Midterm Project Report: Class Imbalance Problem

By: Michael Schmidt

A20460468

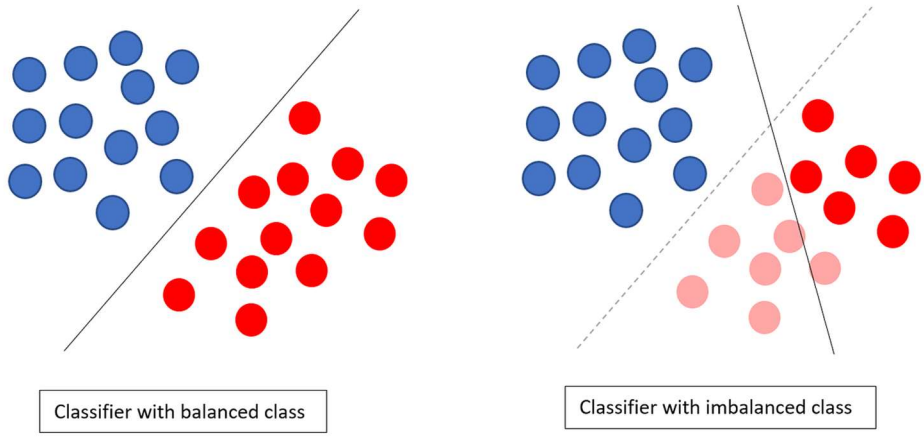
ECE 508-01

Contents

Introduction: Class Imbalance Problem

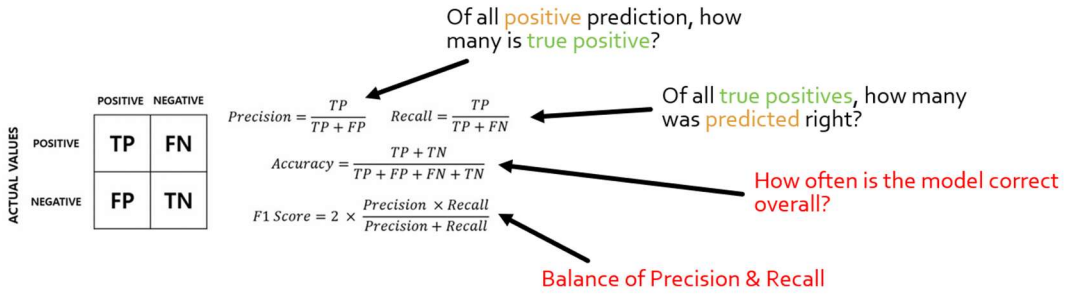
The problem with the class imbalance in datasets starts with the real world where the data samples collected are not uniform. It is difficult to collect rich samples where every class has the same number of samples in them. Most often, different classes will have a different number of samples, which results in the model becoming biased towards head classes, as they are seen more often. While head classes will thrive, tail classes will suffer in performance.

An intuitive example to imagine this scenario is say there is a machine that sorts green apples from red apples mostly based on their color. The machine only wants green apples and throws away anything else. It has been trained to handle the deformation of the fruits and works well for most apples. Now let's say a green pear was thrown into the mix. The machine does not know how to handle such a case, and may misclassify the pear as a green apple, as it has not seen such a case often during training.



[Figure 1 & 2: Images demonstrating how a model will be biased towards class with imbalanced datasets. The left image shows how a classifier can clearly predict the boundary between two different classes with same sample. The Image on the right is how the classifier is skewed when there are less samples.]

When evaluating the performance of models regarding tail classes, it is important to understand that simple metrics, such as overall accuracy, will not suffice in this case, as it is a general score of the accuracy over all classes. To look deeper, this paper will utilize other metrics, such as precision, recall, f1-score, and a confusion matrix, to help determine the performance of the model regarding the tail classes.



[Figure 3: Explanation of the definitions of the evaluation metrics of precision, recall, accuracy, and F1-score]

Explanation of the dataset:

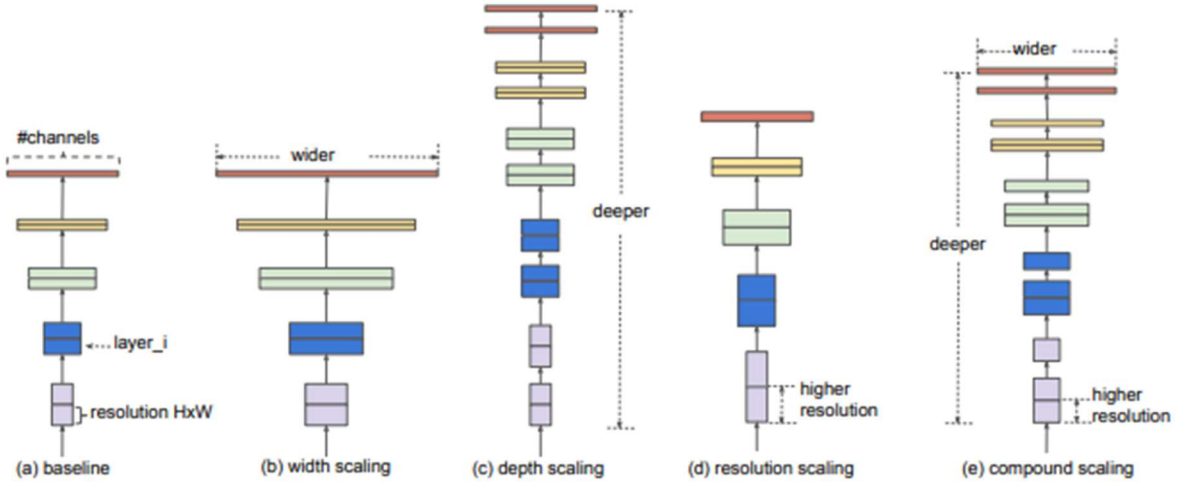
The dataset that will be used in the paper is a custom version of the CIFAR-10 Long Tailed distribution dataset. The Original CIFAR-10 dataset was published by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton in 2009 to provide a benchmark dataset to test image classification models. The original dataset had 10 classes with a balanced distribution between them. It had 5,000 samples per class and had 1,000 samples per class for testing. In total, it contained 60,000 sample images of size 32x32. While the dataset is large, due to training resource limitations, a modified version of this dataset will be used.

In the custom version of the dataset, 10 classes will be used for training, however each class will have a different number of samples in them. With 27,500 samples for training, another 10,000 will be reserved for testing. The testing images will have a balanced distribution. While this dataset is smaller, training will be much faster and can provide an idea on how it will perform on a larger dataset.

[Figure 4: Image distribution of the classes in the custom CIFAR-10 LT dataset. There are head classes that have more samples than the tail classes]

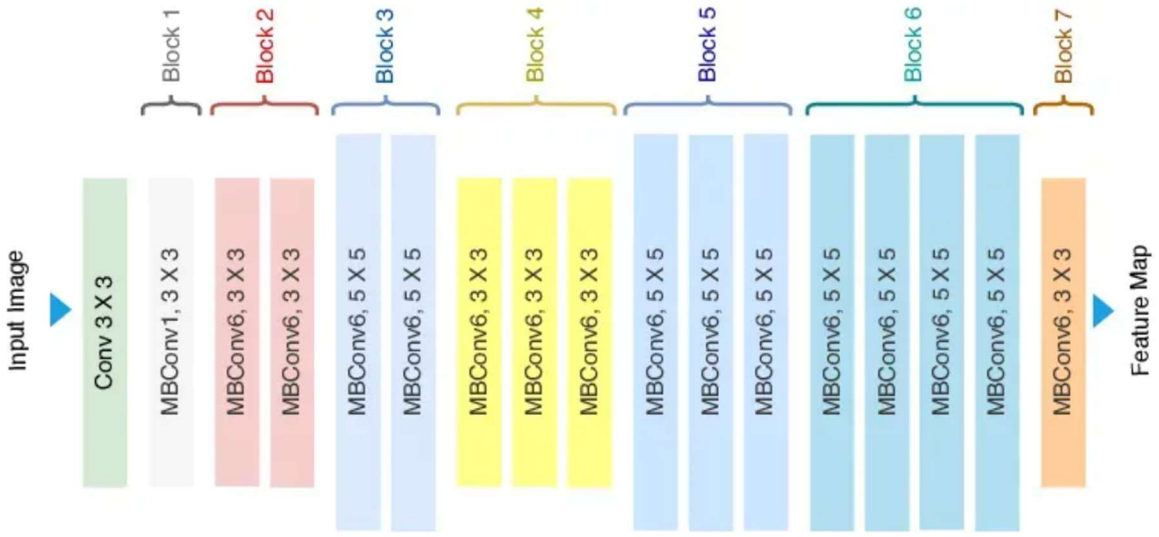
Baseline Model Description: EfficientNet-B0:

The model that will be used in this paper is EfficientNet-B0 [1]. This model utilizes a technique called “compound scaling” that automatically scales the depth, width, and input resolution of the model to maximize efficiency. This is an important feature, as to improve the performance on traditional CNN’s like ResNet, they would increase either of the three mentioned above, but it would have drawbacks. Increasing the depth so the model can learn richer features, may make the model overfit or have vanishing gradients. Increasing the width of the layers so the model can learn more features at once is computationally expensive and can slow the model. Increasing the input size to give the model more information to learn from is computationally expensive as well. By scaling these 3 areas of the model with Efficient Net, the model should be accurate while maintaining efficiency.



[Figure 5: Image demonstrating EfficientNet compound scaling]

In addition to compound scaling, EfficientNet-B0 utilizes MBConv blocks to build its model. MBConv or Mobile Inverted Bottleneck Convolution, is deprived from MobileNetv2's architecture to speed up computation on edge devices. It is inverted as traditionally, CNNs would take a large input, compress it to learn features, then expand the output. This does the opposite, where it takes a small input image, expands it to learn features, does a Max pooling and decreases it again before outputting the results. It is bottlenecked as the input is a small image. In each block, it used a squeeze and excitation attention module, with a swish activation function.



[Figure 6: Architecture of EfficientNet-B0]

Performance on Dataset:

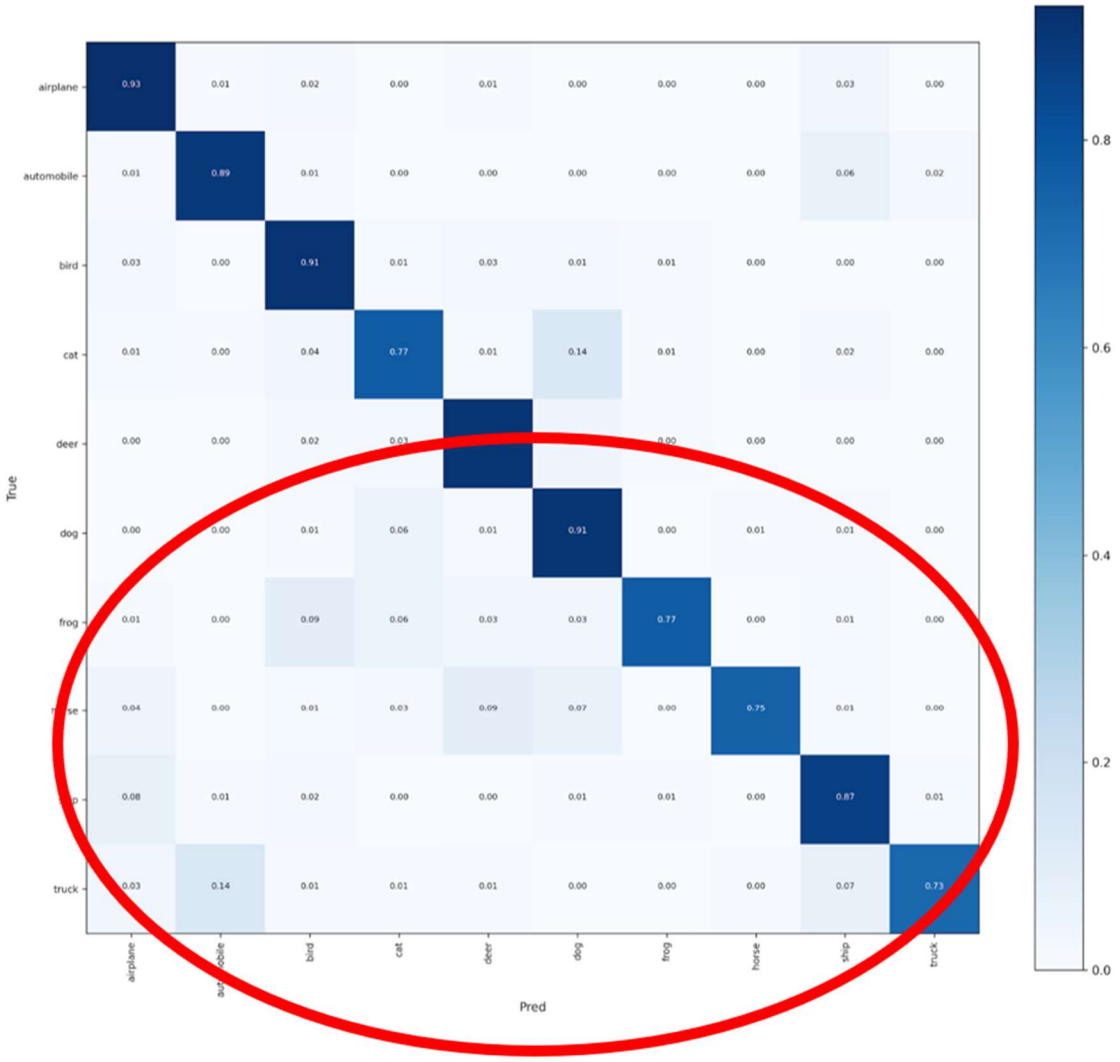
Before beginning to improve the model, there are some limits to this study. The default model parameters must remain the same for all cases. There parameters are LR = 1e-4, Decay = 0.9999, Epoch = 30, Batch Size = 128.

Overall Accuracy : 84.22%

	precision	recall	f1-score	support
airplane	0.8149	0.9290	0.8682	1000
automobile	0.8494	0.8910	0.8697	1000
bird	0.8053	0.9100	0.8545	1000
cat	0.8054	0.7700	0.7873	1000
deer	0.8282	0.9060	0.8653	1000
dog	0.7542	0.9050	0.8227	1000
frog	0.9564	0.7680	0.8519	1000
horse	0.9727	0.7480	0.8457	1000
ship	0.7972	0.8690	0.8316	1000
truck	0.9441	0.7260	0.8208	1000
accuracy			0.8422	10000
macro avg	0.8528	0.8422	0.8418	10000
weighted avg	0.8528	0.8422	0.8418	10000

[Figure 7: Metrics of baseline Model]

After running the baseline model with the dataset, it achieved an overall accuracy of 84.22%. Looking at the metrics of recall, head classes of “airplane”, “automobile”, “bird”, have a higher recall, while tail classes such as “truck”, “horse”, “frog”, have lower recall.



[Figure 8: Confusion Matrix of baseline model]

In the confusion matrix above, there are more misclassifications on the lower half of the image that must be improved. While the diagonal head classes are strong, the tail classes are much more dispersed, particularly on the lower left side. This means that the model is predicting a few tail classes as head classes. The goal with the improvements in this paper is to reduce these misclassifications and try to make the diagonals more equally distributed.

Improvements Made:

Resampling with Class Balanced Sampling:

Currently the data loader in the default code is "shuffling" the data samples to chose from. This would mean that every sample has the same probability of being chosen for the batch. This leads to an inequal distribution of samples being fed into the model, as the model will see head class samples more often than the tail class.

[2] Class Balanced sampling fixes this issue by having the data loader choose samples based on the class and the frequency of the samples in them. This works by counting the samples in each class and calculates the weights for each sample of the class based on the number of samples there are. The formula would be $\frac{1}{n}$ where n is the number of samples from the class. Then update the new weight to the individual samples of that class.

Doing this ensures that the samples from all the different classes have the same probability of being chosen.

No Modification:

Overall Accuracy : 84.22%

	precision	recall	f1-score	support
airplane	0.8149	0.9290	0.8682	1000
automobile	0.8494	0.8910	0.8697	1000
bird	0.8053	0.9100	0.8545	1000
cat	0.8054	0.7700	0.7873	1000
deer	0.8282	0.9060	0.8653	1000
dog	0.7542	0.9050	0.8227	1000
frog	0.9564	0.7680	0.8519	1000
horse	0.9727	0.7480	0.8457	1000
ship	0.7972	0.8690	0.8316	1000
truck	0.9441	0.7260	0.8208	1000
accuracy			0.8422	10000
macro avg	0.8528	0.8422	0.8418	10000
weighted avg	0.8528	0.8422	0.8418	10000

Class-Balanced Sampling:

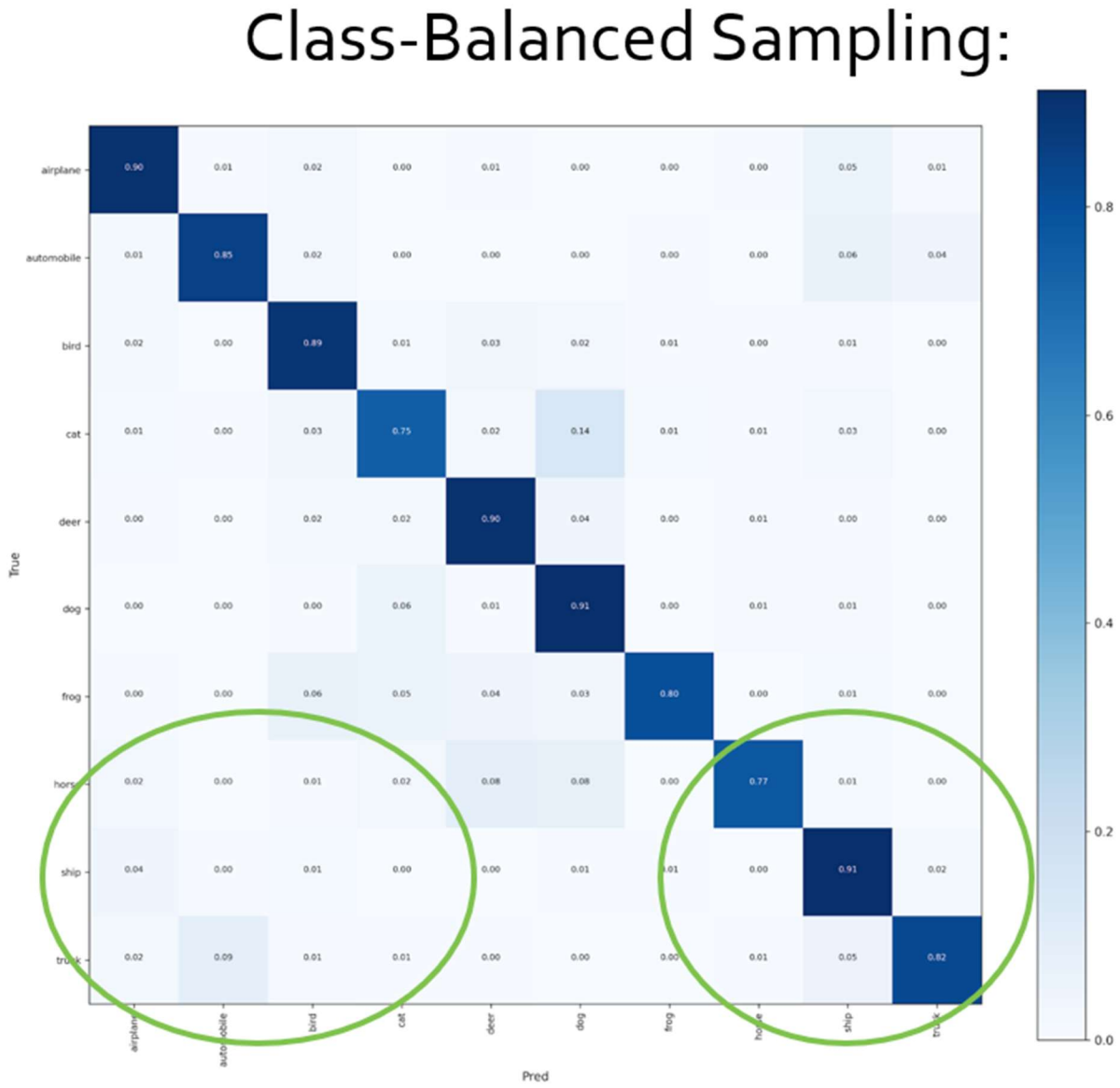
Overall Accuracy : 84.97%

	precision	recall	f1-score	support
airplane	0.8653	0.8990	0.8818	1000
automobile	0.8929	0.8500	0.8709	1000
bird	0.8218	0.8900	0.8545	1000
cat	0.8139	0.7480	0.7796	1000
deer	0.8280	0.9000	0.8625	1000
dog	0.7378	0.9060	0.8133	1000
frog	0.9558	0.8010	0.8716	1000
horse	0.9530	0.7700	0.8518	1000
ship	0.7965	0.9120	0.8503	1000
truck	0.9112	0.8210	0.8638	1000
accuracy			0.8497	10000
macro avg	0.8576	0.8497	0.8500	10000
weighted avg	0.8576	0.8497	0.8500	10000

[Figure 9: Metrics of before and after adding CB-sampling]

Based on the image above, there is an improvement of the predictions for tail classes after the adjustment. Most tail classes have a significant performance, while the head classes suffer. This would make sense as head classes are no longer shown as often, making the model not learn many features from them anymore. Overall accuracy has increased by 0.75%, having a total of 84.97%.

The confusion matrix below shows there is an increase in performance in the bottom corners of the matrix, which means the model is not confusing the tail classes as head classes as often anymore.



[Figure 10: Confusion matrix of CB-sampling]

Data Augmenting with MixUp:

Currently, the model learns the loss by comparing the cross-entropy loss to a single hard label. This causes the decision boundaries to be sharp and guides the model to memorize rather than generalize the samples causing it too overfit.

This fix for this would be to use MixUp data augmentation technique. [3] MixUp works by replacing the batch of images with a linear combination of two random samples. This would effectively "blend" the two images together and their labels, so the model can learn to be more flexible in its decision making. It works by creating a linear combination of samples in the training loop using **λ** as the control for how much the images should blend. This would be applied to both the training images, and their labels, and can be fed into the loss function.

$$\begin{aligned} \hat{x} &= \lambda x_i + (1-\lambda)x_j, \\ \hat{y} &= \lambda y_i + (1-\lambda)y_j, \end{aligned}$$

where $\lambda \in [0, 1]$ is a random number

[Figure 11: Example case of how MixUp works with equation]

By combining both CB-sampling and MixUp, the accuracy of the model has now improved by 1.54% having a new overall accuracy of 85.76%. By testing, the optimal value for **λ** appears to be 0.4. Most of the tail classes have improved, including some of the middle classes. There is an outlier for the class "dog" where it is not improving as much.

No Modification:

Overall Accuracy : 84.22%

	precision	recall	f1-score	support
airplane	0.8149	0.9290	0.8682	1000
automobile	0.8494	0.8910	0.8697	1000
bird	0.8053	0.9100	0.8545	1000
cat	0.8054	0.7700	0.7873	1000
deer	0.8282	0.9060	0.8653	1000
dog	0.7542	0.9050	0.8227	1000
frog	0.9564	0.7680	0.8519	1000
horse	0.9727	0.7480	0.8457	1000
ship	0.7972	0.8690	0.8316	1000
truck	0.9441	0.7260	0.8208	1000
accuracy			0.8422	10000
macro avg	0.8528	0.8422	0.8418	10000
weighted avg	0.8528	0.8422	0.8418	10000

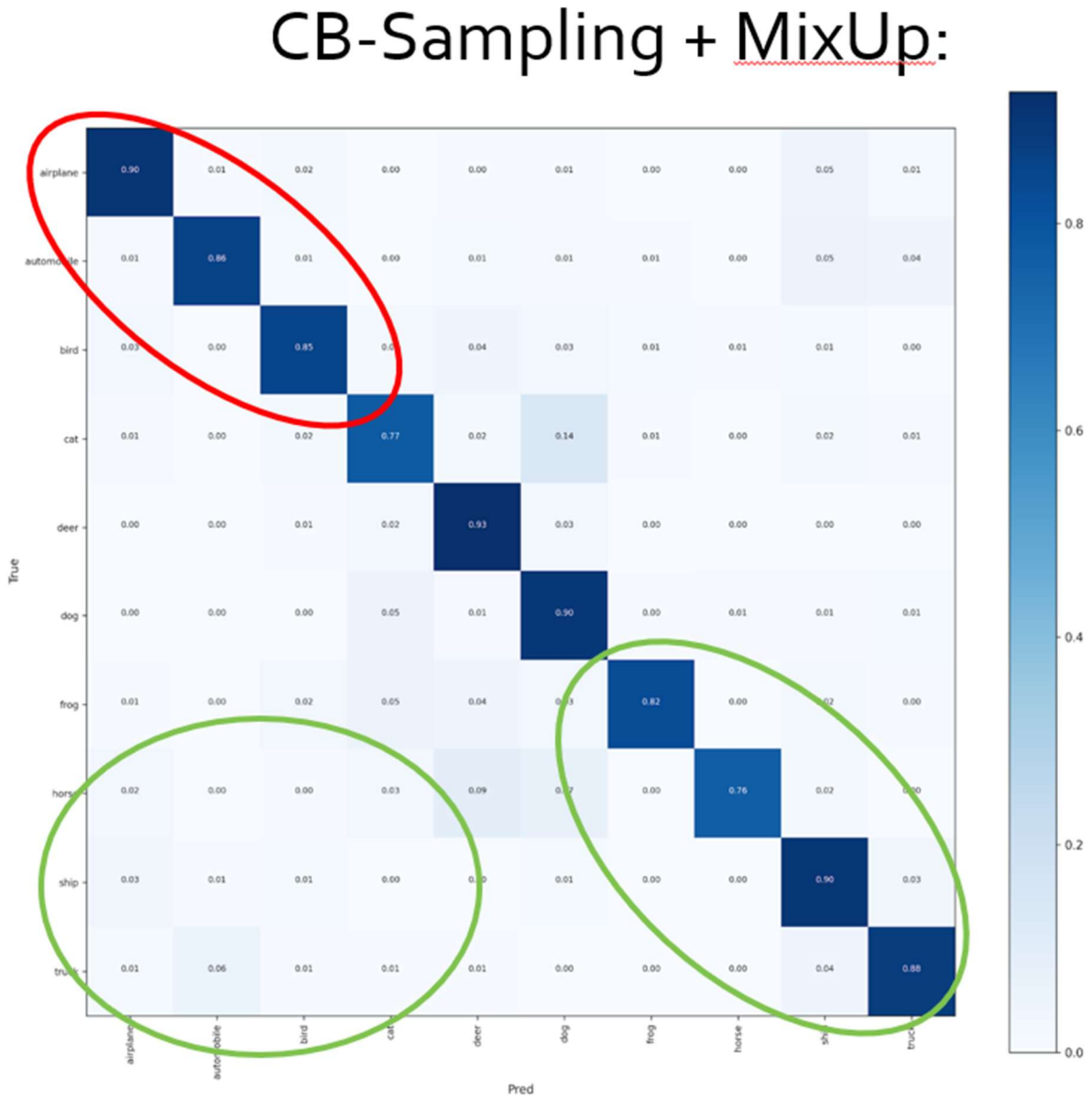
CB-Sampling + MixUp:

Overall Accuracy : 85.76%

	precision	recall	f1-score	support
airplane	0.8802	0.9040	0.8920	1000
automobile	0.9185	0.8570	0.8867	1000
bird	0.8931	0.8520	0.8721	1000
cat	0.8052	0.7730	0.7888	1000
deer	0.8026	0.9270	0.8603	1000
dog	0.7391	0.8980	0.8108	1000
frog	0.9549	0.8250	0.8852	1000
horse	0.9646	0.7630	0.8520	1000
ship	0.8094	0.9000	0.8523	1000
truck	0.8868	0.8770	0.8819	1000
accuracy			0.8576	10000
macro avg	0.8654	0.8576	0.8582	10000
weighted avg	0.8654	0.8576	0.8582	10000

[Figure 12: Metrics of combining CB-sampling with MixUp]

Looking at the confusion matrix below, there is a great improvement in the misclassifications for tail classes, where in the bottom left, tail classes are no longer being misclassified as head classes but rather be correctly identified in their diagonal positions. For the “dog” class, it seems the model is confusing it for a “cat” class the most. This could be that both classes have very similar features that the model is having difficulty setting the decision boundary for.



[Figure 13: Confusion Matrix of CB-sampling + MixUp]

Data Augmenting with CutMix:

In addition to the problem addressed with MixUp, another potential solution is to use CutMix. [4] CutMix works by selecting two random samples, cutting out a random rectangle from one sample, and past it in the same position on the other sample. This would help the model

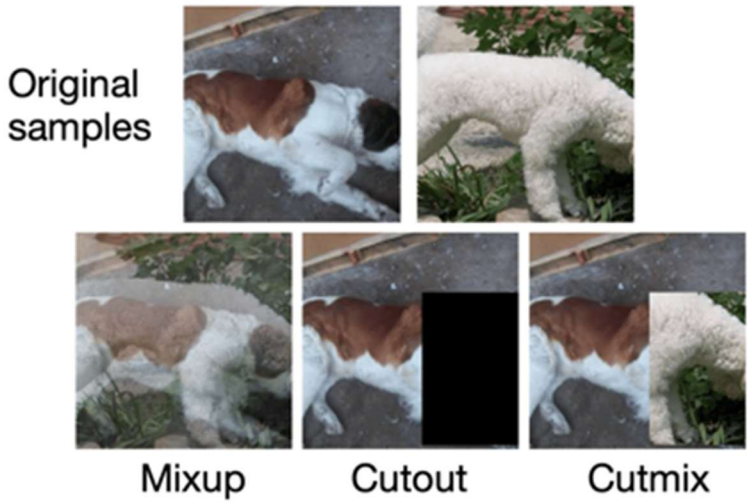
learn from multiple regions of an image, rather than focusing on just one. The model would develop spatial robustness and learn from soft labels.

The labels would be adjusted to be a linear representation of the area of one sample, and the area of another sample. The decision for the area of a sample is guided by the constraint $\boldsymbol{\lambda}$, where $\boldsymbol{\lambda}$ is the ratio of the patch over the entire image $\boldsymbol{\lambda = 1 - \frac{A_{patch}}{A_{image}}}$. To calculate the size of the CutMix patch, the width and height are multiplied by the square root of $\boldsymbol{\lambda}$.

$$\tilde{x} = \mathbf{M} \odot x_A + (\mathbf{1} - \mathbf{M}) \odot x_B \qquad r_x \sim \text{Unif}\ (0, W), \quad r_w = W\sqrt{1-\lambda},$$
$$\tilde{y} = \lambda y_A + (1-\lambda) y_B, \qquad r_y \sim \text{Unif}\ (0, H), \quad r_h = H\sqrt{1-\lambda}$$

[Figure 14 & 15: The left figure is the equation of making the new sample and its targeted label. The right image is how the patch size is calculated]

In addition to using CutMix traditionally, a slight modification is done where CutMix will only be applied if at least one of the two randomly selected images is a tail class. This decision was done so that the model can focus more on generalising the tail images in comparison to the head class, as it already performs well on those.



[Figure 16: Example of how CutMix is done]

After switching MixUp with the modified CutMix with CB-sampling, the performance did not seem to work well with the model. The metric below shows that it achieved an accuracy of 79.06%, which is a 5.16% decrease. While the tail classes have greatly improved, the model had great issues classifying the head classes. This could be due to how the loss function was implemented. In addition, perhaps the value for $\boldsymbol{\lambda}$ was too strong as instead of setting a constant value for it, it was given an equal beta distribution, which is a gaussian distribution. The idea was

to vary the size of the cut patch automatically, where sometimes it can have a max value of 0.5, and sometimes close to non at all. Further testing was not conducted to see the effect of this, while it is suspected that this is the case as historically, research has shown CutMix to outperform MixUp.

No Modification:

Overall Accuracy : 84.22%

	precision	recall	f1-score	support
airplane	0.8149	0.9290	0.8682	1000
automobile	0.8494	0.8910	0.8697	1000
bird	0.8053	0.9100	0.8545	1000
cat	0.8054	0.7700	0.7873	1000
deer	0.8282	0.9060	0.8653	1000
dog	0.7542	0.9050	0.8227	1000
frog	0.9564	0.7680	0.8519	1000
horse	0.9727	0.7480	0.8457	1000
ship	0.7972	0.8690	0.8316	1000
truck	0.9441	0.7260	0.8208	1000
accuracy			0.8422	10000
macro avg	0.8528	0.8422	0.8418	10000
weighted avg	0.8528	0.8422	0.8418	10000

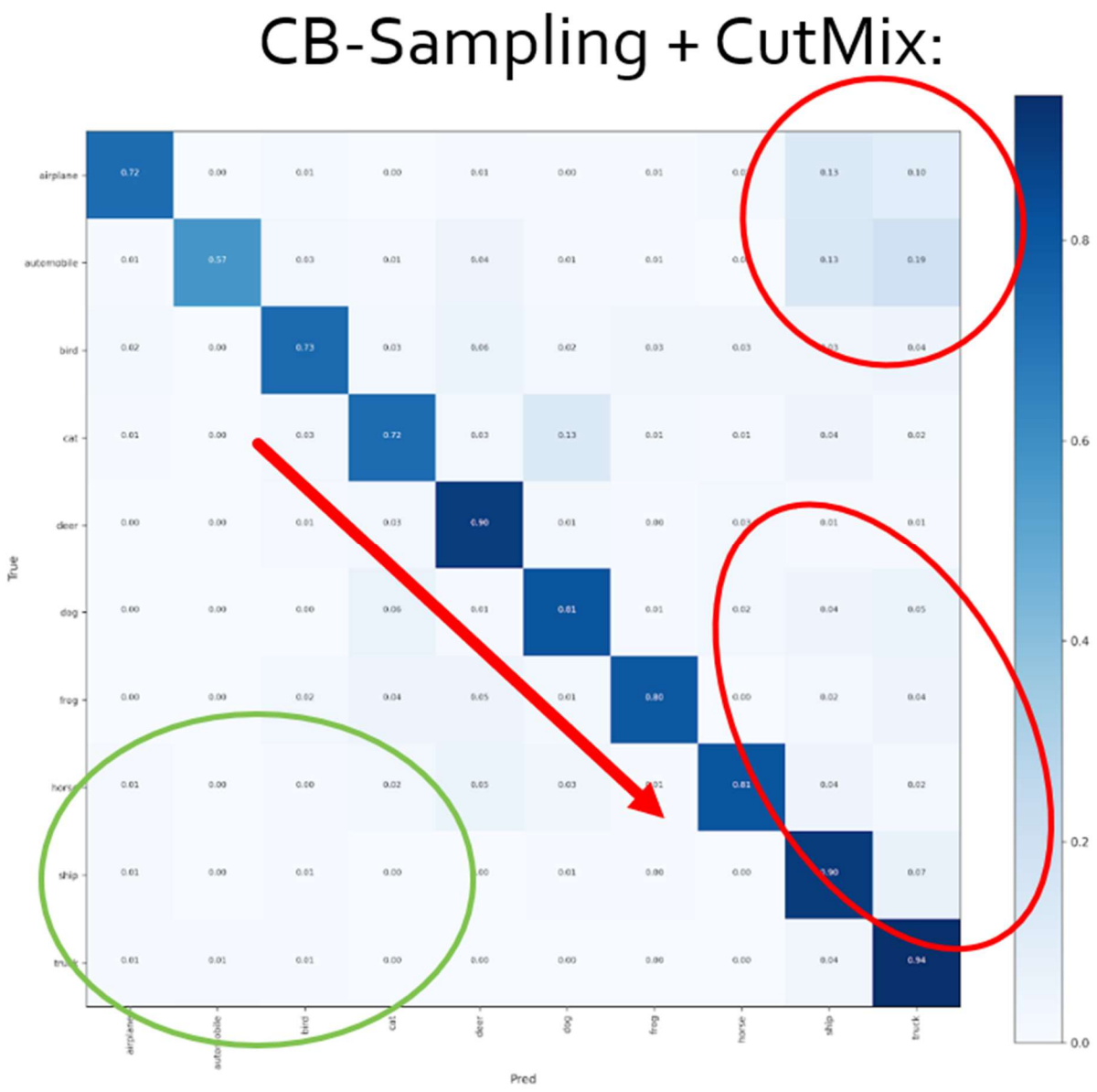
CB-Sampling + CutMix:

Overall Accuracy : 79.06%

	precision	recall	f1-score	support
airplane	0.9149	0.7200	0.8058	1000
automobile	0.9662	0.5710	0.7178	1000
bird	0.8683	0.7320	0.7944	1000
cat	0.7919	0.7230	0.7559	1000
deer	0.7853	0.8960	0.8370	1000
dog	0.7744	0.8100	0.7918	1000
frog	0.9069	0.7990	0.8495	1000
horse	0.8745	0.8080	0.8399	1000
ship	0.6520	0.9030	0.7572	1000
truck	0.6340	0.9440	0.7585	1000
accuracy			0.7906	10000
macro avg	0.8168	0.7906	0.7908	10000
weighted avg	0.8168	0.7906	0.7908	10000

[Figure 17: Metrics of combining CutMix with CB-sampling]

Looking at the confusion matrix below, there has been a significant decrease of misclassification of the tail class in the bottom left corner, however the diagonals of the true classes have shifted to be more weighted towards the tail classes. There is also an increase of misclassifications for “airplane” and “automobiles”, where the model confuses them for a “truck” or “ship”. This could be due to the fact that these four classes have very similar features, where applying CutMix on them with a large, patched region will greatly confuse the model and not extract clear hard boundaries between them.



[Figure 18: Confusion Matrix of CB-sampling + CutMix]

Advanced Loss Function with Label Smoothing:

Currently the model is computing the loss function with hard labels calculated with cross-entropy loss. This would mean that the label for the class “cat” would have a hard label like [0,0,0,1,0,0,0,0,0,0]. By learning from this loss, the model tends to be overconfident in its predictions and memories class-specific patterns. This can be adjusted with label smoothing.

[5] Label Smoothing works by softening the one-hot target labels by setting the distribution with the value **ε**. The new target distribution is calculated by replacing the true target class as $1 - \varepsilon$ if the sample belongs to the true class, otherwise sets the rest to $\frac{\varepsilon}{K-1}$, where K is the total samples in that class. For the "cat" example above, if $\varepsilon = 0.1$, then the label would turn into [0.011, 0.011, 0.011,0.9, 0.011, 0.011, 0.011, 0.011, 0.011, 0.011]. This essentially gives the model leeway to say, "this sample may look like this or that class, but it greatly looks like this class". For testing, having ε set to 0.2 produced the best results.

No Modification:

Overall Accuracy : 84.22%

	precision	recall	f1-score	support
airplane	0.8149	0.9290	0.8682	1000
automobile	0.8494	0.8910	0.8697	1000
bird	0.8053	0.9100	0.8545	1000
cat	0.8054	0.7700	0.7873	1000
deer	0.8282	0.9060	0.8653	1000
dog	0.7542	0.9050	0.8227	1000
frog	0.9564	0.7680	0.8519	1000
horse	0.9727	0.7480	0.8457	1000
ship	0.7972	0.8690	0.8316	1000
truck	0.9441	0.7260	0.8208	1000
accuracy			0.8422	10000
macro avg	0.8528	0.8422	0.8418	10000
weighted avg	0.8528	0.8422	0.8418	10000

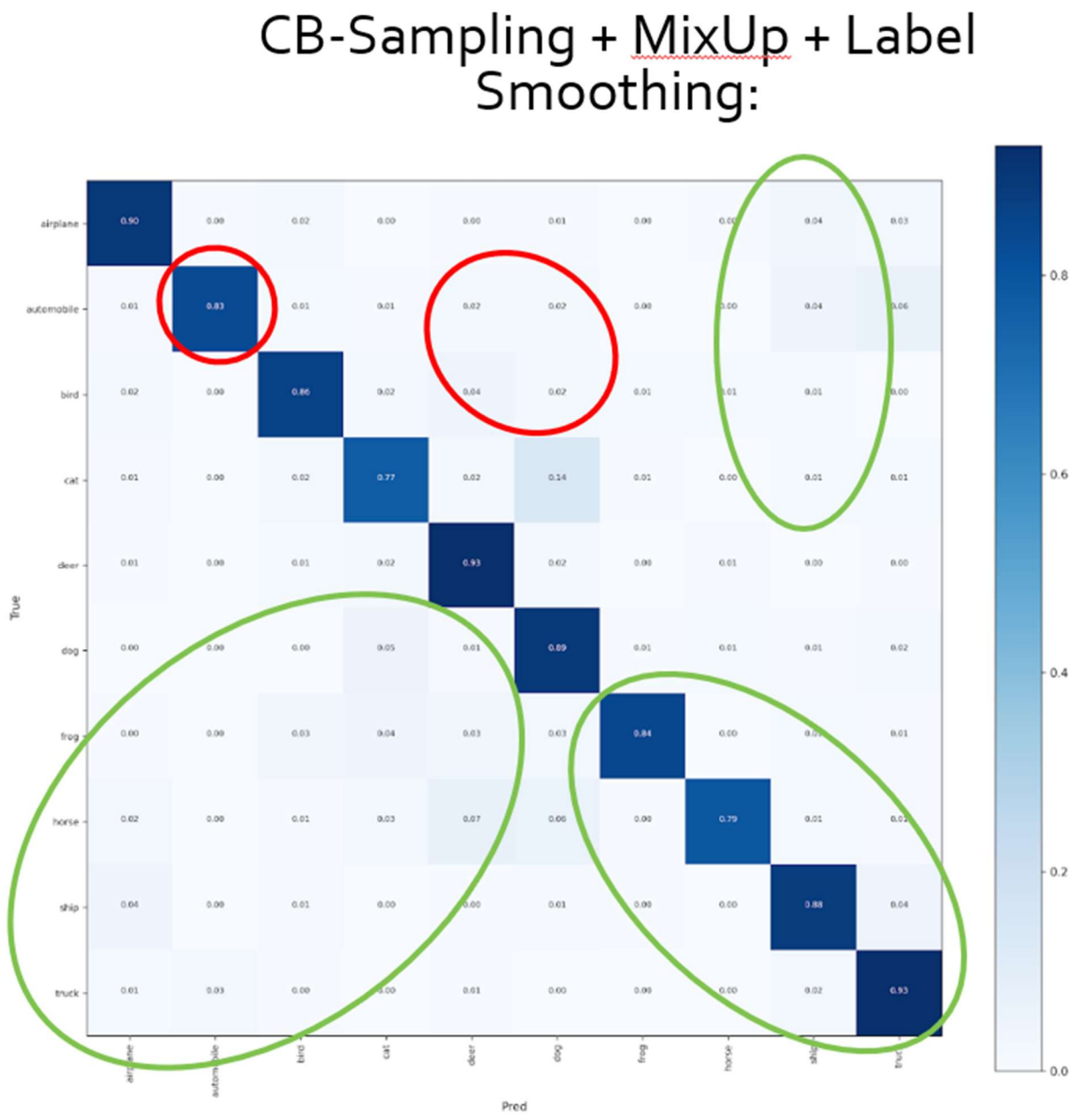
CB-Sampling + MixUp + Label Smoothing:

Overall Accuracy : 86.23%

	precision	recall	f1-score	support
airplane	0.8819	0.8960	0.8889	1000
automobile	0.9541	0.8310	0.8883	1000
bird	0.8787	0.8620	0.8703	1000
cat	0.8223	0.7730	0.7969	1000
deer	0.8245	0.9300	0.8741	1000
dog	0.7464	0.8920	0.8128	1000
frog	0.9569	0.8430	0.8963	1000
horse	0.9460	0.7890	0.8604	1000
ship	0.8489	0.8820	0.8651	1000
truck	0.8296	0.9250	0.8747	1000
accuracy			0.8623	10000
macro avg	0.8689	0.8623	0.8628	10000
weighted avg	0.8689	0.8623	0.8628	10000

[Figure 19: Metrics of combining CB-sampling + MixUp + Label Smoothing]

Based on the results in the metrics above, the overall accuracy increased by 2.01% with the model having an accuracy of 86.23%. Most of the tail classes have shown significant improvements. The class of "dog" is still having a hard time deciding the decision boundary for it. With the confusion matrix below, the model is still misclassifying the "dog" class as a "cat". Though, there is a noticeable decrease of the misclassification of the head class "airplane" and "automobile" to the tail class "ship" and "truck".



[Figure 20: Confusion Matrix of CB-sampling + MixUp + Label Smoothing]

Data Augmenting with TTA:

Currently, once the model has been trained, it will run the entire testing dataset of 10,000 images through it to generate the results. The problem is that the model only looks at each image once to decide the result. Some samples may be difficult to identify from one perspective, and

the model could change its decision if it looked at it another way. This is where Test Time Augmentations (TTA) can help.

[6] TTA works by creating augmented versions of the test sample by “flipping”, “rotating”, or “scaling” the image in addition to the default image to pass through the model. In the end, the model will perform a vote to decide what the image is most likely. For this augmentation, “flipping” the image will suffice and flipping it only in the horizontal direction seemed to perform the best. “Flipping” it vertically or both vertically and horizontally showed diminishing returns.

Looking at the performance metrics below, the model now has an accuracy of 86.55%, which is an increase of 2.33% over the baseline model. Most of the tail classes have significantly improved, and the distribution of the recall over all the classes seems to be the same being around 0.8500. This is the optimal distribution to look for when addressing the class imbalance problem.

No Modification:

Overall Accuracy : 84.22%

	precision	recall	f1-score	support
airplane	0.8149	0.9290	0.8682	1000
automobile	0.8494	0.8910	0.8697	1000
bird	0.8053	0.9100	0.8545	1000
cat	0.8054	0.7700	0.7873	1000
deer	0.8282	0.9060	0.8653	1000
dog	0.7542	0.9050	0.8227	1000
frog	0.9564	0.7680	0.8519	1000
horse	0.9727	0.7480	0.8457	1000
ship	0.7972	0.8690	0.8316	1000
truck	0.9441	0.7260	0.8208	1000
accuracy			0.8422	10000
macro avg	0.8528	0.8422	0.8418	10000
weighted avg	0.8528	0.8422	0.8418	10000

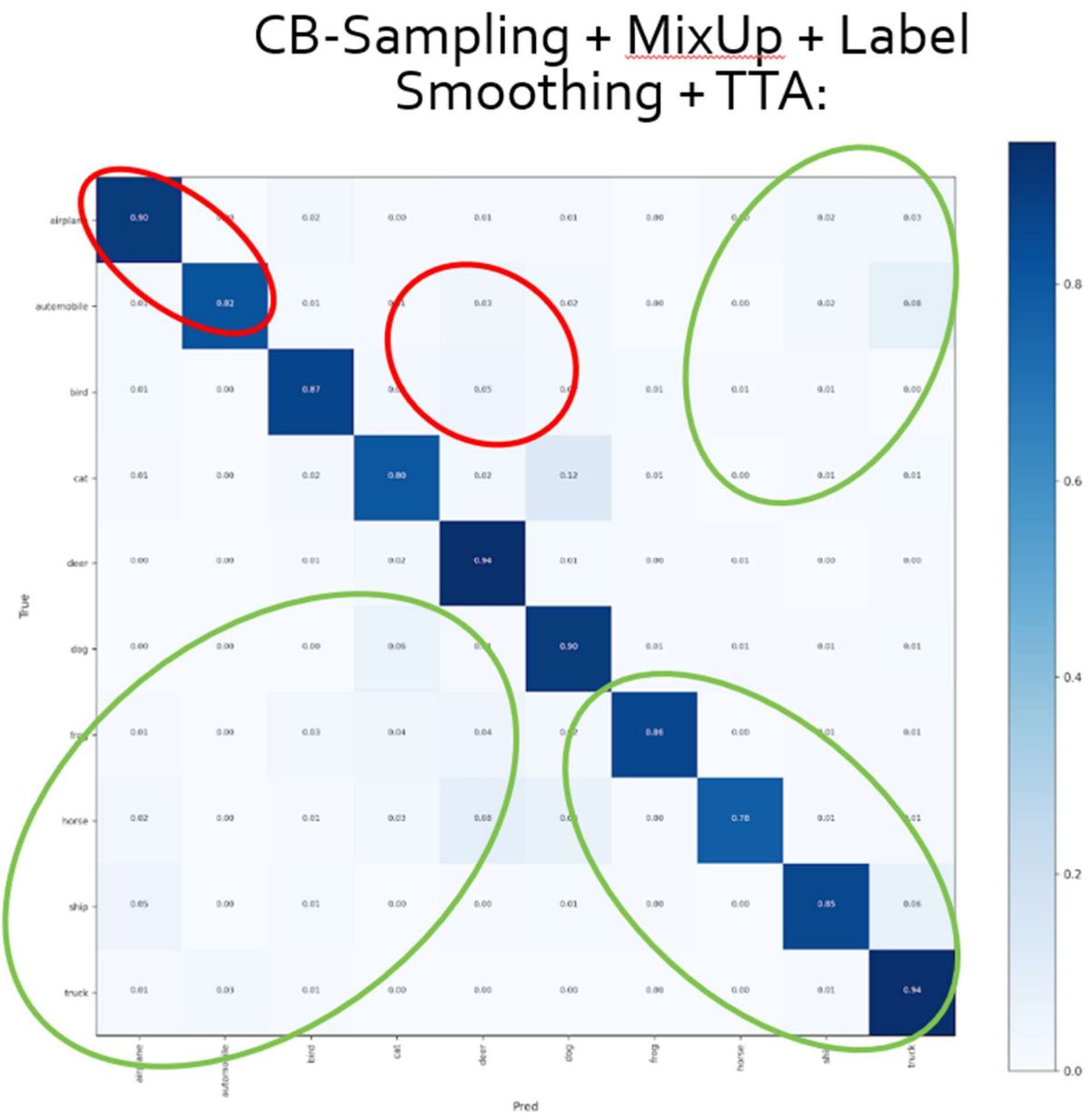
CB-Sampling + MixUp + Label Smoothing + TTA:

Overall Accuracy : 86.55%

	precision	recall	f1-score	support
airplane	0.8805	0.8990	0.8897	1000
automobile	0.9578	0.8180	0.8824	1000
bird	0.8830	0.8680	0.8754	1000
cat	0.8207	0.8010	0.8107	1000
deer	0.7946	0.9440	0.8629	1000
dog	0.7604	0.8950	0.8222	1000
frog	0.9555	0.8590	0.9047	1000
horse	0.9629	0.7780	0.8606	1000
ship	0.9174	0.8550	0.8851	1000
truck	0.8072	0.9380	0.8677	1000
accuracy			0.8655	10000
macro avg	0.8740	0.8655	0.8661	10000
weighted avg	0.8740	0.8655	0.8661	10000

[Figure 21: Metrics of combining CB-Sampling + MixUp + Label Smoothing + TTA]

Looking at the confusion matrix below, the diagonal true class for all classes are now more equally distributed, where they appear to be closely equal. Compared to the baseline confusion matrix, there is significantly less misclassification of the tail class being confused for a head class in the bottom left corner. There is also an improvement in the head class being confused as a tail class in the top right corner too. There is a lower performance of the true head class at the top left compared to the baseline model, but this is expected as explained in the CB-Sampling above. The overall benefits outweigh this decrease in performance.



[Figure 22: Confusion Matrix of CB-Sampling + MixUp + Label Smoothing + TTA]

Decoupling the model:

Currently, the model is set up to have a single stage one shot training regime, where both the feature extractor and classifier are trained together. This would mean that during the training, when computing the loss function, the feature extractor will produce stronger gradients for head classes, and the classifier would have to learn from these strong gradients that are constantly

changing, making it more skewed towards the head classes. A solution to this issue would be to use decoupling techniques.

[7] Decoupling works by training the model normally for a few epochs, so that the feature extractor can learn rich features. Then the model is "frozen" except for the classifier head so that the classifier can learn from stable gradients in the remaining stages of training.

For this study, the model has been trained like normal for the first 20 epochs. The only change is that CB-sampling is disabled for this "stage 1" part, so that the model can learn the "true data distribution" and extract strong features. Using CB-Sampling in this part can over-emphasize the tail classes, making the model have reduced feature diversity.

For the remaining 10 epochs, the backbone is "frozen" by disabling the gradient updates, and CB-sampling is re-enabled. The LR has also been reset to the default 1e-4 so that the classifier has a fair chance to learn. MixUp has also been disabled in this "stage 2" part so that the classifier does not get confused by the distorted class labels. It is also pointless to have MixUp at this stage as it is only used to help the model learn unique features from the blended images.

Looking at the results below, there is an increase of accuracy of 0.79% having the overall accuracy of the model be 85.01%, however this is still lacking compared to the results obtained without the decoupling technique. There appears to be random increase and decrease for class performance overall, identifying that some of the pervious techniques may not be compatible with this decoupling technique as it is confusing the model.

No Modification:

Overall Accuracy : 84.22%

	precision	recall	f1-score	support
airplane	0.8149	0.9290	0.8682	1000
automobile	0.8494	0.8910	0.8697	1000
bird	0.8053	0.9100	0.8545	1000
cat	0.8054	0.7700	0.7873	1000
deer	0.8282	0.9060	0.8653	1000
dog	0.7542	0.9050	0.8227	1000
frog	0.9564	0.7680	0.8519	1000
horse	0.9727	0.7480	0.8457	1000
ship	0.7972	0.8690	0.8316	1000
truck	0.9441	0.7260	0.8208	1000
accuracy			0.8422	10000
macro avg	0.8528	0.8422	0.8418	10000
weighted avg	0.8528	0.8422	0.8418	10000

CB-Sampling + MixUp + Label Smoothing + TTA + Decoupling:

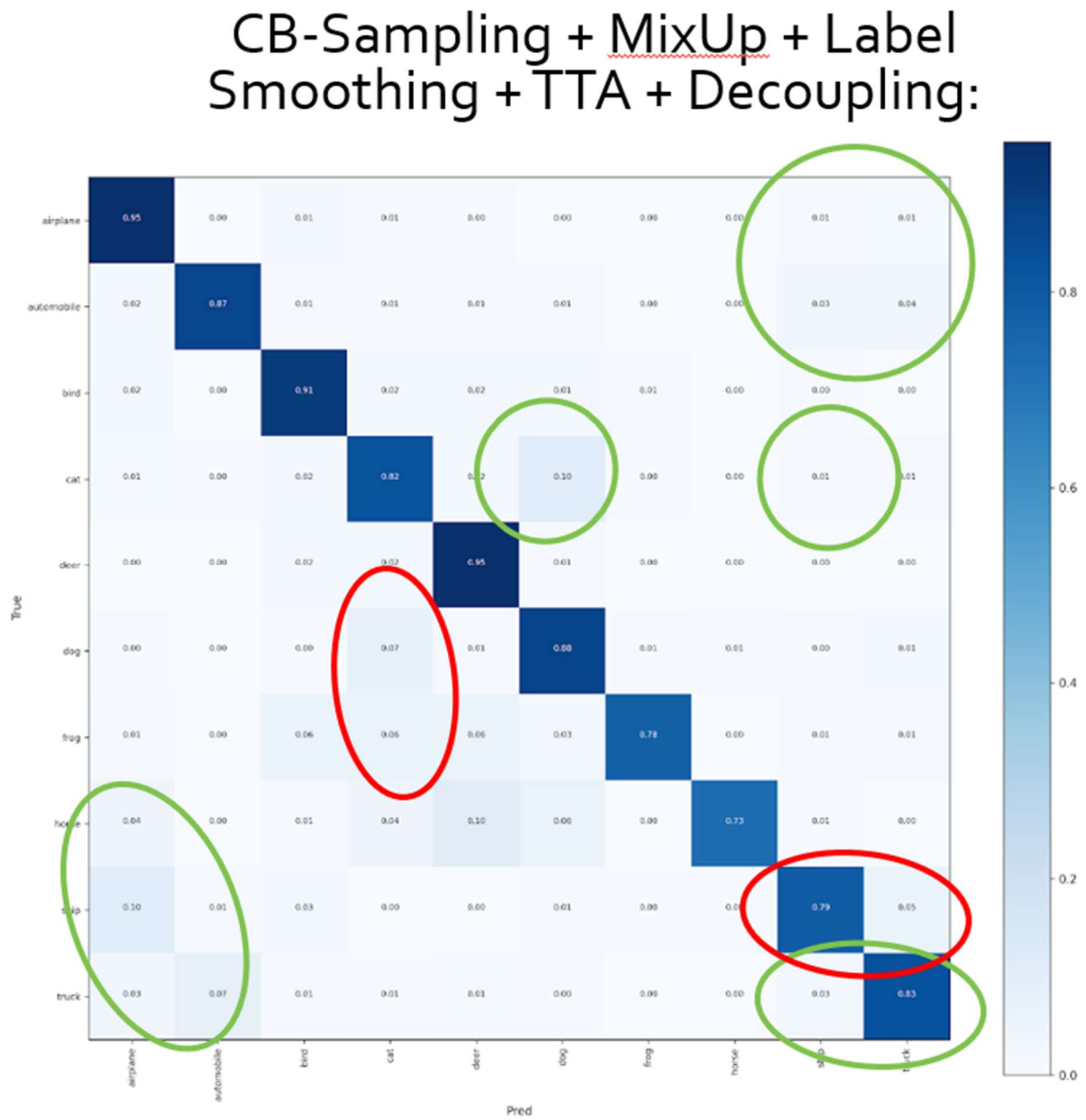
Overall Accuracy : 85.01%

	precision	recall	f1-score	support
airplane	0.7948	0.9530	0.8668	1000
automobile	0.9065	0.8730	0.8895	1000
bird	0.8469	0.9070	0.8759	1000
cat	0.7801	0.8230	0.8010	1000
deer	0.7983	0.9460	0.8659	1000
dog	0.7853	0.8780	0.8291	1000
frog	0.9615	0.7750	0.8583	1000
horse	0.9668	0.7270	0.8299	1000
ship	0.8914	0.7880	0.8365	1000
truck	0.8594	0.8310	0.8449	1000
accuracy			0.8501	10000
macro avg	0.8591	0.8501	0.8498	10000
weighted avg	0.8591	0.8501	0.8498	10000

[Figure 23: Metrics of combining CB-Sampling + MixUp + Label Smoothing + TTA + Decoupling]

For the confusion matrix below, there are places all over the matrix where the model has improved or decreased compared with the baseline. It seems that decoupling overall has increased the generalizability of the model with no specific emphasis on a particular class, while

it still has performance issues for some niche classes. Perhaps the feature extractor was not given enough time to develop rich features for only training it in 20 epochs so that the classifier can learn it better, or that by increasing the LR for the stage 2, the classifier took a too big of a “step” and ended up in a shallow hole. Another possibility would be that some methods are not compatible, such as using MixUp with $\lambda = 0.4$ is too strong of a regularizer for the model to extract rich features from, or by having TTA in the end confuses the model. Further testing was not conducted due to time constraints, though it is suspected that one of the mentioned above are the reasons, as historically, decoupling techniques have been proven to significantly increase the performance of a model.



[Figure 24: Confusion Matrix of CB-Sampling + MixUp + Label Smoothing + TTA + Decoupling]

Conclusion:

Overall, the best model performance achieved was 86.55%. This was done by using the techniques of CB-Sampling, MixUp with $\lambda = 0.4$, Label Smoothing with $\varepsilon = 0.2$, and TTA where the image is only flipped horizontally once. This led to an increased performance of 2.33% over the baseline model, and the distribution of the true classes are more equally distributed, with an average of around 0.85.

Academic papers have suggested that techniques such as CutMix and Decoupling should increase performance further, however this was not achieved in the time frame provided for this paper. The causes are suspected to be either the way the functions were coded, or how some techniques are not compatible with others. Further research has not been conducted; however, it would have been nice to conclude the study with those techniques working.

References:

[1] M. Tan and Q. V. Le, “EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks,” *arXiv preprint* arXiv:1905.11946, 2019.

[2] Y. Cui, M. Jia, T.-Y. Lin, Y. Song, and S. Belongie, “Class-balanced loss based on effective number of samples,” *arXiv preprint* arXiv:1901.05555, Jan. 2019.

[3] H. Zhang, M. Cissé, Y. N. Dauphin, and D. Lopez-Paz, “mixup: Beyond empirical risk minimization,” *arXiv preprint* arXiv:1710.09412, 2018.

[4] S. Yun, D. Han, S. J. Oh, S. Chun, J. Choe, and Y. Yoo, “CutMix: Regularization strategy to train strong classifiers with localizable features,” in *Proc. IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019, pp. 6023–6032.

[5] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, “Rethinking the Inception architecture for computer vision,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2016, pp. 2818–2826.

[6] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 25, pp. 1097–1105, 2012.

[7] B. Kang, S. Xie, M. Rohrbach, Z. Yan, A. Gordo, J. Feng, and Y. Kalantidis, “Decoupling representation and classifier for long-tailed recognition,” in *Proc. Int. Conf. on Learning Representations (ICLR)*, 2020.