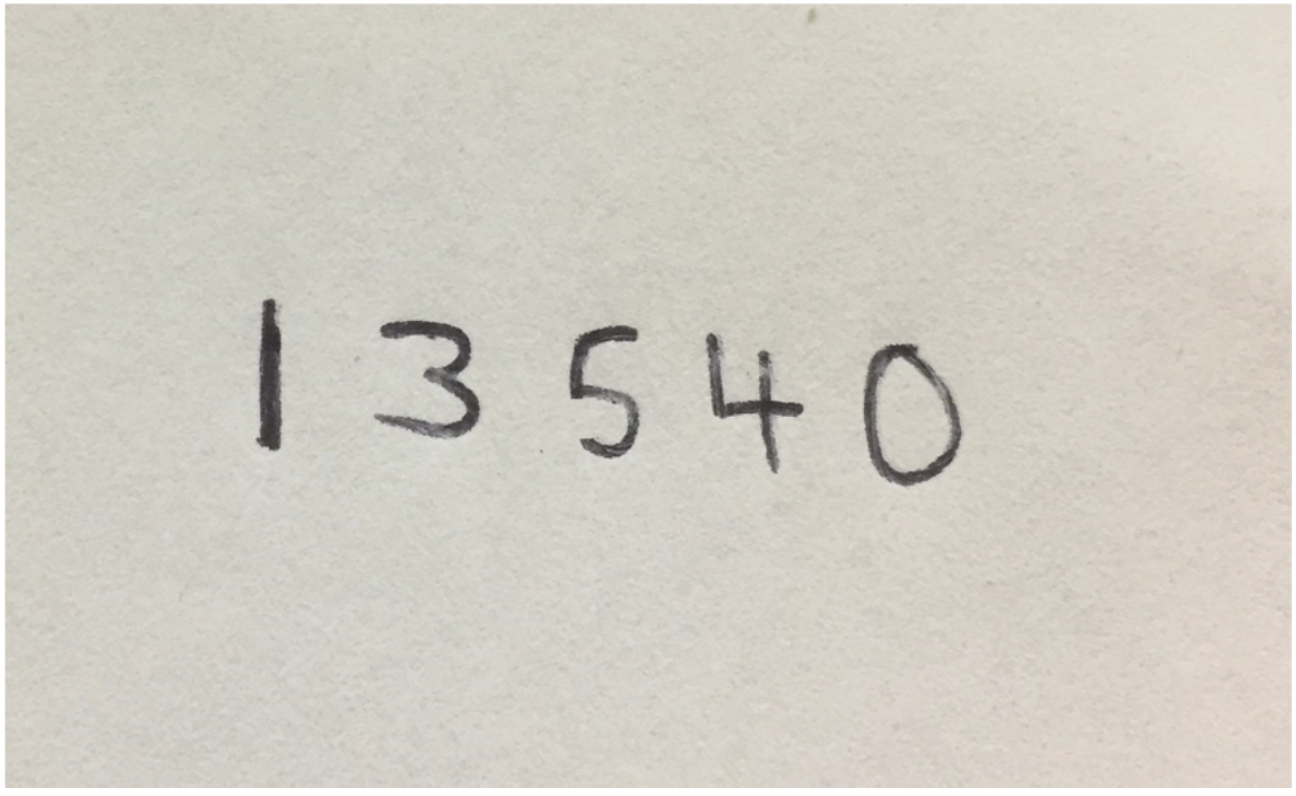


Figure 29. Examples of handwritten numbers from the real world.

As a first step, it is necessary to cut the image so that we isolate each digit that can be achieved using another model or using some image processing techniques. Then convert the image to black and white format and, if necessary, rotate the image and remove noise.

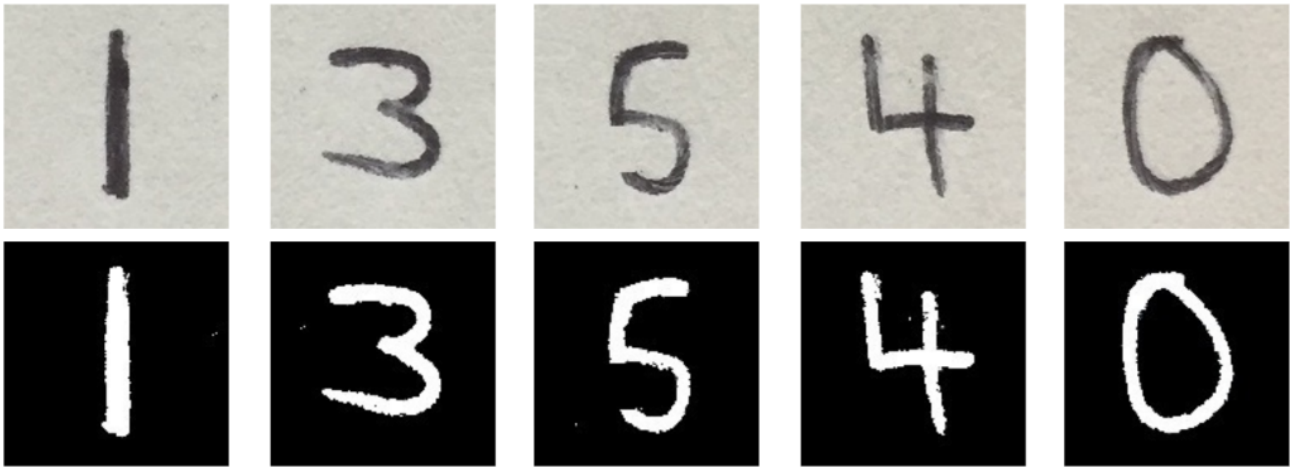


Figure 30. Display images after cropping and converting them to gray-scale format.

Since our input model receives 28x28 images, it is necessary to transform the image. With this treatment, we also eliminate certain noises.

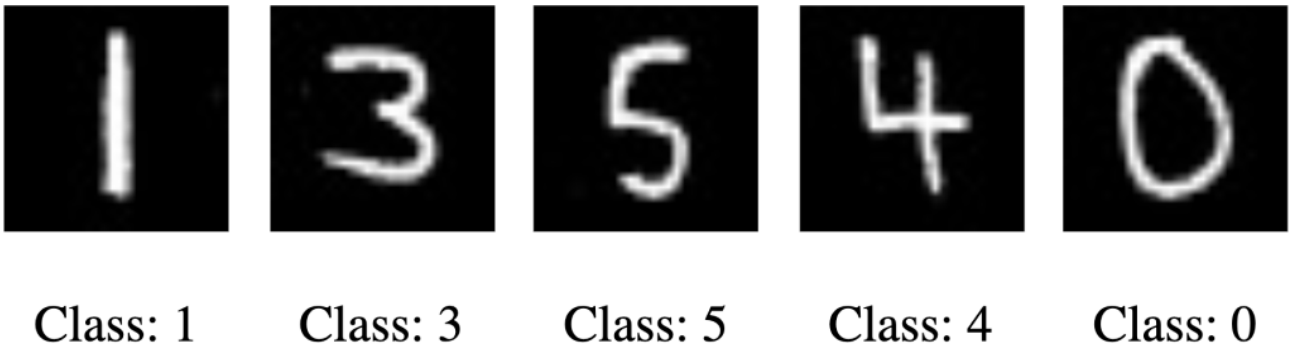


Figure 31. Classification of processed images.

Noise making can also be done by backpropagation, but in our case, it was done manually.

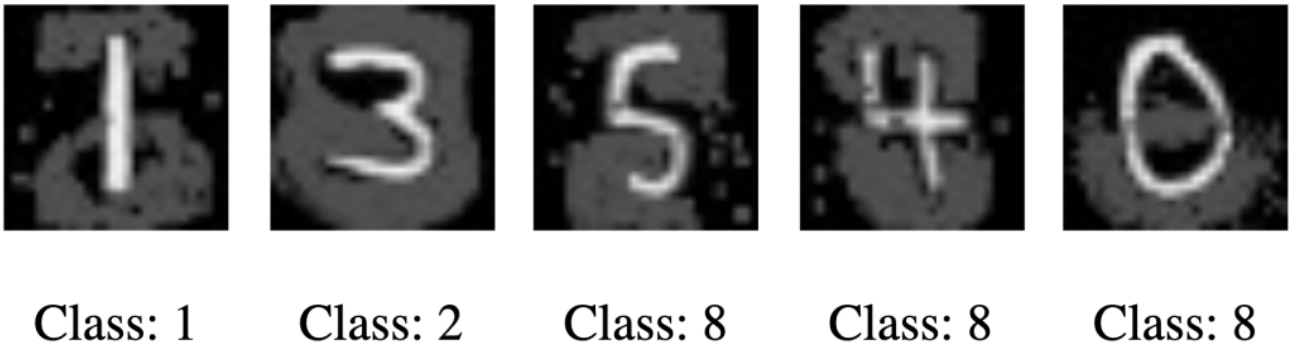


Figure 32. Example of misclassified digits with noise.

5. Conclusion

Fundamental mathematical operations are rarely changed because it is difficult to make an innovative method and because the performance of a method is not immediately obvious. The best example is GAN, which was invented by Goodfellow in 2014. The model is used to generate new examples based on the data over which he was trained, e.g. to make a picture of a human face. Initially, the model failed to generate good examples, but by adding more data and exploring new architectures, good performance can be achieved.

The biggest factor in the success of the model is the architecture of the model, the quality and quantity of the data. When it comes to architecture, it's not important to just add more layers or increase the number of neurons. Architecture is decided on the basis of the problem to be solved, e.g. very well known AlexNet has 60 million parameters, but it turned out worse over MNIST data than our model which has only 3242 parameters, but that is why it is much better on data with a million copies and 1000 classes. It can be said that model making is more researching architectures and data processing than programming the model itself.
Also, we cannot expect the model to find templates on any data we give it. Before training, the data must be processed in such a way that it is easy for the model to extract useful characteristics because that is the greatest condition for success. This does not necessarily mean that the data should not have some kind of noise; as we have seen, a model can be deceived if we give it an image with noises for input without him being trained to recognize them. It is even recommended that the examples be modified before training so that they contain certain noises that we can expect from the real world, but that the example does not lose its characteristics.

6. References

1. Goodfellow, I., Bengio, Y., Courville, A., (2016), *Deep Learning*, p. 96-101, https://www.deeplearningbook.org/contents/ml.html
2. Foote, K. (2019), "A Brief History of Machine Learning", *Dataversity*, https://www.dataversity.net/a-brief-history-of-machine-learning/
3. Foote, K. (2019), "A Brief History of Machine Learning", *Dataversity*, https://www.dataversity.net/a-brief-history-of-machine-learning/
4. Brownlee, J. (2019), "A Gentle Introduction to the ImageNet Challenge", *Machine Learning Mastery*, https://machinelearningmastery.com/introduction-to-the-imagenet-large-scale-visual-recognition-challenge-ilsvrc/
5. Stecanella, B. (2017), "An Introduction to Support Vector Machines (SVM), *MonkeyLearn*, https://monkeylearn.com/blog/introduction-to-support-vector-machines-svm/
6. Panchal, S. (2017), "k Nearest Neighbor Classifier (kNN) - Machine Learning Algorithms", https://medium.com/@equipintelligence/k-nearest-neighbor-classifier-knn-machine-learning-algorithms-ed62feb86582
7. Greenacre, M. (2008), "Measures of Distance Between Samples", http://econ.upf.edu/~michael/stanford/maeb4.pdf
8. Brid, R. (2018), "Decision Trees - A simple way to visualize a decision tree", *GreyAtom*, https://medium.com/greyatom/decision-trees-a-simple-way-to-visualize-a- decision-dc506a403aeb
9. Zhou, V. (2019), „A Simple Explanation of Gini Impurity", https://victorzhou.com/blog/gini-impurity/
10. Goodfellow, I., Bengio, Y., Courville, A., (2016), *Deep Learning*, pp 105-108, https://www.deeplearningbook.org/contents/ml.html
11. Pant, A. (2019), „Introduction to Linear Regression and Polynomial Regression", *Towards Data Science*, https://towardsdatascience.com/introduction-to-linear-regression-and-polynomial-regression-f8adc96f31cb
12. Mehlig, B., „Artificial Neural Network", https://arxiv.org/pdf/1901.05639.pdf
13. Jaksa, R., Katrak, M. (2015), „Neural Network Model of the Backpropagation Algorithm", pp. 1-3, https://pdfs.semanticscholar.org/f7de/25eb027083d4e7144c1ef6831baff35d6d06.pdf
14. Brownlee, J. (2019), „What is Deep Learning?", https://machinelearningmastery.com/what-is-deep-learning/
15. Goodfellow, I., Bengio, Y., Courville, A., (2016), *Deep Learning*, pp. 367-369, https://www.deeplearningbook.org/contents/rnn.html
16. Surmenok, P. (2017), „Estimating an Optimal Learning Rate For a Deep Neural Network", *Towards Data Science*, https://towardsdatascience.com/estimating-optimal-learning-rate-for-a-deep-neural-network-ce32f2556ce0
17. Brownlee, J. (2018), „Difference Between a Batch and an Epoch in a Neural Network", *Machine Learning Mastery*, https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/
18. Weng, L. (2018), „Meta-Learning: Learning to Learn Fast", *Lil'Log*, https://lilianweng.github.io/lil-log/2018/11/30/meta-learning.html
19. Torres, J. (2018), „Learning process of a neural network", *Towards Data Science*, https://towardsdatascience.com/how-do-artificial-neural-networks-learn-773e46399fc7
20. Sharma, A. (2017), „Understanding Activation Functions in Neural Networks", https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0
21. Goodfellow, I., Bengio, Y., Courville, A., (2016), *Deep Learning*, pp. 224-225, https://www.deeplearningbook.org/contents/regularization.html
22. Goodfellow, I., Bengio, Y., Courville, A., (2016), *Deep Learning*, https://www.deeplearningbook.org/contents/regularization.html
23. Jordan, J. (2018), „Normalizing your data (specifically, input and batch normalization)", https://www.jeremyjordan.me/batch-normalization/
24. Goodfellow, I., Bengio, Y., Courville, A., (2016), *Deep Learning*, pp. 200-211, https://www.deeplearningbook.org/contents/mlp.html
25. De Luca, G. (2020), „Bias in Neural Network", *Baeldung*, https://www.baeldung.com/cs/neural-networks-bias
26. Kathuria, A., (2018), Intro to optimization in deep learning: Gradient Descent", *Paperspace*, https://blog.paperspace.com/intro-to-optimization-in-deep-learning-gradient-descent/
27. Goodfellow, I., Bengio, Y., Courville, A., (2016), *Deep Learning*, pp. 326-335, https://www.deeplearningbook.org/contents/convnets.html
28. Powell, V., (2015), „Image Kernels explained visually", https://setosa.io/ev/image-kernels/
29. Jeong, J. (2019), „The Most Intuitive and Easiest Guide for Convolutional Neural Network", *Towards Data Science*, https://towardsdatascience.com/the-most-intuitive-and-easiest-guide-for-convolutional-neural-network-3607be47480
30. Olah, C. (2014), „Visualizing MNIST: An Exploration of Dimensionality Reduction", https://colah.github.io/posts/2014-10-Visualizing-MNIST/

31. Jain, A. (2019), „Breaking neural networks with adversarial attacks", *Towards Data Science*, https://towardsdatascience.com/breaking-neural-networks-with-adversarial-attacks-f4290a9a45aa
32. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P., (1998), „Gradient-Based Learning Applied to Document Recognition", http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf
33. Krizhevsky, A., Sutskever, I., Hinton, G., (2012), „ImageNet Classification with Deep Convolutional Neural Networks", https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf
34. Simonyan, K., Zisserman, A. (2015), „Very Deep Convolutional Networks For Large-Scale Image Recognition" https://arxiv.org/pdf/1409.1556.pdf
35. Goodfellow, I., Bulatov, Y., Ibarz, J., Arnoud, S., Shet, V. (2012), „Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks", https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/42241.pdf