Report: Traffic Sign CNNs

Team: Jierui Zhang (519, Siyuan's cohort), Ricardo Del Rio (519, Chang's cohort), Matthew Krisch (519, Halley's cohort)
Project Mentor TA: Siyuan Tian

Abstract

Self-driving cars will need highly accurate models to succeed on the roads. And one specific type of classification model needed is one that reads traffic signs correctly. This project includes multiple traffic sign convolutional neural network models trained using the GTSRB dataset. The top performing model makes classifications with 99% accuracy on the training data and 96% accuracy on the test data. We also experimented with preprocessing methods PCA and segmentation to see the impact of data transformation on model accuracy. The best performing model with PCA had 98% accuracy on training data and 79% accuracy on test data.

Our conclusion from this project is that in terms of traffic signs, self-driving cars are coming closer to operational based on reading its environment, but using image processing techniques can impact accuracy depending on different factors.

Github: https://github.com/rdelriog/image_classification_for_traffic_signs_GTSRB_PCA_segmentation

Introduction

For a self-driving car to properly work and drive on the road, the car must be able to correctly read traffic signs it's cameras find. If the car cannot do that, it puts people's lives at risk. Also, according to Evtimov, Eykholt et al [2], traffic sign models themselves have been susceptible to problems when small changes take place. We wanted to try to create our own model: a convolution neural network that accurately classifies traffic signs, while giving us the chance to also experiment and analyze the impact of preprocessing methods.

Related Prior Work

German Traffic Sign Recognition Benchmark [13]: This was part of a competition from the early 2010s and had teams write machine learning models on traffic signs. The competition and its models are now almost ten years old, so by using updated technology and techniques, we can see how our models measure up against some of the top performing models including Sermanet and Lecun [5] and Yadav [4], who implemented models with over 98% accuracy using convolutional neural networks. There is an organized structure of images on the GTSRB website which allows people interested in the dataset to quickly import images and run models on them.

Laboratory for Intelligent & Safe Automobiles (LISA) and the LISA Dataset (http://cvrr.ucsd.edu/LISA/lisa-traffic-sign-dataset.html) [9]: LISA dataset has over 7GB of images of signs for training and testing. Papers such as *Robust Physical-World Attacks Learning Models* [2], show that problematic training images can impact the results of a model in a huge way. Though we ended up using the GTSRB dataset, this paper and model using the LISA dataset show that there are remaining questions for these models for vehicles as they continue being developed and put into production.

***BadNets: Identifying Vulnerabilities in the Machine Learning Model Supply Chain* [10]:** Focusing on how small changes in how real-life sign displays can impact a model's accuracy, *BadNets* highlights that any model of signs needs to be comprehensive enough to account for signs that may not be perfectly in view or where the text may not be 100% visible. Using some test data on our own, we potentially see some of that impact in the results.

Formal Problem Setup (T, E, P)

T (Task)

We created convolutional neural networks to train on what type of sign the traffic sign is among various pictures of traffic signs.

E (Experience)

We originally were considering the LISA dataset as well, but found the GTSRB dataset to be more appropriate and manageable for how we were going to process and use the data (see Appendix 1 for more information). We used images that were resized to 32x32 before using them for training. In addition, we experimented with preprocessing using PCA and segmentation using KMeans and the Canny algorithm for edge detection [19].

P (Performance)

We analyzed the performance of our models through the different methods using accuracy and looking at the loss on training, validation, and testing.

Methods

Prior methods done

One previous method done by Lecun and Sermanet converted images from RGB to grayscale and normalized the data [4,5]. In the model, LeCun and Sermanet used two convolutional layers followed by a linear layer. In our model, we experimented with different methods of preprocessing and different hyperparameters. Then, we wanted to see what our model misclassified and what trends we could uncover about our model.

Our group's implementation details

After gathering all the data, we resized all the images to 32x32 to ensure consistency and so we could run all images through the network without much issue. As noted in [15], 32 by 32 was the minimal size of an outdoor image to be able to be recognized correctly, also, we wanted to minimize the size of the images input so that we could run more experiments in not such a long time, considering we were training on more than 39K images. The below illustrations and tables show the details of one of our most accurate networks, CNN4.

Learning Rate	Criterion	Optimizer	Weight Decay
0.001	Cross entropy loss	Adam	0.035

Model Architecture of CNN4

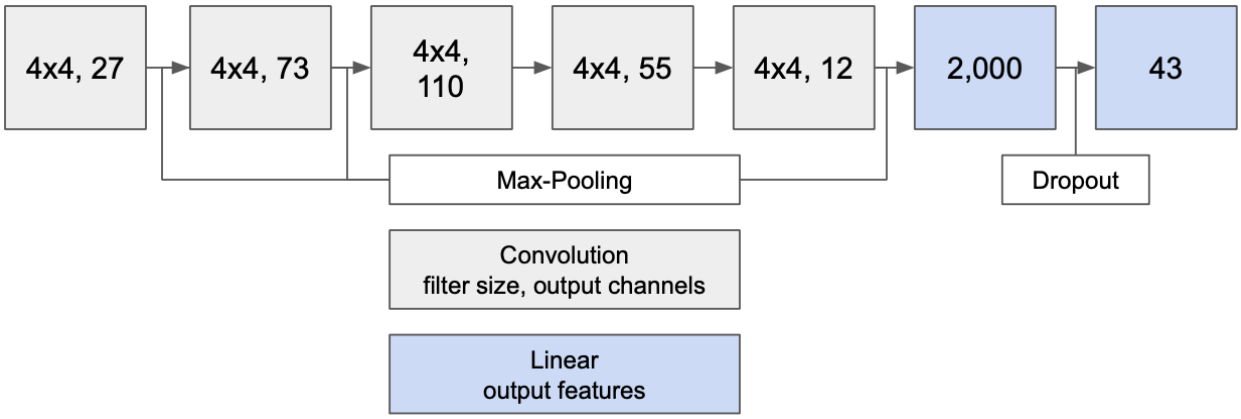


Diagram of the CNN4 neural network architecture

Layer	Type	After: Activation, Pooling, Dropout, Reshape	Detail
1	Convolution	ReLU, Max-Pooling	Filter size: 4x4
2	Convolution	ReLU, Max-Pooling	Filter size: 4x4
3	Convolution	ReLU	Filter size: 4x4
4	Convolution	ReLU	Filter size: 4x4
5	Convolution	ReLU, Max-Pooling	Filter size: 4x4
6	Linear	ReLU, Dropout (0.5)	Input Features: 3,072; Output: 2,000
7	Linear	ReLU	Input Features: 2,000; Output: 43

The image above is the architecture of our top performing model. We used a convolutional neural network with 7 layers, mostly using ReLU for activation and using max-pooling and dropout.

In training the model we experimented with adjusting batch count, learning rate, pooling, and dropout through trial and error. But most impactful once a lot of layers were made was not initializing weights on the model, we hypothesize that this is due to the fact that the random standard initialization of weights makes the same impact as a uniform or Xavier initialization when a model is complex and a local minimum is enough to capture information about the dataset .

During our model preparation, we wanted to see if we could capture patterns being picked up through intermediate layers [16], and to see if we could use a YOLO (You-Only-Look-Once) network [7, 8]. We faced issues with the YOLO network and focused on completing a CNN model. Please see Appendix 1 for more information on the problems we encountered.

Experimental Results

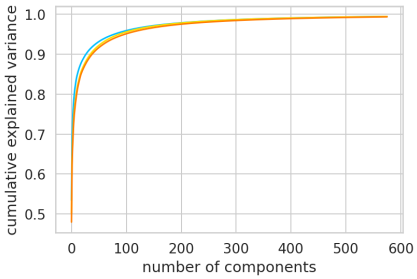
Experiment questions and goals

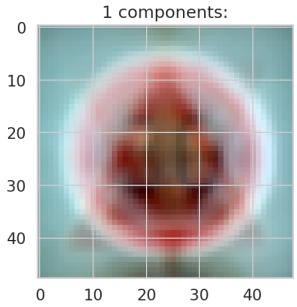
First, we hoped the experiments would lead us to creating a nearly perfectly accurate model. We wanted to see: could we create a model that was as accurate as some from the GTSRB competition [13, 17]? Also, could running a data preprocessing method on the data lead to nearly as accurate models and outputs? If so, which ones?

One last experiment we wanted to try was if the model could be used beyond the scope of the given traffic sign type. We ran tests of the model on alto signs and got over 81% test accuracy.

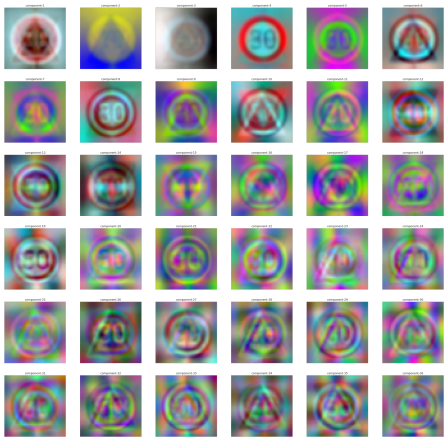
Model Experiment	**CNN #**	**Train Acc.**	**Train Loss**	**Validation Acc.**	**Validation Loss**	**Test Acc.**	**Test Loss**
13 (with PCA)	CNN-PCA	0.983	17.709	0.928	2,539.526	0.7886	16,981.369
6	**CNN4**	**0.994**	**6.076**	**0.9933**	**214.76**	**0.9587**	**2,951.21**
Testing on Alto Signs						0.8148	2,200

Results of applying PCA to the training dataset:





The left figure above shows the cumulative explained variance (image resize dimension = 48, flattened = 2304; number of pc (principal components kept) = 24, flattened = 576; 3 curves correspond each of the RGB channel) that goes into CNN-PCA. Observe that the first PC alone explains 60% ~ 70% of total variance. The right figure is the displayed result of the first PC, this tells us that some information that roughly distinguishes traffic sign images from each other may include: shape of the sign, color around the edge of the sign, the content in the middle, etc.



First 36 principal components, in the order from left to right (account for ~90% of cumulative variance)

As we display more principal components, it shows that the algorithm starts to capture some less relevant features that are more difficult for humans to perceive and understand such as the lighting condition in the background and color patterns of the image. (Appendix 4: result of reconstructing an image using PC)

The use of PCA in this project is mainly for exploratory purposes and to help us understand the traffic sign image structures better. Since the convolutional layers will essentially learn to manipulate the dimensions

of input images and number of channels, the benefits of using PCA here are not so evident. Nevertheless, we tried to feed in PCA-transformed input data (with >95% of total variance kept) into few CNN structures, and the test accuracies were lower than expected (CNN-PCA). The issue of low test accuracy was not investigated further in the project, but in the future could be examined and improved upon.

Conclusions and Future Work

Conclusions

Using at least a few convolutional layers can make a very accurate convolutional neural network. Plus, we saw in our experiment with alto signs that the model could somewhat be generalized, but it is still over 17% below the highest performing models in our experiments. Going forward if we want to create a model that is generalized across countries and roadways, we'll have to incorporate more images in the training from those respective countries, or different models should be used for different countries.

Future Work

In the future it would be really interesting to pursue building a model that can preprocess the images to smaller levels to decrease both the training time and the operational time. In practice these models will be operating in real-time, so to have a model operate as quick as real-time would actually be useful for autonomous vehicles. We can also further explore the use of other segmentation and dimensionality reduction methods to achieve high accuracy and low training time.

Ethical Considerations and Broader Impacts

Directly related to the model we created, there are ethical considerations related to how the data is gathered and generated. For our model, we used publicly available data and signs. But other models could use private data, or use "manipulated" data as noted from Evtimov, Eykholt et al [2].

Furthermore, we wanted a model that could actively read signs for autonomous vehicles. To do that, cars and people using the model will have to use cameras to capture the images. Having many cameras around constantly taking photos and video could capture unintended images in addition to signs. This has consequences including if a picture is taken of someone or something without permission.

Also, even the top models are 99% accurate, so there's still a 1% that needs to be figured out in order to make the right decisions when it comes to possibly avoiding a fatal accident.

Prior Work / References

[1] Prior to the second proposal, we talked with Siyuan and concluded we should come up with our own model on traffic sign data.

[2] Evtimov, I., Eykholt, K., Fernandes, E., Kohno, T., Li, B., Prakash, A., ... & Song, D. (2017). Robust physical-world attacks on machine learning models. arXiv preprint arXiv:1707.08945, 2(3), 4. Code in https://github.com/evtimovi/robust_physical_perturbations

[3] Sebastian Houben, Johannes Stallkamp, Jan Salmen, Marc Schlipsing, & Christian Igel (2013). Detection of Traffic Signs in Real-World Images: The German Traffic Sign Detection Benchmark. In International Joint Conference on Neural Networks.

[4] https://github.com/vxy10/p2-TrafficSigns

[5] P. Sermanet and Y. LeCun, "Traffic sign recognition with multi-scale Convolutional Networks," The 2011 International Joint Conference on Neural Networks, San Jose, CA, USA, 2011, pp. 2809-2813, doi: 10.1109/IJCNN.2011.6033589.
[6] https://github.com/haithamkhedr/Traffic-sign-classification (the code related to [5])
[7] https://github.com/chien-lung/PyTorch-YOLOv3
[8] Redmon, J., & Farhadi, A. (2018). YOLOv3: An Incremental Improvement. https://arxiv.org/abs/1804.02767
[9] Laboratory for Intelligent & Safe Automobiles, http://cvrr.ucsd.edu/LISA/lisa-traffic-sign-dataset.html
[10] *BadNets: Identifying Vulnerabilities in the Machine Learning Model Supply Chain* by Tianyu Gu, et al (https://arxiv.org/abs/1708.06733)
[11] https://cocodataset.org/#home
[12] Originally tried to use this website, http://alexlenail.me/NN-SVG/AlexNet.html, to build a diagram of the architecture, but found Google Slides instead to be easier to use.
[13] Used this website for the image data used in the models: *German Traffic Signs Recognition Benchmark*. https://benchmark.ini.rub.de/gtsrb_news.html
[14] https://arxiv.org/pdf/1404.1100.pdf
[15] Yang Cai. (2003). How Many Pixels Do We Need to See Things https://link.springer.com/content/pdf/10.1007/3-540-44863-2_105.pdf
[16] To try to view patterns in CNN layers, we used *Visualizing Convolutional Neural Networks Using Pytorch* by Niranjan Kumar https://towardsdatascience.com/visualizing-convolution-neural-networks-using-pytorch-3dfa8443e74e
[17] GTSRB Top Performing Model Rankings https://benchmark.ini.rub.de/gtsrb_results.html
[18] Because our project involves CNNs, class materials on deep learning and CNNs were helpful with the project including the lectures, worksheets, and homework 4's template.
[19] J. Canny, "A Computational Approach to Edge Detection," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. PAMI-8, no. 6, pp. 679-698, Nov. 1986, doi: 10.1109/TPAMI.1986.4767851.

Supplementary Materials

Appendix 1: Attempted methods and data

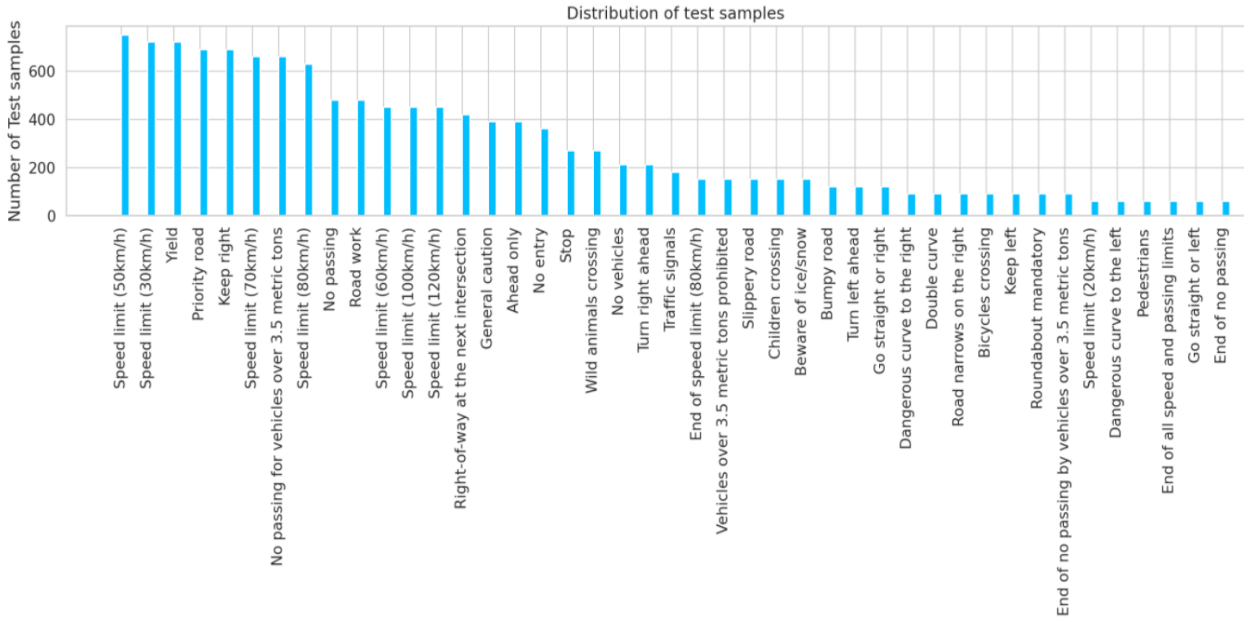
YOLO Network [8]
We decided to not move forward with the YOLO implementation we had because the model included many different modules that would need to be assembled. Also, we didn't see a feasible way to incorporate segmentations and dimensionality reductions in it.

Using GTSRB vs the LISA Dataset
If using LISA dataset, since the images are not cropped to only include the traffic sign, we would have needed to complete both Image detection and image recognition tasks. The former is a task that is harder for us to accomplish and is more beyond the scope of the lectures.

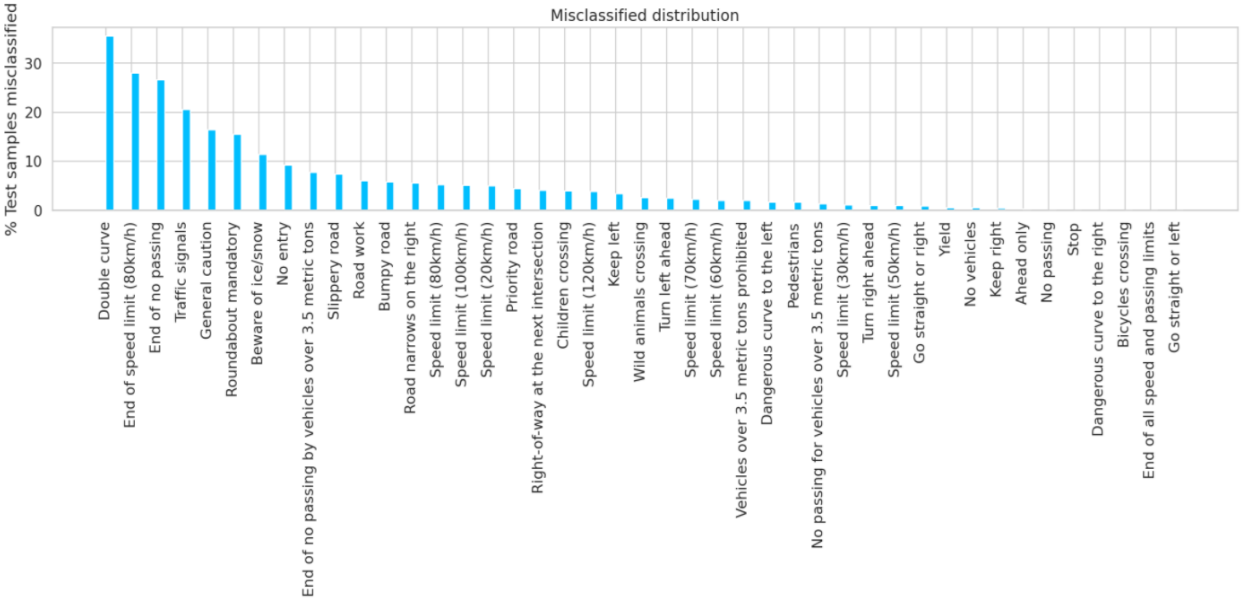
Caveat of using PCA:
Since PCA applies a linear transformation (rotating and stretching) to the standard basis of D-dimensional space (where D is the number of features of data) to find the directions of maximum variance, it works the best when directions of maximum variance form an orthogonal basis. Such assumption of linearity also

implies that PCA removes second-order dependencies in the data. Other techniques such as kernel PCA and ICA may be applied in the cases where higher-order dependencies exist in the data [14].

Appendix 2: Distribution of classes in GTSRB test set and Misclassification in CNN4



Number of images per class in GTSRB dataset



Misclassified images according to CNN 4

Appendix 3: Additional Model Experiment Results

For the below table on experiment results, we programmed several CNN models where we changed different parameters and hyperparameters for each experiment. The CNN # refers to the CNN in the

notebook. A few times, experiments were done with a variation of the main CNN models and is noted as "Other".

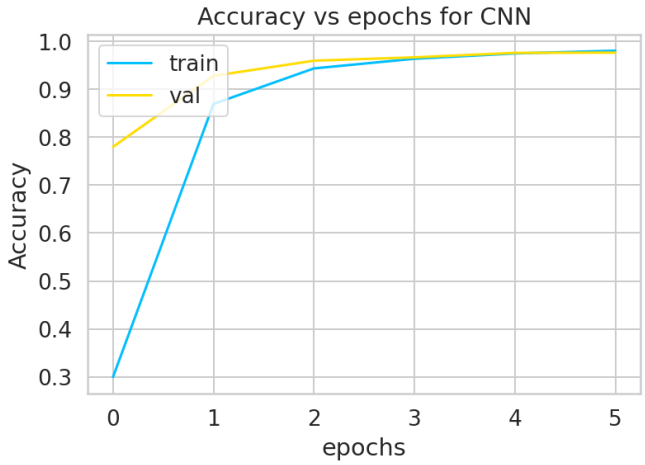
	Image Changes	CNN #	Parameters	Learning Rate	Epochs	Train Acc.	Validation Acc.	Test Acc.
1	None	1	1,203,792	0.0001	5	0.975	0.978	
2	None (initialized weights)	2	12,732,992	0.0001	5	0.411	0.4089	
3	None	1	1,203,792	0.0001	6	0.977	0.9764	
4	None	3	3,384,992	0.0001	10	0.984	0.989	
5	PCA (20 Components)	Other	2,988,829	0.001	15	0.979	0.953	
6	**None**	**4**	**6,500,992**	**0.001**	**18**	**0.994**	**0.9933**	**0.9587**
7	KMeans (20)	Other	6,500,128	0.001	9	0.728	0.72018	
8	None	4	6,500,992	0.001	25	0.995	0.9934	0.9576
9	KMeans (20)	Other	6,500,128	0.001	18	0.838	0.76	0.65
10	None	4	6,500,992	0.001	19	0.994	0.9917	0.9569
11	Edge (0.25)	Other	6,500,128	0.001	18	0.678		0.5954
12	PCA (20)	Other	22,249,565	0.001	5	0.966	0.877	0.6908
13	**Resize to 48x48, PCA (24)**	**CNN-PCA**	**7,788,521**	**0.001**	**10**	**0.983**	**0.928**	**0.7886**
14	**Alto Sign**	**4**						**0.8148**

Unless otherwise noted, all experiments re-sized the images to 32x32. Experiments in bold are discussed in the Results section.

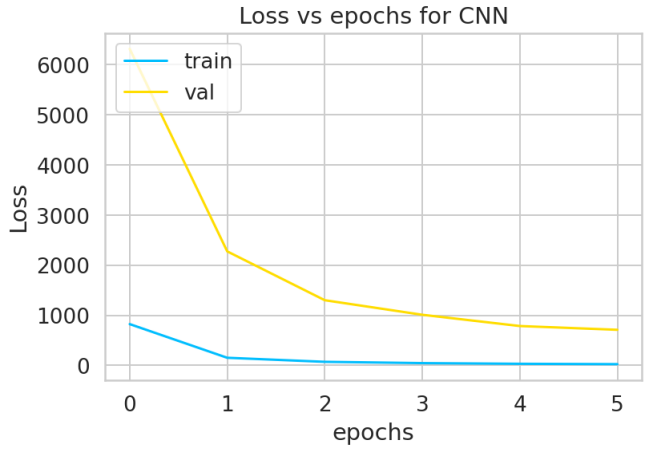
Appendix 4: Reconstruction of an image using first 300 pc, step = 10 (number of pc increase from left to right)

Appendix 5: Chart of epochs over time

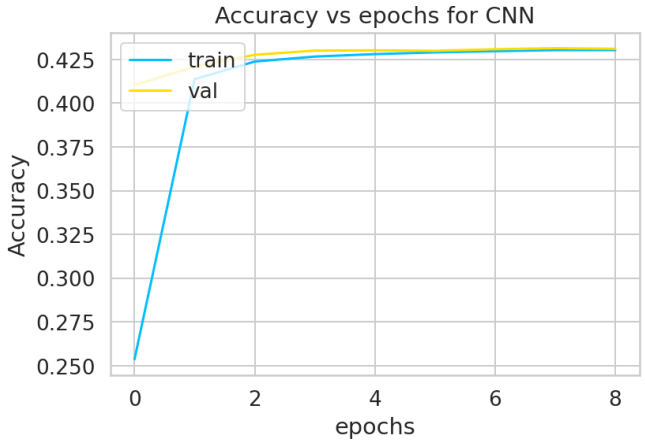
The below charts were made from separate training and validating runs of the CNNs to illustrate that there was no overfitting of the images during training.



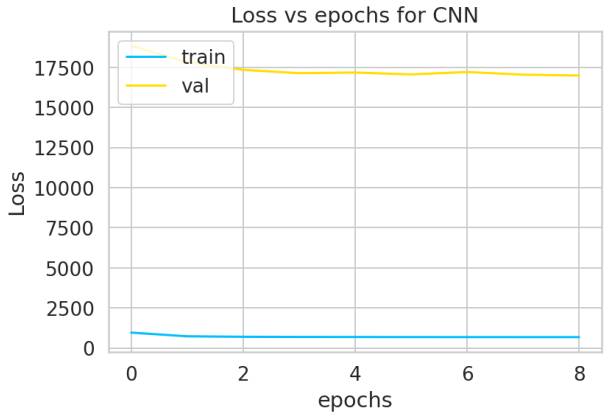
CNN1 accuracy over time



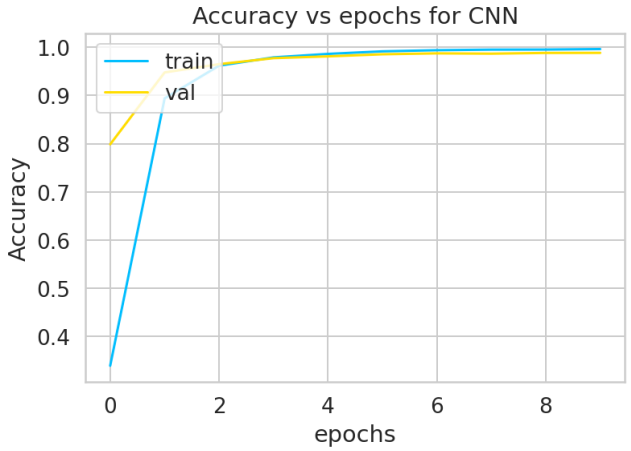
CNN1 loss over time



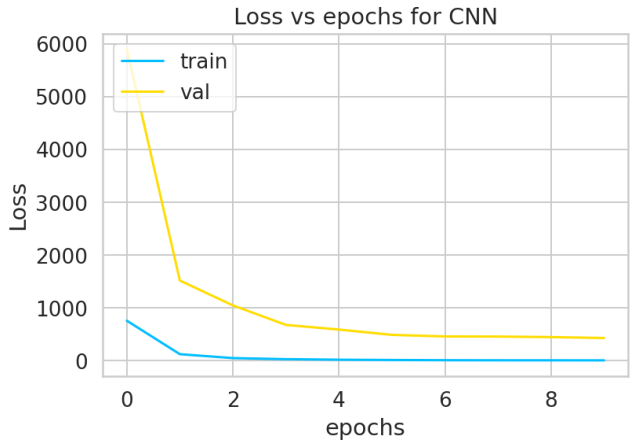
CNN2 accuracy over time



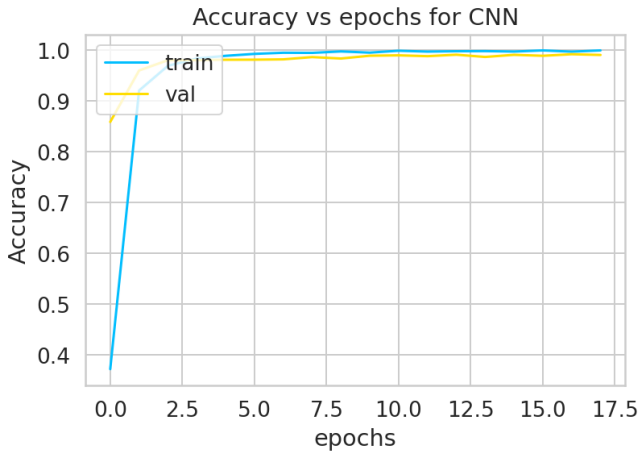
CNN2 loss over time



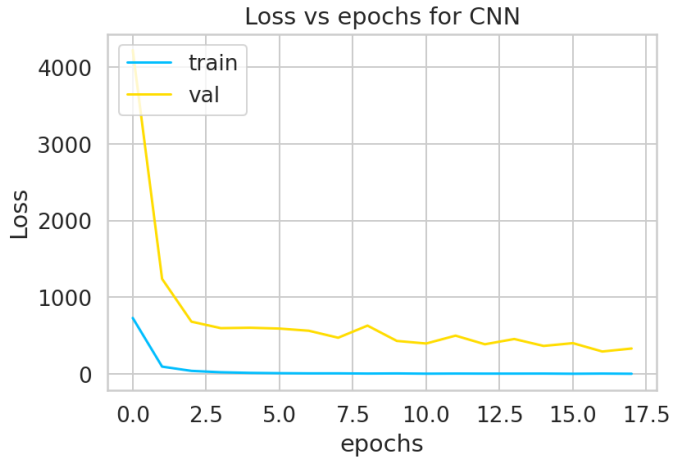
CNN3 accuracy over time



CNN3 loss over time



CNN4 accuracy over time



CNN4 loss over time