I. Definition

Project Overview

Image classification is prevalent today and is used everywhere from image search to pedestrian detection. In this project, two machine learning algorithms—a support vector machine and convolutional neural network—are trained, tested, and compared based on how many traffic signs they can classify correctly on the German Traffic Sign Dataset, a large dataset containing many images of 43 different traffic signs.

Problem Statement

The objective of this project is to compare the performance of a support vector machine versus a convolutional neural network at classifying images of traffic signs. These are the steps:

1. Download the German Traffic Sign Dataset.
2. Preprocess data.
3. Synthesize additional data.
4. Train and tune parameters of support vector machine and convolutional neural network.
5. Test support vector machine and convolutional neural network classifiers.
6. Analyze results.

It is expected that the convolutional neural network outperforms the support vector machine at image classification because convolutional neural networks assume that its inputs are images, enabling its architecture to take advantage of the pixel layout of images [2]. Also, the convolutional neural network achieves state-of-the-art performance on the German Traffic Sign Dataset [1].

Metrics

Each classifier's performance will be measured by its accuracy score on the test set. This is a suitable metric for this project because this is a classification problem—target labels are discrete rather than continuous labels. Note that the accuracy score is calculated as follows:

accuracy score = (number of correctly classified images) / (total number of images)

Note that other classification metrics exist, such as precision, recall, and F1 score. As a reminder, these are the formulas:

precision = (number of true positives) / (number of positives)
recall = (number of true positives) / (number of true positives + number of false negatives)
*F1 = (2 * precision * recall) / (precision + recall)*

While these may be good choices for some binary classification datasets, these are not good choices for this project as the German Sign Dataset contains 43 classes as opposed to two (positive/negative). However, there are cases when precision, recall, and F1 may be relevant to this problem. Suppose that we cared about which classifier classifies safety traffic signs (e.g. stop, yield, do not enter) the best, and we argue hypothetically that false positives are worse than false negatives. Then, we can set all safety signs to a label of '1' and the rest to '0' and use precision, recall, and F1 score to find the best classifier. However, because we care only about how well the SVM and ConvNet can classify 43 classes, false positives and false negatives are irrelevant for this project and accuracy score is used.

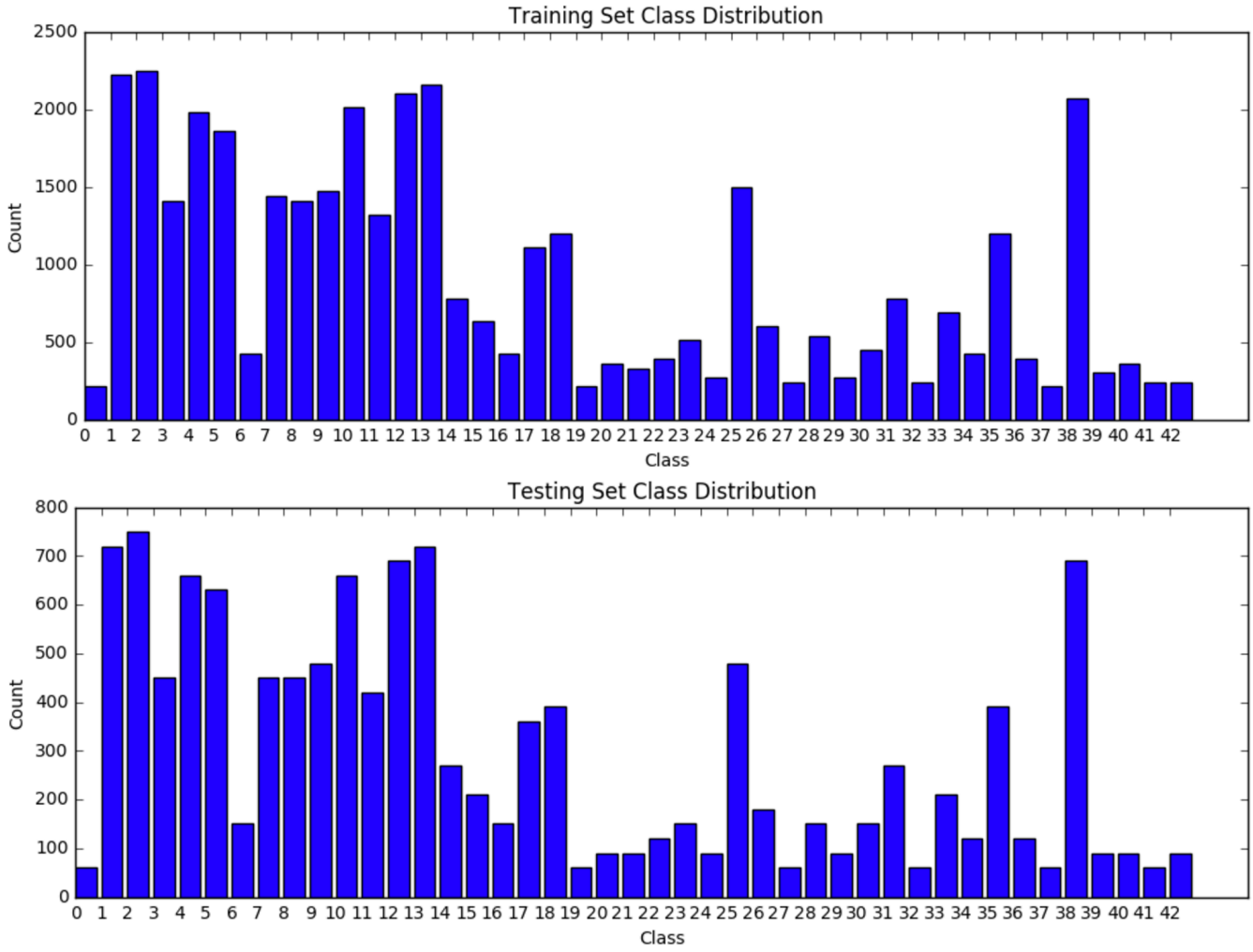
For these reasons, accuracy score is the chosen metric and, therefore, the classifier with the higher accuracy score will be deemed the better classifier.

II. Analysis

Data Exploration

The dataset is pre-split into 39209 training examples and 12630 testing examples, which is approximately a 76:24 split. The dimensions of a single image are 32x32x3, corresponding to its width, height, and RGB values, respectively. There are 43 classes in this dataset, each representing a different traffic sign.

The pre-split training and test sets share approximately the same class distribution, as shown in the following graphs:



Note that some traffic images occur more or less frequently than others. For example, classes 2, 13, and 38 occur the most whereas classes 0, 19, and 27 occur the least.

Shown below is one image per class:

Algorithms and Techniques

Machine learning algorithms used for this image classification project are the support vector machine (SVM) and convolutional neural network of LeNet-5 architecture. There are a few pros and cons for each of these classifiers.

Advantages of SVMs are they are effective in high dimensional spaces and when the number of dimensions is greater than the number of samples, and they are memory efficient as SVMs use a subset of training points in their decision function. A disadvantage is that the SVM is likely to give poor performances when the number of features is much greater than the number of samples [5].

These are all reasons why the SVM is a suitable classifier for this dataset. As a reminder, this dataset contains 1024 features after grayscaling and 62698 training samples after data synthesis. As we can see, the number of training samples is larger than the number of features, making it likely that the SVM will perform well. Also, this is a large dataset where the SVM's memory efficiency can be useful.

The advantage of convolutional neural networks is their very high accuracy on image classification tasks. Disadvantages of ConvNets are high computation cost, a need for a strong GPU, requirement a lot of training data, and inability to train with unlabeled data [6].

Because this large dataset contains 62698 labeled training examples, the ConvNet is a suitable classier for this project. Also, ConvNets of LeNet-5 architecture have produced state-of-the-art results on the German Traffic Sign Dataset [1][2].

Techniques for optimizing these machine learning algorithms included data synthesize, preprocessing, tuning classifiers' parameters, and trial and error.

To improve test set accuracy, data synthesis and preprocessing are necessary steps. Extra training data was synthesized by flipping symmetrical traffic signs. For example, a yield sign (class 13) is symmetrical along its vertical axis and, therefore, an image of a yield sign can be flipped horizontally to synthesize another image of a yield sign. The images were preprocessed by converting to grayscale and normalizing the feature values to 0 mean and unit variance. Because the pixel values range from 0 to 255, normalizing feature f is done simply by the following operation: $(f - 128) / 128$ [2].

Note that while converting to grayscale is unnecessary, reducing the number of features by two thirds significantly reduced the cost of training on 62698 images (after data synthesis). Tuning multiple parameters on 62698 RGB images is computationally expensive for my machine. Also, grayscale traffic signs produce higher test set results than their RGB counterparts for this dataset [1].

Before training a machine learning algorithm, it is important to first split the training data into training and validation sets because tuning a learning algorithm's parameters on the test set may cause us to slowly overfit our test set, leading to an overly optimistic generalization performance [4]. To get around this, 24% of the training set was reserved as the validation set, which was used to tune the classifiers' parameters. The classifiers' parameters that were tuned are the following:

Support Vector Machine:
- C
  - trades off misclassification of training examples against simplicity of the decision surface
- gamma
  - how far the influence of a single training example reaches

Convolutional Neural Network:
- number of epochs
  - number of passes through the training set
- batch size
  - number of training examples per training step

Benchmark

The benchmark for classification accuracy on the German Traffic Sign test set is 98.81%, which is human performance by Team INI-RTCV for GTSRB Competition Phase I [3]. Note that the accuracy score is calculated as follows:

accuracy score = (number of correctly classified images) / (total number of images)

Note that the value was obtained from the "Man vs. Computer: Benchmarking Machine Learning Algorithms for Traffic Sign Recognition" research paper [3].

III. Methodology

Data Preprocessing

These are the data preprocessing steps:

1. Grayscale images
2. Normalize features to zero mean and unit variance
3. Synthesize data
4. Shuffle and reserve 24% of training data for the validation set
5. Flatten training and testing images for support vector classifier

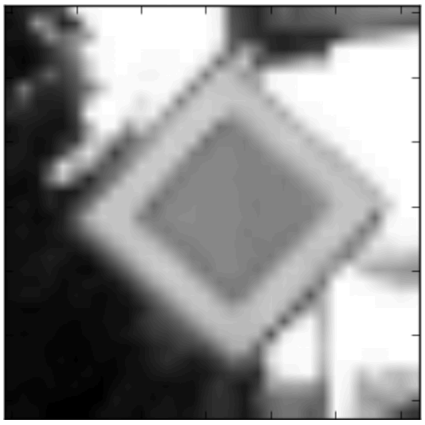
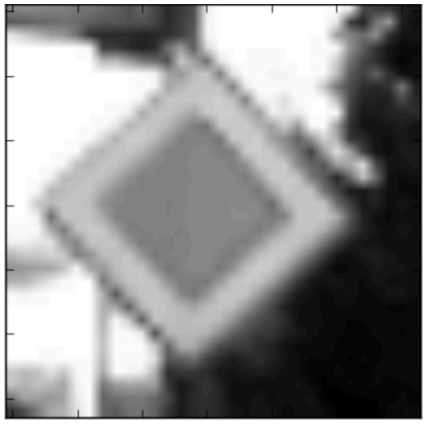
The first step of data preprocessing was gray scaling the images. Note that while converting to grayscale is not entirely necessary, reducing the number of features by two thirds significantly reduced the cost of training on 62698 images (after data synthesis). Tuning multiple parameters on 62698 RGB images is computationally expensive for my machine. Also, grayscale images produce higher classification accuracy on the German Traffic Sign Dataset than their RGB counterparts [1]. This is an example of an image converted from RGB to grayscale:

RGB Image

Grayscale Image

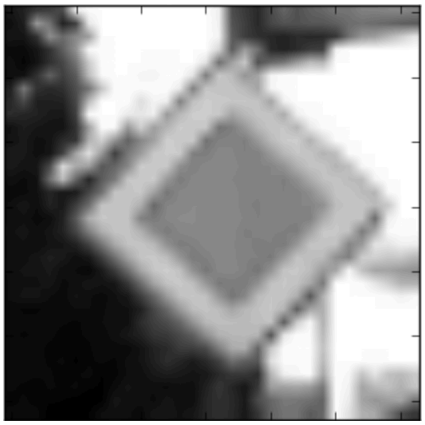
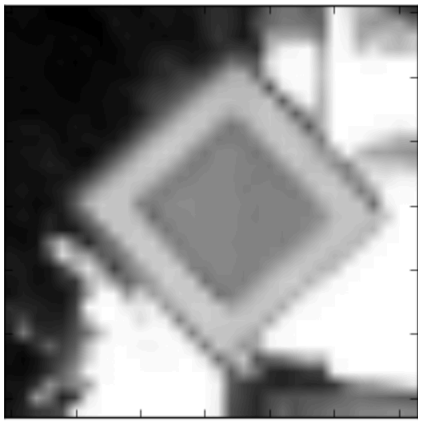
The second step of data preprocessing was synthesizing data. This is an important step because more training data leads to better test results. Some traffic images are symmetrical along their horizontal or vertical axis. For example, the following traffic sign is symmetrical along its vertical axis and was flipped horizontally to generate an extra training image:

Original Image **Synthesized Image**

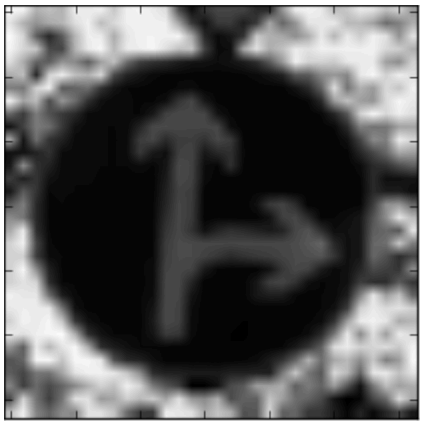
Similarly, because this traffic sign is also symmetrical along its horizontal axis, an extra training image can be synthesized by flipping it vertically:

Original Image **Synthesized Image**

Interestingly, some classes flipped horizontally turn into other classes. For example, the following image of Class 37 was flipped horizontally to synthesize an image of Class 36:

Original Image - Class 37 **Synthesized Image - Class 36**

With data synthesis, a total of 23489 new training examples were generated. This brought the number of training examples from 39209 to 62698, a 59.91% increase.

The third step was to standardize the features to zero mean and unit variance. This not only helps gradient descent converge faster, but it also reduces numerical instability by ensuring that values never get too large or too small. Because the pixel values range from 0 to 255, normalizing feature *f* is done simply by the following operation: (*f* - 128) / 128 [2].

The fourth step of data preprocessing was to shuffle and reserve 24% of the training data as a validation set with the same class distribution. Note that I chose 24% because the training and test data were pre-split at approximately a 76:24 ratio. This is important because tuning the classifiers' parameters based on its performance on the validation set prevents us from overfitting the test set. Also, setting the test set aside until the end ensures that the classifier's generalization performance is accurate rather than overly optimistic [4].

The final step of data preprocessing was to flatten the training and testing images into a single dimension for the support vector classifier.

Implementation

This project was implemented in a Jupyter Notebook in the following order:

1. Open the German Traffic Sign Dataset.
2. Explore and visualize the data.
3. Preprocess data.
4. Synthesize additional data.
5. Train and tune parameters of support vector machine and convolutional neural network.
6. Test support vector machine and convolutional neural network classifiers.

The first step of implementation was to open the pickled files using the `pickle` API. Both the training and testing data (pre-split 76:24 ratio) were stored as `numpy` arrays.

The second step of implementation was to visualize the data. The length and shape of the training and testing sets were easily found by Python's built-in `len` function, as well as printing the `numpy` arrays' `shape` attribute.

The third step of implementation was to preprocess the data. Converting the images from RGB to grayscale was easily done with `cv2`'s `cvtColor` function. Because pixels values range from 0 to 255, normalizing feature *f* to zero mean and unit variance was done by the following operation: (*f* - 128) / 128. For the SVM, the training and testing data were cloned and flattened using `numpy`'s `copy` and `reshape` operations, respectively. 24% of the training data was reserve for the validation set using `sklearn`'s `train_test_split` function.

The fourth step of implementation was data synthesis. This was implemented by observing which traffic images were symmetrical, and flipping them using `numpy`'s `fliplr` and `flipud` functions. The synthesized data was combined with the original data using `numpy`'s `concatenate` function.

The final step of implementation was training and testing the SVM and convolutional neural network on the German Traffic Sign Dataset. The SVM was implemented using `sklearn`'s `SVC` class, and the accuracy score on the validation and test sets were calculated using `sklearn`'s `accuracy_score` operation. The SVM's parameters were tuned by training and testing 9 SVCs—each with different `C` and `gamma` parameters—on the validation set; an SVC with the best `C` and `gamma` parameters were then trained over the entire training set and tested on the test set. The convolutional neural network of LeNet-5 architecture was implemented entirely with `tensorflow` and has the following layers [2]:

- Convolutional Layer 1. Output shape is 28x28x6.
- ReLU Activation 1.
- Pooling Layer 1. Output shape is 14x14x6.
- Convolutional Layer 2. Output shape is 10x10x16.
- ReLU Activation 2.
- Pooling Layer 2. Output shape is 5x5x16.
- Flatten Layer.
- Fully Connected Layer 1. Number of outputs is 120.
- ReLU Activation 3.
- Fully Connect Layer 2 (Logits). Number of outputs is 43.

The ConvNet's learning rate is set to 0.001, its loss operation is cross entropy with logits, and its optimizer is the Adam Optimizer. Similar to the SVM, the batch and epoch sizes were tuned by finding the batch and epoch sizes that maximized classification accuracy on the validation set.

The biggest complication arose when trying to preprocess the images with `cv2`'s Gaussian Blur and Canny Edge Detection algorithms. While this is a good idea in theory as it can potentially remove background noise such as trees, sky, and other irrelevant objects that are not traffic signs, as well as emphasize edges to improve classification accuracy, the edge detection algorithm was not clean in practice. For example, the numbers on the speed limit signs were not readable despite trying various function parameters. Because of this complication, only grayscale and mean/variance normalization was implemented when preprocessing the images.

Refinement

Parameters for both the SVM and convolutional neural network were refined based on their classification accuracies on the validation set.

With the SVM's default parameters values—`C` = 1 and `gamma` = 'auto'—the SVM achieved 77.55% classification accuracy on the validation. For clarity, 'auto' is simply 1 / (number of features) so, with 32x32 images, 'auto' is 1 / 1024. Through trial and error via grid-search, the SVM's parameters were tuned to `C` = 1000 and `gamma` = 'auto', achieving 96.30% classification accuracy on the validation set and 80.60% classification accuracy on the test set. Notable intermediate results occurred when `gamma` = 0.1, averaging just 54.05% on the validation set for `C` = 10, `C` = 100, and `C` = 1000.

The ConvNet's learning rate was held at a constant 0.001, because this value works well in practice [2]. The major two parameters to tune were the batch and epoch sizes. Like the SVM's parameters, these were found by trial and error via grid-search on the validation set. The worst validation accuracy of 98.65% occurred with batch and epoch sizes of 20 and 40, respectively, while the best validation accuracy of 99.30% occurred with batch and epoch sizes of 80 and 80, respectively. Training for 80 epochs with a batch size of 80 on entire training set led to a test set accuracy of 92.70%.

IV. Results

Model Evaluation and Validation

As expected, the ConvNet classified more accurately than the SVM. The final parameters for the ConvNet and SVM are appropriate as they were found by maximizing validation set accuracy through trial and error, and the models are robust as different classification sets produced similar results. These are the classifiers' classification accuracies on the validation set with various parameter values:

Support Vector Machine

		Gamma		
		'auto'	0.01	0.1
C	1	0.7755	0.9060	0.5164
	10	0.9289	0.9551	0.5525
	100	0.9630	0.9587	0.5526

Convolutional Neural Network

		Number of Epochs		
		20	40	80
Batch Size	20	0.9879	0.9865	0.9875
	40	0.9871	0.9875	0.9924
	80	0.9875	0.9889	0.9930

While the final ConvNet generalizes reasonably well to unseen data with a test set accuracy of 92.70%, the SVM does not generalize as well as its test set accuracy is 80.60%. Note that it is safe to assume these generalization accuracies are not overfitting the test set as these classifiers' parameters were tuned on validation sets and the test set was not used until the final testing phase. For this reason, these results can be well-trusted [4].

Justification

While the ConvNet produced higher classification accuracy on the test set than the SVM, neither classifier met the benchmark of human performance. There are two possible causes for the gap between validation performance and test set performance for both classifiers: (1) the classifiers overfit the validation set, or (2) the test set is harder than the training set. If the first reason is true, L2-regularization and more rigorous parameter tuning using k-fold cross-validation can minimize this [2]. If the second reason is true—which I suspect given that this dataset was used for competition [1]—then more rigorous data preprocessing is necessary such as adding noise to images.

It should be pointed out that Team IDSIA in CTSRB Competition Phase I designed a ConvNet that produced 98.98% on the German Traffic Sign test set [3]. For this reason, the convolutional neural network trained in this project is not ideal for classifying traffic sign images, although it has potential to do so with additional tuning and data preprocessing.

V. Conclusion

Free-Form Visualization

As a recap, these were the SVM's and ConvNet's classification accuracies on the validation set with various parameter values, as well as their test set results:

Support Vector Machine

		Gamma		
		'auto'	**0.01**	**0.1**
C	**1**	0.7755	0.9060	0.5164
	10	0.9289	0.9551	0.5525
	100	0.9630	0.9587	0.5526

Test Set Accuracy: 0.8060

Convolutional Neural Network

		Number of Epochs		
		20	**40**	**80**
Batch Size	**20**	0.9879	0.9865	0.9875
	40	0.9871	0.9875	0.9924
	80	0.9875	0.9889	0.9930

Test Set Accuracy: 0.9270

Reflection

This project was heavily involved. Before this project, the most complex dataset I trained a machine learning algorithm was that of *Project 2: Student Intervention System*—396 examples with 30 features and 2 classes. For this project, there were over 50,000 examples with 3072 features and 43 classes. Additionally, I had to learn the `tensorflow` and `cv2` APIs, neither of which were covered in the Machine Learning Engineer Nanodegree.

By far the most challenging part of this project was the data preprocessing. After reading through the research papers that were referenced, it was clear that data preprocessing techniques took the most amount of trial and error. I initially preprocessed the images with Gaussian Blur and Canny Edge detection algorithms after gray-scaling the images, but Canny Edge detection was not clean enough to use despite trying different parameters (e.g. low and high threshold); this may have been due to inconsistent lighting in the images as well as background noise such as trees. Also, it was unclear which image space to use. While the images were originally RGB, I decided to convert to Grayscale because not only performed the best during the GTSRB competition, but also reduced the number of features by two-thirds which was useful for my limited processing power [1]. Moreover, the images had to be normalized to decrease training time and reduce numerical instability [2]. Initially, I used `sklearn`'s standard scaler to normalize all features to zero mean and unit variance, but learned from Udacity's deep learning course that images can simply be preprocessed by executing the following operation on each feature *f*: (*f* - 128) / 128. In the end, this change made the code shorter and simpler [2]. Lastly, the extra training data was flattened for the support vector classifier which was simple thanks to `numpy`'s built-in `reshape` function.

Following data preprocessing, extra data was synthesized. This part was not as difficult as preprocessing the data as `numpy` conveniently provided methods for flipping images horizontally and vertically. However, the insight that extra data could be synthesized based on symmetry did not occur to me until I was about to train my first SVM.

Training the SVM was simple, thanks to `sklearn`. This was done by simply fitting the training data and predicting the testing data. Searching for optimal `C` and `gamma` parameters turned out to be more tedious than challenging. Training nine SVMs over the training set and predicting the validation set took quite a bit of time.

Training the ConvNet was more challenging than training the SVM. To do this, I had to learn about neural networks and `tensorflow` through Udacity's deep learning course. The ConvNet proved to be more complex than the SVM, as the activation function, optimizer, loss function, architecture, learning rate, batch size, and epoch size all had to be considered [2]. Also, in addition to this complexity, I found `tensorflow` to be less intuitive than `sklearn`: using sessions and Tensor objects was foreign prior to this project. Nonetheless, I found convolutional neural networks to be powerful and interesting as they take advantage of images' pixel layouts.

In the end, the convolutional neural network outperformed the SVM as expected, as ConvNets produce state-of-the-art results on image classification tasks [1]. While I wish my convolutional neural network would have come closer to the benchmark of human performance, I have a few ideas as to how to improve my model as discussed in the next section. Nonetheless, I found this project to be an incredible learning experience as I learned about data preprocessing, deep learning, and computer vision.

Improvement

With a more powerful machine, there are a few ways to improve this study. First, a k-fold cross-validation can be run to find the optimal parameters as opposed to a single validation set. Second, more parameters can be explored. For instance, C = 1000 and C = 10000 for the SVM can be tested as well. Third, as the validation accuracy is much larger than the test set accuracy, it is possible that the models overfit the validation set during the parameter-tuning phase. This can be improved using L2-Regularization [2]. Finally, color spaces such as RGB and YUV can be explored and tested in addition to Grayscale [1].

VI. References

[1] Sermanet Pierre and Yann Lecun. "Traffic Sign Recognition with Multi-scale Convolutional Networks." The 2011 International Joint Conference on Neural Networks (2011): n. pag. Web.
[2] Vincent Vanhoucke. “Deep Learning”. Udacity. N.p., 6 June 2016. Web. 16 Dec. 2017.
[3] Stallkamp, J., M. Schlipsing, J. Salmen, and C. Igel. "Man vs. Computer: Benchmarking Machine Learning Algorithms for Traffic Sign Recognition." Neural Networks 32 (2012): 323-32. Web.
[4] Ng, Andrew. Machine Learning Yearning. N.p.: n.p., 2016. Print.
[5] "1.4. Support Vector Machines." 1.4. Support Vector Machines — Scikit-learn 0.18.1 Documentation. N.p., n.d. Web. 5 Jan. 2017.
[6] Mo, Dandan. "A Survey on Deep Learning: One Small Step toward AI." (2012): n. pag. Web. 10 Jan. 2017.