PR Final Project Report

311551059 陳昱丞

Methodology:

First of all, in this final project, the first thing to consider is which network architecture to use for training the data. I have chosen ResNet34. ResNet series is widely used in deep learning and has shown excellent results in various tasks. I only tried ResNet34 due to limited time and computational resources, which prevented me from trying out the entire ResNet series.

The next challenge to address is the classification method and loss calculation. The standard ResNet architecture combined with CrossEntropyLoss can only deal with a single label. However, for Task 2 and Task 3, we need to handle the recognition of multiple characters. My solution is to create a separate model for each task. For Task 2, which involves recognizing two characters, I use two fully connected branches to output the results separately. Branch 1 focuses on recognizing the first character, while Branch 2 focuses on recognizing the second character. I calculate the CrossEntropyLoss for each branch and then combine them to obtain the total loss. This approach allows us to simultaneously address the challenges of multilabel classification and character recognition order. The same approach is applied for Task 3, which involves recognizing four characters.

Furthermore, I found that the data provided for these three tasks was severely insufficient. To improve performance, it was necessary to increase the amount of training data. After trying out several packages, I opted for two Python packages: Augmentor and captcha. Both can be easily installed via pip. Augmentor offers various methods for synthesizing images, and the three methods I used were Distortion, Skew and Tilt, and Rotate. For more details, please refer to the aug_data.py script. Captcha can automatically generate captcha images, and I noticed that the images it generates are quite similar to the training dataset provided by the teaching assistants. Therefore, I attempted to use the generated captcha images for training and found that it produced significantly good results. For more details, please refer to the captcha_gen.py script.

Finally, there are some configurations and hyperparameters that I experimented with, leading to various results and findings. These will be discussed in the following sections.

Environment Details:

Training, Inferencing:

Google Colab

Generating Data:

python	3.9.12
augmentor	0.2.12
captcha	0.4
pandas	1.4.2
pillow	9.0.1

Experimental Setting and Implementation Details:

A. Common Setting, used in all experiments:
   1. Pictures normalize to mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]
   2. Use ResNet34 network with dropout=0.5
   3. Train test split = 8:2
   4. Training epoch=50 for task 1, 2; epoch=75 for task 3
   5. Learning rate = 1e-3
   6. Use lr_scheduler.StepLR, step=10 for task 1, 2; step=15 for task 3; gamma=0.5

B. Data Augmentation, generating more data by:
   1. Augmentor (pip install Augmentor)
      I. Distortion
      II. Skew & Tilt
      III. Rotate
   2. captcha (pip install captcha)

C. Optimizer (provided by pytorch 2.0.0):
   1. Adam(model.parameters(), lr=lr)
   2. RAdam(model.parameters(), lr=lr, betas=(0.9, 0.999), weight_decay=6.5e-4)
   3. AdamW(model.parameters(), lr=lr)
   4. NAdam(model.parameters(), lr=lr, betas=(0.9, 0.999), weight_decay=6.5e-4)

D. Batch Size and Resize:
   1. Pictures resize to 32*32, batch size=256
   2. No resize, batch size=64

Experimental Results:

model	#total training data	#generated data	Data Augmented Method	Optimizer	Batch Size, Resize	Test Score
1	Task1: 9639 Task2: 11956 Task3: 19242	Task1: 10000 Task2: 12500 Task3: 20000	B.1.I	C.1	D.1	0.9078
2	Task1: 4813 Task2: 6035 Task3: 9547	Task1: 4000 Task2: 5000 Task3: 8000	B.1.I + B.1.II	C.1	D.1	0.8898
3	Task1: 6431 Task2: 7999 Task3: 12804	Task1: 6000 Task2: 7500 Task3: 12000	B.1.I + B.1.II + B.1.III	C.1	D.1	0.8808
4	Task1: 7652 Task2: 7964 Task3: 12807	Task1: 7500 Task2: 7500 Task3: 12000	B.1.I	C.1	D.1	0.895
5	Task1: 10045 Task2: 10350 Task3: 17656	Task1: 10500 Task2: 10500 Task3: 18000	B.1.I	C.1	D.1	0.911
6	Task1: 18388 Task2: 18859 Task3: 31973	Task1: 21000 Task2: 21000 Task3: 36000	B.1.I	C.1	D.1	0.9182
7	Task1: 34446 Task2: 34774 Task3: 59902	Task1: 41000 Task2: 41000 Task3: 71000	B.1.I + B.2	C.1	D.1	0.943
8	Task1: 34446 Task2: 34774 Task3: 59902	Task1: 41000 Task2: 41000 Task3: 71000	B.1.I + B.2	C.2	D.1	0.95
9	Task1: 34446 Task2: 34774 Task3: 59902	Task1: 41000 Task2: 41000 Task3: 71000	B.1.I + B.2	C.3	D.1	0.9412
10	Task1: 34446 Task2: 34774 Task3: 59902	Task1: 41000 Task2: 41000 Task3: 71000	B.1.I + B.2	C.4	D.1	0.9442
11	Task1: 67223 Task2: 67591 Task3: 116772	Task1: 82000 Task2: 82000 Task3: 142000	B.1.I + B.2	C.2	D.1	0.94360
12	Task1: 34446 Task2: 34774 Task3: 59902	Task1: 41000 Task2: 41000 Task3: 71000	B.1.I + B.2	C.2	D.2	0.971

Result Analysis:

Based on the experimental data in the table, I tested three different Augmentor methods in the first three experimental setups. I found that Distortion yielded the best results, achieving an accuracy of 90.78% when the data generated by Distortion was mixed with the original training dataset. Interestingly, when I combined the remaining two methods with more data, the accuracy actually decreased. This indicates that the Distortion method is capable of generating more effective data. Consequently, I decided to use this method for all subsequent experiments.

In experiments 4, 5, and 6, I further attempted to generate more data using the Distortion method. As the amount of data increased, the accuracy also improved, but at a gradually slowing rate, and it required more training time. After weighing the pros and cons, I decided to generate 21,000 samples for task 1, 21,000 samples for task 2, and 36,000 samples for task 3. I also applied this setting in subsequent experiments.

In experiment 7, I attempted to incorporate data generated by captchas, and I found a significant improvement in the results. The accuracy jumped from 91.82% to 94.3%. This clearly indicates that the data generated by captchas is also highly effective for training purposes. Therefore, I will continue to apply this setup in subsequent experiments. It is worth noting that the current training data consists of the original training dataset, data generated by the Augmentor method, and data generated by captchas.

In experiments 8, 9, and 10, I tried various different optimizers. The results of the experiments showed that RAdam performed the best, with an accuracy of 95%.

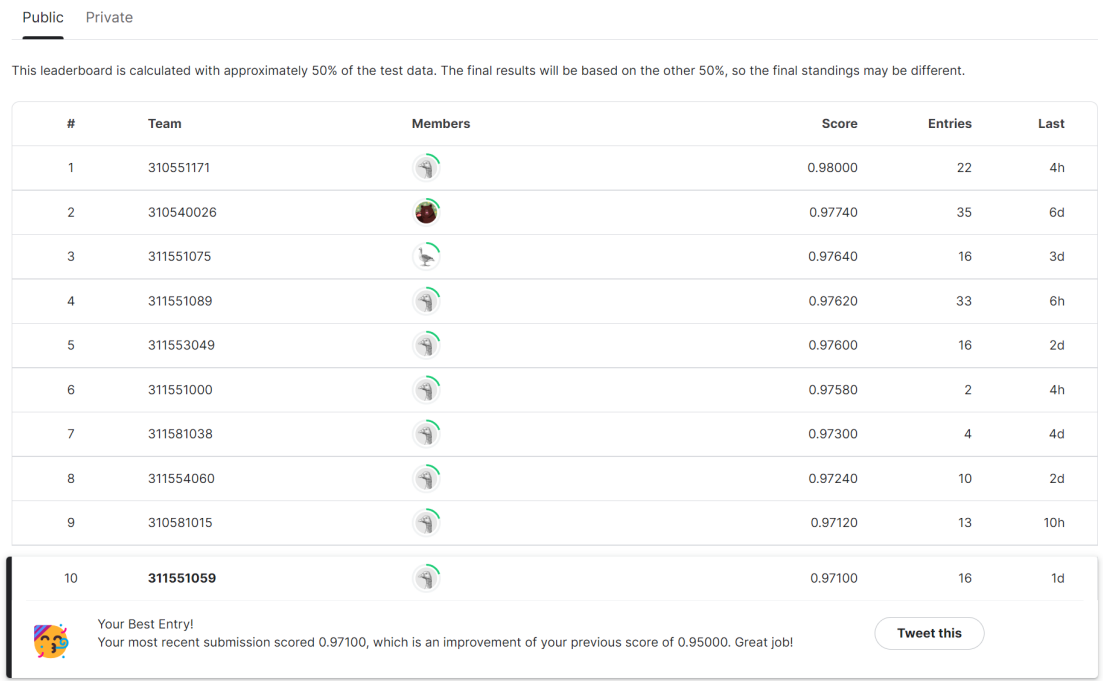
In experiment 11, I attempted to generate more data using captchas. However, there was a slight decrease in accuracy instead. This suggests that generating too much additional data may lead to larger errors.

In experiment 12, I tried removing the resize operation. This required reducing the batch size due to limited GPU memory. However, I found that removing the resize operation and reducing the batch size resulted in even higher accuracy, reaching 97.1%.

Apart from the aforementioned experimental settings, there are many other variables that can be adjusted. However, due to time constraints, I only attempted the above set of experiments.

Kaggle Submission Screenshot:

Rank 10 in 2023/6/6 12:52am.



Public Private

This leaderboard is calculated with approximately 50% of the test data. The final results will be based on the other 50%, so the final standings may be different.

#	Team	Members	Score	Entries	Last
1	310551171		0.98000	22	4h
2	310540026		0.97740	35	6d
3	311551075		0.97640	16	3d
4	311551089		0.97620	33	6h
5	311553049		0.97600	16	2d
6	311551000		0.97580	2	4h
7	311581038		0.97300	4	4d
8	311554060		0.97240	10	2d
9	310581015		0.97120	13	10h
10	**311551059**		0.97100	16	1d

Your Best Entry!
Your most recent submission scored 0.97100, which is an improvement of your previous score of 0.95000. Great job!

Tweet this

How to run inference:

1. Open Google Colab
2. Download the model weights:

https://drive.google.com/drive/folders/1rdE0j3o0dP4bFbutZbWm_7yabCYItZjo?usp=sharing

3. Modify the path including testing dataset, sample_submission path, save to csv path, and the model weights path.

```
[9]  TEST_PATH = "/content/dataset/test"
     SUBMISSION_CSV_PATH = "/content/dataset/sample_submission.csv"
     SAVE_TO_CSV_PATH = "/content/drive/MyDrive/submission.csv"

     TASK1_MODEL_PATH = "/content/drive/MyDrive/models_best/task1.pt"
     TASK2_MODEL_PATH = "/content/drive/MyDrive/models_best/task2.pt"
     TASK3_MODEL_PATH = "/content/drive/MyDrive/models_best/task3.pt"
```

4. Run all.