Visual Recognition using DL HW1 Report

Author: 司徒立中 (111550159) | [GitHub Link]

Introduction

In this task, the dataset consists of numerous categories of plants, insects, and birds, containing a total of 21,024 and 2,344 images for training/validation and testing, respectively. Additionally, there are constraints, including the prohibition of external data and the requirement to use ResNet [1] as the model backbone, though modifications are allowed. From my perspective, the work can be divided into three parts: model selection, data efficiency techniques, and ensemble methods.

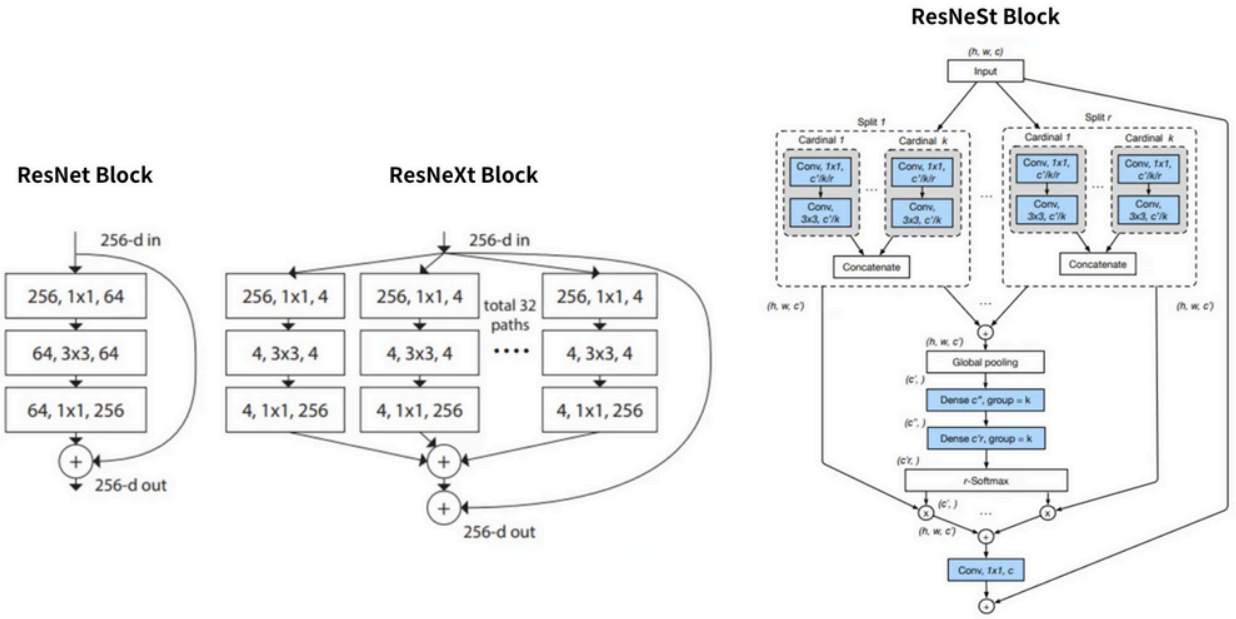
To begin with, model selection is crucial as it significantly impacts the highest achievable performance in this task. Instead of directly using ResNet50 or deeper ResNet variants, more advanced architectures should be considered, such as ResNeXt [2] or ResNeSt [4]. It is worth noting that ConvNeXt [5] has been banned by the TA; therefore, it will not be discussed here.

On top of that, data efficiency techniques are particularly useful when working with a limited dataset. Rather than independently searching for related works, I referred to DeiT [8] to explore modern, powerful strategies such as RandAugment [9], MixUp [10], and CutMix [11]. Additionally, I utilized transform.v2 [16] to easily implement these techniques, which significantly reduced development time and resulted in cleaner code.

Last but not least, in the pursuit of higher performance, the importance of ensemble methods cannot be overstated. A common approach is cross-validation, which partitions the original dataset into multiple folds for training different models. When making predictions, we can aggregate the probabilities of each category from multiple models to obtain a more robust

result. Furthermore, I adopted a technique from Model Stock [14] called Periodic Merging, the key idea of which will be discussed in detail later.

Method



In this work, we tested both ResNeXt101 and ResNeSt200 based on the ResNet architecture. ResNeXt introduces "cardinality" by splitting features into multiple parallel branches (group convolutions) within each residual block, improving representational power without drastically increasing parameters. ResNeSt builds on ResNeXt by adding a "Split Attention" module, which recalibrates feature channels within each branch, leading to stronger performance. Ultimately, ResNeSt200 outperforms ResNeXt101, so we chose ResNeSt200 for our final model.

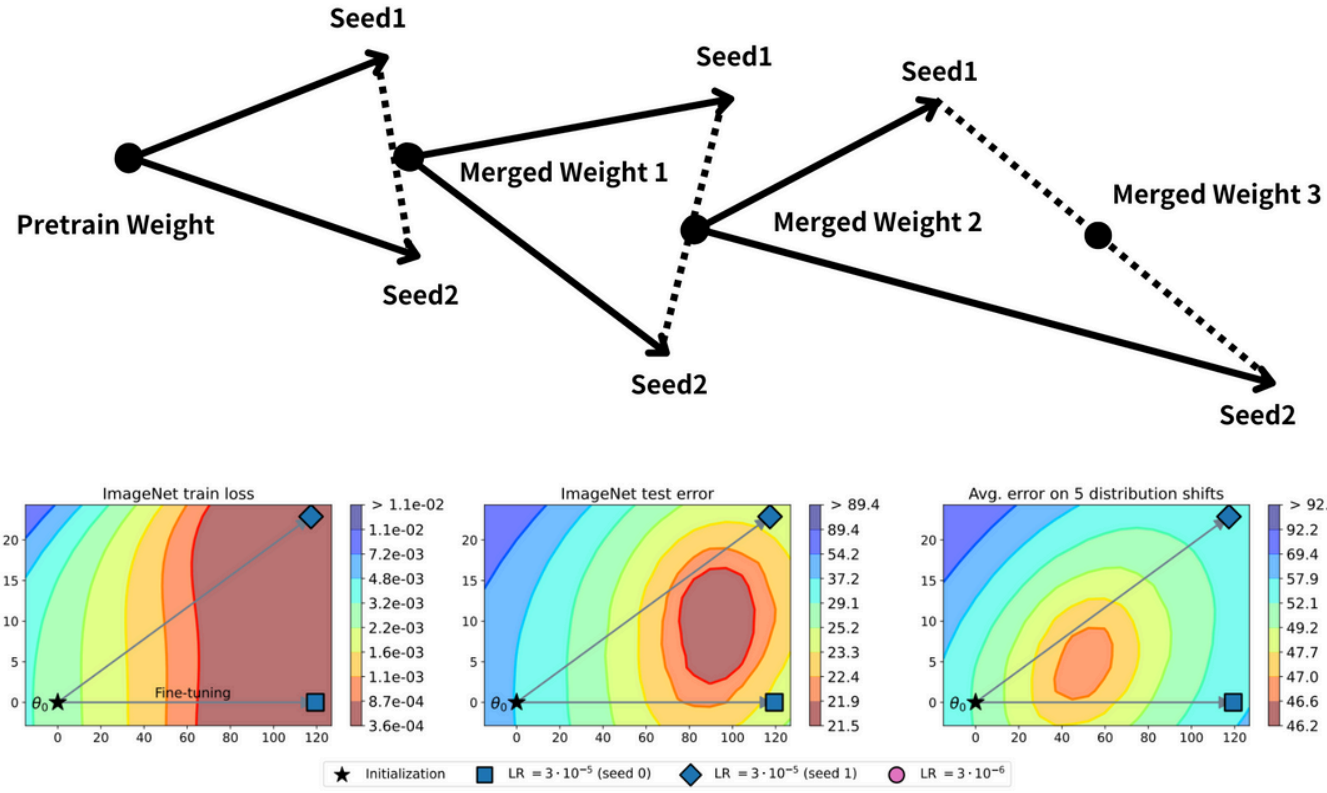
	Seed	Epoch	Patience	Batch	Base LR	Head LR	Weight Decay	Folds
hyper-params	111550159	40	10	64	1e-4	1e-3	1e-5	10

	Resize Crop	Horizon-tal Flip	Color Jitter	Gray scale	Normalize	MixUp	CutMix
train	✓	✓	✓	✓	✓	✗	✓
val/test	✓	✗	✗	✗	✓	✗	✗

Additionally, instead of training the model from scratch, I utilized pretrained weights initially trained on ImageNet. Furthermore, to achieve better fine-tuning performance, I applied different learning rates to the base parameters and the head (fully connected layer) parameters. It is

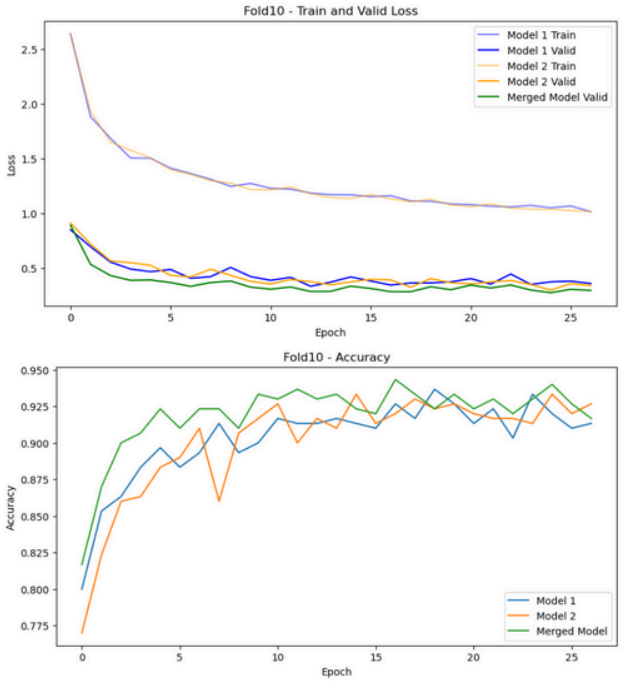
crucial to maintain a lower learning rate for feature extraction parameters to preserve prior knowledge while simultaneously aligning the learned patterns with this dataset.

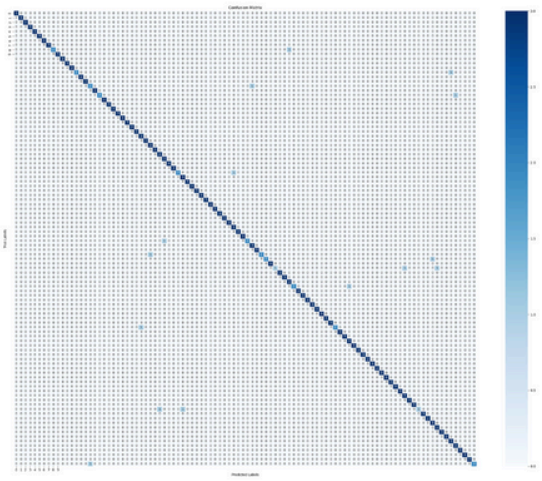
In addition, to enhance dataset diversity without using external data, I apply various data augmentation techniques, as shown in the table. In the training dataset, the labels/classes remain unaffected by basic image processing techniques such as horizontal flipping, color jittering, and so on. Compared to the training dataset, the transformations applied to the validation and test sets preserve the original information.



Finally, ensemble methods are essential for improving performance. In my approach, I utilize not only cross-validation but also periodic merging. In periodic merging, models with the same initial weights are trained using different random seeds. As we can observe from the error landscape, fine-tuned weights may face issues of instability or suboptimal convergence. To address this problem, we can simply average the weights to achieve lower loss and higher performance. This process is repeated in each epoch. According to the strategies introduced in the paper, this approach effectively balances generalization and robustness. (However, in my implementation, I only average the weights instead of applying the formula used in model stock)

Results





These two line charts show the training loss, validation loss, and validation accuracy. Since more data augmentations are applied to the training dataset, its loss is higher than that of the validation dataset. Additionally, the loss of the merged model is lower than that of both model 1 and model 2 for most of the time, indicating that periodic merging is effective for this task. On the right is the confusion matrix computed on the validation dataset, which achieves an accuracy of 0.9500 and exhibits a prominent diagonal pattern. On the public leaderboard, the accuracy reaches 0.98 using these methods.

References

[1] (CVPR 2016) Deep Residual Learning for Image Recognition

[2] (CVPR 2017) Aggregated Residual Transformations for Deep Neural Networks

[3] (CVPR 2018) Squeeze-and-Excitation Networks

[4] (CVPR 2020) ResNeSt: Split-Attention Networks

[5] (CVPR 2022) A ConvNet for the 2020s

[6] (NIPS 2017) Attention Is All You Need

[7] (ICLR 2021) An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale

[8] (ICML 2021) Training data-efficient image transformers & distillation through attention

[9] (NIPS 2020) RandAugment: Practical automated data augmentation with a reduced search space

[10] (ICLR 2018) mixup: Beyond Empirical Risk Minimization

[11] (ICCV 2019) CutMix: Regularization Strategy to Train Strong Classifiers with Localizable Features

[12] (ICLR 2020) AugMix: A Simple Data Processing Method to Improve Robustness and Uncertainty

[13] (ICML 2022) Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time

[14] (ECCV 2024) Model Stock: All we need is just a few fine-tuned models

[15] Pytorch - Models and pre-trained weights

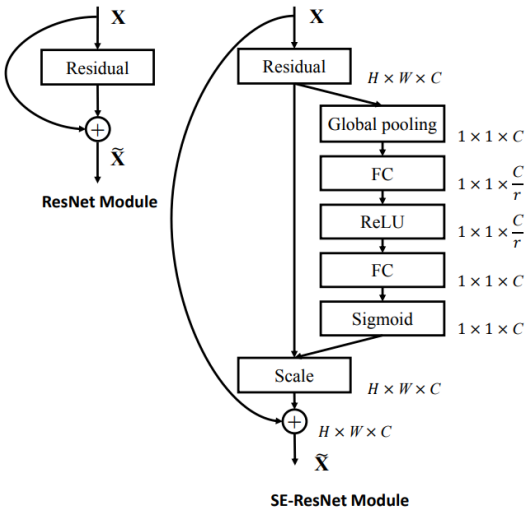
[16] Pytorch - Getting started with transforms v2

[17] ResNeSt - Github Implementation

Additional experiments

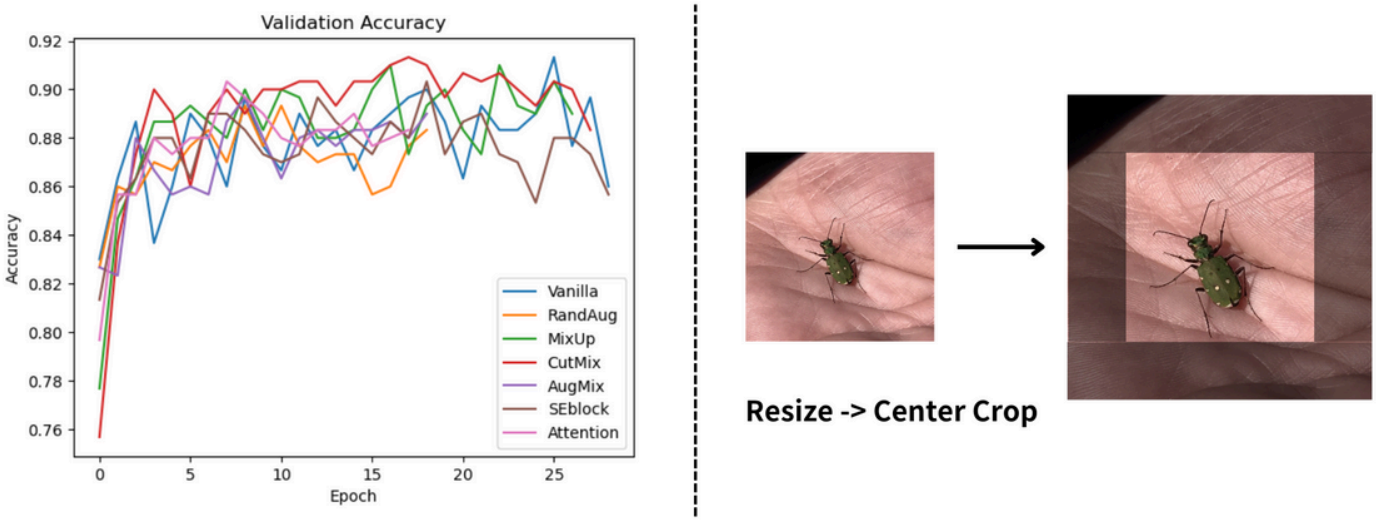
Before introducing the additional experiments, it is important to understand the different techniques used here. For data augmentation, I utilized RandAug, MixUp, CutMix, and AugMix.

First of all, RandAug andomly applies a set of predefined augmentations (e.g., rotation, shifting, brightness adjustments) with random intensity, reducing manual tuning and improving generalization. Secondly, MixUp linearly blends two images (pixel-wise) and mixes their labels proportionally, helping the model generalize by smoothing labels and reducing overfitting. Furthermore, CutMix cuts a random patch from one image and pastes it onto another, mixing labels based on the patch area. This preserves more spatial information than MixUp. Last, AugMix, combines multiple augmentations in parallel or sequentially and merges the results (often with a consistency loss), increasing robustness and diversity of training samples.



In addition to data augmentation, I attempted to change layers and used SE-Block as well. In the SE-Block, a global average pooling operation first "squeezes" each feature map into a single value, summarizing its global information. Next, two fully connected (FC) layers with a ReLU in between learn to "excite" or reweight the channels by producing a set of attention coefficients. Finally, these

coefficients are passed through a sigmoid function and multiplied back ("scaled") with the original feature maps, allowing the network to emphasize more informative channels.



At the beginning of experiments, I attempted to use ResNeXt with various strategies. In terms of data augmentation, I tried RandAug, MixUp, CutMix, and AugMix. While these methods allow the model to learn more complex patterns from the training dataset, only CutMix consistently improved validation performance. This suggests that the original data is already complex enough, or that the model's capacity is insufficient to effectively learn from heavily transformed images. Therefore, I decided to continue using only CutMix.

Meanwhile, I explored different architectures for the final layer (fully connected layer), such as SE-Block and Multi-Head Attention. Surprisingly, their performance was worse compared to a standard fully connected layer. One possible reason is that the dataset may not have significant channel-wise dependencies, making SE-Block less effective. Additionally, Multi-Head Attention could introduce unnecessary complexity, leading to overfitting or inefficient feature extraction in this particular task.

Rather than solely focusing on improving the model's performance, it is also beneficial to reduce the difficulty of the prediction task. As a result, based on the dataset properties, I noticed that some images contain only small parts of objects. In other words, applying center cropping can help the model focus on the central objects more effectively. In my experiments, this approach increased accuracy by 1%.

Code Reliability

```
(env) PS C:\Users\Rain sytwu\Desktop\NYCU\NYCU_3-2\DLCV\HW1> flake8 .\dataloader.py
(env) PS C:\Users\Rain sytwu\Desktop\NYCU\NYCU_3-2\DLCV\HW1> flake8 .\inference.py
(env) PS C:\Users\Rain sytwu\Desktop\NYCU\NYCU_3-2\DLCV\HW1> flake8 .\main.py
(env) PS C:\Users\Rain sytwu\Desktop\NYCU\NYCU_3-2\DLCV\HW1> flake8 .\model_stock.py
(env) PS C:\Users\Rain sytwu\Desktop\NYCU\NYCU_3-2\DLCV\HW1> flake8 .\models.py
(env) PS C:\Users\Rain sytwu\Desktop\NYCU\NYCU_3-2\DLCV\HW1> flake8 .\optim.py
(env) PS C:\Users\Rain sytwu\Desktop\NYCU\NYCU_3-2\DLCV\HW1> flake8 .\trainer.py
(env) PS C:\Users\Rain sytwu\Desktop\NYCU\NYCU_3-2\DLCV\HW1> flake8 .\utils.py
(env) PS C:\Users\Rain sytwu\Desktop\NYCU\NYCU_3-2\DLCV\HW1>
```