Universitat Politècnica de Catalunya

MIRI-KMLMM

Fashion MNIST

Authors:
Julian Kunze
Miro Pütz
Sonja Ginter

January 9, 2019

Contents

1 Introduction

The objective of our project is to classify clothes by using a machine learning approach. For this purpose, the scope of our work covers comparing a 2-layer neural network with a convolutional neural network. The employed dataset is the "Fashion MNIST" from Zalando.

Fashion MNIST data set Fashion MNIST [1] is a dataset of 70'000 images of Zalando articles. It is already split into training and test set (60'000 to 10'000). It is a simple classification dataset which contains 28×28 pixel gray-scale images, associated with labels (see figure 1). The following labels are contained:

1. t-shirt/top
2. trouser
3. pullover
4. dress
5. coat
6. sandal
7. shirt
8. sneaker
9. bag
10. ankle boot

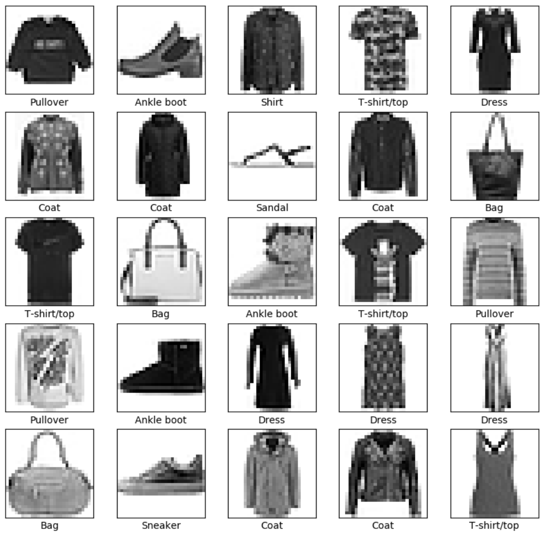

Figure 1: 25 sample images and their respective label from the "Fashion MNIST" dataset

2 Theory

2.1 Basic Theory of Artificial Neural Networks

Artificial Neural Networks (ANNs), often just described as Neural Networks (NNs), also known as Multilayer Perceptrons (MLPs), are currently probably the most researched machine learning technique whereby they also gained a lot of industrial and public attention because of their wide range of fields of application. The term 'neural network' derives from the biological neural network in animal brains, from which the artificial counterpart's structure is loosely inspired. Compared to other machine learning methods, neural networks are more difficult to optimize as their likelihood function generally is non-convex. Therefore, they need special optimization techniques as described in section 2.5 and for the most part more time and data for training.
Neural networks usually consist of one input layer, where the input variables are introduced into the network. From the input layer, the input values are distributed to the first hidden layer. The activation of the hidden layer can be computed as the linear combination of all its weighted incoming nodes plus some constant bias:

$$a_j = \sum_{i=1}^{D} w_{ji} x_i + w_{j0}$$

Here, a_j denotes the activation of the jth neuron in the hidden layer, x_i the input of the ith input neuron, w_{ji} the weight that input x_i get for the jth hidden layer neuron and w_{j0} defines the bias term for that neuron. This activation function then gets processed by a non-linear activation function (see section 2.2) and passed over to the next layer. In case of deep neural networks there are usually several hidden layers present that allow the network to learn complex relationships between input and output. Figure 2 shows a graphical representation of a much simpler single hidden layer (2-layer) fully-connected neural network. When the processed data gets passed to the last layer, the output layer, the same weighted summation of incoming nodes is performed. Afterwards, this activation is again processed by an activation function where usually the hidden layers employ a different function than the output layer. The choice of activation functions is usually data dependent. This pass of the data from the input layer straight to the output layer is commonly called 'feed-forward' and the corresponding type of networks are feed-forward neural networks. There are other kinds of neural networks that are designed differently but due to their irrelevance for this work not described in more detail. One special kind of feedforward neural networks, the so-called convolutional neural networks are a crucial part of this work and thus explained more detailly in section 2.6.

The actual learning of the network is integrated by the backpropagation method where the gradient of the loss function (see section 2.3) is propagated from the output

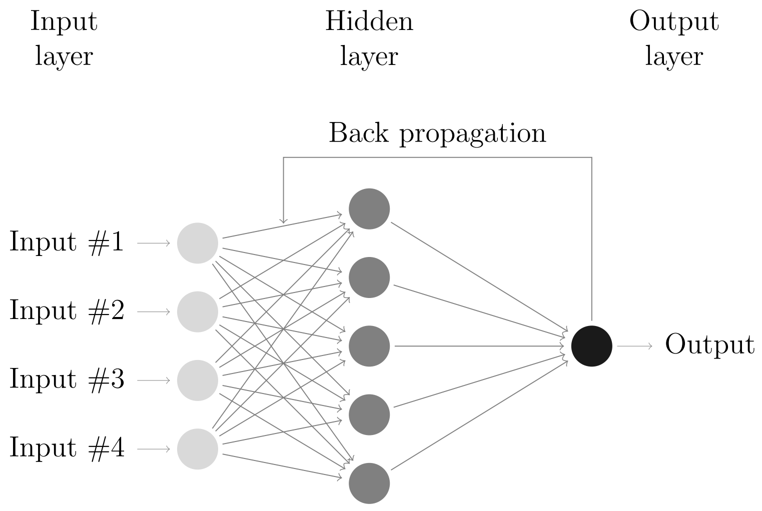

Figure 2: Conceptual drawing of a 2-layer neural network. The network is full connected and is assembled of 4 input neurons, 5 hidden neurons and a single output neuron.

back to all weights in the network in order to gradually converge towards the optimal weight distribution. This is illustrated in figure 2 by the backwards pointing arrow. This is done by computing the derivatives of the loss function with respect to every layer's weights and then using this derivative in combination with the network's learning rate to update each weight individually. [2]

2.2 Activation Function - ReLU

An activation function transforms the weighted sum of inputs of a neuron into a non-linear output in order to give an artificial neural network non-linear capabilities. Most recent neural network use a rectified linear unit (ReLU) as their main activation function. The ReLU function is a piece-wise linear (together non-linear) function defined as $f(x) = \max\{0, x\}$ and thus identical (zero) for all negative inputs. It is easy to optimize due to its similarity to a linear activation function. It also features relatively large and consistent gradients useful for positive activation. This comes with the drawback that they cannot learn with negative activation since the gradient will be zero there. If one wants to incorporate learning for negative inputs it possible to use the parametric (learnable parameter a), or leaky ReLU (fixed parameter a, often 0.1 or 0.01). This activation function is then defined as $f(x) = \max\{0, x\} + a \min\{0, x\}$ and thus also produces gradients for negative activation. [3]
Figure 3 shows the difference between both functions.

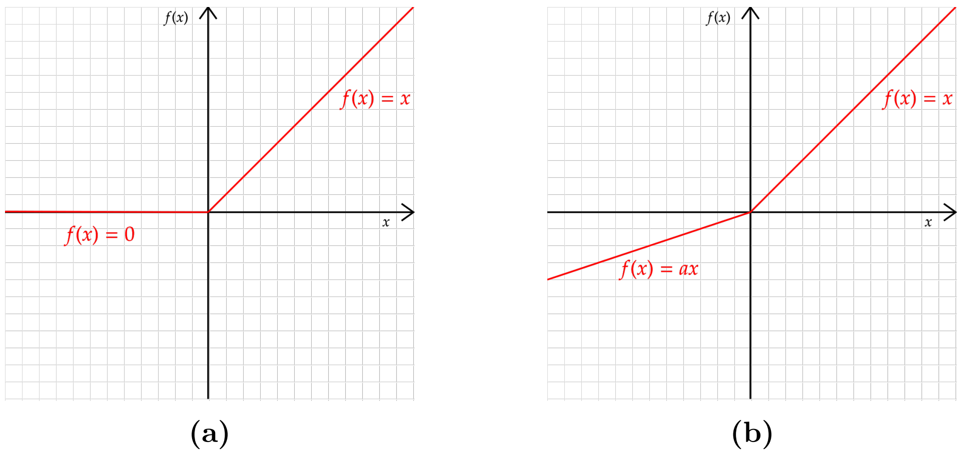

Figure 3: Graphical representation of the ReLU (a) and the parameterized ReLU (b) activation functions.

2.3 Loss function - Cross-Entropy

The loss function, or error function is a function that describes how well our network is currently performing. It usually involves a comparison between prediction and ground truth and is optimized (minimizing the loss) throughout the learning process. Using some form of gradient descent is mostly used to ideally find the global optimum[1] of the loss function. The cross-entropy loss function defined as $H(\boldsymbol{y}, \hat{\boldsymbol{y}}) = -\sum_i y_i \log(\hat{y}_i)$ is a popular choice because it does not suffer that much from saturation and slow learning compared to the mean squared error. In the definition, y_i denotes whether the sample's true class is i ($y_i = 1$) or not (y_i=0), $\hat{y}_i$ denotes the respective predicted probability for the sample being of class i. The summands all become zero except for the i being the true class. Therefore, the only summand is that of the true class and here, the loss becomes logarithmically smaller the higher the predicted probability of that class is. The cross-entropy loss is mostly used in combination with a softmax output layer as this output layer directly gives the respective probability predictions for each class. [3]

2.4 Minibatch Size

Processing a whole dataset at once usually is not efficient. Computation parallelization on multi-core computer architectures often provides a significant improvement in computation-time. Recently, graphics cards are commonly used for efficient training of neural networks as they are especially good at computation parallelization. For this purpose, the dataset is split up in smaller slices, so-called minibatches. The

[1]Often finding a local minimum close to the global one is also acceptable.

most used optimization algorithm applied using minibatches is probably the stochastic gradient descent (SGD) and its derivations. For all samples in the minibatch, it computes the respective gradient. The network parameters are then updated performing a gradient descent with the mean of all gradients of this minibatch:

$$\begin{aligned}\hat{\boldsymbol{g}} &= \frac{1}{m}\nabla_{\boldsymbol{\theta}}\sum_i L\left(f\left(\boldsymbol{x}^{(i)};\boldsymbol{\theta}\right),\boldsymbol{y}^{(i)}\right)\\ \boldsymbol{\theta} &= \boldsymbol{\theta} - \epsilon\hat{\boldsymbol{g}}\end{aligned}$$

where $\hat{\boldsymbol{g}}$ is the average minibatch gradient, m the minibatch size, $\nabla_{\boldsymbol{\theta}}\sum_i L(\dots)$ the gradient of the loss function. $\left(\boldsymbol{x}^{(i)},\boldsymbol{y}^{(i)}\right)$ defines features and label of the ith sample and $\boldsymbol{\theta}$ characterizes the network parameters. Finally, ϵ expresses the learning rate as defined in section 2.5.
Using stochastic gradient descent with smaller minibatches leads to less accurate results. On the other hand, the noise that this more inaccurate computation adds to the process also acts as a regularization effect.
Depending on the available architecture, often minibatch sizes of a power of 2 are used in practice and chosen in a trade-off between computation time and training stability. [3]

2.5 Learning Rate/Optimizer

Optimization of the weights is achieved by backward propagation of the error during the training phase. The network modifies its weighted links to minimize the loss function and thus the discrepancy between prediction and ground truth labels. This is process is usually repeated for several runs over the complete dataset until a certain accuracy is reached on the validation dataset. [4]

Adam (short for 'adaptive moments') is one frequently used optimization algorithm introduced by Kingma and Ba [5] that adaptively changes its learning rate. It is a combination of the RMSProp algorithm and momentum. RMSProp, on the other hand, is an extension of AdaGrad. The algorithms are presented shortly in the following:

- **Ada**ptive **Grad**ient Algorithm (AdaGrad): This algorithm adapts the learning rate to the parameters, performing smaller updates (i.e. low learning rates) for parameters associated with frequently occurring features, and larger updates (i.e. high learning rates) for parameters associated with infrequent features. For this reason, it is well-suited for dealing with sparse data.[6]
- **R**oot **M**ean **S**quare **Prop**agation(RMSProp): In contrast to AdaGrad this method "uses an exponentially decaying average to discard history from the extreme past" [3]. By this, it performs significantly better when applied to non-convex functions compared to AdaGrad[7]

- **Ada**ptive **m**oments (Adam): Adam not only makes use of the first moment (mean) but also uses the second moment (variance) to adapt its learning rate. It also includes a bias correction to both first- and second-order momentum estimations. This makes the algorithm relatively robust to hyperparameter choices. [5]

The Adam algorithm proved to be quite effective for many cases, for instance applied on the original MNIST dataset shown by Kingma and Ba [5].

2.6 Basic Theory of Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a specialized subclass of Artificial Neural Networks mostly used for visual classification and detection tasks. They are first introduced by LeCun et al. [8] and, at that time, improved the state-of-the-art results significantly in terms of generalization capabilities as previous approaches often used hand-crafted features. CNNs, instead, learn the feature extractor themselves and therefore supersede hand-crafted feature representations. They can be fed directly with almost raw (size-normalized) images. Additionally, compared to traditional fully-connected multilayer networks, time for training and need for extensive datasets decreases since the weights of convolutional layers are much more constrained and thus easier to learn. Another huge advantage of these convolutional layers for image classification is its to some degree built-in translational and local distortion invariance due to applying the same weight configurations across the image. Furthermore, CNNs force the extraction of local features which is often desirable since pixels in an images that are spatial neighbors are often highly correlated. They enforce this local extraction by using only locally receptive fields. [9]

The convolution operation, which is eponymous for this type of networks, differs a bit from its definition in other fields of science. In neural networks a convolution usually signifies the weighted sum of all neurons of the previous layer that fall inside of the receptive field of a neuron of the convolutional layer. This can be realized easily as a matrix multiplication. As can be seen in figure 4, the weights of this operation are then shared throughout the whole image. Next, the output of the convolution operation is run through a non-linear activation function as described in section 2.2.

Afterwards, most CNNs usually apply a third step, the so-called pooling. The pooling operation is used to make the network invariant against small translations of objects in the input image as several locally nearby outputs are summarized into one. Figure 5 shows the max pooling as one example of pooling operations.

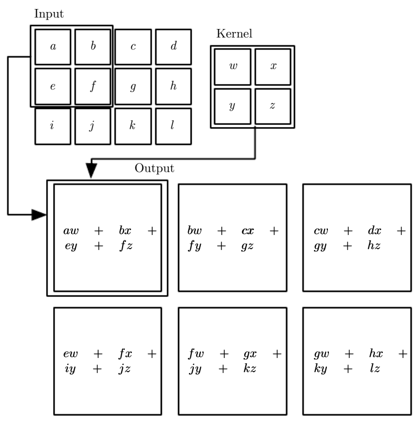

Figure 4: An example of the convolution operation in CNNs with a receptive field of size 2×2 and a stride of 2. Figure taken from [3, p.349].

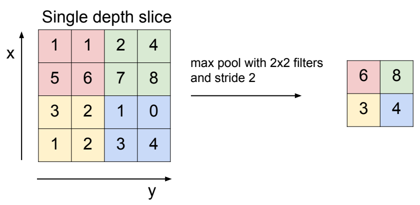

Figure 5: Visualization of a 2×2 max pooling operation with a stride of 2. Figure taken from [10, p.73].

3 Implementation

Both neural networks are implemented in Python version 3.6.7 and Tensorflow version 1.12.0. The training of the model is started via the 'main.py' file called without any arguments. Instead, the model as well as its hyperparameters can be specified in the configuration dictionary at the beginning of the script.
The program loads the data and performs a few preprocessing steps to normalize the range of the pixels of the input images and to create a training and a validation set with the according labels. Some functions are imported from the 'helper.py' file. The NN-models are stored in different classes and are built in the main file. They consist of different layers, the loss function and an accuracy measurement. Before the training of the model is started, the dataset is randomly shuffled and repeated several times in order to extend the training instances. Furthermore, an iterator is utilized to extract batches of samples during training. In each training step, one gradient descent step is performed and every 50 training steps evaluation measures take place. At the end of training the evaluation statistics are saved into a JSON file.

4 Tests

4.1 Performance of the 2-Layer Neural Network

As a base model, a simple neural network with one hidden layer is used. The input layer of the model takes each normalized pixel in the range from 0 to 1 of an image as input and outputs 10 values, one value for each class. At the final step the model calculates the probability of each output node with the softmax and calculates the cross-entropy loss. The weights of the model are initialized as a normal distribution with zero mean and standard deviation of 0.5. Values with more than two standard deviations from the mean are discarded and re-drawn. The activation function of both input and hidden layer are Rectified Linear Units (ReLU). The stochastic gradient decent is done with the Adam optimization algorithm to adapt the learning rate automatically. The model is trained with a batch size of 32 which should result in more stable updates of the model during gradient decent.

4.1.1 Experiment - Number of Nodes in the Hidden Layer

The main hyperparameter of this simple model is the number of nodes in the hidden layer. Different weight initializations do not show much effect as long as the weights are initialized randomly. We want to investigate the influence of the hidden units while we keep the other hyperparameters constant. The number of training steps is restricted to 15'000 because not much progress is observed after 12k steps.

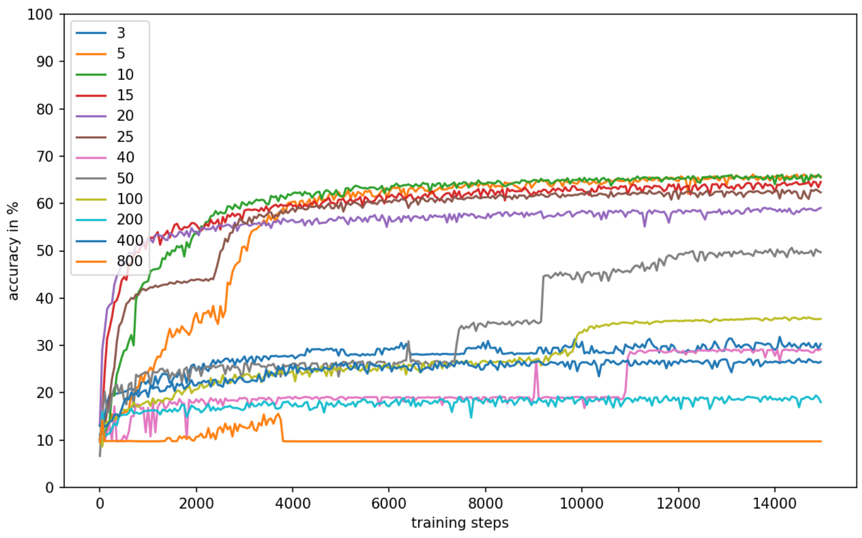

Figure 6: Accuracy after each 50 training steps on the validation set for different number of hidden units

Figures 6 and 7 show the improvements of the different models during training evaluated on the validation set. Figure6 shows the accuracy improvements of the model. Models with a small number of hidden nodes perform the best and the models with 5 nodes and 10 nodes achieve an accuracy of around 65%. Using less

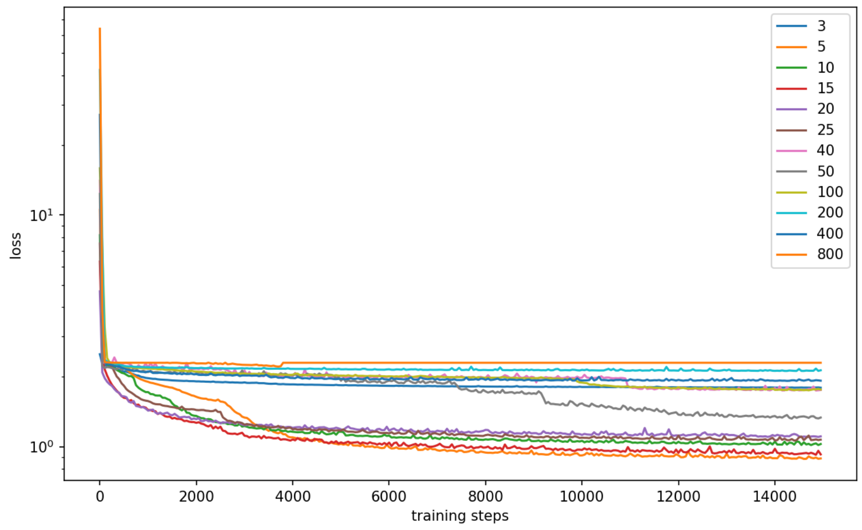

Figure 7: Loss after each 50 training steps on the validation set for different number of hidden units

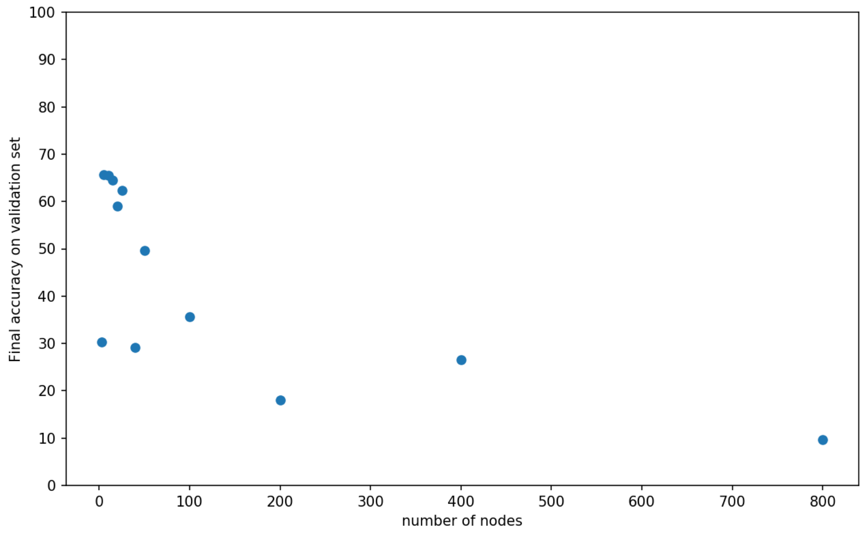

Figure 8: Final accuracy on the validation set in dependence of the number of nodes

nodes results in worse scores. For 3 nodes the accuracy drops to 30% and for 4 nodes to around 50%. Using more than 20 nodes also results in significantly worse results, though it is not clear if these models could further improve with enough training steps or with data augmentation due to their higher model capacity. It turns out that some models are "crashing" when we use a higher number of nodes. This happens to the 800 hidden neuron model before it reaches training step 4000. After that the model predicts totally random outputs and is not able to recover. Changing the learning rate of the Adam optimizer solves this problem. Therefore we changed the learning rate from 0.001 to 0.0001 for models with more than 50 nodes. Figure 7 shows the loss of the different models. The 5 hidden layer neuron neural network has the lowest loss. The final results are presented in figure 8. A model with 5 to 10 hidden nodes results in the best performance of the 2-layer NN after 15k training steps.

4.2 Performance of the Convolutional Neural Network

We first implement a simple convolutional neural network with the same configuration and hyperparameters as the 2-layer fully-connected neural network. Of course, the hidden layer is replaced by a convolutional layer. The layer uses filters to generate the activations. In the first experiment we vary the number of filters and keep the filter- or kernel size fixed. In the second experiment we vary the kernel size. In all runs we use a stride of one and no padding. This time we let the models run for 10k training steps.

4.2.1 Experiment - Number of Filters in the Convolutional Layer

First, we can observe that the performance of our model is much higher compared to the 2-layer base model (figure 9). The top score is 88% for a CNN with 15 filters. However, the number of filters does not have a big influence on the performance of our model. All models achieve scores of over 80% even if only one filter is used.

4.2.2 Experiment - Size of Filters in the Convolutional Layer

Reducing the filter size helps to improve the performance slightly (figure 10). By using a smaller filter size, smaller features of the image are preserved and more parameters are introduced to the network, which increases the capacity of the model, but it is also computational more expensive.

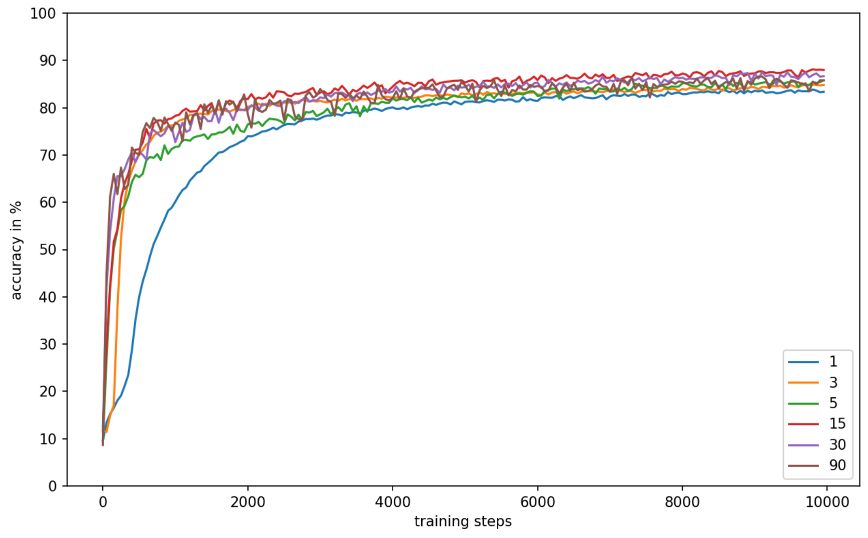

Figure 9: Accuracy after each 50 training steps on the validation set for different number of filters

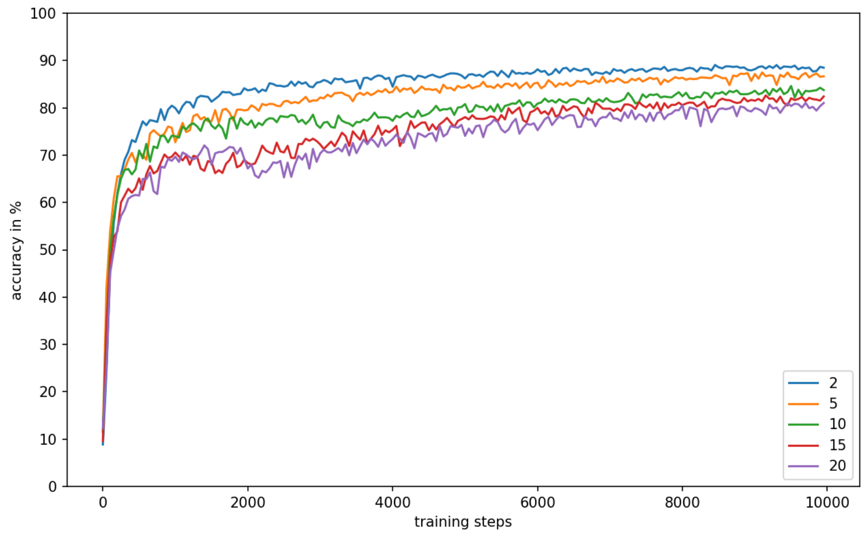

Figure 10: Accuracy after each 50 training steps on the validation set for different size of the filters. The number of filters was 32 in all runs.

5 Discussion

The tests demonstrate why a CNN is preferable over a dense model in image recognition. The scores of the CNNs are persistently higher even with suboptimal tuned hyperparameters. The dense model also shows more instabilities especially when a higher number of hidden neurons is used. It is also noteable that the performance of the dense NN sometimes suddenly improves and then stays on a plateau for a while. For the dense NNs our results show that for a 2-layer network it is advisable to not take to many hidden units. Although the optimal amount of hidden units depends on the amount of units in the input and output layers. Beside the amount of hidden neurons, the learning rate plays a critical role in the training phase of the network. The test for the CNN reveal less instabilities and all tested models perform well. We assume that the influence of the amount of filters and their size is more important for deeper neural networks because more information has to be transferred into the deeper layers. In general we conclude that a smaller filter size can improve the results slightly but comes with a higher computational cost For the amount of filters, it seems that a higher number gets better scores faster. If we use less filters the network holds less parameters and probably needs more time to efficiently tune them to reach good results.
Finally, we can obtain the performance of the two models by evaluating the best configuration of them on the test set. The dense model with 5 hidden neurons has an accuracy of 0.659 and the CNN models with 32 filters and a filter size of two has a score of 0.890. These scores are even slightly better than the scores of the validation set because the test set is bigger and not unbalanced like the validation set.
Zalando Research [11] provides an automatic benchmarking system based on scikit-learn that covers 129 classifiers with different parameters but no deep learning. The best classifier is a support vector machine with a polynomial kernel which achieves a score of 0.897. They also provide a table with not validated deep learning classifiers, in which the best classifiers reach scores of around 95%. This confirms that our models are working properly and obtain excellent results for their simplicity.

6 Future work

Even though our final model already performs well on the dataset there is still room for improvement. First, the dataset could be augmented by introducing random noise to images, changing luminance and contrast. This would probably lead to a more robust classifier. The next step would be to extend the CNN model with more layers. Usually several convolutional layers are stacked on top of each other, so that the first layers learn to detect edges and deeper layers learn more complex features.

At the end of the network dense layers are widely used to make the final prediction [9]. With increasing size of the network, pooling and other regularization techniques could be implemented to prevent overfitting. Lastly, the classifier should be applied to real-life data verifying its capabilities.

7 Conclusion

The success of a neural network depends a lot on the architecture and the hyperparameters of the model. Therefore, it is important to understand the influence of each hyperparameter on the network. In this work, we have shown the advances of using CNNs over dense models in image recognition and the influence of the hyperparameters for a model.
The best configured dense model with 5 hidden neurons achieved an accuracy of 0.659 on the test set. The CNN models achieved much better scores, whereat a convolutional model with 32 filters and a filter size of two achieved the best score of 0.890. These scores are pretty remarkable if we consider that the training of these networks has been done in a few minutes. Compared to non-deep-learning classifiers we perform better or at least equally. The best submitted deep learning benchmarks are around 0.96 due to a more extensive architecture.

References

[1] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms, 2017.

[2] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics).* Springer-Verlag, Berlin, Heidelberg, 2006. ISBN 0387310738.

[3] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning.* MIT Press, 2016. `http://www.deeplearningbook.org`.

[4] Vijay Kumar Sutariya Srinivas Tipparaju Wilfrido Moreno Munish Puri, Yashwant Pathak. *Artificial Neural Network for Drug Design, Delivery and Disposition.* Elsevier, 2016. doi: https://doi.org/10.1016/C2014-0-00253-5.

[5] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL `http://arxiv.org/abs/1412.6980`.

[6] Corrado G. S. Monga R. Chen K. Devin M. Le Dean, J. Large scale distributed deep networks. *Neural Information Processing Systems, 1–11.*, 2012. URL `http://papers.nips.cc/paper/4687-large-scale-distributed-deep-networks.pdf`.

[7] Kevin Swersky Geoffrey Hinton, Nitish Srivastava. Neural networks for machine learning. Computer Science University of Toronto, 2014.

[8] 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*, 1(4):541–551, 1989. doi: 10.1162/neco.1989.1.4.541. URL `https://doi.org/10.1162/neco.1989.1.4.541`.

[9] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov 1998. ISSN 0018-9219. doi: 10.1109/5.726791.

[10] Fei-Fei Li, Justin Johnson, and Serena Yeung. CS231n: Convolutional Neural Networks for Visual Recognition [Course slides, lecture 5], 2018. URL `http://cs231n.stanford.edu/slides/2018/cs231n_2018_lecture05.pdf`.

[11] Han Xiao. Fashion MNIST. https://github.com/zalandoresearch/fashion-mnist, 2017.