Traffic Sign Classification

William Liang
University of Pennsylvania
wjhliang@seas.upenn.edu

Vincent Cai
University of Pennsylvania
caiv@seas.upenn.edu

Yifei (Freddy) Liu
University of Pennsylvania
freddyl6@seas.upenn.edu

Abstract

Identifying traffic signs is a crucial milestone in self-driving technologies. Our work solves this problem with multiple machine learning algorithms; given the unbalanced nature of our data, we compare the macro and weighted F1 scores of classical methods against modern deep learning architectures. We find that in our filtered dataset, a convolutional neural network achieves a macro F1 score of 0.99, with simpler methods also achieving scores of over 0.90. We also investigate our methods on a smaller training dataset, in which transfer learning outperforms other strategies by a huge margin.

Source code on Github, Walkthrough in Notebook.

1. Motivation

With the development of artificial intelligence technologies in the last few decades, we have seen a large increase in the interest for self-driving vehicles, where cars are developed such that they do not need human interaction to drive. Large, multinational firms from both the technology industry, such as Google and Uber, as well the automotive industry such as Tesla and Mercedes-Benz have invested close to if not over billions of dollars in creating effective self driving vehicles. Many issues related to developing such vehicles involve identifying the surrounding environment to avoid any disruptions or crashes in the process.

For our project, we focus on the problem of traffic sign recognition. This is an important problem for self-driving vehicles as traffic signs dictate the rules you need to follow as a driver when on the road to ensure safety. Without a driver, the vehicle itself will need to be able to successfully detect and interpret these signs in order to make adjustments to its current driving state and avoid any accidents.

Machine learning is the backbone of modern day computer vision techniques, and allows for computers to interpret images of the real world. We formulate the above problem into a multi-class classification task of classifying various road traffic signs. Given an image that includes a traffic sign on the road, we train different models that identify the sign's meaning.

2. Related Work

The task of traffic sign recognition can generally be divided into two major topics: traffic sign detection and traffic sign recognition. The objective of the former is to pinpoint the location of traffic signs in its natural scene, and the objective of the latter is to classify a sign into one of several categories, given the detected image of a sign.

For detection problems, modern algorithms often take two approaches either using either region of interest (ROI) or sliding-window based techniques. In [4], a ROI based technique was used, consisting of transforming the image to gray scale to better distinguish the traffic signs image using SVM, shape matching on the gray images, and finally validating whether the ROI is a sign or not using SVM. [9] employed a sliding window technique first using a small sized window to select candidate ROIs (course filtering) which are further verified using a larger sliding window (fine filtering). HOG [3] (histogram of oriented gradients) features which can accurately represent local image regions were used in both filters of the sliding window technique and in the recognition module of the described ROI based technique to validate the images [4] [9].

For the classification problem, traditional methods used to solve this have generally been SVM based [5] [1]. Recently, convolutional neural nets (CNN) have become the main approach to solving this problem. In [8], CNN performance even surpassed that of humans reaching a near perfect accuracy of 99.46%. Multi-scale CNNs which extracts the output from multiple convolutional layers to be used for the classifier [7] and Multi-column Deep Neural Networks which trains several networks in parallel on differently pre-processed inputs [2] have only improved on this near perfect performance. Besides CNNs, random forests have also proven to yield good results [10]. Recent developments in this area have also been focused on creating more lightweight networks that can be used in resource-constrained environments where they train a much

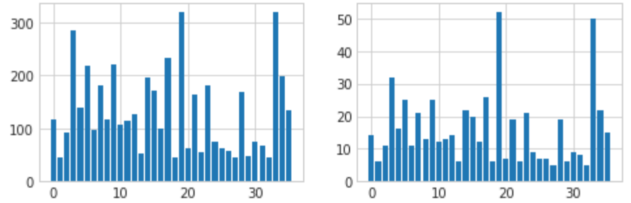

Figure 1. Class sizes in our filtered train (left) and test (right) sets.

more lightweight student-network with much less parameters from a teacher-network [11]. In most of these datasets, class skew posed a problem and was accounted for by discarding classes without enough images [8], data augmentation to increase the number of images in classes that did not have enough [12], and using class weights during training [6].

3. Dataset

3.1. Overview

Our dataset[1] consists of 6000 RGB images of varying sizes split across 54 types of signs. Each image depicts a traffic sign at the center of the image from the front in a variety of light and weather conditions. The dataset does not have any malformed images, though the class distribution is extremely skewed.

3.2. Summary Statistics

We first filter the dataset to remove extremely small classes, with some having as few as 4 images. We found that because the signs belonging to these classes shared many similarities (shapes, colors, symbols) with larger classes, our models generally skipped over the smaller classes, predicting the more common signs instead. Though it would be optimal to incorporate more images into these extremely small classes so that our models can learn their unique patterns, this would be a time-intensive process; for our purposes, we removed the classes from our dataset. After filtering, we have roughly 5500 images and 36 classes, which is then split into train and test sets with 90-10 ratio. Figure 1 plots the size of each class in the train and test set. Figure 2 displays one image per each of our classes.

3.3. Pre-Processing and Data Augmentation

We first resize all images to 32-by-32 pixels. Though the sizes and ratios of the images vary, we find that most are roughly square-shaped, and reshaping them does not significantly change the images in any way.

To account for the unbalanced distribution of class sizes, we augment the smaller classes to create a even distribution.

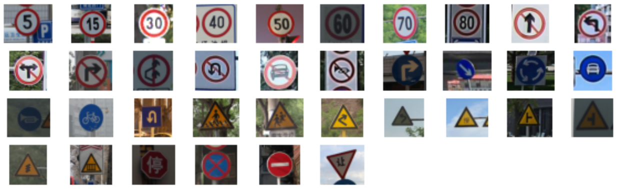

Figure 2. Sample images from each of our 36 classes.

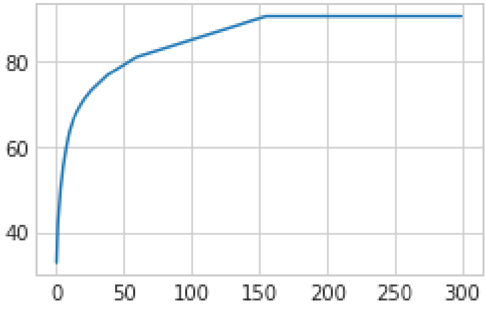

Figure 3. Plot of total variance explained versus number of principal components.

Specifically, for each class, we continuously augment existing images through random rotation until the class consists of approximately 300 images.

We initially also performed random brightness shifts and cropping during augmentation but found that both decreased the performance of our models. This is likely due to the standardized nature of the dataset's image, with signs roughly taking up the same part of the center of the image and in a variety of lighting conditions. Brightness shifts therefore have little effect on the variety of data, and random cropping zooms too far into the image.

3.4. Dimensionality Reduction

Since some methods don't take advantage of the image spatial information, we run flattened images through Principal Component Analysis. Plotting the variance explained by each component, visualized in Figure 3, we settle on taking the first 150 principal components; this is a reduction from 1024 RGB pixels down to 150 features, approximately 5% of the original feature size. In Figure 4, we observe that the top principal components strongly capture outlines, shapes, and colors that commonly show up in our dataset. Figure 5 examines a basic red sign projected onto the principal components, gradually resembling the original image as we incorporate more components.

4. Problem Formulation

We formulate our task as a supervised learning image classification problem. We're given labeled images in our

[1] https://www.kaggle.com/datasets/wjybuqi/traffic-sign-classification-and-recognition

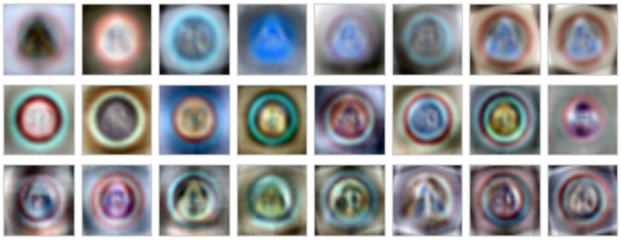

Figure 4. Top 24 principal components, ordered by variance explained.

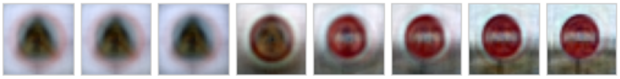

Figure 5. Projections of a sample image onto the first 1, 2, 3, 10, 30, 50, 100, and 150 principal components.

train set, and the goal is to predict the correct classes given images from the test set. We augment our train set to generate an even distribution of class sizes, giving a better chance for our methods to learn each sign type equally.

Each model takes as input a flattened image (of size 3072), a PCA embedding (of size 150), or the original resized image (of shape $32 \times 32 \times 3$); we originally trained on gray-scaled images, but color gives a huge boost in performance due to the color differences between signs. Classical methods and artificial neural networks (ANN) will use the first, second, or both as they don't take into account spatial information whereas CNNs and the transfer learning architectures will use the original unflattened images. Each model will output a probability distribution (or vector of logits) of size 36 representing confidence for each class, and the index of the largest value is the predicted class. Our loss function for this output will be cross entropy loss, the standard loss function for classification problems, that measures the difference in predicted and true (one-hot encoded) distributions.

We originally train our methods on the full dataset and find that they all achieve spectacular performance. To differentiate between models, we run a second experiment on a smaller training dataset with 10 randomly-chosen images per class for a total of 360 train images. With this data-constrained problem, we analyze the effectiveness of certain models when trained on extremely small datasets—relevant in cases of rare traffic signs or costly data gathering.

5. Methods

5.1. Classical Methods

We begin by training classical machine learning classification models on our data and seeing how well it performs. These models are taken from the SciKit-Learn library.

5.1.1 K-Nearest Neighbors (Baseline)

The first classifier we use is a K-Nearest Neighbor classifier. It does not attempt construct any model or relationship between the features, it simply stores instances of the training data. Classification is done by a majority vote between the k datapoints that are closest to the input data. In the case of an image, we look at the differences between the values of each pixel when determining how close two images are.

We choose this model as our baseline because it is a very simple model. There is no training involved, and classifying only requires looking at the k closest images. Because of this it is also very fast, which makes it a good starting point.

Package: KNeighbors Classifier

5.1.2 Logistic Regression

We then train a simple multinomial logistic regression model to classify our data. In the binary case, logistic regression trains a linear model and uses a sigmoid function to map all datapoints to a probability of a class being either 0 or 1. In the multi-class case, we modify logistic regression such that we have a matrix of coefficients, each row corresponding to each class. We then predict the probabilities of each class by using a softmax function.

We choose to use a Logistic Regression classifier because it is also relatively simple and fast to implement compared to the other complex models. It is also one of the more common traditional classification technique used. We are looking to see how these simple models perform compared to the more complex models that will take more time training,

Package: Logistic Regression

5.1.3 XGBoost

XGBoost is another classification model that is popular. The model uses an ensemble of decision trees as well as stagewise estimation. That is, after every iteration of training, the next decision tree is fitted on the residuals of the previous tree. XGBoost also includes a regularization term that prevents the model from overfitting onto the training data.

Ensemble methods like XGBoost are popular because when a group of learners are combined to make a prediction, we are able to get a more accurate prediction without the same level of fear of overfitting. Considering their effectiveness in the Viola-Jones face detector, we chose to train a XGBoost model to see how this popular method performs in our image task.

Package: XGBoost

5.1.4 Kernelized Support Vector Machine

Support Vector Machine is a widely used classifier for many classification tasks. It achieves classification by finding a hyperplane in an p dimensional space, where p is the number of features of the data, to separate the datapoints into different classes. The separating hyperplane that is chosen is the one that maximizes the margin between the classes. In multiclass classification, when a new point is being classified the SVM generates a probability using a softmax function comparing the scores of the new point for each class. The classification is then done by selecting the class with the largest probability.

Kernelized SVMs are used when we are unable to find a separating hyperplane in the current dimension. We use a kernel to transform our data into higher dimensions such that they are more easily separable. These techniques are more popular with multiclass classification, and so we decide to explore how this technique compares to the models described earlier, and with more complex models later on.

Package: Support Vector Machine

5.2. Deep Learning

The modern rise of deep learning strategies with exceptional performance and flexibility warrants an investigation into their effectiveness on our problem. We train both an ANN and CNN, built from the Pytorch library.

There is a slight difference in the output: rather than outputting probabilities, our models output logits that do not go through softmax. This difference has no affect on the predictions and results, however, and is only due to implementation choices.

5.2.1 Artificial Neural Network

ANNs mimic biological neurons, organizing them in layers and connecting all pairs of neurons within consecutive layers. Images are flattened and fed into the first layer of neurons, and activations propagate toward the end as predictions.

5.2.2 Convolutional Neural Network

CNNs are similar but instead of densely connected layers, they employ convolutional layers to take advantage of structured image data. They are also commonly used with max pool and dropout layers to reduce dimensions and regularize training respectively.

5.3. Transfer Learning

Furthermore, we run four pre-trained image models, import from PyTorch, on our data: Resnet18, VGG16, EfficientNet, and a Vision Transformer. Each architecture is trained on ImageNet, and we replace the final dense layer to output 36 classes. During training, we only allow weight updates to the final dense layer since previous pre-trained convolutional layers are already capable of capturing spatial information; we aim to train our models to associate high-level patterns with traffic signs rather than train from the ground-up.

However, we find that though recent studies demonstrate Vision Transformers' effectiveness in image classification, optimization was too slow to be practical for our full dataset. Thus, we solely examine its performance on the smaller dataset.

6. Experiments and Results

Since our test set class distribution is skewed, we evaluate our methods using macro and weighted F1 scores. We calculate F1 scores for each individual class, and the former averages them equally while the latter calculates an average weighted by the size of each class. Our desired model should perform equally well on smaller class compared to larger ones, so we place greater emphasis on the macro F1 scores as our evaluation metric.

We run the same models and tuning methods on both full and small datasets. Table 1 displays the results of all methods from both datasets. Figure 6 plots the most significant performance differences seen while tuning our classical methods on the full dataset.

Most experimentation below is centered around the full dataset as it's our core problem, but most tuned hyperparameters also succeed on the small dataset as well.

6.1. K-Nearest Neighbors

With K-Nearest Neighbors, we use a 5-fold cross validation method to determine the best hyperparameter value, k, to use in our model. Comparing 8 different values of k on the dataset, we find that the optimal hyperparemeter was $k = 1$.

We run the K-Nearest Neighbors classifier on both the original dataset and the dataset that is modified by PCA. From the values in Table 1, we see that KNN performs slightly better on the PCA data. We believe that this is because PCA transforms our data towards the eigenvectors that capture the most variance. This means that theoretically, we generalize all the images in the same class and reduce a lot of the noise that does not provide as much information for us. Within our data, we believe that this would lead to images in the same class being transformed to embeddings that are more similar to each other, hence explaining the slight increase in KNN accuracy—achieving a 0.962 macro F1 score, compared to the 0.959 macro F1 score without PCA on the large dataset and 0.451 and 0.446 macro F1 on the small dataset for without and with PCA respectively as well.

Method	Macro F1	Weighted F1
Full Dataset		
KNN	0.959	0.969
KNN (PCA)	0.962	0.971
Log Reg	0.981	0.984
Log Reg (PCA)	0.975	0.981
XGBoost (PCA)	0.914	0.927
SVM	0.984	0.988
SVM (PCA)	0.987	0.988
ANN	0.903	0.924
CNN	0.988	0.990
Resnet18	0.607	0.642
VGG16	0.644	0.685
EfficientNet	0.410	0.443
Small Dataset		
KNN	0.451	0.439
KNN (PCA)	0.446	0.444
Log Reg	0.647	0.682
Log Reg (PCA)	0.619	0.652
XGBoost (PCA)	0.514	0.551
SVM	0.599	0.613
SVM (PCA)	0.581	0.597
ANN	0.115	0.143
CNN	0.336	0.406
Resnet18	0.323	0.323
VGG16	0.441	0.476
EfficientNet	0.287	0.307
ViT	0.884	–

Table 1. F1 scores across our methods, trained on the full train set and tested on approximately 500 test images.

6.2. Logistic Regression

We include the use of a L2 regularization penalty in our Logistic Regression classifier. This shrinks the weights on the classifier to prevent overfitting to the training dataset. We use cross validation by taking out part of the training dataset as validation here to determine the best value of C, the inverse of the regularization strength constant. After searching over a range of 5 different values, we find the hyperparameter value that provides the best validation accuracy to be $C = 1$.

One thing to note is that with PCA, our optimal C value here is 10. This means that our regularization strength applied to the model is weaker compared to the one trained on a dataset without PCA. We hypothesize that this is because PCA is already a sort of regularization. Since we are reducing the dimension by over 10 times, there is a lower chance that our model trained overfits on this dataset, and thus we are in less need of a stronger regularization term. The use of PCA here did not improve model performance as expected, as it only slightly decreased the macro F1 score from 0.981 without PCA to 0.975 with PCA. This decrease in performance is likely due to the fact that the extra information in the non-PCA data may have been beneficial enough to improve the model's performance despite also the extra unremoved noise. We see a similar pattern in the small dataset, with F1 scores of 0.647 without PCA and 0.619 with PCA.

6.3. XGBoost

We also perform hyperparameter tuning on our XGBoost model. Here we explore 8 different learning rate values and found that the validation accuracy with all 8 learning rates are the same. This most likely implies that we are eventually converging to the same minima no matter what value of learning rate we use.

Given this information, we still choose to use a learning rate value towards the middle of our range for our final model. This is because a learning rate that is too large still has a risk of being unstable, whereas a learning rate that is too small may take too long to converge. Overall, we achieved a macro F1 score of 0.914 on the full dataset and 0.514 on the small dataset.

6.4. Kernelized Support Vector Machine

We also perform hyperparameter tuning with the Support Vector Machine to determine which kernel to use with our model. After performing cross validation, we found that an RBF kernel was the most effective for the Support Vector Machine with a macro F1 of 0.984 without PCA and 0.987 with PCA. PCA actually decreased our performance, similar to in Logistic Regression, on the small dataset, with 0.599 without PCA and 0.581 with PCA.

6.5. Artificial Neural Network

Our ANN consists of four total layers, two of which are hidden, that takes a flattened RGB image and outputs 36 logits. The layer sizes are $32 \times 32 \times 3, 128, 64, 36$ in order. Each are activated with the ReLU function, and we use the Adam optimizer with a constant learning rate of 10^{-3}.

This architecture fits the full dataset nicely, but when we applied regularization or decreased model complexity for the small dataset, we found that test performance actually decreases. Therefore, we apply this architecture to both the full and small datasets, achieving 0.903 and 0.115 macro F1 scores respectively after 10 epochs.

6.6. Convolutional Neural Network

Our CNN is significantly larger, consisting of multiple convolutional layers and ending with two dense layers of size 128 and 36. We organize the convolutional layers into

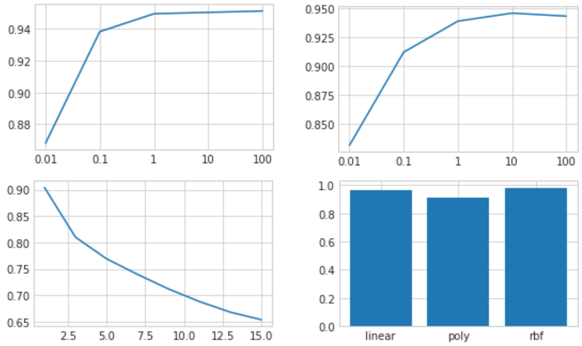

Figure 6. Hyperparameter tuning for Logistic Regression (top left), Logistic Regression with PCA (top right), K-Nearest Neighbors (bottom left), and Kernelized Support Vector Machines (bottom right).

blocks: each block has two convolutions, each followed by batch normalization and ReLU activation; the convolutions are then followed by dropout and a max pooling layer. As with the ANN, we use the Adam optimizer with 10^{-3} learning rate.

During hyparameter tuning on the full dataset, we adjust the number of convolutional blocks in the CNN. We also examine the effects of dropout and batch normalization on our optimal architecture. Table 2 summarizes our results from CNN tuning, and Figure 7 displays the training curves from our optimal 3-block architecture with and without dropout.

We find that even though the architecture without dropout ultimately performs better on the testing data, the validation curves are much more unstable than the architecture trained with dropout. Therefore, we still choose to include dropout in our final result; the regularization stability it has on validation data is much more important than the few extra images it gets wrong during testing.

Thus, we settle on a final CNN architecture consisting of three blocks. The first does not have dropout, the second and third use dropout with probability 0.5. We also insert dropout between the two dense layers that has probability 0.25.

After trying smaller architectures and stronger regularization on the small dataset, we find that similar to the ANN, performance decreases; thus, we also use this architecture for the small dataset. After training for 10 epochs, we get 0.988 macro F1 on the full dataset and 0.336 on the small one.

Architecture	Macro F1	Weighted F1
1 block, MP in between	0.932	0.942
1 block	0.910	0.929
2 blocks	0.965	0.973
3 blocks	0.988	0.990
3 blocks, no dropout	0.994	0.996
3 blocks, no batch norm	0.929	0.944
4 blocks	0.955	0.956

Table 2. F1 scores across CNN hyperparameter architecture tuning.

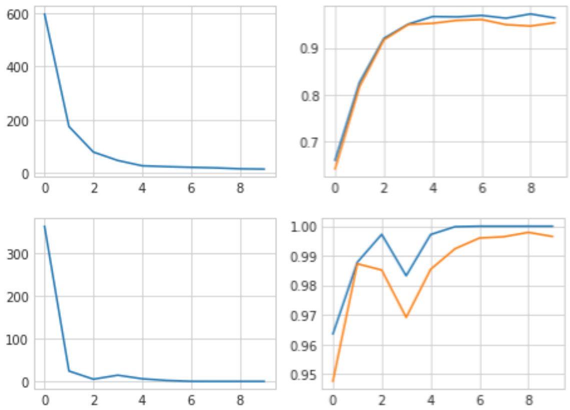

Figure 7. Training loss and validation macro F1 curves for 3 convolutional blocks with (top) and without (bottom) dropout.

6.7. Transfer Learning

With Resnet18, VGG16, and EfficientNet, we replace the final dense layer with one that has 36 neurons. We freeze all weights except the final layer and train with Adam optimizer on 10^{-3} learning rate for 10 epochs. On the full dataset, these networks reached F1 scores between 0.4 and 0.65 while on the small dataset, performance decreased to 0.25 through 0.45, with the exception of the Vision Transformer.

For the Vision Transformer, which we only trained on the small dataset, we replaced the final layer with two dense layers of size 128 and 36 with a dropout of 0.5 in between, which provides a strong regularization effect required for this small dataset. However, due to its long training time, we were only able to train for 5 epochs (taking over 12 hours) before running out of compute; our macro F1 score of 0.884 in Table 1 is measured on the validation set rather than test set, and weighted F1 score was not measured during training.

7. Discussion

Our experiments yield a number of interesting and successful results, and we analyze results for each dataset separately. Moreoever, since the CNN reached near-perfect performance, we also examine intermediate activations to explain its inner workings.

7.1. Full Dataset

We first observe that though all classical methods perform extremely well on the full dataset, XGBoost lags behind by around 0.05. This is slighly surprising given its effectiveness in the Viola-Jones face detector; however, upon further investigation, we find that the original paper used window sizes of 24×24 on grayscaled images along with Haar wavelets whereas we fed it PCA embeddings due to computational time limitations. Our format for the input may be the cause of its slightly lackluster performance.

Our core observation is that classical machine learning methods perform extremely well on the traffic sign recognition problem, beating out all deep learning methods except the tuned CNN. We attribute this disparity to two main causes: problem simplicity and transfer learning's domain shift.

7.1.1 Problem Simplicity

Traffic sign recognition is a more simple problem that other image classification tasks. Our images follow a much narrower range than normal images: signs must be taken from the front side, and they are generally centered in the image. Since traffic signs are also designed to be easily recognizable, they have distinct colors, symbols, and shapes.

Our KNN result with $k = 1$ being optimal strongly suggests this characteristic of our dataset. Checking only the most similar image is good enough to achieve over 0.95 macro F1, implying that our data is strongly clustered, and there's little variation within each class.

Compared to the traditional MNIST handwriting recognition problem, traffic signs have the advantage of also including color information. Though our dataset has more classes than MNIST, the similarity in symbol recognition, single-perspective images, and the effectiveness of classical methods is striking.

Our dataset is in contrast to other common ones like Dogs vs. Cats. Objects (including living things) in the real world can be identified in a multitude of angles and in a variety of poses; this is entirely different from signs, which must be front-facing and unchanging. Our input images has much lower variance compared to other classification problems where deep learning methods truly shine. The decision boundaries are therefore more simple, and classical methods can capture them extremely well.

7.1.2 Domain Shift

Similarly, the reason why transfer learning is not as effective is due to the domain shift from its pre-trained weights, optimized over real-world images in ImageNet, to traffic signs, which represent an extremely small subset of the images our models have seen. Its frozen convolutions may not be tailored well to this kind of image data as signs, writing, and other symbols are not common in ImageNet; the patterns these weights recognize are likely more fuzzy, organic, and variable compared to the straight lines and angles in traffic signs.

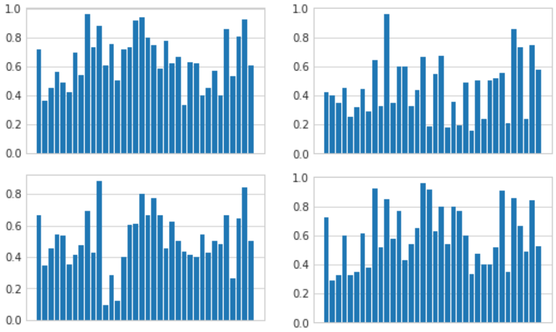

Figure 8. F1 scores in each class for Logistic Regression (top left), K-Nearest Neighbors (top right), XGBoost (bottom left), and Kernelized SVM (bottom right).

Optimizing only the final layer of the pre-trained model is not enough to account for this domain shift. Indeed, when we freed some convolutional layers to also update in gradient descent, we saw an increase in performance; the training speed, however, is much slower than the CNN we custom-designed, and we did not have enough compute to compile final results from these experiments.

7.2. Small Dataset

Prompted by the classical methods' effectiveness on our full dataset, we formed a small dataset to examine whether their performance will also hold in a much sparser environment. We also experimented with this dataset to see if a more fine-tuned transfer learning model outperform the classical methods.

As described in Table 1, classical methods still perform better than deep learning architectures. Interesting, as shown in Figure 8, we see that the F1 score distribution over classes is extremely similar; this suggests that these models learned similar decision boundaries, even with few examples. We attribute this finding to the simplicity of the traffic sign image manifold.

We found that most architectures were overfitting the training data, and during hyperparameter tuning, adding slight regularization effects would cause the training performance to stay extremely low as well, suggesting that there is simply not enough data to train on.

Our sole exception is the Vision Transformer, which af-

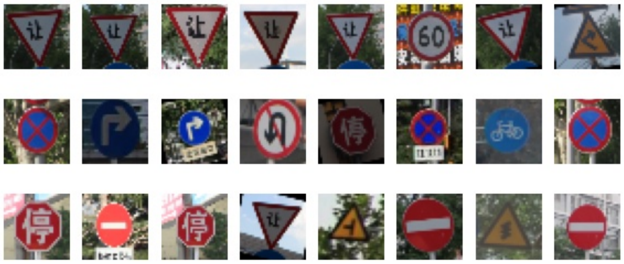

Figure 9. Images that most strongly activated neurons in three layers of the final convolution's activations.

ter tuning the final dense layers, reached a surprising macro F1 score of 0.884, over 0.2 higher than the next-best classical method, logistic regression.

We hypothesize that the Vision Transformer, having a more flexible learning system than the CNN architectures of the other pre-trained networks, learned more information from ImageNet that can be applied to traffic sign recognition. In other words, it learned the bold, solid colors and geometric shapes much more than other architectures, thereby allowing it to adapt more readily to this new domain. Indeed, during training, we only update the weights in our two new dense layers; everything before was frozen, and so basic features and general patterns were propagated down far enough to be useful for our new layers.

7.3. Activation Interpretation

Given the CNN's strong performance on our full dataset, we performed an investigation into its activations. We target the activations of the final convolutional layer as this layer is deep enough to capture larger patterns in the data.

7.3.1 Top Activations

First, in Figure 9, we examine the images in our dataset that produced the strongest activations for three neurons (one per row) in the final convolutional layer. We see that the first neuron mostly targets triangular signs with red borders, occassionally also activating for a round sign with red border or triangular sign with no red border. The second neuron targets round signs in general, and the third targets both round red signs and triangular signs.

7.3.2 t-SNE Activation Visualization

Next, we take activations for 200 images and perform dimensionality reduction into two dimensions using t-Distributed Stochastic Neighbor Embedding (t-SNE), a technique that focuses on preserving local similarities (small pairwise distances) that works extremely well for non-linear manifolds like our set of images. In Figure 10, we plot the original image onto its corresponding t-SNE embedding. As expected, a CNN that successfully learned to classify our images groups similar images together in this space since their activations would be similar in the final convolutional layer; we observe that round red-outlined signs are on top, round solid-blue signs are toward the right, triangle solid-yellow signs are in the bottom left, and round solid-red signs are near the bottom.

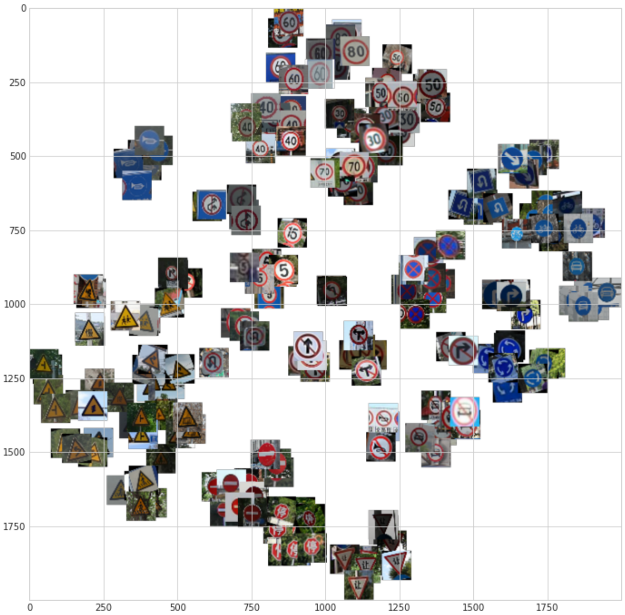

Figure 10. t-SNE projections of 200 images' final activations onto 2-dimensional space.

7.3.3 LIME

Another way to observe high-level features is with the LIME algorithm, which identifies regions of interest, groups of pixels that most strongly influence our classification.

Figure 11 shows 3 examples of running the LIME explainer on images. The green areas on the bottom row represent areas that carry a larger or more significant weight to the classification. The yellow line draws a boundary, and helps us identify what parts of the image is important.

We can see from the circular yellow outlines in the 3 examples that one important feature our CNN looks for is the shape of the traffic sign. This makes sense, as determining the shape yields large information gain. There is a balanced split between circular and triangular signs, and the network determining which shape the sign is will narrow down the possible classes. The other important part of the image is the center of the sign, which is what in the end determines what sign the image is of.

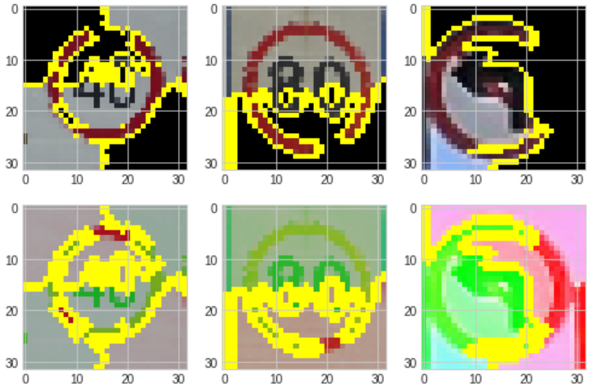

Figure 11. Examples of LIME explained images

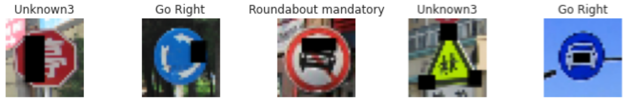

Figure 12. Failed adversarial images.

7.3.4 Adversarial Attack

Finally, we test the CNN with an adversarial dataset. We take a subset of images from the original dataset and delete certain regions in the image. The incorrect predictions shown in Figure 12 are especially informative.

We observe that for signs with Chinese characters, our CNN seems to focus on a certain part in the character; when we block out half of it, as with the first failed image, we get an incorrect prediction.

The second, third, and fifth images are also interesting. Our CNN predicted sign classes that share near-zero resemblance to the signs in the image. We hypothesize that similar to the case above, the CNN actually looks for specific patterns in each symbol—the tip and tail of arrows and the top or center of a car—that, when missing, can cause complete confusion.

One more intriguing observation is the failed fourth prediction, where we only deleted the triangle corners, keeping the symbol intact. In accordance with our LIME analysis, this example suggests that our CNN also relies on the shape of a sign; if the triangle shape is lost, the prediction is thrown completely off track.

8. Conclusion

Traffic sign classification is an important task for self-driving vehicles with high industrial relevance. Therefore, in this paper, we explore the effectiveness of both classical and modern machine learning architectures in solving this problem. While it's not too surprising our results demonstrate the effectiveness of a CNN at solving this problem, reaching a macro F1 score of 0.988, we also manage to achieve similar scores with simpler models due to the simple nature our problem. We see that next best score came from SVM with PCA, reaching a macro F1 score of 0.987. In our more constrained small dataset environment, however, we were able to show the effectiveness of transfer learning over all other methods, with our Vision Transformer model reaching a F1 score of 0.884, much higher than the other's on this reduced dataset. Lastly we used a variety of techniques to understand how the CNN was making classification decision. We found that our CNN largely relied on the overall shape of the sign and specific patterns on the sign to make its decision.

The biggest takeaway from this project is the importance of data exploration and pre-processing. Our initial code ran our models on the entire dataset without filtering out extremely small classes, which achieved subpar results even after significant data augmentation. Moreoever, we also initially used grayscale images, which reduced a lot of helpful color information that we later realize were extremely important, as visualized in the top PCA components. After these two changes to our data, our performance drastically improved, surpassing 0.9 F1 for the full dataset.

Another lesson learned is the importance of considering other metrics beyond pure performance on test data. After achieving near-perfect results on our original dataset, we shift to examine a more constrained problem in a sparse data environment. A model's ability to learn from few examples is also crucial, and our results on the small dataset are equally as exciting.

8.1. Future Work

Considering the classical methods' strong performance on classification, a natural next step is to generalize our problem to traffic sign detection, combining the classification problem with the object localization problem. Real-world self-driving must not only understand a traffic sign but also find it within a image in the first place. Works like YOLO and Faster-RCNN have done extremely well in the past, and it would be interesting to apply similar techniques to traffic sign recognition as well.

Other avenues for future work include expanding the number of traffic signs our models can identify and also designing systems that can learn to identify a class from a few examples. Such algorithms would be essential to learning the sheer variety of symbols and patterns in the modern world. Performance of these models have also been increasingly emphasized in recent studies, and being able to make faster classifications will be important for this task to be applicable to real world self-driving vehicles.

A more far-fetched future vision is to incorporate traffic signs directly into reinforcement learning algorithms for self-driving, forgoing the classification step altogether. Instead, the traffic sign will be part of the state observation (likely some form of video, depth, and motion data), and the presence of a sign should prompt the action to be a response to its meaning; for example, seeing a red octagon that says ”stop” would prompt the action to be ”slow down” or ”brake.” In this sense, classification is completely absorbed into the inner workings of the policy model instead of being a separate part of the prediction pipeline.

References

[1] Yuantao Chen, Jie Xiong, Weihong Xu, and Jingwen Zuo. A novel online incremental and decremental learning algorithm based on variable support vector machine. *Cluster Computing*, 22(3):7435–7445, 05 2019. 1

[2] D. Ciregan, U. Meier, J. Masci, and J. Schmidhuber. Multi-column deep neural network for traffic sign classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3642–3649, 2012. 1

[3] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 886–893 vol. 1, 2005. 1

[4] Ming Liang, Mingyi Yuan, Xiaolin Hu, Jianmin Li, and Huaping Liu. Traffic sign detection by roi extraction and histogram features-based recognition. In *The 2013 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2013. 1

[5] Saturnino Maldonado-Bascon, Sergio Lafuente-Arroyo, Pedro Gil-Jimenez, Hilario Gomez-Moreno, and Francisco Lopez-Ferreras. Road-sign detection and recognition based on support vector machines. *IEEE Transactions on Intelligent Transportation Systems*, 8(2):264–278, 2007. 1

[6] Adrian Rosebrock. Traffic sign classification with keras and deep learning, 2019. 2

[7] Pierre Sermanet and Yann LeCun. Traffic sign recognition with multiscale convolutional networks. In *Proceedings of the 2011 International Joint Conference on Neural Network (IJCNN)*, pages 2809–2813, 2011. 1

[8] Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural Networks*, 32:323–332, 2012. 1, 2

[9] Gang Wang, Guiqing Ren, Zhihai Wu, Yuan Zhao, and Li Jiang. A robust, coarse-to-fine traffic sign detection method. In *Proceedings of IEEE International Joint Conference on Neural Networks*, 2013. 1

[10] Fatin Zaklouta, Bogdan Stanciulescu, and Omar Hamdoun. Traffic sign classification using k-d trees and random forests. In *The 2011 International Joint Conference on Neural Networks*, pages 2151–2155, 2011. 1

[11] Jian Zhang, Weiqiang Wang, Chao Lu, and et al. Lightweight deep network for traffic sign classification. *Annals of Telecommunications*, 2020. 2

[12] Zhe Zhu, Dun Liang, Songhai Zhang, Xiaolei Huang, Baoli Li, and Shimin Hu. Traffic-sign detection and classification in the wild. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016. 2