Big Data Intelligence Assignment 3

Gausse Mael DONGMO KENFACK (董沫高斯)
Student ID: 2024403346

December 2024

Contents

Introduction

With the rise of deep learning, computer vision has garnered significant attention from researchers in recent years. Deep learning models have proven highly effective for various computer vision tasks, with applications integrated into our daily lives. One classical problem in this field is handwritten character recognition, where AI is used to identify characters from images of handwriting. The goal is to train a deep learning model using labeled images of handwritten characters so that it can accurately predict the corresponding character for a new, unseen image. This assignment focuses on tackling this task with different approaches.

1 Experiments

1.1 Dataset

The Dataset provided for these experiments is the "***Omniglot Dataset***" [1]. It is a widely used benchmark in machine learning for character recognition and few-shot learning tasks. It is challenging and diversified, with 1,623 handwritten characters from 50 various alphabets, including imaginary and real-world scripts. There are 20 distinct examples for each character, each written by a different person, guaranteeing a variety of writing styles. The dataset mimics human learning because it simulates a situation in which models must learn to recognize new characters with few instances like humans. Because of its adaptability, Omniglot has become a mainstay for testing algorithms in one-shot classification, transfer learning, and meta-learning. It provides a rich environment for investigating AI's potential to identify patterns in several languages and writing systems.

Each handwritten image is represented as 28x28 black and white pixel points, so it requires less computational resources than real image classification datasets. The TAs provided a ***utils*** file to help us load the images and labels and split the dataset as required for the experiments (15 for training and 5 for the tests).

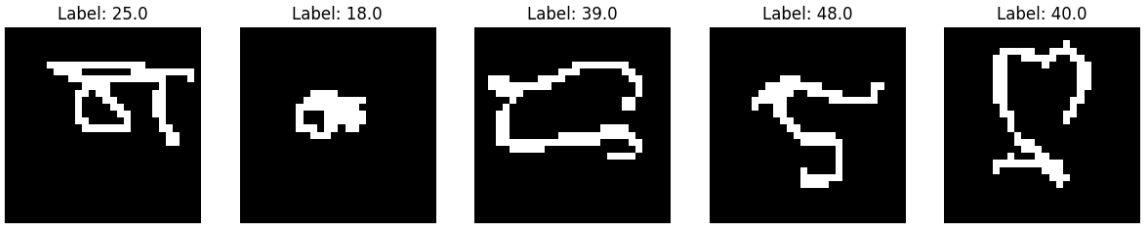

Figure 1: Sample example of the elements in the dataset

We are going to classify the images using 2 different neural networks. The first one is a fully connected neural network (FCNN), and the second one is convolutional neural network (CNN). Both models have been implemented with ***PyTorch*** and trained with the same dataset samples.

1.2 Fully Connected Architecture

The implemented fully connected network has 2 hidden layers with 1100 neurons each and an output layer with 50 neurons (the number of classes). Every layer has the ***ReLu*** function as activation except for the output layer, because the model is trained with a Cross Entropy Loss and it's PyTorch implementation takes logits. This network has a total of more than 2 millions parameters. The Figure 2 shows a summary of the model.

We trained it for 7 epochs with an Adam Optimizer initialized with a 0.001 learning rate and we got an accuracy of 54.40% on the test dataset. The Figure 3 shows the training and testing results.

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1              [-1, 1, 1100]         863,500
            Linear-2              [-1, 1, 1100]       1,211,100
            Linear-3                [-1, 1, 50]          55,050
================================================================
Total params: 2,129,650
Trainable params: 2,129,650
Non-trainable params: 0
----------------------------------------------------------------
```

Figure 2: Summary of the fully connected network

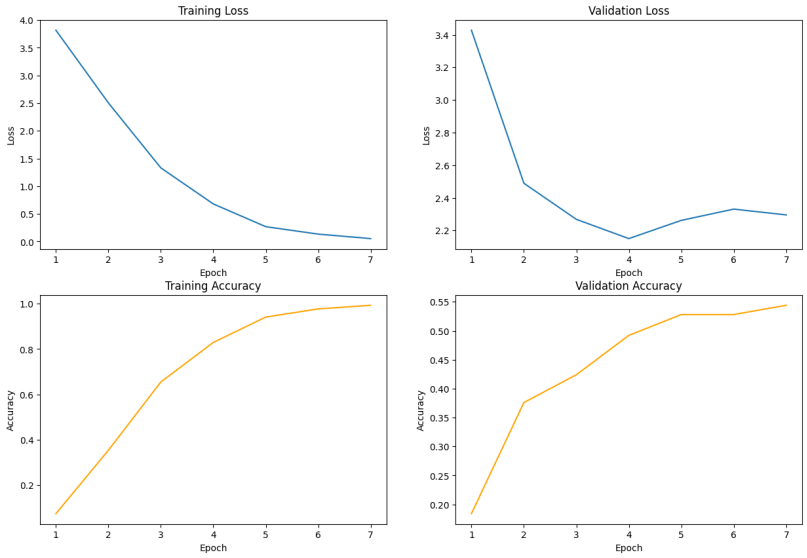

Figure 3: Experiment results for the fully connected network

1.3 CNN Architecture

The implemented CNN has 2 convolutional blocks and 2 fully connected layers. The first convolutional block has 2 convolutional layers with 32 kernels of size 3×3 followed each a batch normalization. The second one has the same structure but with 64 kernels. A max poling is done at the end of each convolutional block. The first fully connected layer has 512 neurons and is followed by a batch normalization and a Dropout. The second one has 50 neurons, the number of classes. The Figure 4 shows a summary of the model. Overall, the network has 616,850 parameters, with convolutional layers extracting hierarchical features and fully connected layers performing the classification.

We trained it for 30 epochs with an Adam Optimizer initialized with a 0.001 learning rate and we got an accuracy of 85.60% on the test dataset.

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 32, 26, 26]	320
BatchNorm2d-2	[-1, 32, 26, 26]	64
Conv2d-3	[-1, 32, 24, 24]	9,248
BatchNorm2d-4	[-1, 32, 24, 24]	64
MaxPool2d-5	[-1, 32, 12, 12]	0
Conv2d-6	[-1, 64, 10, 10]	18,496
BatchNorm2d-7	[-1, 64, 10, 10]	128
Conv2d-8	[-1, 64, 8, 8]	36,928
BatchNorm2d-9	[-1, 64, 8, 8]	128
MaxPool2d-10	[-1, 64, 4, 4]	0
Linear-11	[-1, 512]	524,800
BatchNorm1d-12	[-1, 512]	1,024
Dropout-13	[-1, 512]	0
Linear-14	[-1, 50]	25,650

Total params: 616,850
Trainable params: 616,850
Non-trainable params: 0

Figure 4: Summary of the fully connected network

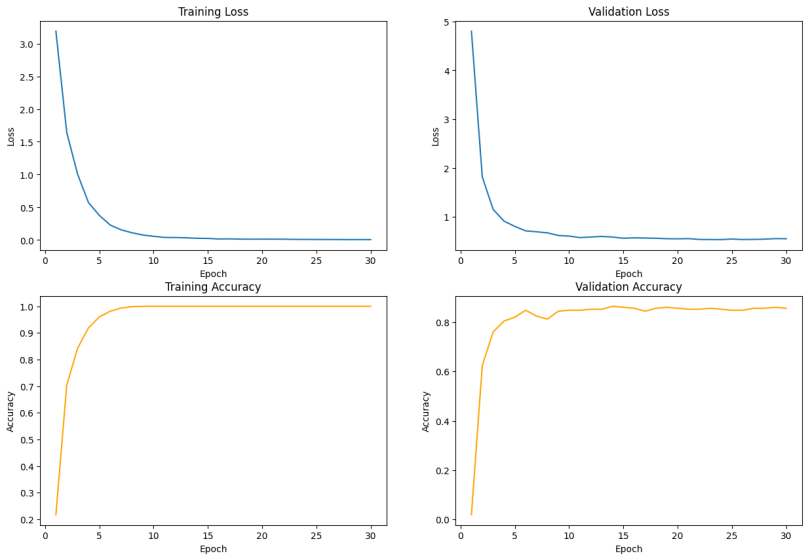

Figure 5: Experiment results for the CNN

1.4 Advantages and Disadvantages

Before starting to study the influence of some factors on the network performance we made the Table 1 to compare the 2 baselines.

Criteria	Fully Connected	CNN
Accuracy	54.40%	85.60%
Number of parameters	2 129 650	616 850
Advantages	Better than a random guess, Simple to implement, The network is structure agnostic	Better than a random guess, Captures spacial features, Better accuracy, Less parameters, Robust to image variations
Disadvantages	Too much parameters, Weaker performance, Training time scales with network size, Ignore spacial information, Overfits a lot	Complicated to design and implement Specific only for grid-like data

Table 1: Comparison of the 2 networks

2 Trials and Analysis

In this second section, we will conduct trials and comparisons across various aspects of the networks, including the dataset, the training process, and the model architecture.

2.1 Influence of the Dataset and Sampling

In the first part of this report, we highlighted that the *Omniglot* dataset is particularly challenging. A model with average performance on *Omniglot* could achieve significantly better results on a simpler dataset, such as *MNIST*. Previously, we also mentioned that for each category, 15 out of 20 images were used for training, and the remaining 5 for testing. To better understand the impact of this split, we trained the same models using different train-test splits (10/10, 11/9, 12/8, ..., up to 19/1). The results of these experiments are shown in the Figure 6.

In that Figure, the experiment i represent the experiment with the split $(10 + i)/(10 - i)$. As we expected, the accuracy generally increases as the number of training samples grows. This is a logical outcome since providing the model with more data typically allows it to learn more robust features and improve its performance. However, it is important to note that the accuracy curves do not always grow in a perfectly smooth or monotonic fashion. This behavior can be attributed to the randomness involved in sampling and shuffling the data during the train-test splits. Small variations in the specific images selected for training or testing can lead to fluctuations in performance, which is especially noticeable when working with smaller datasets. Despite these irregularities, the overall trend remains clear: increasing the number of training samples leads to better accuracy, as the model benefits from a richer learning experience.

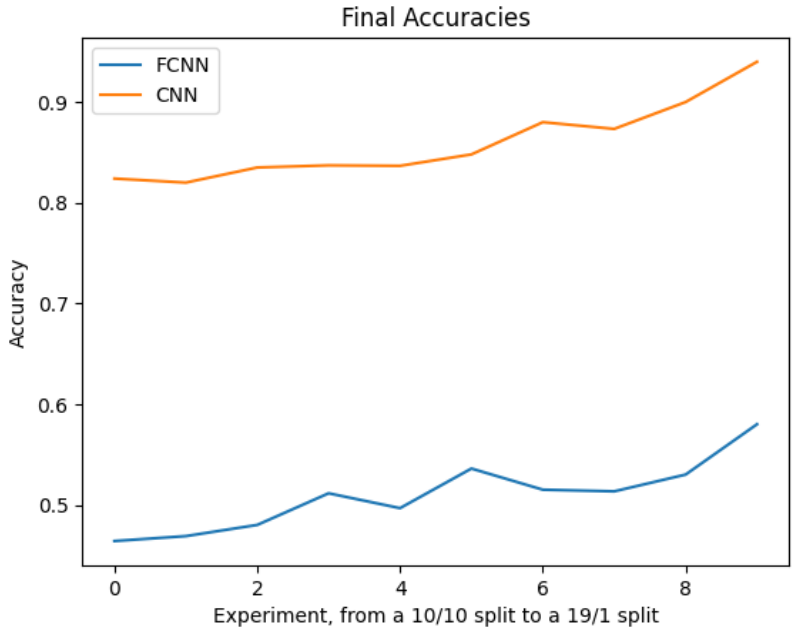

Figure 6: Evolution of the accuracy with the number of training/test samples

2.2 Influence of the model Architecture

In the Table 1 we listed the advantages and disadvantages of the 2 different types on neural networks and the conclusion we can get from it is that the CNN is more suitable for this image classification task. But what if we want to make a deeper analysis? not just the general architecture of the network but the low level architecture ? (how many layers? which layers?) That's the purpose of this part.

Let's start with the Fully connected network We trained 5 fully connected networks for 10 epochs with the same hyper-parameters as before. The only difference here is that the network i has i hidden layers with 1100 neurons each. We then plotted the final training and test accuracy for each network, you can see it on Figure 7.

As the number of hidden layers increases, the training accuracy begins to decrease. This suggests that deeper networks are struggling to effectively fit the training data, possibly due to optimization issues such as vanishing gradients, lack of data, or insufficient training time (we only trained for 10 epochs). The test accuracy is consistently lower than the training accuracy, indicating overfitting for networks with fewer hidden layers. The figure highlights that increasing the number of hidden layers beyond a certain point can hurt both training and test performance. In this case, a network with 1 or 2 hidden layers strikes a balance between simplicity and generalization.

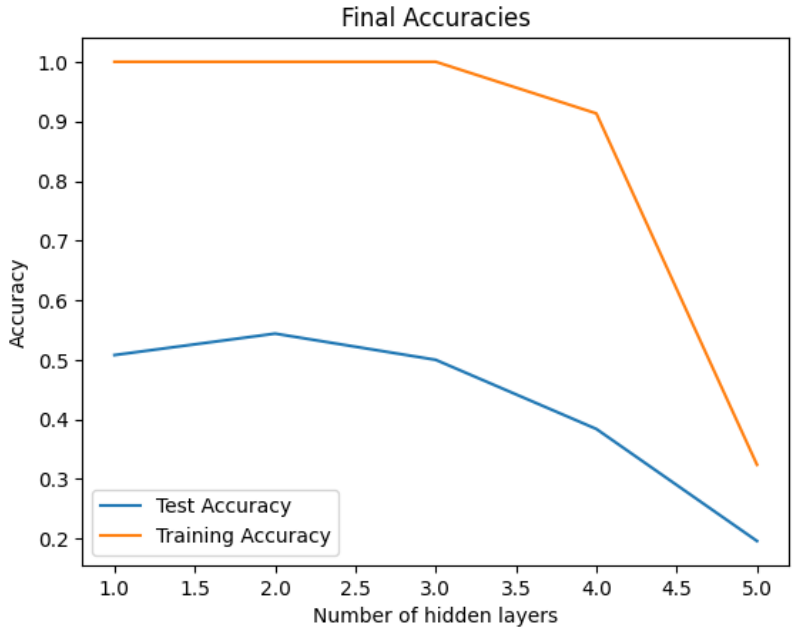

Figure 7: Evolution of the accuracies for 10 epochs with the number of hidden layers

Now for the CNN To see the influence of the architecture on the CNN we compared the previous CNN with another one that does not have any pooling layer at the end of its convolutional blocks. The results are in the Table 2.

Criteria	CNN with max pooling	CNN without max pooling
Accuracy	85.60%	78.40%
Number of parameters	616 850	13 199 762
Training Time (s)	48.48	151.50
Total size (MB)	3.18	52.18

Table 2: Comparison of the 2 CNN

This table show the importance of the max pool layers in reducing the models parameters and directly the training time, it's size because it helps in shrinking the feature maps size. It also contribute to a better accuracy and helps avoid overfitting.

2.3 Influence of the hyper-parameters

To finish with this section of Trials and Analysis we are going to analze the impact of some hyper-parameters.

Influence of the Optimizer on the fully Connected Network We trained the same fully connected network with 3 different optimizers and here are the results. We notice the Stochastic

gradient descent has really bad results compare to the one we used from the beginning (Adam). We can also notice that RMSProp optimizer helps achieve better accuracy.

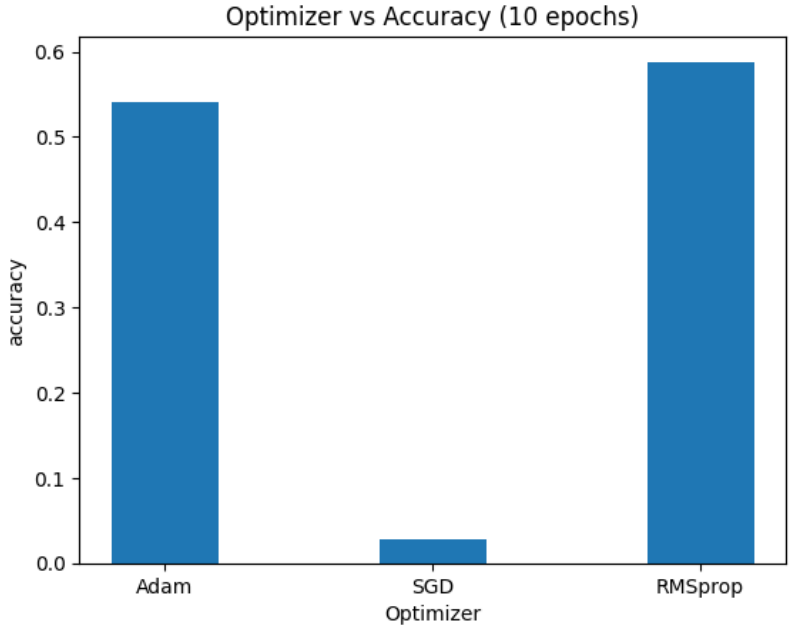

Figure 8: Accuracy vs Optimizer for the FCNN

Influence of the Learning rate on the CNN We know that the learning rate helps to achieve better convergence, We tried to visualize it's importance by training the CNN for 30 epochs with 5 different learning rates and the results of Figure 9 shows that it's a parameters that needs to be well chosen.

2.4 Potentially Interesting Analysis

- Influence of the conv2d layer's kernel size and padding
- Try changing the activation functions
- Try using average pooling instead of max pooling
- Add some dropout layers to the FCNN

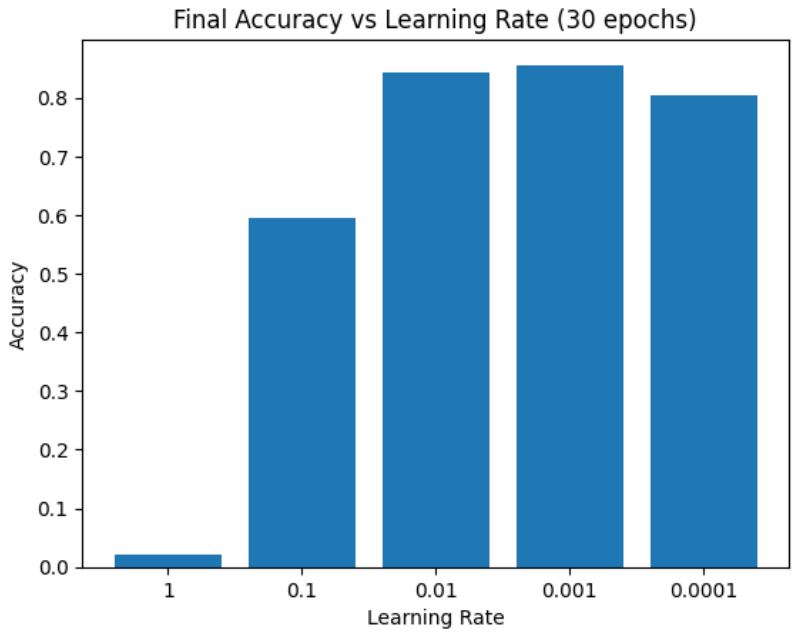

Figure 9: Accuracy vs Learning rate for the CNN

Conclusion

To conclude, we classified images from the *omiglot* dataset with a convolutional neural network (CNN) and a Fully connected neural network (FCNN). We analyzed their architectures and our results. We then gave the advantages and disadvantages of each network. After that we tested some features so see the influence of neural network environment (dataset, architecture, hyper-parameters) on its results. It was a great learning experience to work with a dataset as challenging as *omniglot* and the real difficulty here was to balance the thinking investments towards the 2 networks because they are both interesting.

References

[1] Brenden M. Lake, Ruslan Salakhutdinov, and Joshua B. Tenenbaum. "The Omniglot Challenge: A 3-Year Progress Report". In: *CoRR* abs/1902.03477 (2019). arXiv: 1902.03477. URL: http://arxiv.org/abs/1902.03477.