# EARIN Final Report

**Aayush Gupta**

## Description of Project

In this project, a heavily imbalanced data set consisting of aerial images of ten different vehicles was used to train a convolutional neural network to identify vehicles from images it hasn't seen. Additionally, a model was proposed that could potentially be used to predict the accuracy of predictions on an unlabeled but balanced validation set.

# Dataset

## Overview

The data set consists of ten sets of black and white 32 by 32 pixel images of different vehicles. There is a significant difference in the number of images between the different sets. Half of the vehicles have fewer than 1000 images, whereas Sedans have 234,209 images. Furthermore, most of the vehicles that have fewer images have rather insignificant variations in their images. In fact, in many cases, there are only three or four different vehicles pictured in the entire set with only slight variations in the images. The fact that that many of the smaller classes have unique images numbering in the single digits resulted in significant overfitting of those classes by the neural network.

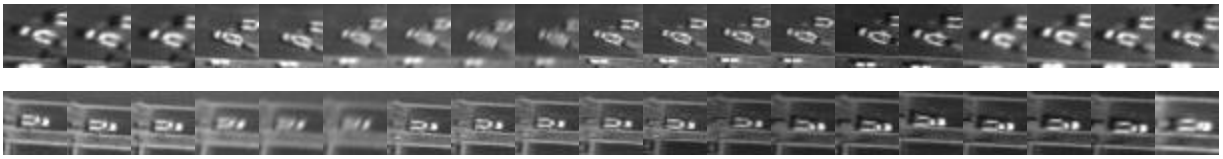

Figure 1: Images of flatbed trucks with trailers

## Preprocessing

The validation set was created by taking 30 percent of images (chosen randomly) from each class and placing them in their own validation set. The remaining 70 percent of images became the training set.

In order to reduce the impact of overfitting and encourage the neural network to generalize instead of memorizing, every image in the training set was augmented with some random combination of brightness change, x and/or y translation, rotation, image flip, and rescale. The parameters for these augmentations were chosen carefully to not risk removing the vehicle from the image.

Since many images in the validation set were very similar to each other, a second validation set was created using augmented versions of images from the original validation set. The augmentations applied to this validation set were the same as the ones applied to the training set. The augmented validation set provides a more realistic estimate of the performance of the neural network.

## Post-processing

The output layer of the neural network consists of ten values, one for each class. The neural network chooses the highest of these values when making predictions, with the corresponding class becoming the prediction for that input.

# Technical Approach

## Architecture

### Neural Network

A convolution neural network was implemented to classify the vehicles in the images. The entire neural network consists of two convolution layers of kernel size five followed by three linear hidden layers. Each convolution layer has an activation function of ReLU and is followed by a max pooling layer. Each linear hidden layer except the last (to avoid issues with the loss function) also has a ReLU activation function. After the second max pooling layer is applied, the initial input of 32 by 32 is reduced down to 5 by 5. The third linear hidden layer is followed by the output layer which has a width of ten. For most of the tests, convolution layer widths of 256 and linear hidden layer widths of 2048 were used.

### Probability Density Simulation

As the original problem called for a quantitative method to analyze the accuracy of the neural network's predictions on a balanced unlabeled set, the probability density simulation was implemented. This feature estimates the accuracy of a set of predictions in a balanced unlabeled set by analyzing the distribution of the predictions. More specifically, it finds the difference between the number of predictions there should be for each class and the number of predictions there actually are for those classes. Then, using the results of a simulation that was run beforehand, it estimates the accuracy of that distribution of predictions.

The simulation is run using at least a few thousand samples, with each sample containing a set of predictions. It starts by assuming that each sample contains only correct predictions. Then it randomly chooses an index for each sample—one prediction belonging to that class will become incorrect. A second index is randomly chosen (it cannot be the same as the first), and the new (now incorrect) prediction becomes the class that is represented by that

index. Each sample continues to make predictions until there are no more correct predictions left. By considering each iteration of every sample as its own state, the accuracy for every distribution of predictions can be obtained.

## Training Details

### Hardware/Software

Deep learning problem calculations can often be performed much more efficiently with a GPU than with a CPU. As such, an RTX 2070 max-p graphics card with 8 GB of dedicated VRAM was utilized when training the neural network. This made calculations nearly three times faster when compared to using the CPU.

The neural network and all tests were implemented in Python. The Pytorch library was used to create the neural network, create and apply transforms to the training and validation data sets, train the neural network on the training set, and make predictions with the neural network on the validation set.

### Optimizer

The optimizer used was the mini batch gradient descent method with a momentum parameter, which was set to 0.9 for all tests. The mini batch variant of the gradient descent method allows for the weights and biases of the neural network to be updated several times per epoch (as opposed to just once). This makes the optimization significantly faster.

The momentum parameter enables the gradient descent method to be influenced by previous gradients (and not just the current gradient). By having a momentum, the gradient descent method should be less prone to being slowed down by large oscillations and sudden flat regions (which have a small gradient).

### Batch Size

To take advantage of the GPUs efficiency with larger batch sizes, a batch size of 256 was selected. This number is high enough to take advantage of the increased GPU efficiency, but not so high that there is a significant risk of overfitting.

### Learning Rate

The learning rate is essentially the step size of the gradient descent method. Early on, it was found that, while a higher learning rate performed well in the first few epochs, its performance rapidly plateaued. Meanwhile, a lower learning rate could get much better results before plateauing at the cost of a significantly longer computation time. As such, an exponentially decaying learning rate was used to take advantage of the benefits of both higher and lower learning rates. The following exponential decay formula was used for the learning rate:

$$\text{current LR} = \text{starting LR} \cdot \text{decay rate}^{\text{epoch}}$$

### Loss Function

The loss function utilized was Cross Entropy Loss, which measures the dissimilarity between the predicted probability distribution and the true distribution. The loss function is zero when the predicted probability is identical to the true distribution, and it goes to infinity if the predicted probability diverges from the true label.

### Dataset Expansion

Since many of the images in the dataset were very similar to one another (particularly those belonging to the smaller classes), an option for expanding the smaller subsets was added. Dataset expansion creates augmented duplicate images for each class until the total number of

images (original + duplicates) in each class is equal to 'dataset expansion multiplier' * number of images in largest class. If the number of images belonging to a class exceeds this product, no augmented duplicates are created for that class.

In theory, setting the multiplier to 1 should solve the class imbalance problem. However, due to performance issues when higher values were used, the dataset expansion multiplier was set to 0.1 for most tests.

## Weighted Random Sampling

The class imbalance problem was solved by using a weighted random sampler, which uses weighted probabilities to randomly select samples when training in a way that the number of samples chosen from each class is, more or less, equal. This results in each class contributing equally to the loss. However, up-sampling small data sets too much can cause the neural network to overfit those classes. One way to alleviate this is to down-sample classes that are being overfit and up-sample classes that are being underfit.

## Dynamic Sampling

Dynamic sampling was implemented to resolve the overfitting and underfitting issues of the weighted random sampler and hasten the training process. After each epoch, the f1 score of each class is used to recalculate the weights that are used by the weighted random sampler. A quadratic relationship was found to generally perform better than a linear relationship. As such, new weights are calculated by multiplying the original weights by the square of the f1 score. This means that the weighted random sampler will no longer sample the classes equally, but instead will sample them in a way that is most optimal for learning without overfitting.

## Dynamic Dampening

Using the square of the f1 score when calculating new weights often causes the dynamic sampler to overshoot the ideal sampling number of one or more classes. This results in oscillations of both the sample sizes of each class and the total accuracy after each epoch. In extreme cases, it can even cause the network to diverge.

Adding a dampening factor, where a weighted average is taken with the old weights and the newly calculated weights resolves this issue if the selected dampening parameter is high enough. If it is too high though, it will significantly slow down the neural network, since it will take many epochs for the dynamic sampler to reach the ideal sampling number for each class. Dynamic dampening allows dampening to be applied and adjusted as needed with little to no user input.

Dynamic dampening has the user choose min dampening, max dampening, forward step size, reverse step size, and step threshold. If, after resampling, the number of samples of the most sampled class is reduced below 'step threshold', 'current step' increments by 'forward step size' and 'current dampening' increases according to the following function:

$$\text{current dampening} = \frac{\text{min dampening} - \text{max dampening}}{e^{\text{current step}_2}} + \text{max dampening}$$

If the 'step threshold' condition is not met after resampling, 'current step' decrements by 'reverse step size'. If doing so reduces 'current step' below zero, 'current step' becomes zero.

## Probability Density Simulation Parameters

As the distribution of errors of most neural networks is seldom uniform, uniform random number generation was not used when choosing classes to move from the correct predictions to the incorrect predictions. Instead, a weighted random number generator was used.

For choosing classes to move out of the correct predictions (which can be thought of as the recall), the weights used were obtained with the following formula:

$$\text{recall weights} = \text{rng}(\text{base min}, \text{base max}) \text{ ^ } \text{rng}(\text{power min}, \text{power max}) + \text{rng}(\text{offset min}, \text{offset max})$$

For choosing classes to move the now incorrect predictions into (which can be thought of as the precision), the weights used were obtained with the following formula:

$$\text{precision weights} = (1 - \text{correlation}) * (\text{rng}(\text{base min}, \text{base max}) \text{ ^ } \text{rng}(\text{power min}, \text{power max}) + \text{rng}(\text{offset min}, \text{offset max})) + (\text{correlation} * \text{recall weights})$$

Using a constant base of 2 resulted in the most reliable accuracy estimates. Power, on the other hand, ranged from 0 to 19 (inclusive). Meanwhile, offset was varied from 500 to 50000. Finally, correlation values between 0.1 and 0.2 were tested.

The resulting weights are almost entirely influenced by the offset when the power is low or moderately high. When the power is high, though, the exponent 2^power completely dominates the offset. The correlation parameter determines the strength of the correlation between the precision and the recall. Mathematically, this is represented as a weighted average, where the recall weights contribute some amount to the precision weights.

The final result is a distribution of weights that is mostly uniformly random with one or a few significant outliers.

## Evaluation Details

### Labelled Unbalanced Validation Sets

Since the validation set is unbalanced, both precision and recall are important to keep track of. With a significantly unbalanced validation set, the biggest risk to the model's integrity is having the dominating class's low (or even middling) recall tank the other class's precision. As

such, the macro average f1 score, which combines precision and recall and weighs each class equally, was used to evaluate the model's effectiveness in the comprehensive tests. To reduce the impact of outliers, the f1 scores from the last 15% of epochs were averaged.

The f1 score and several other useful statistical measurements were included in the classification report, which was printed after every epoch. Additionally, a confusion matrix was printed after every epoch to track predictions made by each class.

The performance of the neural network was also shown graphically with plots of the accuracy as well as the loss at each point.

**Simulated Unlabeled Balanced Validation Sets**

Using results from the probability density simulation performed at the beginning of each test, the accuracies of the neural network's predictions on two balanced validation sets were estimated. These blind accuracy estimates were then compared to the actual accuracy of the neural network's predictions. To measure the effectiveness of the estimations, the instantaneous and cumulative mean squared errors were found for each set of hyperparameters tested. The accuracy estimates were also plotted along with the actual accuracies to show similarities and deviations.

# Results

## Preliminary Tests

### Learning Rate Tests

The following tests were performed on a validation set consisting of 100 images from each class. A mini batch size of 256 was used.

| Learning Rate | Training Set Accuracy | Validation Set Accuracy |
|---|---|---|
| 0.05 | 96% | 95% |
| 0.02 | 97% | 95% |

The smaller learning rate (0.02) performed marginally better than 0.05.

For the next two tests (and all subsequent tests), more aggressive transformations were applied to the images so that the neural network would favor generalizing over memorizing. As a result, the accuracies were lower than those of the previous two tests. Additionally, the augmented validation set was added to represent a more difficult and more realistic test.

| Learning Rate | Training Set Accuracy | Validation Set Accuracy | Augmented Val Set Accuracy |
|---|---|---|---|
| 0.01 | 85.0% | 89.5% | 84.7% |
| 0.02 | 88.7% | 92.4% | 87.6% |

The larger learning rate (0.02) performed significantly better than 0.01. Since the learning rate 0.02 performed marginally better than 0.05 in the previous test, it was assumed that 0.02 was the ideal learning rate.

### Layer Width Tests

The following tests were performed on validation sets consisting of 100 images from each class with a constant learning rate of 0.02.

| Convolution Layer Width | Linear Layer Width | Validation Set Accuracy | Augmented Val Set Accuracy | Computation Time (sec) |
|---|---|---|---|---|
| 24 | 128 | 94.0% | 91.2% | 6148 |
| 48 | 256 | 95.8% | 93.2% | 6304 |
| 64 | 512 | 96.9% | 95.5% | 6432 |
| 128 | 1024 | 97.6% | 97.2% | 6583 |
| 256 | 2048 | 98.2% | 97.4% | 7793 |
| 512 | 4096 | 97.8% | 97.1% | 9810 |

Increasing the layer widths improved the accuracy of the neural network until the convolutional layer width was 256 and the linear layer width was 2048. Increasing the layer widths further reduced the accuracy of the neural network. Computation time also increased with increasing layer width. Initially, this increase was minor, but it rapidly accelerated when the convolution layer width exceeded 128.

A convolution layer width of 256 and linear layer width of 2048 produced the best results.

## Comprehensive Tests

### Dataset Expansion Tests

These tests were performed to observe the effect that the dataset expansion multiplier had on the neural network's reliability. The neural network was trained on approximately 12000000 images in each test.

| Dataset Expansion Multiplier | Validation Set Macro Avg F1 Score | Augmented Val Set Macro Avg F1 Score |
|---|---|---|
| 0 | 95.7% | 91.9% |
| 0.1 | 95.8% | 91.4% |
| 0.3 | 94.5% | 91.4% |

A dataset expansion multiplier of 0 offers marginally better results than a multiplier of 0.1 and noticeably better results than a multiplier of 0.3.

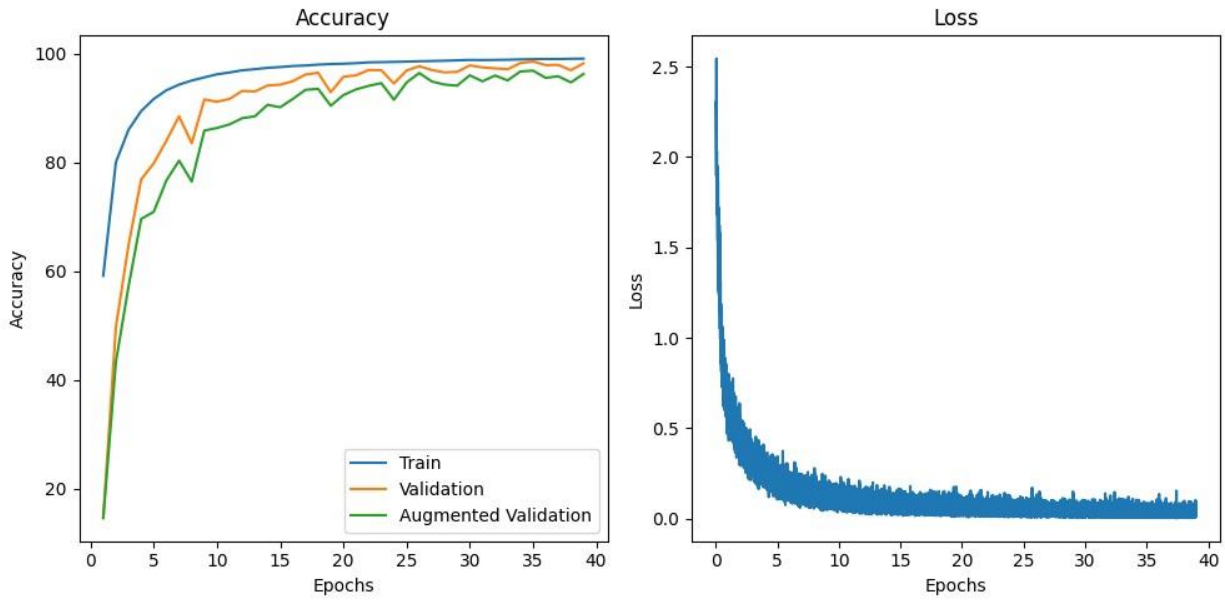


Figure 2: Accuracy and Loss of Neural Network with Dataset Expansion Multiplier of 0.1

**Dynamic Sampling and Learning Rate Decay Tests**

Dynamic sampling and decaying learning rate were implemented in these tests. Since a learning rate of 0.02 was found to produce the best results (at least in the first 15 epochs), the starting learning rate (in cases where it decays) was set to 0.03. This ensured that the learning rate stayed near 0.02, even after decaying over several epochs. The neural network was trained on approximately 12000000 images in each test.

A learning rate decay of 1 indicates that the learning rate is static, whereas a dynamic sampling power of 0 indicates that sampling is static.

| Learning Rate Decay | Dynamic Sampling Power | Validation Set Macro Avg F1 Score | Augmented Val Set Macro Avg F1 Score |
|---|---|---|---|
| 1 | 0 | 95.8% | 91.4% |
| 1 | 1 | 97.7% | 96.4% |
| 0.95 | 1 | 98.4% | 97.8% |
| 0.95 | 2 | NA | NA |

Applying dynamic sampling (linear model) substantially improved the neural network's ability to make correct predictions on the validation sets, especially on the augmented validation set. Applying a decaying learning rate further improved the neural network's performance.

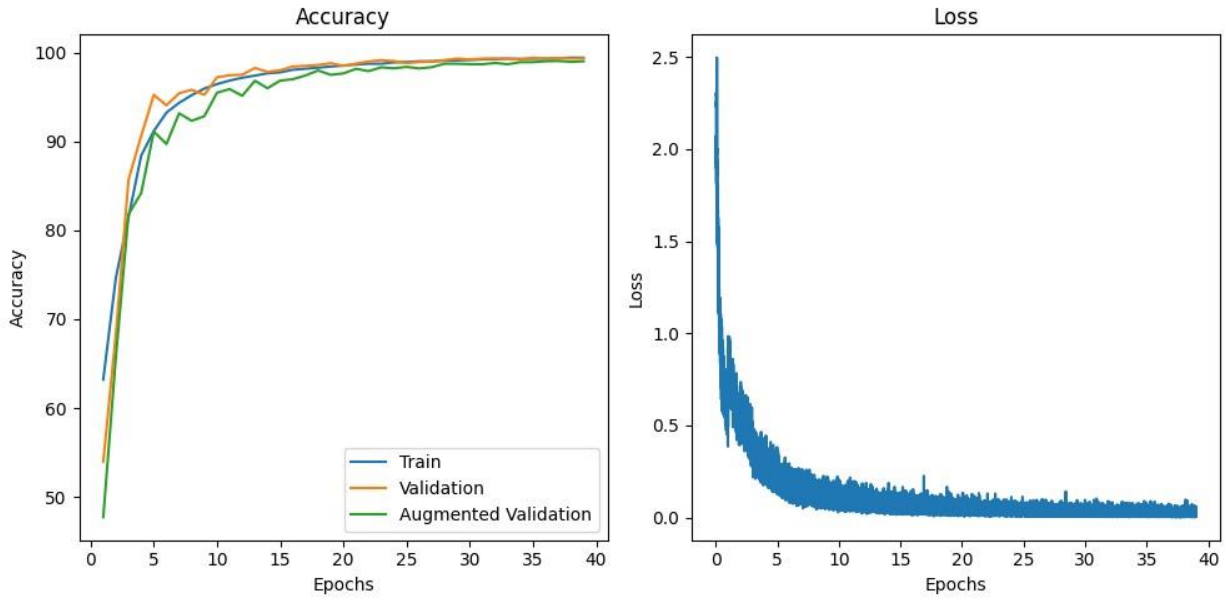


Figure 3: Accuracy and Loss of NN with Linear Dynamic Sampling and Decaying Learning Rate

However, when the dynamic sampling model was changed from linear to quadratic, the neural network diverged after 12 epochs, where all images in the training set were classified as sedans. The divergence was caused by uncontrolled oscillations in the sample sizes between

epochs. Applying a dampening parameter (which takes the weighted average of the current sample sizes and the previous epoch's sample sizes) significantly reduced these oscillations and allowed the quadratic model to function.

| Learning Rate Decay | Dynamic Sampling Power | Dynamic Sampling Dampening | Validation Set Macro Avg F1 Score | Augmented Val Set Macro Avg F1 Score |
|---|---|---|---|---|
| 0.95 | 1 | 0 | 98.4% | 97.8% |
| 0.95 | 1.2 | 0 | 98.5% | 97.8% |
| 0.95 | 2 | 0.1 | 98.4% | 97.3% |
| 0.95 | 2 | 0.15 | 98.5% | 97.7% |
| 0.95 | 2 | 0.2 | 98.7% | 98.2% |

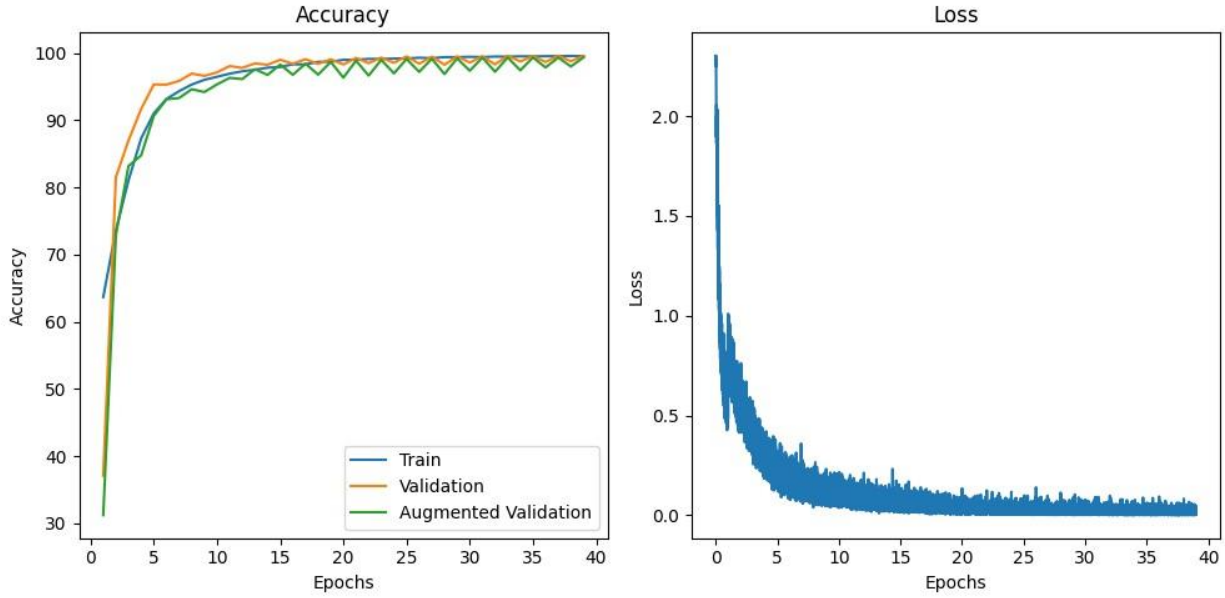


Figure 4: Accuracy and Loss of NN with Under-dampened (0.1) Quadratic Dynamic Sampling

When properly dampened, the quadratic model of dynamic sampling produced noticeably better results than both the linear model and the 1.2 power model.

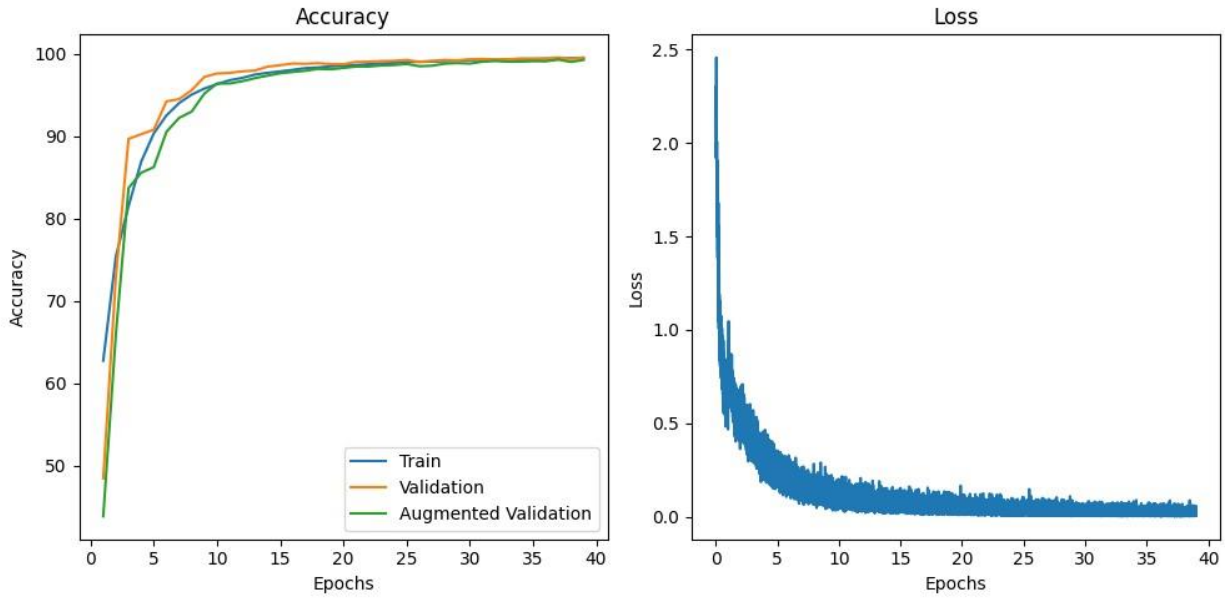


Figure 5: Accuracy and Loss of NN with Properly Dampened (0.2) Quadratic Dynamic Sampling

**Dynamic Dampening**

While static dampening was sufficient at stabilizing quadratic dynamic sampling when the appropriate dampening parameter was used, there was no way to quickly determine what that parameter was supposed to be. Furthermore, even seemingly good dampening parameters could potentially result in oscillations after 30 or 40 epochs. When the parameters that resulted in the best performance in the previous test (power of 2, dampening of 0.2) were used on a neural network that trained on approximately 18000000 images (as opposed to 12000000), the neural network began oscillating after about 40 epochs.

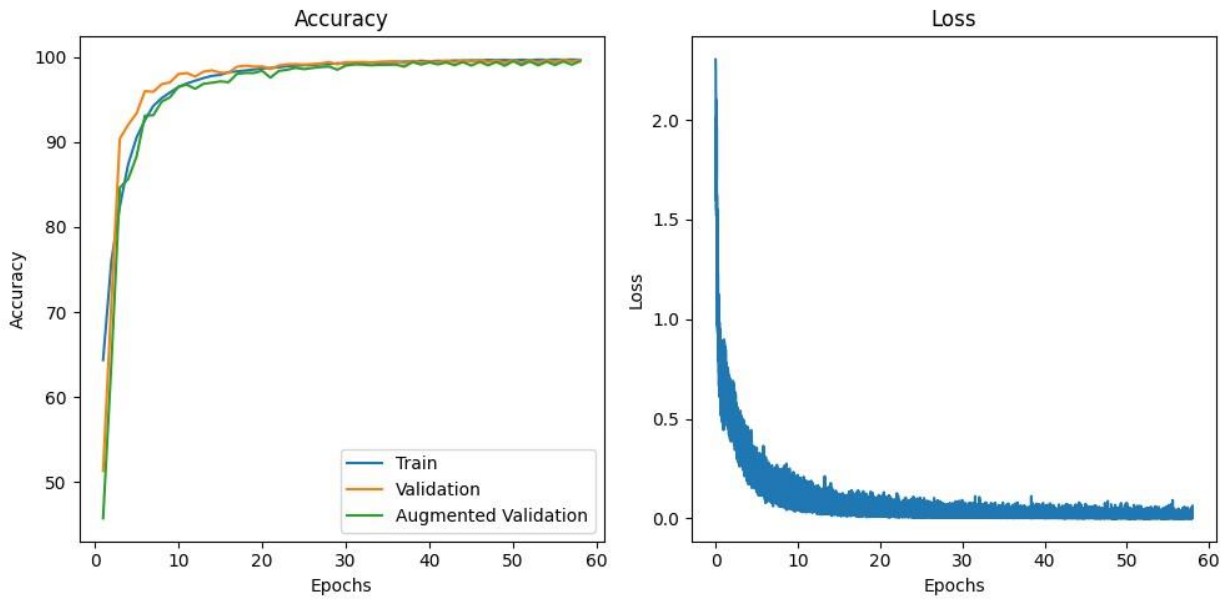


Figure 6: Accuracy of NN with Quadratic Dynamic Sampling Oscillating after 40 Epochs

Dynamic dampening increases the dampening parameter when oscillations are detected and decreases it when they are not detected. The forward step size used for all tests was 0.1, whereas the reverse step size was 0.04.

All tests were performed using quadratic dynamic sampling with a total training size of approximately 18000000 images. Additionally, the performance of the neural network with dataset expansion multiplier set to zero was retested. Since that neural network has a smaller training set (since it doesn't create duplicate images), the initial learning rate and learning rate decay parameters were set to 0.0303 and 0.967, respectively. This ensured that every test started and finished with the same learning rate.

| Dataset Expans Mult | Initial Dampen | Min Dampen | Max Dampen | Dampen Step Threshold | Val Set Macro Avg F1 Score | Augment Val Set Macro Avg F1 Score |
|---|---|---|---|---|---|---|
| 0.1 | 0.2 | 0.2 | 0.2 | 0 | 98.8% | 98.2% |
| 0.1 | 0.05 | 0 | 1 | 0.95 | 98.5% | 98.3% |
| 0.1 | 0.05 | 0 | 1 | 0.9 | 99.0% | 98.4% |
| 0 | 0.05 | 0 | 1 | 0.9 | 98.9% | 98.5% |

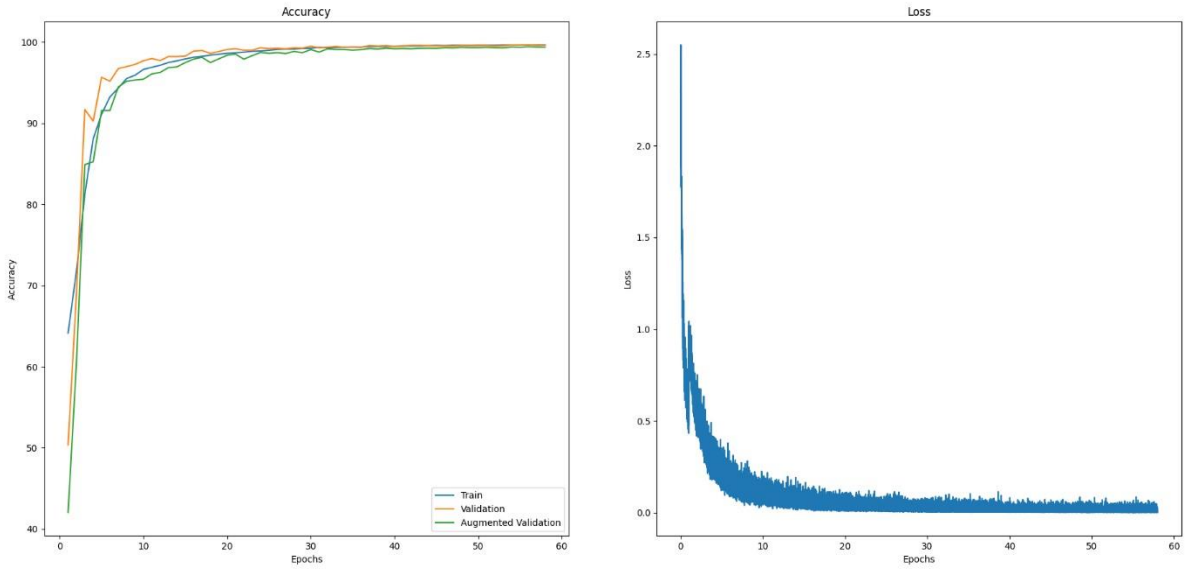


Figure 7: Accuracy and Loss of NN with Quadratic Dynamic Sampling and Dynamic Dampening

The neural networks with quadratic dynamic sampling with dynamic dampening applied both performed better than the neural network with static dampening. There was no appreciable difference in performance between the neural network with a dataset expansion multiplier of zero and the neural network with a multiplier of 0.1.

However, a neural network with a dataset expansion multiplier of 0.1 will have fewer epochs for a given total training size, which means that the validation sets will be analyzed fewer times. This can potentially save a significant amount of time. The test with dataset expansion multiplier set to 0.1 took 33778 seconds to complete, whereas the test with dataset expansion multiplier set to zero took 38973 seconds to complete.

## Probability Density Simulation Tests

### Smaller Dataset

In order to speed up testing, a smaller dataset consisting of ~25% of the images in the original dataset was initially created. The data set was run through the neural network as before, except that after each time the validation set was analyzed, the simulation attempted to estimate the accuracy of the predictions using only information about the distribution of the predictions and the fact that the validation set was balanced. The estimates were then graphed against the true accuracy and compared using mean squared error.

After testing several hyperparameter combinations, a set was found that produced the following result.

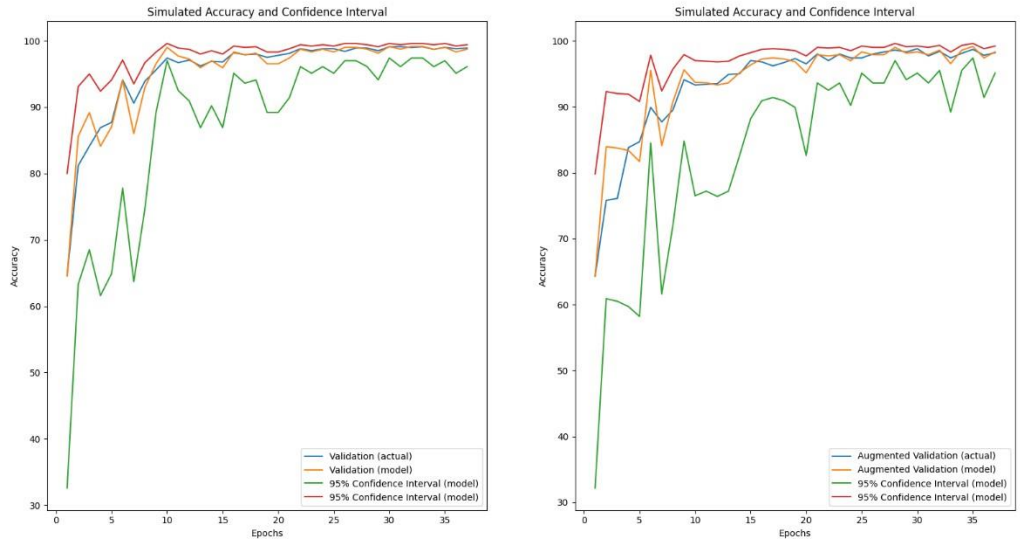


Figure 8: Comparison of Estimated Accuracy and Actual Accuracy of the Validation Set Predictions for Smaller Dataset

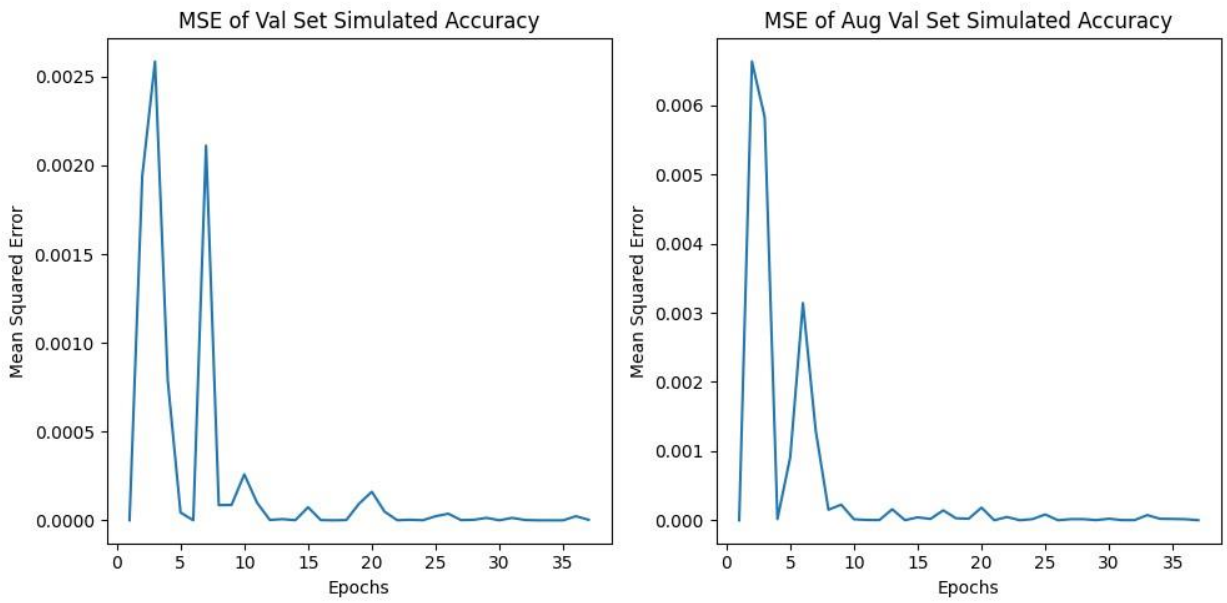


Figure 9: Mean Squared Errors of the Accuracy Estimates for Smaller Dataset

The overall mean squared error was found to be 0.000230 for the validation set and 0.000519 for the augmented validation set. The hyperparameters that produced these results were correlation = 0.17, probability weights offset = 1000 to 5000, probability weights base = 2 to 2, probability weights power = 0 to 19.

**Original Dataset**

This process was then repeated on the original dataset. Because the class weights of the reduced data set were not the same as the original dataset, the hyperparameters that were found using the smaller dataset did not perform as well on the full dataset. However, the results were still more than sufficient for obtaining a reasonably good estimate of the accuracy.

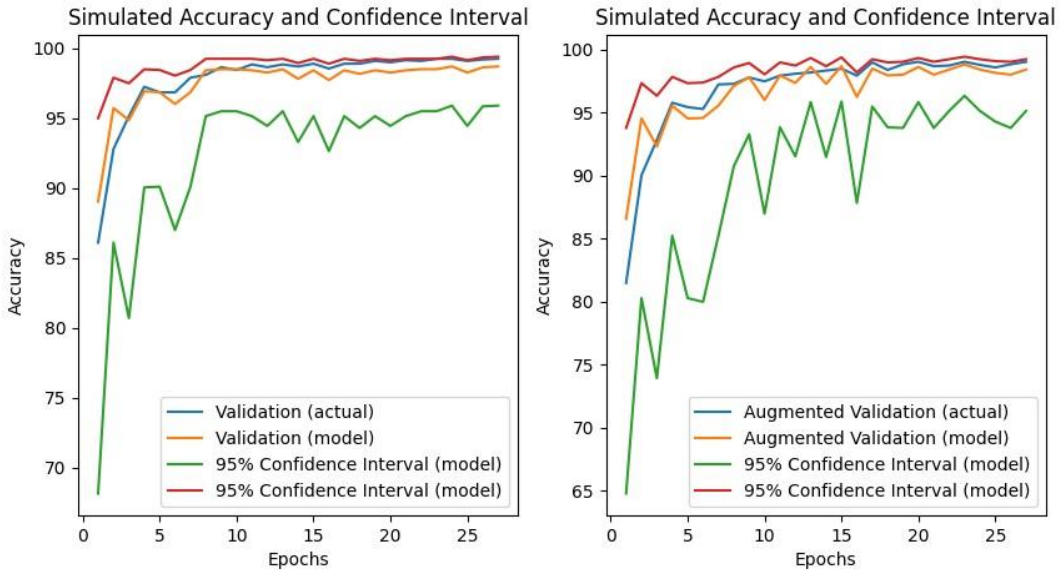


Figure 10: Comparison of Estimated Accuracy and Actual Accuracy of the Validation Set Predictions for Original Dataset

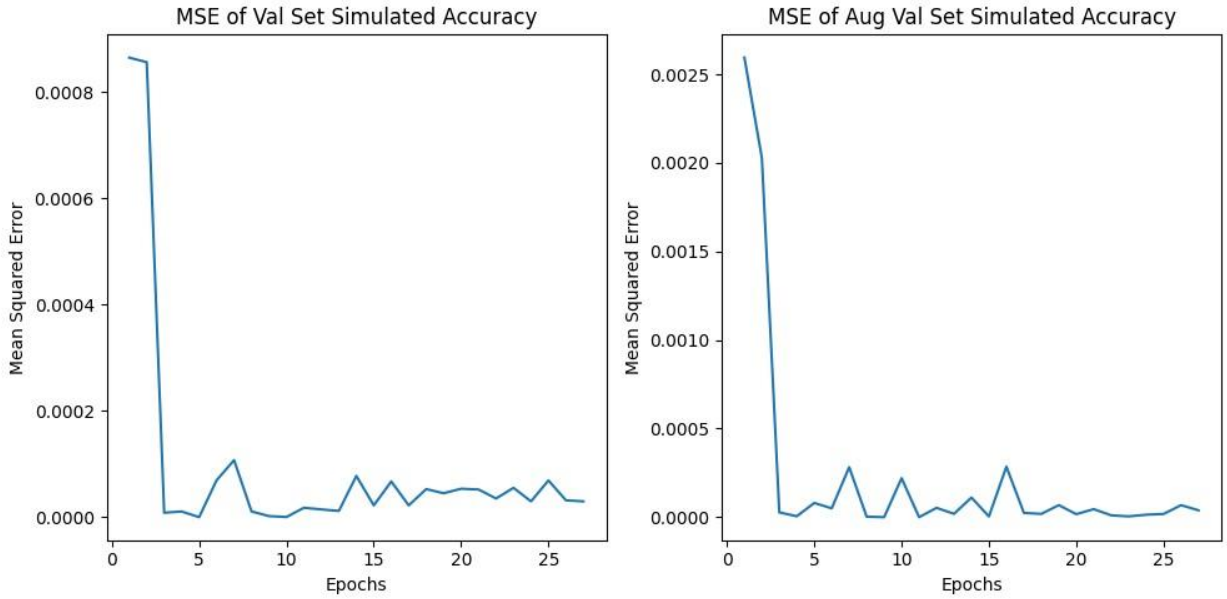


Figure 11: Mean Squared Errors of the Accuracy Estimates for Original Dataset

The model tended to underestimate the accuracy in both validation sets by ~0.5% when the predictions were more than 98% accurate. The mean squared error over all predictions was found to be 0.0000970 for the validation set and 0.000226 for the augmented validation set.

The model was also tested at lower accuracies (by running the neural network with a very low learning rate) to see if it stayed consistent when estimating the accuracy of predictions made by poorer neural networks.

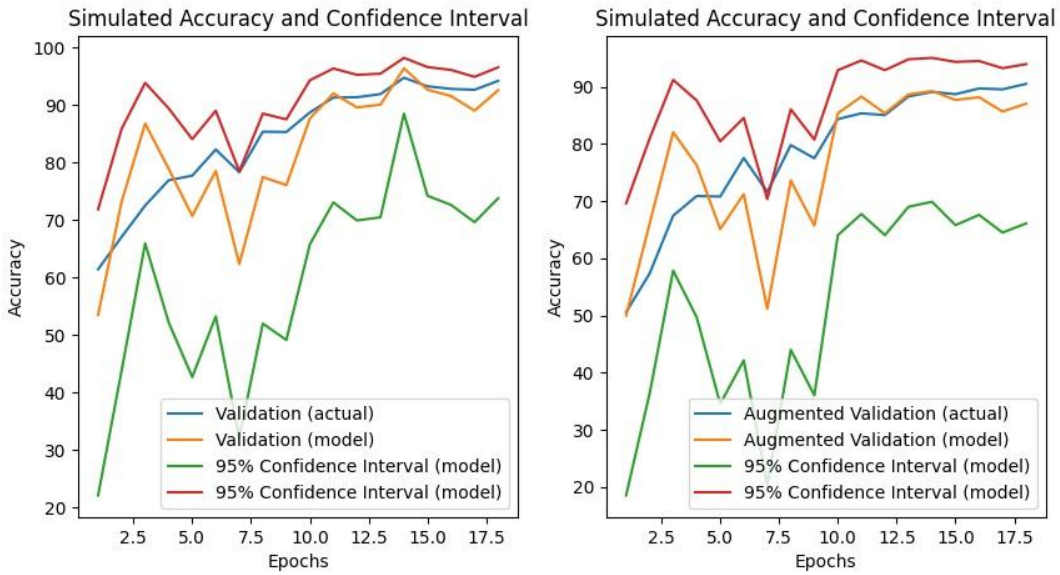


Figure 12: Comparison of Estimated Accuracy and Actual Accuracy of the Validation Set Predictions for Original Dataset at Low Accuracies

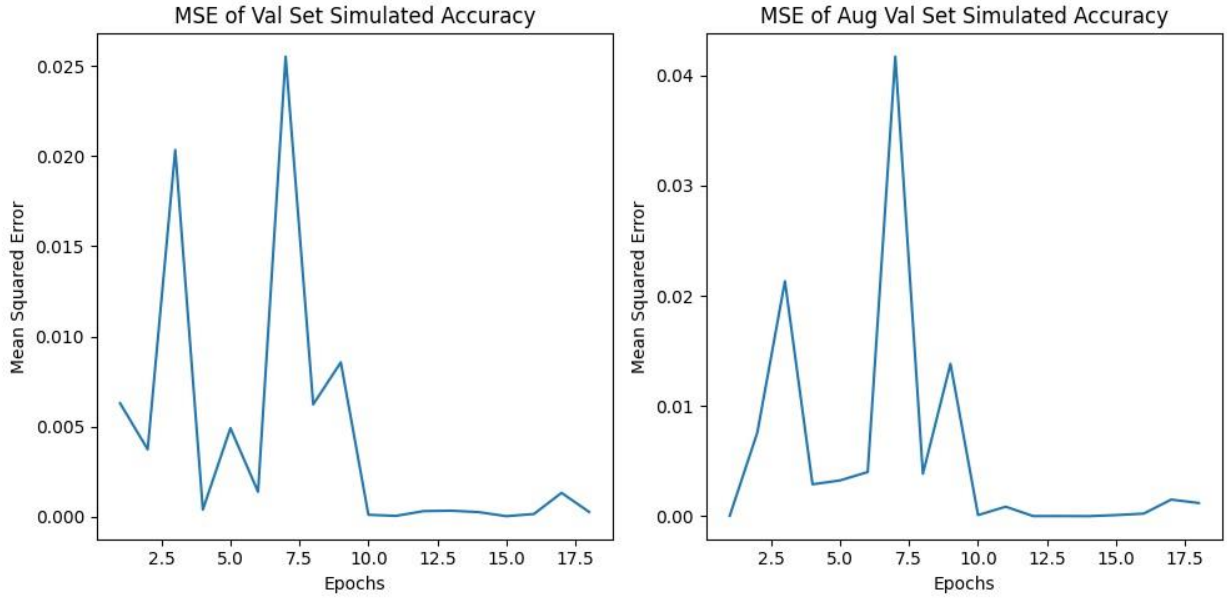


Figure 13: Mean Squared Errors of the Accuracy Estimates for Original Dataset at Low Accuracies

Here, the model performed noticeably worse. Unlike at higher accuracies, at lower accuracies the model seems to be very inconsistent, and any individual, or even small group of estimates cannot be relied upon. The mean squared error over all predictions was found to be 0.00446 for the validation set and 0.00570 for the augmented validation set.

Even if the model is not terribly reliable at estimating accuracies when they are low, the ability to consistently estimate accuracies when they are high (more than 90 to 95%) is still very useful. With this capability, the model can classify a set of predictions as either good or poor, and if they are good, it can provide an accurate estimate of the accuracy. This makes it potentially useful for fine tuning classification systems that are working with unlabeled balanced data sets.

## Prediction Results for some Samples

The neural network's final layer has a width of ten, which is equal to the number of different classes. After an image's pixels' values are sent to the neural network's input layer, the neural network performs a series of calculations based on its structure, weights, and biases, before ultimately outputting ten resulting values. The largest of these values determines the neural network's prediction.

By applying softmax to the output layer (which normalizes the total of all the values to one), the probabilities of each class being correct for the given input can be analyzed. The image predictions were obtained from the $32^{nd}$ epoch, where the neural network has mostly plateaued.

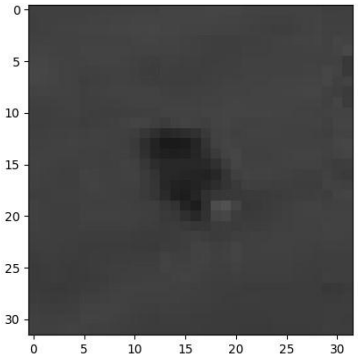

Image 1:

Prediction: 0, Actual label: 0

Probabilities: tensor([9.9999e+01, 3.2419e-06, 1.8678e-06, 8.8187e-04, 1.1561e-08, 8.0746e-05, 4.1297e-10, 5.7411e-10, 3.4955e-11, 3.9648e-12], device='cuda:0')

Here, the neural network is 99.999% certain that the vehicle in the image is a sedan. This indicates that this image is likely far away from the edges that define the sedan class. The next most likely candidate is a van, with a 0.00088% probability. Since it is the next highest probability, it indicates that the closest class separating boundary to the image is likely the one that separates sedans and vans.

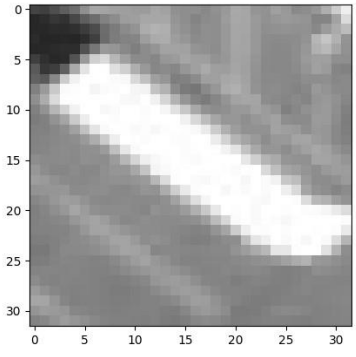

Image 8:

Prediction: 7, Actual label: 7

Probabilities: tensor([1.0176e-14, 1.2206e-21, 1.4421e-23, 2.3307e-32, 2.2195e-14, 4.8680e-23, 4.2830e-35, 1.0000e+02, 1.0268e-25, 4.2167e-29], device='cuda:0')

Here the neural network is 100.00% certain that the vehicle in the image is a bus. The next most likely candidate is a box truck with a 2.2195 * $10^{-14}$% chance. Such low probabilities indicate that this image is in a region that is very well defined as one belonging to the bus class.

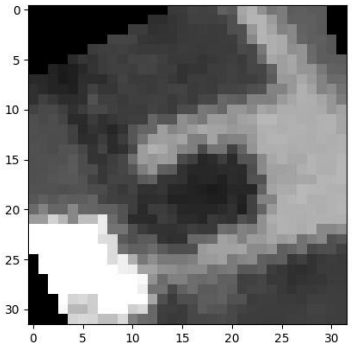


Image 9:

Prediction: 8, Actual label: 8

Probabilities: tensor([1.9137e+01, 4.0354e-02, 2.3890e-13, 5.5377e-11, 3.9421e-09, 1.5538e-15, 4.5983e-10, 4.6913e-15, 8.0822e+01, 3.2996e-15], device='cuda:0')

Here the neural network is 80.8% certain that the vehicle in the image above is a pickup truck with a trailer. However, the neural network also thinks that there is a 19.1% chance that the vehicle above is actually a sedan. The fact that there is a much higher uncertainty (than the previous two examples) indicates that the region that this image occupies is not very well defined.

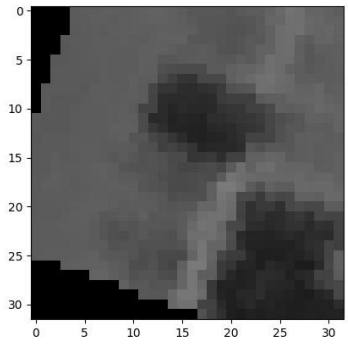

Image 13:

Prediction: 0, Actual label: 2

Probabilities: tensor([5.4741e+01, 4.0434e-01, 4.4855e+01, 3.2353e-04, 3.1109e-08, 8.3234e-09, 3.0767e-09, 1.2498e-10, 1.2050e-11, 1.3440e-14], device='cuda:0')

The neural network predicted that the vehicle in this image is a sedan with a 54.7% certainty, when it is actually a pickup truck, which it gave a 44.9% probability. The fact that the two probabilities are so close to one another indicates that this image occupies a region that is very poorly defined. It is possible that with additional training, the boundary between the two classes will become more well defined and possibly shift so that this image will be classified correctly in subsequent epochs.

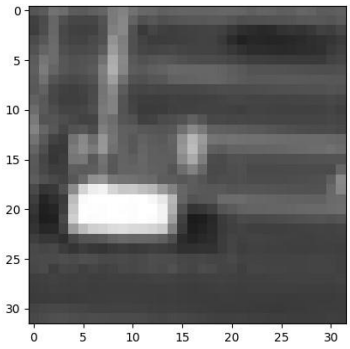

Image 15:

Prediction: 0, Actual label: 4

Probabilities: tensor([9.9833e+01, 1.5699e-02, 8.2682e-02, 6.4639e-02, 3.2814e-03, 4.6618e-04, 1.3093e-08, 2.7997e-07, 3.1739e-05, 4.8436e-06], device='cuda:0')

The neural network predicted that the vehicle in this image is a sedan with a 99.8% certainty, when it is actually a box truck, which it gave a 0.00328% probability. In this case, there is a massive discrepancy in the probabilities. Since the neural network has already mostly plateaued, it is unlikely that this prediction will correct itself with time. One possible cause for such a huge discrepancy in probabilities is that there may be a large number of very similar images that belong to the sedan class. If that is the case, then if the boundary separating box trucks and sedans were to be redrawn in a way that would correct this prediction, it may cause a significant number of other predictions to suddenly become incorrect.

## Conclusion

The aim of this project was to train a convolutional neural network (CNN) to identify vehicles from aerial images using a heavily imbalanced dataset. The CNN architecture used consisted of two convolution layers followed by three linear hidden layers, and the model was trained using a mini-batch gradient descent method with a momentum parameter.

The dataset was preprocessed by creating a validation set and augmenting the training set to reduce overfitting. A second validation set with augmentations was created from the original validation set to provide a more realistic estimate of the CNN's performance.

Additionally, a probability density simulation was implemented and used to estimate the accuracy of a balanced unlabeled set by analyzing the distribution of the predictions.

The results showed that the CNN's performance improved significantly with the implementation of dynamic sampling (which utilized f1 scores) and a decaying learning rate (which decayed exponentially). Dynamic sampling not only solved the class imbalance problem, but also accelerated the fitting of underfit classes, while reducing overfitting. The quadratic model of dynamic sampling, when properly dampened with dynamic dampening, produced noticeably better results than both the linear model and the 1.2 power model.

The probability density simulation was found to reliably estimate the accuracy of predictions on a balanced unlabeled data set using only the distribution of predictions. It performed best when predictions were mostly correct (over 90%).

Using the methods discussed, the CNN was successfully trained to identify vehicles from aerial images with a macro average f1 score ranging from 98.5% to 99.0%. This project highlighted the importance of careful parameter tuning and the potential issues that can arise from an improper selection (such as the oscillations observed). Future work could focus on further refining these parameters and exploring other techniques to improve model performance.

## References

Devansh. (2022). How does Batch Size impact your model learning. Medium.

https://medium.com/geekculture/how-does-batch-size-impact-your-model-learning-2dd34d9fb1fa