HomeWork 1

ResNet-Based Model Classification

國立陽明交通大學

NATIONAL YANG MING CHIAO TUNG UNIVERSITY

Prepared by

110550124 林家甫

1 INTRODUCTION

The core idea of this homework is to use the RasNet-Based machine learning model to classify pictures into 100 categories. The training dataset contains 21,024 images, while the validation dataset contains 2,344 images. For training traditional computer vision models, this training dataset isn't quite large. Hence, I decided to use a small but powerful method to train this model, which is ResNeXT 50. Compared to ResNeXT 101, ResNeXT 50 is more suitable to train on this relatively small dataset. Moreover, it has been proven to be outperformed by ResNet. In fact, I have implemented both ResNet and ReNeXT 101 which I put the final result in the additional experiments.

Code Reliability: pep8
GitHub Link: https://github.com/chiafu2018/Image_Classification.git

2 METHOD

I use ResNeXT50 to implement this task, which has been proven that it's a stronger model than ResNet. With limited GPU resources, ResNeXT50 has shown to be my best choice.

- **Data Preprocessing**

For training data, I first resize all the input images so that all images can be in the same size. Then I add flip, rotation techniques to force the model to learn invariant representations, enhancing data diversity. In the end, I normalize the input vectors to let the model converge faster, because all the input vectors are in the same scales. The numbers I used to normalize are the means and standard variations of ImageNet.

For test data, I resize the images into the size same as the size transformation applied in the train set. To make sure that all the images will go through the same resizing and cropping technique. Meanwhile, I skip using flip, rotation techniques because I want to maintain consistency and realism. Rotating and Flipping validation and test images may cause the model to see the unseen patterns and lower the accuracy.

```
transform_train = transforms.Compose([
    transforms.Resize((500, 500)),
    transforms.RandomHorizontalFlip(p=0.2),
    transforms.RandomVerticalFlip(p=0.2),
    transforms.RandomRotation(20),
    transforms.CenterCrop(400),
    transforms.ToTensor(),
```

```
        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224,
0.225])
    ])

    transform_test = transforms.Compose([
        transforms.Resize((500, 500)),
        transforms.CenterCrop(400),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224,
0.225])
    ])
```

- **Model Architecture**

Pretrained Weights:

Like I mentioned before, I use ResNeXT50 to be the backbone of my model and slightly modify the final fully connected output layer. The ReNeXT structure is provided by pytorch library, and it contains 25M parameters. The default output size is 2048 * 1000, which means there will be 1000 classes for prediction. Since we only have 100 classes, I changed the fully connected layer into 2048 * 100. I set batch size to 16, epoch to 50, and learning rate to 25e-5.

```
    net =
models.resnext50_32x4d(weights=models.ResNeXt50_32X4D_Weights.IMAGENET1K_V2)
    num_classes = 100
    net.fc = nn.Sequential(
        nn.Linear(net.fc.in_features, 1024),
        nn.Dropout(0.3),
        nn.Linear(1024, num_classes)
    )
```

- **Hyperparameters**

During the experiment, I found that learning rate plays an important role in model performance. Learning rate on a smaller scale not only achieves a better result in the end, but also converges faster. I guess that is because the size of the dataset is huge, and I update my learning rate after an epoch, which the parameters may already update several times. The scheduler I use is ReduceLROnPlateau, which the learning rate will decrease when the loss is increased or the validation accuracy is decreased.

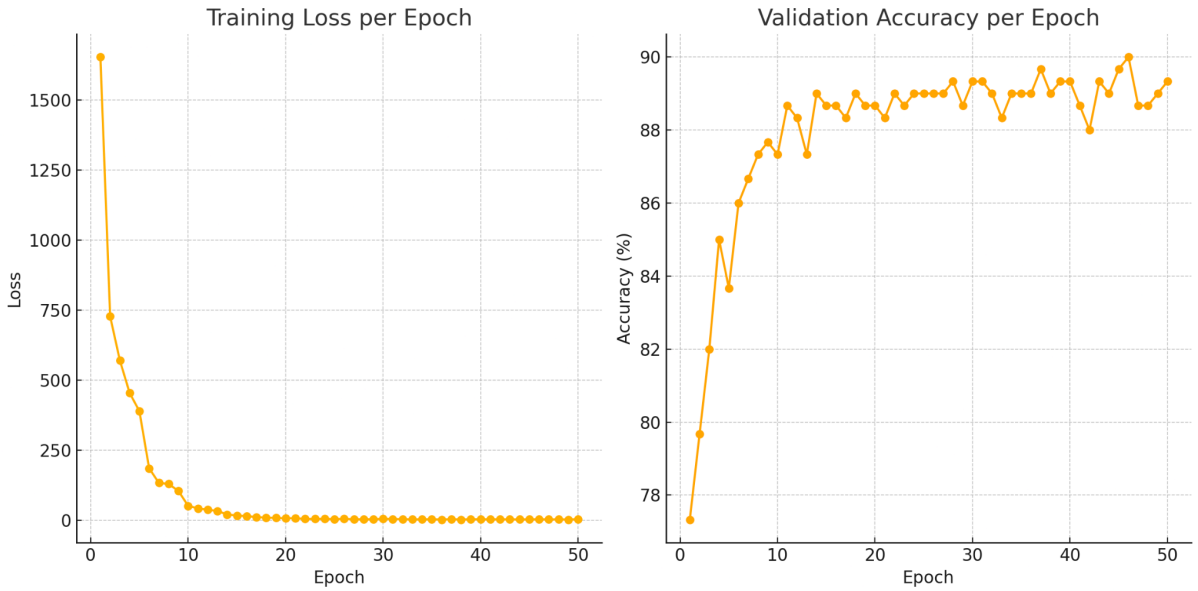
As for the loss function, CrossEntropyLoss is ideal for multi-class classification as it works well with softmax and probabilities. And for the optimizer, AdamW improves optimization by decoupling weight decay, leading to better generalization and stability in training deep models.

```
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.AdamW(net.parameters(), lr=25e-5,
weight_decay=0.0000001)
    scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, factor = 0.5,
patience=3, min_lr=1e-20, mode='max')
```

I save the model every time I finish training. If I find that the loss hasn't converged yet, I will load the saving model and manually set the learning rate into a smaller number than before.

3 RESULTS

The validation accuracy of the last epoch is 89.33%, and the test accuracy achieves 94% on Codabench. The loss I showed below is the total loss from 21024 images.



4 REFERENCES

ResNet: https://arxiv.org/pdf/1512.03385

ResNeXT: https://arxiv.org/pdf/1611.05431

5 ADDITIONAL EXPERIMENTS

- Different ResNet-Based Models
    1. ResNeXT 101

```
    net =
models.resnext101_64x4d(weights=models.ResNeXt101_64X4D_Weights.IMAGENET1K_V1)
```

    2. ResNeXT 101

```
    net = models.resnet50(weights=models.ResNet50_Weights.IMAGENET1K_V2)
```

I have tried different ResNet-Based Models on this dataset, and found that ResNeXT50 achieves the best performance.

- ❖ Accuracy: ResNeXT 50 > ResNeXT101 > ResNet50
- ❖ Training Time: ResNeXt50 < ResNet50 < ResNeXT101

- Additional Layers in Fully Connected Layer

The performance of adding multiple layers isn't as good as I thought before. Moreover, it takes more time to train and more epochs to converge. Maintaining the same ResNeXT50 based architecture, I have gone through different activation functions as well as different layers. As I said before, the performances aren't as good as I mentioned before.

```
    num_classes = 100

    # replace Default Fully Connected Layer, which is 2048 * 1000
    net.fc = nn.Sequential(
        nn.Linear(net.fc.in_features, 1024),
        nn.ReLU(),
        nn.Dropout(0.4),
        nn.Linear(1024, 512),
        nn.Hardshrink(),
        nn.Linear(512, num_classes)
    )
```

- Different type of schedulers & optimizers

```
    scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer,
T_max=EPOCHS)
    optimizer = optim.Adam(net.parameters(), lr=0.0005, weight_decay=0.0001)
```