Team OverxAIted

xAI-Proj-B: Bachelor Project Explainable Machine Learning

A Semester in the Life of a Deep Learning Engineer

Ole Dziewas[*]
Otto-Friedrich University of Bamberg
96049 Bamberg, Germany
ole-elija.dziewas@stud.uni-bamberg.de

Leopold Gierz[†]
Otto-Friedrich University of Bamberg
96049 Bamberg, Germany
leopold-heinrich.gierz@stud.uni-bamberg.de

Kian Hinke[‡]
Otto-Friedrich University of Bamberg
96049 Bamberg, Germany
kian-fried.hinke@stud.uni-bamberg.de

Sebastian Schick[§]
Otto-Friedrich University of Bamberg
96049 Bamberg, Germany
sebastian-michael.schick@stud.uni-bamberg.de

Abstract

The following report presents the results of the development of a street number recognition system using Google Colab, WandB and PyTorch. It will be shown that real image datasets are much more difficult to learn than scientific datasets such as MNIST. Different neural network architectures are explained and tested on the datasets, with ResNet34 being the best performing. The aim of the report is to show what neural models are capable of when trained from scratch with limited computational resources, and to explain the typical obstacles faced in developing these systems. All the code for this report is available on GitLab: `https://gitlab.studium.uni-bamberg.de/sebastian-michael.schick/xai-proj-b`.

1 Introduction - Kian Hinke

In the realm of machine learning, Convolutional Neural Networks (CNNs) have emerged as a popular tool for image classification tasks, proving effective in extracting intricate patterns and features from vast arrays of images with exceptional accuracy. The importance of CNN-based image classification is not only of an academical nature but holds profound real-world implications. From aiding medical

[*]Degree: B.Sc. AI, matriculation #: 2028300
[†]Degree: B.Sc. AI, matriculation #: 2042084
[‡]Degree: B.Sc. AI, matriculation #: 2042105
[§]Degree: B.Sc. AI, matriculation #: 2045546

diagnostics through precise image analysis to enhancing autonomous vehicle safety with accurate object detection, CNNs empower a multitude of applications across industries. Their ability to process and classify images at unprecedented scales and accuracies makes them indispensable in our increasingly digitized world, where data-driven decisions and automations are paramount.

1.1 Motivation

The task of single digit house number classification tackled by us during the semester is related to many similar problems with enormous practical use like street-sign recognition, essential to the development of autonomous vehicles. Throughout the first half of the semester, we also worked with the MNIST Dataset and delved into the field of handwriting recognition. This area is crucial for automating the interpretation of handwritten forms and streamlining the reading of addresses as used by postal services.

This project aimed to provide a practical insight into the tasks and steps faced by deep learning engineers. It exposed us to the processes, methods and challenges involved in researching, implementing, training, optimising and evaluating a CNN model. Through this, we not only gained theoretical knowledge but also an appreciation for the practical challenges and intricacies that come with real-world applications.

1.2 Related Work

The task of single digit housenumber classification through a CNN rests upon a foundation of complex technologies that have been shaped through several decades of rigorous research and development. In this section, we take a moment to review the pioneering work in CNNs and image classification, that have served as cornerstones in their development, eventually paving the way for our project.

1.2.1 Neocognitron and LeNet

The roots of modern CNNs can be closely linked to the pioneering work of Yann LeCun and his esteemed collaborators. While the concept of convolutional layers can be traced back to works such as Kunihiko Fukushima's Neocognitron [2] in the 1980s, it was LeCun's team that revolutionised their application. Their seminal 1989 paper "Backpropagation applied to handwritten zip code recognition" presented a remarkable application of the backpropagation technique to a convolutional network structure, refining it specifically for image classification tasks [6]. This innovation was further expanded in their 1998 paper "Gradient-based learning applied to document recognition", which provided a comprehensive introduction to the LeNet5 architecture. LeNet5, a cornerstone of subsequent neural network research, demonstrated its ability to recognise and classify handwritten digits using the then newly introduced MNIST dataset [7]. The continuing relevance of the MNIST dataset is underlined by its continued popularity, as evidenced by its inclusion in the initial phase of our project.

1.2.2 AlexNet

In 2012, the field of convolutional neural networks took a significant leap forward with the introduction of AlexNet, a collaborative effort by Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. Prior to its debut, the focus had been on traditional computer vision techniques, but the introduction of AlexNet marked a significant paradigm shift. AlexNet's architecture was both deep and complex. It consisted of multiple convolutional layers, utilised the groundbreaking ReLU activation function, and strategically incorporated dropout layers to mitigate overfitting problems. In addition, its design was enriched with a local response normalisation layer to improve the extraction of salient features. With 60 million parameters, the number of parameters in AlexNet was dramatically higher than the well-known LeNet5, which had 60,000 parameters. [5]

The profound impact was underlined when AlexNet significantly reduced the error rate in the ImageNet Large Scale Visual Recognition Challenge, also becoming the first CNN to ever win the competition 1. This achievement went beyond a mere numerical victory - it triggered a broader transformation in the research community.

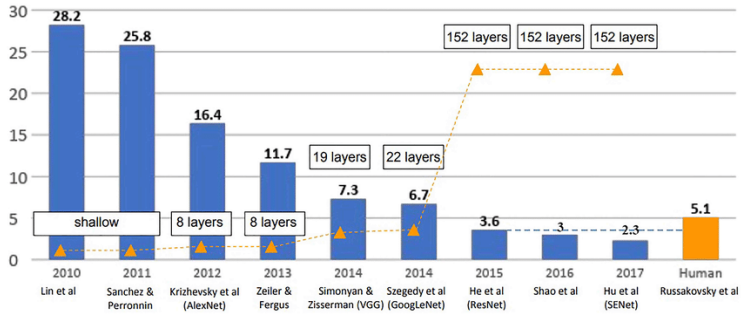

Figure 1: ImageNet Challenge Winners 2010-2017 [8]

AlexNet's success not only solidified the dominance of deep neural networks in image classification, but also set the stage for an era in which deep learning architectures inspired by its innovations would become the foundation of both academic research and real-world applications in image recognition.

1.2.3 VGG

In 2014, the deep learning community witnessed the unveiling of the VGG architecture, a groundbreaking development that represents another milestone in reducing the error rate in the ImageNet Challenge1. Originating from the Visual Geometry Group at Oxford, VGG marked a fresh approach to the design of deep neural networks.

Moving away from the diverse layer types that typified earlier architectures, VGG charted a distinctive course with a consistent and intentional design philosophy. At its core, the architecture committed to building networks using homogeneously stacked convolutional layers, each with small 3x3 filters. This commitment was more than just a design aesthetic; it resulted in the famous VGG-16 and VGG-19 models, each named after the number of layers. Delving deeper into this design reveals its genius: the methodology of stacking two 3x3 convolution layers was empirically proven to parallel a single 5x5 convolution layer. This discovery was not only about efficiency, but also implied a significant reduction in computational requirements. While it demonstrated the profound implications of network depth, highlighting its ability to recognise intricate image patterns with unparalleled precision, it also pushed the boundaries of classification performance, setting new benchmarks for subsequent architectures. But like many pioneering efforts, VGG wasn't without its challenges. Its dense and complex design, characterised by a huge number of parameters, posed difficulties for its seamless deployment, especially on devices constrained by computational resources. However, VGG's legacy remains undeniable, and it has set the stage for a new era of deep learning research and innovation in the broader context of the evolution of neural networks. [10]

1.2.4 Vanishing Gradients

In the wake of the explosive growth of deep learning, driven by models like VGG, the depth of neural architectures had increased, enabled in part by increasing computational power, allowing researchers to explore more complex and sophisticated models. However, this architectural expansion wasn't without its pitfalls. Deep networks began to face the emerging challenge of vanishing gradients.

Deep neural networks, especially as they expand in depth, face the problem of vanishing gradients. To understand this, it's important to first understand the nature of the backpropagation process in CNN training. In backpropagation, gradients of the loss function are computed and propagated backwards through the network, from the last layer back to the first. These gradients play a crucial role in determining how the weights of the neurons should be adjusted to minimise error.

Now, as we dive into deeper architectures, the gradients of loss as a function of network parameters can become extremely small, essentially vanishing. This phenomenon is particularly pronounced in the early layers of the network and is extremely problematic because when gradients are tiny or approach zero, the weight updates during training become negligible. This means that the weights of the neurons in these layers hardly change, rendering them almost static and essentially untrainable. As a result, these early layers fail to capture meaningful representations of the input data, severely limiting the network's ability to learn.

1.2.5 ResNet

This is where ResNet comes into the picture, a revolutionary architecture that was introduced in the greatly influetial 2015 paper “Deep residual learning for image recognition” by a dedicated team at Microsoft. Going beyond traditional boundaries, their innovative approach tackled the problem of gradients directly. Instead of simply stacking layers, ResNet used ’residual blocks’ equipped with ’skip connections’ or ’shortcuts’. These links essentially allowed gradient information to bypass certain layers and flow freely through the network. Such a strategy preserved the gradients by bypassing layers that might otherwise have obstructed their passage. [4]

Validating its groundbreaking approach, a deep variant of ResNet, with 152 layers, not only entered the ImageNet challenge but triumphed, showing a significant reduction in error rate1. ResNet’s remarkable success paved the way for subsequent deep learning efforts, confirming the benefits of depth while providing solutions to its inherent challenges.

1.2.6 House Number Classification

In the development of deep learning methods for digit recognition, and in the context of the project’s remit, two seminal papers stand out, both of which used the Street View House Numbers (SVHN) dataset. Sermanet and LeCun’s 2012 paper, "Convolutional Neural Networks applied to House Numbers Digit Classification"[9], laid the groundwork by introducing a multi-stage CNN architecture tailored to the SVHN dataset. Their approach focused on single digit classification from cropped images, shedding light on the hierarchical feature learning capabilities of CNNs.

A subsequent work in 2013 "Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks" [3] broadened the scope by targeting end-to-end recognition of multi-digit numbers in Street View images, highlighting the real-world challenges posed by such data. With regard to our project, in which we trained a CNN to classify single-digit house numbers, it’s worth mentioning these works because of their relevance to the field. Although they did not directly influence our methods, their historical significance and foundational methodologies provide context to the broader landscape of house number recognition using CNNs.

1.3 Contribution

In light of previous research, our project does not stand as a groundbreaking contribution to the field. Rather, its primary purpose was to provide us with an entry point into the field, allowing us to gain first-hand experience in developing, training and evaluating a deep learning model for image classification on a real-world example. Our endeavour was less about producing original research results and more about understanding the intricacies of the process. As beginners, this project was invaluable and allowed us to connect theoretical knowledge with practical application, especially within the constraints of limited resources. Through this experience we gained a comprehensive understanding of the challenges and nuances involved in the whole process.

2 Methods - Sebastian Schick

After reviewing the materials provided and familiarising ourselves with the topic of deep learning, our team decided to tackle the first part of the project, which was to build a CNN to classify the numbers in the MNIST dataset. We chose the Pytorch framework to do this. To meet the challenge of classifying the custom street number dataset, additional data augmentation and updating of the model architecture was required. Kindly take note of the following correction: Regrettably, there was a naming mix-up in our code regarding the validation set and the test set. To eliminate any potential confusion, the accurate names have been rectified in the subsequent explanations.

2.1 Number Classification on MNIST Dataset

2.1.1 Data Preparation

The MNIST dataset images were accessible via Kaggle, organized into two main folders: `train` and `test`. Within the `train` folder, there were images of handwritten digits, and alongside these images was a corresponding CSV file. This CSV file contained labels associated with the images,

indicating the actual numeric values. These labels were matched to their respective image files based on the unique `SampleID` of each image. For the images within the `test` folder there was no CSV file, since this was the part of the submission process. We decided to use a separate Jupyter notebook to manually preprocess the data. The script retrieves label information from the mentioned CSV file and stores it in a numpy array. Subsequently, it divides and stores the labels from the `train` folder, 50.000 labels for training and 10.000 for validation. Additionally, the image files are transformed into numpy arrays, with pixel values normalized from the original range of 0-255 to the range of -1 to 1. The data was splitted accordingly and then saved into numpy files named `train-inputs.npy` and `validation-inputs.npy`.

2.1.2 Model Architecture

Our second Jupyter Notebook contains the loading of the data, model architecture as well as the training algorithm. The implemented CNN architecture for the MNIST dataset consists of three main components: two sets of convolutional layers `conv_layers1` and `conv_layers2` followed by a fully connected layer `fc_layer` [1]. The model takes an input tensor representing images and processes them through the layers to produce a classification output. The convolutional layers are responsible for learning and extracting relevant features from the input images. The architecture employs a series of convolutional operations, each followed by batch normalization and a Rectified Linear Unit (ReLU) activation function. Dropout layers are integrated after each set to enhance generalization by randomly deactivating specific neurons during training, preventing overreliance on any particular feature. This promotes a more robust and effective model. The amount of dropout was set via the WandB sweep function. After the convolutional layers, the resulting feature maps are flattened into a 1-dimensional tensor and fed into a fully connected layer for classification of the digits 0 to 9.

2.1.3 Loading Data and Model Training

Before Training, the stored numpy-arrays of the labels and pictures are loaded and transformed into tensors. For efficient handling of the data, dataloaders are used. The DataLoader is a critical component in deep learning pipelines, especially when dealing with large datasets that cannot fit entirely into memory. It efficiently loads and processes data in small batches, facilitating model training by providing the required data in manageable chunks. In each training epoch, the training loss is initialized to 0 and the model is set to training mode, it processes batches of images and labels from the `train_loader`. For each batch, the optimizer's gradients are reset, and the model's predictions are obtained. The loss between the predictions and actual labels is computed using the specified criterion, and the gradients are backpropagated to update the model's weights. Then, the training loss for the epoch is computed. After the training phase of each epoch, the model is set to evaluation mode. The code then evaluates the model's performance on the validation dataset. It iterates through batches of validation images and labels from the `validation_loader`, computes the model's predictions, and compares them to the actual labels. The number of correct predictions is tallied, and the overall accuracy is calculated as a percentage. We also used the WandB sweep functionality here to choose the optimizer and number of epochs, as well as the learning rate. After learning, a simple script loads the images of the test set and generates a CSV file for the submission of the determined labels through our trained model.

2.2 Number Classification on Street Number Dataset

2.2.1 Data Preparation and Data Augmentation

For the street number dataset, preprocessing was done in a similar fashion, however due to the limited size of the dataset, we chose to augment the data. After loading, data augmentation is applied through a image transformation pipeline. The transformation consist of:

- Random affine transformation with translation by up to 20% in both directions.
- Random rotation by up to 45 degrees.
- Color jittering for brightness, contrast, saturation, and hue.
- Random cropping to a size of 256x256 pixels.
- A custom function adding noise to the picture.

An example can be seen in figure 2. After augmentation, the images were transformed to numpy-arrays and normalized to values from 0 to 1. The final step was splitting the data into training set and validation set and storing it.

Figure 2: Augmented Image

2.2.2 Model Architecture

As we achieved poor results with the model architecture we already used for MNIST, we implemented the popular networks ResNet34 and ResNet50. The basic building block of ResNet-34 is the residual block, which contains two convolutional layers. Each block also introduces a shortcut connection that bypasses one or more layers, helping to prevent the vanishing gradient problem. The architecture can be summarized as followed:

- Input Layer: Tensor with appropriate dimensions.
- Initial Convolution: A convolutional layer with a small kernel size to capture basic features.
- Stacked Residual Blocks: These blocks make up the majority of the architecture. Each residual block consists of two convolutional layers with a skip connection that adds the input of the block to the output. The number of residual blocks increases the depth of the network.
- Global Average Pooling: After the residual blocks, global average pooling is applied to reduce the spatial dimensions of the features.
- Fully Connected Layer: A fully connected layer with 9 output classes, hence street numbers do not contain "0".

ResNet50 is a more advanced variant of the ResNet architecture with 50 layers. It is designed to handle more complex tasks and introduces a "bottleneck" architecture in each residual block. This architecture reduces computational cost. Each bottleneck block consists of three convolutional layers: a 1x1 convolution for dimension reduction, a 3x3 convolution for feature extraction and another 1x1 convolution for dimension expansion.

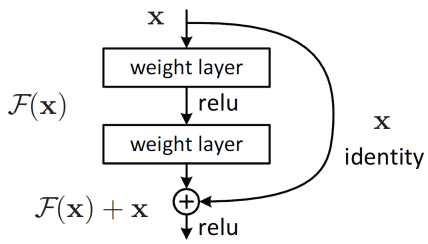

Figure 3: Residual Block [4]

2.2.3 Loading Data and Model Training

The algorithm for training remained the same. However, due to the lack of computational resources we were not able to use the sweep functionality from WandB.

3 Experiments and Results - Ole Dziewas

The following section will present the experimental results that were achieved during the course of the semester. For clarity and understanding purposes there will be a brief introduction on the datasets that were used as well as the overall experimental setup. After that, the experimental results on the different datasets as well as neural network architectures will be explained with the help of performance diagrams.

3.1 Experimental Setup

All of the experiments were done in Google Colaboratory and tracked by Weights and Biases. Google Colaboratory is a coding environment very similar to the well-known Jupyter notebooks. It works with executable code cells and text cells for a better understanding of the code.

Our code can be downloaded from our GitLab repository and then imported into Google Colab. The reason for using Google Colab was the free computing resources. The experiments were run on a T4 GPU and run with 12 GB of RAM. Due to those resources, as well as bad resource management, there were still a lot of limitations we were faced during the experiments, which will explained later on.

To track our results as well as hyperparameters and neural network weights, we used the open-source API Weights and Biases (WandB). It is not only to able to track these values, but also to visualise them. All of the following diagrams were therefore created by WandB. Especially useful was also the so called hyperparameter sweep, a feature of WandB for hyperparameter optimization, which we could only use on the MNIST dataset due to computational limitations.

3.2 Datasets

The experiments were performed on two datasets: The MNIST dataset and a handmade street number dataset. Although both represent single-digit numbers, they are very different in many ways. The MNIST dataset (Modified National Institute of Standards and Technology database) is a scientific dataset of handwritten numbers often used for benchmarks in the field of deep learning. It consists of over 70.000 images, which are split into a training set of size 60.000 and a test set of size 10.000.

Due to its size, it is a perfect dataset for testing different neural network architectures but not really comparable to our handmade dataset which only contains 714 images. The 714 images then again had to be split into a test set of size 144 and a training set of size 570. Despite being a much smaller dataset the images themselves contained much more information than the MNIST ones. While the MNIST images were black and white and of size 28 by 28 pixels, the street numbers were coloured images of size 256 by 256 pixels. Example images can be seen in figure 4.

Figure 4: Left - MNIST (upscaled), Right - Street Number

The street numbers were also real photographs and therefore full of noise and much more context, such as cars or houses, which were often also part of the image itself. Overall the task of running a recognition system on the street number dataset was much more difficult than on the MNIST dataset.

3.3 Results on MNIST

The first experiments were performed on the MNIST dataset. Due to the learnability, even small CNNs with two to three layers achieved accuracies of 0.95 and above. Before starting the hyperparameter optimization, we chose an architecture based on the kaggle article by Chris Deotte, a data scientist at NVIDIA.

Therefore we were able to focus on the optimization of hyperparameters, for which we used the WandB sweep feature. Such a sweep is shown in figure 5.

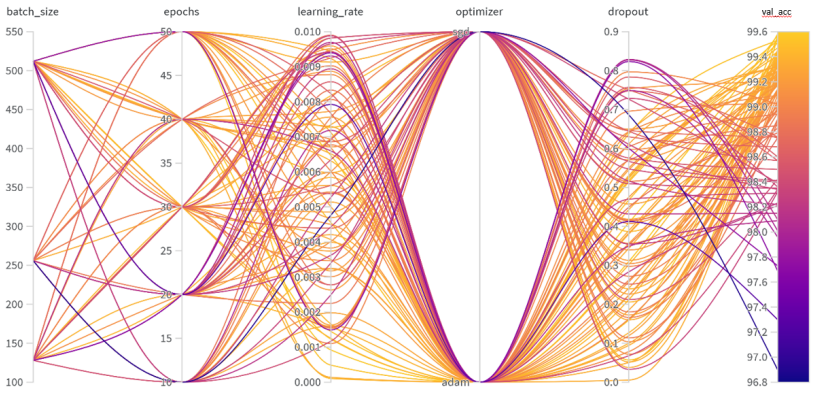

Figure 5: WandB sweep over MNIST dataset

It consists of 130 runs each with its own configuration of hyperparameters like batch size, epochs, learning rate, optimiser or dropout. To be seen is a colouring based on the performance of each run, in this case measured with the validation accuracy. Yellow lines are good performing runs, while blue ones perform bad. The more yellow lines there are at a certain value of a hyperparameter, the greater the positive influence on the actual result. For example low learning rates at around 0.001 seem to perform very well while high dropout values at around 0.7 and higher perform bad.

We then used the network which achieved the best run of this sweep as the detection system for our final kaggle submission. After only three epochs it achieved validation accuracies of 0.99 and after 40 epochs of training a final test accuracy of 0.9979.

3.4 Results on Street Numbers

After completing the experiments on the MNIST dataset we started working on our own handmade dataset of street numbers. Due to the larger complexity of the images and no data pre-processing, the training of the different networks took much longer than on the MNIST dataset. For comparison, the entire sweep of 130 runs on the MNIST dataset took two hours, which was the same amount as a single run on the street numbers.

Therefore, we were not able to perform a sweep on the street number dataset, not allowing a real hyperparameter optimisation. Instead, we focused on experimenting with single runs on different architectures. We decided not to use any pre-trained models to increase the learning effect for our team, but trained every network from scratch. As network architectures we chose large CNNs similar to VGG16 as well as the original ResNet34 and ResNet50 builds. For every run, we took the same hyperparameters that also worked well on the MNIST dataset. Every run took 200 epochs of training with a batch size of 32 and a learning rate of 0.0027 and a dropout of 0.4 for the CNNs builds.

3.4.1 Tests on CNN

For reasons of comparison we first tried to use the same architecture as in our MNIST experiments, with only a few adjustments to the new input size. But as to be expected the 570 images we had for training were not enough to actually learn such complex features and there was nearly no learning effect as to be seen in figure 6. This problem is not only a result of the small dataset but also the architecture itself, which probably led to the problem called vanishing gradients. The validation accuracy was consistently close to 0.12 and therefore close to just guessing the number.

Even after increasing the dataset to 7000 images with different augmentation techniques, there was not really a learning to be seen (figure 7). Although the validation accuracy was higher with 0.22, that was probably due to a mistake during the data augmentation phase. Instead of first splitting the dataset into training and validation set, we first did augmentation, leading to a lot of augmented

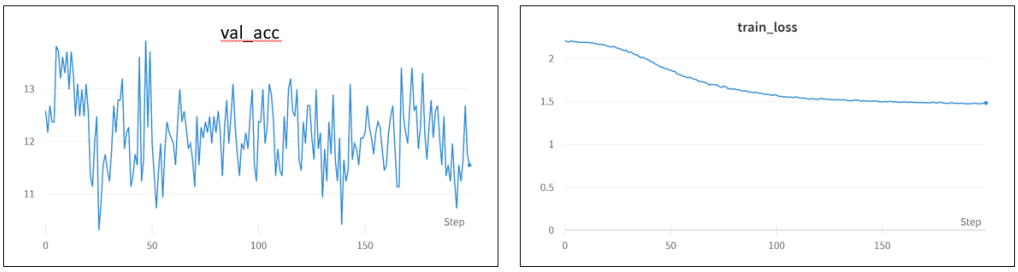

Figure 6: CNN on the not augmented dataset

versions of images in the validation set that were also in the training set. This overfitting of the validation accuracy will also be a problem in the other runs with different architectures.

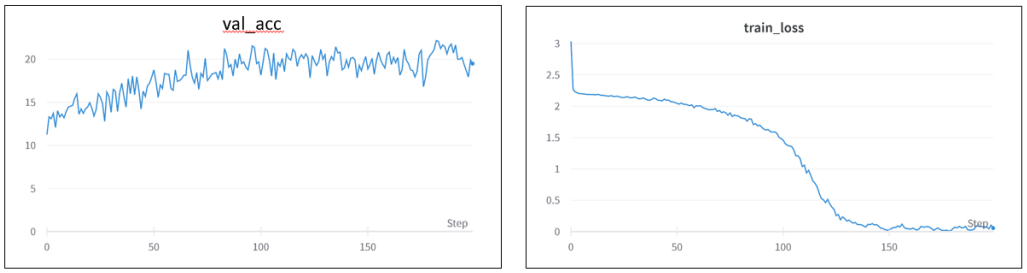

Figure 7: CNN on the augmented dataset

3.4.2 Tests on ResNet34

To compensate for the problems of vanishing gradients, we tried out the ResNet architecture, which is supposed to tackle this issue. And as to be seen in figure 8 ResNet34 improved the results drastically. The validation accuracy went up to 0.47 and the network achieved a test accuracy of 0.24. The huge discrepancy is again due to the mistakes during augmentation. It is easily recognisable when looking at a converging training loss at epoch 130, but still improvements in the validation accuracy.

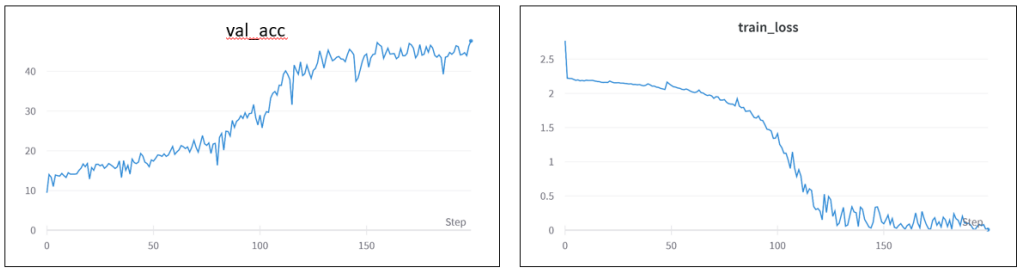

Figure 8: ResNet34 on the augmented dataset

3.4.3 Tests on ResNet50

The last major experiment was on the ResNet50 architecture which is only slightly deeper than the ResNet34. As you can see in figure 9 the results got worse in comparison to ResNet34 but took less epochs to converge. Reason for the results could be that ResNet50 is too deep for the amount of variation in the dataset and therefore not able to train all the parameters.

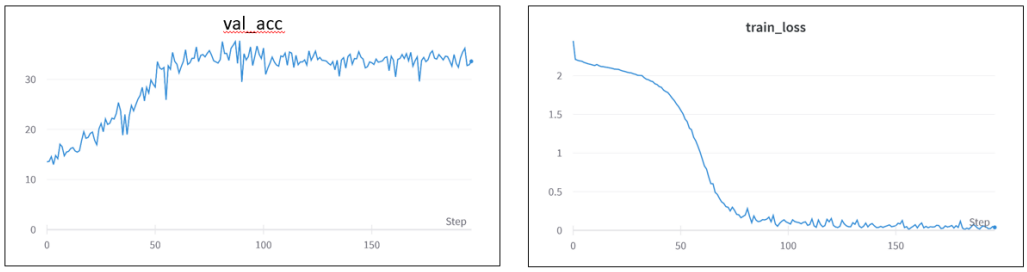

Figure 9: ResNet50 on the not augmented dataset

3.5 Overall Results

Overall the results we achieved on the handmade dataset were not good in comparison to the results we got with MNIST, and also worse than the results of the other student groups who used pre-trained models. Nevertheless, we achieved rates much better than guessing with only 714 images to work with and training everything from scratch. In our case, the ResNet34 architecture was the best performing one with a final kaggle submission score of 0.2431, also shown in table 1.

Table 1: Results on Street Numbers

	CNN	ResNet34	ResNet50
Valid. Acc.	0.2231	0.4768	0.3361
Train Loss	0.055	0.007	0.038
Test Acc.	/	0.2431	0.1944

4 Discussion - Leopold Gierz

The following section discusses the difficulties and limitations encountered at different stages of the project, and aims to encourage future projects to learn from them.

4.1 Key Findings

In the recent experimentation, a fresh ResNet34 yielded an accuracy of approximately 25%, a commendable feat considering the constraints of the limited and error-prone dataset as well as restricted computational resources (due to the use of google colab) at our disposal. A notable observation was the substantial enhancement in model accuracy achieved through data augmentation, albeit to a certain threshold. While deeper networks exhibited superior performance potential, it became evident that their efficacy would be heightened with a more extensive original dataset.

4.2 Limits

In our endeavor to train models for the identification of house numbers ranging from 1-9, several limiting factors surfaced which impacted the optimal performance of our machine learning systems. Here, we discuss the primary challenges that stemmed from data collection mistakes, computational constraints, and inherent issues within the dataset.

4.2.1 Mistakes

As Team OverxAIted, we have unfortunately made a significant contribution to reducing the quality of the data. Because we took one picture per member when collecting the data (often from different angles and distances), there are 4 versions of many house numbers, but they are all similar. It is important to note, however, that without this redundancy, the size of the original dataset would be significantly smaller. Our team contributed 508 samples out of the 714 available for training and testing. (Without the redundancy, we would have contributed only about 127 images, resulting in a

total of about 333 samples for the project). These images were multiplied by increasing the number of training samples from 570 to 7410 (a factor of 13) using data augmentation, which greatly increases the likelihood of data leakage and consequent overfitting of the model. In addition, we performed the augmentation before splitting the data into validation and training sets, which virtually guarantees that some of the images we trained on will be present (in slightly modified form) in the validation dataset. (For example, if 3 of the 4 similar images were in the training set, after the augmentation there would be 39 images of the same house number, some in the training set and some in the validation set).

Figure 10: dataset samples: 0111.png, 0246.png, 0480.png (rotated by 270°), 0481.png (rotated by 270°)

A separate challenge to the whole project, which was beyond the scope of our team, was the problem of many of the samples being rotated by 90 degrees. No corrective measures had been taken to deal with these misaligned images, which undoubtedly undoubtedly added a further layer of complexity to the task, as it is impossible for the model to say for example, whether a rotated 6 is actually a 6 or a 9. For example, the two examples shown above, 0480.png and 0481.png, are rotated 90 degrees in the original dataset.

4.2.2 Computational Resources

Our reliance on Google Colab presented some computational challenges. Attempting to load all the data at once limited us to using only the 7410 samples we had obtained through augmentation. From our previous experience, we recognised a positive correlation between the volume of data samples and the resulting model accuracy. Although augmentation is beneficial to overfitting as discussed above, we would have liked to test the effect of further augmentation of the dataset on model accuracy.

However, one avenue we did not explore that could have mitigated these challenges was incremental learning. By employing such techniques, we could have potentially processed larger datasets without overwhelming the computational capacity provided by Colab.

In addition, the inherent variability in computing power due to fluctuating user workloads on Google Colab added an unpredictable dimension to our project's resource allocation. The free version of Colab that we used imposed further constraints. The notebooks had a maximum runtime of 12 hours, and we were limited to around 12GB of RAM. These limitations became particularly apparent when we attempted to perform tasks that required more memory, such as loading the dataset for training, causing system crashes and compromising the consistency and efficiency of our training processes.

4.2.3 Dataset

The composition and quality of our dataset posed numerous challenges. Firstly, it was pre-biased, reducing the chances of generating a well-trained model solely reliant on this data, especially given the small size of our dataset and the plethora of misleading details within the images. Bamberg's standardization of house numbers further complicated matters. A model trained on these numbers might struggle to identify unique or specialized house numbers, with an inherent bias towards identifying any blue square as a potential house number. As shown in figure 11, it may be difficult to recognise the rightmost seven (0166.png) compared to the other images containing the blue square pattern.

Figure 11: dataset samples: 0222.png, 0187.png, 0217.png, 0166.png

There are also external factors such as shadows, the presence of different house number designs, and sometimes ambiguous designs that are difficult even for humans to identify (for example, determining whether a number is a '9' or a '2'), further hampered effective pattern recognition. A noteworthy mention is the presence of textual elements within images, such as letters like 'E' or 'R'. These letters, while innocuous in everyday scenarios, could mislead a model that's primed to identify a number like '3'. The large variety of angles and sizes of the house numbers in the picture also complicated matters. An example for each of these difficulties is shown in figure 12.

Figure 12: dataset samples: 0683.png, 0361.png (rotated by 270°), 0695.png, 0158.png

In sum, while our project had a well-defined goal, the journey was fraught with challenges, shedding light on the myriad considerations needed when embarking on image recognition tasks.

4.2.4 Proposal of Improvements

Addressing the limitations previously highlighted requires a multi-pronged approach to refine our methodology and ultimately achieve better model performance. The following measures are recommended:

1. **Data Collection with Regards to Sources of Error:** To maximize the quality and relevance of our data, a more coordinated and collaborative approach is vital. Teams should collaboratively define the data collection strategy, standardize criteria, and monitor for common sources of error. This unified approach will ensure a more diverse and robust dataset.
2. **Correctly Rotate All Images:** Unfortunately, we don't know why some of the images are rotated in the first place, but given the dataset containing rotated samples, all images should undergo a pre-processing step to check for and correct orientation problems. It may also be useful to investigate techniques for identifying rotated images in order to automate the orientation correction process.
3. **Remove Duplicates:** Post-collection, a deduplication process should be implemented. Using algorithms that can recognize and flag identical or nearly identical images will ensure that the dataset remains lean and relevant, minimizing the chances of overfitting and redundancy.
4. **Split the Data into Training and Validation Set:** Before any augmentation or further processing, the dataset should be split into training and validation subsets. This ensures that our model validation is done on fresh, unseen data, giving a more accurate representation of the model's real-world performance.

5. **Apply Data Augmentation:** Only after the above steps should data augmentation techniques be applied, and specifically to the training data. This approach will exponentially increase the volume of training data without introducing biases in the validation set, providing a varied and rich dataset to train on without compromising the integrity of the validation process.

Incorporating these steps into our methodology will tackle the previously encountered challenges head-on and pave the way for a more streamlined, efficient, and effective machine learning workflow.

5 Conclusion - Leopold Gierz

Underneath the layers of this project's methodologies and findings, several key takeaways emerge that underscore the significance of data quality and model choices in achieving optimal results.

5.1 Summary

In this project, we made several observations and drew crucial conclusions. One predominant realization was that, with the given dataset, the model's choice becomes secondary unless we opt for a pre-trained variant. The data quality emerges as the more significant influencer on the outcome. Although data preprocessing can elevate this quality to some extent, it eventually hits a threshold. The ultimate enhancement lies in expanding the dataset with distinct, appropriately oriented, and easily discernible images. Moreover, there's potential in exploring more intricate networks and delving deeper into ResNets.

5.2 Outlook

Looking forward, there are promising avenues to explore. Leveraging pre-trained models, such as VGG16, seems to be a viable strategy. These models, trained on extensive datasets with the aid of robust computational resources, have learned rich feature representations from varied data sources. These acquired features encapsulate general patterns and knowledge, allowing for more seamless transfer and application to our problem. Another consideration is adjusting the resolution by decreasing image size, which can simplify feature recognition. However, striking a balance is crucial, as over-reducing resolution might lead to overfitting, rendering features indistinguishable. This approach has the added advantage of reduced computational complexity, streamlining the training process.

References

[1] C. Deotte. How to choose cnn architecture mnist. 2018. URL `https://www.kaggle.com/code/cdeotte/how-to-choose-cnn-architecture-mnist`.

[2] K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4):193–202, 1980.

[3] I. J. Goodfellow, Y. Bulatov, J. Ibarz, S. Arnoud, and V. Shet. Multi-digit number recognition from street view imagery using deep convolutional neural networks. *arXiv preprint arXiv:1312.6082*, 2013.

[4] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[6] 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.

[7] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[8] M. Saeidi and A. Arabsorkhi. A novel backbone architecture for pedestrian detection based on the human visual system. *The Visual Computer*, 38, 06 2022. doi: 10.1007/s00371-021-02280-6.

[9] P. Sermanet, S. Chintala, and Y. LeCun. Convolutional neural networks applied to house numbers digit classification. In *Proceedings of the 21st international conference on pattern recognition (ICPR2012)*, pages 3288–3291. IEEE, 2012.

[10] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

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, August 30, 2023	
(Place, Date)	(Signature)
Bamberg, August 30, 2023	
(Place, Date)	(Signature)
Bamberg, August 30, 2023	
(Place, Date)	(Signature)
Bamberg, August 30, 2023	
(Place, Date)	(Signature)