Visual Recognition using Deep Learning HW1

110550085 房天越

GitHub Repo Link

https://github.com/TianYueh/NYCU-Visual-Recognition-Using-Deep-Learning-2025-HW1/

Introduction

In this project, we aim to address the image classification problem by leveraging deep convolutional neural network to learn image features and ultimately classify images. The core idea is to use a pretrained ResNeXt50 model for transfer learning and enhance the model's generalization ability through various data augmentation techniques. By fine-tuning the fully connected layer, the model is adapted to our dataset which has 100 classes.

Method

I. Data Preprocessing

To improve the accuracy of the model. Data augmentation techniques are applied during training, including:

A. Random Resized Crop, crops the image to 224*224.
B. Random Horizontal Flip
C. Random rotation
D. Random Affine Transformation
E. Color Jitter
F. Normalization, using ImageNet's mean and standard deviation
G. Random Erasing, erases parts of the image by p = 0.5

```
train_transforms = transforms.Compose([
    transforms.RandomResizedCrop(224),
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(15),
    transforms.RandomAffine(degrees=0, translate=(0.1, 0.1)),
    transforms.ColorJitter(brightness=0.3, contrast=0.3, saturation=0.3),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
    transforms.RandomErasing(p=0.5)
])
```

For the validation and testing set, only Resize, CenterCrop, ToTensor and Normalize are applied to ensure the evaluation data is not affected by augmentation.

```python
val_transforms = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])
```

II. Model Architecture

The base model is ResNeXt50_32x4d, with pretrained weights for initialization, which is an evolution of the ResNet architecture. This model introduces the concept of "cardinality" based on the structure of ResNet. With classification of the convolutional computation, it can improve the performance and generalization ability without increasing too much computation.

III. Hyperparameter

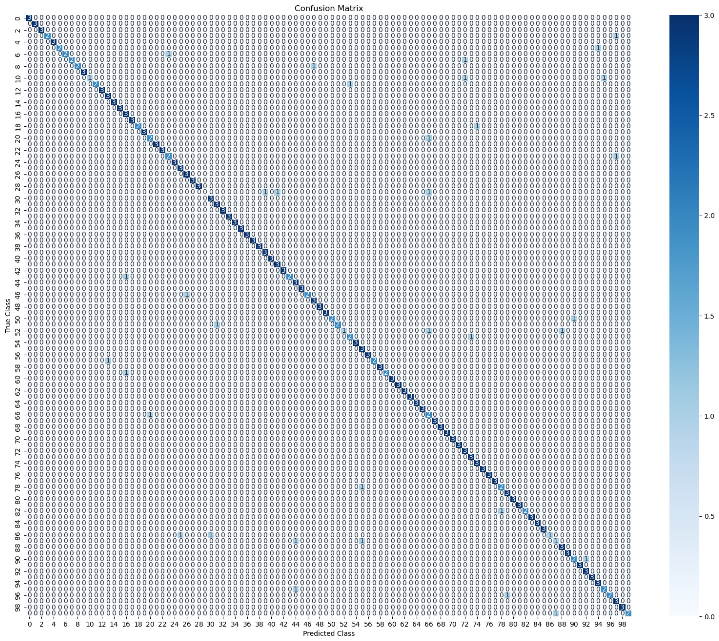
A. Loss Function: CrossEntropyLoss is used to measure the discrepancy between predictions and ground truth labels.
B. Optimizer: Stochastic Gradient Descent (SGD) with momentum set to 0.9 is used to stabilize the update direction and accelerate convergence.
C. Learning Rate: 0.001.
D. Batch Size: 32.
E. Epochs: 25, the best model is saved when a new high accuracy of the valid dataset is observed.

Results

1. My Findings.

   A. Confusion Matrix

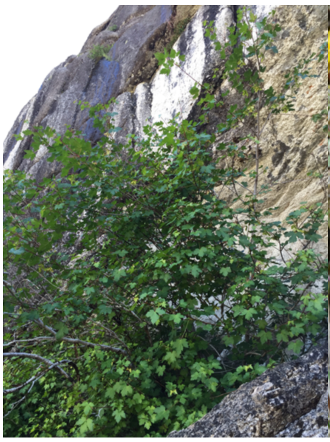
   Here is the confusion matrix to the validation data after training by 25 epochs:



In this confusion matrix, we can observe that the diagonal is quite clear, indicating that the final accuracy is high.

However, some of the classes (class 10, 29, 52, 86, 87) have low accuracy, even for class 29, none of the images are classified correctly.

Here are two images (class 29 and 0, accuracy 0% vs. 100%) in the validation set:




We can see that for the image in class 29, the characteristics are more difficult to catch, thus makes the accuracy than that in class 0.

B. Training Accuracy

Here is the training accuracy for the first few epochs:

```
Epoch 1/25
----------
train: 100%|██████████| 648/648 [05:40<00:00,  1.90it/s, loss=1.73]
train Loss: 2.6826 Acc: 0.4051
val: 100%|██████████| 10/10 [00:01<00:00,  7.08it/s, loss=1.21]
val Loss: 2.0442 Acc: 0.4833
** Model saved at epoch 1 with acc: 0.4833
Epoch 2/25
----------
train: 100%|██████████| 648/648 [05:55<00:00,  1.82it/s, loss=2.1]
train Loss: 1.4033 Acc: 0.6488
val: 100%|██████████| 10/10 [00:01<00:00,  6.60it/s, loss=0.819]
val Loss: 1.2165 Acc: 0.6700
** Model saved at epoch 2 with acc: 0.6700
Epoch 3/25
----------
train: 100%|██████████| 648/648 [05:58<00:00,  1.81it/s, loss=0.49]
train Loss: 1.0857 Acc: 0.7229
val: 100%|██████████| 10/10 [00:01<00:00,  6.02it/s, loss=0.666]
val Loss: 0.9302 Acc: 0.7700
** Model saved at epoch 3 with acc: 0.7700
Epoch 4/25
----------
train: 100%|██████████| 648/648 [06:08<00:00,  1.76it/s, loss=0.924]
train Loss: 0.9313 Acc: 0.7585
val: 100%|██████████| 10/10 [00:01<00:00,  5.53it/s, loss=0.51]
val Loss: 0.8118 Acc: 0.8000
** Model saved at epoch 4 with acc: 0.8000
Epoch 5/25
----------
train: 100%|██████████| 648/648 [05:59<00:00,  1.80it/s, loss=1.01]
train Loss: 0.8323 Acc: 0.7837
val: 100%|██████████| 10/10 [00:01<00:00,  5.80it/s, loss=0.533]
val Loss: 0.7359 Acc: 0.8133
** Model saved at epoch 5 with acc: 0.8133
Epoch 6/25
----------
train: 100%|██████████| 648/648 [06:02<00:00,  1.79it/s, loss=1.09]
train Loss: 0.7754 Acc: 0.7932
val: 100%|██████████| 10/10 [00:01<00:00,  5.97it/s, loss=0.222]
val Loss: 0.7483 Acc: 0.8033
```

We can see that the accuracies for both training and validation dataset grow up fast, and already reached 0.8 in the fifth epoch.

And here is the training accuracy for the last few epochs:

```
Epoch 21/25
----------
train: 100%|██████████| 648/648 [05:36<00:00,  1.93it/s, loss=0.595]
train Loss: 0.2532 Acc: 0.9291
val: 100%|██████████| 10/10 [00:01<00:00,  6.26it/s, loss=0.0722]
val Loss: 0.5475 Acc: 0.8733
Epoch 22/25
----------
train: 100%|██████████| 648/648 [05:28<00:00,  1.97it/s, loss=0.311]
train Loss: 0.2582 Acc: 0.9294
val: 100%|██████████| 10/10 [00:01<00:00,  6.53it/s, loss=0.334]
val Loss: 0.5425 Acc: 0.8733
Epoch 23/25
----------
train: 100%|██████████| 648/648 [05:36<00:00,  1.93it/s, loss=0.218]
train Loss: 0.2480 Acc: 0.9294
val: 100%|██████████| 10/10 [00:01<00:00,  6.42it/s, loss=0.449]
val Loss: 0.5452 Acc: 0.8767
Epoch 24/25
----------
train: 100%|██████████| 648/648 [05:21<00:00,  2.01it/s, loss=0.73]
train Loss: 0.2496 Acc: 0.9299
val: 100%|██████████| 10/10 [00:01<00:00,  6.82it/s, loss=0.41]
val Loss: 0.6012 Acc: 0.8600
Epoch 25/25
----------
train: 100%|██████████| 648/648 [05:22<00:00,  2.01it/s, loss=0.0488]
train Loss: 0.2518 Acc: 0.9300
val: 100%|██████████| 10/10 [00:01<00:00,  6.33it/s, loss=0.464]
val Loss: 0.5902 Acc: 0.8533
Best val Acc: 0.8867
```

We can see that the accuracy doesn't increase a lot and stayed at about 0.875 while the accuracy for training set keeps growing up slowly. This indicates the emergence of overfitting. The highest validation accuracy is in epoch 20.

```
Epoch 20/25
----------
train: 100%|██████████| 648/648 [05:31<00:00,  1.96it/s, loss=0.255]
train Loss: 0.2716 Acc: 0.9236
val: 100%|██████████| 10/10 [00:01<00:00,  6.20it/s, loss=0.156]
val Loss: 0.5175 Acc: 0.8867
```

2. Model Performance

   I tried ResNet50 and ResNeXt50.

   For ResNet50, the accuracy grows significantly slower than ResNeXt50.

   The final accuracy for ResNet50 in valid set is about 0.84, while that for ResNeXt50 goes to about 0.887.

References

- Xie, S., Girshick, R., Dollár, P., Tu, Z., & He, K. (2017). **Aggregated Residual Transformations for Deep Neural Networks.** (ResNeXt paper)
  https://arxiv.org/abs/1611.05431
- PyTorch Official Documentation and torchvision Model Library
  https://pytorch.org/vision/stable/models.html
- ChatGPT
  https://chatgpt.com/

Additional Experiments

Due to the limit of time, I was unable to run the training process for too many times, so I did not really implement additional experiments. However, here are some methods that might work.

1. Using Other Optimizers
   In my training code, SGD with momentum is used because it's widely recognized to perform well on such tasks.
   However, some other optimizers might also work:
   A. AdamW
      Unlike standard Adam, AdamW decouples weight decay from the gradient update, allowing for more effective regularization and improved model generalization.
   B. Ranger
      Ranger combines the benefits of RAdam (Rectified Adam) with those of the Lookahead optimizer. It features adaptive learning rate adjustments along with the stable update characteristics of Lookahead, potentially yielding better results in certain scenarios.
2. Specializing On Poorly Performed Classes
   As observed in the confusion matrix, the model performs especially bad in some classes, we can probably train on these classes, and make the model recognize the details of those classes, which might improve the accuracy.