Which established neural network structure is most suitable for fashion-MNIST and does ensembling enhance performance?

Youssef Baccouche (2695472), Anton Donle (2704950), Caspar Grevelhörster (2707848), Sidharth Singh (2708756), Antoni Stroinkski (2673064)

April 1, 2022

Abstract

The handwritten digit MNIST data set plays an important role in the evolution of neural networks as they are known today, more precisely, the use of CNNs in combination with this data set. This paper studies how different neural networks perform for classifying the fashion MNIST data set. The difference in performance between CNNs, LSTMs and Feed Forward Networks has been investigated as well as an ensemble of the best performing networks was created; to determine the possibility of accuracy enhancement within established neural networks, while retaining generalization. The yielded results deduce the following: an ensemble based on majority-voting does not increase accuracy significantly, and the CNN structure was the most successful.

1 Introduction

Machine learning, frequently defined as a machine's ability to mimic intelligent human behaviour, is a branch of artificial intelligence that delves deeper into how machines are trained to learn from past data, without the need for explicit programming. Nowadays, machine learning systems are implemented into countless complex tasks that constantly appear and contribute to our daily lives. Neural networks are a considerable subset of machine learning that congruently encompasses the functions of the human brain. Through the combination of multiple algorithms, neural networks are able to construct correlations and patterns within large sets of data. They can be applied in numerous applications such as facial recognition, reading handwriting, and even medical diagnosis. However, just like us humans, not all neural networks operate identically and we, therefore, have different types. In this experiment, the efficacy of the following three neural network types was studied: Convolutional Neural Network (CNN), Feed Forward Network (FFN), and Long Short-Term Memory (LSTM). To compare the neural network types, tested them on the fashion-MNIST dataset, leading to the following research question:

Which established neural network structure is most suitable for fashion-MNIST and does ensembling enhance performance?

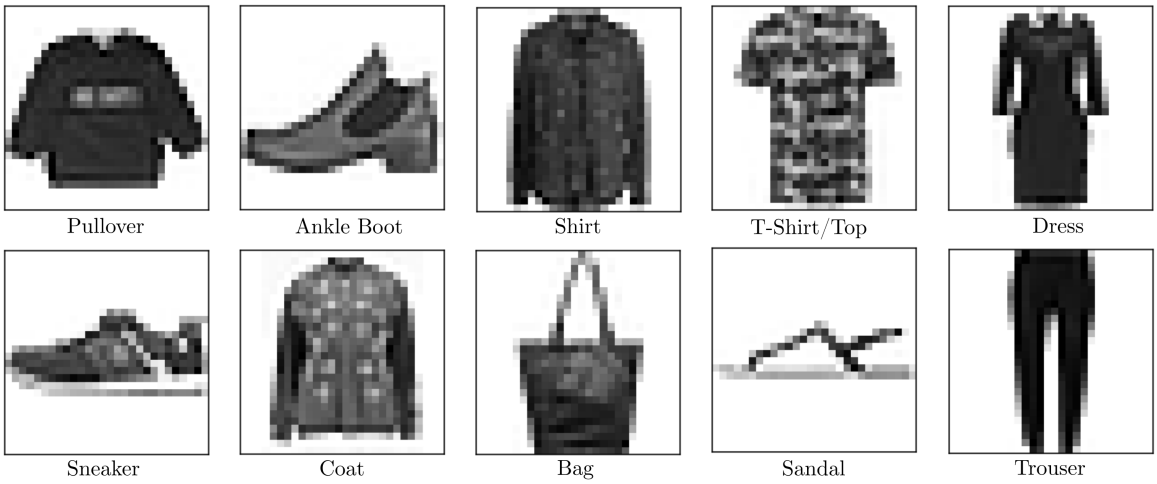
The hypothesis is that an ensemble, based on majority-voting will yield the highest accuracy, while still maintaining its generalization.

2 Dataset

The most popular image recognition dataset is the MNIST handwritten digits dataset. However, as this data is extremely popular and lacks complexity there are a lot of articles about achieving above 99% accuracy. The Fashion-MNIST dataset provided by Zalando is en route to being a successor of the handwritten digits dataset and becoming a new standard for the machine learning industry. This dataset consists of a collection of 28x28 grayscale images of clothing articles, labeled into 10 different categories (Shirt, trousers, etc.; see figure 1). This dataset is divided into a training set of 60,000 examples and a test set of 10,000 examples, to aid in the validation of different algorithms. The aforementioned categories are set into the following 10 classes: pullover, ankle boots, shirt, trousers, sneakers, dress, coat, bag, sandal, t-shirt. The dataset represents each of these classes through 1000

instances (as shown in figure 1) to prevent class imbalance. With a definition of 28x28, each image contains 784 pixels, with each of them carrying individual values for brightness; the higher the value, the darker the pixel. These values are normally decided on a scale of 0 to 255, however, to maximize learning flexibility the data was normalized to scale from 0 to 1,allowing for inputs to be labeled as positive or negative. Collected into an array, these pixels are then analyzed to determine the overall image and establish its class. However, as not all algorithms process input identically, some have to be provided 1-dimensional data whereas others can work with 2-dimensional data. In this case, FNN and LSTM (need a sequence) require the data to be 1d whereas CNN is able to operate on 2d data (specializes in image classification).

Figure 1: All Instances Visualized



3 Experimental Setup

Following the research question, the objective of this paper was the comparison of the most commonly utilized structures of neural networks. Each architectural type will be tested individually and brought to its peak performance using a thorough hyper-parameter optimization process. This will help show the distinct performance limitations of each network type.
As stated in section 2, the MNIST dataset was split into testing and training data, the latter of which was again divided into the actual training and the validation data. This was done in order to avoid overfitting, meaning that the trained model becomes over-optimized for the classification of the training samples leading to the performance reached during training not generalizing to unseen data. This method of cross-validation data is, however, not only used for performance estimation, it also serves as a means for hyper-parameter optimization. In [1], this idea was proposed for the first time; a model is trained with certain specifications which - once performance is evaluated on validation data - can be altered, trained and evaluated again. This process is repeated while the testing data is withheld until the end of the optimization phase and the model performance is finally evaluated using that reserved, unseen data.

Using this procedure of the test-validation split, the hyper-parameter optimization procedure was performed before comparing the network types using one set of testing data that was withheld during optimization. The final aim will be to fuse the presented architectures using the same optimization technique and to land on a new network structure that outperforms the presented ones.

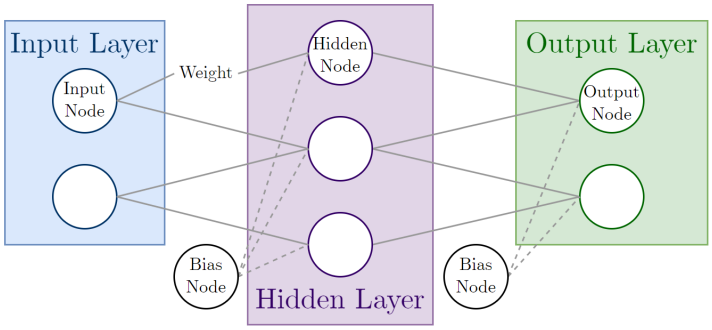
4 Detailed description of the networks

4.1 Feed forward

The first and most simpLeNetwork architecture is a feed-forward network (FFN). This structure was firstly presented in 1965 where [2] proposed the first working learning algorithm for a supervised learning problem using "feed-forward multilayer perceptrons". A later paper by the same authors (see [3]) described the "Group Method of Data Handling", which would today be called a deep network

with several hidden layers that missed back-propagation. Back-propagation is a concept of passing the produced error back through the network to update weights (first described in [4]) plays a fundamental role in today's FFNs since it aids the automatic aggregation of knowledge about importance and potential correlation of individual input features.

Figure 2: Feed-Forward Network Structure

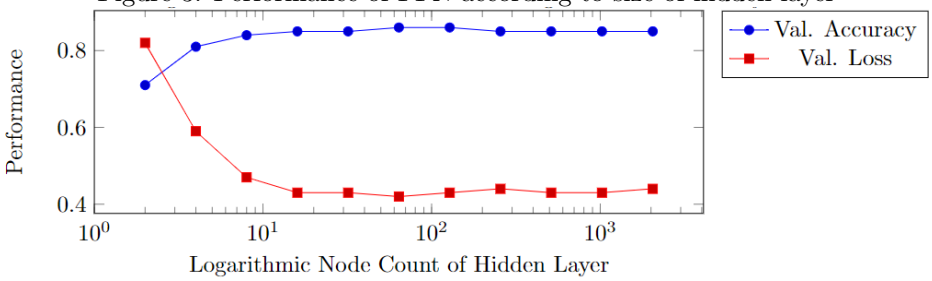


A FFN in its most simple representation can be seen in fig. 2. The input is connected to a hidden layer where weights connect all input nodes to all hidden ones. Since all components are connected with each other, this type of layer is called a *fully connected* layer. Since the numerical values that are handled in this layer only matter for computation but are insignificant for observation of the model, they are called *hidden* layers. The fully connected hidden layer is also fully connected to the output nodes. The final output is calculated using a softmax activation function which ensures that the values of all output nodes together sum to 1. Of each output value o_i, the positive value $(\exp o_i)$ is taken and divided by the sum of all positive output values $exp(o_j)$: $y_i = \frac{\exp o_i}{\sum_j \exp o_j}$. This means that resulting output values can be interpreted as class probabilities, assigning the respective probabilities to the ten different types of clothing.
All FFNs follow the same three basic principles:

1. Information flows from input to output in one direction
2. Information does not flow back and forth between hidden layers
3. Information does not loop within one layer (i.e. not within neurons themselves)

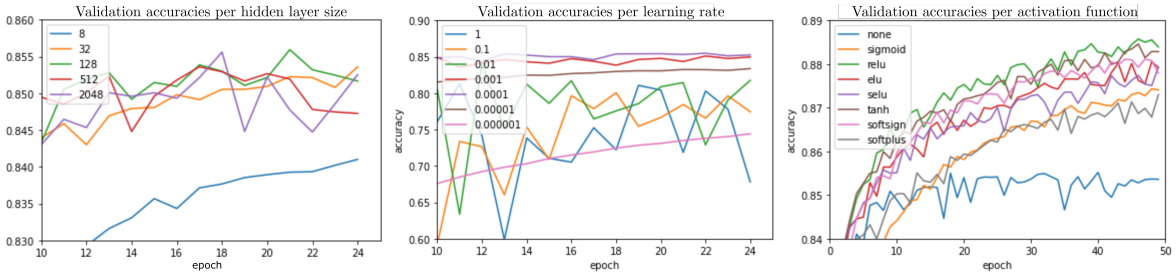
Due to its simplicity, the model structure in figure 2 is the most straightforward to implement. For the MNIST dataset, there are 728 input pixels which are represented by the input nodes in the input layer. Each of those inputs makes up a brightness value of the respective pixel. Due to the abundance of pixel input values compared to the depicted network structure, it would not make much sense to limit the size of the hidden neural network to three hidden nodes as depicted in figure 2. To test the different network performances according to the size of the hidden layer, a test was conducted, producing figure 3:

Figure 3: Performance of FFN according to size of hidden layer



Instead of three nodes in the hidden layer, the performance of the model seems to be reaching a performance plateau above around 128 nodes in the hidden layer. From there, performance does not increase significantly, while computation costs go up due to the increased number of weights. According to this testing, the size of 1024 hidden nodes represents the most well-performing size. However, figure 3 is not fully indicative of performance since it only depicts the overall performance and does not

Figure 4: Performance variation with different hyper-parameters



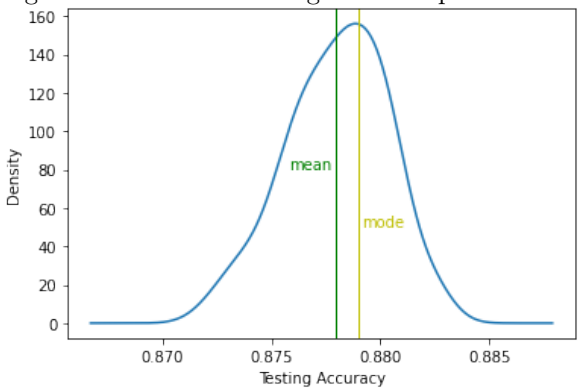
reflect performance fluctuation over epochs. The latter becomes visible when directly comparing the training history of the different sizes over the training epochs (see left figure in 4).
Through plotting the training history of the different models, it becomes clear that with increasing size of the hidden layer, the performance varies more. Combining both findings from both figure 3 and 4, the hidden layer size of 128 is a good trade-off size between fluctuation and validation performance.

Another adjustable hyper-parameter is the learning rate. During backpropagation in this network structure, weights are adjusted according to the following formulas: $w_{ij} = w_{ij} - \eta * 2(y - t) * v_j * x_i$ for all weights connecting the input to the hidden layer and $v_i = v_i - \eta * 2(y - t) * h_i$ for all weights connecting the hidden to the output layer. In both formulas, the learning rate η determines how fast the network *learns* by adjusting how much weights are altered per iteration. As seen in figure 4, a learning rate of $\eta = 0.0001$ is a stable and the best performing choice.

The last improvement of this simple model is an activation function that can be added to the hidden layer. Those functions compute the output of a neuron given the input or set of inputs of the neurons, which makes the neural network a nonlinear function. This means that it can learn more complex relations of data that transcend linear complexity. The function updating the first set of weights is altered in the case of sigmoid to $w_{ij} = w_{ij} - \eta * 2(y - t) * v_j * h_j(1 - h_j) * x_i$ since the pre-processing of the hidden layer must be included in the backward pass. Testing (see figure 4) shows that ReLu is the most performant and fastest converging activation function.

A model was trained 100 times combining all aforementioned optimized hyper-parameters. With a hidden layer size of 128, a learning rate of 0.0001 and the ReLU activation function [5] applied to the hidden layer, after 25 epochs the model has a mean test accuracy of 0.876 and a mode test accuracy of 0.879 (see fig.5).

Figure 5: Statistical testing of model performance

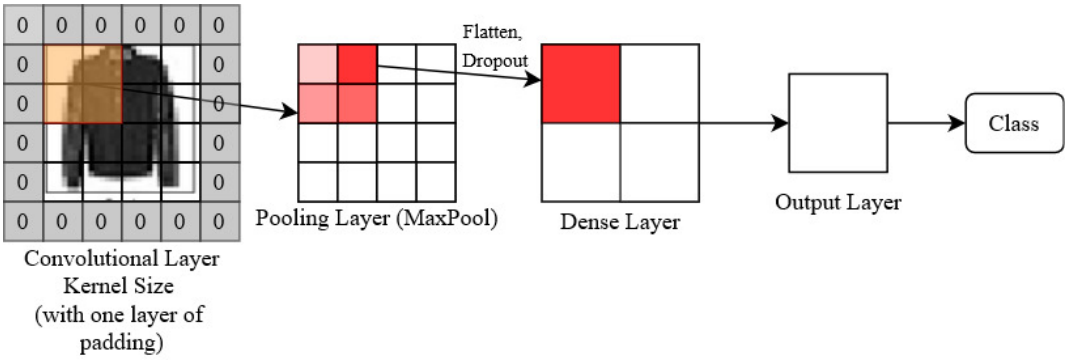


As can be seen in figure 5, this model results in a validation accuracy of around 0.85, which means that approximately 85% of instances in the validation data are correctly classified. This is an impressive result given the simplicity of this FFN with its single hidden layer.

There was also an extensive test conducted altering this single layer perceptron to a multilayer perceptron and experimenting with the amount of layers and their respective hidden nodes. This experiment, however, showed (see results in appendix) that most layer combinations had similar validation accuracy and only those with a very small number of hidden nodes performed significantly worse.

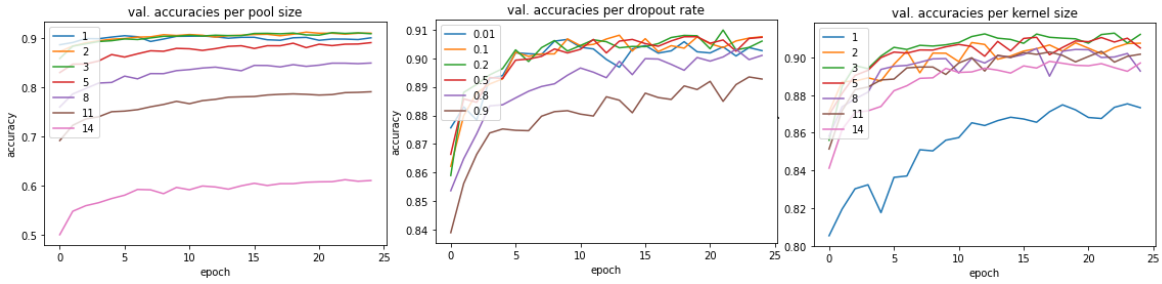
4.2 CNN

Figure 6: Architecture of the simple Convolutional Neural Network



CNN's have achieved great success in image recognition and classification on MNIST datasets. The handwritten digit MNIST dataset was one of the first datasets where a CNN algorithm has been used [6] and became popular. By optimizing the LeNet as suggested by [7], researchers achieved an accuracy of up to 98 % [8]. In this paper, the approach has been taken to firstly experiment with the hyper-parameters on a small scale, making use of a simple, straight-forward CNN network with only one convolutional layer and to use gained insight to design a more powerful LeNet inspired network. The architecture of this network is visualized in figure 6. This has been done in order to investigate the effect that changing certain hyper-parameters has on the performance of a CNN. The difference between both a 1D convolutional layer and a 2D convolutional layer has been investigated. This defines the way the input needs to be shaped. A 1D Input contains all the pixel features in a 1x784 array. For the 2D convolution, the data is arranged in a 28x28 feature space. There has not been much research that compares 1D convolutional to 2D convolution directly, however other researchers also suggest that 2D performs better [9] and 2D is the standard for image processing and other MNISTs. Experiments have been conducted to optimize the architecture of the convolutional layer. In the first place, the effect of the kernel size has been analyzed. The kernel size defines how many nodes the network looks at once. Given a kernel size of 2, the kernel is a square of 2x2 nodes. To keep the dimensionality of the data and to make sure all features are equally often weighted it would apply one layer of padding around the input. Moreover, in this step, a stride is defined, which determines how many steps the kernel moves. With a kernel of two and a stride of one, the output of this layer would again be of the shape 28x28 with updated values. Every connection in the kernel has its own weight that is the same for the whole input. So there are in total 9 weights that are being adjusted by backpropagation. The experiments in figure 8 have proven a kernel size of 3 to be optimal. After the convolutional layer, there is a Pooling layer. Here it has been experimented with the pool size which determines which pixels out of a certain pool of pixels will be kept for further training. Pooling is usually used to reduce the dimensionality of the data, while still retaining as much information as possible. With a pool size of 1, the 28x28 input will be weighed and transferred to the next layer 1 by 1. When changing the pool size to 2 that means it will look at different pairs of 2x2 neighboring nodes. So if no padding is applied it would reduce the dimensionality of the data from 28x28 to 14x14. By conducting experiments a pool size of 2 has proven to be the most suitable for this problem. As can be seen in figure 8. Moreover, average pooling was compared to max pooling, coming to the conclusion, that MaxPooling performs better. This seems to also have been the case for other researchers [10] Average pooling takes the average of the nodes (in this case 4) from the convolutional layer and passes its value to the corresponding node in the next layer whereas MaxPooling passes on the highest value (out of the four). The pooling layer is followed by a dropout and flatten function. Hyper-parameter optimization showed that a dropout in between 0.2 to 0.5 is a good choice. The figure 6 shows a simplified version of our CNN using kernel size 2x2 and a pool size 2 as an example. As mentioned above the LeNet is a state-of-the-art algorithm for the MNIST dataset. After having investigated the hyper-parameters most suitable for our dataset these insights have been used to design a more complex convolutional network, inspired by the structure of the LeNet. The network makes use of 3 different convolutional layers, followed by pooling layers. All parameters, activation functions, and learning rates that were established previously have been used in this model. This network achieved a test accuracy of 93%.

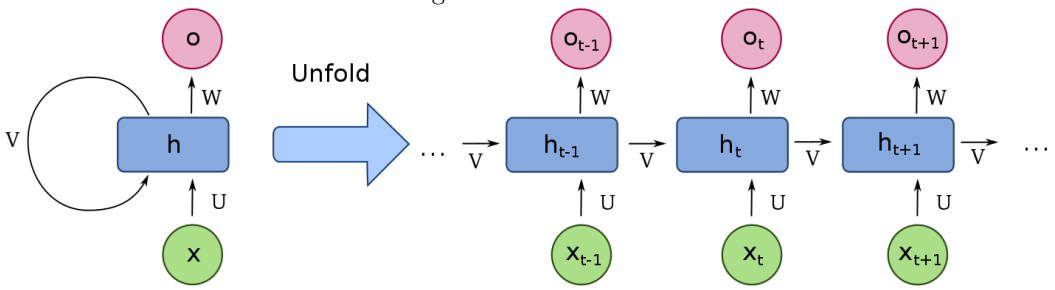
Figure 7: Results of CNN hyper-parameter testing



4.3 LSTM

The last network architecture is a type of recurring neural network - long short-term memory. A recurring Neural network creates a loop between hidden layers, which makes the previous layer influence the next one. The number of cycles is a hyper-parameter, which also significantly influences the training time, as the backward pass has to calculate everything in a particular order.

Figure 8: RNN structure



LSTM was firstly introduced in 1997 [11] to solve the issue of disappearing gradient (which happened a lot in RNN, because of the layer loops). In recurring neural networks, this issue influences the models even more than in the previous examples, because, with each iteration, there are more and more calculations that later on have to be evaluated for the weight update. To prevent the decaying loss error LSTM uses memory cells and gate units. The firstly introduced LSTM structure was missing one important part, which is being used nowadays, the forget gate. It was introduced in the year 2000 [12]. The LSTM memory cell is built with three main parts: forget gate; input gate; output gate. LSTM is a sequential network, that allows information to persist. Treating an image as a sequence of pixels allows the usage of LSTM for the classification problem. The main difference between classical RNN and LSTM is the influence of information that has been learned in the past. As previously mentioned RNNs are struggling with reusing the information that is distant in the sequence (i.e. data from the beginning, when the model is getting close to the end), which results in low accuracy and high error. The three parts of the memory cell, that build LSTM, prevent that from happening. The forget gate decides which information should be stored, and which information should be discarded because it is useless for the next iteration. The values range in between 0 and 1, 0 meaning to forget and 1 to remember. In order to update the memory cell state, the input gate is used. It's achieved by combining the previous state with the current input and applying the sigmoid function. As before values range between 0 and 1, informing about their importance. The last output gate decides what to pass to the next memory cell.
As previously mentioned the similar testing has been done to choose the hyper-parameters. The interesting finding is that the learning rate highly influences the performance. The best accuracy is achieved with 0.001 learning rate.

The different activation functions were also tested, however it yielded extremely poor results and it was not worth the effort as it took approximately 3 hours (50 times more than normal LSTM training).

The Classification report shows that LSTM (other models as well) is struggling with Shirt, Coat and Pullover classification. The best performance is achieved for Bag and Trousers and there are no other clothes categories that are of the similar shape.

Figure 9: LSTM Structure

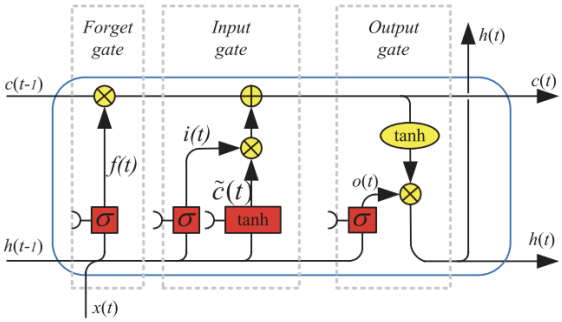


Figure 10: LSTM learning rates

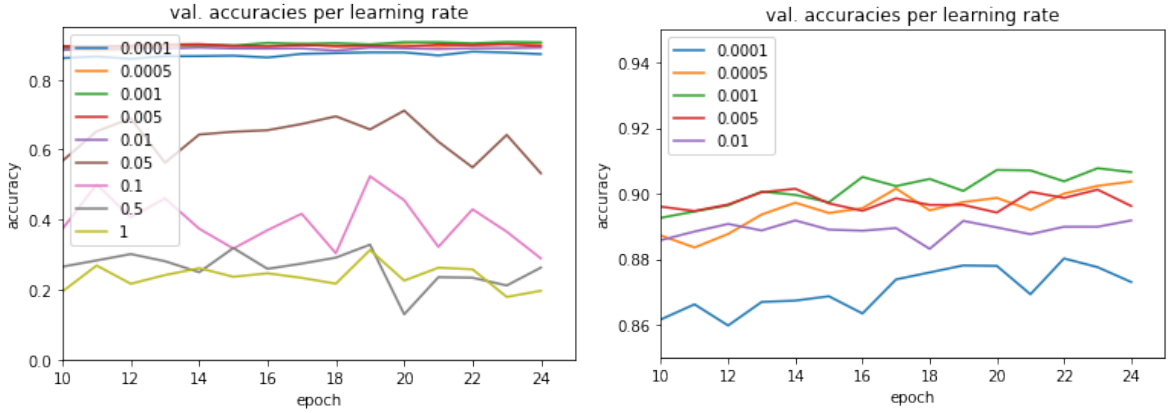
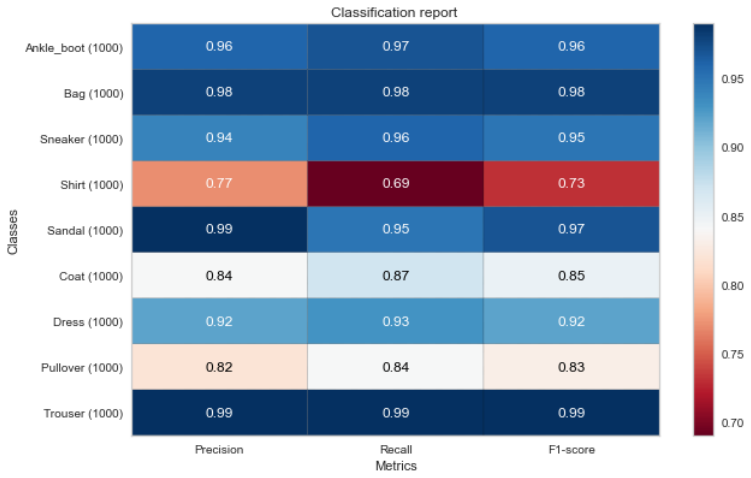


Figure 11: LSTM Classification report



5 Ensembling

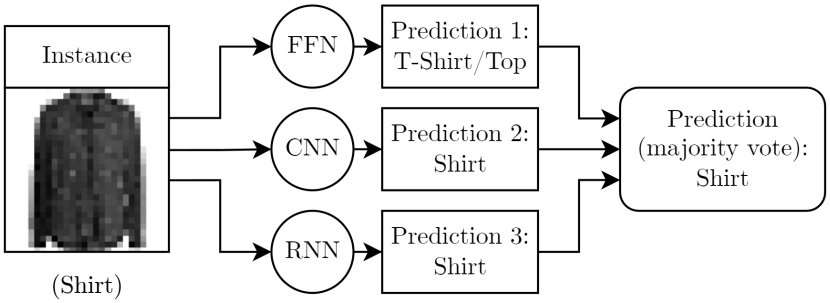
Machine learning algorithms are often hard to optimize beyond a certain threshold. Often, a balance between obtaining a highly accurate model and still having a model that generalizes well has to be found. In an effort to enhance the model performance while maintaining its generalization, model ensembling can be used.

In an ensemble model, different models are combined to aggregate their predictions and yield one accurate outcome. This can be done with so-called weak learners, where every weak learner is one algorithm that focuses on one aspect of the data, making that algorithm inaccurate for the whole

data set on its own. However, multiple weak learners that focus on different aspects of the data in conjunction can be highly effective. This approach is known as adaptive boosting or AdaBoost. Another technique that is often used is bagging, also called bootstrap aggregating. In bagging the training dataset is resampled for each algorithm in the ensemble to reduce variance. This technique is used in random forest ensembles.

However, for this research, the goal was to combine the three prior introduced models. To achieve this it was evaluated whether stacking would be viable. In stacking the prediction made by the trained models on the original dataset are added to the feature space to train a new model, the combiner. However, It was decided to not implement this method because of the high risk of data leakage associated with stacking. The approach that was ultimately chosen is that of majority voting. Hereby each classifier classifies the input, in this case, a 28 x 28 picture from the Fashion-MNIST dataset, and the class the instance was classified as by the majority of the classifiers will be assigned to the instance 12. This entails that this ensembling method does not require the training of any additional models, because it uses the convolutional neural network, feed-forward network, and LSTM that were trained beforehand and simply uses the mode of their aggregated predictions as its final prediction. In case all three classifiers assign different classes to one instance, the classifier that had the highest confidence will determine the assigned class.

Figure 12: Ensemble with majority-voting



After implementing the described approach, it was found that the ensemble yields a performance of 91.2% on the test data. While this performance confirms that ensembling in the form of majority voting can be a viable method, it must be considered that this method takes significantly more time in making a prediction for a single instance, because all three classifiers have to classify the picture before the final classification can be made. Especially in a production or business environment, this could be a major downside, as prediction time is essential. Furthermore, the accuracy of the overall ensembling model is not significantly higher than the individual models that together make up the ensemble. A reason for this could be that all of the three base models were very similar in performance. In figure 13 it is seen that the three models are accurate for the same classes as well as inaccurate for the same classes as can be seen for the class "Shirt". When ensembling models, combining base models that perform similarly on the same aspects of the data will likely result in a very similar performance for the ensemble.

6 Evaluation

This research focused on classifying pictures of clothing types provided by the Fashion-MNIST dataset. This dataset turned out to be suboptimal to find performance differences between different network architectures as even simple neural networks achieved high test accuracies. Nevertheless, it was found that the three network architectures of FFN, CNN, and RNN in the form of an LSTM that were tailored to this task all yielded highly accurate classifiers. The simplest and least accurate of the classifiers was the feed-forward network which was implemented as a simple baseline model in order to investigate how more complex models would fare against it. Both LSTM and CNN yielded similar results, but due to the lower training time for a CNN, for this problem, a CNN model can be seen as the most favorable out of the three. When the three models were combined into one ensemble it was found that, whereas the ensemble was very accurate, it is not significantly more accurate than its individual parts. When evaluating the results of the hyper-parameter optimization for CNNs (figure

Figure 13: Comparison of Confusion Matrices of different Model types

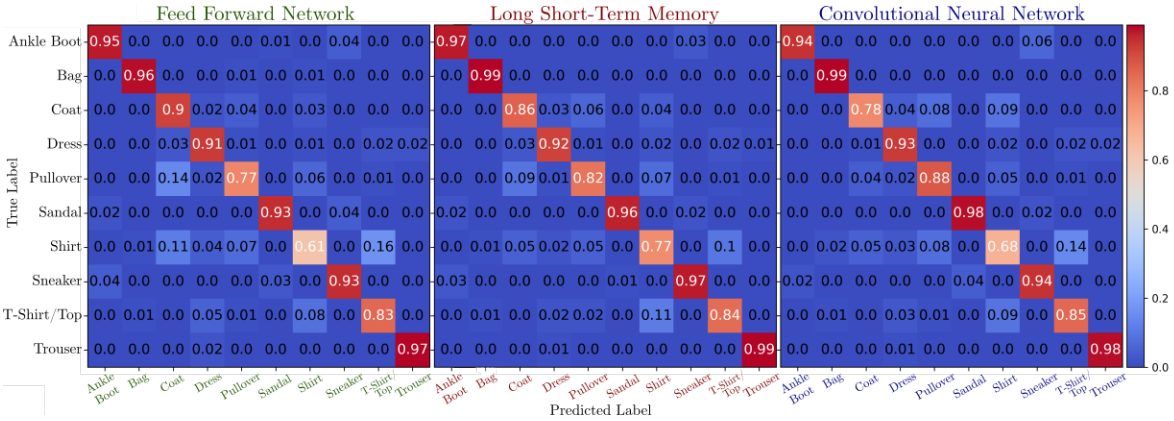
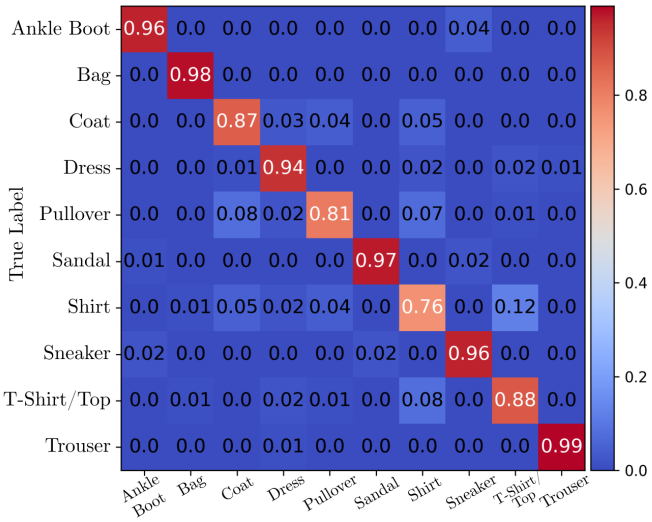


Figure 14: Confusion matrix of the Ensembling Model



8) it was outstanding, how similar models with different hyper-parameters performed. When altering the kernel size the results from 2-14 were within a margin of 5% in the first epochs and that difference nearly vanished after 10 epochs. Similar results can be seen in the dropout rate and the pool size. Only very extreme values like a dropout of 0.9 or a pool size of 14 could significantly change the performance of the algorithm

It was also concluded that all models struggled with distinguishing the same three classes that were very similar in appearance as can be seen in figure 14. This issue cannot be solved with an ensembling method that is based on majority voting if all of its base models perform similarly in the same aspects of the data. One way to solve this would be to reconsider stacking or blending. Constructing several weak learners that perform well on specific partitions of the data and adding their predictions to the existing features to train a combiner, might result in a model that can handle the whole of the data well, while still generalizing. Therefore, for future research, it would be interesting to train models specifically set up to distinguish pullover, shirt, and t-shirt - classes that are often confused with one another.

Furthermore, it was also investigated why all implemented models reached similarly high performance on the testing data. Even a simple FFN architecture with 128 nodes in the hidden layer reached 89% accuracy according to our testing. [13] proposed to delete instances from the testing that were very similar to the training data. While this makes sense because instances that were already seen by the model artificially inflate performance on the test set, it could be argued that this similarity might also occur in a business environment. The method of [13] was deployed and the test data was cleaned, however, it was found that the performance did not significantly differ between the datasets; the FFN still performed reasonably well and much better than random chance. The similar instances are important examples of the data distribution and thus should remain in the data. For fashion-MNIST the most suitable structure seems to be a CNN based on the combination of the observed accuracy

and low training time. Overall, the ensemble out of the FNN, CNN, and LSTM did not result in a significant enough performance increase over its base models, so we can reject the hypothesis that a majority-vote ensemble will yield the highest accuracy out of the tested models. For future research, it would be interesting to investigate whether a different, more complex, ensembling approach would yield significantly higher accuracy.

References

[1] M. Stone, "Cross-validation and multinomial prediction," *Biometrika*, vol. 61, no. 3, pp. 509–515, 1974, doi: 10.1093/biomet/61.3.509.

[2] A. G. Ivakhnenko and V. G. Lapa, "Cybernetic predicting devices," PURDUE UNIV LAFAYETTE IND SCHOOL OF ELECTRICAL ENGINEERING, Tech. Rep., 1966, doi: no_doi_available.

[3] A. G. Ivakhnenko, "Polynomial theory of complex systems," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-1, no. 4, pp. 364–378, 1971, doi: 10.1109/TSMC.1971.4308320.

[4] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, pp. 533–536, 1986, doi: 10.1038/323533a0.

[5] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun, "What is the best multi-stage architecture for object recognition?" in *2009 IEEE 12th international conference on computer vision*. IEEE, 2009, pp. 2146–2153, doi: 10.1109/ICCV.2009.5459469.

[6] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998, doi: 10.1109/5.726791.

[7] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989, doi: 10.1162/neco.1989.1.4.541.

[8] Y. Wang, F. Li, H. Sun, W. Li, C. Zhong, X. Wu, H. Wang, and P. Wang, "Improvement of mnist image recognition based on cnn," in *IOP Conference Series: Earth and Environmental Science*, vol. 428, no. 1. IOP Publishing, 2020, p. 012097, doi: 10.1088/1755-1315/428/1/012097.

[9] J. Zhao, X. Mao, and L. Chen, "Speech emotion recognition using deep 1d & 2d cnn lstm networks," *Biomedical signal processing and control*, vol. 47, pp. 312–323, 2019, doi: 10.1016/j.bspc.2018.08.035.

[10] D. Yu, H. Wang, P. Chen, and Z. Wei, "Mixed pooling for convolutional neural networks," in *International conference on rough sets and knowledge technology*. Springer, 2014, pp. 364–375, doi: 10.1007/978-3-319-11740-9_34.

[11] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, pp. 1735–80, 12 1997, doi: 10.1162/neco.1997.9.8.1735.

[12] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with lstm," *Neural Computation*, vol. 12, no. 10, pp. 2451–2471, 2000, doi: 10.1162/089976600300015015.

[13] C. Geier, "Training on test data: Removing near duplicates in fashion-mnist," *arXiv preprint arXiv:1906.08255*, 2019, doi: arXiv:1906.08255.

A Appendix: Feed Forward Network Training

Table for testing that was conducted in order to find the most optimal combination of hidden layers. The results were not as indicative as expected which is why this table was only included in the appendix.

- input layer - hidden layer (relu activation) - output layer (softmax activation)

Figure 15 depicts the performance of the LENET-inspired CNN.

Table 1: Testing hidden layer sizes of multilayer perceptron

Sizes of hidden layers	Validation Accuracy	Validation Loss
128 64 128	0.89	0.32
128 128 128 128 128	0.89	0.32
128 256 128	0.89	0.31
128 512 128	0.89	0.31
128 256	0.89	0.31
128 128 128	0.89	0.31
128 128 128 128	0.89	0.31
128 64 32 16 8	0.88	0.36
128 64 32	0.88	0.35
128	0.88	0.35
128 32 8	0.88	0.35
128 64 32 16	0.88	0.33
128 64	0.88	0.33
128 128	0.88	0.33
128 16 128	0.88	0.33
128 4	0.86	0.44
128 4 128	0.86	0.4
128 64 32 16 8 4	0.83	0.55
128 2 128	0.8	0.54
128 1 128	0.67	0.87
128 64 32 16 8 4 2	0.61	1.33

Figure 15: Performance During Testing of CNN