EE 443 Final Report

Jonathan Wong

Abstract

In this capstone project, I investigate five modeling techniques for improving model accuracy on an imbalanced class distribution. After exploring techniques in single model optimization techniques and ensemble learning, I found that stacking efficientnet models with weighted loss yields the best accuracy with an average 8% accuracy improvement across all dataset imbalance ratios. In addition to improved accuracy, this stacking with weighted loss approach is also lightweight with lower the 25 million hyperparameters. Compared with common ResNet architectures, this ensemble uses half the number of parameters while resulting in greater accuracy. Finally, this approach is easy to implement and explain making the approach applicable for practical use.

Objective

For this final project, student teams were tasked to improve accuracy upon a long-tailed dataset sampled from the CIFAR-100 dataset. Class imbalance is common in real-world data, and oftentimes the true underlying distribution. For instance, in medicine, collecting data such as chest scans or symptoms for patients with a rare disease is difficult as meeting with these patients is rare in the first place. Despite limited data, classification of rare events is often of more importance and developing techniques for accurate imbalanced classification is in active research.

Tooling

To perform experimentation, I leveraged libraries of PyTorch Lightning, Timm Models, and GridAI. The PyTorch Lightning library provides a lightweight wrapper around PyTorch that automates the model training loop in addition to other boilerplate code. The Timm Models library provides hundreds of state-of-the-art image classification backbones pretrained on ImageNet. Finally, GridAI is a startup cloud training service, similar to AWS and GCP, that provides easy hyperparameter sweeping at a low cost. These tools were invaluable for fast iteration and parallel training that enabled me, a one person team, to conduct multiple experiments at once.

Baseline

Before beginning experimentation, I developed a baseline as reference for evaluating the performance of future experiments. To establish a baseline, I used techniques of model architecture sweep and hyperparameter sweep. In the search space of the model architecture sweep, I included six models: resnet34, resnet50, resnet50d, densenet121, efficientnet0, and efficientnet2. Fixing the batch size and learning rate, I trained these for 10 epochs on the full

CIFAR-50 dataset and received the following results in Fig 1. I had selected these architectures as they were relatively small which would promote short training times and therefore faster experimentation. Among the architectures sweeped, I selected **efficientnet0** as the base model architecture as it showed the highest performance for the fewest parameters.

Model Name	Test Accuracy	# Parameters
efficientnet_b2	0.804	10 M
resnet50d	0.793	22 M
efficientnet_b1	0.7892	8 M
efficientnet_b0	0.788	5 M
densenet121	0.770	20 M
resnet50	0.758	22 M
resnet34d	0.730	20 M

Fig 1: **Architecture Sweep Results**

Next, to perform hyperparameter sweeping, I defined a search space on the batch size and learning rate for each dataset. The results of hyperparameter sweeping are summarized in Fig 2. Fig 2. summarizes the baselines for each dataset.

Dataset	Test Accuracy
Full	0.792
0.1	0.7296
0.02	0.6304
0.005	0.5446

Fig 2: **Baseline Accuracies**.

Experiments

Following the baseline, I executed the following 5 experiment ideas in listed order:

1. Artificial Balancing
2. Weighted Loss
3. Ensemble Training: Bagging
4. Ensemble Training: Stacking
5. Stacking + Weighted Loss

Artificial Balancing

Artificial balancing is a data-centric, brute-force approach to imbalanced model training. The procedure is as follows. First, calculate the histogram of the training dataset. Then, provided a sampling threshold, determine the downsampling or upsampling rate to satisfy this threshold. A visual is shown in Fig. 3. The sampling rate is calculated by the following formula, with an example calculation:

Formula: Sampling Threshold = Threshold Value // Count

Ex: (Class = 0, Count = 500) => 300/500 => 0.6

(Class = 20, Count = 100) => 300/100 => 3

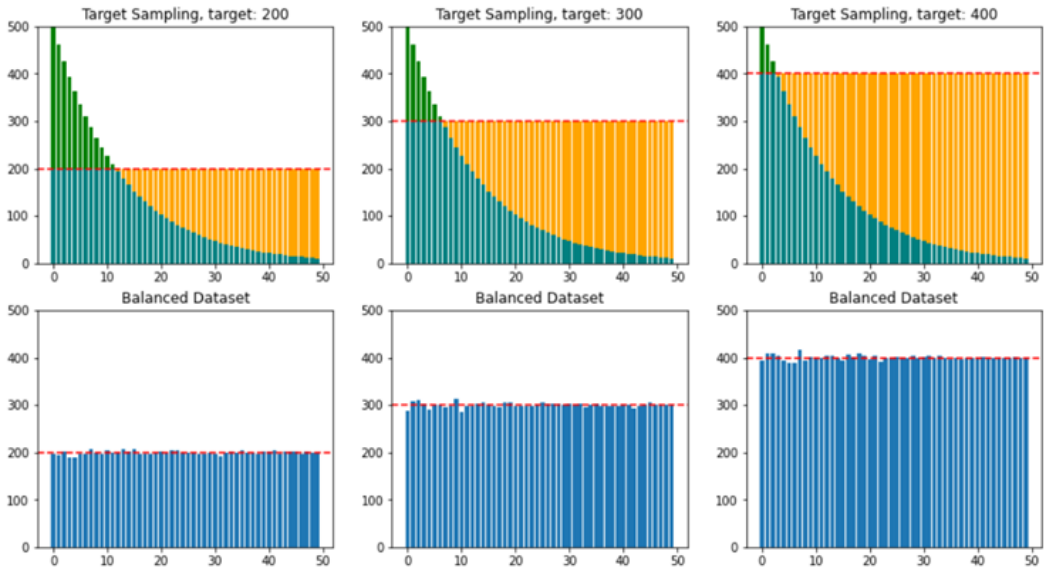


Fig 3: Artificial Balancing Algorithm. The green regions indicate samples that are downsampled and the orange regions indicate regions that are upsampled. The sampling threshold is configurable and may be sweeped for optimal performance.

Although this approach is effective for small datasets, as a dataset grows, producing an artificial dataset as a preprocessing step incurs significant memory and runtime costs. Additionally, the artificial balancing algorithm must be custom reimplemented in a new classification context with different labels. Therefore, this approach does not scale, which motivated the next experiment in sequence.

Weighted Loss

Weighted loss is a model-centric, more elegant approach to imbalanced model training. Instead of conducting a manual preprocessing step to produce an artificially balanced dataset, we may define weights to samples of the minority class such that these minority samples play a big role in the loss computation aggregate whereas majority samples play less of a role. Therefore, we can mathematically emulate the presence of more minority samples without actually duplicating these data points. I tested three weighted loss equations defined below. Each of the equations are different variations of weighting rarer samples with greater weight:

- Inverse Number of Samples (INS):

$$w_{n,c} = \frac{1}{Number\ of\ Samples\ in\ Class\ c}$$

- Inverse Square Number of Samples (ISNS):

$$w_{n,c} = \frac{1}{\sqrt[2]{Number\ of\ Samples\ in\ Class\ c}}$$

- Effective Number of Samples (ENS) [1]:

$$w_{n,c} = \frac{1}{E_{n_c}} \qquad E_{n_c} = \frac{1-\beta^{n_c}}{1-\beta}$$

Based on experimental results, the choice of weighted loss function did not matter all too much, with each weighted loss function exhibiting the same improvement in performance. The similarity of performance is demonstrated in Fig 4.

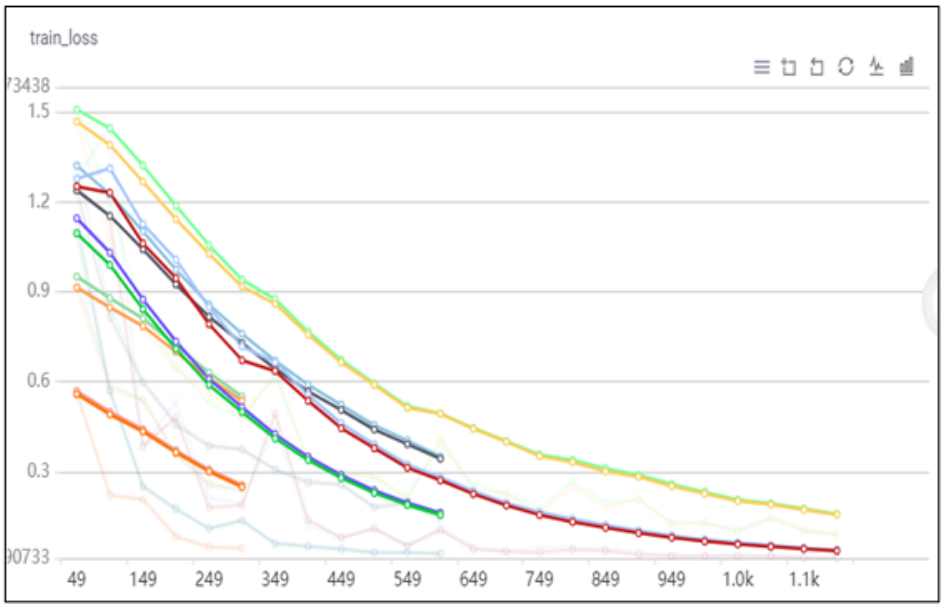


Fig 4: Weighted Loss Sweep. Each pair of curves indicate two different loss functions which are trained under an identical set of hyperparameter conditions. In this case, weighted loss functions produce the same optimal weights which is why two different functions produce the same loss trajectories.

Ensemble Training: Bagging

To this point, all experiments have focused on improving the performance on a single model, by potentially optimizing its data preparation, model architecture, and loss function. Ensemble training enables us to leverage the gains of the previous experiments across multiple models. Bagging is a technique that involves training multiple copies of the same architecture in parallel on different random subsets of the training data. While conventional bagging forms learner datasets randomly, random sampling would produce imbalanced datasets which are sub-optimal for training. Therefore, I developed a partitioning algorithm that copies the minority class samples to each learner dataset and a random share of the majority dataset. The algorithm goes as follows:

1) Compute the training dataset histogram
2) Partition the dataset into majority and minority classes according to the median value
3) Copy the minority samples to each dataset
4) Shuffle and deal the majority samples to each dataset

The bagging algorithm is visualized in Fig 5 and Fig 6.

While this dataset curation does not produce a perfectly balanced dataset, it is certainly helpful. We can reuse weighted loss from the previous experiment to compensate for this balance programmatically. After training, this technique does indeed yield additional performance, which shows promise for future ensembling techniques.

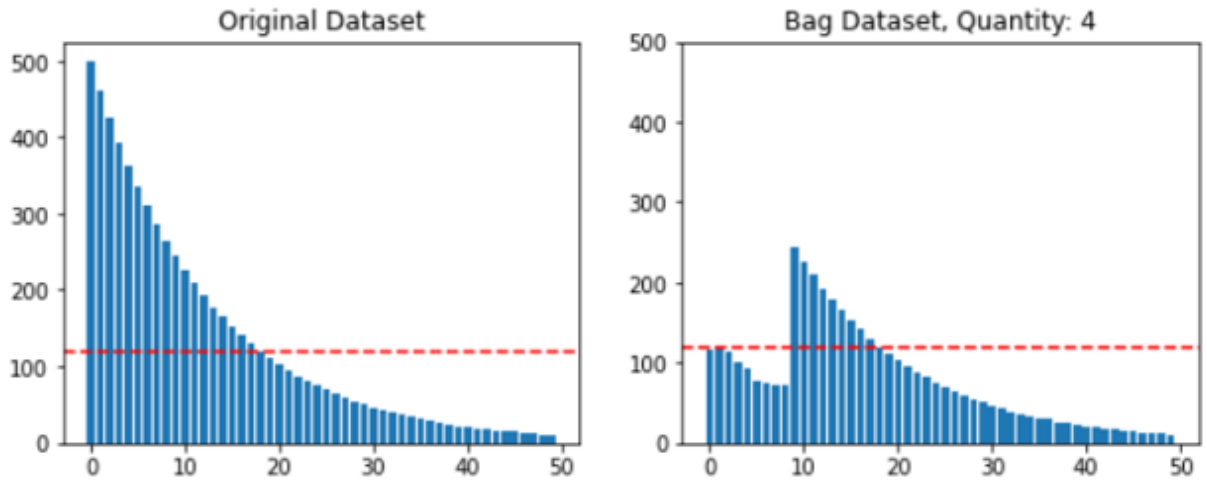


Fig 5: Balanced Bagging. The left plot is a histogram of the original imbalanced dataset. The right plot is the histogram of a “balanced” bag produced by the dataset curation algorithm. The number of learners is determined by a specified sampling threshold, indicated by the red dotted line. The equation for the number of learners is simple, simply the maximum sampling count // the specified threshold. In this case, 500 // 120 yields 4 learners.

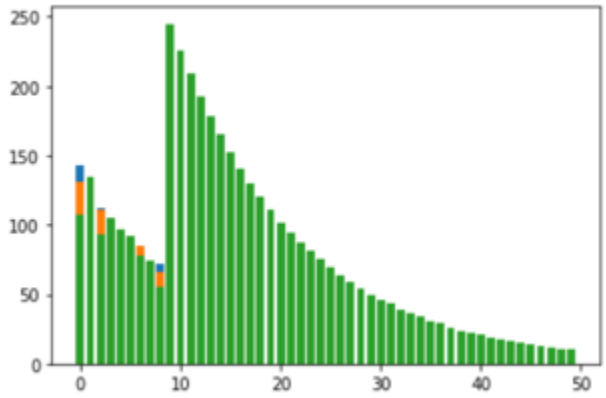


Fig 6: Representative Learner Dataset. Each dataset receives a copy of all minority classes but a random split of the majority classes. Therefore, all learners receive signal from the minority classes but learn different representations based on the majority samples they are provided.

Ensemble Training: Stacking

To improve the predictive power of the model ensemble, each learner in the ensemble needs to improve its accuracy. With bagging, each learner only had access to a balanced subset of the training dataset. Therefore, there is still more potential for each learner to improve if each is exposed to the entire dataset. Stacking involves training each learner on the entire training dataset in which each model architecture employs a different feature extractor backbone.
For this stacking experiment, I selected an ensemble of three models of comparable size and performance, efficientnet0 through efficientnet2.

For both bagging and stacking, a new module is required to fuse the trained learners into a single predictor. The essential logic for this module is defined in Fig 7. In which the outputs of each model's softmax output is weighted equally into the final prediction.

```
class ModelEnsemble(LightningModule):
  def __init__(self, models):
    super().__init__()
    # Create Models
    self.models = models

    # For Metrics
    self.test_acc = torchmetrics.Accuracy()
    self.preds = []
    self.labels = []

  def forward(self, x):
    prediction = torch.zeros(50).cuda()
    weight = 1.0 / len(self.models)
    for model in self.models:
      model.eval()
      prediction = prediction + (weight * F.softmax(model(x), dim=1))
    return prediction
```

Fig 7: Model Ensemble Inference Module. In the module's constructor, the inference module stores a list of trained models. In the module's forward pass, the input is passed through each model and the softmax predictions are averaged as the final prediction.

As the results show, training on the entire dataset improves the average learner's accuracy which results in a higher ensemble accuracy.

Stacking and Weighted Loss

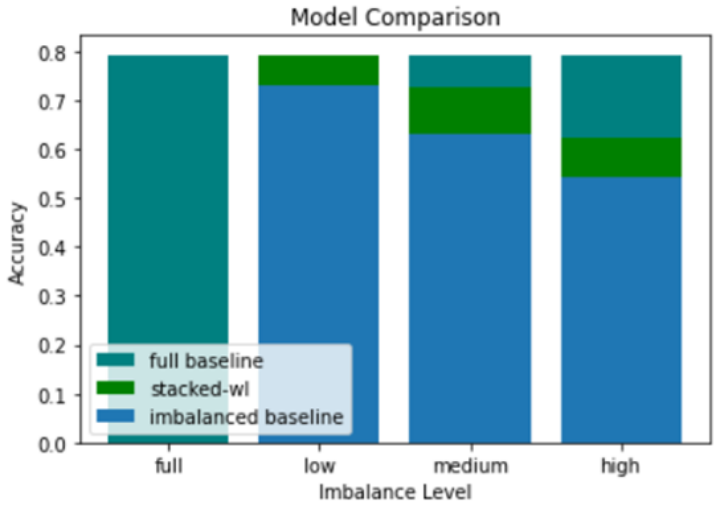
My final experiment combines the best techniques from individual model optimization (weighted loss) and ensemble training (stacking). By simply changing a single parameter in my ensemble training script, this experiment achieves significant accuracy gains across all datasets.

Results

All training results are summarized by the following table. The first two experiments, artificial balancing and weighted loss, focus on a single model optimization. The next two experiments, bagging and stacking, focus on ensemble techniques. The last experiment combines the best results from these two experiment categories to produce maximum results.

Dataset	Baseline	Artificial Balancing	Weighted Loss	Bagging / WL	Stacking	Stacking / WL
Full	0.792					
0.1	0.7296	0.733	0.7458	0.7350	0.7720	0.7933
0.02	0.6304	0.6788	0.6798	0.6760	0.6614	0.7253
0.005	0.5446	0.5688	0.5352	0.5454	0.5626	0.6218
Average Improvement:		0.024	0.019	0.015	0.030	0.079

Extracting the baseline and final column, we may visualize the performance boost achieved by the last experiment in the following bar chart:



Combining stacking and weighted loss produces a neat staircase effect. The modelling technique is able to achieve the baseline accuracy of the next imbalance class up. Additionally, the ensemble model is economic in the number of parameters. Referencing Fig 1, the combined number of hyperparameters of the ensemble closely matches that of resnet50d.

Conclusion:

Following experimentation of single model optimization techniques and ensemble learning techniques, stacking efficientnet models with weighted loss yields the best accuracy with an average 8% accuracy improvement across all dataset imbalance ratios. In addition to improved accuracy, this stacking with weighted loss approach is also economic with lower the 25 million hyperparameters. Compared with common ResNet architectures, this ensemble uses half the number of parameters while resulting in greater accuracy. Finally, this approach is easy to implement and explain, which is appropriate for practical use.

References

[1] “Class Balanced Loss of Effective Number of Samples”, Cui https://openaccess.thecvf.com/content_CVPR_2019/papers/Cui_Class-Balanced_Loss_Based_on_Effective_Number_of_Samples_CVPR_2019_paper.pdf