Georg-August-Universität Göttingen

Project Report

Advanced Algorithms in Physics: Pattern Recognition I

Authors:	Philip Dietrich Roland Zimmermann
E-Mail:	philip.dietrich@stud.uni-goettingen.de roland.zimmermann@stud.uni-goettingen.de
Processing period:	12.02.2018 - 19.02.2018

Contents

1. Introduction

Automatic recognition of handwritten digits is one of the easiest problems in the field of pattern recognition. In the end of the 1990's, many different approaches gave very good results for this problem [12]. During the last years, neural networks with advanced techniques were able to outperform humans in classification tasks like object recognition in images [7].

This report examines the application of a simple neural network to the task of classifying handwritten digits from the MNIST (Modified National Institute of Standards and Technology) data set [6]. Additionally, morphological image processing methods and their ability to improve classification results are examined.

2. Theory and Methods

2.1. Morphological Image Processing

The word morphology refers to the Greek word for the study of shape [15, p. 63]. The goal of morphological processes is to extract geometrical features in the image. The following sections explain the basic morphological operations and their applications. Finally, the ability of morphological operations to denoise images are explained.

2.1.1. Erosion and Dilation

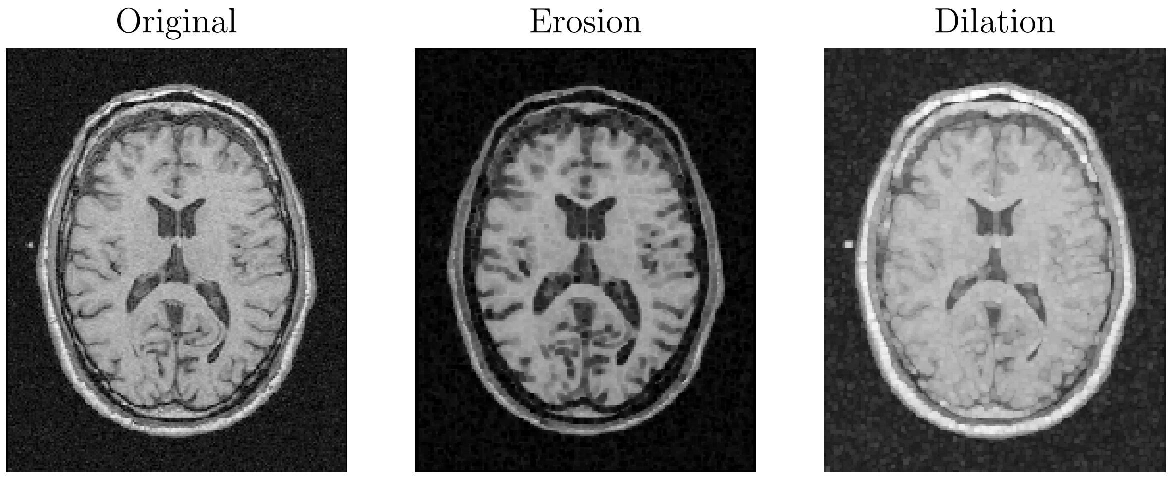
Morphological image processing describes the probing of an image $I \in \mathbb{N}^2$ with a binary matrix $S \in \mathbb{N}^2$. There are basically two different morphological operations. The first one is called dilation $\delta(I, S)$ and is defined as [15, p. 71]

$$\delta(I, S)(x) = \max_{s \in S} I(x + s). \tag{1}$$

Similarly, the erosion operation $\epsilon(I, S)$ is defined as [15, p. 71]

$$\epsilon(I, S)(x) = \min_{s \in S} I(x + s). \tag{2}$$

Considering equations 1 and 2, one finds a nice illustration of morphological operations. Each pixel is replaced by the maximum (dilation) or minimum (erosion) intensity value in its neighborhood. The neighborhood is defined by the structuring element S. Typical shapes of structuring elements are squares or circles. Depending on the problem, there are many more shapes, that can be used.



Original

Erosion

Dilation

Figure 1: Effects of the dilation and erosion on an example image using a quadratic structuring element. While the erosion shrinks the structures of the image, the dilation enlarges them.

One detail about morphological operations is, how they are applied on the edges of an image. As these operations consider a local neighborhood of pixels, edge pixels have to be treated differently. One typical idea for that is to use so called zero-padding [11]. Additional pixels with zero intensity are added to the edge of the image, such that the morphological operations can be applied as they are to the rest of the image. Another way to overcome the problem of boundaries, would be to define the resulting image, such that it is reduced in size. Therefore, the morphological operations are applied on the part of the image where the structuring element fits onto [11].

The effects of those operations on a real image can be seen in Figure 1. The left image shows the original image, the middle one the eroded image and the right one the dilated image using a quadratic structuring element. As the figure shows, an erosion enlarges dark regions in an image. The dilation operation on the other hand enlarges the bright regions in the image.

2.1.2. Morphological Gradient

The morphological operations can be combined in different ways. One possible combination is called morphological gradient *grad(I)*, which is defined as

$$grad(I) = \delta(I, S) - \epsilon(I, S). \tag{3}$$

To represent an image, the intensity values of the result have to be normalized again. Figure 2 shows the result of the morphological gradient operation on the example image for different structuring elements. As can be seen, the morphological gradient can be used to find edges in an image.

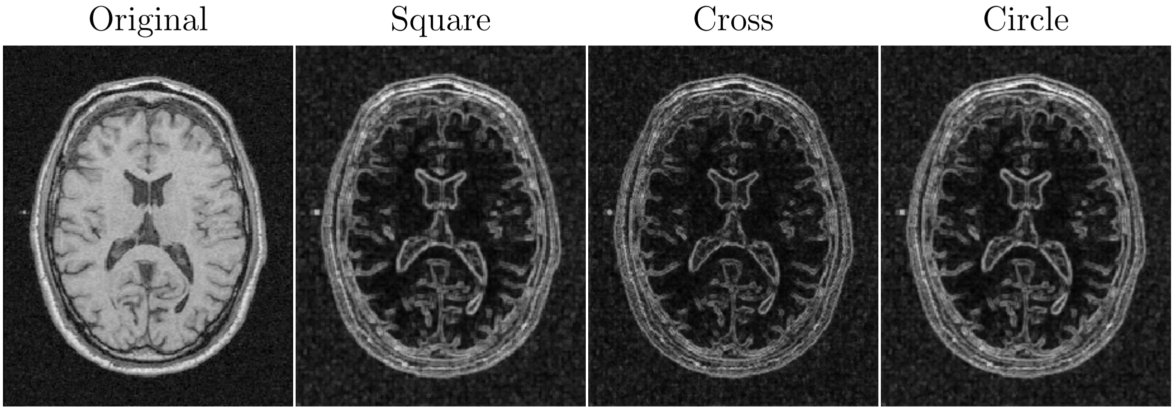


Figure 2: Morphological gradients of the example image for three different structuring elements.

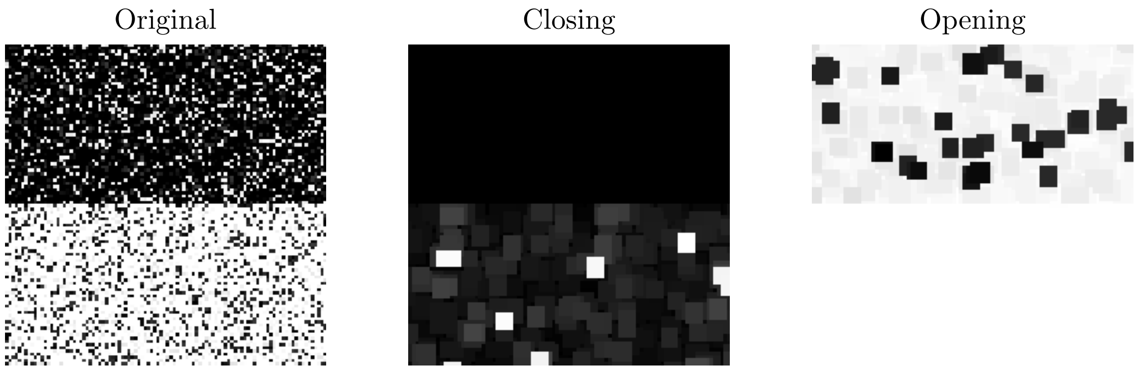
2.1.3. Opening and Closing

Another way to combine erosion and dilation operations is to concatenate them, so that at first one of them is applied and afterwards the other one is applied, too. Those operations are called opening and closing and are defined as [15, p. 68]

$$opening(I, S) = \delta(\epsilon(I, S), S) \tag{4}$$
$$closing(I, S) = \epsilon(\delta(I, S), S) \tag{5}$$

At first glance these operations may seem redundant. While the erosion enlarges dark patterns, dilation will shrink them again. The usefulness of opening and closing operations can be seen by considering their effects on small structures. Imagine a small bright pattern in an image, which can be identified as noise. By applying an erosion with a sufficiently large structuring element, the noisy pixels will be replaced by darker pixels. Therefore, applying a dilation afterwards will not make the bright pattern reappear.

Dark noise on a bright background on the other hand can be reduced using the opening operation. Figure 3 shows an example for a noisy image with a partly dark and partly bright background. As the figure shows, the closing operation completely removes the bright noise, while the opening operation removes the dark noise.



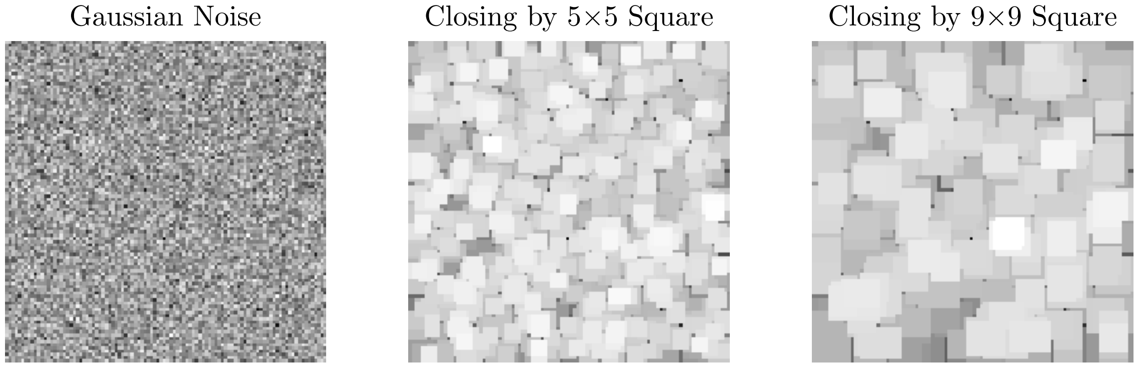
Original

Closing

Opening

Figure 3: Exemplary illustration of the morphological operations to remove salt and pepper (binary, normal distributed) noise. The left image shows the noisy input, the center image the closing by a 5×5 square and the right image the opening by a 5×5 square.

In contrast to the case of high contrast noise, Gaussian noise cannot be removed by a closing operation. Figure 4 illustrates this effect.



Gaussian Noise

Closing by 5×5 Square

Closing by 9×9 Square

Figure 4: Exemplary illustration of the closing operation on Normal distributed noise, to show the failure of the operation. The left image shows the noisy input, the center image the closing by a 5×5 square and the right image the closing by a 9×9 square.

3. Neural Networks

Neural networks are a popular approach to solve pattern recognition and machine learning tasks. They have proven to yield better results than classical and purely statistics based methods [7, 12]. In the past years many different architectures emerged which are specialized for specific tasks. One of this architectures is based on the so called sigmoid neuron. In this section the usage of this network for image classification shall be examined.

3.1. Sigmoid Neuron

In this project, a neural network consisting out out sigmoid neurons shall be used. The sigmoid neuron is a model used in machine learning which tries to mimic the behaviour of a biological neuron. Its structure is shown in Figure 5. Given an input $\vec{x} \in \mathbb{R}^k$ the neuron calculates the real valued output $y \in \mathbb{R}$. The different inputs are mapped onto a real value z by using a so called weight vector $\vec{w} \in \mathbb{R}^k$ and a bias $b \in \mathbb{R}$ by

$$z = \vec{w} \cdot \vec{x} + b. \tag{6}$$

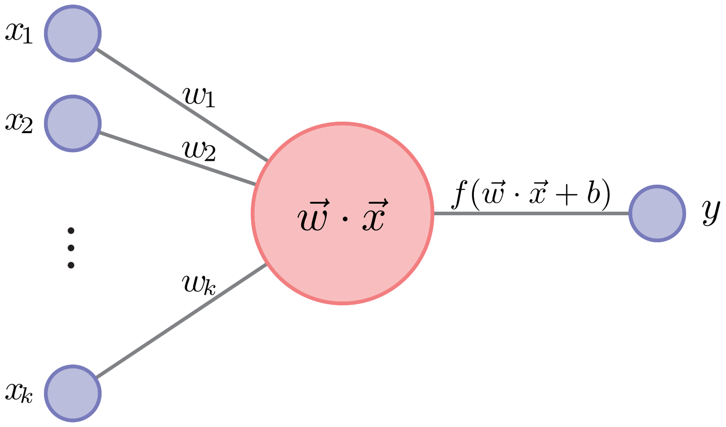


Figure 5: Schematic illustration of a sigmoidal neuron. The input $\vec{x}$ is mapped multiplied by a weight vector $\vec{w}$ before it is transformed into the output y using the non-linear activation function $f(\cdot)$ (according to [13]).

This method is the artificial analogy to the summation method which happens at the soma of a biological neuron [5, Chapter 5]. In a real neuron this value is processed by a non-linear activation function which gives the output of the neuron. Exactly the same is done in the model of the sigmoid neuron by introducing a function $f : \mathbb{R} \to \mathbb{R}$ to computed the final output of the neuron

$$y = f(\vec{W} \cdot \vec{x} + b). \tag{7}$$

This function is chosen to be sigmoidal [5, Chapter 5], i.e. has the form $f(x) = \frac{1}{1+e^{-x}}$. All in all, this allows one to compute the output as

$$y = \frac{1}{1 + e^{-\vec{w} \cdot \vec{x} - b}}. \tag{8}$$

To ease the formulation one can hide the bias b in the weight vector $\vec{w}$ by expanding the input vector $\vec{x}$ with a constant 1, so that it is $\in \mathbb{R}^{k+1}$. Simultaneously one also expands the weight vector by an additional weight, which then describes the bias.

3.1.1. Network Architecture

The previously introduced sigmoid neuron is able to perform a regression, i.e. assign input vectors a real value between zero and one. This output can be used for two label classifications by interpreting the output value as a probability that the input corresponds to class one. When one has to perform a multi label classification, one can use multiple of these neurons together. Each of the neurons shall represent one of the different labels. They all get the exactly same input and output the probability that the input belongs to the corresponding class [14].

For this method the target data, i.e. the labels, have to be one-hot encoded. This means that for a k class classification for each target data a vector $\in \mathbb{N}^k$ is created which consists only out of zeros, but has a single one in the index which corresponds to its label. A better understanding of this method can be obtained by taking a look at the following example for $k = 5$ classes:

$$y = 3 \rightarrow \vec{y} = (0, 0, 0, 1, 0) \tag{9}$$

$$y = 1 \rightarrow \vec{y} = (0, 1, 0, 0, 0). \tag{10}$$

This methods gives a target value for each of the k neurons.

In this project images of the MNIST data set (see section 4.1) shall be examined. To be able to use the introduced sigmoidal network for images, one has to reinterpret the images as vectors: $\mathbf{x} \in \mathbb{R}^{28\times 28} = \vec{x} \in \mathbb{R}^{784}$. This input vectors are then expanded by a constant one to allow a bias. Therefore, for each neuron there is now a $784 + 1 = 785$ dimensional weight vector. A schematic illustration of the entire procedure is shown in Figure 6.

The output of the network is obtained by finding the neuron which gives the highest output value, i.e. the highest class probability. This class label is prediction by the network.

3.2. Stochastic Gradient Descent

To evaluate the quality of a network's prediction one introduces an error function, which is also called loss [14, 5]. This loss is a function of the network's prediction $\vec{y}$ and the target variable $\vec{y}_t$. It is a measurement for the poorness of the network's predictions. The loss is always defined as a function of data. Therefore, one needs to introduce two new notions - the training and the test data set. Those are in the following called $\mathbf{X}_{\text{train}}$ and $\mathbf{X}_{\text{test}}$. While the former is used to train the network's weights, the later is used the evaluate the quality of the network's predictions. For the classification performed in this

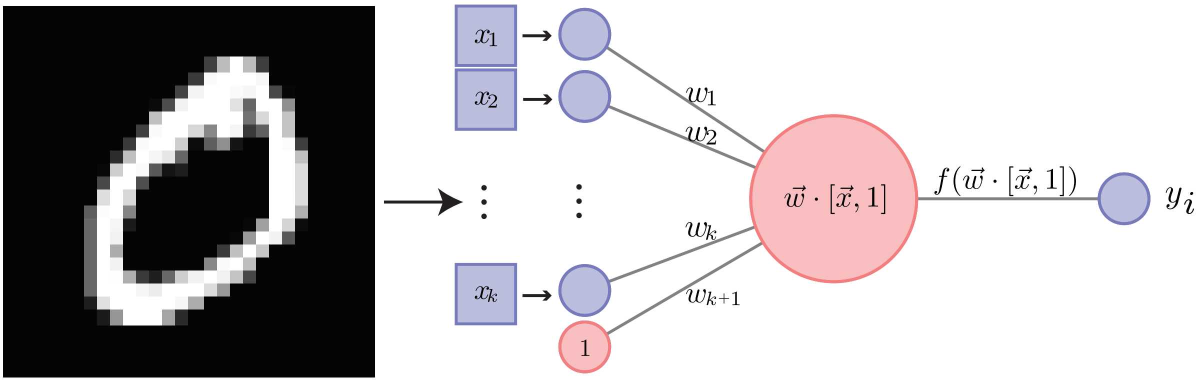


Figure 6: Schematic illustration of the entire network architecture. The network consists out of 10 of the illustrated rows. The input images are reinterpreted as vector $\vec{x}$. A constant value of 1.0 is appended to them. Then they are weighted by the weight vector $\vec{w}$ before the output y_i for class i is calculated by the non-linear activation function of the i-th neuron.

project one can define the loss L as the fraction of falsely classified images

$$L(\vec{w}) = \frac{1}{2|\mathbf{X}|} \sum_{\vec{x} \in \mathbf{X}_{\text{train}}} (\vec{y}(\vec{w}, \vec{x}) - \vec{y}_t(\vec{x}))^2, \tag{11}$$

with the real target value $\vec{y}_t(\vec{x})$ for the input $\vec{x}$. This loss has values between zero and one.

After a loss has been defined, one has to change the network's weights so that this loss function is minimized. This can be done using the gradient descent technique. Here, one calculates the gradient of the loss function with respect to the weights $\frac{\partial L}{\partial \vec{w}}$ of the network. This gradient is then used to update the weights [2, 3]

$$\vec{w} \leftarrow \vec{w} + \eta \frac{\partial L}{\partial \vec{w}}, \tag{12}$$

whereby η is a small constant, called the learning rate.

In the case of vanilla gradient descent, the gradient is always calculated for all sample of the training data. Therefore, one has to perform a lot of calculations to update the weights. When the data set is large enough and the data contains redundant information, one can approximate the gradient for all data by only taking a small fraction of the data set into account. As it is faster to calculate the gradient over a small amount of data, one can perform more update steps in the same time compared the vanilla gradient descent. This modification of the gradient descent is known as the Stochastic Gradient Descent [2, 3]. It can be expressed by modifying the loss function

$$L(\vec{w}, \{\vec{x}\}) = \frac{1}{2|\{\vec{x}\}|} \sum_{\vec{x} \in \{\vec{x}\}} |\vec{y}(\vec{w}, \vec{x}) - \vec{y}_t(\vec{x})|, \tag{13}$$

so that it only takes a subset $\{\vec{x}\}$ of the entire training data $\mathbf{X}$ into account. This allows one to split the training data into small subsets and to calculate the gradient of the loss for each of these sets separately. This small subsets are also called mini batches, and their size is called batch size. Whenever all data of the data set has been processed, one epoch has finished. One often performs the training for multiple epochs. After each epoch the training data set will be randomly shuffled, so that the mini batches are not the same for different epochs. All in all, this method can lead to a faster convergence of the network's weights compared to vanilla gradient descent, when the batch size is chosen appropriate [3].

Another property of this version of the gradient descent is its efficacy: while the vanilla gradient descent can get stuck at local minima the Stochastic Gradient Descent is able to escape local minima of the error surface, as a stationary point of the loss function for the entire training data set does not have to be a stationary point for a small mini batch. Therefore, the stochastic version of the gradient descent might escape from local minima, which allows the network to find global minima with a higher probability [5, Chapter 5].

4. Application and Results

After the most important concepts and methods have been introduced by the previous sections, those will now be used to solve an image classification problem.
This classification problem will be implemented in *Python 3.x* using the default framework for scientific computing consisting out out *numpy* [16], *scipy* [10] for calculations and *matplotlib* [9] for plotting.

The source code can be accessed on github.com/FlashTek/AAIP_PR. The implementation of the morphological operations is done in the folder *task1*. There, different structuring elements are defined as well as the dilation, erosion, opening and closing operation. The implementation of the neural network is done in the folder *task2*. This folder contains a class for the sigmoid neuron and another class for the neural network. Furthermore, there is code to test different hyper parameters as well as the influence of noise on the success rate.

4.1. Data Set

As already mentioned before, this project deals with the problem to classify images of the MNIST (Modified National Institute of Standards and Technology) data set [6]. This data set consists out of 60 000 images showing handwritten digits between 0 and 9. Each of the images has a size of 28×28 pixels. The data set is often split into 50 000 and 10 000 data points for training respectively testing. The data set is balanced. This means, that all of the ten different classes are equally represented in both the training as well as the test data set. An example of these images are displayed in Figure 7.

4.2. Influence of Hyper parameters

The network architecture introduced beforehand has two hyper parameters, the batch size n and the learning rate η. They influence the learning method (Stochastic Gradient Descent) and therefore have in direct influence of the convergence speed of the network's weights. Furthermore, hyper parameters in general can influence the quality of the network's prediction, too. At the beginning, this influence shall be examined and analyzed. Therefore, the network has been trained for different combinations of n and η for 30 epochs. The values tested for the hyper parameters are shown in Table 1. As there are 10 different batch sizes and 5 different learning rates, all in all 50 different networks have been trained and evaluated. The temporal development of the training is shown in Figure 8 and Figure 9. While the former highlights the influence of the batch size n on the loss, the later focuses on the influence of the learning rate η.

Figure 8 shows that the effect of the learning rate η on the learning speed is weak when the batch size n is small. When the batch size is rather large ($n > 50$) the network with

Figure 7: Example images from the MNIST data set for each of the ten different handwritten digits from 0 to 9.

Hyper parameter	Values examined
Batch size n	$\{1, 5, 10, 25, 50, 100, 200, 400, 600, 1000\}$
Learning rate η	$\{0.001, 0.005, 0.01, 0.05, 0.1\}$

Table 1: Enumeration of the values examined for the network's two hyper parameters.

a lower learning rate learn show a worse performance than those with greater learning rates. This can be explained in the following way: The network's weights are updated $|\mathrm{X}_{\mathrm{train}}|/n$ times per epoch. This means, that a small batch size means that the weights are updated more often. Therefore, the learning rate can be small without the weight changes being to little. But when the batch size is big, there are only a few update steps per epoch. Therefore, the learning rate has to be comparable bigger so that the total weight change is not too low.
The same insights can also be obtained from Figure 9. Here, the effect of the learning rate is more highlighted. For larger learning rates ($\eta \in \{0.05, 0.1\}$) the loss does not differ too strongly for different batch sizes n. This does not hold anymore for small η values.

When one compares the development of the loss computed on the training data set with the loss computed on the test data set, one does not see a strong difference. This is a good sign, as this means that there is no overfitting happening. The overfitting effect describes the situation in which a model (network) is fitted too strong on the training data set and is therefore only able to process these samples in a good way. When different, new samples from the test data set are entered, the model cannot process them correctly which causes a higher loss. Fortunately, this does not happen for the networks examined.

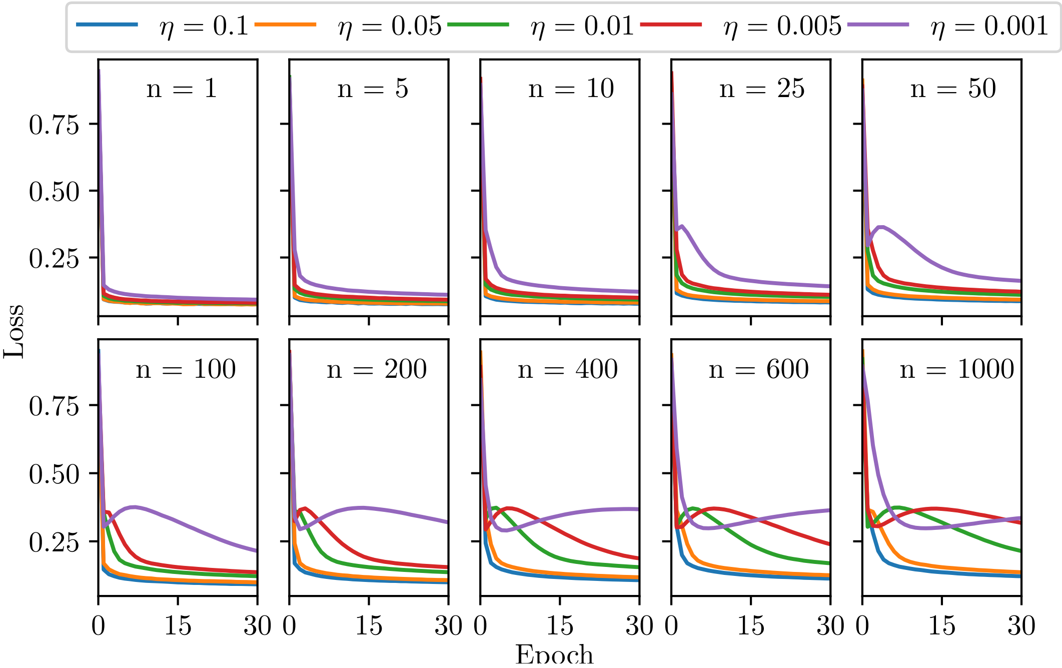


Figure 8: Influence of the learning rate η on the training loss for different batch sizes n. Each box corresponds to a value of n and shows curves for 5 different values of η.

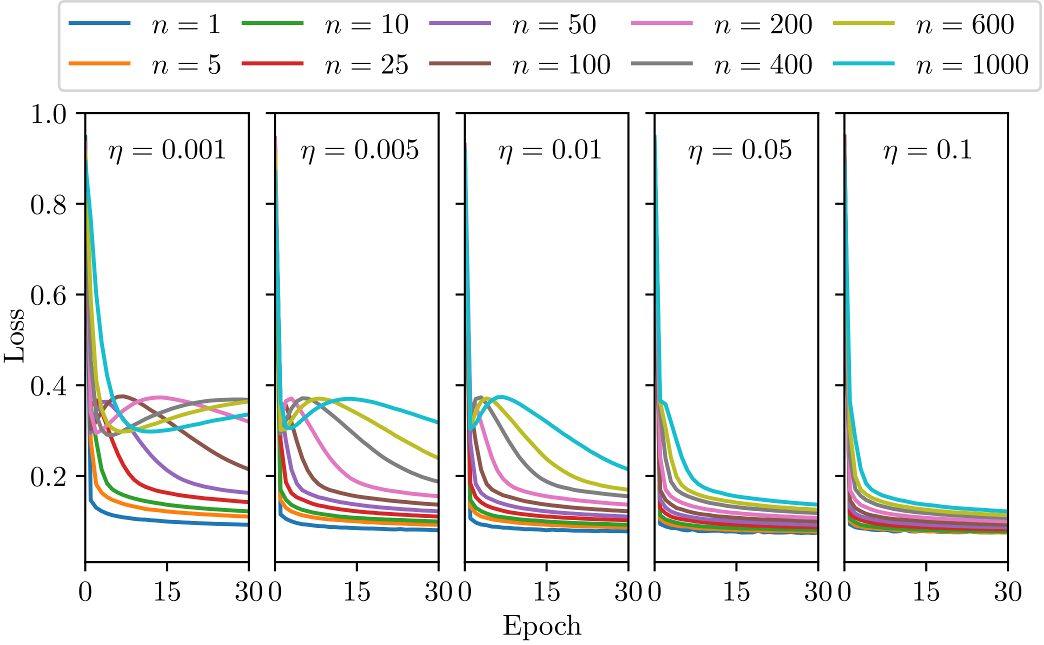


Figure 9: Influence of the batch size n on the training loss for different learning rates η. Each box corresponds to a different value of η and shows curves for 10 different values of n. This is the counterpart to Figure 8.

All in all, the losses seem to converge to a value, i.e. there is not strong fluctuation on the loss's value at the end of the training. This indicates that the network's weights have converged, i.e. the Stochastic Gradient Descent found a (local) minimum. The best performing network (according to the final training loss) has been trained with $n = 10$ and $\eta = 0.01$. It has been able to classify 92.2% of the test samples correctly. This is much less than the (at least to the best of our knowledge) current state of the art approach by Wan et al. [17], which obtained a rate of 99.79% with the network architecture. Compared to the results of LeCun et al. [12] from 1995, this result matches the classical approach used here. But even the more elaborated network architectures, including convolutional neural networks, outperformed this approach by six to seven percentage points [12].

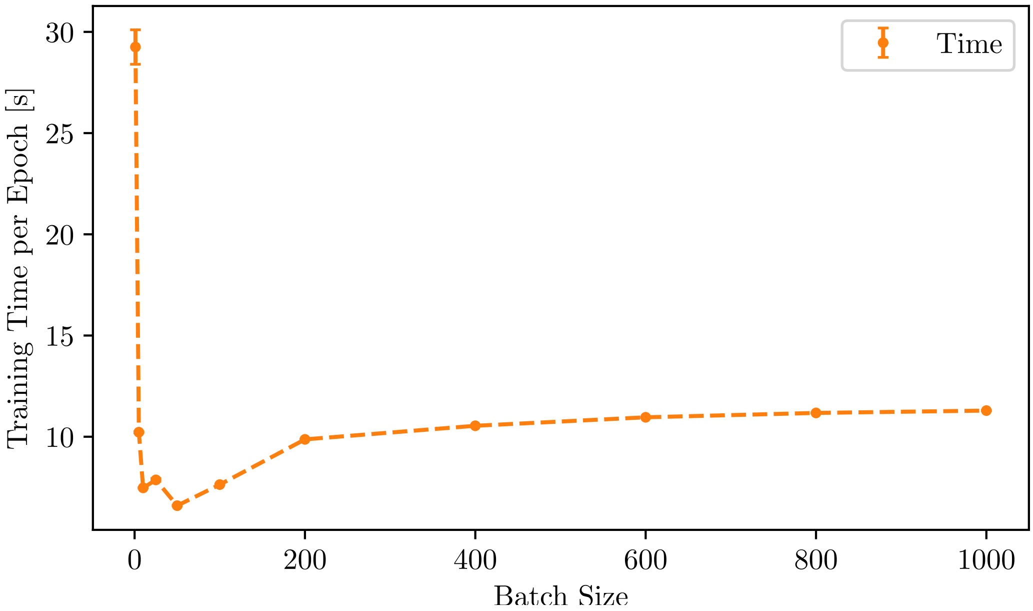


Figure 10: Average time elapsed per training epoch as a function of the batch size n.

Another interesting fact in the context of machine learning is always the training speed of a model, i.e. how long does the model's training actually need? In the used network architecture this question is mainly determined by the size n of mini batches used in the Stochastic Gradient Descent. Figure 10 shows the averaged training time needed per epoch as a function of the batch size n. Only the shape of the graph and not its actual values are interesting here, as the values depend on the hardware used, while the shape itself does only depend on the training method used. The graph shows, that the training can be sped up by increasing the batch size. This fits the previously stated observations, that the number of batches used per epoch decreases with an increasing batch size. Nevertheless, the speed is limited by a lower limit which cannot be surpassed even for great batch sizes n. The graph has a minimum which is at $n = 50$; after this minimum the value slightly increases again and than stays more or less constant for increasing n. This might be caused by the implementation of the linear algebra used in the update steps, as it contradicts the theoretical considerations.

4.3. Influence of Noise

After the effect of the hyper parameters has been examined the robustness of the network shall be analyzed. For the sake of this, the effect of normal distributed noise ($\sim \mathcal{N}(0, 0.01)$) on the test loss shall be observed. For this, five different scenarios (combinations of loss and morphological operations) are used. These are listed in Table 2. Furthermore, the usefulness of the application of morphological operations on the noisy data is examined. As shown before, noise in images can partially be removed by applying a $closing(\cdot)$ operation on the image.

All networks have been trained with the same batch size $n = 10$ and learning rate $\eta = 0.1$ - these are the hyper parameters of the best performing network found in the section beforehand. The network trained in the first scenario N_0 can be used as a benchmark to compare the other networks to. The resulting training and test losses of the networks are displayed in Figure 11. To quantify the quality of the network the most important value is the test and not the training loss. The test loss is for three of the networks (N_0, N_1, N_3) almost identical. For N_2 and N_4 the loss is much higher. Both of the networks were tested on noisy data which have been denoised by the $closing(\cdot)$ operation using a 3×3 cross structuring element. The network N_4 has also be trained on such data. This indicates, that training on denoised data does not improve the network's quality. Surprisingly the loss of network N_1 is lower than those of N_2 and N_4. The network N_1 is also evaluated on noisy data, but no morphological operation has been applied. This shows that the application of these operations does not improve the network's prediction quality but can lower it, too. This might happen when details of the images, which are important for the network's classification, are also removed by these operations.

Network's Name	Situation
N_0	No noise in training and test data
N_1	Noise only in test data
N_2	Morphologically removed noise only in test data
N_3	Noise only in training data
N_4	Morphologically removed noise in training and test data

Table 2: Overview of the five different scenarios used to examine the influence of noisy data on the training and test performance of the network.

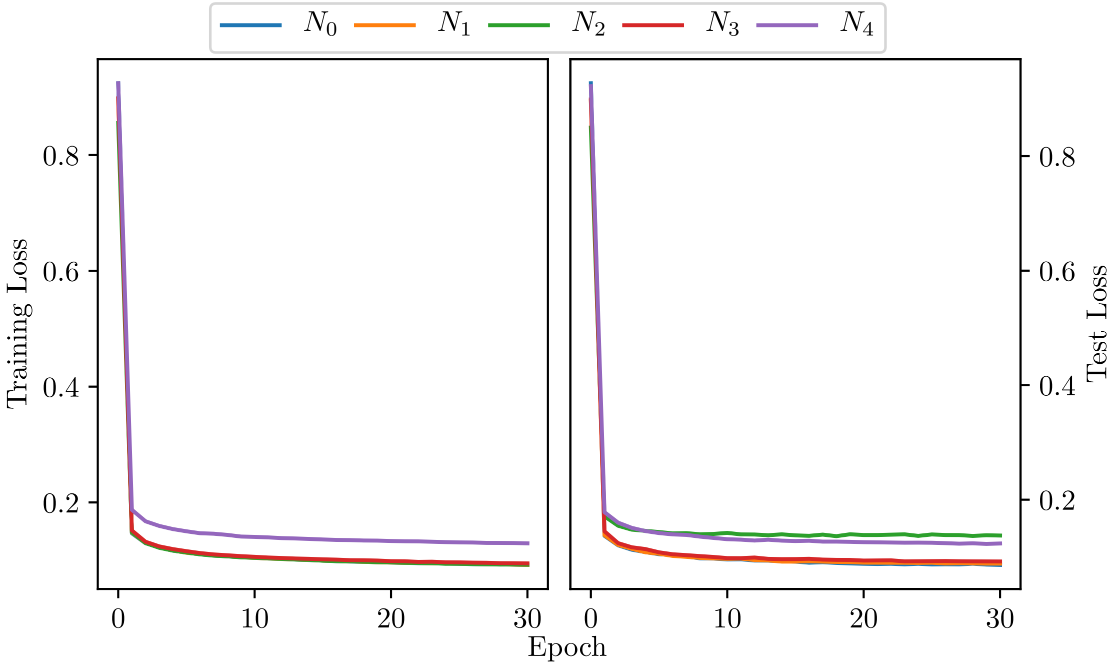


Figure 11: Comparison of the losses of the networks used for the five different scenarios listed in Table 2. The networks for N_3 and N_1 have the both the best training and test losses.

4.4. Visualization of the Network

An interesting possibility to visualize the function of a neural network is to consider its filters, i.e. its weights. Mathematically speaking, each input pixel is multiplied by a certain weight to compute the probability of a sample belonging to a certain class. Therefore, these filters can be interpreted as images, too. Figure 12 shows these filters for each of the ten different output classes for the network trained on no noisy data introduced in section 4.3. To increase the contrast in the image the weights displayed here have been normalized.

Considering these filters, one finds a strong resemblance between the input images and the network weights. In each of the filters one finds, that bright pixels support classifications into a certain class, while dark pixels reduce this probability. In each of the filters displayed one can see certain regions which have a high weight, which means that these regions are characteristic for the output class.

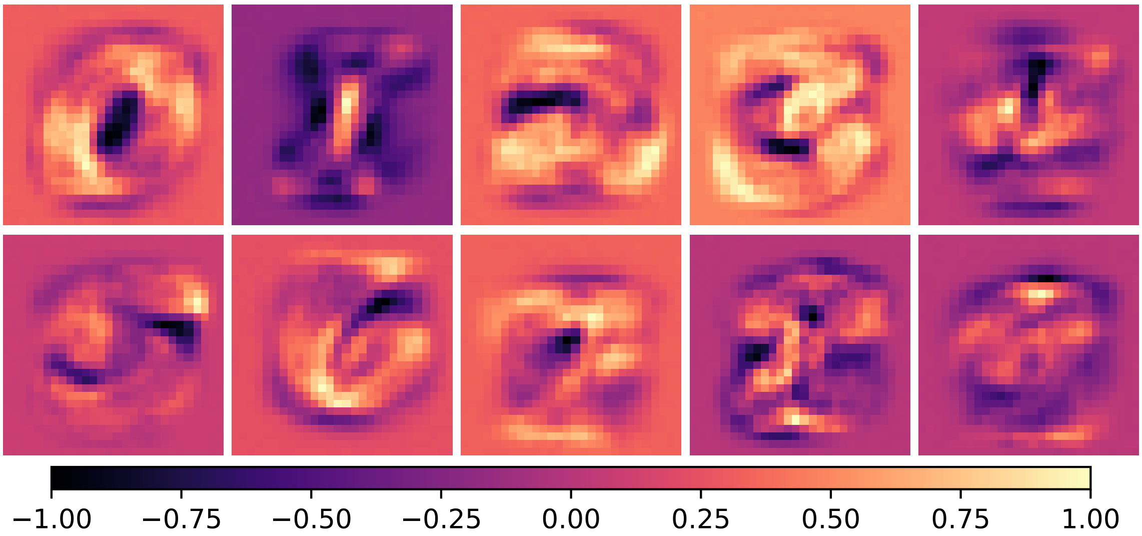


Figure 12: Normalized weights between the input and the output for each of the neurons in the output layer of the network. The weights are taken from the network which is trained on no noisy data introduced in section 4.3.

5. Conclusion

After the results of the neural network and the morphological image operations have been analyzed in the previous sections, a brief summary and conclusion shall be given now.

The morphological image processing has proved to be a practical way to perform interesting image operations, such as the denoising of an image or the edge detection. However, to calculate the dilation/erosion operation one needs to analyze each pixel on its own. This causes - with the current implementation - a runtime which grows quadratic with the image's size. Therefore, the operation can only be applied efficiently on small images. This might be solved by parallelizing the operations. This has not been performed, as it goes beyond the scope of this project. Furthermore, the quality of the denoised images increases with an increasing image size, as the structuring element can be finer, too: The smallest meaningful structuring element has a size of 3×3 pixel; when images with a size of 28×28 pixels are analyzed, these minimal structuring elements might already be too big, so that information about fine structures inside the image can get lost. This might explain, why the application of the morphological operations has not been able to increase the prediction's quality for noisy images.

The best performing network gained an accuracy of 92.22%. Even though this rate is lower than current approaches to this problem, it still shows that even simple neural networks can learn to classify images in only a short time. Due to the low complexity of the network the training has not cost a lot of time.

The application of normal distributed noise in section 4.3 has showed that once the network is trained it is robust to noise in a way, that the accuracy does not strongly decrease for noisy images. While applying noise to the training data, too, has not decreased the accuracy considerable, the attempt to remove this noise with the morphological operations decreased the performance dramatically. This shows that the application of the operations used here, is not sufficient to remove the noise applied. This might change if the signal to noise ratio is increased further. This might be an interesting topic for future research.

A next step for future work on this problem can be to make the network deeper, i.e. add more layers which consist out of neurons, too [8]. These additional layer allow the network to have a more complex behavior, which ultimately results in a better performance. Convolutional neural networks are specialized on the detection, recognition and classification of images due to their structure [12]. Therefore, they can yield much better results than non-convolutional approaches (less than 1% error rate) [12].

One other approach to improve the accuracy of the network can be to replace the loss function with the binary cross entropy [4, 1]. This loss function has a steeper form. This steeper form can increase the training speed and accuracy as the gradient might be greater than for the quadratic loss function used here.

References

[1] Christopher Beckham and Christopher Pal. A simple squared-error reformulation for ordinal classification. *arXiv preprint arXiv:1612.00775*, 2016.

[2] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

[3] Léon Bottou. Stochastic gradient descent tricks. In *Neural networks: Tricks of the trade*, pages 421–436. Springer, 2012.

[4] Andreas Buja, Werner Stuetzle, and Yi Shen. Loss functions for binary class probability estimation and classification: Structure and applications. *Working draft, November*, 3, 2005.

[5] M Bishop Christopher. *Pattern Recognition and Machine Learning*. Springer-Verlag New York, 2016.

[6] Li Deng. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[9] John D Hunter. Matplotlib: A 2d graphics environment. *Computing in science & engineering*, 9(3):90–95, 2007.

[10] Eric Jones, Travis Oliphant, and Pearu Peterson. {SciPy}: open source scientific tools for {Python}. 2014.

[11] Andrej Karpathy. Cs231n convolutional neural networks for visual recognition. `http://cs231n.github.io/convolutional-networks/`, 2017. Accessed: 17.02.2018.

[12] Yann LeCun, LD Jackel, Léon Bottou, Corinna Cortes, John S Denker, Harris Drucker, Isabelle Guyon, UA Muller, Eduard Sackinger, Patrice Simard, et al. Learning algorithms for classification: A comparison on handwritten digit recognition. *Neural networks: the statistical mechanics perspective*, 261:276, 1995.

[13] Sebastian Paeckel and Salvatore R. Manmana. *Hausarbeit zur Vorlesung Fortgeschrittene Algorithmen der numerischen Physik.* 2018.

[14] Sebastian Raschka. *Python machine learning.* Packt Publishing Ltd, 2015.

[15] Frank Y Shih. *Image processing and pattern recognition: fundamentals and techniques.* John Wiley & Sons, 2010.

[16] Stéfan van der Walt, S Chris Colbert, and Gael Varoquaux. The numpy array: a structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22–30, 2011.

[17] Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In *International Conference on Machine Learning*, pages 1058–1066, 2013.

A. Appendix

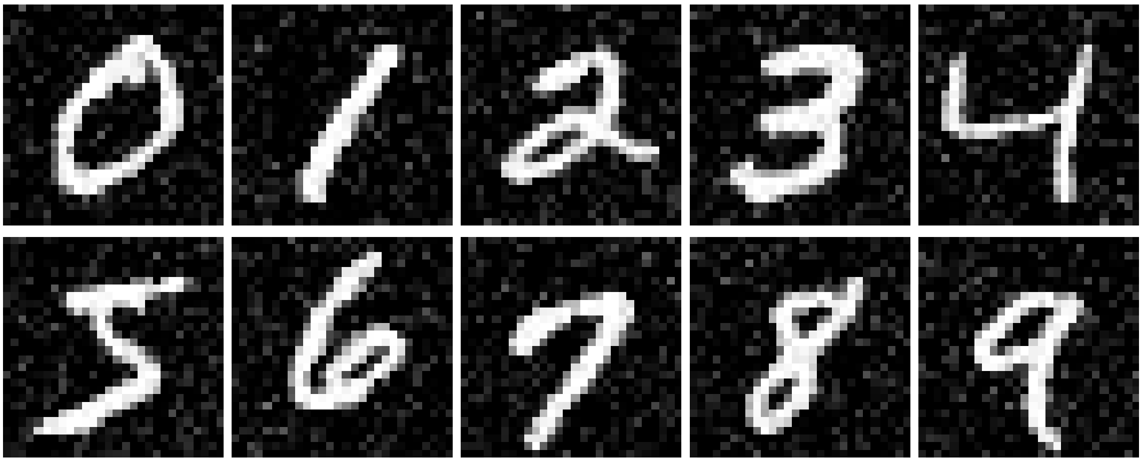


Figure 13: Example images from the MNIST data set for each of the ten different handwritten digits from 0 to 9 on which a noise $\sim \mathcal{N}(0, 0.01)$ is added.

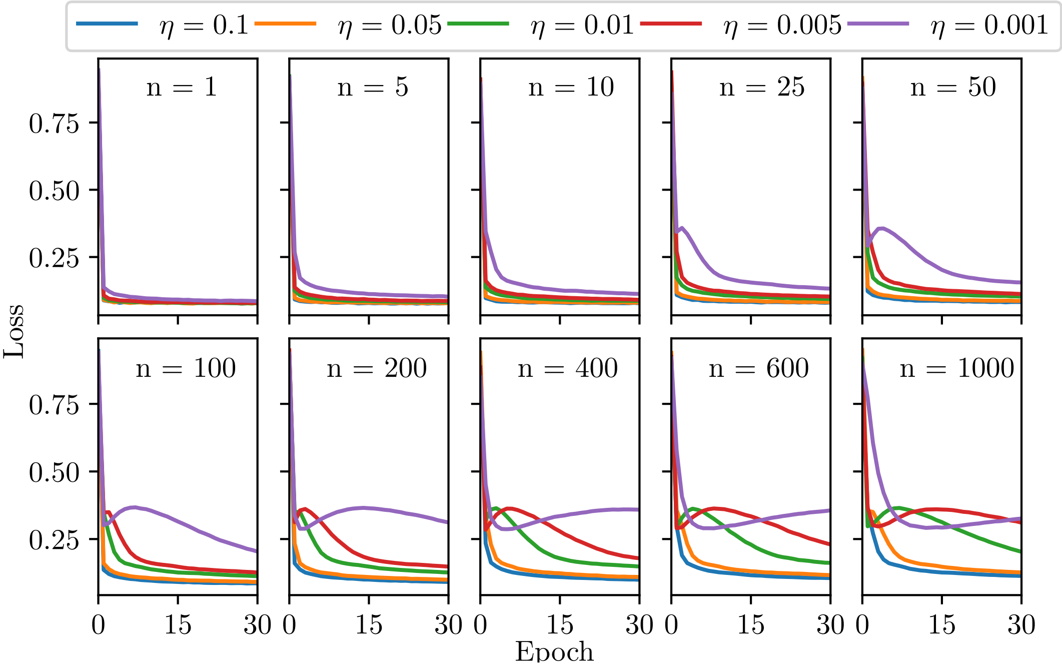


Figure 14: Influence of the learning rate η on the test loss for different batch sizes n. Each box corresponds to a value of n and shows curves for 5 different values of η..

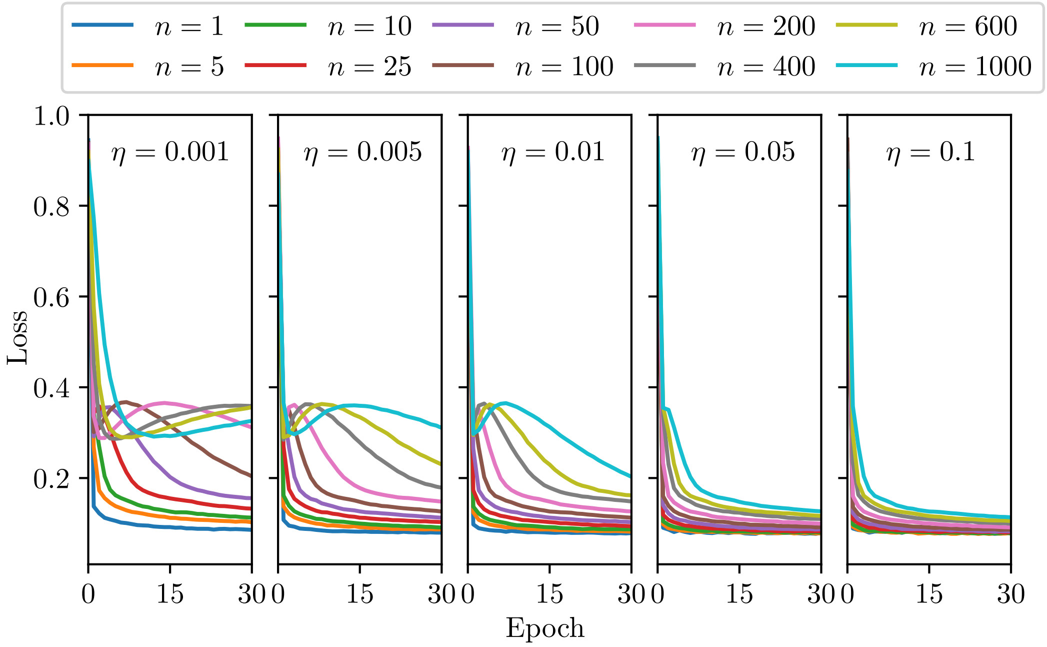


Figure 15: Influence of the batch size n on the test loss for different learning rates η. Each box corresponds to a different value of η and shows curves for 10 different values of n. This is the counterpart to Figure 14.