The Omniglot Challenge

Silin DU

Department of Management Science and Engineering, Tsinghua University
dsl21@mails.tsinghua.edu.cn

June 6, 2023

清華大學
Tsinghua University

Contents

1 Environments and Project Structure

We use `PyTorch` to implement several deep neural networks for the Omniglot challenge [4]. The required environments are as follows.

- torch-1.11.0
- torchinfo-1.7.1
- scikit-learn-1.2.0
- Pillow-8.4.0

We also need other common packages, including `pandas` and `numpy`. The structure of our project is as follows.

- /codes: contain all the codes.
  - main.py: the entrance of our project.
  - train.py: define the class to train the model.
  - model.py: define all the models.
  - utils.py: load the data for training and evaluations.
  - config.yaml: store the configurations for model training.
- /omniglot_resized: contain the data.
- /output: the path to save models, training logs and results.

To meet the requirement of this homework, we do not submit /omniglot_resized and /output, and put all the codes into one file named as 2021311827.py, which contains a group of predefined configurations. The meanings of each configuration are shown in Table 1.1. We leverage a NVIDIA V100 (32GB) to train all models.

To run our codes, use the following command.

```
python 2021311827.py
```

In this report, we first show some samples in the data set in Section 2, and then implement a convolutional neural network (CNN) and a fully-connected neural network in Section 3. After that, we build a new CNN and explore the influence of model structure, the split of dataset and some deep learning tricks in Section 4. Finally, we highlight several conclusions in Section 5.

2 Data

The Omniglot data set [3, 4] contains 1623 different handwritten characters from 50 different alphabets. Each of the 1623 characters was drawn online via Amazon's Mechanical Turk by 20 different people. In other words, the data set contains $1623 \times 20 = 32460$ samples, which are divided into 1623 classes.

Each sample in the data set is a 28×28 one-channel image. We visualize several samples in Figure 2.1. In this report, we will randomly select 50 characters to conduct our experiments. The small data set contains 50 classes, and we select 15 samples of each character to form the

Table 1.1: Configurations

Name	Description
Model Saving	
model_name	Model name used to save a model.
model_dir	Directory to save models, training logs and results.
Data	
dataset_path	Directory of the dataset.
num_class	The number of classes that make up the data in the training set.
train_sample	The number of samples in each class that used to train the model, expected to be between 1 and 19.
Model Setting	
model_type	The type of deep models, expected to be one of [`'fc'`, `'cnn_base'`, `'my_cnn'`].
dropout	Dropout rate.
batch_norm	Whether to use batch normalization, expected to `True` of `False`.
Training Setting	
batch_size	Batch size used in training, validation and testing.
learning_rate	Learning rate during training.
epochs	Total epochs of training.
weight_decay	The regularization parameter used in `torch.optim`, can be expressed like `2e-5`.

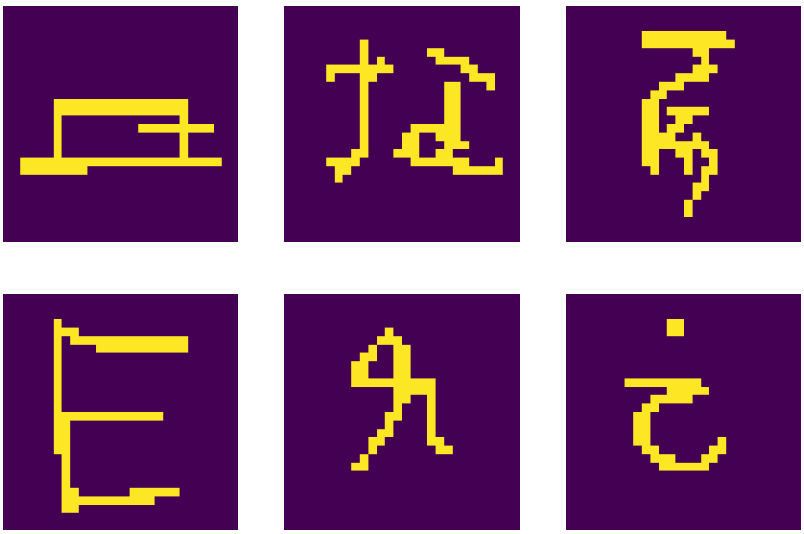

Figure 2.1: Several samples in the data set.

training set and treat the rest 5 samples as the test set. Therefore, the training set contains $15 \times 50 = 750$ samples, and the test set has $15 \times 5 = 75$ samples. We modify this split of the data set in Section 4.

At last, to select the best parameters of each deep model, we randomly select 10% samples (i.e., 75 samples) of the training set as the validation set.

3 Base Models

In this part, we implement the following two deep models.

1. CNN_base: a convolutional neural network contains two blocks, each of which has two convolutional layers and a max-pooling layer. We use a 3×3 filter in each convolutional layer, and the filter size in the max-pooling layer is 2×2. A fully-connected neural network with one hidden layer is utilized to make the final predictions, where the size of the hidden layer is 512. Figure 3.1(a) summarizes the details of this model.
2. FC: a fully-connected neural network equipped with two hidden layers, where the size of each hidden layer is 512. Figure 3.1(b) summarizes the details of this model.

```
==========================================================
Layer (type:depth-idx)                   Param #
==========================================================
Vanilla_CNN                              --
├─Sequential: 1-1                        --
│    └─Conv2d: 2-1                       320
│    └─BatchNorm2d: 2-2                  64
│    └─ReLU: 2-3                         --
│    └─Conv2d: 2-4                       9,248
│    └─BatchNorm2d: 2-5                  64
│    └─ReLU: 2-6                         --
│    └─MaxPool2d: 2-7                    --
│    └─Conv2d: 2-8                       18,496
│    └─BatchNorm2d: 2-9                  128
│    └─ReLU: 2-10                        --
│    └─Conv2d: 2-11                      36,928
│    └─BatchNorm2d: 2-12                 128
│    └─ReLU: 2-13                        --
│    └─MaxPool2d: 2-14                   --
├─Sequential: 1-2                        --
│    └─Linear: 2-15                      1,606,144
│    └─BatchNorm1d: 2-16                 1,024
│    └─ReLU: 2-17                        --
│    └─Dropout: 2-18                     --
│    └─Linear: 2-19                      25,650
│    └─Softmax: 2-20                     --
==========================================================
Total params: 1,698,194
Trainable params: 1,698,194
Non-trainable params: 0
==========================================================
```

(a) Details of CNN_base

```
==========================================================
Layer (type:depth-idx)                   Param #
==========================================================
FC_Network                               --
├─Sequential: 1-1                        --
│    └─Linear: 2-1                       803,840
│    └─BatchNorm1d: 2-2                  2,048
│    └─ReLU: 2-3                         --
│    └─Linear: 2-4                       524,800
│    └─BatchNorm1d: 2-5                  1,024
│    └─ReLU: 2-6                         --
│    └─Linear: 2-7                       262,656
│    └─BatchNorm1d: 2-8                  1,024
│    └─ReLU: 2-9                         --
│    └─Dropout: 2-10                     --
│    └─Linear: 2-11                      25,650
│    └─Softmax: 2-12                     --
==========================================================
Total params: 1,621,042
Trainable params: 1,621,042
Non-trainable params: 0
==========================================================
```

(b) Details of FC

Figure 3.1: The structure of two base models.

The CNN_base model is provided as a baseline in this homework, and we design the FC in order to make the number of parameters comparable to the CNN_base.

To train the CNN_base model, we resize each image as a 28×28 array. While for FC, we can directly feed the data into the model. During training, we set the batch size as 128, and the learning rate is 0.001. Two models converge after 300 epochs. For the FC model, we set the weight decay rate as $1e-4$ to avoid overfitting, but we do not use this trick when training the CNN_base model. We use Adam [2] to train the whole network.

Table 3.1: Experimental results of two base models.

Settings	Batch Size	Learning Rate	Epochs	Weight Decay	# Parameters
CNN_base	128	0.001	300	0	1,698,194
FC	128	0.001	300	0.0001	1,621,042
Results	Accuracy	Precision	Recall	F1	
CNN_base	0.9040	0.9122	0.9040	0.9020	
FC	0.5280	0.6022	0.5280	0.5206	

Table 3.1 summarizes several key settings during training and the evaluation results of two models. Unless noted, we report the macro average of the precision, recall and F1 score hereafter. Since our small data set contains 50 classes, a random guess will get around 2% accuracy score. From this point of view, both CNN_base and FC can learn useful patterns for better predictions compared with random guess.

However, despite the number of parameters of these two models is similar, the performance of CNN_base is much better than that of FC. CNN_base can improve the accuracy score by about 71.21%.

We highlight several advantages of convolutional neural networks here.

1. When handling large images (e.g., 1080P $= 1920 \times 1080 \times 3$), a fully-connected network needs a huge amount of parameters. If we design 20 units in the first hidden layer, the number of weight parameters in this layer is

   $$1920 \times 1080 \times 3 \times 3 \times 20 = 124,416,000$$

   But if we design a convolutional layer with twenty $(5 \times 5 \times 3)$ filters, the number of all parameters (including bias) in this layer is

   $$5 \times 5 \times 3 \times 20 + 20 = 1520$$

   Although, in our problem, the image is one-channel and is quite small, we can build a deeper network with convolutional blocks, holding the number of parameters roughly the same.
2. Each unit in the fully-connected network has connection with all the units in the previous layerm which means it will pay attention to the global features of the image. It might be not helpful in many situations.
   When analyzing an image, we learn several patterns from several local features, like the shape and edge. Therefore, a convolutional kernel is suitable for extracting local features and the pooling operation can aggregate the local features into high-level features, which makes CNN models powerful dealing with image data.

4 Explorations

In this section, we implement another CNN model, and test the influence of model structure, the split of dataset and some deep learning tricks.

4.1 A New CNN Model

Inspired by the VGG network (Simonyan and Zisserman (2014) [5]), we build a new CNN model, named CNN_new. In this model, we have four convolutional blocks, each of which contains a convolutional layer with 3×3 filters and a max-pooling layer with 2×2 filters. Figure 4.1 displays the details of this model. Note that the number of parameters in CNN_new is slightly more than that of CNN_base and FC.

```
==========================================================
Layer (type:depth-idx)                   Param #
==========================================================
CNN_classifier                           --
├─Sequential: 1-1                        --
│    └─Conv2d: 2-1                       640
│    └─BatchNorm2d: 2-2                  128
│    └─ReLU: 2-3                         --
│    └─MaxPool2d: 2-4                    --
│    └─Conv2d: 2-5                       73,856
│    └─BatchNorm2d: 2-6                  256
│    └─ReLU: 2-7                         --
│    └─MaxPool2d: 2-8                    --
│    └─Conv2d: 2-9                       295,168
│    └─BatchNorm2d: 2-10                 512
│    └─ReLU: 2-11                        --
│    └─MaxPool2d: 2-12                   --
│    └─Conv2d: 2-13                      1,180,160
│    └─BatchNorm2d: 2-14                 1,024
│    └─ReLU: 2-15                        --
│    └─MaxPool2d: 2-16                   --
├─Sequential: 1-2                        --
│    └─Linear: 2-17                      131,328
│    └─ReLU: 2-18                        --
│    └─Linear: 2-19                      65,792
│    └─ReLU: 2-20                        --
│    └─Dropout: 2-21                     --
│    └─Linear: 2-22                      12,850
==========================================================
Total params: 1,761,714
Trainable params: 1,761,714
Non-trainable params: 0
==========================================================
```

Figure 4.1: The structure of CNN_new.

We use nearly the same training settings to train CNN_new, and the evaluation results are shown in Table 4.1. The Adam optimizer is selected here.

Table 4.1: Experimental results of the new CNN model.

Settings	Batch Size	Learning Rate	Epochs	Weight Decay	# Parameters
FC	128	0.001	300	0.0001	1,621,042
CNN_base	128	0.001	300	0	1,698,194
CNN_new	128	0.001	300	0.001	1,761,714

Table 4.1 —— Next Page

Table 4.1 —— Continued

Results	Accuracy	Precision	Recall	F1
FC	0.5280	0.6022	0.5280	0.5206
CNN_base	0.9040	0.9122	0.9040	0.9020
CNN_new	0.9400	0.9470	0.9400	0.9399

We use a relatively large weight decay when training CNN_new to avoid overfitting, considering that the number of parameters is larger. We can observe a 3.98% improvement in accuracy compared with CNN_base, which demonstrates the effectiveness of the design of convolutional blocks in VGG-16.

4.2 Dropout and Batch Normalization

In previous CNN models, we add a batch normalization (BN) layer after each convolutional layer (See Figure 3.1 and 4.1). In all models, we add a dropout layer with 0.3 dropout rate before the output layer. To verify the effect of these two tricks, we remove the dropout layer and the BN layer in CNN models. All models are trained over 300 epochs, the learning rate is 0.001, and the batch size is 128. Other settings and the evaluation results can be found in Table 4.2.

Table 4.2: Experimental results of models without dropout layers and BN layers.

Settings	Dropout	BN	Weight Decay	
CNN_base	0.3	Yes	0	
CNN_new	0.3	Yes	0.001	
CNN_base_no_dropout	0	Yes	0	
CNN_new_no_dropout	0	Yes	0.0001	
CNN_base_no_BN	0.3	No	0.0001	
CNN_new_no_BN	0.3	No	0.0005	
Results	Accuracy	Precision	Recall	F1
CNN_base	0.9040	0.9122	0.9040	0.9020
CNN_new	0.9400	0.9470	0.9400	0.9399
CNN_base_no_dropout	0.8800	0.8912	0.8800	0.8767
CNN_new_no_dropout	0.9240	0.9349	0.9240	0.9242
CNN_base_no_BN	0.7400	0.7911	0.7400	0.7277
CNN_new_no_BN	0.7520	0.7643	0.7520	0.7460

From Table 4.2, we can observe that both batch normalization layers and dropout layers make contributions to the performance. Removing the batch normalization layer will signif-

icantly hurt the performance of deep models. Concretely, the accuracy score of CNN_base is 22.16% better than that of CNN_base_no_BN, and the accuracy score of CNN_new is 25.00% better than that of CNN_new_no_BN.

Batch Normalization [1] is a common tricks to solve internal covariate shift when training deep neural networks. One of the key assumptions in machine learning is the i.i.d. assumption. The training of the shallow layers will change the outputs of these layers, which are also inputs of deep layers. The changes will make it difficult to train the deep layers. To solve this problem, BN layers will normalize each batch of data after each layer, which will make the training more stable and speed up the convergence rate. Therefore, deep models with BN layers will outperform those without this tricks.

4.3 The Split of the Data Set

In previous experiments, we randomly select $K = 50$ classes data to form a toy dataset, and treat $N = 15$ samples in each class as the training set. Now we change K and N to evaluate our CNN models.

Now we set $K = 60$ and modify the output layer in CNN_base and CNN_new. We named the models trained in this data set as CNN_base_60 and CNN_new_60, respectively. Then we set $N \in \{13, 17\}$, and named the models as CNN_base_13, CNN_base_17, CNN_new_13, and CNN_new_17 respectively. We do not modify the training settings.

Table 4.3: Experimental results of models with different N and K.

Settings	# Class (K)	# Train (N)	# Test	Ratio of Validation Set
CNN_base	50	15	5	0.1
CNN_new	50	15	5	0.1
CNN_base_60	60	15	5	0.1
CNN_new_60	60	15	5	0.1
CNN_base_13	50	13	7	0.1
CNN_new_13	50	13	7	0.1
CNN_base_17	50	17	3	0.1
CNN_new_17	50	17	3	0.1
Results	Accuracy	Precision	Recall	F1
CNN_base	0.9040	0.9122	0.9040	0.9020
CNN_new	0.9400	0.9470	0.9400	0.9399
CNN_base_60	0.8967	0.9139	0.8967	0.8901
CNN_new_60	0.9033	0.9196	0.9033	0.9024
CNN_base_13	0.8914	0.9047	0.8914	0.8884
CNN_new_13	0.9314	0.9359	0.9314	0.9301
CNN_base_17	0.9200	0.9317	0.9200	0.9090

Table 4.3 —— Next Page

Table 4.3 —— Continued

Results	Accuracy	Precision	Recall	F1
CNN_new_17	0.9533	0.9650	0.9533	0.9520

More details and evaluation results can be found in Table 4.3. We get the following observations.

1. When increasing K, the problem itself becomes more complicated, which will reduce the performances of models. Particularly, the accuracy score of CNN_base is 0.81% better than that of CNN_base_60, and the accuracy score of CNN_new is 4.06% better than that of CNN_new_60.
2. When increasing N, models will gain better performance due to the larger training set and less testing samples. In particular, the accuracy score of CNN_base_17 is 1.77% better than that of CNN_base, and the accuracy score of CNN_new_17 is 1.41% better than that of CNN_new.
3. In contrast, a smaller N will slightly degrade model performances. Concretely, the accuracy score of CNN_base is 1.41% better than that of CNN_base_13, and the accuracy score of CNN_new is 0.92% better than that of CNN_new_13.

5 Conclusion

In this report, we focus on the Omniglot Challenge and build several toy data sets to conduct experiments. In Section 2, we state the main strategy to build the data set and visualize some samples in our data. Then we build a convolutional neural network and a fully-connected neural network in Section 3. After comparison, we conclude that the intrinsic features of convolutional operations can (1) build deeper models with fewer parameters; and (2) learn local features for better image classification.

After that, we do more explorations in Section 4. We first build a new CNN model using the convolutional blocks proposed in VGG-16. The experimental results show that the new structure with 3×3 convolutional filters and 2×2 pooling filters can get better performance, which demonstrates the effectiveness of the new design of convolutional blocks. Next, we verify the usefulness of dropout layers and batch normalization layers. We find that the batch normalization layers can make much more contributions to the performance gain compared with the dropout layers. At last, we slightly change the split of data set and the building procedure of the toy data set. The results are in line with our expectations. When the number of training samples increases, the model will get better scores. If the number of classes increases, the problem becomes more complicated, which will hurt the performance of each model.

References

[1] Ioffe, S., and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning* (2015), PMLR, pp. 448–456.

[2] Kingma, D. P., and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[3] Lake, B. M., Salakhutdinov, R., and Tenenbaum, J. B. Human-level concept learning through probabilistic program induction. *Science 350*, 6266 (2015), 1332–1338.

[4] Lake, B. M., Salakhutdinov, R., and Tenenbaum, J. B. The omniglot challenge: a 3-year progress report. *Current Opinion in Behavioral Sciences 29* (2019), 97–104.

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