HOMEWORK 3
LONG-TAILED DATASET

Le Van Hung (黎文雄)

Student ID: 0860831

1. Training model and experimental details

In this project, I using VGG16 to training this CIFAR10 data set (see in the model.py file). For the model detail about the model, you can use print(model) to see all layer in this model like below:

(features): Sequential:

```
 (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
(2): ReLU(inplace=True)
(3): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(4): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
(5): ReLU(inplace=True)
(6): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
(7): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(8): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
(9): ReLU(inplace=True)
(10): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(11): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
(12): ReLU(inplace=True)
(13): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
(14): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(15): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
(16): ReLU(inplace=True)
(17): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(18): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
(19): ReLU(inplace=True)
(20): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(21): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
```

```
(22): ReLU(inplace=True)
(23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
(24): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(25): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
(26): ReLU(inplace=True)
(27): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(28): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
(29): ReLU(inplace=True)
(30): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(31): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
(32): ReLU(inplace=True)
(33): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
(34): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(35): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
(36): ReLU(inplace=True)
(37): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(38): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
(39): ReLU(inplace=True)
(40): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(41): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
(42): ReLU(inplace=True)
(43): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
(44): AvgPool2d(kernel_size=1, stride=1, padding=0)
(classifier): Linear(in_features=512, out_features=10, bias=True)
```

Parameters for the training process:

Training batch size	128
Testing batch size	100
Random seed (for easy to compare in two cases)	25
Number of epochs	20

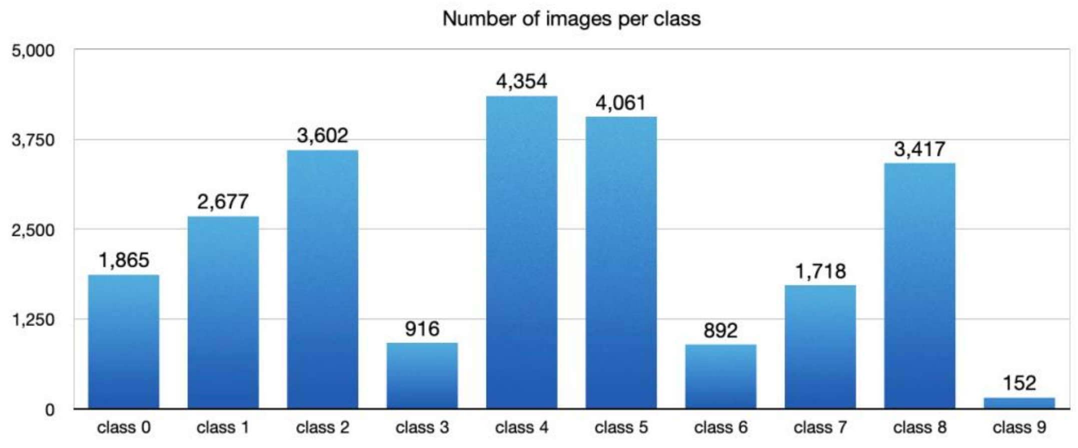
2. How to deal with the imbalance problem

To solve the imbalance problem, I use two techniques: data augmentation and oversampling.

a. Data augmentation

- For the data augmentation, I will create some new images from given training data set by cropping images randomly, flip images.
- transforms.RandomCrop(32, padding=4) : zero padding in left, top, right, button of image and crop the padded image at a random location to get the output images with the same size.
- transforms.RandomHorizontalFlip(): after we random crop image, we will flip them to get the some new image.
- With all data images (both training set and testing set) we use transform.Nomalize() function to normalize all image with given mean and standard deviation. transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010)) : This function will normalize the given image with given mean and standard deviation.

b. Oversampling

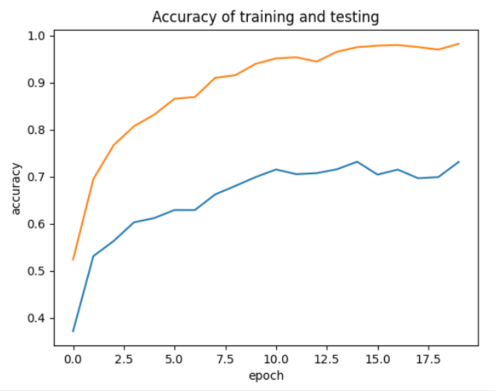
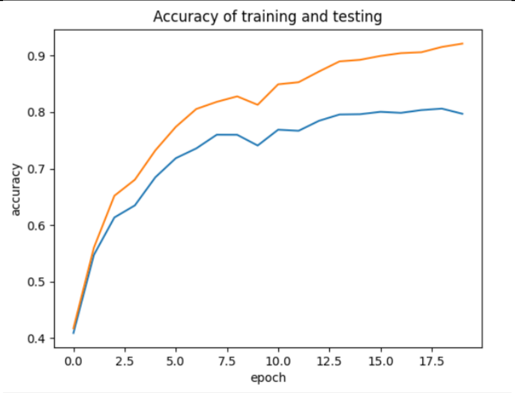


- Because of imbalance on this given training dataset, the model will most likely overfit on the majority classes (for example overfit on class 3,4,5,8), so we will get more result with incorrectly classify on data of class 3 and 9.
- To counter the imbalanced dataset, we can resample data set using function WeightedRandomSampler, which draws the samples using probabilities. For more detail, see the code below:

```
# For unbalanced dataset, create a weightes sampler
targets = train_ds.target
class_count = np.unique(targets, return_counts= True)[1]
#class_count = [1865, 2677, 3602, 916, 4354, 4061, 892, 1718, 3417, 152]
class_count = np.array(class_count)
weight = 1. / class_count
samples_weight = weight[targets]
samples_weight = torch.from_numpy(samples_weight)
sampler = data.sampler.WeightedRandomSampler(samples_weight,
len(samples_weight))
train_dl = data.DataLoader(dataset=train_ds,
batch_size=train_bs,sampler=sampler,pin_memory=True)
```

3. Comparison the result

a. Comparison the accuracy

Before counter imbalance	After counter imbalance
	
Training accuracy: 98.28% **Test accuracy: 73.15%**	**Training accuracy: 92.11%** **Test accuracy: 79.07%**

b. Comparison the accuracy of 10 per class

Class .No	Accuracy before counter imbalance	Accuracy after counter imbalance
0	77.8%	84.4%
1	92.7%	94.5%
2	84.2%	70.5%
3	**39.9%**	**68.3%**
4	85.2%	82.5%

5	86.2%	70.3%
6	70.1%	75%
7	70.1%	90..5%
8	91.3%	89.9%
9	**34.4%**	**71.5%**

c. Comments

- After applying the oversampling and data augmentation for training set, the accuracy of testing will be improved clearly. From 73.15% to 79.07%.
- Let focus on the two classes (class 0 and class 9) with very few training data (916 and 152 respectively), before applying counter imbalance technique, accuracy of those two classes is very low (only 39.9% and 34.4% respectively). After applying counter imbalance technique, accuracy of those two class have been improved clearly (68.3% and 71.5% respectively).
- Class 6 also have few training data, but the accuracy is not bad even without counter imbalance problem. I think because we have 4 animal classes (cat, deer, dog, horse) which are easy to miss classification, so maybe it can explain why class 6 (dog) sill don’t have bad result.