Goals and Discussion

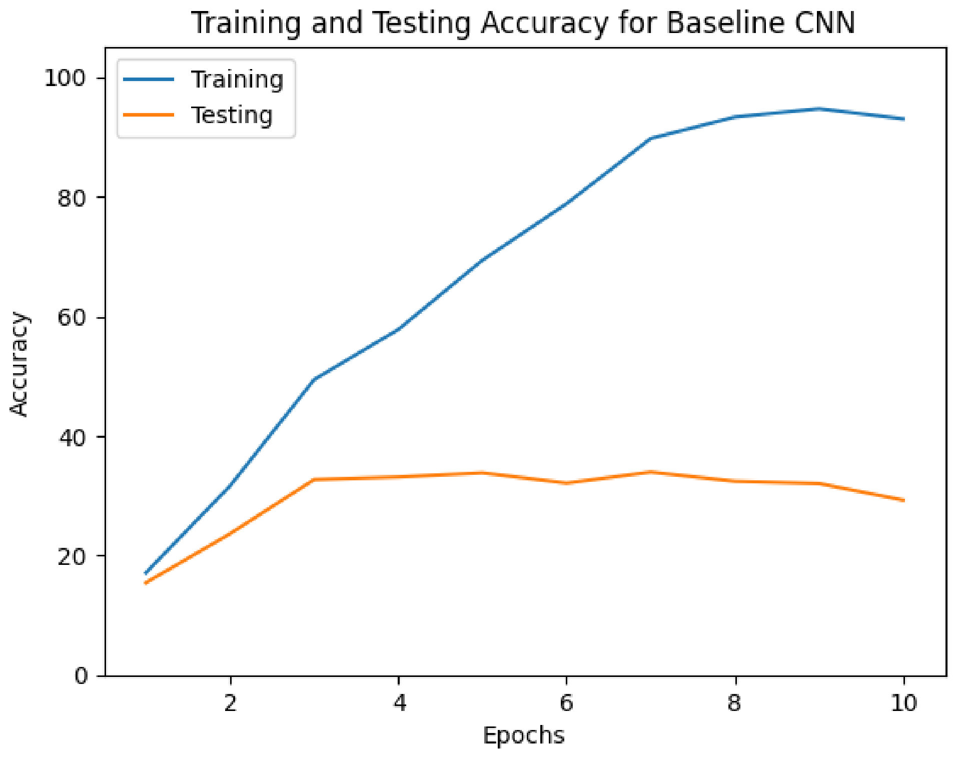
Essential goals

Logistic Regression Model

We used scikit-learn to train a logistic regression model on our dataset to serve as a point of comparison for our CNN models. A difficult part of completing this goal was that the images had to be loaded on to the RAM when converting the tensors to numpy arrays which ate up around 50 GB of RAM and then increased to 80 GB when training of the logistic model began. What was interesting is that the model achieved an 85% training accuracy but only 1.7% testing accuracy.

Baseline CNN Model

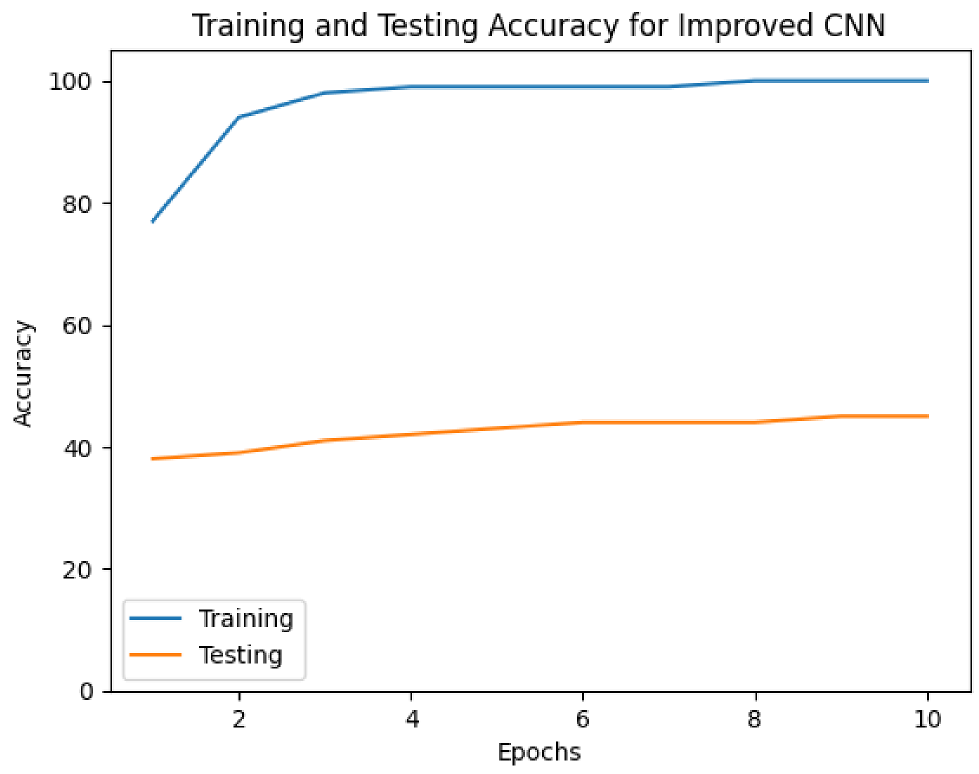
We built our baseline CNN model by recreating the model in a Microsoft tutorial (found here: https://learn.microsoft.com/en-us/windows/ai/windows-ml/tutorials/pytorch-train-model). After training it on our data, we achieved a testing accuracy of 29%. It was interesting to see how easy it was to make a simple CNN model that could classify our data somewhat, especially since our data has 45 different classes to choose from.



Desired goals

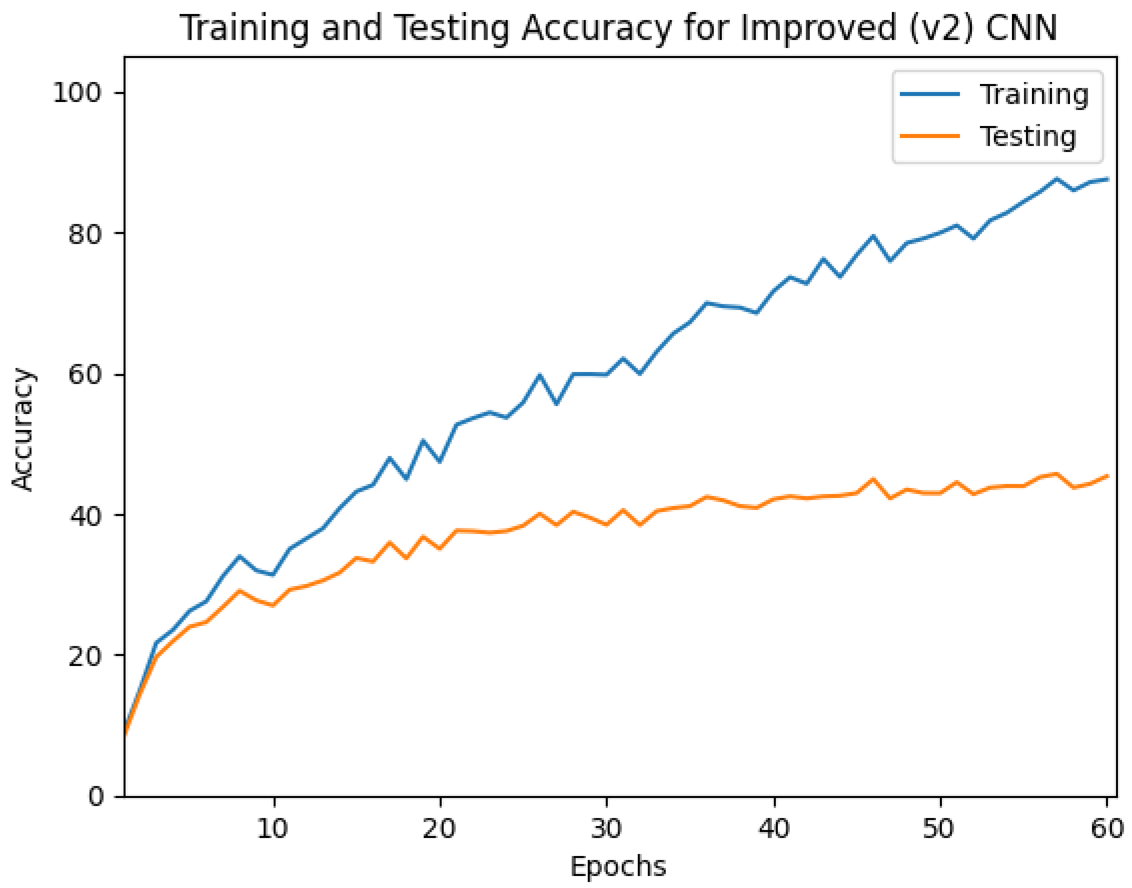
Improved CNN Model

We altered our simple CNN's model architecture and fine-tuned the hyperparameters, ultimately achieving a testing accuracy of 45%. It was interesting to see the results of changing aspects of the model architecture, but it was difficult to find the optimal solution since there were so many variables to change.



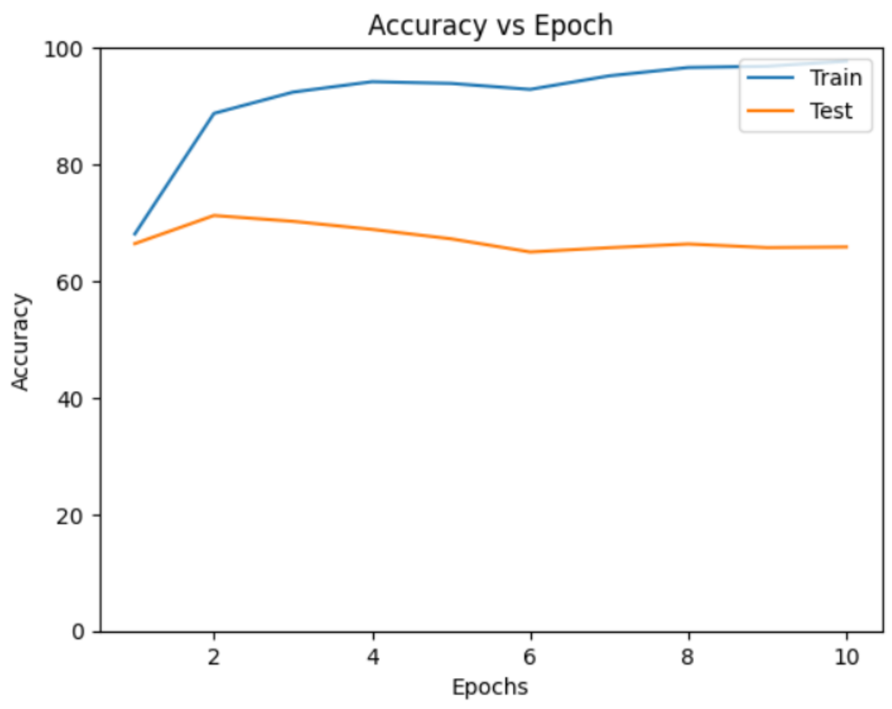
Small Improved CNN Model

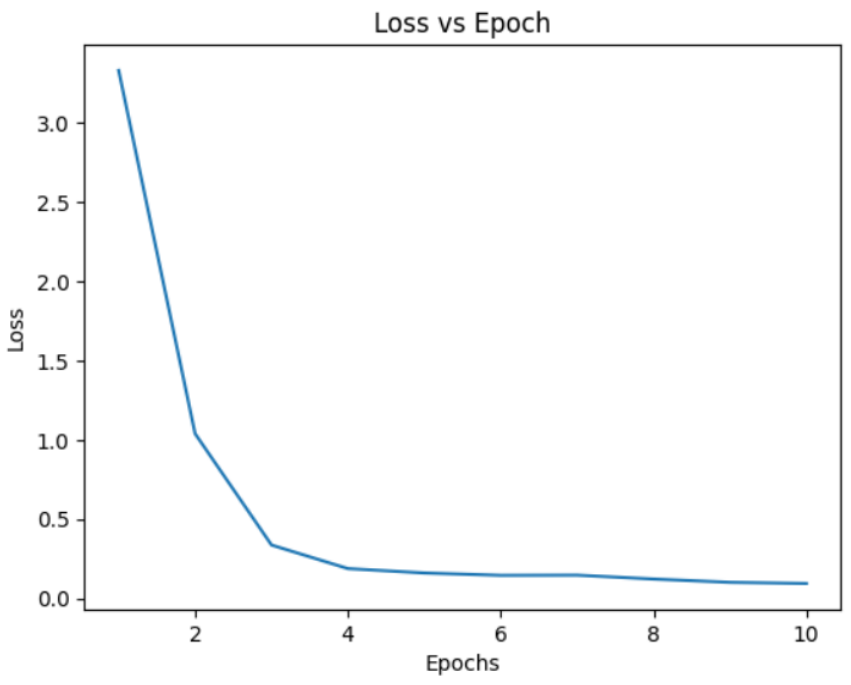
This model was based on the improved CNN's architecture. It was designed to reduce model size and overfitting in order to allow for faster training times, especially for the data augmentation. This model still achieved a testing accuracy of 45%, but with a quarter of the number of parameters. The main modifications were adding more CNN layers and reducing the size of the linear layers.



Data Augmentation

We performed data augmentation on our training set by rotating the images in our training set by 90, 180, and 270 degrees, which quadrupled the size of our data. We used this augmented dataset to train on an improved CNN model (around 43% accuracy), which resulted in an additional 20% increase to the accuracy. This was interesting as it showed us that more data can also help improve the accuracy of small datasets like the one we worked with. We struggled with preventing overfitting since the training set became so much larger than the testing set. One thing to note about the below images is that the training accuracy is only calculated on the original images just like the testing accuracy is tested on the original images.





Evanston Satellite Images

We did not realize the Google Maps Static Images API was a paid API when creating this goal for ourselves. We searched for suitable alternatives, including many datasets provided by USGS (US Geological Survey) but none were high-quality enough to be usable by our model.

Stretch goals

Train on Adversarial Images

Adversarial image training required modifying the training code, slightly perturbing the training images in the opposite direction of their correct labels. This adversarial training didn't seem to make much of a difference in terms of increased test accuracy. Additionally, the training took much longer than normal training. Thus, because early epoch testing performance wasn't promising and the long training time, we stopped training early.

Train on GAN Images

We unfortunately did not have enough time to complete this goal, due to training requirements.

Code and documentation

File name	**Description**
logistic/basicLogistic.ipynb	This file contains the code for training our logistic regression model. This serves as a baseline for comparing to our CNN.
cnn/baseline_CNN.ipynb	This file contains the code for training our baseline CNN model. This model will be improved upon in several different ways to compare resulting accuracies.
cnn/improved_CNN.ipynb	This file contains the code for training our improved CNN model. This is one way we attempted to improve on our baseline CNN.
cnn/improved_CNN_2.ipynb	This file contains code based on the original improved CNN. This model was modified to be smaller and reduce overfitting, while still achieve similar accuracy.
cnn_augmented/cnn_augment_final.ipynb	This file contains the code for training our baseline CNN model with additional data that were augmented. This is one way we attempted to improve on our baseline CNN.

Reflections

What was interesting?

This project allowed us to take a very deep dive into image classification. We learned the incredible importance of tuning the hyperparameters of your model, the importance of pairing the size of the model with the amount of data you have. We also saw the effects of dropout and how it really helped create a more generalizable model (and how it reduces training time).

What was difficult?

The long training times compounded all other difficulties. A change to the model can't simply be tested, there is a significant time investment you must make. So, updates to the model must be very well thought out.

Another difficulty was the nondeterministic nature of training. This made some improvements seemingly very beneficial, but another round of training would prove the "improvement" to be a fluke, perhaps a favorable initialization of parameters or training/testing data split. This made testing changes unpredictable, especially with our limited training hardware/time.

What's left to do?

The largest gains in accuracy we saw stemmed from increasing our data and model size. So, the bulk of the money would be spent on acquiring more training data and accordingly increasing the size of our model. Additionally, other types of model architectures would be tested. We remained using our relatively basic CNN-based model. This worked well, but there are certainly improvements that have been outlined in published papers to improve the accuracy of our model.