

ASSIGNMENT 3 – LONG-TAILED DATASET

REQUIREMENTS:

```
conda install seaborn, matplotlib, NumPy
```

HOW TO RUN:

Default (Without resampling and weight rebalancing)

```
python 0860812.py
```

Turn on Resampling only

```
python 0860812.py --resampling_balance True
```

Turn on loss reweighting only

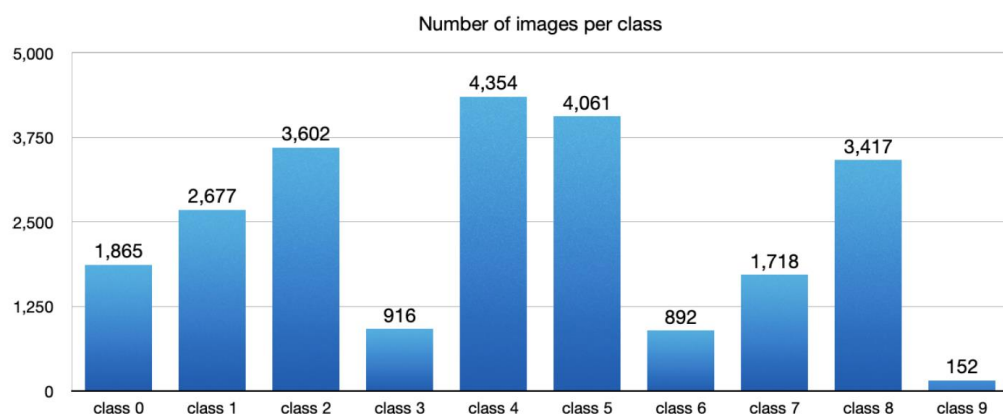
```
python 0860812.py --reweight_balance True
```

HOW TO DEAL WITH THE IMBALANCE PROBLEM

A long-tail dataset as known as data imbalance usually reflects an unequal distribution of classes within a dataset. For example, in a credit card fraud detection dataset, most of the credit card transactions are not fraud and very few classes are fraud transactions. This leaves us with something like a 50:1 ratio between the fraud and non-fraud classes.

In this case, we want to solve the imbalance dataset of CIFAR-10. The CIFAR-10 dataset consists of 60000 32x32 color images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. But in this assignment, we make the data training imbalance with current composition:

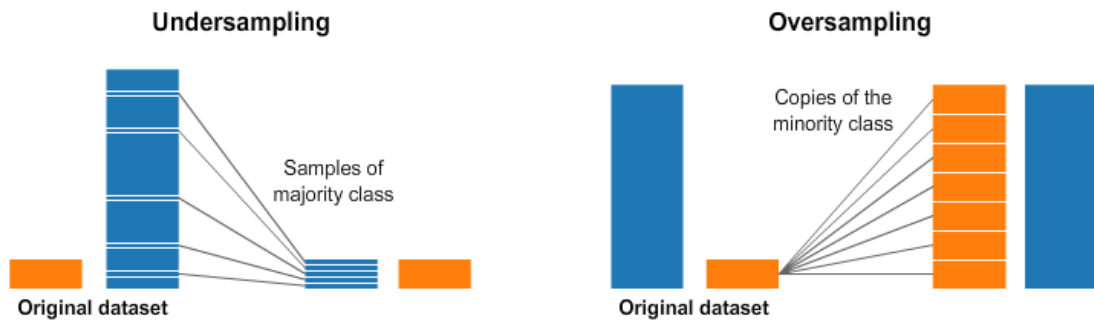
Long-Tailed CIFAR-10



The data test will remain the same, it will be tested with 1000 data for each class so the total of data test will be 10000 data.

FIRST METHOD – RESAMPLE THE DATA TRAINING

A widely adopted technique for dealing with highly unbalanced datasets is called resampling. It consists of removing samples from the majority class (under-sampling) and / or adding more examples from the minority class (over-sampling) like what shown below:



Despite the advantage of balancing classes, these techniques also have their weaknesses (there is no free lunch). The simplest implementation of over-sampling is to duplicate random records from the minority class, which can cause overfitting. In under-sampling, the simplest technique involves removing random records from the majority class, which can cause loss of information.

In torch, we could use `WeightRandomSampler` for sampling the unbalanced dataset. In this assignment, I use methods as follow:

```
def resampling_balance(data):
    targets = data.target
    class_count = np.unique(targets, return_counts=True)[1]
    print("Class number before resampling: ", class_count)

    weight = 1. / class_count
    samples_weight = weight[targets]
    samples_weight = torch.from_numpy(samples_weight)
    sampler = WeightedRandomSampler(samples_weight, len(samples_weight))
    return sampler
```

data is training dataset, then we will count all the labels we have from each class, from that we could know how many data for each class. We then set the resampling weight for each class according to how many data. We then create a sampler using `WeightedRandomSampler` that we could use for doing oversample or undersampling to make the data balance.

We will compare the confusion matrices before and after sampling in the next section of the report.

SECOND METHOD – APPLY THE LOSS REWEIGHTING

While handling a long-tailed dataset (one that has most of the samples belonging to very few of the classes and many other classes have very little support), deciding how to weight the loss for different classes can be tricky. Often, the weighting is set to the inverse of class support or inverse of the square root of class support. Re-weighting by the effective number of samples gives a better result. The Class-Balanced Loss is designed to address the problem of training from imbalanced data by introducing a weighting factor that is inversely proportional to the effective number of samples.

The effective number of samples can be imagined as *the actual volume that will be covered by n samples where the total volume N is represented by total samples.*

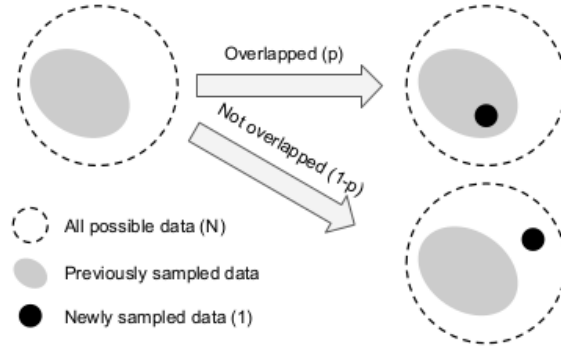


Figure 2. Given the set of all possible data with volume N and the set of previously sampled data, a new sample with volume 1 has the probability of p being overlapped with previous data and the probability of $1 - p$ not being overlapped.

Formally speaking, given a sample from class I that contains n_i samples in total, we propose to add a weighting factor to the loss function, with a hyperparameter of β from 0 – 1. The weighting factor could be written as:

$$E_n = (1 - \beta^n) / (1 - \beta) :$$

The class-balanced (CB) loss can be written as:

$$\text{CB}(\mathbf{p}, y) = \frac{1}{E_{n_y}} \mathcal{L}(\mathbf{p}, y) = \frac{1 - \beta}{1 - \beta^{n_y}} \mathcal{L}(\mathbf{p}, y),$$

The code implementation is shown below:

```
target = train_dl.dataset.target
dicts_target = {}

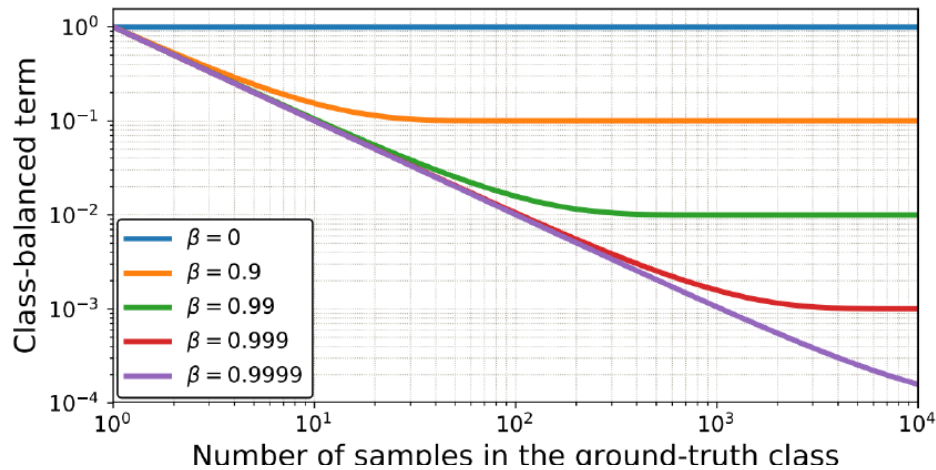
'''Count each label to determine weight later'''
for a in range(0, 10):
    dicts_target[a] = countX(target, a)

beta = (sum(dicts_target.values()) - 1) / sum(dicts_target.values())
```

First, we will take all the labels and determine how many data for each class. We will save this information to `dicts_target`. Next, we will try to estimate β where β could be count as follow:

Proposition 1 (Effective Number). $E_n = (1 - \beta^n) / (1 - \beta)$, where $\beta = (N - 1) / N$.

Where N is the number of total data training, here we will get beta of 0.9999577238522026, beta will represent class-balanced term as shown in the graph below:



After getting the beta, we will count weight (E_n) for each class according to the number of that class in the dataset using this code:

```
'''Count the weight according to the number of labels (Class
Balanced Loss)'''
weights = []
for key, value in dicts_target.items():
    weight = (1 - beta) / (1 - (beta ** value))
    weights.append(weight)

class_weights = torch.FloatTensor(weights).cuda()
print("Weights according to imbalanced dataset", weights)
criterion = nn.CrossEntropyLoss(weight=class_weights)
```

After getting all the weights in list form, we put the weight inside the weight parameters of cross-entropy loss. Here the example of weight value according to the amount of data in this assignment:

Weights according to the imbalanced dataset

```
[0.0005575975182772732, 0.00039508129586853094, 0.0002992920316904384,
0.0011129544819594077, 0.000251455212004829, 0.0002679821947249365,
0.001142323463118114, 0.0006034538076618386, 0.00031429502261202393,
0.006599969014314407]
```

TRAINING MODEL AND EXPERIMENTAL DETAILS

For this assignment I use DenseNet161 as the model for predicting CIFAR-10 result. DenseNet is a type of convolutional neural network that utilises dense connections between layers, through Dense Blocks, where we connect all layers (with matching feature-map sizes) directly with each other. To preserve the feed-forward nature, each layer obtains additional inputs from all preceding layers and passes on its own feature-maps to all subsequent layers.

This DenseNet 161 already a pretrained network of CIFAR-10 with an accuracy of 94.24% so it could help the training and could reduce the epoch of training. The optimizer using Stochastic Gradient Descent, Loss using cross-entropy loss, and also use CossineAnnealinglr as learning rate scheduler. The environment setting shown as below:

```
criterion = nn.CrossEntropyLoss()

optimizer = optim.SGD(net.parameters(), lr=lr,
                      momentum=0.9, weight_decay=5e-4)
scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=100)
```

Here is the code for training:

```
def train(epoch):
    print('\nEpoch: %d' % epoch)
    net.train()
    train_loss = 0
    correct = 0
    total = 0
    for batch_idx, (inputs, targets) in enumerate(train_dl):
        inputs, targets = inputs.to(device), targets.to(device)
        optimizer.zero_grad()
        outputs = net(inputs)
        loss = criterion(outputs, targets.long())
        loss.backward()
        optimizer.step()

        train_loss += loss.item()
        _, predicted = outputs.max(1)
        total += targets.size(0)
        correct += predicted.eq(targets).sum().item()

        progress_bar(batch_idx, len(train_dl), 'Loss: %.3f | Acc: %.3f%% (%d/%d)'
                     % (train_loss / (batch_idx + 1), 100. * correct / total, correct,
total))

    return train_loss / (batch_idx + 1), 100. * correct / total
```

Experimental Detail

The experiment uses Core i9 10th gen processor with RTX2080 Super Max Q Design, I do the following experiment:

1. Train using DenseNet161 without resampling and reweighting loss.
2. Train using DenseNet161 by apply reweighting loss.
3. Train using DenseNet161 by apply resampling dataset

After we got all the results, we will compare the result in the next section. We will do training in 75 epochs like as shown below:

```
for epoch in range(start_epoch, start_epoch + 75):
    adjust_learning_rate(optimizer, epoch, lr)
    train_loss, train_acc = train(epoch)
    test_loss, test_acc = test(epoch)

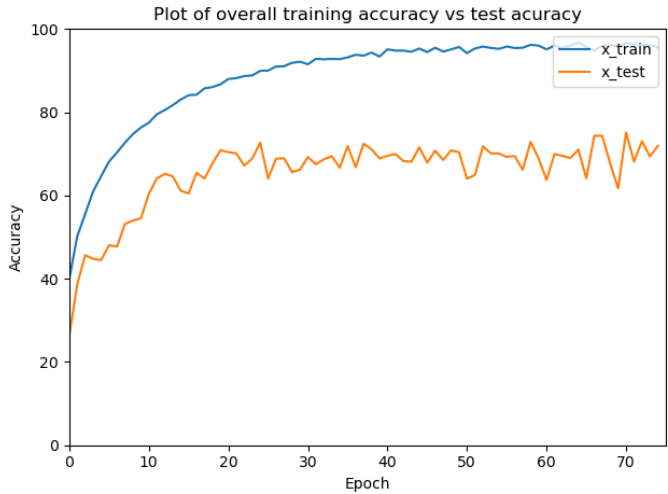

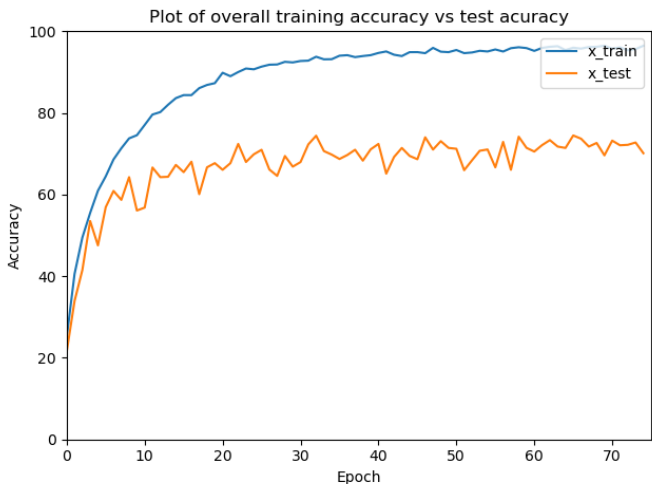
    training_loss.append(train_loss)
    training_accuracy.append(train_acc)

    testing_loss.append(test_loss)
    testing_accuracy.append(test_acc)
```

I do test in every epoch to know if the training works or not. After 75 epochs, I took weight with the best accuracy to compare the accuracy and confusion matrices.

TEST AND COMPARISON

Comparison of test accuracy

Experiment 1 (Without resampling and reweighting loss)	Comment
 <p>Plot of overall training accuracy vs test accuracy</p>	<p>Without applying anything we could see that model training is too confident hence the result looks like overfitting.</p>
 <p>Plot of overall training accuracy vs test accuracy</p>	<p>When we apply loss reweighting, we could see that we could make the model not too confident for class with more number of data. From the graph, we could see that the training accuracy could not be too far with testing accuracy. Applying reweighting loss could reduce overfitting of the long-tail datasets.</p>
 <p>Plot of overall training accuracy vs test accuracy</p>	<p>From the graph, we could see that it almost the same as the first graph. Although this graph looks like overfitting, it is not as overfit as the previous graph. The training also seems more stable than the second method.</p>

Comparison of Accuracy per Class in DataTest

Experiment 1 (Without resampling and reweighting loss)	Comments																																																																																																																									
<table><tr><th></th><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th></tr><tr><th>0</th><td>0.8</td><td>0.011</td><td>0.039</td><td>0.019</td><td>0.018</td><td>0.018</td><td>0.004</td><td>0.012</td><td>0.078</td><td>0.006</td></tr><tr><th>1</th><td>0.012</td><td>0.92</td><td>0.001</td><td>0.007</td><td>0.002</td><td>0.013</td><td>0.005</td><td>0.001</td><td>0.015</td><td>0.027</td></tr><tr><th>2</th><td>0.045</td><td>0.004</td><td>0.66</td><td>0.055</td><td>0.055</td><td>0.096</td><td>0.043</td><td>0.024</td><td>0.015</td><td>0</td></tr><tr><th>3</th><td>0.017</td><td>0.002</td><td>0.031</td><td>0.53</td><td>0.058</td><td>0.24</td><td>0.073</td><td>0.03</td><td>0.01</td><td>0.004</td></tr><tr><th>4</th><td>0.007</td><td>0.003</td><td>0.043</td><td>0.03</td><td>0.82</td><td>0.045</td><td>0.031</td><td>0.009</td><td>0.012</td><td>0</td></tr><tr><th>5</th><td>0.007</td><td>0</td><td>0.014</td><td>0.059</td><td>0.034</td><td>0.85</td><td>0.016</td><td>0.021</td><td>0.001</td><td>0</td></tr><tr><th>6</th><td>0.005</td><td>0.005</td><td>0.029</td><td>0.069</td><td>0.044</td><td>0.062</td><td>0.77</td><td>0.005</td><td>0.006</td><td>0</td></tr><tr><th>7</th><td>0.016</td><td>0.005</td><td>0.018</td><td>0.029</td><td>0.097</td><td>0.094</td><td>0.006</td><td>0.73</td><td>0.002</td><td>0.003</td></tr><tr><th>8</th><td>0.031</td><td>0.017</td><td>0.004</td><td>0.019</td><td>0.002</td><td>0.007</td><td>0.003</td><td>0.006</td><td>0.9</td><td>0.007</td></tr><tr><th>9</th><td>0.1</td><td>0.19</td><td>0.01</td><td>0.029</td><td>0.01</td><td>0.024</td><td>0.029</td><td>0.024</td><td>0.047</td><td>0.53</td></tr></table>		0	1	2	3	4	5	6	7	8	9	0	0.8	0.011	0.039	0.019	0.018	0.018	0.004	0.012	0.078	0.006	1	0.012	0.92	0.001	0.007	0.002	0.013	0.005	0.001	0.015	0.027	2	0.045	0.004	0.66	0.055	0.055	0.096	0.043	0.024	0.015	0	3	0.017	0.002	0.031	0.53	0.058	0.24	0.073	0.03	0.01	0.004	4	0.007	0.003	0.043	0.03	0.82	0.045	0.031	0.009	0.012	0	5	0.007	0	0.014	0.059	0.034	0.85	0.016	0.021	0.001	0	6	0.005	0.005	0.029	0.069	0.044	0.062	0.77	0.005	0.006	0	7	0.016	0.005	0.018	0.029	0.097	0.094	0.006	0.73	0.002	0.003	8	0.031	0.017	0.004	0.019	0.002	0.007	0.003	0.006	0.9	0.007	9	0.1	0.19	0.01	0.029	0.01	0.024	0.029	0.024	0.047	0.53	<p>Overall acc: 75.3%</p> <p>Class 9 and class 3 has the worst accuracy because they both are the tail in long-dataset. Without applying anything, the dataset in that class is not enough to get a good accuracy</p>
	0	1	2	3	4	5	6	7	8	9																																																																																																																
0	0.8	0.011	0.039	0.019	0.018	0.018	0.004	0.012	0.078	0.006																																																																																																																
1	0.012	0.92	0.001	0.007	0.002	0.013	0.005	0.001	0.015	0.027																																																																																																																
2	0.045	0.004	0.66	0.055	0.055	0.096	0.043	0.024	0.015	0																																																																																																																
3	0.017	0.002	0.031	0.53	0.058	0.24	0.073	0.03	0.01	0.004																																																																																																																
4	0.007	0.003	0.043	0.03	0.82	0.045	0.031	0.009	0.012	0																																																																																																																
5	0.007	0	0.014	0.059	0.034	0.85	0.016	0.021	0.001	0																																																																																																																
6	0.005	0.005	0.029	0.069	0.044	0.062	0.77	0.005	0.006	0																																																																																																																
7	0.016	0.005	0.018	0.029	0.097	0.094	0.006	0.73	0.002	0.003																																																																																																																
8	0.031	0.017	0.004	0.019	0.002	0.007	0.003	0.006	0.9	0.007																																																																																																																
9	0.1	0.19	0.01	0.029	0.01	0.024	0.029	0.024	0.047	0.53																																																																																																																
Experiment 2 (Apply reweighting loss)	Comments																																																																																																																									
<table><tr><th></th><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th></tr><tr><th>0</th><td>0.8</td><td>0.023</td><td>0.073</td><td>0.007</td><td>0.019</td><td>0.008</td><td>0.006</td><td>0.022</td><td>0.027</td><td>0.013</td></tr><tr><th>1</th><td>0.01</td><td>0.92</td><td>0.008</td><td>0.004</td><td>0.003</td><td>0.008</td><td>0.004</td><td>0.011</td><td>0.005</td><td>0.027</td></tr><tr><th>2</th><td>0.044</td><td>0.002</td><td>0.69</td><td>0.034</td><td>0.082</td><td>0.094</td><td>0.038</td><td>0.014</td><td>0.001</td><td>0.003</td></tr><tr><th>3</th><td>0.016</td><td>0.007</td><td>0.081</td><td>0.51</td><td>0.085</td><td>0.2</td><td>0.077</td><td>0.021</td><td>0.002</td><td>0.007</td></tr><tr><th>4</th><td>0.014</td><td>0.001</td><td>0.06</td><td>0.034</td><td>0.79</td><td>0.033</td><td>0.02</td><td>0.043</td><td>0.003</td><td>0</td></tr><tr><th>5</th><td>0.005</td><td>0.003</td><td>0.03</td><td>0.095</td><td>0.046</td><td>0.77</td><td>0.018</td><td>0.037</td><td>0</td><td>0</td></tr><tr><th>6</th><td>0.003</td><td>0.003</td><td>0.059</td><td>0.063</td><td>0.052</td><td>0.032</td><td>0.77</td><td>0.008</td><td>0.004</td><td>0.001</td></tr><tr><th>7</th><td>0.013</td><td>0.003</td><td>0.031</td><td>0.022</td><td>0.041</td><td>0.059</td><td>0.009</td><td>0.82</td><td>0</td><td>0.002</td></tr><tr><th>8</th><td>0.1</td><td>0.047</td><td>0.017</td><td>0.011</td><td>0.006</td><td>0.009</td><td>0.008</td><td>0.005</td><td>0.79</td><td>0.011</td></tr><tr><th>9</th><td>0.053</td><td>0.16</td><td>0.022</td><td>0.009</td><td>0.004</td><td>0.01</td><td>0.006</td><td>0.033</td><td>0.018</td><td>0.69</td></tr></table>		0	1	2	3	4	5	6	7	8	9	0	0.8	0.023	0.073	0.007	0.019	0.008	0.006	0.022	0.027	0.013	1	0.01	0.92	0.008	0.004	0.003	0.008	0.004	0.011	0.005	0.027	2	0.044	0.002	0.69	0.034	0.082	0.094	0.038	0.014	0.001	0.003	3	0.016	0.007	0.081	0.51	0.085	0.2	0.077	0.021	0.002	0.007	4	0.014	0.001	0.06	0.034	0.79	0.033	0.02	0.043	0.003	0	5	0.005	0.003	0.03	0.095	0.046	0.77	0.018	0.037	0	0	6	0.003	0.003	0.059	0.063	0.052	0.032	0.77	0.008	0.004	0.001	7	0.013	0.003	0.031	0.022	0.041	0.059	0.009	0.82	0	0.002	8	0.1	0.047	0.017	0.011	0.006	0.009	0.008	0.005	0.79	0.011	9	0.053	0.16	0.022	0.009	0.004	0.01	0.006	0.033	0.018	0.69	<p>Overall acc: 75.5%</p> <p>Applying for reweighting loss helps class 9's accuracy (Only 152 data) increase significantly from 0.53 to 0.69. Class 7's accuracy also increased from 0.73 to 0.82</p>
	0	1	2	3	4	5	6	7	8	9																																																																																																																
0	0.8	0.023	0.073	0.007	0.019	0.008	0.006	0.022	0.027	0.013																																																																																																																
1	0.01	0.92	0.008	0.004	0.003	0.008	0.004	0.011	0.005	0.027																																																																																																																
2	0.044	0.002	0.69	0.034	0.082	0.094	0.038	0.014	0.001	0.003																																																																																																																
3	0.016	0.007	0.081	0.51	0.085	0.2	0.077	0.021	0.002	0.007																																																																																																																
4	0.014	0.001	0.06	0.034	0.79	0.033	0.02	0.043	0.003	0																																																																																																																
5	0.005	0.003	0.03	0.095	0.046	0.77	0.018	0.037	0	0																																																																																																																
6	0.003	0.003	0.059	0.063	0.052	0.032	0.77	0.008	0.004	0.001																																																																																																																
7	0.013	0.003	0.031	0.022	0.041	0.059	0.009	0.82	0	0.002																																																																																																																
8	0.1	0.047	0.017	0.011	0.006	0.009	0.008	0.005	0.79	0.011																																																																																																																
9	0.053	0.16	0.022	0.009	0.004	0.01	0.006	0.033	0.018	0.69																																																																																																																
Experiment 3 (Apply resampling balance)	Comments																																																																																																																									
<table><tr><th></th><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th></tr><tr><th>0</th><td>0.69</td><td>0.027</td><td>0.092</td><td>0.018</td><td>0.044</td><td>0.01</td><td>0.029</td><td>0.005</td><td>0.069</td><td>0.0089</td></tr><tr><th>1</th><td>0.0051</td><td>0.94</td><td>0.0069</td><td>0.012</td><td>0.002</td><td>0.002</td><td>0.007</td><td>0.0061</td><td>0.011</td><td>0.012</td></tr><tr><th>2</th><td>0.033</td><td>0.003</td><td>0.69</td><td>0.051</td><td>0.092</td><td>0.033</td><td>0.09</td><td>0.002</td><td>0.0091</td><td>0</td></tr><tr><th>3</th><td>0.0041</td><td>0.005</td><td>0.086</td><td>0.63</td><td>0.077</td><td>0.12</td><td>0.068</td><td>0.01</td><td>0.0041</td><td>0.002</td></tr><tr><th>4</th><td>0.001</td><td>0.004</td><td>0.018</td><td>0.021</td><td>0.84</td><td>0.037</td><td>0.046</td><td>0.014</td><td>0.023</td><td>0</td></tr><tr><th>5</th><td>0</td><td>0.002</td><td>0.066</td><td>0.14</td><td>0.046</td><td>0.69</td><td>0.014</td><td>0.019</td><td>0.01</td><td>0.0059</td></tr><tr><th>6</th><td>0.001</td><td>0.002</td><td>0.026</td><td>0.067</td><td>0.034</td><td>0.0061</td><td>0.85</td><td>0.003</td><td>0.0081</td><td>0.0009</td></tr><tr><th>7</th><td>0.002</td><td>0.0099</td><td>0.044</td><td>0.044</td><td>0.13</td><td>0.079</td><td>0.021</td><td>0.65</td><td>0.0091</td><td>0.0069</td></tr><tr><th>8</th><td>0.061</td><td>0.017</td><td>0.02</td><td>0.0098</td><td>0.007</td><td>0.002</td><td>0.012</td><td>0</td><td>0.86</td><td>0.0069</td></tr><tr><th>9</th><td>0.028</td><td>0.26</td><td>0.018</td><td>0.031</td><td>0.013</td><td>0.013</td><td>0.017</td><td>0.018</td><td>0.058</td><td>0.55</td></tr></table>		0	1	2	3	4	5	6	7	8	9	0	0.69	0.027	0.092	0.018	0.044	0.01	0.029	0.005	0.069	0.0089	1	0.0051	0.94	0.0069	0.012	0.002	0.002	0.007	0.0061	0.011	0.012	2	0.033	0.003	0.69	0.051	0.092	0.033	0.09	0.002	0.0091	0	3	0.0041	0.005	0.086	0.63	0.077	0.12	0.068	0.01	0.0041	0.002	4	0.001	0.004	0.018	0.021	0.84	0.037	0.046	0.014	0.023	0	5	0	0.002	0.066	0.14	0.046	0.69	0.014	0.019	0.01	0.0059	6	0.001	0.002	0.026	0.067	0.034	0.0061	0.85	0.003	0.0081	0.0009	7	0.002	0.0099	0.044	0.044	0.13	0.079	0.021	0.65	0.0091	0.0069	8	0.061	0.017	0.02	0.0098	0.007	0.002	0.012	0	0.86	0.0069	9	0.028	0.26	0.018	0.031	0.013	0.013	0.017	0.018	0.058	0.55	<p>Overall acc: 73.9%</p> <p>Although the overall accuracy going down, resampling the dataset could increase the accuracy of class 3 and class 6 significantly. Class 9 with the fewest dataset also increased from 0.53 to 0.55.</p>
	0	1	2	3	4	5	6	7	8	9																																																																																																																
0	0.69	0.027	0.092	0.018	0.044	0.01	0.029	0.005	0.069	0.0089																																																																																																																
1	0.0051	0.94	0.0069	0.012	0.002	0.002	0.007	0.0061	0.011	0.012																																																																																																																
2	0.033	0.003	0.69	0.051	0.092	0.033	0.09	0.002	0.0091	0																																																																																																																
3	0.0041	0.005	0.086	0.63	0.077	0.12	0.068	0.01	0.0041	0.002																																																																																																																
4	0.001	0.004	0.018	0.021	0.84	0.037	0.046	0.014	0.023	0																																																																																																																
5	0	0.002	0.066	0.14	0.046	0.69	0.014	0.019	0.01	0.0059																																																																																																																
6	0.001	0.002	0.026	0.067	0.034	0.0061	0.85	0.003	0.0081	0.0009																																																																																																																
7	0.002	0.0099	0.044	0.044	0.13	0.079	0.021	0.65	0.0091	0.0069																																																																																																																
8	0.061	0.017	0.02	0.0098	0.007	0.002	0.012	0	0.86	0.0069																																																																																																																
9	0.028	0.26	0.018	0.031	0.013	0.013	0.017	0.018	0.058	0.55																																																																																																																

ANALYSIS AND CONCLUSIONS

From the experiment above, we could see that the accuracy graph for both training and test does not change with or without applying loss reweighting or resampling dataset. The overall test accuracy stays around 75%. But applying reweighting or resampling at a long-tail dataset is important. We could see in the confusion matrices that without apply any balancing effort, our model just learns from class with many datasets only, so our class with few datasets like class 9 will have really bad accuracy at only 0.51.

After apply reweighting loss and resample, our overall accuracy did not improve, but we could see that class from a few numbers of datasets have some improvement in terms of accuracy. But we should realize that accuracy in some of the class is also decreased. It is better to make sure the model learns from all the classes in the datasets than keep it learn from classes with many datasets only. We should remember there is no free lunch theory, so **apply loss reweighting or rebalancing samples does not mean that the overall accuracy is higher, but it will make sure the model will learn from all the class even from a class with really few datasets.**

It is not always when we get a high overall accuracy then it means our model is a good one especially when dealing with an imbalanced dataset. What we should try, is to get a higher precision with a higher recall value by examining **confusion matrix**. But there is often a **trade-off** between these two metrics. There is a general inverse relation between these two metrics. Meaning if we increase one, more often than not we will decrease the other.

REFERENCE

https://www.tensorflow.org/tutorials/structured_data/imbalanced_data

<https://github.com/ufoym/imbalanced-dataset-sampler>

<https://github.com/MaxHalford/pytorch-resample>

<https://elitedatascience.com/imbalanced-classes>

<https://medium.com/analytics-vidhya/accuracy-on-imbalanced-datasets-and-why-you-need-confusion-matrix-937613bf89bf>