Dive into Deep Learning

xAI-Proj-B: Bachelor Project Explainable Machine Learning

Florian Gutbier[*]
Otto-Friedrich University of Bamberg
96047 Bamberg, Germany
XXX@stud.uni-bamberg.de

Marius Ludwig Bachmeier[†]
Otto-Friedrich University of Bamberg
96047 Bamberg, Germany
XXX@stud.uni-bamberg.de

Andreas Schwab[‡]
Otto-Friedrich University of Bamberg
96047 Bamberg, Germany
andreas-franz.schwab@stud.uni-bamberg.de

Abstract

//TODO: write abstract? Was muss da rein?

1 Introduction

The emergence of deep learning has led to unprecedented advances in various fields, including medical image analysis. This project seeks to explore the fundamental principles of deep learning and to leverage its potential in a practical setting. Our investigation is divided into two parts: First, we focus on the classification of handwritten digits using the MNIST dataset (Deng, 2012), followed by a more complicated challenge of classifying nine different tissue patterns within the PathMNIST dataset (Kather et al., 2018, 2019), a subset of MedMNIST (Yang et al., 2021). These tasks not only serve as a basis for understanding the mechanisms of deep learning, but also highlight the impact of the application of neural networks in medical diagnostics, emphasizing the relevance of our research.

The initial phase of our project was dedicated to learning the basics using the MNIST dataset, which was chosen for its many resources and tutorials to help us get started with deep learning. In this phase, we developed a basic convolutional neural network (SimpleCNN) to introduce us to the architectures of neural networks and their ability to classify different digits.

The transition to the MedMNIST dataset, in particular the PathMNIST subset, represented a significant increase in the complexity of our project. This phase was crucial as it enabled us to apply and refine advanced techniques, such as experimenting with pre-trained models, testing a wide range of hyperparameters, and exploring different strategies for data pre-processing and augmentation. The PathMNIST subset was chosen to emphasize the critical importance of neural networks in medical image analysis. By tackling the classification of tissue patterns, we have not only explored some technical intricacies of deep learning, but also contributed to an area where such technologies could potentially revolutionize diagnostic methods.

This report is structured by first presenting the datasets used - MNIST and MedMNIST - which form the basis for a deeper investigation. We then explain some theoretical foundations that are crucial for understanding the methods used, including dropout layers and ReLU. Next, we explain the architectures of different deep learning models that were investigated in the project, from our initial SimpleCNN to more complex models such as AlexNet, ResNet and Xception. Subsequently,

[*]Degree: B.Sc. AI, matriculation #: 12345678
[†]Degree: B.Sc. AI, matriculation #: 12345678
[‡]Degree: B.Sc. AI, matriculation #: 2017990

Figure 1: Example grayscale images of the MNIST dataset

Figure 2: Example images of the MedMNIST dataset.

we present the results obtained with each model, addressing the specific challenges. The discussion section provides a critical evaluation of our results and leads to a reflective conclusion about the lessons learned and the potential impact of our research on medical image analysis.

1.1 MNIST

The "modified National Institute of Standards and Technology" dataset comprises a collection of 70,000 handwritten digits carefully divided into a training set of 60,000 images and a test set of 10,000 images. Each digit is represented in a grayscale image of 28×28 pixels, and offers a wide range of styles and shapes. This dataset is widely recognized for its simplicity and effectiveness in benchmarking classification algorithms, making it an ideal starting point for those new to deep learning. Some examples of the dataset can be seeen in Figure 1

Chosen for our initial challenge, MNIST provided a fundamental platform to explore neural network basics and experiment with simple model architectures. It allowed us to grasp the essentials of model training, and hyperparameter testing.

1.2 MedMNIST

MedMNIST, a more specialized and challenging dataset than MNIST, is tailored for medical image classification tasks. It extends the concept of handwritten digit classification to a diverse range of medical imaging modalities, including dermatology or radiology. Unlike MNIST's uniform format, MedMNIST encompasses 12 subsets for 2D and 6 subsets for 3D data. For our project, we focused on the PathMNIST (Kather et al., 2018, 2019) subset, which includes "100.000 non-overlapping image patches from hematoxylin and eosin stained histological images, and a test dataset [...] of 7.180 image patches from a different clinical center" (Yang et al., 2021). The images could be classified into nine different types of tissues.

Initially, the images were of high resolution ($3 \times 224 \times 224$ pixels), but the authors of MedMNIST resized them to $3 \times 28 \times 28$ pixels. The 100.000 training images were then divided into training and validation sets in a 9:1 ratio. Some examples of the various images can be seen in Figure 2.

The PathMNIST subset of MedMNIST2D provides a unique challenge by introducing the complexity of medical image analysis. It requires the use of advanced deep learning techniques and models to accurately classify different types of tissue, making it an excellent progression from the simpler MNIST dataset.

2 Methods

2.1 Model Architectures

2.1.1 SimpleCNN

Our SimpleCNN model, designed as an initial exploration of deep learning, contains two primary convolutional layers. The choice of this architecture was motivated by the goal of understanding the basic mechanisms of neural networks in processing and classifying image data. The model uses LeakyReLU activation to avoid the vanishing gradient problem and to ensure effective backpropagation even at small gradient values. The first convolution layer uses 32 filters with a kernel size of 5×5, a stride of 1 and 'equal' padding, which preserves the dimension of the input images. This is followed by a max-pooling layer with a kernel size of 2, which aims to reduce the spatial dimensions while keeping the important features to optimize the model's ability to detect significant patterns without considerable data loss. Subsequently, a second convolution sequence increases the depth to 64 filters, improving the model's ability to extract more complex features. This setup is again followed by LeakyReLU activation and max-pooling, further refining the feature extraction process. The architecture concludes with a linear layer that maps the high-level features to the output classes, with a softmax function to interpret the outputs as class probabilities.

As our project progressed and our understanding deepened, we enhanced the initial SimpleCNN model by introducing an additional convolutional layer and incorporating batch normalization and dropout techniques, aiming for improved accuracy and generalization. The modified SimpleCNN architecture begins with a convolutional layer designed for single-channel (grayscale) images to match the format of the MNIST dataset. This layer, consisting of 32 filters with a kernel size of 3×3 and 'same' padding, is followed by batch normalization and ReLU activation, which promotes nonlinearity while maintaining normalization across the inputs of the network. A dropout rate of 0.25 after max pooling aims to mitigate overfitting by randomly omitting a portion of the features during the training process. As the model progresses, the second and third convolutional layers increase the filter count to 64 and then 128, respectively, each augmented with batch normalization, ReLU activation, max pooling, and a consistent dropout rate of 0.25. This architectural depth ensures the extraction of increasingly complex features essential for recognizing diverse patterns in handwritten digits. The concluding segment of the model comprises a fully connected layer transitioning from the convolutional output to 256 units, followed by batch normalization and a higher dropout rate of 0.5, further combating overfitting. The final linear layer maps these processed features to the ten class outputs corresponding to the MNIST dataset's digit categories.

The transition from the SimpleCNN model, which was tailored for the MNIST dataset, to the model customized for the PathMNIST challenge required some adjustments to account the different characteristics of the two datasets. The most important change was to adapt the input layer to include 3-channel RGB images for PathMNIST, as opposed to the single-channel configuration developed for MNIST's grayscale images. This change is important to take advantage of the color information that is critical in medical imaging for identifying different tissue types. Despite this adaptation, the core architecture - consisting of convolutional layers, batch normalization, ReLU activation, and dropout layers - remained consistent between models.

For a detailed implementation of all versions of the model, refer to the Appendix Section A.1.

2.1.2 ResNet

//TODO

Resnet18 //TODO

Resnet50 //TODO

ResnetXX //TODO

2.1.3 Xception

To understand the Xception model, one must become familiar with depth-separable convolutions. This technique is a convolution operation that divides a standard convolution into two different stages

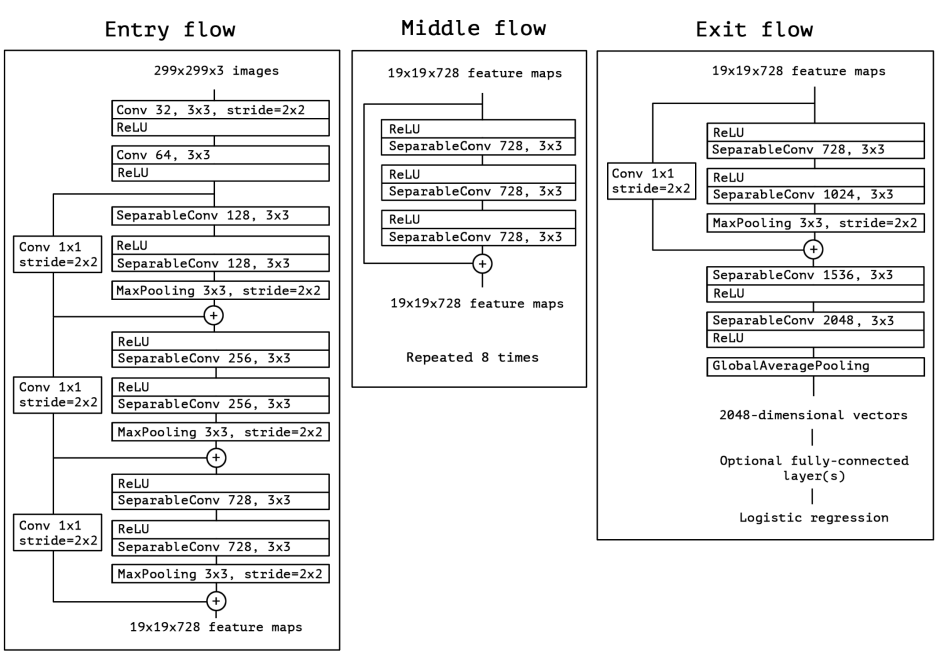

Figure 3: Xceptions architecture as outlined by Chollet.

to increase computational efficiency. Initially, a depth-wise convolution applies a single filter to each input channel. Subsequently, a point-wise convolution - characterized by a 1×1 kernel - combines the outputs from the depth-wise step across the channels. This factorization significantly reduces the number of parameters and calculations and enables more efficient training. Batch normalization follows each convolution and promotes stable learning by normalizing the ReLU activations of the layer.

As seen in Figure 3 the architecture of Xception, as detailed by (Chollet, 2017), is structured into three primary flows.

The entry flow prepares the network with initial convolutions and pooling to create condensed feature maps from the input images. It starts with two standard convolutions, followed by a series of separable convolutions that increase the depth while compressing spatial dimensions. This is achieved using a combination of 1×1 convolutions for channel-wise feature processing and 3×3 convolutions for capturing spatial information, each followed by max pooling to halve the feature map dimensions progressively. This step reduces the initial input image size from $299\times299\times3$ to $19\times19\times728$

Central to Xception's design is the middle flow, which repeatedly applies depthwise separable convolutions to process and refine the features. This part of the network is designed to be repeated eight times, allowing the model to learn increasingly complex patterns without a significant increase in computational cost. The dimensions do not get affected in this stage.

The exit flow then expands the feature maps through additional separable convolutions, incorporating a mix of channel-wise and spatial feature extraction before concluding with global average pooling. This step reduces each map to a single vector, capturing the essence of the input data in a form suitable for classification. The network concludes with a logistic regression layer, such as softmax, to output the final class probabilities.

3 Discussion

//TODO

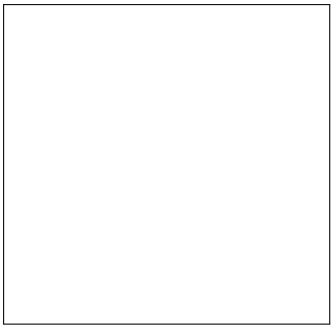

Figure 4: Sample figure caption.

Table 1: Sample table title

	Part	
Name	Description	Size (μm)
Dendrite	Input terminal	$\sim$100
Axon	Output terminal	$\sim$10
Soma	Cell body	up to 10^6

4 Conclusion

//TODO

References

F. Chollet. Xception: Deep learning with depthwise separable convolutions, 2017.

L. Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

J. N. Kather, N. Halama, and A. Marx. 100.000 histological images of human colorectal cancer and healthy tissue, 2018. URL `https://doi.org/10.5281/zenodo.1214456`.

J. N. Kather, J. Krisam, P. Charoentong, T. Luedde, E. Herpel, C.-A. Weis, T. Gaiser, A. Marx, N. A. Valous, D. Ferber, et al. Predicting survival from colorectal cancer histology slides using deep learning: A retrospective multicenter study. *PLoS Medicine*, 16(1):e1002730, 2019.

J. Yang, R. Shi, and B. Ni. Medmnist classification decathlon: A lightweight automl benchmark for medical image analysis. In *IEEE 18th International Symposium on Biomedical Imaging (ISBI)*, pages 191–195, 2021.

Declaration of Authorship

Ich erkläre hiermit gemäß § 9 Abs. 12 APO, dass ich die vorstehende Projektarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Des Weiteren erkläre ich, dass die digitale Fassung der gedruckten Ausfertigung der Projektarbeit ausnahmslos in Inhalt und Wortlaut entspricht und zur Kenntnis genommen wurde, dass diese digitale Fassung einer durch Software unterstützten, anonymisierten Prüfung auf Plagiate unterzogen werden kann.

Bamberg, February 26, 2024

(Place, Date)	(Signature)

Bamberg, February 26, 2024

(Place, Date)	(Signature)

Bamberg, February 26, 2024

(Place, Date)	(Signature)

A Appendix

A.1 SimpleCNN for MNIST

Initial version of our SimpleCNN, including two convulutional layers:

```
class SimpleCNN(nn.Module):
    def __init__(self, num_classes=10):
        super(SimpleCNN, self).__init__()
        self.conv1 = nn.Sequential(
            nn.Conv2d(
                in_channels = 1,
                out_channels = 32,
                kernel_size = 5,
                stride=1,
                padding="same"
            ),
            nn.LeakyReLU(),
            nn.MaxPool2d(kernel_size=2),
        )
        self.conv2 = nn.Sequential(
            nn.Conv2d(32,64,5,1,"same"),
            nn.LeakyReLU(),
            nn.MaxPool2d(kernel_size=2),
        )
        self.out = nn.Linear(64*7*7, num_classes)

     def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        x = x.view(-1, 64*7*7)
        output = self.out()
        return torch.log_softmax(output, dim=1)
```

Structure of the improved version of the SimpleCNN using three convolutional layers, Batch normalization and Dropout:

```
class SimpleCNN(nn.Module):
    def __init__(self, num_classes=10):
        super(SimpleCNN, self).__init__()
        self.conv1 = nn.Sequential(
            nn.Conv2d(1, 32, kernel_size=3, stride=1, padding="same"),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2),
            nn.Dropout(0.25)
        )
        self.conv2 = nn.Sequential(
            nn.Conv2d(32, 64, kernel_size=3, stride=1, padding="same"),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.MaxPool2d(2),
            nn.Dropout(0.25)
        )
        self.conv3 = nn.Sequential(
            nn.Conv2d(64, 128, kernel_size=3, stride=1, padding="same"),
            nn.BatchNorm2d(128),
            nn.ReLU(),
            nn.MaxPool2d(2),
            nn.Dropout(0.25)
        )
        self.fc1 = nn.Linear(128 * 3 * 3, 256)
        self.fc_bn = nn.BatchNorm1d(256)
        self.dropout_fc = nn.Dropout(0.5)
        self.fc2 = nn.Linear(256, num_classes)

    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        x = self.conv3(x)
        x = x.view(-1, 128 * 3 * 3)
        x = F.relu(self.fc_bn(self.fc1(x)))
        x = self.dropout_fc(x)
        x = self.fc2(x)
        return torch.log_softmax(x, dim=1)
```

Structure of the improved version of the SimpleCNN for the PathMNIST Dataset:

```
class SimpleCNN(nn.Module):
    def __init__(self, num_classes=10):
        super(SimpleCNN, self).__init__()
        self.conv1 = nn.Sequential(
            nn.Conv2d(3, 32, kernel_size=3, stride=1, padding="same"),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2),
            nn.Dropout(0.25)
        )
        self.conv2 = nn.Sequential(
            nn.Conv2d(32, 64, kernel_size=3, stride=1, padding="same"),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.MaxPool2d(2),
            nn.Dropout(0.25)
        )
        self.conv3 = nn.Sequential(
            nn.Conv2d(64, 128, kernel_size=3, stride=1, padding="same"),
            nn.BatchNorm2d(128),
            nn.ReLU(),
            nn.MaxPool2d(2),
            nn.Dropout(0.25)
        )
        self.fc1 = nn.Linear(128 * 3 * 3, 256)
        self.fc_bn = nn.BatchNorm1d(256)
        self.dropout_fc = nn.Dropout(0.5)
        self.fc2 = nn.Linear(256, num_classes)

    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        x = self.conv3(x)
        x = x.view(-1, 128 * 3 * 3)
        x = F.relu(self.fc_bn(self.fc1(x)))
        x = self.dropout_fc(x)
        x = self.fc2(x)
        return torch.log_softmax(x, dim=1)
```