Selected Topics in Visual Recognition using Deep Learning, Homework 1

111550061, 邱冠崴

Part. 1, Introduction

In this assignment, our task is to classify images into one of 100 categories, aiming for the highest possible accuracy. As a constraint, we must use ResNet as the backbone. Initially, I chose ResNet-101 as my model, but it only achieved 87% accuracy in testing data, which was not satisfactory.

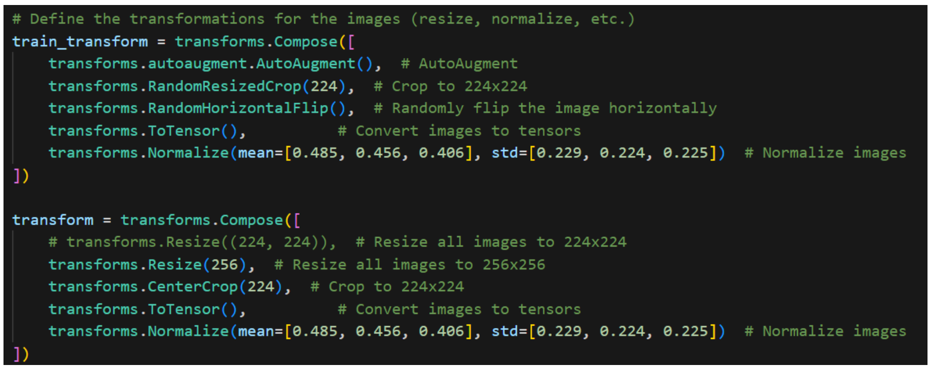
To improve accuracy, I explored various ResNet variants. After conducting multiple experiments, I decided to use ResNeXt-101. My findings showed that ResNeXt-101 (32×8d) and ResNeXt-101 (64×4d) improved accuracy to 90% and 93% in testing data, respectively.

Part. 2, Method

(1) Data Pre-processed:

For training data, I use transforms.autoaugment.AutoAugment() with the default ImageNet policy to apply geometric transformations, as well as color and contrast adjustments. After that, I apply transforms.RandomResizedCrop(224) to randomly resize and crop the images to match the input size required by ResNeXt. Subsequently, I use transforms.RandomHorizontalFlip() with a probability of 0.5 to randomly flip images horizontally, enhancing data diversity. Finally, convert them into tensors and normalize them.

For validation and test data, I first resize the images to 256×256 and then crop the center 224×224 region. After that, I convert them into tensors and normalize them.



```python
# Define the transformations for the images (resize, normalize, etc.)
train_transform = transforms.Compose([
    transforms.autoaugment.AutoAugment(),  # AutoAugment
    transforms.RandomResizedCrop(224),  # Crop to 224x224
    transforms.RandomHorizontalFlip(),  # Randomly flip the image horizontally
    transforms.ToTensor(),          # Convert images to tensors
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])  # Normalize images
])

transform = transforms.Compose([
    # transforms.Resize((224, 224)),  # Resize all images to 224x224
    transforms.Resize(256),  # Resize all images to 256x256
    transforms.CenterCrop(224),  # Crop to 224x224
    transforms.ToTensor(),          # Convert images to tensors
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])  # Normalize images
])
```

(2) Model architecture and hyperparameters:

Model backbone	ResNeXt-101 (64x4d)
Number of total model parameters	81,611,172
Number of trainable model parameters	81,217,956
Optimizer	Adam
Learning rate	8*10^{-5}

Model architecture:

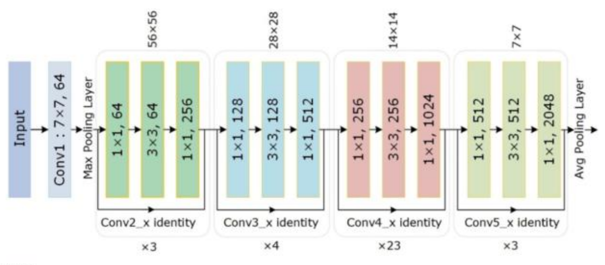


FIGURE 2
The architectural view of ResNet-101.

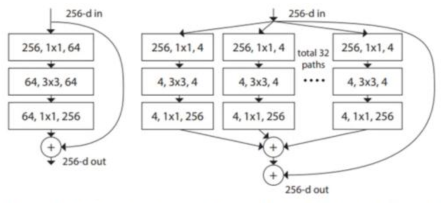


Figure 1. **Left**: A block of ResNet [14]. **Right**: A block of ResNeXt with cardinality = 32, with roughly the same complexity. A layer is shown as (# in channels, filter size, # out channels).

The image on the left illustrates the architecture of ResNet-101, where each residual block contains standard 3×3 convolutions. In ResNeXt, these convolutions are split into multiple smaller groups. The number of groups is called cardinality in the original paper. The image on the right demonstrates a ResNeXt architecture with a cardinality of 32, meaning 3×3 convolution is divided into 32 groups, each processing a subset of input channels.

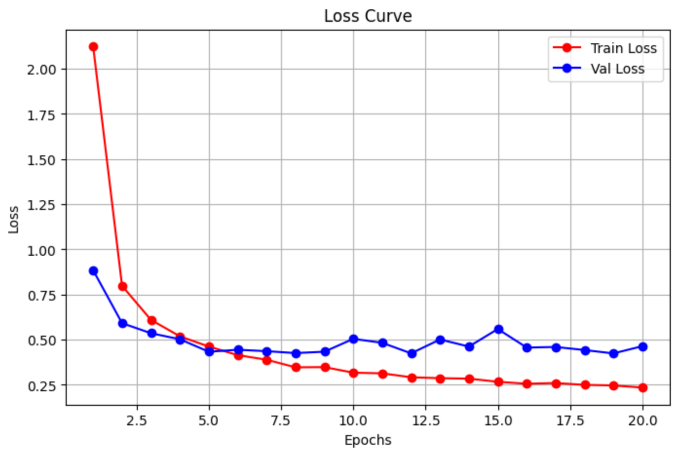
In the backbone I used, ResNeXt-101 (64×4d), every 3×3 convolution in ResNet is split into 64 groups, with each group handling 4 channels. This grouped convolution approach allows the network to learn a more diverse set of features, ultimately improving accuracy.

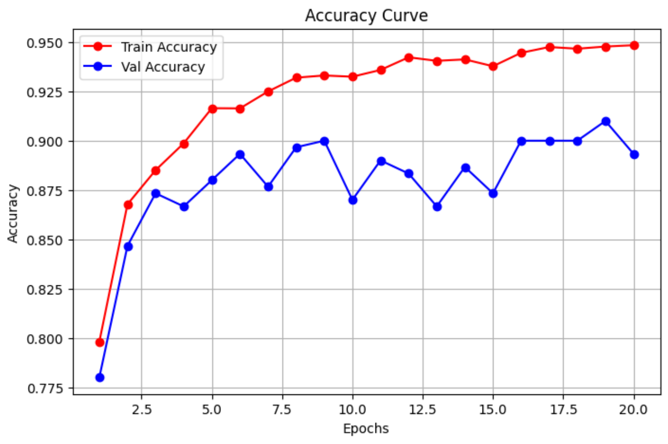
Train strategy:

I found that fine-tuning the entire ResNeXt-101 model made it prone to overfitting on the training dataset, resulting in lower accuracy on the validation and test data. In order to mitigate this issue and ensure the model captures more generalizable features, I froze the first layer of ResNeXt-101. That is why the number of trainable model parameters is lower than the total number of parameters. As a result, the test accuracy increased from 92% to 93%.

Part. 3, Results

(1) Training curve



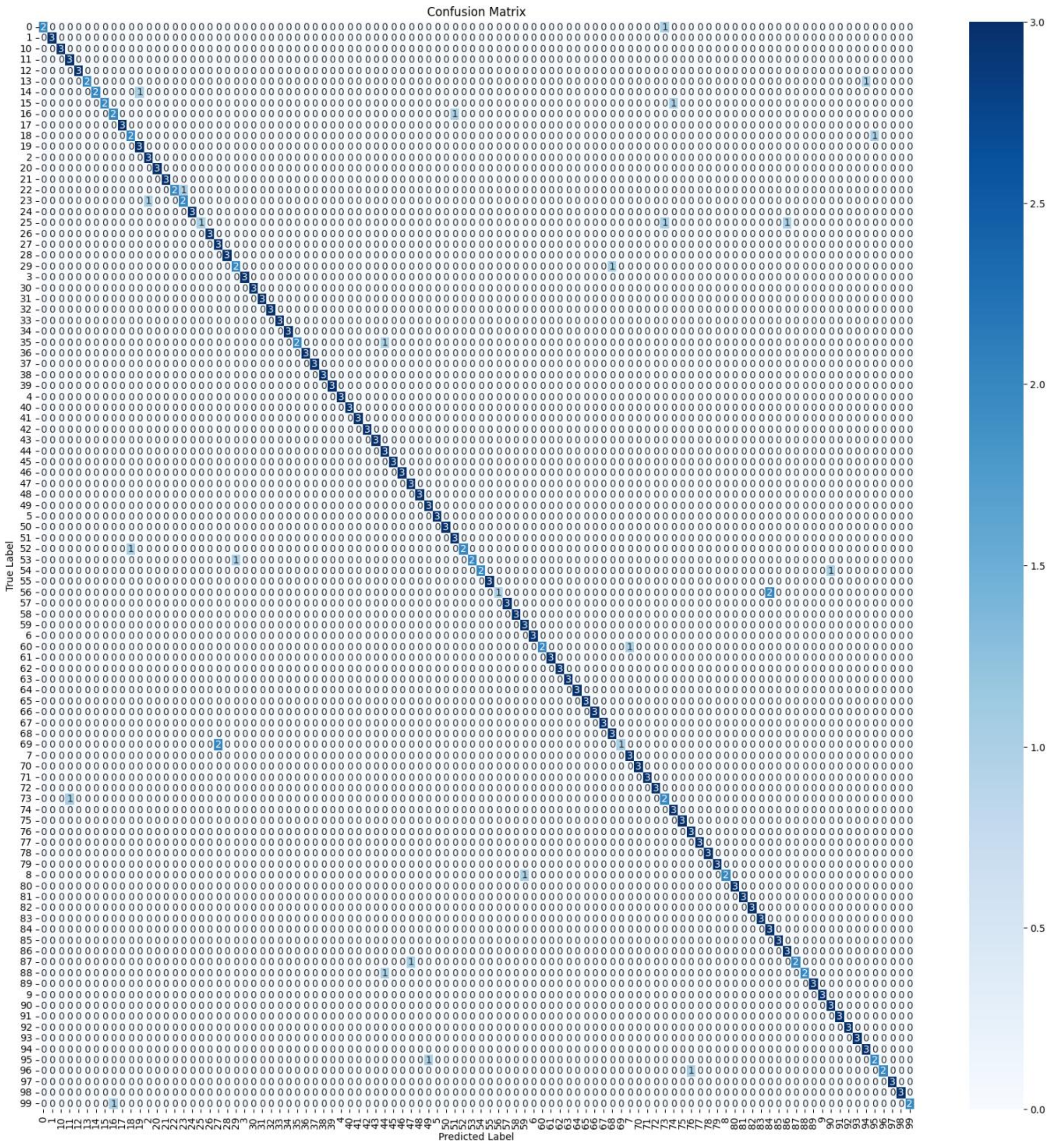


First, we can see the graph on the left, which shows the loss curves for training and validation. The training loss (red) decreases sharply and stabilizes at a low value. As for the validation loss (blue), while it also decreases initially, it begins to fluctuate after a few epochs.

Next, look at the graph on the right. It shows the accuracy curves for training and validation. The training accuracy (red) rises stably and reaches near 95%, while validation accuracy (blue) fluctuates around 90%. The widening gap between them further suggests overfitting, meaning the model performs well on training data but does not generalize perfectly.

The final accuracy I achieved on the test data is 93%. While this is a strong result in classification, I believe there is still room for improvement. In my opinion, the key to further enhancement lies in how to reduce overfitting and make the model more generalizable.

(2) Confusion matrix:



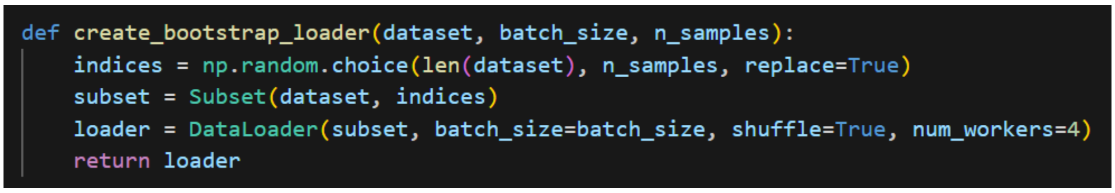
Part. 4, References

Paper: [Aggregated Residual Transformations for Deep Neural Networks]

Part. 5, Additional experiments

In addition to freezing the first layer of ResNeXt-101, I want to explore other methods to mitigate overfitting and enhance the model's generalization ability. To achieve this, I plan to conduct an experiment to evaluate whether an ensemble method, specifically bagging, can effectively address this issue and improve accuracy.

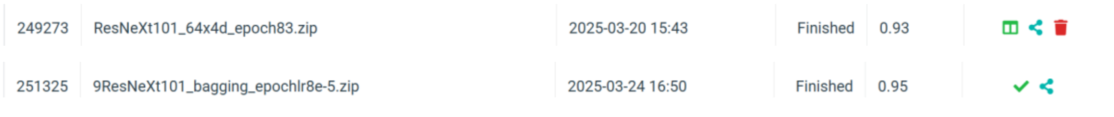
In my experiment, I want to train nine ResNeXt-101 models, each for 20 epochs, using the following function to get random subsets of the training dataset. By introducing diversity among the models, I guess that it can reduce variance and improve overall performance through ensemble learning.



```python
def create_bootstrap_loader(dataset, batch_size, n_samples):
    indices = np.random.choice(len(dataset), n_samples, replace=True)
    subset = Subset(dataset, indices)
    loader = DataLoader(subset, batch_size=batch_size, shuffle=True, num_workers=4)
    return loader
```

As a result, my test accuracy improved from 93% to 95%, demonstrating that bagging is an effective method for enhancing the model's performance and robustness while preventing it from focusing too much on specific details of the training data.

(ResNeXt101_64x4d_epoch=8(3).zip is the prediction from single ResNeXt101 model. 9ResNeXt101_bagging_epoch=20_lr=8e-5 is the prediction from nine ResNeXt101 models trained by different subset of training data.)

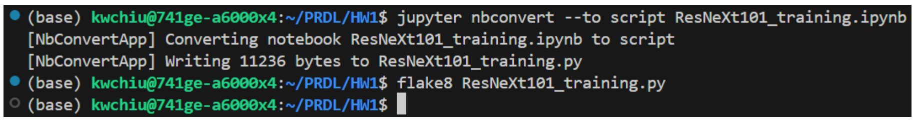


249273	ResNeXt101_64x4d_epoch83.zip	2025-03-20 15:43	Finished	0.93	
251325	9ResNeXt101_bagging_epochlr8e-5.zip	2025-03-24 16:50	Finished	0.95	✓

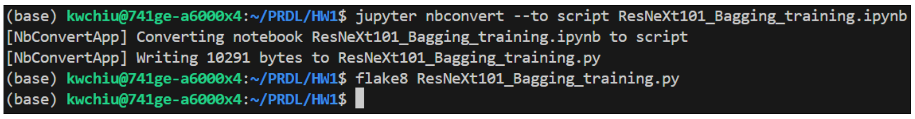
Part. 6, Code Reliability

(1) Lint the code:

Since I am using a .ipynb file, which does not support Flake8 directly, I first convert it to a .py file before running Flake8 for code checks.



```
(base) kwchiu@741ge-a6000x4:~/PRDL/HW1$ jupyter nbconvert --to script ResNeXt101_training.ipynb
[NbConvertApp] Converting notebook ResNeXt101_training.ipynb to script
[NbConvertApp] Writing 11236 bytes to ResNeXt101_training.py
(base) kwchiu@741ge-a6000x4:~/PRDL/HW1$ flake8 ResNeXt101_training.py
(base) kwchiu@741ge-a6000x4:~/PRDL/HW1$
```



```
(base) kwchiu@741ge-a6000x4:~/PRDL/HW1$ jupyter nbconvert --to script ResNeXt101_Bagging_training.ipynb
[NbConvertApp] Converting notebook ResNeXt101_Bagging_training.ipynb to script
[NbConvertApp] Writing 10291 bytes to ResNeXt101_Bagging_training.py
(base) kwchiu@741ge-a6000x4:~/PRDL/HW1$ flake8 ResNeXt101_Bagging_training.py
(base) kwchiu@741ge-a6000x4:~/PRDL/HW1$
```

(2) GitHub: HW1_LINK