313551139 陳冠豪(Gary Chen)

Github: https://github.com/Gary123fff/NYCU-Computer-Vision-2025-Spring-HW1

Introduction

For this Homework, we're building an image classification model .

To achieve good results, I used the pretrained **ResNeXt-101_64x4d** model and applied the following techniques to improve performance:

Model Architecture:

- Uses **torchvision.models.resnext101_64x4d** as the backbone.
- Ends with **Global Average Pooling (GlobalAvgPool)** to handle different input image sizes.

Loss Function:

- Uses **Focal Loss**, which helps the model focus more on hard-to-classify samples, making it great for handling imbalanced datasets.

Data Augmentation:

- Applies **MixUp** and **CutMix** to make the model generalize better by blending images and labels.
- Uses **AutoAugment** to introduce more variations in the training images.

Data Loading:

- The function **get_balanced_dataloader()** includes **WeightedRandomSampler** to ensure class balance in each batch.

Training Process:

- Uses **AdamW optimizer**, with different learning rates for different layers in **ResNeXt-101**.
- Uses **OneCycleLR** to dynamically adjust the learning rate, helping the model avoid getting stuck in local minima.
- Implements **Early Stopping** to monitor validation loss and stop training early if needed to prevent overfitting.

The goal is to improve the model's **generalization** by using **smart data augmentation and learning rate strategies**, while **Focal Loss** helps handle class imbalance.

Method

Model architecture

Base Model: ResNeXt101 64x4d

Fully Connected Layers

```
base_model.fc = nn.Sequential(
    nn.Linear(in_features, 1024),
    nn.BatchNorm1d(1024),
    nn.ReLU(inplace=True),
    nn.Dropout(0.2),
    nn.Linear(1024, num_classes)
)
```

Activation Functions: ReLu

Dropout Rates: 0.2 for each layer

Hyperparameters

Optimizer: AdamW with different learning rates for different layers

- Layer 2: 2e-4
- Layer 3: 3e-4
- Layer 4: 5e-4
- Fully Connected Layer: 5e-4

Weight Decay: 1e-4

Learning Rate Scheduler: OneCycleLR

- Max LRs: [2e-4, 3e-4, 5e-4, 1e-4]

- Epochs: 1000
- Percent Start: 0.2
- Div Factor: 25
- Final Div Factor: 1e4
- Annealing Strategy: Cosine

Batch size: 64

Epoch: 1000 (However, since early stopping is applied, training usually stops at the 50th epoch)

Total parameters :83.6M

Data Augmentation

Since the dataset is imbalanced and not large enough, I used various data augmentation techniques during training to improve the model's generalization. These include random cropping, flipping, color jittering, rotation, and perspective transformation, which help simulate different angles, lighting conditions, and partial occlusions, allowing the model to learn more diverse features. I also applied normalization to match the input distribution of pretrained models and used random erasing to make the model more adaptable to occlusions. This helps the model maintain good classification performance in different environments.

To improve the model's generalization, I applied several data augmentation techniques during training. The selected augmentations include:

- **RandomResizedCrop (224, scale=(0.7, 1.0))**: Randomly crops and resizes the image to simulate different object scales and viewpoints.
- **RandomHorizontalFlip()**: Horizontally flips images to enhance robustness to left-right variations.
- **RandomVerticalFlip (p=0.3)**: Vertically flips images with a 30% probability to increase variation in object orientation.
- **AutoAugment (policy=AutoAugmentPolicy.IMAGENET)**: Dynamically applies a learned augmentation strategy from ImageNet to introduce additional variations in color, contrast, and geometry.
- **ToTensor()**: Converts the image to a PyTorch tensor format.
- **Normalize (mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])**: Normalizes the input to align with pretrained model distributions.
- **RandomErasing (p=0.3, scale=(0.02, 0.2))**: Randomly masks small regions of the image with a probability of 30% to improve occlusion resistance.

By using these augmentation techniques, the model learns more diverse features and gains better robustness to real-world variations.

mixup_data

Due to the limited amount of data, I applied the Mix-up technique, which generates new training samples by randomly blending the input features and labels of two samples. This approach helps improve the model's generalization ability, reduces overfitting, and enhances its stability when encountering unseen data.

cutmix_data

Since the dataset is limited, I used the Cut Mix technique, which randomly cuts out a part of an image and replaces it with a patch from another sample to create new training data. This helps the model learn both local and global features, improves generalization, reduces overfitting, and makes it more stable when handling unseen data.

rand_bbox

During training, I used the CutMix data augmentation technique to improve the model's generalization and reduce overfitting. CutMix works by randomly selecting a rectangular region in an image and replacing it with the corresponding region from another image, effectively blending different class features within a single image. This helps the model learn more diverse semantic information. The rectangular region used in this process is determined by the rand_bbox function.

Early Stopping

To prevent overfitting and avoid wasting resources by continuing training when the model's performance is no longer improving, I applied early stopping. Training is terminated early if the validation loss does not improve for 15 consecutive steps.

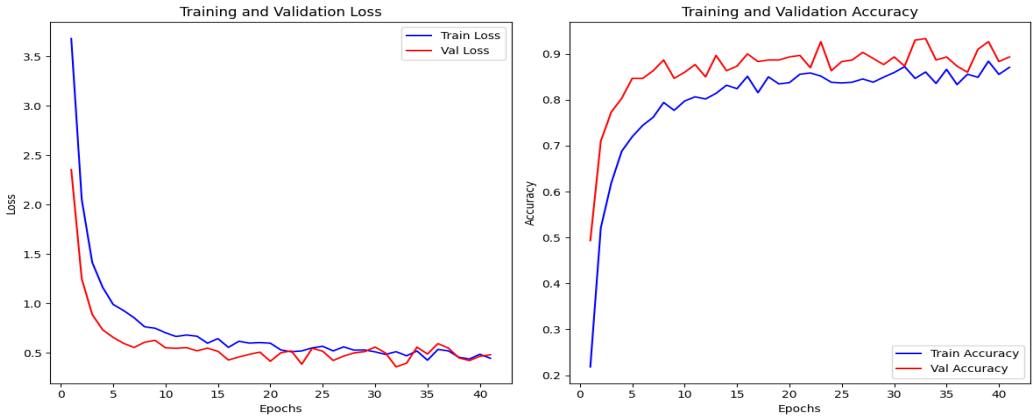
get_balanced_dataloader

This addresses the issue of class imbalance in the dataset. It assigns different weights to each class so that during training, each class has an equal chance of being sampled. This prevents the model from being biased toward the more frequent classes, improving the model's ability to recognize minority classes.

OneCycleLR

OneCycleLR is a learning rate scheduler that adjusts the learning rate and momentum during training. It first increases the learning rate from a small value to a peak, then gradually decreases it. This helps the model train faster and generalize better by avoiding getting stuck in local minima and allowing fine-tuning towards the end. It's a popular method for improving training efficiency.

Results



The model training process exhibits good convergence, with the loss decreasing rapidly in the early stages and gradually stabilizing as training progresses, indicating effective learning of data features. Both training and validation accuracy improve with increasing epochs, and the final validation accuracy is slightly higher than the training accuracy, suggesting strong generalization ability without significant overfitting or underfitting. Additionally, the similarity in trend between the two curves indicates consistent performance across training and test data, implying that the current hyperparameter configuration is well-tuned, and the model is stable with good predictive capability.

38	313551139	1	2025-03-26 15:55	252886	313551139	0.94

準確率大約落在 92~94%

References

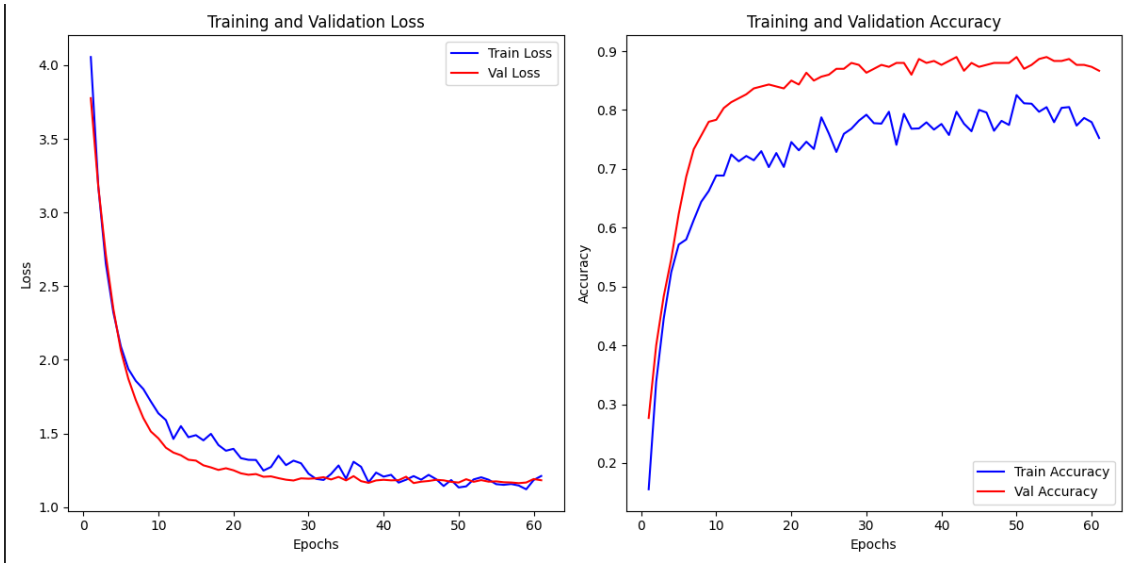
Without references

Additional experiments

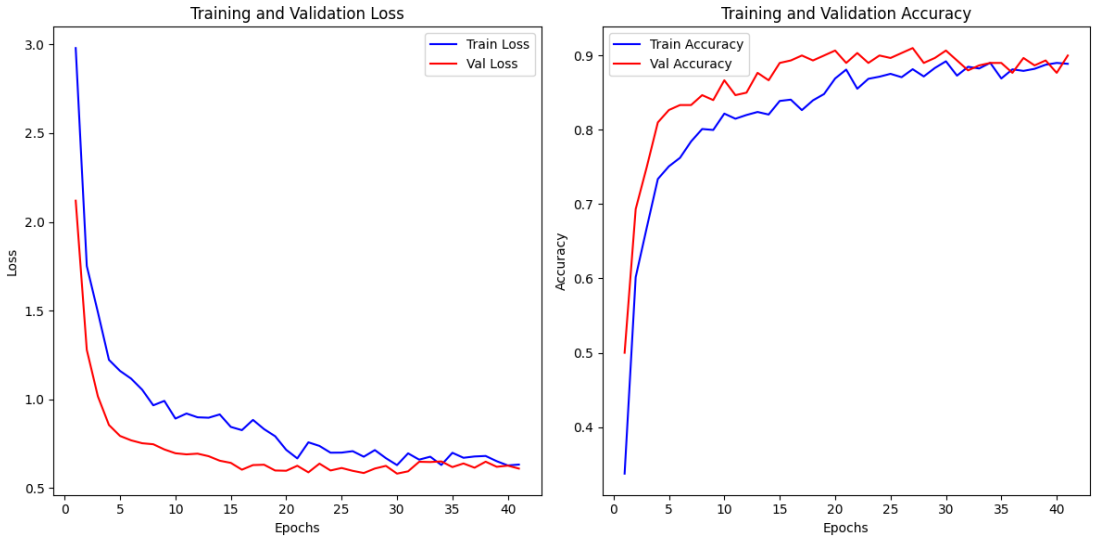
Loss

Originally, I used CrossEntropy as my loss function. However, after examining the training data, I found that the number of images in each class was imbalanced. After researching, I discovered that Focal Loss is more suitable for handling data imbalance. So, I switched to Focal Loss to see if it would improve performance. Additionally, I experimented with a hybrid approach by combining CrossEntropy and Focal Loss in different ratios to balance stability and class-aware weighting. The training curve confirms that both pure Focal Loss and the mixed loss strategy outperform using only CrossEntropy. Moreover, the mixed loss strategy showed a slight improvement over pure Focal Loss, further boosting model performance.

CrossEntropy



Focalloss



Focalloss(80%) + CrossEntropy(20%)

1. CrossEntropy

- **Loss Curve:**
    - Both training loss (blue) and validation loss (red) decrease rapidly at the beginning, but the validation loss fluctuates significantly later, suggesting potential overfitting.
- **Accuracy Curve:**
    - Training accuracy improves steadily, but validation accuracy plateaus and shows noticeable fluctuations, indicating weaker generalization ability.

2. Focal Loss

- **Loss Curve:**
    - The training and validation losses are significantly lower compared to CrossEntropy, showing that the model has learned better features.
- **Accuracy Curve:**
    - Both training and validation accuracy curves are more stable than CrossEntropy, with validation accuracy being higher, suggesting improved generalization.

3. Focal Loss (80%) + CrossEntropy (20%)

- **Loss Curve:**
    - Similar to pure Focal Loss, but with an even smoother validation loss, indicating improved model stability.
- **Accuracy Curve:**
    - Shows the best overall performance, with validation accuracy surpassing pure Focal Loss, suggesting that this hybrid approach further enhances model effectiveness.

Conclusion

- **CrossEntropy** tends to overfit and has weaker generalization.
- **Focal Loss** helps handle class imbalance, leading to a more stable model.

- **Focal Loss + CrossEntropy** combines the strengths of both, achieving the best generalization and validation performance.
- Based on the results, using **80% Focal Loss + 20% CrossEntropy** appears to be the most effective strategy, balancing model stability and accuracy.