**4 Ways to Improve Class Imbalance for Image Data**

The pros and cons of several rebalancing techniques

Chart, bar chart

Description automatically generated

Instances per category of the highest- and lowest-represented categories. 60% of the classes have < 1000 instances compared to the 300,000 instances of the highest-represented class.

If you were to predict that an object in the dataset was a “Building,” you’d be right 50% of the time. This class imbalance therefore leads you to believe your model is better than it really is.

**TECHNIQUES**

**Adjustment #1: Chipping instead of downsampling**

In a nutshell, the raw images are too large to fit into the neural network’s input layer. A 12 megapixel drone image is 4000 x 3000 pixels. A common image size to feed into an object detector is 512 x 512 pixels or smaller.

While it’s possible to downsample the raw images, you’ll lose important information and the model will have lower precision for classes of small objects.

This example dataset’s minority classes are all much smaller than the #1 majority class of “Buildings,” so downsampling disproportionately impacts minority classes.

**Chipping raw images into smaller tiles is an alternative to downsampling.** By preserving full resolution, we can ensure that smaller-item minority classes don’t become even harder to identify.

A picture containing surrounded

Description automatically generated

Example of a full image chipped into 49 tiles.

Each image is tiled into smaller square cells, which can be fed into the network individually while retaining the original image resolution. To ensure that no information is lost for objects split across tile boundaries, you can chip the image with overlap between consecutive tiles (e.g. 25% overlap).

***Note****: some datasets may contain objects with high size disparity. You may want to retain the full original resolution to be able to identify tiny objects but also have objects so large they don’t fit in a single tile. In this case, it may be necessary to build your training dataset with some full-resolution tiles and some downsampled full-picture images.*

Our model’s [mean average precision (mAP)](http://cs230.stanford.edu/section/7/#object-detection-iou-ap-and-map) was higher when trained against the tiled dataset versus a version of the dataset where the image resolution was rescaled.

**Downside:** while the model can perform better with small-object classes, the tiling process does lead to longer training times. For example, this chipping method increased our mAP by 30% but also increased our training run time by 60%.

→ Worth it? Highly likely.

**Adjustment #2: Merging near-identical classes**

Some of the classes are almost identical, e.g. “Fixed wing aircraft” and “Cargo plane.” Both of those classes have an average bounding box diameter of ~18 pixels. This level of class granularity on low-pixel items may be difficult for the model to learn — especially if there aren’t many data points from each class.

We decided to merge similar looking objects into single merged classes, which reduced the number of classes and reduced slightly the class distribution imbalances.

***Note:*** *It’s best to involve somebody with domain expertise when merging labels. Not only can they recommend which classes make sense to group together, but they can also describe places in the end user workflow where original classes must absolutely be preserved. For example, it may be tempting to re-classify “Tank” as a “Truck,” but the user workflow might take highly specific actions for the “Tank” class. In addition, to help streamline the merging process, you could create a merge candidate list for human review by identifying groups of classes that create the most confusion for the model.*

After merging, we had 16 new classes vs. 50 original classes — but how much better is the class imbalance? Let’s look at the two distributions.

Chart, box and whisker chart

Description automatically generated

After merging minority classes together, the higher median count per class means we have fewer hard-to-learn classes. (image by author)

Before merging, 75% of the classes contained fewer than 1,500 instances. After under-represented classes were merged, the median count per class was over 1,500. (Outliers not shown on graph, including the 426,000-count “Buildings” class).

→ Worth it? Likely, but viability depends on the domain and end user workflow.

**Adjustment #3: Resampling specific classes**

A traditional way to combat large class imbalances in machine learning is to adjust class representation in the training set.

**Oversampling infrequent classes** is augmenting entries from the minority classes to match the quantity of the majority classes. This may be performed several ways, such as by generating synthetic data or by essentially copying entries from the minority class (e.g. via [sklearn’s `resample`](https://scikit-learn.org/stable/modules/generated/sklearn.utils.resample.html" \t "_blank) or [TensorFlow’s tf.data sampler](https://www.tensorflow.org/tutorials/structured_data/imbalanced_data" \l "using_tfdata" \t "_blank) or [PyTorch’s WeightedRandomSampler](https://pytorch.org/docs/stable/data.html" \l "torch.utils.data.WeightedRandomSampler" \t "_blank)). The downside is that it can result in overfitting of the oversampled classes.

Undersampling frequent classes is removing entries from the majority classes so they match the quantity within the minority classes. The downside is that, by removing data points, you may remove valuable information or lead to poor generalization for real-world data. Or the imbalance may be so bad that the result of undersampling would be too small of a dataset.

Chart, bar chart

Description automatically generated

Oversampling vs. Undersampling

***Note:*** *both of these changes should occur*after*splitting data into train and validation sets to ensure that no data in the validation set is included in the training set.*

For this example dataset, we implemented undersampling by selectively including tiles in the final training dataset. Labels contained in each image tile were counted and stored in a dataframe. We then created a sorting procedure to either include or discard each tile. All images containing at least one instance of a minority label (anything besides “Building” and “Small Cars”) were included in the training dataset. Then, to round out the dataset, we also included 10% of the tiles containing only the most frequent classes. No tiles with zero labels were included.

Chart, bar chart

Description automatically generated

Change in object count by class.

**Bonus: tile curation also makes training more efficient.**

Besides increasing average precision for minority classes, we’ve also decreased training time (by decreasing the dataset size) while only removing mostly redundant information.

Another boon to curating which tiles are included in training is that you can weed tiles that contain zero bounding boxes. With aerial photography, the original full-scale image can sometimes contain only 5–10 annotations, which leads to a significant number of “empty” tiles. A model trained on too many empty tiles may learn to predict no bounding boxes as the optimal solution.

Across use cases and types of content, many large scale images contain dead space that can be removed before training. For example, look how many tiles of a digitized pathology slide are 100% white:

Chart

Description automatically generated with medium confidence

Example digital pathology slide from [PAIGE.ai](https://paige.ai/).

So, when you have large raw images guaranteed to include white space, it’s often better to start thinking of the tiled dataset as the new training data. You can start exploring and cleaning it out with more granular control.

→ Worth it? Yes.

**Adjustment #4: Adjusting the loss function**

Beyond just having 100x more instances of the “Building” class than some minority classes, the empty background areas of the images are actually another dominating majority class. The model will see a high number of easily-classified negative areas — sometimes 1:1000 foreground to background areas. And when over-represented classes are relatively easily classified, they can dominate the overall loss, which steers the gradient descent to optimize for detection of those majority classes.

Instead of treating each mistake equally, treat mistakes on rare classes as more important than mistakes on common classes.

In response, we implemented focal loss à la FAIR’s [Focal Loss for Dense Object Detection](https://arxiv.org/pdf/1708.02002.pdf) paper. The loss function is dynamic based on the predicted probability of each object. The tunable “focusing parameter” `γ` between [0, 5] drives the loss for well-classified examples (p > 0.5) towards zero. This change decreases the dominance of over-represented classes in the total loss term.

Diagram

Description automatically generated

[Source](https://arxiv.org/pdf/1708.02002.pdf): Facebook AI Research (FAIR)

For the merged-class dataset, implementing focal loss increased the average precision on minority classes and conserved relatively good average precision on majority classes.

→ Worth it? Yes.

**Conclusion**

It’s not uncommon for datasets to start off with significant class imbalance, which can be compounded further for image datasets that may include a large volume of the “empty” (or background) class.

When under-represented classes aren’t accounted for, models can reach an accuracy ceiling where majority classes are predicted easily, but overall model accuracy may not improve until steps are taken to account for class imbalance and improve performance of minority classes.