Deep Learning for Artificial Intelligence Engineering

Assignment 2

Convolutional Neural Networks

Student:
Amer Delić (01331672)

Lecturers:
Ozan Özdenizci
Simon Hitzginger

Institute of Machine Learning and Neural Computation
Graz University of Technology

Graz, December 2025

Contents

List of Figures

1 Convolutional neural networks

The main goal and detailed description of this assignment are provided in the attached assignment sheet. This report focuses on presenting the implementation, experimentation, and results corresponding to those tasks.

1.1 Dataset preprocessing

After loading the Fashion-MNIST datasets, the original training set was further split into a training subset (50 000 samples) and a validation subset (10 000 samples). The validation set was then used for model selection, while the test set was held out for final evaluation.

1.2 CNN design

For the classification task, a customized convolutional neural network (CNN) was designed and trained using the Adam optimizer. The architecture included multiple convolutional layers followed by ReLU activation functions and max pooling layers with a kernel size of $(2, 2)$ and stride 2. After the convolutional part, network contained two fully connected layers with 120 and 84 neurons respectively, following the classical LeNet design, and a final fully connected output layer with 10 neurons — one for each class.
CrossEntropyLoss was selected as the loss function since it is well-suited for multi-class classification tasks and provides a direct measure of the discrepancy between predicted class probabilities and true labels. It penalizes incorrect predictions more heavily when the model is confident but wrong, which guides the network to output probabilities that are more reliable and realistic. In this way, the model not only learns to classify correctly but also to provide probability estimates that better reflect its actual certainty.
To identify an effective CNN architecture for this classification task, five different configurations were trained and evaluated. Starting with a simple network containing a single convolutional layer, the depth and width of the architectures were systematically increased by varying the number of convolutional layers and kernels. The Adam optimizer was configured with the default values for exponential decay rates for moment estimates ($\beta_1 = 0.9$, $\beta_2 = 0.999$), while the learning rate was varied. Furthermore, an early stopping with a patience of 5 was applied based on validation loss to prevent overfitting, and the best-performing model on the validation set was saved for final evaluation. Finally, training and validation losses, as well as the number of trainable parameters for each configuration were compared, and the results are summarized in Table 1.1.
As can be observed, all five configurations achieved very similar performance in terms of validation loss. Considering both validation performance and the trade-off between model complexity and generalization, the third configuration with two convolutional layers ($[32, 64]$) was identified as the optimal baseline for further investigations.

Configuration	Convolutional Layers	Kernels per Layers	Trainable Parameters	Learning Rate	Train Loss	Validation Loss
1	[16]	[(3,3)]	335 774	1e-4	0.1700	0.2482
2	[16, 32]	[(3,3), (3,3)]	111 934	5e-4	0.1879	0.2458
3	[32, 64]	[(3,3), (3,3), (3,3)]	221 950	3e-4	0.1823	0.2383
4	[16, 32, 64]	[(3,3), (3,3), (3,3)]	42 110	1e-3	0.2257	0.3065
5	[32, 64, 64]	[(5,5), (3,3), (3,3)]	75 070	1e-3	0.2483	0.2845

Table 1.1: Comparison of different configurations

1.3 Parameter analysis

In the following calculations, n_{w_i} and n_{b_i} denote the number of weight and bias parameters in the i-th layer respectively. Furthermore, $C_{\text{in},i}$ and $C_{\text{out},i}$ denote the number of input and output channels of the i-th convolutional layer, and K_i denote the kernel size (height $\times$ width). For fully connected layers, $N_{\text{in},i}$ and $N_{\text{out},i}$ denote the number of input and output neurons.
First convolutional layer ($C_{\text{in},1} = 1$, $C_{\text{out},1} = 32$, $K_1 = 3 \times 3$):

$$n_{w_1} = C_{\text{out},1} \times C_{\text{in},1} \times K_1 = 32 \times 1 \times 3 \times 3 = 288, \quad n_{b_1} = C_{\text{out},1} = 32$$

Shape after the first convolutional and pooling layer:

$$1 \times 28 \times 28 \to 32 \times 26 \times 26 \to 32 \times 13 \times 13$$

Second convolutional layer ($C_{\text{in},2} = 32$, $C_{\text{out},2} = 64$, $K_2 = 3 \times 3$):

$$n_{w_2} = C_{\text{out},2} \times C_{\text{in},2} \times K_2 = 64 \times 32 \times 3 \times 3 = 18\ 432, \quad n_{b_2} = C_{\text{out},2} = 64$$

Shape after the second convolutional and pooling layer:

$$32 \times 13 \times 13 \to 64 \times 11 \times 11 \to 64 \times 5 \times 5$$

Total number of neurons after flattening:

$$N_{\text{in},1} = 64 \times 5 \times 5 = 1600$$

First fully connected layer ($N_{\text{in},1} = 1600$, $N_{\text{out},1} = 120$):

$$n_{w_3} = N_{\text{in},1} \cdot N_{\text{out},1} = 1600 \times 120 = 192\ 000, \quad n_{b_3} = N_{\text{out},1} = 120$$

Second fully connected layer ($N_{\text{in},2} = 120$, $N_{\text{out},2} = 84$):

$$n_{w_4} = N_{\text{in},2} \cdot N_{\text{out},2} = 120 \times 84 = 10\ 080, \quad n_{b_4} = N_{\text{out},2} = 84$$

Output layer ($N_{\text{in},3} = 84$, $N_{\text{out},3} = 10$):

$$n_{w_5} = N_{\text{in},3} \cdot N_{\text{out},3} = 84 \times 10 = 840, \quad n_{b_5} = N_{\text{out},3} = 10$$

Total parameters:

$$N_{\text{total}} = \sum_{i=1}^{L=5} (n_{w_i} + n_{b_i}) = 221\ 950$$

where L denotes the total number of layers in the network.
The majority of parameters are concentrated in the first fully connected layer, which results from the large number of input neurons after the flattening operation. This concentration of parameters highlights one of the key advantages of convolutional neural networks - convolutional layers rely on local connectivity and weight sharing, which significantly reduces the

number of parameters compared to fully connected layers (e.g. 192 120 parameters in the first fully connected layer compared to only 18 496 in the second convolutional layer and 320 in the first convolutional layer). As a result, CNNs can efficiently learn spatial features while maintaining lower memory and computational requirements.

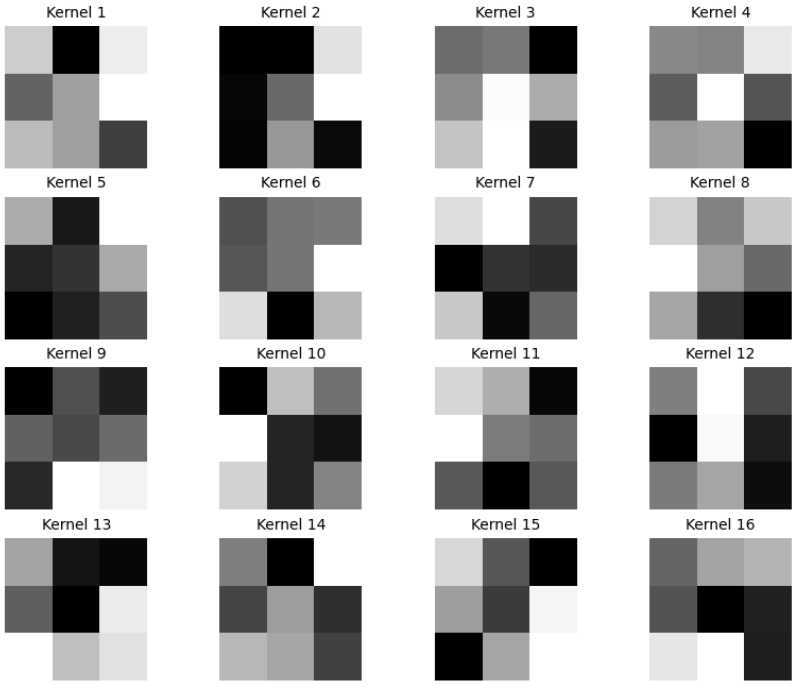

Figure 1.1: Kernels visualization

The visualization of the first convolutional layer, shown in Figure 1.1, shows that several kernels act as edge detectors (e.g., kernels 3 and 15), while some look very similar and might appear redundant (e.g. kernels 8 and 11), and some have near-uniform values and are almost inactive (e.g. kernel 14). In addition, a few kernels exhibit irregular patterns (kernels 5 and 9). Overall, the layer captures a diverse set of low-level features, but also includes some overlapping and weak filters, which is typical for early CNN layers.

1.4 Model regularization and data augmentation

To investigate the impact of $l2$ regularization, three different approaches were explored - manual $l2$ regularization added directly to the loss function, weight decay and decoupled weight decay within the Adam optimizer. All experiments were conducted following the same training procedure as before - using the Adam optimizer and including early stopping with a patience of 5 and saving the best performing model based on validation loss. Furthermore, dropout was also tested by inserting dropout layers into the fully connected part of the network and evaluating models with different dropout rates. The results were summarized and compared in Table 1.2 and Table 1.3 showing training and validation losses along with

accuracies across all tested approaches.

Approach	λ	Train Loss	Validation Loss	Train Error	Validation Error	Train Accuracy	Validation Accuracy
No regularization (baseline)	0	0.1823	0.2383	6.5%	8.5%	93.5%	91.5%
Manual $l2$ (loss term)	1e-3	0.3090	0.2303	6.75%	8.4%	93.25%	91.6%
Manual $l2$ (loss term)	1e-4	0.2497	0.2553	8.15%	9.2%	91.85%	90.8%
Manual $l2$ (loss term)	1e-5	0.1878	0.2330	6.8%	8.3%	93.2%	91.7%
Weight Decay (Adam)	1e-3	0.1958	0.2370	7.1%	8.2%	92.9%	91.8%
Weight Decay (Adam)	1e-4	0.1473	0.2305	4.5%	8.2%	95.5%	91.8%
Weight Decay (Adam)	1e-5	0.1680	0.2381	6.1%	8.55%	93.9%	91.45%
Decoupled Weight Decay (Adam)	1e-3	0.2018	0.2434	7.5%	8.9%	92.5%	91.1%
Decoupled Weight Decay (Adam)	1e-4	0.1602	0.2310	5.9%	8.3%	94.1%	91.7%
Decoupled Weight Decay (Adam)	1e-5	0.1752	0.2443	6.4%	8.9%	93.6%	91.1%

Table 1.2: Comparison of different $l2$ regularization approaches

As can be observed, all regularization approaches achieve similar performance and not a significant improvement compared to the baseline model without regularization. This suggests that the baseline model is already well-regularized — likely due to the simplicity of the Fashion-MNIST dataset and the fact that early stopping with saving the best performing model was used during training, which already helps avoid strong overfitting. In line with expectation, the results show that the regularization approaches, including optimizer-based weight decay, provide stronger optimization performance compared to the standard $l2$ regularization approach and adding the loss term directly to the loss function. Adam's use of gradient history and adaptive scaling causes $l2$ penalties to be inconsistently applied, while integrated (decoupled) weight decay ensures more stable and uniform reduction of weights.
Overall, the weight decays integrated into the Adam optimizer at $\lambda = 1e\text{-}4$ yield the lowest validation loss and highest validation accuracy. However, due to its slightly higher training loss and lower training accuracy — which reflects a lower tendency to overfit — the decoupled weight decay with $\lambda = 1e\text{-}4$ emerges as the most balanced option.

Configuration	p	Train Loss	Validation Loss	Train Error	Validation Error	Train Accuracy	Validation Accuracy
1 (baseline)	0	0.1823	0.2383	6.5%	8.5%	93.5%	91.5%
2	0.2	0.1662	0.2239	6.1%	7.6%	93.9%	92.4%
3	0.3	0.1954	0.2328	7.1%	8.2%	92.9%	91.7%
4	0.5	0.2132	0.2403	7.7%	8%	92.3%	92%

Table 1.3: Comparison of different dropout configurations

A similar observation can be made for the dropout strategy - while different configurations yield comparable results overall, the setup with $p = 0.2$ emerges as the most effective option, providing the strongest generalization while maintaining high training accuracy.

Finally, to improve model generalization, several different data augmentation strategies were explored. Since the Fashion-MNIST dataset is relatively small and consists of simple grayscale images, data augmentation is particularly relevant and helps the model learn more robust features. The used augmentation pipelines combine geometric transformations such as rotations, flips, translations, and zooming with photometric adjustments like brightness and contrast changes, as well as distortion. Geometric transformations shall make the model more robust to rotations, shifts, and size changes, which often occur in practical applications of this classification task. Color and brightness adjustments imitate different lighting and contrast conditions and shall help the model focus on shapes and textures instead of raw pixel values. Distortion methods like random erasing simulate partial blocking or folds, so the model learns to handle partially visible items. Each augmentation setup was applied during training and evaluated on the validation set to investigate its impact on performance. The results are summarized in the Table 1.4, which shows training and validation set errors along with accuracies for different data augmentation pipelines.
The results show that different transformation setups yield comparable validation performance

Augmentation Setup	Transformation	Train Loss	Validation Loss	Train Error	Validation Error	Train Accuracy	Validation Accuracy
1 (baseline)	-	0.1823	0.2383	6.5%	8.5%	93.5%	91.5%
2	Horizontal Flip, Rotation, Random Crop	0.1564	0.2303	5.7%	8.1%	94.3%	91.9%
3	Rotation, Brightness/Contrast Jitter, Random Erasing	0.1895	0.2447	6.9%	8.6%	93.1%	91.4%
4	Random Resized Crop (zoom in/out), Affine Translation, Horizontal Flip	0.1354	0.2272	5%	8%	95%	92%

Table 1.4: Comparison of different data augmentation approaches

overall, with only moderate improvements compared to the baseline model. While some augmentation setups improve generalization, the results show that not all configurations outperform the baseline model, highlighting that augmentation must be carefully selected to yield meaningful benefits.

1.5 Final model and evaluation

As identified in the previous chapter , the best-performing architecture from chapter 1.2, expanded with dropout layers ($p = 0.2$) in the fully connected part of the network, proved to be the most effective option. This configuration provided the strongest generalization and was therefore selected as the final CNN architecture for this classification task. The final model architecture is summarized in detail in Table 1.5.

Layer	**Specifications**	**Output Dimensionality**
Input	Grayscale image	$1 \times 28 \times 28$
Convolutional Layer 1	Conv2d (32 filters, 3×3, stride=1, pad=1) + ReLU + MaxPool (2×2)	$32 \times 13 \times 13$
Convolutional Layer 2	Conv2d (64 filters, 3×3, stride=1, pad=1) + ReLU + MaxPool (2×2)	$64 \times 5 \times 5$
Fully Connected Layer 1	Linear ($1600 \rightarrow 120$) + ReLU + Dropout ($p = 0.2$)	120
Fully Connected Layer 2	Linear ($120 \rightarrow 84$) + ReLU + Dropout ($p = 0.2$)	84
Output Layer	Linear ($84 \rightarrow 10$ classes)	10 classes

Table 1.5: Final CNN architecture summary

Figure 1.2: Kernels visualization

As outlined in chapter 1.2, the training procedure used CrossEntropyLoss as the objective function together with the Adam optimizer at a learning rate of 0.0003. A batch size of 64

was used, and the model was trained for up to 100 epochs. An early stopping strategy based on validation loss with a patience of 5 epochs was applied, ensuring that the best performing model was saved during training. The evolution of the training and validation error during training of the final CNN architecture are shown in Figure 1.2. The lowest validation error was reached at epoch 19.
Lastly, the final model was trained using the full training dataset. The final test metrics as well as the confusion matrix of the test set predictions are shown in Table 1.6 and Figure 1.3 respectively. Compared to the validation performance during training, the final test results remained consistent, indicating that the model did not overfit when exposed to the full training set. Furthermore, as expected, the most frequently confused classes were T-Shirt and Shirt, Shirt and Coat, as well as Shirt and Pullover.

Test Loss	Test Error	Test Accuracy
0.2545	8.7%	91.3%%

Table 1.6: Final model evaluation

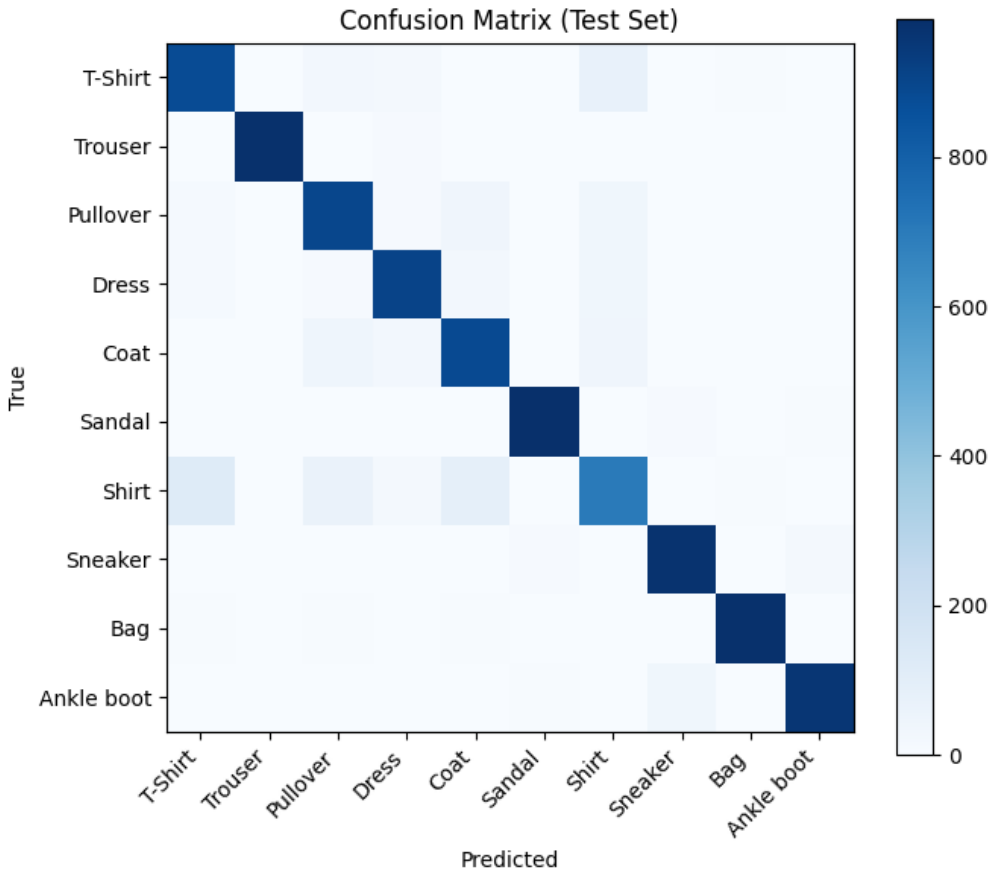

Figure 1.3: Confusion matrix

Appendix

Deep Learning for AIE (708.002) WS25
Assignment 2

Convolutional Neural Networks

Ozan Özdenizci, Simon Hitzginger
Institute of Machine Learning and Neural Computation, TU Graz
oezdenizci@tugraz.at, simon.hitzginger@tugraz.at

Points to achieve:	15 pts
Assignment Issued:	20.11.2025 16:00
Deadline:	18.12.2025 16:00 (**no extensions**)
Hand-in procedure:	You **can** work in groups of **at most two people**. **Exactly one** team member uploads two files to TeachCenter: **The report (.pdf)** and **the Python code as .py or .ipynb**. **Do not upload a folder. Do not zip the files.** The first page of the report **must** indicate the group partner.
Assignment interviews:	You might be invited for an assignment interview.
Plagiarism:	If detected, 0 points for all parties involved.

We will use PyTorch to train a CNN for the Fashion-MNIST image classification dataset. This dataset originally consists of 60,000 training and 10,000 test images of size 28x28 in grayscale. You can load the dataset by using the `torchvision` library. Your task will be to construct a CNN that achieves high performance on the Fashion-MNIST test set.

Task details:

For all the tasks below, create the appropriate code and discuss your experimentation and reasoning process, findings and choices in your report. **All results should be shown in the PDF report. The submitted code should be able to reproduce your results**.

a) (1 pts) : Load and prepare your data and labels. Familiarize yourself with the dataset and problem. Construct a *validation set* consisting of samples from the training data, which will be used during the model selection process. You will use the *test set* only to evaluate the final model (i.e., *after* model selection).

b) (5 pts) : Design and train your customized CNN for this multi-class classification task, that achieves good performance on the grayscale Fashion-MNIST test set. Explain your design choices for the output layer and the loss function that you will use during training. Minimize the training loss using Adam, and use early stopping based on the validation set to avoid overfitting. Test at least 5 different architectures with varying numbers of convolutional layers and number of kernels (i.e., change the architecture in depth and width). Provide a table comparing training and validation losses, the number of trainable parameters, and the learning rate used for the different architectures.

c) (2.5 pts) : For the best architecture from part (b), verify the calculation of parameters by hand. Write down and add up the number parameters (weights and biases) for each layer. Clearly

state where all the used values come from. Which layer contains most of the parameters? How does this highlight one of the main benefits of CNNs? Also visualize all the convolutional kernels of the first layer of your model. What do you observe (e.g., duplicate channels, dead or uniform channels, specific feature detectors, ...)?

d) (4.5 pts) : For the best architecture from part (b), investigate and compare different l_2 regularization approaches, as well as dropout for regularization. You should use the validation set to find the appropriate regularization hyper-parameters (e.g., l_2 regularization weight, or a good dropout rate). Furthermore, try and discuss different data augmentation approaches to achieve better generalization of your model. Provide a table where training and validation set errors and accuracies of various regularization settings are compared.

e) (2 pts) : Clearly summarize your final model architecture in detail (i.e., convolutional and/or pooling kernel sizes and specifications, input and output dimensionalities per layer, activation functions). Report your training specifications regarding the loss function, optimization scheme and learning rate, batch size and number of training epochs. Provide a plot where the evolution of the training and validation error during training is shown throughout iterations. Perform a final training with the whole training set. Report the final test accuracy and a confusion matrix of your test set predictions. Which classes are confused most often with each other?