VRDL HW1: Image Classification:

Name:陳政彥

Student ID:310552006

Link of my code:

Github repo link:
https://github.com/egghead2630/VRDL_HW1
Train Code:
https://github.com/egghead2630/VRDL_HW1/blob/main/train.py
Eval Code(inference.py):
https://github.com/egghead2630/VRDL_HW1/blob/main/inference.py
Models:
https://drive.google.com/drive/folders/1pyId96KiD4XHN25f7EQhh9PYi792z6Pi?usp=sharing
Note that there is a filename extension issue when downloading Models: I have specified it in the README.md, please read about it first in evaluation part.

README:
https://github.com/egghead2630/VRDL_HW1/blob/main/README.md

Reference:

1.https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html
2.https://www.cnblogs.com/denny402/p/7512516.html

Brief introduction:

My model is trained for the HW1 fine-grained classification problem, I use pytorch to accomplish the HW1.

Basic structure:
I split the training data into train-part and validate-part with proportion: **99:1**. The model which performs best on validate-part would be chosen.

Technics applied:
1. Some preprocess and augmentation on the train-part
2. Proper hyperparameters setting(Tuning) based on the before results
3. Transfer learning based on the pretrained model
4. Model ensembling

In the HW1, I apply above methods to make my predictions more accurate, I will explain them in detail in the following part: **Methodology**

Methodology:

This part, I simply introduce:

1.How the methods be performed in code: **How**

2.The goal of performing the methods**: Goal**

About how these methods actually influence the model's accuracy, I will specify it in part Experimental Results.

Data Preprocessing(Normalization)::

How:

Do data normalization on the image data, to let them have specific values on mean and std, in code it's: **mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]**.

Goal:

Data normalization is beneficial to our training, based on the suggestions of some experts.

Data Augmentation:

How: Do

1. Random Horizontal flip, with probability of 0.5
2. Random Crop the resized (300,300) image to size (270,270)
3. Random flip the image in the range 0 degree to 30 degrees.

You can check the **transform** variable in train.py or check **README.md Training part** for detail

Goal:

Generate a lot more extra train data to train my model.

Proper hyperparameters setting(Tuning):

How:

Try diffirent values on learning-rate, scheduler step, gamma value, batch_size, and observe after training process, in which combination the model has the best average performance on validate-part. Finally I use: **learning rate = 0.02, scheduler step_size = 8, gamma value = 0.3, and batch_size = 10** in code.

Goal:

To choose a set of proper hyperparameters to train our models.

Transfer learning:

How:

Use pre-trained model as basic model, and train the basic model with our train-part data. I use **pretrained resnet50** as basic model in code.

Goal:

We expect a better prediction accuracy by training based on a pretrained model, rather than untrained ones.

Model ensembling:

How:

Bagging and Random forest can be performed, I choose Bagging, that is, train multiple models and make them vote for the prediction result. I use **10-bagging** in code.

Goal:

Make use of multiple models to get a more accurate prediction result.

Experimental Results:

I will show the results in **with** / **without** comparison for each technic below, with short explaination, and I will make a summary for each technic in part: **Summary**

Note that for the **With**, I use mostly the same result because I just canceled out each technic separately to show you the difference between applying and not applying the technics. For example, for **Without** part in **Data Preprocessing**, I used all the technics including **Data Augmentatnion, Transfer learning, etc,** except **Data Preprocessing** itself**.**

1.About **Data Preprocessing(Normalization)**::

With

```
train Loss: 0.2680 Acc: 0.9704
vali
running loss 35.57786047458649
vali Loss: 0.9467 Acc: 0.7467
```

Without

```
train Loss: 0.2303 Acc: 0.9722
vali
running loss 26.080031394958496
vali Loss: 0.9123 Acc: 0.7500
```

Quick view: 10 epoch training nearly the same

2.About **Data Augmentation**:

With

```
train Loss: 0.2680 Acc: 0.9704
vali
running loss 35.57786047458649
vali Loss: 0.9467 Acc: 0.7467
```

Without

```
train Loss: 0.0588 Acc: 0.9981
vali
running loss 13.581867814064026
vali Loss: 1.1653 Acc: 0.6600
```

Quick view: 10 epoch training, big difference about 10% accuracy

3.About **Proper hyperparameters setting(Tuning):**

With

```
train Loss: 0.2680 Acc: 0.9704
vali
running loss 35.57786047458649
vali Loss: 0.9467 Acc: 0.7467
```

Without: Use a learning rate that is 10 times higher than proper one

```
train Loss: 4.9848 Acc: 0.0178
vali
running loss 96.30699157714844
vali Loss: 5.1845 Acc: 0.0100
```

Quick view: 10 epoch training, no converge if without

4.About **Transfer learning:**

With

```
train Loss: 0.2680 Acc: 0.9704
vali
running loss 35.57786047458649
vali Loss: 0.9467 Acc: 0.7467
```

Without: Using only network structure of resnet50 without pretrained

```
train Loss: 4.8470 Acc: 0.0274
vali
running loss 164.0876817703247
vali Loss: 5.9770 Acc: 0.0200
```

Quick view: 10 epoch training, no converge if without

5.About **Model ensembling: Grab from my codalab upload**

With

17	0.729311

Without: Show results on codaLab to compare

7	0.664688

Quick view: about 7% higher if bagging used

Summary:

I will summarize not only each of the technic's influence on the experimental result, but also what I have learned in this homework

Each experiment will be conclude in two parts: 1.**Conclusion 2.Reason**

1.About **Data Preprocessing(Normalization)**:

Conclusion:

Slightly influence.

Reason:

Although data scientists and statistics suggest that we normalize the data beforehand to grab a better result. However, in the experiment, I don't see a great difference between normalized and non-normalized data. Since this is suggested, so I still perform it just in case there's some implicit factors not observed in my sight.

2.About **Data Augmentation**:

Conclusion: **Strongly** influence

Reason:

According to the experiment, we have a 10% accuracy difference between using and not using augmentation. Moreover, I think the result reasonable because we need sufficient data to train our model. For a cifar-10 that classify only 10 classes, the training dataset consists of more than 20000 images. In contrast, we have only 3000 images with 200 classes, 20 times in class number with much less training data. Therefore, if we don't do data augmentation, there's no surprise that we get a non-satisfying result.

3.About **Proper hyperparameters setting(Tuning):**

Conclusion: **Strongly** influence

Reason:

Take learning rate as example

Too Large learning rate leads to the result our model won't converge at all, that is, the loss remains very high and that's because we fix the model **too much** in each step. Every time the model updated, it just 'fly' from one extreme to the other, thus no good result would be observed for sure.

Too Small learning rate takes a great amount of time to train because we fix the model **too little** in each step. In this way, although we fix the model in the right direction, we can hardly see the change.

4.About **Transfer learning:**

Conclusion: Strongly influence

Reason:

In our case, in the same hyperparameter setting and training process, we can observe that a non-pretrained resnet-50 would never converge, while the pretrained one performs quite well after training.

5.About **Model ensembling:**

Conclusion: Strongly influence

Reason:

In math, it is reasonable that if we use bagging technic, we would get a better prediction result. Moreover, in reality, I observe a great improvement about 7% in accuracy using bagging algorithm. This result proves bagging can actually help.

Final: What I have learned in the homework:

Basically I learn a complete process about how to train and pick models, including:

1.How to write the deep-learning code:

Before writing the homework, I have no idea what does the code look like about deep-learning neural network. And now I figure out how to write a basic training and testing program in the field.

2.How to do proper data processing before the training:

Size: I learn to care about the size of the image, At first, I resize it too small, and there was too little information for model too learn, causing a bad result. I fixed it back by resize images to no far away from the original size.

Augment: I realize augmentation is important part in training, especially when we have no sufficient training data.

3.How to pick proper hyperparameters:

It's a hard work to find out good hyperparameters for training. In fact, I try many combinations. Finally I find out that if they're good hyperparameters, I will see a significant loss lowering in first few epochs, and it will converges to a over 95% accuracy in the train-part at last.

4.How to further improve the performance of my model:

After implementing Bagging algorithm, I realize there are a lot more ways to get a better result, as long as we are willing to perform them.

Above is my report for HW1, thanks for the patience reading it all down here!

Have a Nice day!

Sincerely,
Chen.