# Visual Recognition Using Deep Learning 2025 HW1: Image Classification

313551073 顏琦恩

https://github.com/miayan0110/NYCU-Visual-Recognitionusing-Deep-Learning-2025-Spring/tree/master/hw1

## I. Introduction

Homework 1 focuses on image classification, a fundamental task in computer vision where a model learns to categorize images into predefined classes. The objective is to develop a machine learning model capable of accurately classifying input images based on their visual features.The assignment involves several key steps, starting with data preprocessing, which includes loading the image dataset and applying necessary transformations to images. Next, the model is designed using ResNet as the base model. Finally, I implement the training, validation, and evaluation processes to enable the model to classify new images effectively.

For training the model, I use the dataset provided by the TAs. During training, I validate the model at every epoch to assess its image classification performance. Further implementation details will be discussed in the next section.

## II. Method

I implemented the image classification model using ResNeXt101 as the base model and introduced several modifications to the final layers to adapt it to our dataset, which consists of 100 categories, compared to the original 1000-class output of ResNeXt101. I trained the model for 100 epochs with a learning rate of 0.0001 and batch size 64.

For data preprocessing, I first applied a random horizontal flip with a 50% probability to images to introduce data augmentation. After that, I resized and normalized the images to ensure that the input images are properly formatted for the model.

For the model architecture, I leveraged ResNeXt101 (64×4d) with pretrained weights as the feature extractor.Since the original model produces a 1000-dimensional output, I introduced an attention mechanism to refine feature selection before classification. The attention module consists of two fully connected layers, with a hidden dimension of 500, using a Tanh activation followed by a Sigmoid function. This mechanism generates attention weights that are applied element-wise to the ResNeXt101 output, emphasizing important features while suppressing less relevant ones.

The final classification head includes a batch normalization layer followed by three fully connected layers with ReLU activations and dropout for regularization. The last layer outputs predictions corresponding to our 100-class dataset, ensuring the model is well-adapted to our specific classification task. The detailed code is shown in the following block.

```python
class ResNext(nn.Module):
    def __init__(self, num_classes):
        super().__init__()
        self.resnet = models.resnext101_64x4d(weights='DEFAULT')
        self.attention = nn.Sequential(
            nn.Linear(1000, 1000 // 2),
            nn.Tanh(),
            nn.Linear(1000 // 2, 1000),
            nn.Sigmoid()
        )
        self.classifier = nn.Sequential(
            nn.BatchNorm1d(1000),
            nn.Linear(1000, 512),
            nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(512, num_classes)
        )

    def forward(self, img):
        x  = self.resnet(img)
        attn_weights = self.attention(x)
        x = self.classifier(x * attn_weights)
        return x
```

## III. Results

During training, I saved a checkpoint at the end of each epoch and recorded the losses to plot the loss curve (Fig. 1). Additionally, I tracked the validation accuracy for every checkpoint and visualized it in Fig. 2.

From these figures, we can observe that the validation accuracy plateaus around epochs 17 to 20, aligning with the point where the loss converges. Furthermore, in the later stages of training, particularly around epoch 90, the validation accuracy begins to decline. This drop is likely due to overfitting, where the model becomes too specialized to the training data, leading to reduced generalization performance.

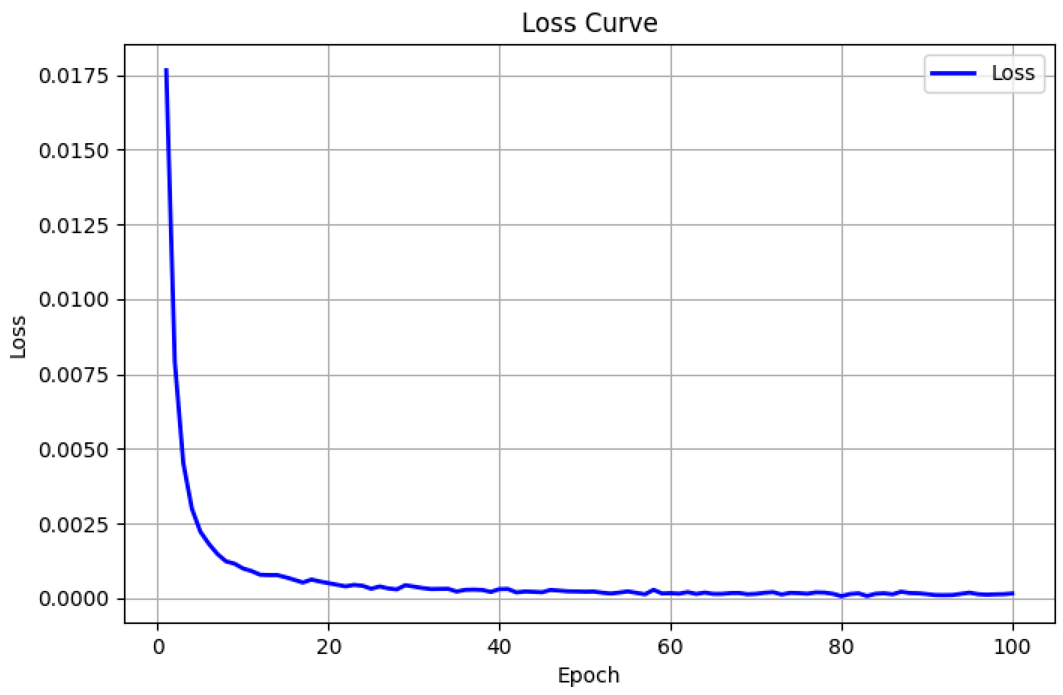

Figure 1. Loss curve.

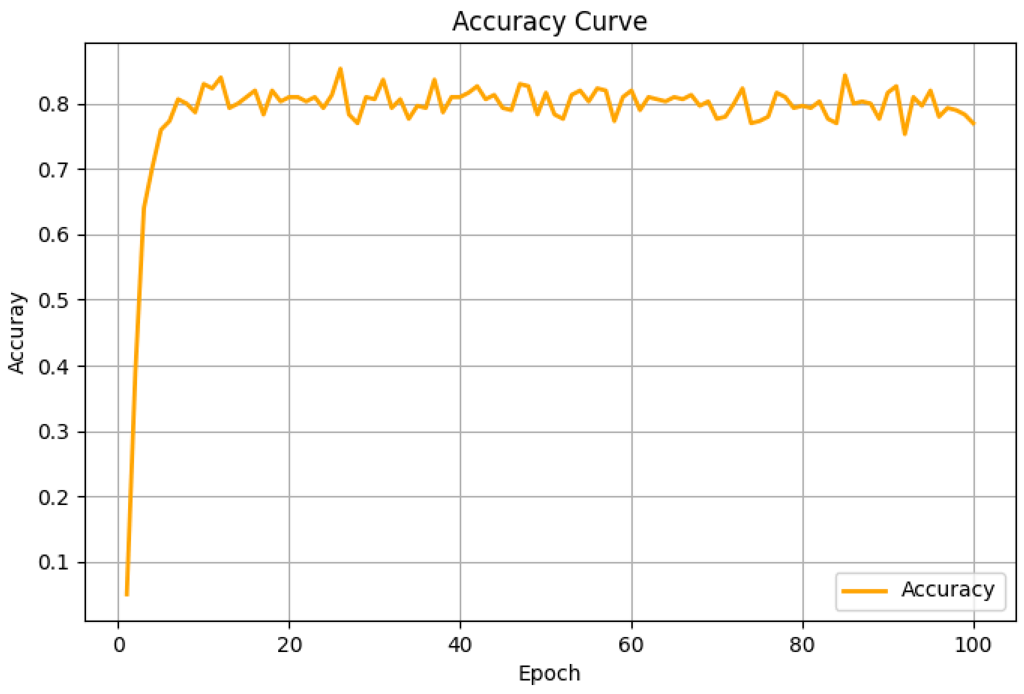

Figure 2. Accuracy curve.

## IV. References

1. Pytorch Doc: resnext101_64x4d
https://pytorch.org/vision/main/models/generated/torchvision.models.resnext101_64x4d.html
#torchvision.models.resnext101_64x4d

## V. Additional Experiments

To increase the accuracy of image classification, I conducted some experiments. Before the last version of the model structure I used, I simply changed the output class number of ResNeXt101 to 100. The result was quite good since I was using the pretrained weights. However, to achieve a better score, I attempted to add more layers to the model.

The first modification was adding an attention module to help the model focus on key features, along with a linear layer for the final classification. This modification became my last version of the model structure, and the results are shown in the previous section. Although the performance was already good, I still aimed for further improvement. Therefore, I tried adding more linear layers to the classifier. The detailed code is provided in the following code block.

```python
class ResNeXtL(nn.Module):
    def __init__(self, num_classes):
        super().__init__()
        self.resnet = models.resnext101_64x4d(weights='DEFAULT')
        self.attention = nn.Sequential(
            nn.Linear(1000, 1000 // 2),
            nn.Tanh(),
            nn.Linear(1000 // 2, 1000),
            nn.Sigmoid()
        )
        self.classifier = nn.Sequential(
            nn.BatchNorm1d(1000),
            nn.Linear(1000, 1024),
            nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(1024, 512),
            nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(512, num_classes)
        )

    def forward(self, img):
        x  = self.resnet(img)
        attn_weights = self.attention(x)
        x = self.classifier(x * attn_weights)
        return x
```

From Fig. 3, we can see that the result of this modification is worse than the previous attempt. The performance degradation may be due to the fact that adding only one additional linear layer to the classifier had a minimal impact on performance. Additionally, increasing the number of parameters could have led to overfitting, negatively affecting the model's generalization ability.

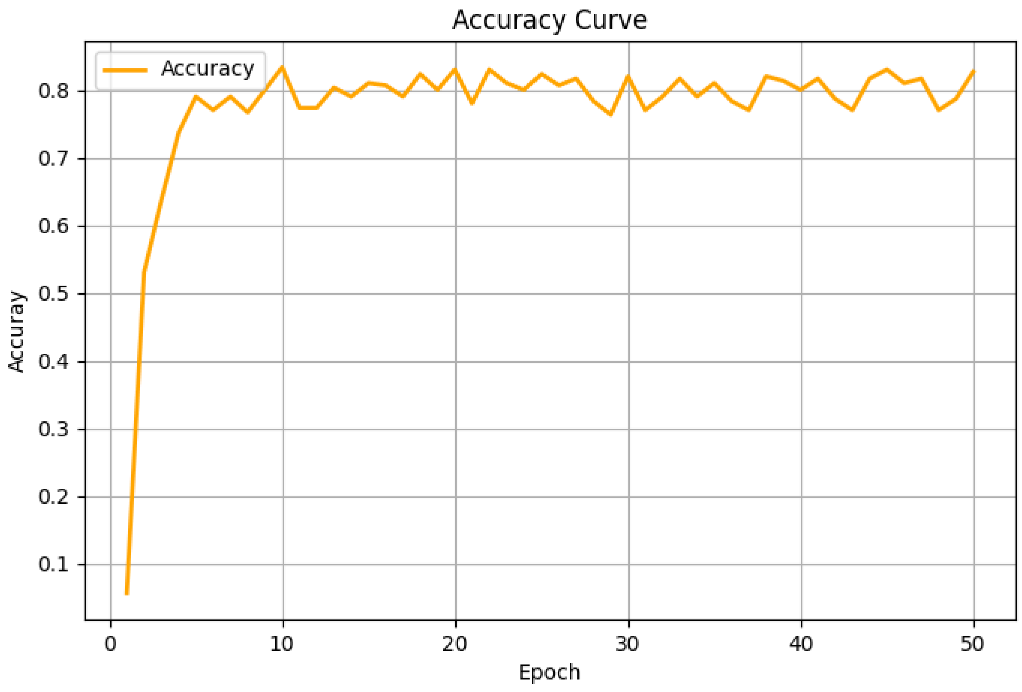

Figure 3. Accuracy curve of experiment.