Analyzing handwritten digits using neural networks, convolutional neural networks and boosting methods.*

Erling Nupen
Ole Kristian Rustebakke
Brage Andreas Trefjord

December 18, 2023

Abstract

This report explores the application of deep learning in image analysis, focusing on the recognition of handwritten digits. Using the widely used MNIST dataset, consisting of 60,000 training images and 10,000 test images, we transition from a feed forward neural network to a convolutional neural network. The convolutional neural network's two-dimensional approach and depth concept enhanced the validation accuracy, greatly outperforming a Neural Network with 76% maximum accuracy, to the convolutional neural network's 94%, with $\eta = 0.1$ and $\lambda = 0.0001$ using the LRELU activation function.

We also delve into boosting algorithms like Adaboost, gradient boosting, and XGBoost, employing weak learners (decision trees with reduced depth) to make a classification model. Lastly, we present a comparison between the neural networks and the boosting algorithms. As expected XGBoost performed the best out of the boosting algorithms with 93% accuracy, with $\eta = 0.1$ and λ in the range from 10^{-5} to 0.1, whilst overall the convolutional neural network had the best accuracy.

Contents

*GitHub Repository: `https://github.com/Bragit123/FYS-STK3155/tree/main/Project_3`

1 Introduction

Image analysis is an important part of deep learning. Creativity is your limit, as you can use it for self-driving cars [1], automatic detection of video events [2], or labeling images into the categories deer, frog, horse, cat etc. [3], as well as plenty more. In our report we have focused on the classic deep learning problem of recognizing handwritten digits.

We have decided to use a popular dataset describing images of various handwritten numbers and letters [4]. The dataset consists of the normalized bitmaps of handwritten digits from a preprinted form, obtained by using MNIST's preprocessing programs. The dataset has contributions amounting to a training set of 60 000 images and a test set of 10 000 images. The images are divided into 28×28 pixels, where each pixel has a value between 0 and 255.

Using this dataset, we will update our previous feed forward neural network (FFNN) [5] by transitioning to a convolutional neural network (CNN). CNN is introduced in Section 2.2. The shift to a CNN is particularly beneficial for image analysis, as it takes into account the two-dimensional structure of the image, and introduces the concept of 'depth' to the neural network. This depth refers to the ability of CNNs to effectively capture hierarchical features in images, reducing the number of parameters and enhancing the network's capacity for intricate image understanding.

We will also explore various boosting algorithms such as Adaboost, gradient boosting, and XGBoost. These various boosting algorithms employ weak learners, to create one good classification model. Boosting is outlined in Section 2.3. Lastly, we will present an analysis of the performance of the different methods in analyzing handwritten digits in Section 3, as well as a conclusion of our findings in Section 4.

2 Theory

We will use our regular neural network, discussed along with different activation and cost functions in [5], on the MNIST dataset, as well as Tensorflow's convolutional neural network [6], AdaBoost [7], gradient boosting [8] and Extreme Gradient Boosting [9] to train and predict the data. The MNIST dataset that Tensorflow provides has a training set of 60 000 images and a test set of 10 000 images, but we will only use 1/10 of this, i.e. 6 000 training images and 1 000 test images, to save some computation time.

2.1 Softmax

For our regular neural network, we need softmax as an activation function for the output layer, since we are doing multiclass classification with 10 categories, $j =$

$0, 1, ..., 9$. The activation for each output neuron j is then the softmax activation function given by

$$P(j|\mathbf{a}) = \frac{\exp(\mathbf{a}^T \mathbf{w}_j)}{\sum_{c=0}^{9} \exp(\mathbf{a}^T \mathbf{w}_c)},$$

i.e. the probability of each output j. Here $\mathbf{a}$ denotes the input from the last hidden layer, and $\mathbf{w}_j$ are the weights of neuron j.

2.2 Convolutional neural networks

Convolutional neural networks (CNNs) are central to deep learning methods. Especially for image recognition/classification, it can be very useful, which is how we intend to use it. They are similar to regular neural networks and consist of weights and biases for the neurons in each layer, that can be trained. They are different in that they take images as input parameters, to make the feed-forward process more efficient for the large amount of input points. Neural networks can be applied by flattening the input of a picture (or a sound clip), which has two dimensions, but CNNs can be applied and keep the images' intrinsic structure since the ordering in each direction matters. Images can also be colored, thus having a red, green, and blue color channel, so the number of weights would be enormous (think of a larger 200x200 image with 3 color channels, giving 120 000 weights for each neuron in the first hidden layer). Note however that we will not consider colored images in this project, so our last dimension is just 1.

CNNs reasonably constrain the structure and utilize the fact that there is usually a strong local correlation between neighboring pixels, as we know it is not often you see a single pixel with a color very different from its neighbors. The layers of a CNN are structured in 3 dimensions; width, height, and depth, it is important to note that depth is not the total amount of layers for the neural network, just the third dimension of the activation. Figure 1 [10] shows the structure of a layer in a CNN. The width and height are just the image dimensions, and the depth corresponds to the 3 color channels. As mentioned, in our report we have looked at black-and-white pictures of handwritten digits, so the depth is just 1. To reduce the amount of weights, the neurons in a layer are only connected to a smaller region of the layer before it, and not fully connected, as is the problem with the regular neural network. At the end of the neural network, we will have an output layer of dimension $1 \times 1 \times 10$, such that the image can be classified into a single digit corresponding to the dominating index in the last layer.

The CNN is structured into a sequence of layers, where each layer takes one volume of activations and transforms it through a differentiable function. First, we have the input layer, which in our case consists of a 28×28 black and white picture, where each element gives the intensity in that position (a higher number corresponds to

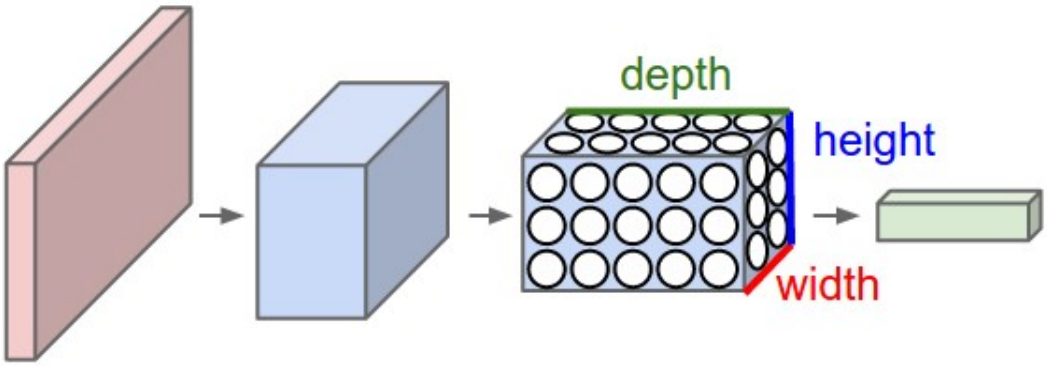


Figure 1: Illustration showing the structure of a layer in a CNN, with three dimensions; height, width, and depth.

where the digit has been written). We have a convolutional (CONV) layer, which will compute the output of neurons corresponding to smaller local regions in the input. This output is given by the product of the local region and the weights, and the amount of filters the transformation goes through gives the shape of the convolutional layer. We will use 10 filters such that the shape would be $28 \times 28 \times 10$. The pooling (POOL) layer then performs a downscaling along the width and height dimensions. We will halve the size of the dimensions such that the new dimensions are $14 \times 14 \times 10$. We then have a layer that flattens the result to one dimension, so that the resulting data can be run through a regular one-dimensional hidden layer. We will vary the activation function of this hidden layer. Lastly, we use a so-called fully connected (FC) layer to compute the classification scores, this layer will have 10 output neurons in our case. Note that this structure corresponds to the output from the flattening layer being run through a regular neural network with one hidden layer. The weights and biases in the fully-connected and convolutional layer are trained with gradient descent, such that the class scores will correspond to the training images, while the RELU and POOL layers implement fixed functions as activation functions and for the downsampling.

2.2.1 Convolution

CNNs are based on the mathematical operation of convolution. Convolution is represented by a mathematical operation such as the integral, on two functions, to see how they affect the shape of each other. It is defined by the expression

$$y(t) = \int x(a)w(t-a)da,$$

where $x(a)$ is the input function, a the input, and $w(t-a)$ is called a weight function or kernel and will affect the input in some way. This function can also be written

with the following notation

$$y(t) = (x * w)(t).$$

This is used in for example Fourier transformations. For a two-dimensional input (such as the width and height of an image) we can use the two-dimensional (discretized) convolution, with a two-dimensional kernel K for our filter,

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n).$$

Since convolution is commutative we can instead write

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

where i and j corresponds to a pixel in the image, and m and n denotes the size of the whole image I. This principle of convolution is as mentioned what can be used in machine learning for image analysis, to transform the image, utilizing it's intrinsic structure.

2.2.2 Pooling

Another important operation in CNNs is pooling. Pooling uses some function to contract subregions into a single output for the next layer. A very normal approach in machine learning is to find the maximum value of the subregion and use this as output, but one can also for example use the average value of the subregion. We will use max pooling, i.e., take the maximum value. The output size, o, of a pooling layer can be given by

$$o = \left\lfloor \frac{i-k}{s} \right\rfloor + 1,$$

where i is the input size, k is the size of the pooling window, and s is the stride, i.e. the distance between two positions in the pooling window, $\lfloor \cdot \rfloor$ is the floor operation. We add 1, to account for any remaining portion if the amount of pooling windows does not add up to cover the whole input (since we use the floor operation).

2.3 Boosting

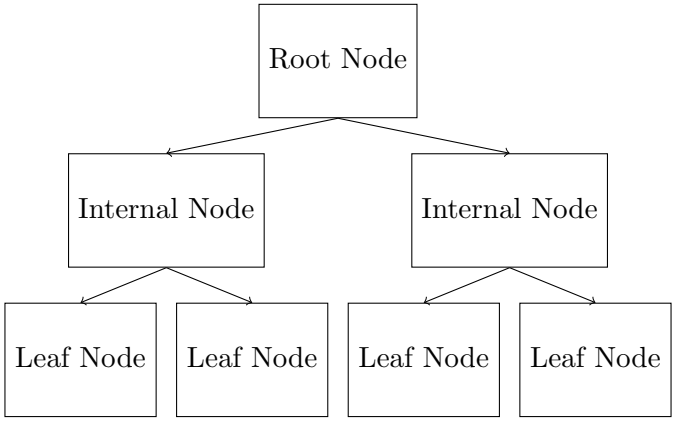
Boosting is an ensemble learning technique in machine learning that combines the predictions of multiple weak classifiers to create a strong classifier. A weak classifier is a model that performs slightly better than random chance. An example of a weak classifier could be a simple linear function such as $b(x;\gamma) = 1 + \gamma x$, however, with the boosting methods here, we will explore the use of decision stumps, and decision trees.

The Boosting algorithms will iteratively train these weak learners, and weigh heavier the learners who made the correct classification for each iteration. This allows the

algorithm to focus on the difficult-to-classify instances and improve overall predictive performance.

Decision trees are supervised learning algorithms used for both classification and regression tasks. It models decisions as a tree-like structure with a root node representing the initial decision based on a specific feature. From the root node comes internal nodes representing a decision based on the value of that particular feature, branches representing the outcome of the decision, and lastly, leaf nodes representing the final decision or the output.

The leaf nodes contain the predictions we will make for new query instances presented to our trained model. This is possible since the model has learned the underlying structure of the training data and hence can, given some assumptions, make predictions about the target feature value (class) of unseen query instances.



Adaboost

Adaboost uses decision stumps as a learner. Stump is a reference to the fact that this is just a one-level tree, i.e. a tree with a depth of 1. Adaboost uses several tree stumps, as depicted in 2.3, however, without the internal nodes. The basic steps of the AdaBoost algorithm, are to first initialize the weights and then rewrite the misclassification error as outlined in [11]. The misclassification error is then defined as Eq (1)

$$\overline{\text{err}}_m = \frac{\sum_{i=0}^{n-1} w_i^m I\left(y_i \neq G\left(x_i\right)\right)}{\sum_{i=0}^{n-1} w_i}. \tag{1}$$

The function I is one for misclassification and zero for correct classification. G is one weak classifier. Eq (1) essentially amounts to how often our weak learners can correctly classify an outcome. All attempts at classifying are looped over, and the

learner is fitted to the training set. After this process, a new quantity α given in Eq (2), is calculated

$$\alpha_m = \log\left(1 - \overline{\text{err}}_m\right) / \overline{\text{err}}_m. \tag{2}$$

α will then be used to update the weights, as given in Eq (3)

$$w_i = w_i \times \exp\left(\alpha_m I\left(y_i \neq G\left(x_i\right)\right)\right. . \tag{3}$$

Adaboost aims to put more emphasis on wrongly classified observations, weighing wrong classification heavier, forcing each classification step to concentrate on those observations that were missed in the previous iterations.

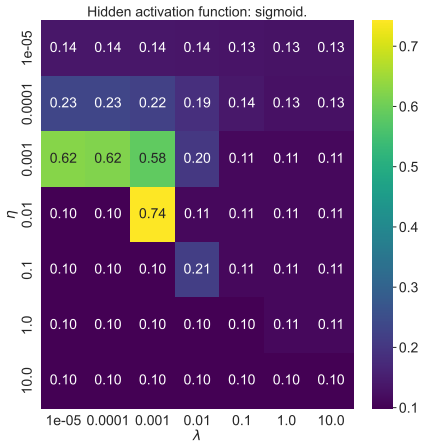
Gradient Boosting

In contrast to AdaBoost, gradient boosting uses a user-defined cost function to fit weak learners to the negative gradient of the loss function. The weak learner can be a wide variety of models, but usually, they are simple decision trees with limited depth to prevent overfitting. Instead of updating the weights using a quantity α, the weights are fitted to the residuals or errors of the previous weak learner's predictions. This iterative process allows gradient boosting to sequentially refine the model by focusing on minimizing the errors made by the ensemble.

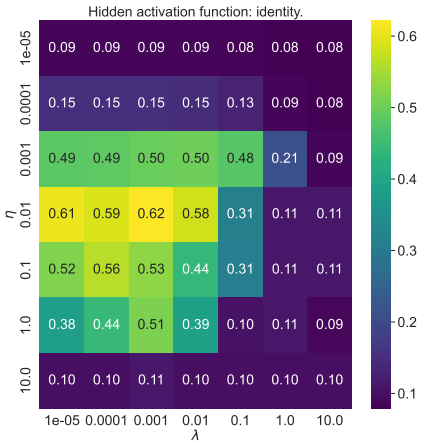
Extreme Gradient Boosting

Extreme Gradient Boosting (XGBoost) is a specific and highly efficient implementation of the gradient boosting algorithm. Renowned for its computational speed and model performance[12], XGBoost goes beyond traditional gradient boosting methods by incorporating both L1 and L2 regularization parameters. This dual regularization strategy enhances XGBoost's ability to prevent overfitting.

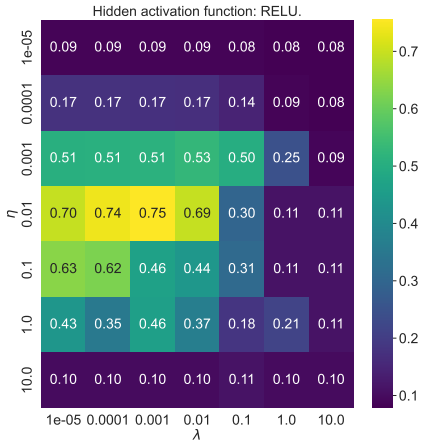
In addition to regularization, XGBoost introduces a range of advanced features and optimizations that contribute to its exceptional performance. For instance, it employs a 'max depth' constraint on decision trees, preventing them from growing too deep and mitigating the risk of overfitting the training data. Furthermore, XGBoost uses a process called 'pruning,' which selectively removes branches of the tree that do not contribute significantly to the overall model improvement[12]



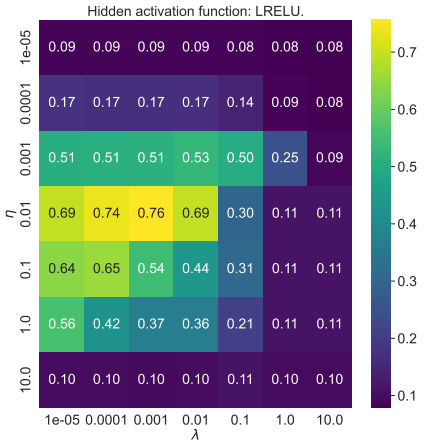
(a) Accuracies with the sigmoid activation function.



(b) Accuracies with the identity activation function.



(c) Accuracies with the RELU activation function.



(d) Accuracies with the leaky RELU activation function.

Figure 2: Accuracy of the regular neural network on the validation images, plotted against the regularization parameter λ and the learning rate η. All the trained networks consisted of one hidden layer with 50 nodes, and the different subfigures correspond to the different activation functions that were used for the hidden layer.

3 Results and discussion

3.1 Regular neural network

We trained a regular neural network on the MNIST dataset by flattening the two-dimensional image into a one-dimensional array. For this neural network, we had one hidden layer with 50 nodes and ran 50 epochs of training with a batch size of 400. In Figure 2 you can see the validation accuracies of our model, using different hidden activation functions, regularization parameters, and learning rates. We use Adam as scheduler, with hyperparameters $\rho_1 = 0.9$, $\rho_2 = 0.999$ and momentum 0.01.

In Figure 2a you can see the validation accuracies when we used the sigmoid activation function. The best value with this activation function appears at $\lambda = 0.001$ and $\eta = 0.01$, where the accuracy is 0.74. Small adjustments in λ or η give very different results, with accuracies as low as 0.10, which is the accuracy we would expect if we chose a number from 0 to 9 at random. It appears obvious that the sigmoid activation function does not give very reliable results.

In Figure 2b you can see the validation accuracies when we used the identity activation function. This activation function appears to give more consistent results, with accuracies that do not vary as much as with the sigmoid activation, but the results are, however, a lot worse. The best accuracy is 0.62, which is obtained at $\lambda = 0.001$ and $\eta = 0.01$, the same as the peak accuracy for the sigmoid function. 0.62 is however very low, so the model is not reliable in recognizing the handwritten digits.

The validation accuracies from the RELU activation function can be seen in Figure 2c. The RELU activation function has a peak accuracy of 0.75, which is close to the peak accuracy of the sigmoid function, however, there is more consistency in small variations of λ and η. The peak accuracy is obtained at the same values as sigmoid and identity, at $\lambda = 0.001$ and $\eta = 0.01$. Although small variations in λ and η give less variations in the accuracy in the case of RELU than sigmoid, the accuracies are still quite low. Slightly varying the regularization parameter λ gives accuracies of 0.69 and 0.74, while varying η gives accuracies of 0.46 and 0.51, which are only about 50% accurate.

Finally, the validation accuracies obtained from using the leaky RELU activation function are given in Figure 2d. Here the peak accuracy is 0.76, which is the highest of the four activation functions. This accuracy is again obtained at $\lambda = 0.001$ and $\eta = 0.01$, the same as for the other three. Small variations in λ or η give approximately the same results as in the RELU case.

From the plots in Figure 2 it appears that the regular neural network does not give very accurate models for recognizing handwritten digits. The best accuracy obtained

is 0.76, which means that around one in every four images will be mislabeled by our model. This is not very impressive.

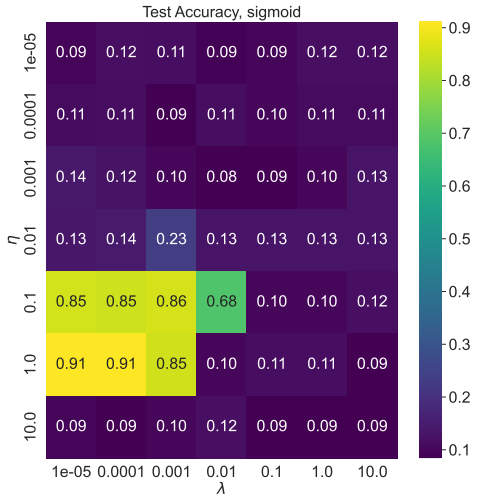
3.2 Convolutional neural network

For our simulation of the CNN on the MNIST data set [4] we used Tensorflows inbuilt functions. The convolutional layer we use looks at regions of size 3×3, and uses 10 filters, such that the input of dimensions $28 \times 28 \times 1$ turns into $28 \times 28 \times 10$, before we halve the height and width using maxpooling, such that the dimension is $14 \times 14 \times 10$ when we flatten the input. Thereafter it goes into a layer of 50 hidden neurons using RELU, LRELU, sigmoid, and identity as activation function (the same activation function is used in the convolutional layer) before it goes through softmax as an activation function to the output layer and produces 10 output neurons, where the index with maximum value corresponds to the digit it predicts. For the training part we used 50 epochs and batch size of 400 on the 6000 images, i.e. 15 batches. We use L2 regularization for the convolutional and the fully-connected layers. Figure 3 shows the test accuracy for the different activation functions.

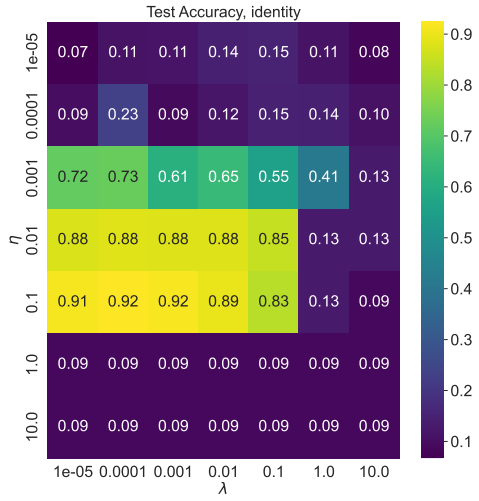
We can see that LRELU performs the best with a maximum accuracy of 0.94 for learning rate $\eta = 0.1$ and regularization $\lambda = 0.0001$. RELU, which is an often used activation function for CNNs, has a maximum accuracy 0.93 for $\eta = 0.1$ and $\lambda = 10^{-5}$, $\lambda = 0.0001$ or $\lambda = 0.001$. 93% and 94% accuracy can be considered as quite good, also considering that we only use 1/10 of the available data. Since the training accuracy for RELU is 98% (not shown here, but we made the plots for the training accuracies as well), it is clear that a larger training set or more batches would give an even better test accuracy. Others have gotten up to 97% using CNN [13]. Sigmoid and identity performs slightly worse with a maximum accuracy of respectively 0.91 for $\eta = 1.0$ and $\lambda = 10^{-5}$ or $\lambda = 0.0001$, and 0.92 for $\eta = 0.1$ and $\lambda = 0.0001$ or $\lambda = 0.001$. Sigmoid also has a smaller region of learning rates and regularization parameters which gives good results. In general, the CNN performs way better than the regular neural network, as we would expect, since it takes the structure of the images into account, and uses its properties to effectively downscale it, and train the data more efficiently.

3.3 AdaBoost

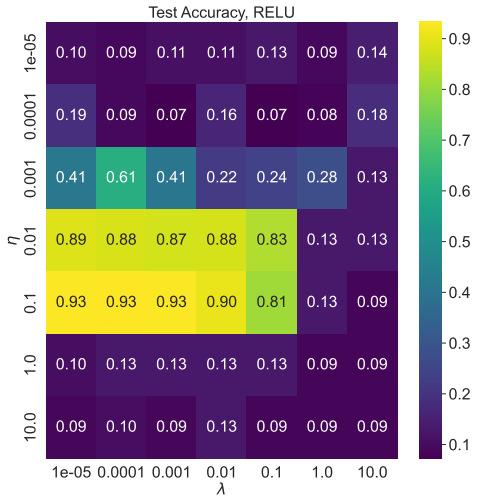
A barplot showing the accuracy of the AdaBoost algorithm for various learning rates is seen in Figure 4. Boosting is terminated if the number of estimators exceed 50.



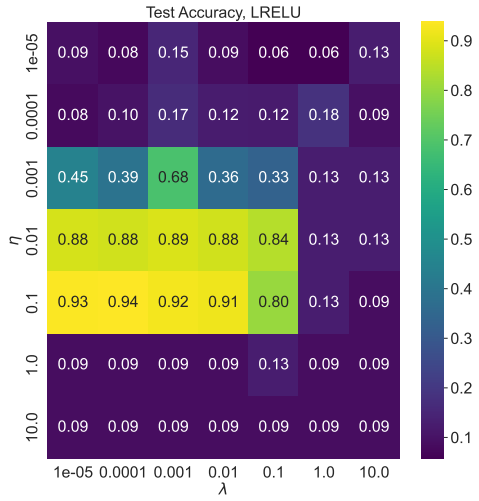
(a) Using sigmoid as activation function.



(b) Using identity as activation function.



(c) Using RELU as activation function.



(d) Using LRELU as activation function.

Figure 3: Testing accuracy for the convolutional neural network by Tensorflow Keras on the MNIST handwritten digits dataset. We used L2 regularization, and stochastic gradient descent with 50 epochs and batch size 400. Learning rate η and regularization λ is what we plot for.

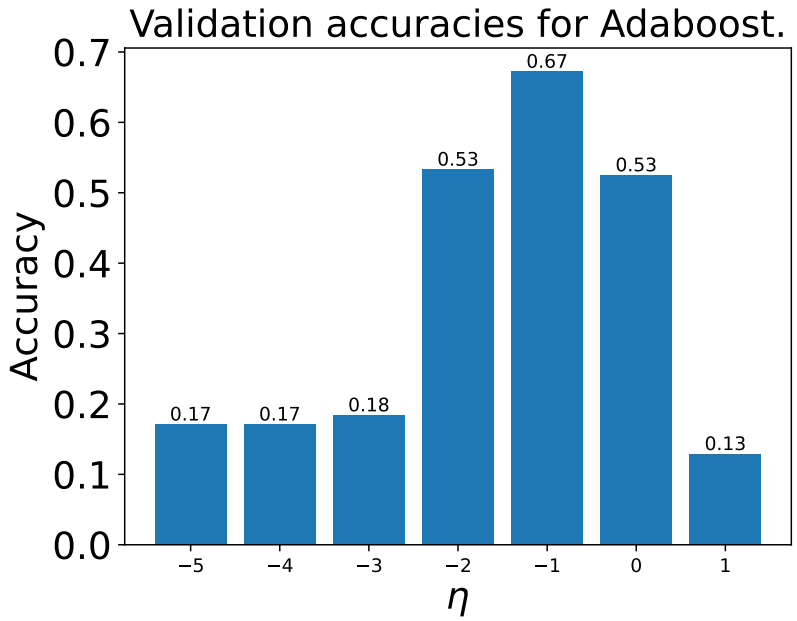


Figure 4: Barplot showing how accurate the AdaBoost algorithm was in predicting the right outcome for various learning rates.

The best accuracy for the AdaBoost algorithm was 67% for $\eta = -1$. This is considerably worse than the other boosting algorithms and is likely due to the highly complex nature of the input data, which decision tree stumps would struggle to capture.

3.4 Gradient Boosting

A barplot showing the accuracy of the gradient boosting algorithm for various learning rates is seen in Figure 5. 100 boosting stages are performed, and a maximum depth (limit for number of nodes in the tree) is 3.

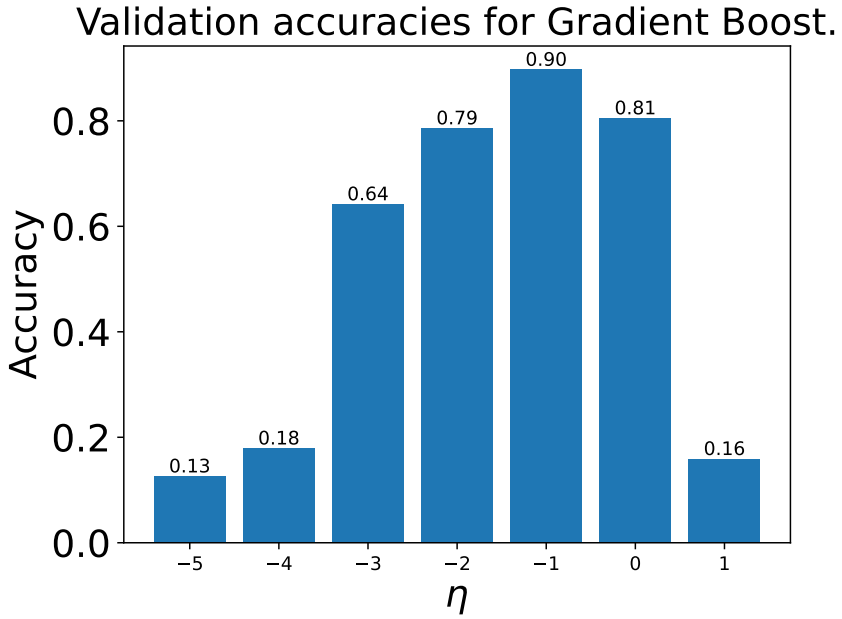


Figure 5: Barplot showing how accurate the gradient boosting algorithm was in predicting the right outcome for various learning rates.

As we can see from Figure 5 for $\eta = -1$ the accuracy is far better than that of AdaBoost, at 90%. The gradient boosting classifier has a deeper decision tree, making it easier to capture the more complex relationship in the training data. It also is more consistent in updating its weights, since it uses the residuals of the previous model, to fit the new learners in a way that minimizes the cost-loss function. The optimal cost-loss function for multinomial deviance is the logarithmic loss function.

3.5 XGBoosting

A barplot showing the accuracy of the XGboosting algorithm for various learning rates, and different regularization parameters is seen in Figure 6. Maximum depth is 6 and we use a logistic loss function.

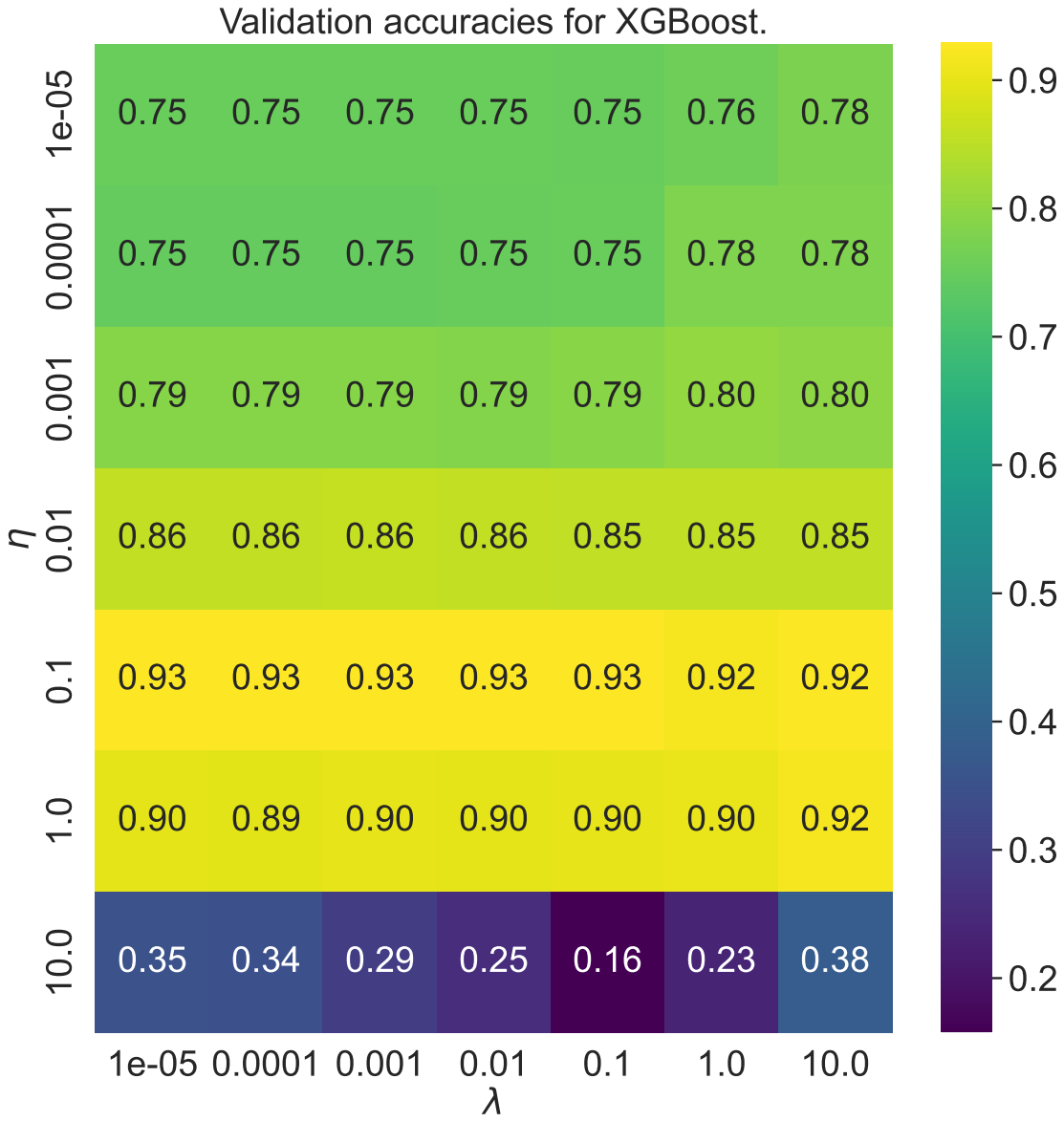


Figure 6: Heatmap-plot showing how accurate the XGBoosting algorithm was in predicting the right outcome, for various learning rates and varying L2 regularization parameters.

The peak accuracy for XGBoost was 93% for a $\eta = 0.1$, and a regularization parameter in the range $\lambda = 0.00001 - 0.1$. XGBoost, as expected, is the best of the boosting algorithms, and even rivals the accuracies of the CNN. More than 9 out of 10 times the XGBoosting algorithms were able to correctly classify an image. This high accuracy is likely due to the regularization parameter preventing overfitting.

4 Conclusion

In this project, we have looked at different machine learning methods for classifying handwritten digits in the MNIST dataset. We have considered a regular neural network, a convolutional neural network, and the boosting methods Adaboost, gradient boost, and XGBoost with decision trees as the weak learner.

The regular neural network did not perform very impressively. At best, it gave an accuracy of 0.76 with the LRELU activation function, learning rate $\eta = 0.01$, and the regularization parameter $\lambda = 0.001$. This means that even our best model with the regular neural network would on average mislabel one in every four images.

CNN performed way better than the neural network, and the AdaBoost boosting algorithm, as it is made especially for the task of image analysis. It had a maximum accuracy of 0.94 for LRELU as an activation function in the fully connected layer and convolutional layer and for learning rate $\eta = 0.1$ and regularization $\lambda = 0.0001$. However, gradient boosting, and XGBoosting algorithms both had accuracies in the range of the CNN, only 4% and 1% lower accuracy respectively.

References

[1] M. Bansal, *Image processing in autonomous vehicles: Seeing the road ahead*, Medium, 2023. [Online]. Available: `https://medium.com/@mohanjeetbansal777/image-processing-in-autonomous-vehicles-seeing-the-road-ahead-b400d176f877` (visited on 12/14/2023).

[2] O. Nergård Rongved, M. Stige, S. Hicks, *et al.*, "Automated event detection and classification in soccer: The potential of using multiple modalities," *Mach. Learn. Knowl. Extr.*, pp. 1030–1054, 2021. DOI: `https://doi.org/10.3390/make3040051`.

[3] A. Dosovitskiy, L. Beyer, A. Kolesnikov, *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *ICLR*, 2021.

[4] Y. LeCun, C. Cortes, and C. Burges, "Mnist handwritten digit database," *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, vol. 2, 2010, The dataset is obtained using tensorflow, see https://www.tensorflow.org/datasets/catalog/mnist.

[5] Nupen, Erling and Rustebakke, Ole Kristian and Trefjord, Brage Andreas, *Comparing gradient descent methods, and using neural networks and logistic regression to analyze the Wisconsin Breast Cancer dataset.* University of Oslo, 2023.

[6] *Convolutional neural network (cnn)*, TensorFlow Core, 2023. [Online]. Available: `https://www.tensorflow.org/tutorials/images/cnn` (visited on 12/14/2023).

[7] *Sklearn.ensemble.adaboostclassifier*, scikit-learn, 2023. [Online]. Available: `https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html` (visited on 12/14/2023).

[8] *Sklearn.ensemble.gradientboostingclassifier*, scikit-learn, 2023. [Online]. Available: `https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html` (visited on 12/14/2023).

[9] *Xgboost documentation*, dmlc XGBoost, 2022. [Online]. Available: `https://xgboost.readthedocs.io/en/stable/index.html` (visited on 12/14/2023).

[10] M. Hjorth-Jensen, *Week 44, convolutional neural networks (cnn)*, University of Oslo, 2023. [Online]. Available: `https://compphysics.github.io/MachineLearning/doc/pub/week44/html/week44.html` (visited on 12/13/2023).

[11] M. Hjorth-Jensen, University of Oslo, 2023. [Online]. Available: `https://compphysics.github.io/MachineLearning/doc/pub/week47/html/week47.html` (visited on 12/13/2023).

[12] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '16, ACM, Aug. 2016. DOI: `10.1145/2939672.2939785`. [Online]. Available: `http://dx.doi.org/10.1145/2939672.2939785`.

[13] V Viswanatha, A. C. Ramachandra, S. D. Nalluri, S. M. Thota, and A. Thota, "Handwritten digit recognition using cnn," *International Journal of Innovative Research in Computer and Communication Engineering*, vol. 11, 2023. DOI: `10.15680/IJIRCCE.2023.11010012`.