Project 3 FYS-STK4155

Trond Skaret Johansen, Eirik Salberg Pedersen, Jonatan Winsvold

December 18, 2023

Abstract

In this project we explore 3 methods for image classification. The goal was to expand on the methods introduced in our second project [1] to a more advanced classification problem. The methods in this project include multinomial logistic regression also known as softmax regression, neural networks and convolutional neural networks. We find that while simpler models like softmax regression can provide baseline insights, the intricate nature of image classification necessitates more advanced approaches like neural networks and, more specifically, CNNs. These results underscore the importance of selecting appropriate models based on the complexity of the task at hand and pave the way for future research in optimizing these models for even more challenging classification problems. In the end we achieved an accuracy of 92.3% using the simple multinomial regression model, a 98.2% with the normal neural network model and a 99.0% using a convolutional neural network.

1 Introduction

This project focuses on the Modified National Institute of Standards and Technology (MNIST) dataset, a rich and complex dataset consisting of handwritten and labeled letters and digits. Building on insights from Project 2, where simple regression models performed comparably well to neural networks on interpolating the Franke function and predicting malign tumors, we wanted to find a challenge suited to distinguish the capabilities of neural networks from those of simple regression.

We explore three key neural network models: a basic Softmax regression classification, a plain neural network, and a convolutional neural network (CNN). These models were chosen for their potential to excel in image classification tasks. The project will examine the implementation of softmax in neural networks, delve into the architecture and functionality of plain neural networks, and explore the advanced capabilities of CNNs, particularly focusing on convolutional layers, pooling techniques, and their impact on image recognition.

Our methodology involves preprocessing and scaling the MNIST dataset, followed by a detailed examination of the logistic probability cost function. We

then implement and optimize each model using appropriate descent methods and initialization strategies. The evaluation includes a grid search for optimal parameters and an analysis of each model's performance through confusion matrices and identification of incorrect predictions.

In conclusion, we will provide a comprehensive comparison of the models, reflecting on their strengths, limitations, and the implications of our computational constraints on exploring more complex architectures. This project aims to deepen our understanding of neural networks in advanced classification tasks and guide future research directions in this field.

2 Methods

2.1 Data handling

We use the MNIST letter data set [2] imported through tensorflow. It consists of 70,000 grayscale images of handwritten digits from 0 to 9, each 28x28 pixels in size. Given with the split of a training set of 60,000 images and a test set of 10,000.

Figure 1: Some letters from the MNIST dataset.

Scaling and Preprocessing

To prepare the MNIST dataset for neural network training it had to undergo a scaling process consisting of a few key steps:

We begin by loading the data using TensorFlow and Scikit-learn. For our non-convolutional neural network models, we flatten the images. Flattening transforms each 28x28 image into a 784-value array. This step simplifies the data structure, making it compatible with the input layer of our neural network models.

The labels in the dataset are converted into a one-hot encoded format. This process changes categorical integer labels into binary vectors for multi-class classification tasks. Utilizing our custom train-test-split function from project 2 [1], we divide the dataset into training and testing sets for evaluating our models' performance on unseen data. We control the randomness of this split using a fixed seed, ensuring consistency across experiments.

The images are scaled to consist of floating point color values in the range $[0, 1]$, i.e. min-max scaling. This normalization aids in speeding up the learning process and ultimately achieving better performance.

2.2 Cross-entropy loss

In our neural network models, the loss is computed using the cross-entropy method, which is particularly effective for classification tasks involving multiple classes. This method is mathematically represented as follows:

$$L(\hat{y}, y) = -\frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{C} y_j^{(i)} \log\left(\hat{y}_j^{(i)}\right) \tag{2.1}$$

Here, $L(\hat{y}, y)$ denotes the cross-entropy loss function. The predicted probabilities matrix is represented by $\hat{y}$, where $\hat{y}_j^{(i)}$ is the predicted probability that the i-th sample belongs to class j. The true labels are given by y, where $y_j^{(i)}$ is a binary indicator (0 or 1) if class label j is the correct classification for the i-th sample. N is the total number of samples, and C is the number of distinct classes. The double summation first sums over all the classes for each individual sample and then accumulates these values across the entire dataset of samples.

For the logarithm operation we also clip the values of y_{pred} to the range [0.0000001, 0.9999999] to ensure numerical stability, it is however omitted from equation 2.2 for the sake of readability.

2.3 Softmax function

The softmax function is implemented for the neural network setup to give the model the ability to perform multi-class classification. It effectively transforms the raw outputs of the network into a meaningful probability distribution, guiding the model to learn appropriate weights during training. The use of Xavier initialization [3] supports robust and effective training of the softmax-based neural network model. The function is given as

$$\text{softmax}(x^{(i)}) = \frac{e^{x^{(i)}}}{\sum_{j=1}^{C} e^{x^{(j)}}} \tag{2.2}$$

In this equation, $x^{(i)}$ denotes the raw output for the i^{th} class. The exponential transformation of this output, ensures non-negative values. The denominator, $\sum_{j=1}^{C} e^{x^{(j)}}$, is the sum of exponential transformations for all C

classes, providing normalization. The softmax output, $\text{softmax}\left(x^{(\text{i})}\right)$, thus reflects the probability that the input belongs to the i^{th} class, balancing across the total number of classes C.

2.4 Multinomial Regression

As a baseline comparison to the neural networks, we explore the efficacy of Multinomial Regression (MR) on the MNIST dataset. This approach mirrors a neural network with no hidden layers, where inputs are directly subjected to a softmax transformation 2.3. Such a configuration simplifies the network architecture, focusing solely on the classification capability of the softmax function in a multi-class setting and is intended to give us some insight of to which extent the additional hidden layers in our deeper neural networks contribute to improved performance.

Our experimental framework systematically varied key hyperparameters to assess their impact on MR performance. Batch sizes were selected to cover a broad spectrum, ranging from 32 to 8192, offering insights into the model's behavior under different computational loads. Learning rates were tested from 0.1 down to 0.001, providing a gradient of step sizes for the optimization process. Regularization, an essential factor in preventing overfitting, was varied from 0, indicating no regularization, up to 0.000001.

The training regimen for each parameter combination extended to a maximum of 50 epochs. We recorded performance metrics at predetermined epochs (2, 5, 8, 10, 15, 20, 35, 50) to monitor the model's evolution and learning trajectory. The primary objective was to discern the influence of these hyperparameters on the model's accuracy and its ability to generalize when classifying handwritten digits.

2.5 Neural networks

As a reference point we repurposed the neural network class from project 2 [1] which was designed to be modular to handle a varying number of hidden layers and any target type.

The network supports custom activation functions for both hidden layers and the output layer. The network's weights and biases are initialized using the same Xavier Initialization technique as for our softmax regression [3]. By

default, the sigmoid function is used for hidden layers, and a softmax function is used for the output layer. This setup allows for flexibility in tailoring the network's behavior based on the specific requirements of the task.

As in project 2, the predictions are made through forward propagation. For the description of forward propagation, assume that we have computed $h_1^{l-1}, \ldots, h_n^{l-1}$ for the n nodes in layer $l-1$. The value of node j in the next layer is then computed as:

$$h_j^l = f\left(\sum_{i=1}^{n} w_{i,j}^l h_i^{l-1} + b_i^l\right). \tag{2.3}$$

This process of signals "propagating" through the network motivates the name. [1]

We considered three different configurations for the number of hidden layers: 1, 2, and 3. For each hidden layer configuration, we tested a range of nodes per layer, specifically 512, 256, 128, 64, 32, 16, and 8. This allowed us to assess the impact of both depth (number of layers) and width (number of nodes) on the network's performance.

Each neural network was trained for 10 epochs, with a learning rate of 0.01 and a batch size of 128. Regularization was not applied in these runs ($\lambda = 0$). The activation function for the hidden layers was set to the sigmoid function, a common choice for neural networks due to its non-linear properties and its ability to handle binary classifications effectively.

Due to technical restrictions of loops in the JAX library we had to deviate from best practices and create a workaround for dynamically initializing our network models with a varying structure of layers. The network architecture allows for the creation of neural network models with 0 to 8 hidden layers. Its initialization returns a lambda function that represents the neural network model corresponding to the specified number of hidden layers.

The training of the neural network is conducted using Stochastic Gradient Descent with Adam optimization. This method iteratively updates the network's parameters (weights and biases) to minimize a loss function. The gradient of the loss function with respect to the network's parameters is computed using grad from JAX.

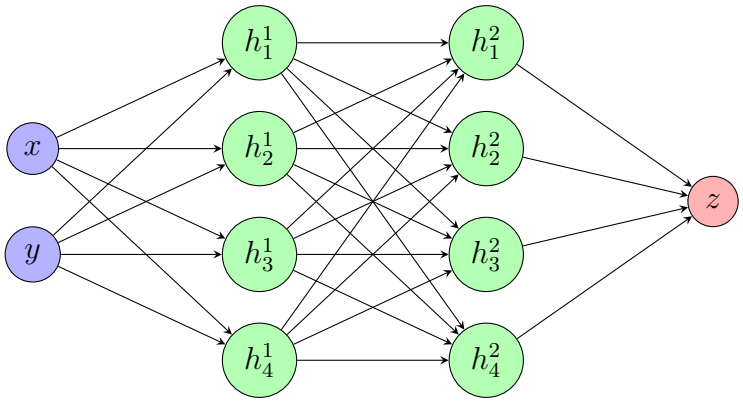

Figure 2: A representation of a neural network architecture with two hidden layers.

For training and evaluation, the cross-entropy loss function is used. This loss function is appropriate for regression tasks and is a standard choice for measuring the performance of neural networks in such scenarios.

2.6 Convolutional neural networks

We implemented a Convolutional Neural Network (CNN) using the JAX library, which provides efficient numerical computations. The network comprises convolutional layers followed by fully connected layers, and employs softmax for classification.

For our model we use filters, also called kernels, which are utilized to extract features from the input images, such as edges, textures and complex patterns. Each filter is a small matrix that convolves across the image. This involves element-wise multiplication of the filter with image regions, followed by summing the results, to produce feature maps. Multiple filters are employed, each detecting different features. The network learns these filter values adaptively, enabling the hierarchical extraction of features, from simple to complex.

Pooling layers are introduced to downscale the spatial dimensions (width and height) of the feature maps, thus reducing the computational burden and the model's parameter count. More specifically we use max pooling which outputs the maximum value from each non-overlapping subregion of

the feature map. While pooling layers reduce the spatial dimensions, they retain critical information in the feature maps and do not alter the depth (number of feature maps).

The combination of filters and pooling layers allows CNNs to effectively learn and represent spatial hierarchies in visual data. Filters capture and learn diverse features, while pooling layers condense the feature maps, ensuring computational efficiency and focusing on the most prominent features in the image.

The CNN consisted of several convolutional layers, each followed by a ReLU activation function and a max pooling operation. The number of convolutional layers and their specific parameters, like the number and size of filters, were varied across different runs to assess their effect on model performance. The filter sizes in the convolutional layers were experimented with, alongside the number of filters per layer, to understand their influence on learning spatial features.

Following the convolutional layers, the output was flattened and fed into a series of dense (fully connected) layers. The number of nodes in these dense layers was also varied. The use of multiple dense layers allowed us to investigate how the depth of the network influences the model's ability to learn complex patterns from the reduced feature map provided by the convolutional layers.

The final layer of the network was a softmax layer, ensuring the output probabilities sum to one, making the model suitable for multi-class classification. The network was trained using the Adam optimizer as it showed to be the best performing optimzer in project 2 [1].

Throughout the training process, we employed cross-entropy loss as the primary metric for optimization, alongside regularization to prevent overfitting. The model was evaluated based on its accuracy in classifying the MNIST digits, with results from various configurations compared to identify the most effective architecture.

The experiments involved systematically varying the convolutional window sizes and the number of filters, as well as adjusting the number of nodes in the dense layers. This approach provided a comprehensive analysis of how different CNN architectures perform on the task of handwritten digit recognition, offering valuable insights into the strengths and limitations of

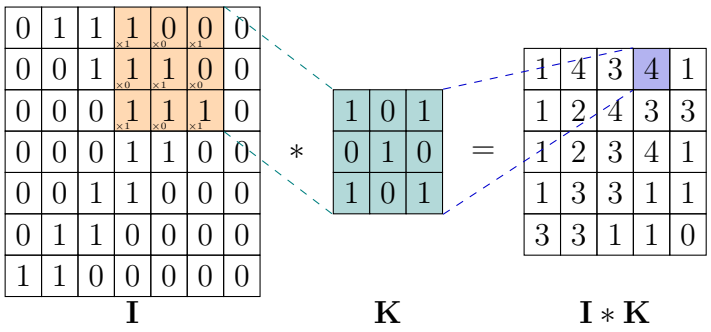

Figure 3: An illustration of the convolution operator. Figure from https://tikz.net/conv2d/.

various configurations.

3 Results and analysis

3.1 Softmax regression

One of the goals of the project has been to present results in a more compact manner. With this in mind, we decided to use parallel coordinate plots to showcase our training runs while varying several hyperparameters. We immediately notice that the two highest learning rates have the worst runs. One can also note that the strongest regularisation negates the issues from the large learning rate. Still, for the best model we opt for a smaller learning rate with little regularisation. Smaller batch sizes also seem to correlate well with a good accuracy score. While there is no worsening from increasing the number of epochs from 20 to 50, there is not much to be gained. We may therefore strike a balance and use fewer epochs. The best run we god using softmax regression gave a test accuracy score of 92.9%.

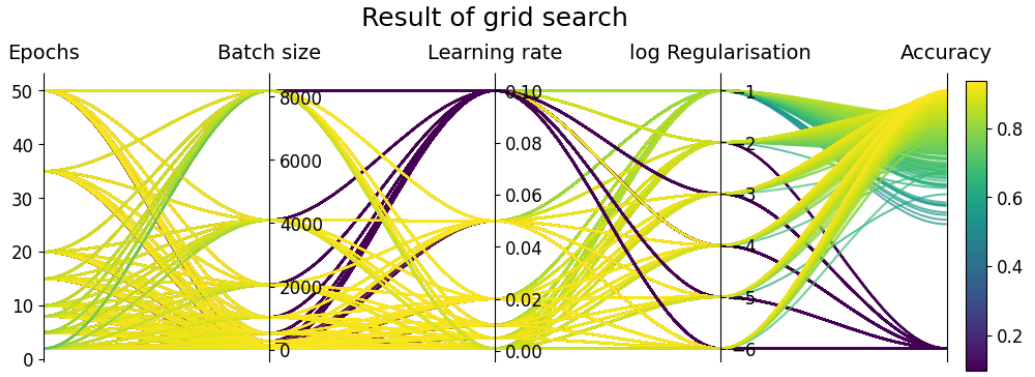

Figure 4: Parallell coordinates plot showing the dependence of accuracy on each hyperparameter.

Having seen which parameters give good models, we are ready to evaluate one of the better ones. The results are given as a confusion matrix in the appendix, as we would rather look at some examples to illustrate the performance. This is shown in Figure 5. The model failing on the letters 5, 4 and 9 seem reasonable, while we expect a good model to recognise the numbers 3 and 6 as they are written here. This illustrate a common controversy surrounding the use of AI, namely that it makes mistakes a human would never do. It would have been an interesting to see how a human would perform in

recognizing the letters in the dataset to see if classifying the entire dataset is a feasible task. However given that the model struggles even on some images that are clearly legible tells us that there is room for improvement in our models and that a more complex model may be needed to aquire the performance we desire.

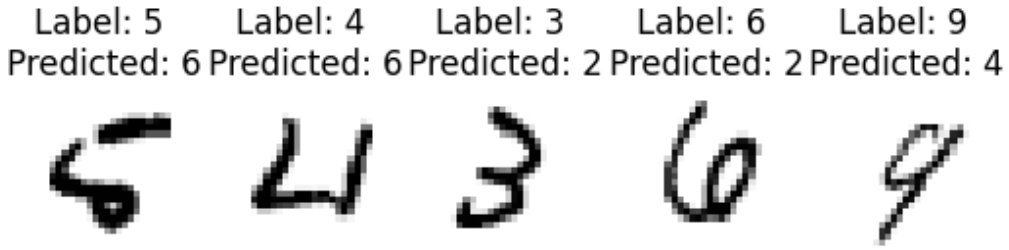

Figure 5: Images that the model failed to correctly predict.

3.2 Neural network

As we saw that when using a softmax model we could only achieve around 93% accuracy on our test, we will now explore a more powerful set of neural network models containing several hidden layers. As we know that the performance of neural network models are highly dependent on the number of hidden layers, their size and the activation functions used in them we will explore how these three variables impact the ability of our model and based on that make a decision on what kind of architecture suits the task of predicting digits the best.

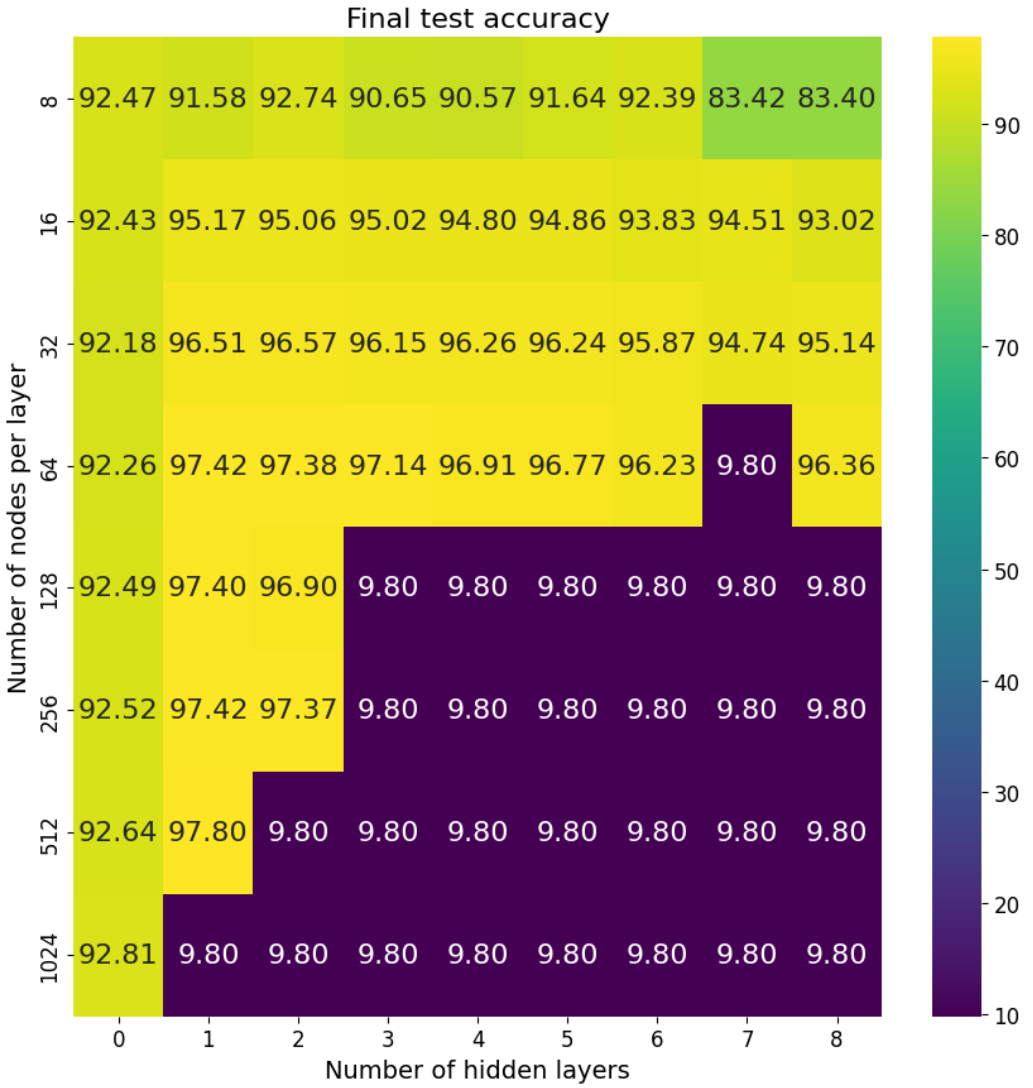

Figure 6: Experiment with training neural networks with varying amount of layers and nodes per layer with ReLu as the hidden layer activation function. Models were trained for ten epochs with a learning rate of 0.01, a batch size of 128 and no regularization.

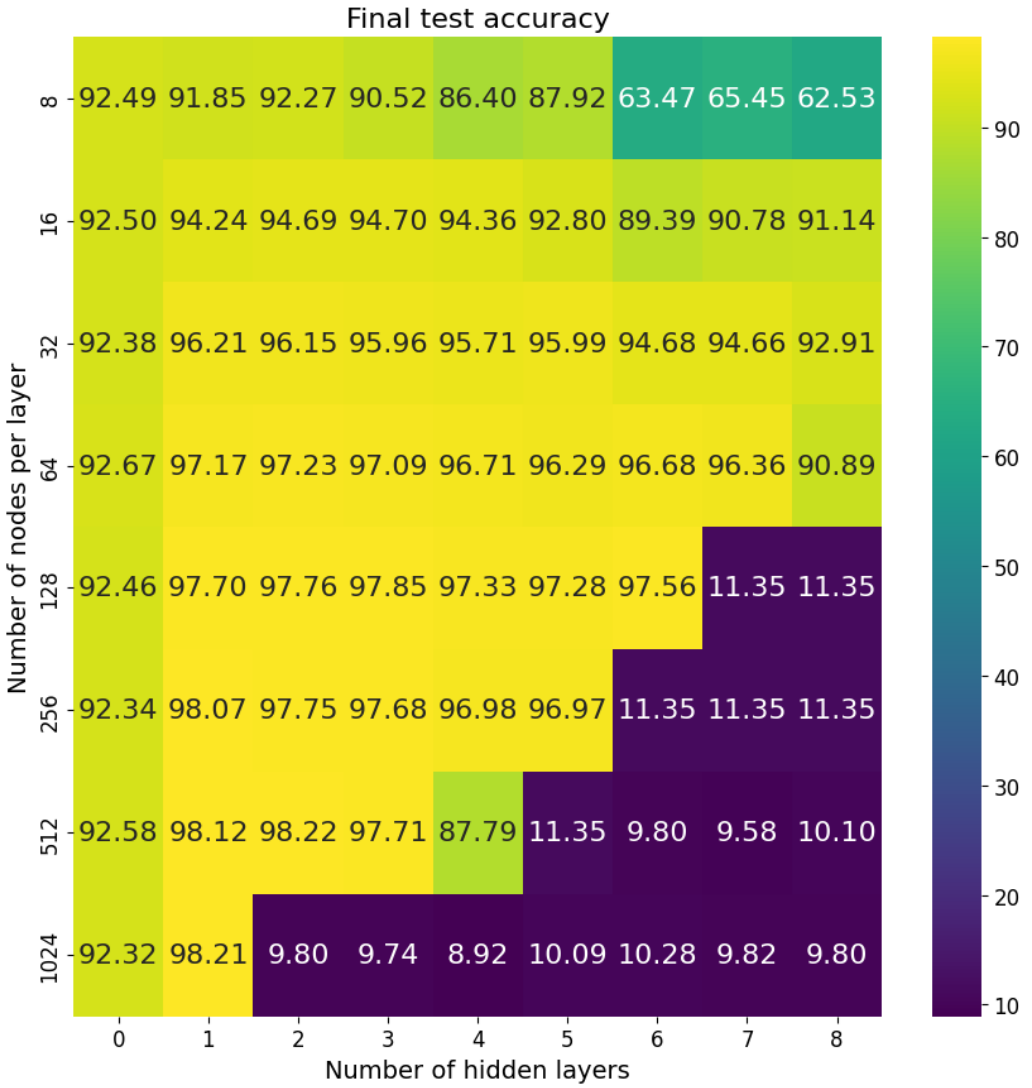

Figure 7: Experiment with training neural networks with varying amount of layers and nodes per layer with Sigmoid as the hidden layer activation function. Models were trained for ten epochs with a learning rate of 0.01, a batch size of 128 and no regularization.

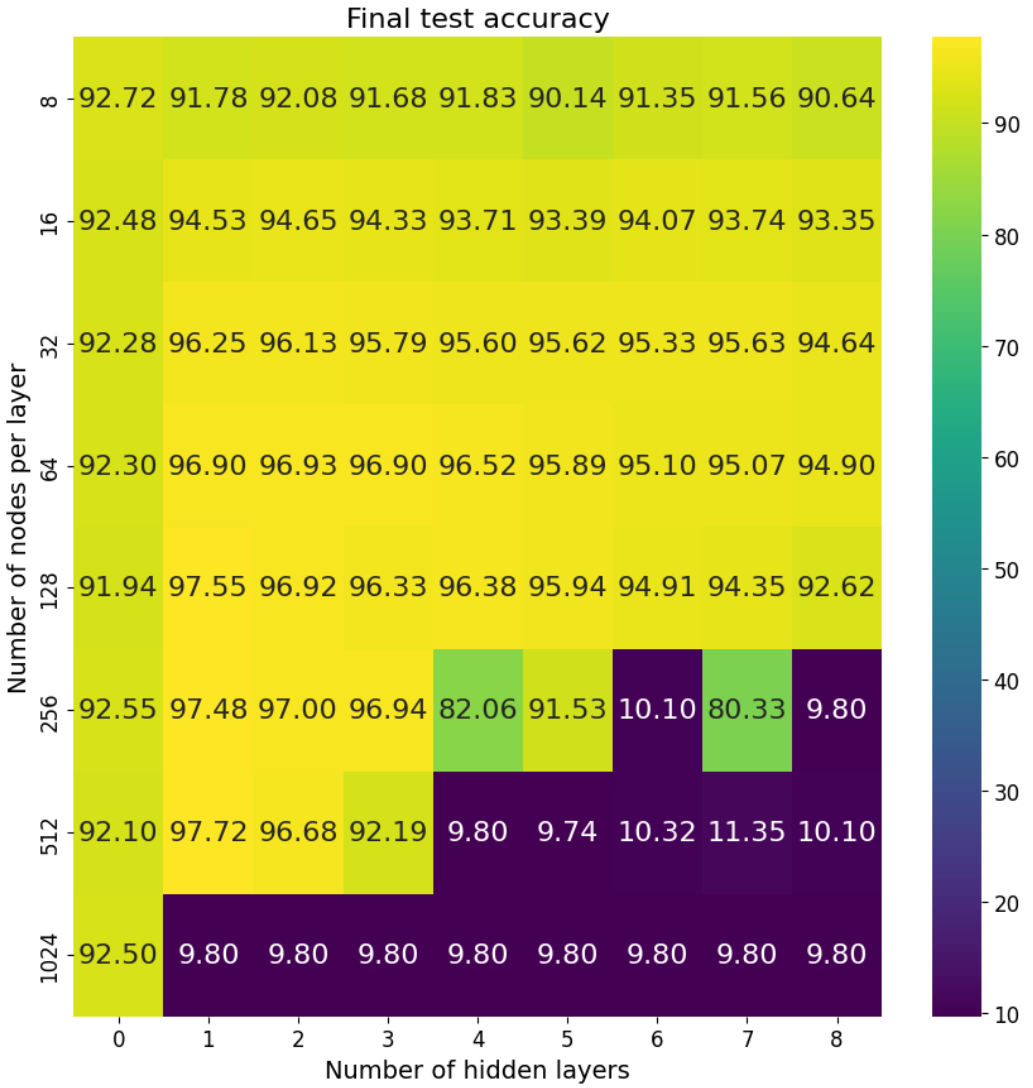

Figure 8: Experiment with training neural networks with varying amount of layers and nodes per layer with Tanh as the hidden layer activation function. Models were trained for ten epochs with a learning rate of 0.01, a batch size of 128 and no regularization.

We see from the figures that for all of the activation functions the model greatly benefits from having one or more hidden layer and that the accuracy jumps from around 92% up to the high 90s. We also see another pattern

in that if the model gets too large then it struggles to learn anything at all and remains at the level where it performs similarly to a random guess. This might be due to the model needing more training time when it is larger and as we trained all of the models for the same number of epochs they might not have converged to a solution in time. We also see in the figures that the model improves as the number of nodes per hidden layer increases, while increasing the number of hidden layers has a negative effect beyond a few hidden layers. The model seems to perform best when the number of hidden layers is one or two, but with a high node count per layer of either 512 or 1024. This pattern can be seen across all of the activation functions, but we do see a small advantage for the model using the sigmoid function as its activation function where we get our best model with an accuracy of 98.2%.

When comparing the neural network model to the softmax model we see that it is able to capture far more of the dataset through its more complex architecture. However we also see that the neural network architecture has a risk of failing completely if the wrong model size is chosen. The neural network model also has the downside of being more computationally expensive to train when compared with the simple softmax model. Despite these downsides the accuracy gained by using the neural network model is so large that it would be the better choice of the two.

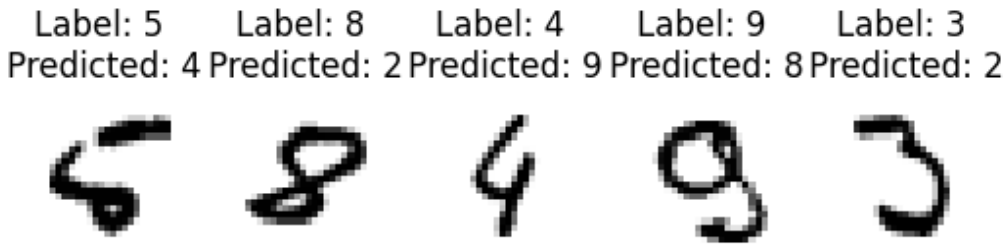

Figure 9: Images that one of the models failed to correctly predict.

Although the neural network model performs well on the dataset we still suspect that there is room for improvement given the 2% error rate. We see in figure 9 that some of the images that the model misinterprets are understandably difficult to distinguis, such as the image with label five. However

some of the other images such as the images with label eight or nine are harder to justify as being something else. From these images we can tell that the model does not have a complete understanding of digits. For this reason we will now explore a slightly more complex model namely a convolutional neural network which will hopefully improve upon the neural network model.

3.3 Convolutional neural network

When studying the we will explore some of the hyperparameters specific to convolutional neural networks and how they affect the performance on the MNIST task. More specifically we will be using a convolutional neural network with two convolutional layers with a max pooling after each layer and then two fully connected layers after that with 128 nodes in each. The size of the filter as well as the number of filters in the first and second layer are the hyperparameters we will vary in this investigation. We chose to use the ReLu activation function for the convolutional layers and the tanh activation function for the fully connected layers.

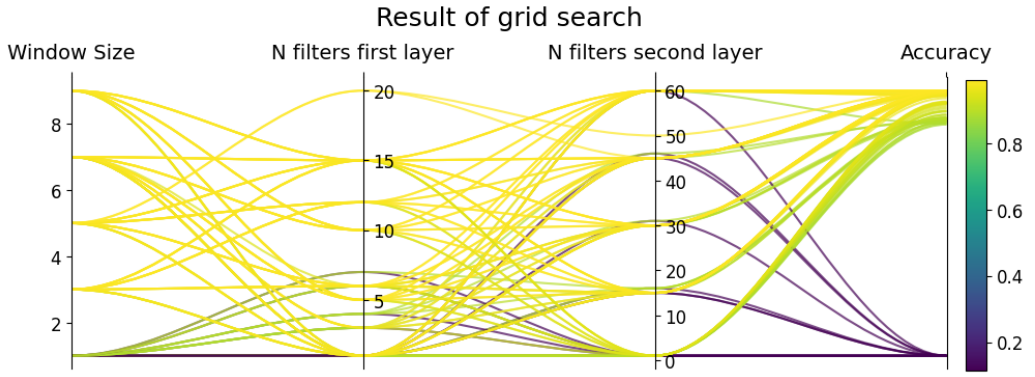

Figure 10: Grid search experiment using a convolutional neural network with two convolutional layers with a max pooling after each layer and two 128 node fully connected layers after the convolutional layers. The models were trained for ten epochs with a learning rate of 0.001, a batch size of 128 and no regularization.

We see in figure 10 that the model performs well on most of the hyperparameter configurations, although the runs with a window size of one has a clear lower performance than the others. This makes sense as with a window size of only one you would not be able to pick up any spatial patterns in

the image making the convolutional part of the network ineffective. With higher window sizes however we see that most of the runs end up performing with an accuracy of above 90%. In our investigation we also found that the number of filters used was highly important to the performance of the model. In order to reach a performance above what the neural network could the model usually would need above 30 filters in the second layer and around ten or more in the first layer. With these configurations we found that the model surpassed the neural network model reaching just under 99% accuracy. Considering the neural network model only had a 2% error rate this new model halves the number of errors on the test set making it a substantial improvement, likely enough to warrant the increase in model size and complexity. A confusion matrix with the results for one of the stronger models is shown in appendix in figure 13.

4 Conclusion

Our study demonstrates that parameter optimization in softmax regression models is crucial for performance. Notably, high learning rates consistently resulted in inferior models, while strong regularization effectively mitigated issues arising from these rates. Optimal performance was achieved with a modest learning rate and minimal regularization, alongside a preference for smaller batch sizes. This configuration yielded a notable test accuracy of 92.9%. Interestingly, increasing the number of epochs from 20 to 50 did not significantly enhance performance, suggesting a balance between computational efficiency and model accuracy. The model's failure to accurately recognize certain digits (e.g., 5, 4, and 9) while easily identifying others (e.g., 3 and 6) highlights a key limitation of neural networks: their propensity for errors uncharacteristic of human cognition. This raises intriguing questions about the potential accuracy of a human evaluator on this dataset, though the task's tedious nature makes it impractical.

Transitioning to a more complex neural network architecture, we observed a significant performance leap, with accuracy rates soaring into the high 90s. This improvement underscores the neural network's ability to capture more nuanced patterns in the data compared to the softmax model. However, this comes at a cost: neural networks are more computationally intensive and risk total failure with inappropriate model sizing. Despite these drawbacks, the considerable accuracy gains justify the preference for neural networks over simpler models for this task. The neural network's failure to completely understand certain digits, even with a low error rate, suggests that while it outperforms the softmax model, there is still room for improvement. This leads us to explore convolutional neural networks (CNNs), which may offer further enhancements in digit recognition accuracy.

Our examination of CNNs, particularly the interplay of hyperparameters like filter size and number, revealed their impact on model performance. The study showed that small window sizes in convolutional layers significantly reduce effectiveness, as they fail to capture spatial patterns. Conversely, models with larger window sizes and an adequate number of filters (around 30 in the second layer and 10 in the first) substantially surpassed the performance of the neural network model, achieving nearly 99% accuracy. This improvement effectively halves the error rate of the neural network model, a significant advancement that likely justifies the increased complexity and

size of CNNs.

Our comprehensive analysis across three model types - softmax regression, neural networks, and convolutional neural networks - highlights the importance of parameter optimization and model selection in machine learning tasks. While simpler models like softmax regression can achieve respectable accuracy, more complex architectures like neural networks and CNNs demonstrate superior performance, capturing finer details in the data. However, this comes with trade-offs in terms of computational demand and model complexity. The progression from softmax regression to CNNs in our study illustrates a critical journey in machine learning: starting with simpler models to establish a baseline, then progressively moving to more complex architectures as needed to enhance performance. The success of CNNs in our tests, especially in terms of accuracy and error rate reduction, underscores their suitability for tasks requiring nuanced pattern recognition, such as digit classification. This journey through various model architectures not only enhances our understanding of their strengths and limitations but also guides future explorations in similar machine learning tasks.

5 Appendix

5.1 Github

Our code as well as jupyter noteboooks used to generate the figures in this paper can be found at: `https://github.com/jonatanwins/Project3_FYS_STK4155`

5.2 Figures

5.2.1 Softmax regression

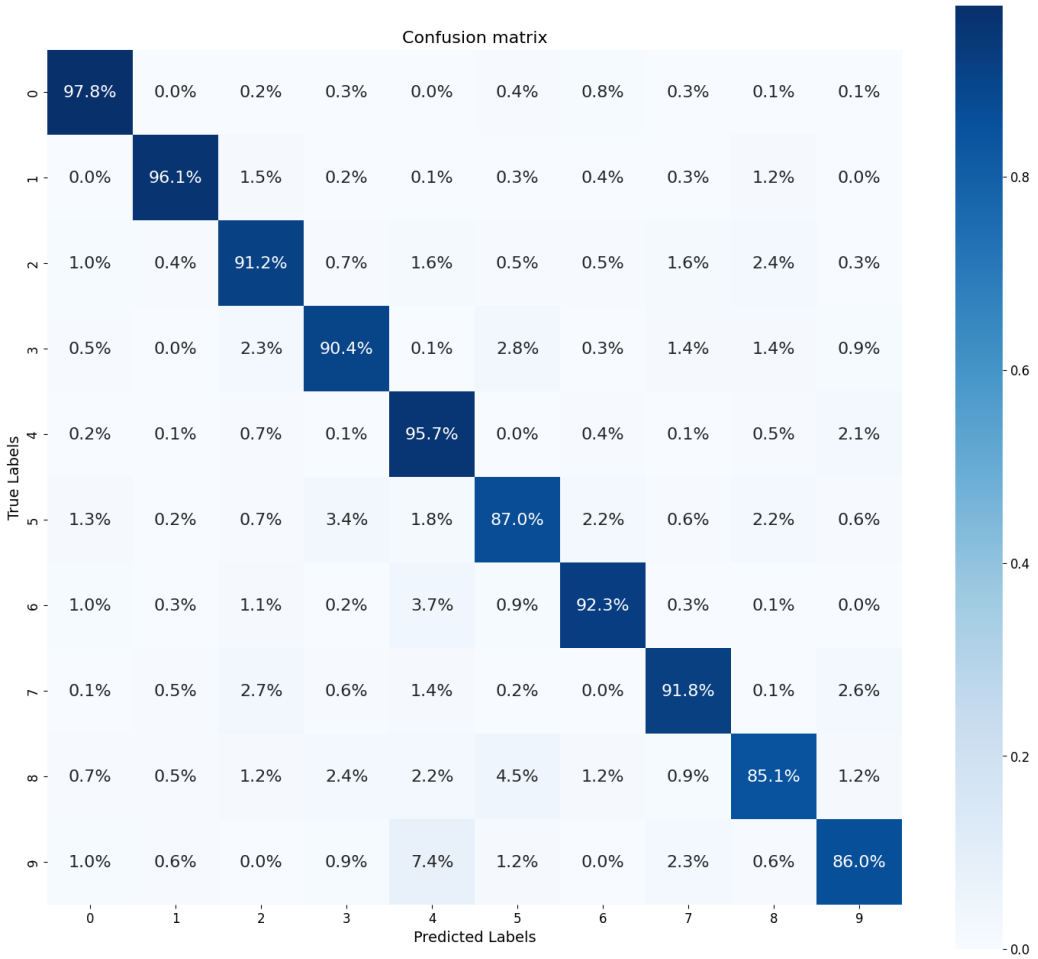

Figure 11: The confusion matrix for the final softmax model.

5.2.2 Neural network

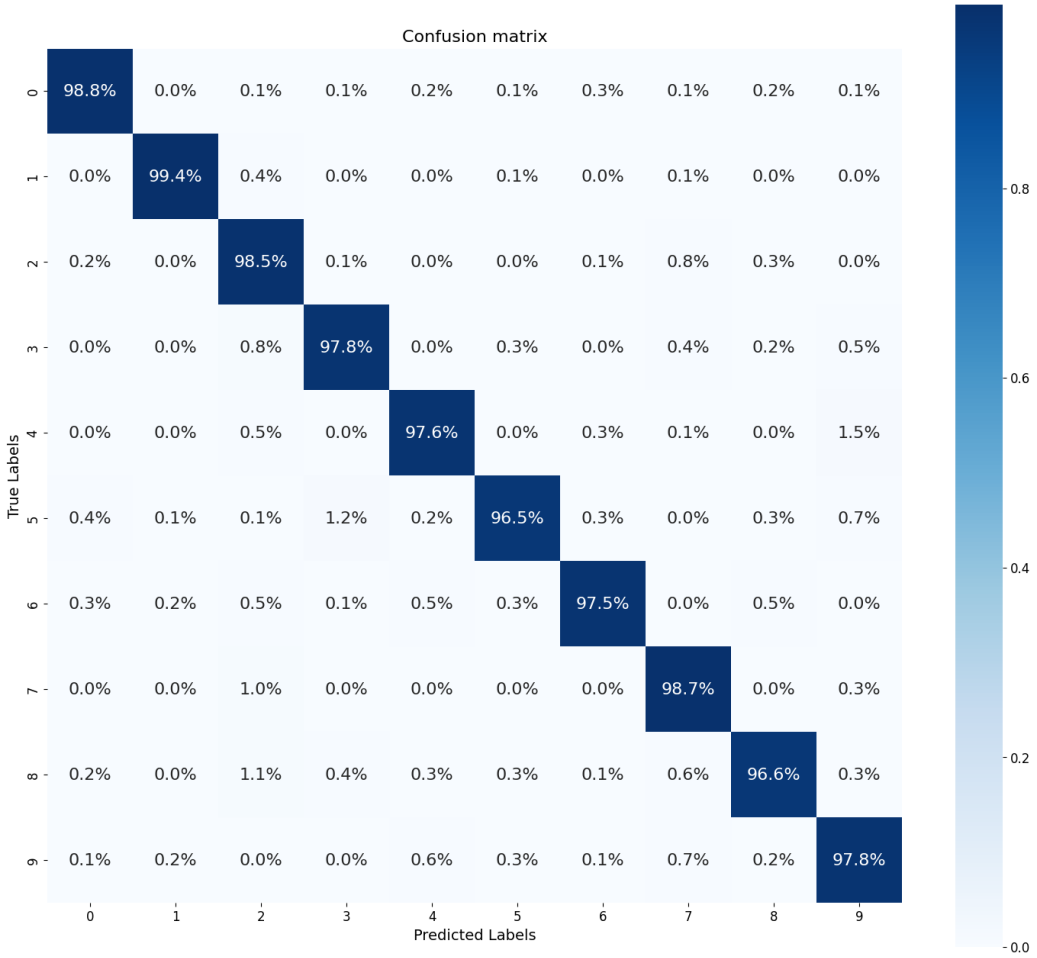

Figure 12: The confusion matrix for the final neural network model.

5.2.3 Convolutional neural network

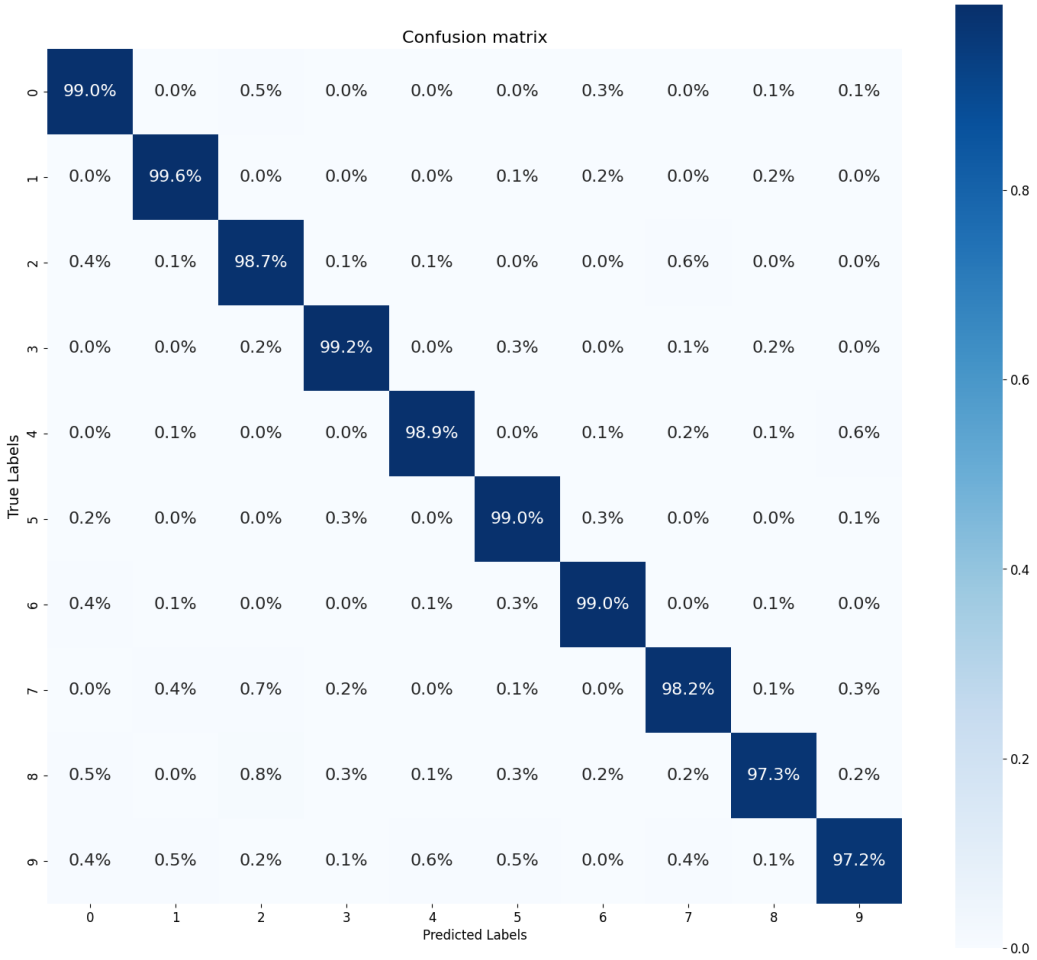

Figure 13: Confusion matrix for a convolutional model with window size 3, with 10 and 60 filters in the first and second layer respectively.

References

[1] Johansen, T. S., Pedersen, E. S., and Winsvold, J. "Project 2". Project report for FYS-STK4155. Nov. 2023. URL: https://github.com/Trond01/Project2_FYS_STK4155.

[2] LeCun, Y., Cortes, C., and Burges, C. "MNIST handwritten digit database". In: *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist* 2 (2010).

[3] Stanford University. *Section 4 (Week 4): Xavier Initialization and Regularization*. https://cs230.stanford.edu/section/4/. CS230 Deep Learning. 2022.