Traffic Sign Recognition using Convolutional Neural Networks

Spencer R. Karofsky

Computer Science Major; Mathematics Minor

College of Engineering and Mathematical Sciences at the University of Vermont

May 2023-June 2023

Abstract:

Artificial Intelligence fascinates me, and I have spent my time outside of my computer science curriculum teaching myself machine learning, deep learning, neural networks, and computer vision.

To develop my skills in deep learning and computer vision, I trained a Convolutional Neural Network to classify eight common traffic signs: Stop, Yield, Do Not Enter, No U-Turn, No Left Turn, No Right Turn, One Way (Left), and One Way (Right).

First, I custom-built a dataset by creating miniature, freestanding traffic signs, and took dozens of pictures of each sign. I then applied OpenCV transformations to each image to artificially enlarge the dataset.

I used the ResNet-50 Neural Network architecture to train the dataset. I chose this architecture experimentally, as it yielded the best accuracies of all the architectures that I tested over numerous runs.

During the initial stages of training, I only achieved accuracies that were at best 60-80% on the validation set. After experimenting with numerous architectures and tuning the hyperparameters, I achieved a 94.04% accuracy on the validation set and a 95.24% accuracy on the testing set.

Definitions:

- Artificial Intelligence: Computers mimicking and possessing characteristics of human-like intelligence.
- Artificial Neural Network: A method in artificial intelligence that teaches computers to process data through layers of interconnected nodes consisting of inputs, weights, and biases.
- Convolutional Neural Network: A type of artificial neural network that uses convolution layers (a filter/kernel that performs a mathematical operation on the pixels of an image) to classify images.
- Deep Learning: A subset of machine learning that uses artificial neural networks to mimic the human brain. A neural network is considered deep learning if it has three or more layers.
- GPU (Graphical Processing Unit): A specialized computer component that improves processing time on images through the manipulation of memory allocation and de-allocation.
- Machine Learning: A subfield of artificial intelligence, which teaches computers to learn from prior data to make predictions (regression) and classify unseen data.
- NumPy: A Python open-source linear algebra library used primarily in working with arrays.
- OpenCV: A open-source computer vision and machine learning library
- Preprocessing: Organizing and formatting the data to produce an output for another program. In the context of deep learning, preprocessing is organizing and formatting the data to feed into the artificial neural network.
- Python: high-level general purpose programming language

- TensorFlow: an open-source library primarily used for machine learning and deep learning applications.

Project Description:

I printed out images of 8 common traffic signs: Stop, Yield, Do Not Enter, No U-Turn, No Left Turn, No Right Turn, One Way (Left), and One Way (Right).

Stop  i	Yield  ii iii	DO Not Enter  iv	No U-Turn  v
No Left Turn  vi	No Right Turn  vii	One Way (Left)  viii  x	One Way (Right)  x  xi

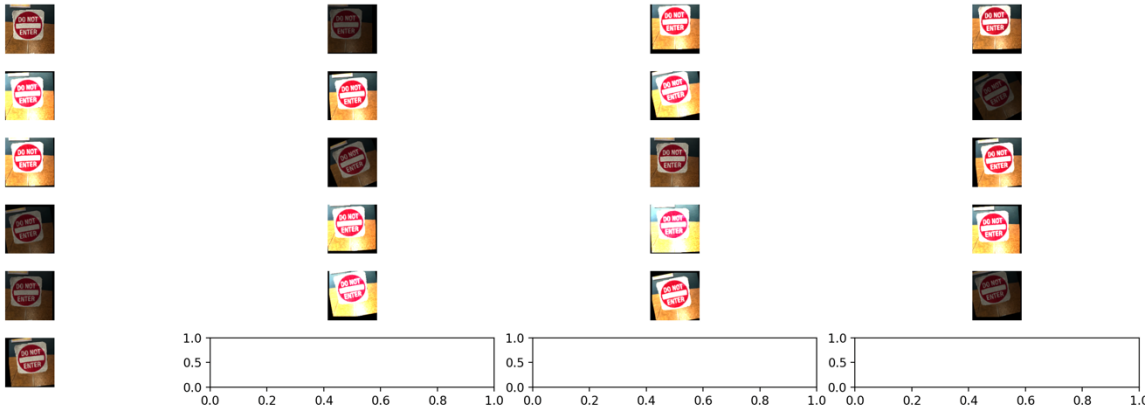
I built the dataset by printing images of each sign and creating miniature freestanding traffic signs using paper cardstock, paperclips, scotch tape, and toothpicks; I printed out two different signs for some classes which have different variations.

1 Miniature Traffic Signs I made to build the dataset

I then took between 40 and 90 pictures of each element, taking pictures of the signs at different angles and with different backgrounds and lighting conditions to mimic real-life conditions. I then used OpenCV's `imread()` function to read in the images and convert to NumPy arrays so that each image can be resized to size (224,224,3), where 224 is the width and height of the image, and 3 is the number of color channels (red, green, and blue). Choosing a width and height of 224 pixels was motivated by the ResNet-50 Convolutional Neural Network architecture, which conventionally takes in image inputs of size (224,224,3). Converting the images to NumPy arrays also makes them compatible with TensorFlow models, which only accept image inputs in NumPy array format.

Below are four example images in the dataset from the Do Not Enter class after being loaded into the program as NumPy arrays by OpenCV and resized to size (224,224,3).

Other datasets I have worked with have closer to 40,000 to 60,000 images in their datasets; my custom-built dataset contains only 559 images. I built a function, `augmentImage(img)`, that creates new images from the base images using the following transformations: rotation, translation, contrast adjustment, and brightness adjustment. The transformations follow a random value on a normal distribution (also known as a Gaussian distribution) to introduce Gaussian noise to the dataset. Including Gaussian noise additionally ensures that the augmented images are entirely random.



1 The top left image is the base image, and the other images are all generated by augmentImage()

This function allowed me to artificially enlarge the dataset to $n = b(1 + c)$ images for the dataset size, where n is the total number of images, b is the number of base images, and c is a constant represented by the variable `AUGMENT_SIZE`. This enlarges the dataset to $n = 559(1 + 5) = 3{,}354$ images, where 5 is the constant.

My reasoning for choosing 5 as the constant (which still only enlarges the dataset to a relatively small size) is twofold. Primarily, the dataset is very simple and lacks a diversity of backgrounds and setups for each class. Accordingly, the model training on such a large dataset of such similar images would not improve the accuracy of the model. My secondary reason for a smaller dataset concerns the training time and computational complexity. Increasing the size of the dataset increases the number of calculations in the neural network, which increases the training time.

I then split the entire dataset into 3 separate datasets: 80% for training, 15% for validation, and the remaining 5% for testing in another file.

Finding an optimal neural network architecture that yielded a high accuracy was the most difficult task. My goal for this project was to build a model that could classify traffic signs with 90% accuracy; going into this project, I didn't know about the different popular neural network architectures in academia and industry. I started with a very simple architecture that I had seen in a TensorFlow tutorial for classifying the MNIST handwritten digits dataset. It had two sets of 2D convolution layers and subsequent max pooling layers, a dropout layer, a flatten layer, and 2 dense layers. This simple architecture wasn't accurate, so I tried using the Keras Tuner library,

testing for the optimal kernel sizes, number of nodes, and number of epochs to optimize the validation accuracy.

The hyperparameter tuning took almost an hour to complete. In order to speed up the training and computation times, I installed the TensorFlow metal plugin to enable GPU support. This made the model training time roughly 5 times faster, bringing the training time per epoch from roughly 45 seconds to 11 seconds.

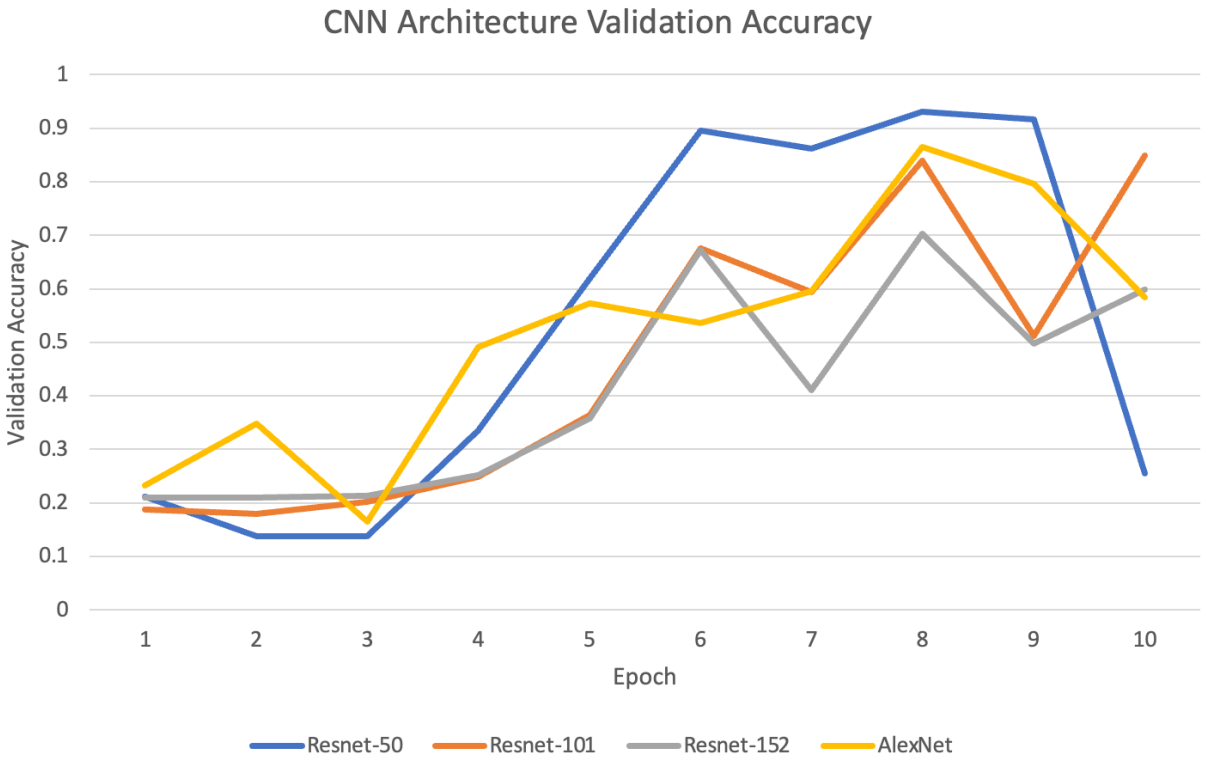
The enabled GPU support[xii] allowed me to train and test different hyperparameters and architectures much faster, and the tuner found the optimal hyperparameter values to be what seemed unreasonably high. Moreover, using the most optimal hyperparameter values still only yielded a 60-70% accuracy for the validation set at best, which was nowhere near my goal of >90% validation accuracy.

I researched the most effective convolutional neural network architectures for image classification, and settled on AlexNet, and the 50-, 101-, and 152-layer versions of ResNet. I tested these four architectures over 10 epochs each and experimentally determined that the

ResNet-50 architecture yielded the best accuracy.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

2 Different ResNet Architectures[xiii]



ResNet-50 is clearly the best performing architecture; it achieved a 90% validation accuracy during the 8th and 9th epochs, but then dropped off to below 30 during the 10th epoch due to

overfitting. I ran the trainer with the ResNet-50 architecture over a dozen times, and I would continuously encounter this issue no matter the epoch hyperparameter setting.

To amend this issue, I researched ways to stop the model from training once the validation accuracy surpassed a predetermined threshold. I discovered the EarlyStopping[xiv] class in TensorFlow's callbacks module. I then implemented early callback functionality into the model and set it to stop training if the model achieved a 90% or high validation accuracy.

It still took multiple runs using this new code to achieve 90% accuracy but using this functionality, I achieved a 94.04% validation accuracy.

Areas for Improvement and Analysis:

This model is extremely accurate on the dataset that I created, but it fails to correctly classify the same traffic signs in real life. I attribute this discrepancy to the quality of my dataset, as the images all used one or two objects for each class, and there was a lack of diversity in backgrounds. This caused severe overfitting in the dataset, which is why it fails to accurately predict real-world traffic signs.

A reason why this project is not accurate on real-world traffic signs is due to the size of the dataset. The dataset has only 3,354 images after augmentation, which is far below other famous datasets, which commonly have at least 10,000 images. I chose to not augment the dataset to this size due to the lack of diversity in the dataset; increasing the dataset larger would not have increased the accuracy on real-world datasets due to the similarities between the images.

Overall, I am very satisfied with the final product. This project significantly developed my skills and confidence in TensorFlow, OpenCV, deep learning, convolutional neural network architectures, image augmentation, and debugging, and I feel much more capable of taking on larger and more complex deep learning and computer vision projects in the future.

References:

[i] https://www.vecteezy.com/free-vector/stop-sign

[ii] https://etc.usf.edu/clipart/74500/74541/74541_75_r1-2_c.htm

[iii] https://www.squaresigns.com/templates?name=yield-sign-with-triangle-shape-on-a-yellow-background

[iv] https://techedsafety.com/do-not-enter-sign/

[v] https://www.amazon.com/U-Turn-Symbol-Sign-Turn-Around/dp/B08KZDNFZ9

[vi] https://www.municipalsigns.com/products/r3-2-no-left-turn-sign?variant=24204918979&currency=USD

[vii] https://www.roadtrafficsigns.com/turn-sign/no-right-turn-symbol-sign/sku-x-r3-1

[viii] https://www.allstatesign.com/one-way-left-sign.html

[ix] https://www.roadtrafficsigns.com/one-way-sign/one-way-left-arrow-sign/sku-x-r6-1l

[x] https://www.roadtrafficsigns.com/one-way-sign/one-way-right-arrow-sign/sku-x-r6-1r

[xi] https://www.roadtrafficsigns.com/one-way-sign/one-way-right-arrow-symbol-sign/sku-x-r6-2r

[xii] D. Ganzaroli, “Install tensorflow on Mac m1/m2 with GPU support,” Medium, https://medium.com/mlearning-ai/install-tensorflow-on-mac-m1-m2-with-gpu-support-c404c6cfb580 (accessed Jun. 25, 2023).

[xiii] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 2016, pp. 770-778, doi: 10.1109/CVPR.2016.90.

[xiv] [1]B Chen, “A practical introduction to early stopping in machine learning,” Medium, https://towardsdatascience.com/a-practical-introduction-to-early-stopping-in-machine-learning-550ac88bc8fd (accessed Jun. 25, 2023).