Homework 1 Image classification

1 Introduction

In this assignment, I aim to train a deep learning model for image classification using a dataset provided for the HW1 Image Classification Problem. My goal is to develop a model that can accurately predict the class of given test images.

To achieve this, I experimented with different model architectures, including ResNet-50, ResNet-101, ResNeXt-50, and ResNeXt-101. Additionally, I explored various fully connected (FC) layer structures and fine-tuned the learning rate along with the hyperparameters of the CosineAnnealingLR scheduler, such as T_max and eta_min, to optimize performance. After extensive evaluation, I found that ResNeXt-101, combined with a well-tuned learning rate and CosineAnnealingLR scheduler, achieved the best results. I also applied data augmentation techniques to enhance training robustness and implemented Test-Time Augmentation (TTA) to further improve the model's generalization ability.

The dataset consists of training, validation, and test sets. The model is trained using a cross-entropy loss function, and performance is evaluated based on classification accuracy. My final best model achieves an accuracy of 93% on the validation set and 96% on the public test set, demonstrating its strong generalization capability. I expect to obtain competitive results on the final test set.

Github: https://github.com/vayne1125/NYCU-Visual-Recognitionusing-Deep-Learning/tree/main/HW1_Image-Classification-Problem

2 Method

The parameters I selected are based on the results of experiments, and detailed information can be found in "3 Experiments".

2.1 Pre-process data

I first analyzed the test and train data and noticed that the training data contains many special cases, such as images where a person's hand occupies 50% of the frame, while the object to be identified is very small. In contrast, the test data generally contains objects centered in the frame, which is a more typical case. Based on this observation, I concluded that data augmentation is necessary for the training data to prevent the model from overfitting. I applied the following augmentation techniques to the training data, executing them randomly:

- ✧ RandomResizedCrop(224)
- ✧ RandomHorizontalFlip()
- ✧ ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.1)
- ✧ Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])

- ✧ RandomRotation(10)
- ✧ RandomAffine(degrees=0, translate=(0.1, 0.1))

The following figure show the result:

Additionally, I applied Test-Time Augmentation to enable the model to make predictions based on the style learned during training. Each image undergoes the following transformations, and the average softmax is calculated at the end:

- ✧ RandomHorizontalFlip(p=1.0)
- ✧ RandomRotation(10)
- ✧ RandomResizedCrop(224, scale=(0.8, 1.0))

2.2 Model architecture

Since directly dividing the results into 100 classes caused significant overfitting, I added dropout and initially divided the output into 500, then into 100, using a stepwise learning strategy. Other strategies were also tried in the third section "Experiments", and this was the best result.

- ✧ Backbone: ResNeXt-101
- ✧ Pre-trained Weights: IMAGENET1K_V2
- ✧ Last fully connected layer (fc) was modified as follows:

```
nn.Sequential(
    nn.Linear(num_ftrs, 500),
    nn.ReLU(),
    nn.Dropout(0.5),
    nn.Linear(500, class_number)
)
```

✧ Total Parameters: 87,816,936 (about 87M)

2.3 Hyperparmeters

Parameter	Value
Epochs	100
Batch Size	64
Learning Rate	0.0001
Weight Decay	0.001
Optimizer	AdamW
Criterion	nn.CrossEntropyLoss()

2.4 Learning Rate Scheduler

Based on the experiments, I used CosineAnnealingLR with T_max = 80 and eta_min = 0.00001.

2.5 Additional Details

To prevent overfitting and save time, I used Early Stopping with a value of epoch // 50.

3 Experiments

I divided the experiments into six major categories: model depth experiments, model pretraining parameter experiments, model architecture experiments, hyperparameter tuning (learning rate and CosineAnnealingLR parameters) experiments, other experiments, and ResNeXt101 experiments. Due to hardware limitations (RTX 3060 Ti / 8GB), I primarily conducted the first five experiments using ResNet-50 and ResNeXt-50, aiming to find the optimal parameter combinations.

For the sixth experiment, based on my findings from the first five, I hypothesized that ResNeXt-101 could achieve significantly better performance. To test this, I utilized the best parameters from the previous experiments and borrowed an A6000 (48GB) and a 3090 (24GB) from my peers to run the experiment.

Additionally, due to my lack of experience, I initially did not set a random seed, which may have introduced slight variations in the results. The first four experiments were conducted without applying Test-Time Augmentation (TTA), while the fifth experiment included tests incorporating TTA and other techniques.

3.1 Model Depth Experiments

Due to GPU memory limitations, different model depths were paired with different batch sizes to ensure that each epoch remained within four minutes.

ID	Model	Batch size	Trainable layer
R34_64	Resnet34	64	All
R50_64	Resnet50	64	All
R101_32	Resnet101	32	All
R101_64_l2	Resnet101	64	Last 2
R101_64_l1	Resnet101	64	Last 1
X50_32	ResNeXt50	32	All

3.2 Model Pre-training Parameter Experiment

The purpose of this experiment is to determine the most suitable pre-trained weights for fine-tuning. "None" indicates that no pre-trained weights were used.

ID	Pre-trained Weights
W0	None
W1	Default
W2	IMAGENET1K_V2

3.3 Model Architecture Experiment

The goal of this experiment is to explore different fully connected (FC) layer structures to find the most suitable configuration for this task. The structural adjustments were primarily based on intuition—typically deepening the network until underfitting started to occur, at which point modifications were stopped. Additionally, I consulted ChatGPT for common tuning strategies.

ID	Architecture
V0	nn.Dropout(0.5), nn.Linear(num_ftrs, class_number)
V1	nn.Linear(num_ftrs, 500), nn.ReLU(), nn.Dropout(0.5), nn.Linear(500, class_number)
V2	nn.Linear(num_ftrs, 500), nn.ReLU(), nn.Linear(500, 250),

	nn.Dropout(0.5), nn.Linear(250, class_number)
V3	nn.Linear(num_ftrs, 500), nn.BatchNorm1d(500), nn.ReLU(), nn.Linear(500, 250), nn.ReLU(), nn.Dropout(0.5), nn.Linear(250, class_number)
V4	nn.Linear(num_ftrs, 1024), nn.BatchNorm1d(1024), nn.ReLU(), nn.Dropout(0.5), nn.Linear(1024, 512), nn.BatchNorm1d(512), nn.ReLU(), nn.Dropout(0.4), nn.Linear(512, 256), nn.BatchNorm1d(256), nn.ReLU(), nn.Dropout(0.3), nn.Linear(256, class_number)

3.4 Hyperparameter Tuning Experiment

This experiment aims to test whether a fixed learning rate or a scheduled learning rate performs better. The scheduling method chosen is Cosine Annealing (recommended by GPT).

ID	Learning rate
Lr_fixed (w/o CosineAnnealingLR)	0.0001
Lr_C50 (w/CosineAnnealingLR)	0.0001 / eta_min = 0.00005 / T_max = 50
Lr_C80 (w/CosineAnnealingLR)	0.0001 / eta_min = 0.00005 / T_max = 80

3.5 Other Experiments

This section includes additional tests related to model architecture and data preprocessing.

3.5.1 Dropout Experiment

Different dropout values were tested to evaluate their impact.

ID	Dropout Value
D0.25	0.25
D0.5	0.5

3.5.2 TTA transformations Experiment

Different Test-Time Augmentation (TTA) strategies were tested.

ID	TTA transformations
w/ TTA	With TTA transformations
w/o TTA	Without TTA transformations

3.6 ResNeXt101 Experiment

Based on the best results obtained from sections 3.1 – 3.5, the optimal combination of parameters was selected to train ResNeXt101. The only difference in this experiment is the model architecture, while all other parameters remain the same:

Category	Corresponding ID	Value
Model	X101_64	ResNeXt101 / batch size = 64 / train all layers
Pretrained Weight	W2	IMAGENET1K_V2
Learning rate	Lr_C80	0.0001 / eta_min = 0.00005 / T_max = 80
TTA transformations	w/ TTA	With TTA transformations
Dropout	D0.5	0.5

Training Architectures:

ID	Model Architecture Used
X101_V0	V0
X101_V1	V1
X101_V2	V2
X101_V3	V3

4 Results

This section presents the experimental results. Since this is my first time conducting deep learning-related experiments, some aspects may not have been thoroughly considered, and the completeness of the experiments may be limited.

4.1 Model Depth Experiment Results

The table below shows the validation and test accuracy for different model configurations. The reported results are based on the most frequently observed values. Since the validation accuracy of R34_64 was too low, I did not proceed with testing its accuracy on the test set.

ID	Validation Accuracy	Test Accuracy
R34_64	0.86	-
R50_64	0.90	0.92
R101_32	0.88	0.92
R101_64_l2	0.89	0.93
R101_64_l1	0.89	0.92
X50_32	0.91	0.94

Based on the table, the following observations can be made:

- **R34_64 v.s R50_64:** A deeper model achieves better performance. (I was unable to test R101_64 due to hardware limitations.)
- **R50_64 v.s R101_64_l2 v.s R101_64_l1:** Since R50, despite being a simpler model, achieves similar results to R101 without training all layers, it can be inferred that training all layers yields better performance.
- **R50_64 v.s R101_32:** Since R50, despite being a simpler model, achieves comparable results to R101_32, it suggests that a batch size of 64 is more effective.
- **R50_64 v.s X50_32:** The results indicate that ResNeXt outperforms ResNet.

From these observations, the best settings appear to be a **deeper model**, **training all layers**, **a batch size of 64**, and using **ResNeXt**. Based on this, I decided to borrow a GPU from other classmates to train **ResNeXt101**.

4.2 Pretrained Weight Experiment Results

The following table presents the results of using R50_64 under the same conditions but with different pretrained weights. Since W0 achieved a very low validation accuracy, it was not tested on the test set. Based on the results, W2 produced the best performance.

ID	Validation Accuracy	Test Accuracy
W0	0.5	-
W1	0.89	0.92
W2	0.89	0.93

4.3 Model Architecture Experiment Results

Based on the results in Section 4.1, X50_32 exhibited the best inference capability, while R50_64 had the second-best validation accuracy, and R101_64_l2 had the second-best test accuracy. Therefore, these three models were selected for further experiments.

The table below presents the best test accuracy achieved for each architecture after experimenting with different configurations. Additionally, since V4 exhibited signs of underfitting, the X50_32 + V4 experiment was not conducted.

ID	Validation Accuracy	Test Accuracy
R50_64 / W2		
V0	0.92	0.93
V1	0.90	0.94
V2	0.90	0.94
V3	0.90	0.95
V4	0.88	0.93
R101_64_l2 / W2		
V1	0.89	0.93
V2	0.90	0.94
V3	0.88	0.94
V4	0.89	0.93
X50_32 / W2		
V0	0.92	0.94
V1	0.92	0.94
V2	0.90	0.94
V3	0.91	0.94

From the experimental results of R50_64 and R101_64_l2, it can be observed that deepening the model further (V4) led to underfitting. Therefore, I did not attempt even deeper models. Additionally, the R50_64 + V3 architecture achieved the best test accuracy so far. Meanwhile, V0 and V1 performed well with X50_32, and V2 showed better results with R101_64_l2.

4.4 Hyperparameter Tuning Experiment Results

According to the table below, Lr_C50 outperformed Lr_fixed, and Lr_C80 further improved upon Lr_C50.

ID	Validation Accuracy	Test Accuracy
R50_64 / W2 / V0		
Lr_fixed	0.89	0.93
Lr_C50 (w/CosineAnnealingLR)	0.92	0.93
R50_64 / W2 / V1		
Lr_C50 (w/CosineAnnealingLR)	0.90	0.93
Lr_C80 (w/CosineAnnealingLR)	0.90	0.94

4.5 Other Experiments

Using Dropout 0.5 yielded better results, and applying TTA led to an average accuracy improvement of 0.5%.

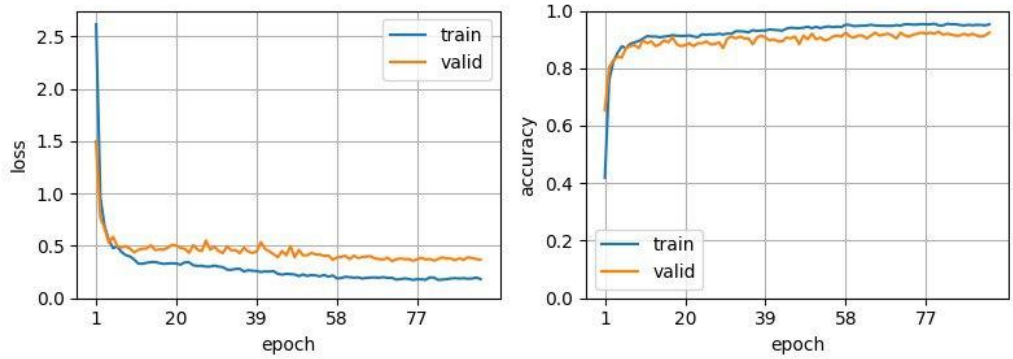
ID	Validation Accuracy	Test Accuracy
X50_32 / W2 / V0		
D0.25	0.91	0.94
D0.5	0.92	0.94
Average Performance Across Models		
w/ TTA	-	+0.5% (avg increase)
w/o TTA	-	0.0

4.6 ResNeXt101 Experiment

ID	Validation Accuracy	Test Accuracy
X101_V0	0.93	0.95
X101_V1	0.94	0.96
X101_V2	0.92	0.94
X101_V3	0.93	0.94

Since V3 achieved the highest accuracy (0.95) on R50_64, I attempted to run ResNeXt101 with V3. However, the results were not as promising as expected, suggesting that the previous high test accuracy may have been coincidental. Instead, V1 demonstrated more consistent performance and achieved better results overall.

Below is the loss and accuracy plot for X101_V1:



The submitted result ranked 12th (with five entries marked as N/A ahead). Although I experimented with many different methods to improve accuracy, none were successful. In total, I trained nearly 50 models. Hopefully, I can achieve a higher ranking next time!

17	313551052	1	2025-03-24 21:43	251521	313551052	0.96

5 References

No papers or GitHub sources were used or referenced—only official documentation was consulted.