VRDL-Homework1

Name: Kuok-Tong Ng
Student ID: 309652030

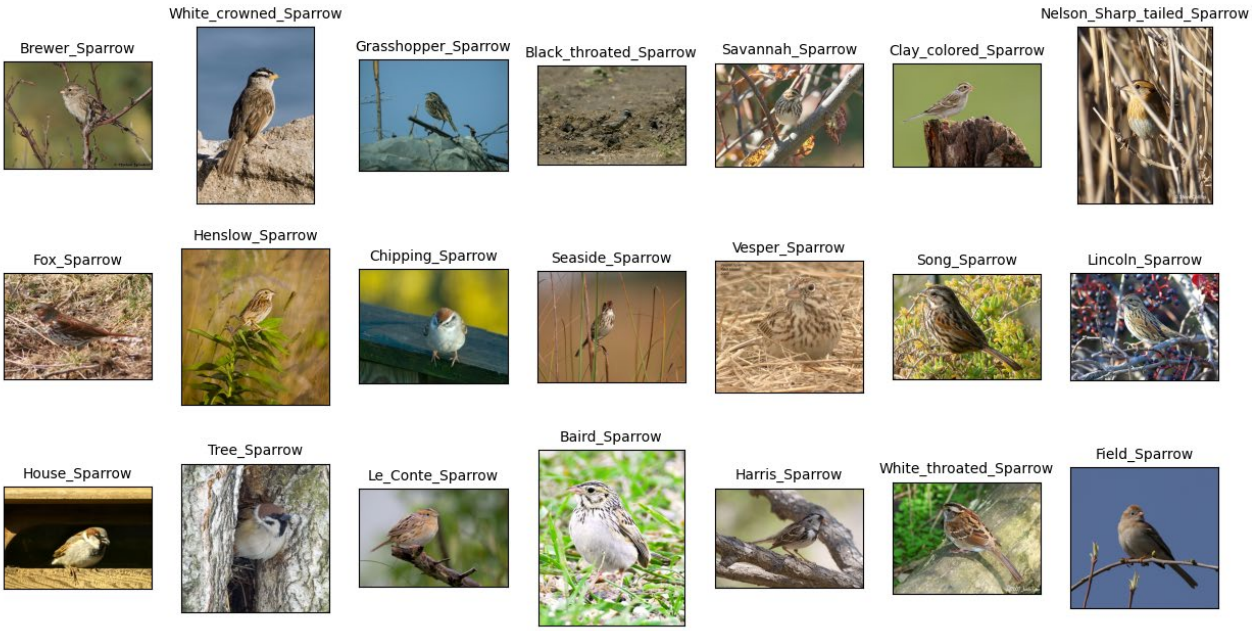
Project repository: https://github.com/K-T-Ng/VRDL-Homework1

Introduction

This homework is about **bird images classification task**. There are 6033 birds images belonging to 200 bird species, 3000 (3033) of them are training (testing) data, respectively.

There are some reasons that make this task becomes challenging:

- **Lack of training data**: As I mentioned earlier, there are only 3000 training images in total, this is hard for the model to learn well (easy to get overfit and hard to generalize).
- **Similar species**: This task is concentrate on classifying bird species, some of them looks very similar (There are 21 species about Sparrow, like Brewer Sparrow, Grasshopper Sparrow, etc.).



Methodology

Data Pre-Processing

- Image in our dataset with different size (see the figure above), so we resize them to (500,500) and crop them into (400,400).
- Image are represented by a tensor, the range of pixel value is from 0 to 255, so we scale it to 0 to 1 (by dividing 255) and then normalize it with mean = [0.485, 0.456, 0.406],

std = [0.229, 0.224, 0.225] (the normalize method that torchvision pretrain model does)

Data Augmentation

In order to obtain more training data, we apply the following data augmentation.

- Random horizontal flip with probability 0.5
- Apply grid mask with probability 0.25, removing 30% information from the original image. Gird mask is an augmentation that remove some information from the source, for more detail, see [1].

Model Architecture

Basically, I use **ResNet50 with pretrained weights** as the model architecture.
I have also accessed the layer before fully connected layer (more precisely, before average pooling layer) since I have used **Mutual Channel Loss** during the training process (which is introduced in the next paragraph)

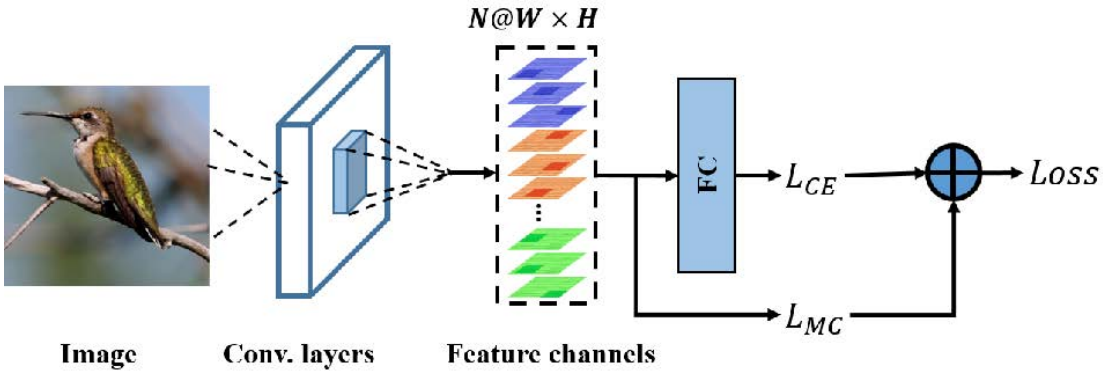


Figure 1: Model architecture, extracted from [2]

Mutual Channel Loss (MC-Loss)

As I mentioned earlier, Mutual Channel Loss has been applied in this homework, which is a method proposed in February 2020 [2].

This is an interesting method that trying to solve fine-grained image classification problem. This method can be applied directly to ResNet50 model. As in Figure1 shown, the only thing that I need to do is to access the layer before average pooling layer.

For ResNet50, the size of Feature channels (layer before avg. pooling) is 2048@WxH (see Figure1). MC-loss trying to **represent each class by a numbers of feature channels**. For example, in this task, there are 200 classes in total, so, there are 10(11) channels to represent each of the class in the first 152(last 48) class, respectively.

MC-Loss is composed by 2 main components:

- **Discriminality component**: force all feature channels belonging to the same class to be discriminative.

- **Diversity component**: force the feature channels in the same class to discover different discriminative regions.

So, the loss function during training process looks like

$$L_{total} = L_{CE} + \mu(L_{dis} - \lambda L_{div})$$

Where μ, λ are hyper-parameters.

Five-Fold & Ensemble

Usually, I will split the dataset to 80% (20%) for training (validation) set once, respectively. However, based on the following reasons, I apply 5-fold instead of split them once.

- When comparing the performance between two methods (or hyper-parameters), I can obtain a more reliable conclusion running on 5-fold (when there is a significant gap between two methods in multiple folds, say A > B, then I will trust A is better than B).
- In this dataset, 3000 images are equally distributed over 200 classes (i.e, 15 images for each class). Hence, it is easy to split the dataset into 5 groups, each of them contains 600 images (3 images for each class).
- I also want to perform bagging (an ensemble method) to produce a more robust model. Although bagging requires bootstrapping (randomly drawing with replacement), I reuse the 5 folds above (in order to keep the "equally distributed" property for the original training data).

As a result, 5 models will be produced after the training process. For each testing data, we average the probabilities that returned by 5 models and choose the highest average probability.

Hyper-parameters

- Batch size = 24
- Maximum epoch = 20
- Optimizer: AdamW
  - Learning rate for layers before fully connected layer: 1e-4
  - Learning rate for fully connected layer: 5e-4
  - Weight decay factor: 5e-4
- Learning scheduler: Exponential LR
  - Decay factor: 0.95
- Parameters of loss function
  - $\mu = 0.01$
  - $\lambda = 10$

Experiment

In this homework, I have test whether MC-loss has a significant improvement or not.

As I mentioned before, I use 5-fold to compare whether which setting is better. I also fix the random seed (42) in the following experiment.

With the hyper-parameter setting above (except μ and λ), and with Random horizontal flip data augmentation with probability 0.5, I get the maximum accuracy on each fold as follows:

	Fold1	Fold2	Fold3	Fold4	Fold5
$\mu = 0, \lambda = 0$	0.7667	0.7717	0.7717	0.7767	0.7967
$\mu = 0.01, \lambda = 10$	0.7783	0.7917	0.7833	0.7850	0.7983

This convince me that adding MC-loss can have a significant improvement. I also try with a higher parameter μ, but it seems like the MC-loss become bigger, it dominated the total loss (Recall that $L_{total} = L_{CE} + \mu L_{MC}$) and makes the accuracy become lower.

I have also test online augmentation. That is, an training image perform some transformation (Horizontal Flip, Grid Mask, Add Gaussian noise) with a probability p during training phase, but this doesn't gives a significant improvement.

Conclusion

What I have done

In this homework, I fine tune a model with ResNet50 as model architecture with pre-train weight. In order to increase the testing accuracy, I find an interesting paper (MC-loss) that dealing with fine-grained image classification and re-implemented MC-loss by referencing to [3].

Some exploration

- <u>It is important to choose an appropriate batch size</u>

  In the beginning, I try to resize the image into (224, 224) and using a mini-batch size 100. With this setting, the model can't even beat the baseline. Afterwards, I try to reduce the mini-hatch size to 24 and obtain an improvement.

  I am wondering whether a large batch size will decrease the generalization ability of the model or not. Using a large-batch size will fit the training loss surface well, but our goal is not fit the training loss surface only. Is it possible that using a lower mini-batch size like adding some "randomness" and this gives a better generalization ability?

Other things that I also want to do (implement)

The following are some method I want to try, but running out of time.

- Circle Loss (refer to [4])

  This is a method that working on loss function, unified a loss function to point out some drawback (same penalty strength for minimizing (maximizing) between (within) class similarity, respectively) with the existent loss function (e.g, softmax cross entropy loss) and modify that loss function to mitigate the drawback.

- Work harder on data augmentation

  As I mentioned above, I don't get a better result when I perform (online) data augmentation. But my friend (another student in this class) told me he gets an improvement when he use offline data augmentation.

  Maybe I need to try offline augmentation as well or work harder on online augmentation (e.g. try more combination of different transformation).

Reference

[1] Grid Mask Data Augmentation: 2001.04086.pdf (arxiv.org)

[2] Mutual Channel Loss: https://arxiv.org/pdf/2002.04264.pdf

[3] Mutual Channel Loss implementation: Kurumi233/Mutual-Channel-Loss: Reimplementation of Mutual-Channel Loss for Fine-Grained Image Classification. (github.com)

[4] Circle Loss: Circle Loss: A Unified Perspective of Pair Similarity Optimization (thecvf.com)