Traffic Sign Classification: Exploring the Accuracy and Prediction Speed of Machine Learning Models

Robert Dam, Cindy Li, Nathan Castle

Abstract:

Autonomous vehicles have been popularized in recent years and are beginning to become more commonplace. Fully autonomous vehicles have not been manufactured yet due to limitations in accuracy, prediction speed, and computation demands but are becoming more achievable due to advances in machine learning, especially artificial neural networks. This analysis focuses on showing image classification methods that have practical advantages for implementation in autonomous driving by classifying traffic sign images taken from vehicle mounted cameras with high accuracy and rapid prediction speed. Support vector machines, random forests, and several convolutional neural network procedures are implemented and compared to identify if vehicle imagery has real-world application for autonomous vehicles.

1) Introduction:

In recent years, the growth in technology and its outreach on many different parts of our everyday lives has become ever present as more tasks are being assisted or entirely performed by automated systems. One such example of these technological advances can be seen in the vehicles many people operate daily throughout the world as they transition towards a higher form of autonomy. Autonomous vehicles have been around for many years, in fact one of the earliest known forms of autonomous vehicles was manufactured in 1925 by Francis Houdina when he maneuvered a car through Manhattan using radio signals without a driver at the wheel [1]. Almost 100 years later, autonomous vehicles are more popular now than ever with the expectation of about 64 million units being on the road worldwide by 2030 [2]. Although autonomous cars are more abundant now than ever, no vehicle has been able to attain a fully autonomous level of driving without the need for a human operator's inputs. Human drivers are expected to have nearly perfect accuracy when determining driving conditions and road signs along with being able to make very rapid decisions in order to safety operate a vehicle which technology has not been able to recreate. In most cases, autonomous vehicles operate off of several different types of inputs such as GPS coordinates and proximity sensor in order to reach a similar, but still inferior, level of human reaction and recognition. However Tesla, a leading manufacturer of autonomous vehicles, hopes to create fully autonomous vehicle without the need for a steering wheel based primarily on an artificial neural network which primary input would be visible light camera imaging [3].

With all of this in mind, the goal of this analysis was to attempt to improve upon autonomous vehicle systems using visible light camera imaging in a meaningful way that could be applicable to real-world situations, such as Tesla's idea for a fully autonomous vehicle. Since the input information would be imaging from a camera mounted on a vehicle, an image classification method was implemented in our analysis in order to convert the image inputs into a meaningful output in regards to an autonomous driving situations. One key issue with current autonomous vehicles is the lack of accuracy in interpreting the environment around the vehicle and responding to it in a similar or more efficient way than a human operator. Having an inaccurate response to a particular input, whether it be a roadway hazard of traffic sign, could potentially lead to serious accidents and make the autonomous vehicle unsuitable to real-world use. Therefore, the analysis focused on trying to ensure the image classification method had the highest possible accuracy in interpreting the image inputs. Another key feature addressed in our analysis is the importance of prediction speed and computational demands, especially upon weaker computational systems such as ones found in vehicles. Human operators are able to see traffic signs or roadway hazards and react in fractions of a second

so it became imperative for the image classifier to exhibit similar prediction speeds in order for it to be a contender for replacing a human operator. These two ideas became paramount to the analysis for quantifying if an image classification method from visible light imaging could be used to safety and effectively replace a human vehicle operator with an autonomous system.

2) Dataset and Exploratory Data Analysis

The dataset used in the analysis is called the German Traffic Sign Recognition Benchmark which contains about 40,000 images of traffic signs taken from a visible light camera mounted on a vehicle as it approached a given traffic sign along the road [4]. The images taken contain 43 different traffic signs ranging from speed limits to hazard warnings which can be seen in the grid below:

Figure 1: All 43 traffic sign types in GTSRB

The example images of each traffic sign class above characterizes some of the differences between the images contained in the dataset. One such trait can be seen in the similarity of traffic sign stylings where several different traffic signs have very similar shapes, similar colors, or a mixture of both qualities which may make identifying each sign a little more difficult. Another trait of the images is the range in lighting

where some images appear darker or brighter than others due to the time of day the images were taken. This may lead to a more rigorous image classifier since the model will be able to account for variations in light saturation, similar to how it would need to perform on a real-world autonomous car system being used at various times of the day. Aside from the light saturation, each individual traffic sign imaged along the roadway in the dataset contains a batch of images for the same traffic sign taken at different distances and with different pixel densities as the car made its approach towards the sign. This will also allow the model to be more rigorous for identifying various formats of images for the same traffic sign.

Below we can see a two-dimensional histogram for the pixel densities for each image in the dataset.

Figure 2: Dataset image dimensions

From the plot we can see that the vast majority of images have a pixel density of around 30x30, but range all the way to about 200x200, which will be important during the analysis as some image classifiers need the input images to be the same density. In these instances, we used a conversion method to convert the image pixel densities to a uniform density based the most common dimensions in order to retain as much information in each image as possible. Aside from the image themselves, the dataset also contained a metafile with information on how to crop each image to only contain the traffic sign. Cropped images, as well as grayscale images, were implemented in the analysis as forms of feature engineering in order to try and increase the level of accuracy and reduce computational demands on the image

Figure 3: Examples of cropped and grayscale images

Below we show a count plot for the total number of images within each of the 43 traffic sign classes.

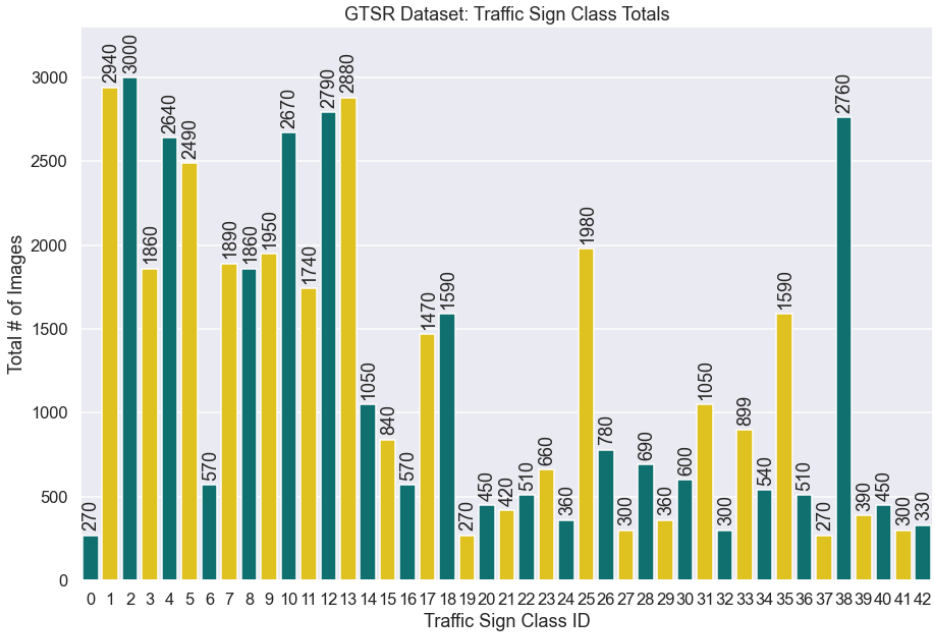

Figure 4: Count of number of training images in each class type

The plot shows that the traffic sign classes have an unbalanced amount of images, where some classes have as low as 270 images while others have as many as 3,000 images. This could potentially lead to issues with certain classification models that aren't well suited for unbalanced datasets so when determining the proper classification methods, this was considered in order to make the analysis as clear and accurate as possible. One possible remedy could be to omit the traffic sign classes with minimal amounts of images, but since the images in each class contain such variation all traffic sign classes were retained in this analysis.

3) Setting the Benchmark: Random Forest and Support-Vector Machine (SVM)

Before diving into deep learning for image classification, some traditional machine learning models will be used for getting benchmarking performances for the project. After initial runs on a variety of traditional classification algorithms,[5] Random Forest and Support-Vector Machine are selected for our study based on the performance. Both models are fitted on the original, unprocessed image dataset at first. Then feature engineering is performed on the dataset to obtain grayscale images and cropped images and the two classification models are retrained with these two kinds of processed image datasets respectively for improving performances.

3.1) Random Forest Classifier

Random forest is a supervised machine learning algorithm by constructing multiple decision trees on different observations. Random forest is a commonly used model for both classification and regression problems. For classification tasks, it generates the classification results with the majority votes from those decision trees. Basically, the working principle of random forest algorithm involves ensemble learning, including bagging for create training sample subsets and boosting for generating sequential models.[6] The workflow of random forest is shown as below:

1. Random number of samples are taken from the data set.

2. For every sample, a decision tree will be constructed, and output generated from decision tree will be collected.

3. The majority voting from all decision trees will be set as the final classification output.

3.2) Support-Vector Machine (SVM)

Support Vector Machine (SVM) is another supervised learning algorithm, which is widely used for classification tasks. In classification problems, hyperplanes are the boundaries for identifying classes which the data points belong to. When the feature dimension gets large and the hyperplane becomes multi-dimensional planes, it could be really hard to find hyperplanes correctly[7]. With SVM algorithm, data points are plotted in n-dimensional space and with feature values on different coordinates. SVM is then used to find the hyperplanes that distinguish different classes as good as possible.[8] SVM is relatively effective in high dimensional data and memory efficient by using subsets for training data.

3.3) Dimensionality Reduction

In machine learning tasks, dimensionality reduction techniques are usually applied for high dimensional data set to reduce the number of input features to lower the model complexity and increase model performance, as well as reduce the computational efforts. The image data set used for our study has large number of dimensions in the feature space.[9]

For potential improvements in model accuracies and prevention of "curse of dimensionality", we conducted two methods for lowering the input information: converting the images to grayscale or cropping the edges not providing information from images. As mentioned in many other image classification studies, converting images to grayscale may help reduce the signal to noise rate of the input images, which helps simplify algorithms and reduce model runtimes.[10] Image cropping enables the models to focus only on the regions carrying information and is proven to improve classification accuracy.

3.4) Comparing Results for Fitting on Different Image Types

Upon completing the EDA and image preprocessing, we train random forest and SVM on original images, grayscale images, and cropped images respectively. The model fitting progresses are briefly explained below:

1. Firstly, the training and testing images are imported and converted from NumPy array to PIL image.

2. Then, images are resized to the same height and width and training images are shuffled. Meanwhile, labels for training and testing image data are also retrieved and stored as arrays.

3. Also, images are normalized by dividing 255 (since image pixel values range from 0 to 256).

4. Since the RF and SVM algorithms from scikit-learn expect the inputs to be 2D array, the read-in images (with 4 dimensions) are then reshaped to 2D.

5. Fit the random forest and SVM using train data and make predictions with test data. Confusion matrix and classification accuracy are obtained for all three types of input images.

3.4.1) Accuracy Comparison

ML Models	Original Images	Grayscale Images	Cropped Images
Random Forest	78.61%	77.04%	81.92%
SVM	82.30%	82.47%	87.67%

Figure 5: Accuracies for RF and SVM on three image types

The above table provides a summary of prediction accuracies for both models with original, grayscale, and cropped images accordingly. By fitting the model on original image data, we obtain 78.61% classification accuracy for random forest model, while 82.30% accuracy for SVM, which are not bad for being a benchmark for model performance. The accuracies obtained by retraining both models on grayscale images are around the same (slightly lower for RF) compared to those using the unprocessed images. However, we can see there are significant improvements in prediction accuracies with cropped images: a 3% increase for RF and 5% increase for SVM. This further proves the conclusions from some earlier study that cropping can help improve model accuracy for image classification.[11]

3.4.2) Runtime Comparison

The image formats do not only influence the model performances, but also affect the computation complexities. To learn the potential associations between image types and model efficiencies, training and predicting times using three types of images for both classification models are recorded, as shown in the figures below.

For model training stage, we can see that a large amount of runtime is saved by converting images to grayscale for both random forest and SVM. There are also decent reductions in runtimes for both models by using the cropped images, although the decrease for SVM is not as significant as that using grayscale.

For model prediction stage, making predictions with random forest model for all image types is relatively fast, which only takes less than 1 second. Hence, the change in runtime using different image types is not obvious. However, for SVM, fair amounts of model predicting time are saved by either converting the images to grayscale or cropping them.

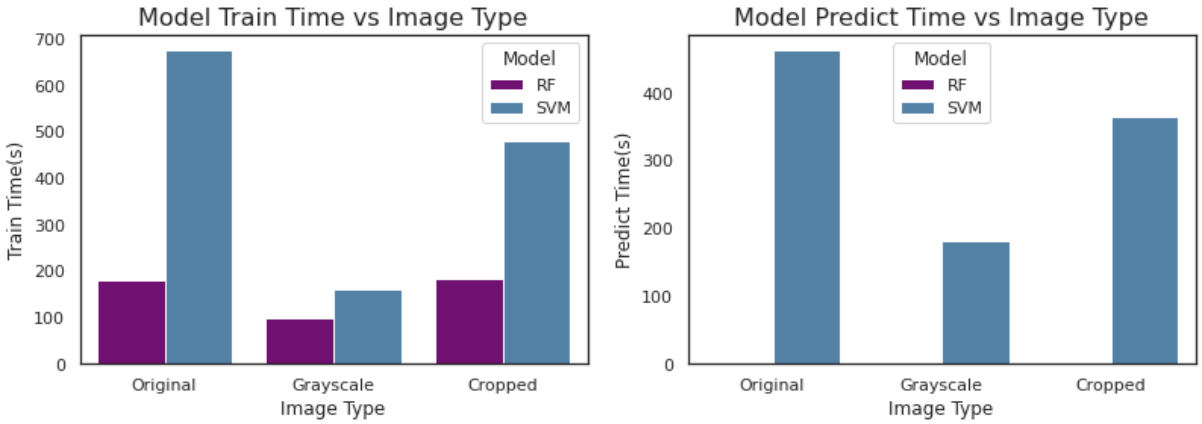

Figure 6: Training time and prediction time comparisons

In conclusion, both grayscale images and cropped images help improve our model performances. Apparently, cropping the images helps us achieve higher model accuracy and consume relatively less runtime. Although grayscale images don't aid much in improving prediction accuracies, they are super useful for boosting model efficiencies.

3.5) ML Model Deficiencies for Image Classification

Traditional machine learning algorithms have many advantages: they are easy and quick for deployment, relatively effective in making predictions, and comparatively light weight with less computing power required. Thus, they are widely used in image recognition tasks for years. However, to train traditional models with high accuracy, lots of efforts are required for data pre-processing, feature engineering, as well as parameter tuning parts prior to model constructions. Also, the performances of classical machine learning models tend to be limited by the "curse of dimensionality" when it comes to high-dimensional data. In contrast, deep learning models perform better on datasets with higher volumes, and they don't need extra feature engineering to boost performance when features are limited. In addition, deep learning fits better on unstructured data, such as text, voice, and images, with higher self-learning rates.[12] Combining the facts above, we then choose to apply deep learning models to solve the image classification task in our study.

4) Introducing the Model: Convolutional Neural Networks (CNN)

For our image classification task, we will use Convolutional Neural Networks (CNN) as our model of choice. Since the introduction of LeNet in 1988 by Yann LeCun [13], CNNs have become the go-to method for image classification because of their high accuracies, scalability to high dimensions, and lower data preprocessing demands. We will first introduce the CNN model and its general principles then we will present our methodologies and results from using this model on the German traffic sign dataset.

4.1) CNNs

CNNs are a type of neural network used to specifically exploit the structure of image data in order to have more efficient and effective classifications. On a high level, neural networks are non-parametric function approximators represented as a collection of nodes arranged in various architectures and layers with weighted connections to other nodes within adjacent layers.

CNNs come with no formal assumptions, but they will perform better with more training data in order to have enough cases to pick up patterns from. Furthermore, the testing data should not be too drastically different than the training data in order to have CNNs actually perform well on the testing data.

A CNN is an arrangment of specific types of neuron layers called convolutional layers, pooling layers, and a final set of dense layers in order to output predicted class probabilities. In this section we will explain each element of a CNN and in the next we will provide some intuition and explanations on why these elements allow a CNN to be well suited for image classification.

Let each of the inputted images be a tensor of $H \times W \times D$ pixel values, where H is the height, W is the width, and D is the number of color channels. We will be using the color channel format RGB where there are three channels and each channel represents the red, green, or blue intensity of a pixel.

A convolutional layer C_ℓ is a special type of neuron layer that consists of a set of J parallel filters/kernels. Each $K_{j\ell} \in C_\ell$ is a $n \times m$ matrix of dimension less than $H \times W$ containing weights. Each $K_{j\ell}$ is designed to detect the prescence of some feature. Features are certain aspects that differentiate images from each other given the pixel data. So for example, some simple features can be the prescence of straight lines, or the presence of round lines. These are primitive features than can be combined to create more abstract features like a dog shape.

Kernels can detect features by performing the following operation. Starting from left to right then top to bottom, for each color channel the dot product of each $K_{j\ell}$ is taken with overlapping $n \times m$ sections of the input matrices. Then the values for each channel are summed for that section. For example, if we move by one pixel, we would take the dot product for one section then move off by one pixel to the left and take the dot product with the new overlapping section. Letting these $n \times m$ sections be denoted B_{id}, the feature scores are defined as follows

$$s(B_i, K_{j\ell}) = f\left(\sum_{d=1}^{D}(K_{j\ell} \cdot B_{id}) + b_{j\ell}\right)$$

where $f()$ is an activation function that introduces non-linearity into the model, and $b_{j\ell}$ is a bias term. High feature scores represent high detection of a certain feature in section B_i.

The output of each C_ℓ are J matrices corresponding to each kernel $K_{j\ell}$ where the entries of those matrices are all of the feature scores $s(B_i, K_{j\ell})$ for every B_i.

For the first layer, the input matrix is the raw pixel data from the image, but in subsequent layers the input matrix is the output of the previous layer. This stacking of layers allows the initial layers to detect primitive features like simple lines and shapes while the later layers detect more abstracted combinations of the simpler features.

Between sets of convolutional layers are pooling layers. Pooling layers are layers that aggregate information within a partition of $p \times p$ block for the outputted matrices from the previous convolutional layer. The typical aggregations done are by taking the maximum or average of the $p \times p$ blocks. Since pixels in close proximity often provide similar information, pooling outputs downsampled image data to lessen the complexity of the CNN while retaining most of the information. The operation of pooling also lessens the noise in each block and extracts out more of the signal.

Furthermore, there are layers in between that provide regularization to combat the tendency of flexible neural networks to overfit. Overfitting is when models learn the noise in the data and memorize the dataset rather than pick up on the underlying signals and patterns from the data. These regularization steps include dropout, where a certain percentage of node are deactivated to lessen the network's flexibility, batch normalization, where inputs to each layer are standardized to have mean 0 and variance 1, and more.

Once features are learned, the convolutional output is flattened into a vector which is then fed into fully connected dense layers in order to produce class probability estimates. In the dense layers every neuron in one layer is connected to every neuron in adjacent layers. This is much like how classical neural networks were designed. Using the feature information from the convolutional and pooling layers, these dense layers will then try to approximate the function that produces accurate classifications. The final layer uses the soft-max activation function which scales the output into the $[0, 1]$ range to be used as class probabilities. Let C be the number of classes and $\mathbf{z}$ be the C-dimensional flattened vector after the dense layers. The softmax function assigns an image a probability of being in the i'th class as follows:

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{C} e^{z_j}}$$

The weights in each kernel are obtained by minimizing a loss function after a forward pass through the neural network. Namely, at after each forward pass we find

$$arg\min_{w} \mathscr{L}(\mathscr{N}, w)$$

where $\mathscr{L}$ is the loss function, $\mathscr{N}$ is our neural network, and w is the collection of weights.

We repeat the steps of passing data through the network then finding the weights through optimization until we reach some level of convergence. Neural networks use techniques called backpropagation and stochastic gradient descent to efficiently compute optimal values for these weights despite the vast size of neural networks. See [14] and [15] for more details.

For image classification, we use the categorical cross-entropy loss which has the form

$$\mathscr{L}_{CE}(\hat{y}, y) = -\sum_{t=1}^{T}\sum_{i=1}^{C} y_{ti} log(\hat{y}_{ti})$$

where T is the number of training cases, y_{ti} is an indicator that the true image is in class i, C is the number of classes, and $\hat{y}_{ti}$ is the predicted probability that image t is in class i.

The cateogrical cross-entropy loss function measures the closeness of two discrete distributions so it measures how close the predicted probabilities are to the real image labels.

4.2) Explanations and Advantages of CNNs

CNNs offer advantages to other machine learning classification models since they lower the need of data preprocessing, often have better accuracy, and scale well to high dimensions.

As we saw in the results of the random forest and support vector machines, feature engineering did play an important part in both the accuracy and runtime of those models. However, feature engineering also introduces more overhead, more data processing, and one does not know a priori which feature engineering steps would lead to improvements.

Convolutional layers are a key advancement over past feature engineering in computer vision because the weights of the kernels are learned through the minimization of the loss function that we described in the previous section. So the setup of the model allows the CNN to find meaningful features. This effectively eliminates the need of the researcher to manually specify meaningful features for specific image classification tasks.

To provide some intution on why kernels can detect features let us consider an example of a 3×3 kernel that can detect vertical lines. Consider the kernel

$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

The dot product of this kernel with another 3×3 matrix will be high when the pixel values of the center section are high which indicates a vertical line through the middle of this section.

Finally, CNNs are better suited our task of image classification because they can handle high dimensions in a prediction and efficiency sense. CNNs have a high number of parameters and a non-linearity introduced by the activation function which allows them to handle complicated functions and the high dimensionality that comes with representing images as three channels of pixel data. Additionally since CNNs are not fully connected and they downsample with pooling layers they are able to lessen the complexity of the network which increases efficiency.

5) Design Choices of Our CNN Architecture

Now, we discuss the specific architecture and design choices of our CNN model. As our main goals is to explore the possibility of real time accurate traffic sign classification, these choices were made with the both the goals of accuracy and speed in mind. Since most of the images in our dataset are smaller and we were able to achieve respectable accuracies using smaller images sizes, we committed to designing our neural network based on 32×32 image dimensions in order to have a more efficient model. This size is large enough to have two pooling layers while still being quite small. Also this image size is around the most common dimensions for the images in our dataset which means we don't need to drastically upscale or downscale as many images. Too much upscaling or downscaling of images would obscure the information of the images. Below, we have a diagram of our simple architecture and the code in the appendix will provide a more detailed look at our model.

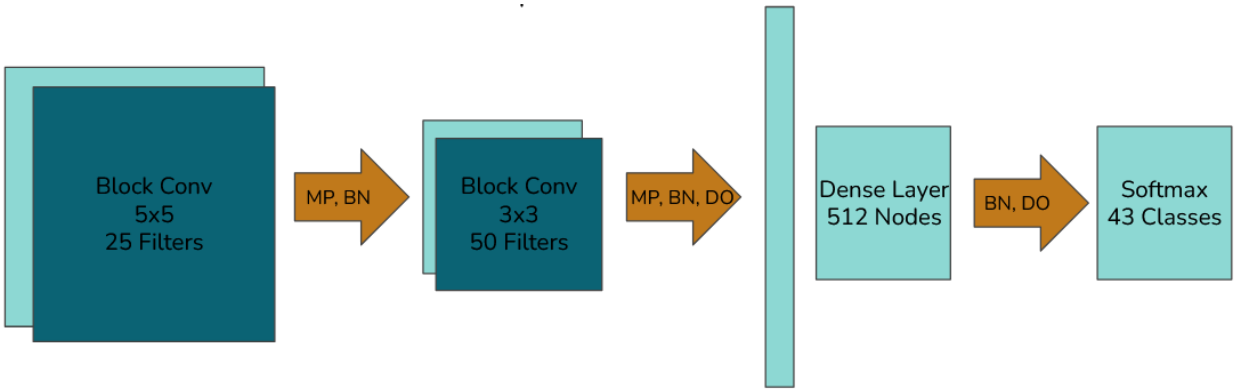

Figure 7: Simple Custom CNN Architecture

Our architecture takes after the classic LeNet [13] which is a lightweight neural network that can serve as a solid base for a simple CNN. We make some modifications to better suit our dataset and to increase accuracy. The original LeNet was a simple sequential design of two convolutional layers separated by pooling layers, followed by two fully connected dense layers, and a final softmax layer for classification. Our architecture still follows that same overall layout, but instead of single convolutional layers we use two convolutional layers stacked on top of each other at each convolution step. This technique was introduced by a CNN called VGG [23] and it has been shown to increase the ability of a neural network to learn abstracted features in the second layer after learning simple features in the first layer.

The first convolutional block uses 5×5 sized filters and 25 filters per layer. This is a typical amount of filters to use on smaller images and we determined the number of filters to use through experimenting with different sizes until we found one with a reasonable performance. At the deeper convolutional block, we use 3×3 sized filters and 50 filters per layer. This technique of increasing the number of filters, but decreasing the filter dimensions at deeper levels is one used in LeNet and the famous extension of LeNet called AlexNet [24]. The intuition behind increasing the number of filters is that deeper levels deal with more abstracted features so they will need more filters for a wider variety of features. The intuition behind decreasing the filter dimensions is that pooling decreases the input size to deep levels so decreasing the filter dimension makes those filters more suited to the smaller input sizes.

For the pooling layers, we chose max pooling which takes the maximum value in each block to downsample inputs. Pooling choice is a heuristic without clear guidelines so through experimenting with max pooling and average pooling we found that there was little difference in performance between the two. Max pooling reduces computational load so we opted for it instead.

The fully connected dense layers have the most parameters since each node is connected to the nodes in adjacent layers so they are the most expensive layers. So we opt for only one dense layer to reduce complexity, but provide it with 512 nodes to have enough parameters to train for the classification task.

As for the activation function, we opted to use to Leaky ReLu function which has the following form

$$f(x) = \begin{cases} \alpha x & x < 0 \\ x & x \geq 0 \end{cases}, \ 0 < \alpha < 1$$

Here, α is typically a small number close to 0. Intuitively, the Leaky ReLu function is a piecewise linear function that shrinks smaller feature values while maintaining the influence of larger values. This function is used in practice since it introduces non-linearity, but is still quite simple. This reduces computation time, leads to simple gradient calculations, and tends towards numerical stability. As a note, another very popular activation function for CNNs is the ReLU function which is the same as the Leaky ReLu function except negative values are assigned to 0. This often leads to the issue of neurons on the later levels being stuck at 0 so the Leaky ReLu function provides a bit more flexibility while maintaining the benefits of simplicity.

Finally, we take some steps to help combat the overfitting of our model to increase general testing performance. After both pooling steps and the dense layer, we apply batch normalization. Batch normalization standardizes the outputs of the previous layers by subtracting the mean values and dividing by the standard

deviation. Since layers are connected, at each step when the weights are changed the inputs to other layers are changed which causes instabilities. Batch normalization helps combat this difficulty and it provides a small aid against overfitting by introducing some noise into the model. It has also been shown to improve the efficiency of the model. For more details see [25]. In addition to batch normalization, we also place a 30% dropout layer after the convolutional steps and a 50% dropout layer between the dense layer and the final softmax layer. These values were chosen as reasonable values through common practices and through our own experimentation. Dropout layers randomly deactivate a certain percentage of the neurons from the input which reduces the number of parameters that the next layer sees in the current epoch. What this does is that it reduces the flexibility of the model so only larger trends can stand out. This helps with generalizability of the model so that the model does not learn the unimportant details of the data.

The last step that we take to combat overfitting is data augmentation. Data augmentation is the technique of applying random transformations to your training images in order to create more training data and give your model more variety to train on. As we noted in the data exploration, we do have unbalanced classes where some traffic sign types had small amounts of training data. Data augmentation can help combat this issue by creating more training data. In this case, we applied the random transformations of zooms, small rotations, shears, and translations. Each of these transformations are ones that could reasonably happen to traffic sign imaging while driving. We did not apply image flipping because image flipping changes the meaning of traffic signs.

With these design choices in mind, we have kept simplicity of a lightweight model, but have introduced not too expensive techniques to improve image classification performance for this task.

6) CNN: Comparing Results of Several Architectures

In this section, we will explore the effectiveness of our model and compare its results to four other top-performing architectures on the ImageNet classification task. Our task is focused on prediction and efficiency. We wish to find an implementation of a CNN that produces the highest test accuracy for our traffic signs dataset while still being efficient enough to deploy on weaker hardware in real time. To deploy a model into autonomous driving systems we would like very precise results in order to limit hazardous situations on the road.

ImageNet is an expansive image database that contains over 14 million labelled images in over 20,000 categories. ImageNet serves as the go-to benchmark for computer vision tasks and the search for effective classification of this database has driven much research [16]. The four architectures with among the top ImageNet accuracies that we trained are ResNet, DenseNet, Xception, and EfficientNet. See [17]-[20] for more details on these architectures. Each of these models implement novel ideas to increase the accuracy of the CNN while maintaining manageable complexity. We then compared the performance of these larger architectures to our simple sequential CNN.

Each neural network was trained on our test data for a number of epochs until training and validation accuracy stopped clearly increasing. The final train, validation, and test accuracies are summarised below in Table 1. In Figure 8 the training times, classification times, and number of trainable parameters are summarised. The number of trainable parameters are the weights of the neural network.

Table 1: Accuracies of Different CNN Architectures on Traffic Sign Classification

Architecture	Train Accuracy	Validation Accuracy	Test Accuracy
Custom Simple	99.32%	99.91%	98.76%
ResNet50V2	99.37%	99.25%	94.14%
DenseNet121	99.52%	99.95%	97.56%
EfficientNetB0	98.81%	98.99%	95.53%
Xception	98.93%	99.85%	96.79%

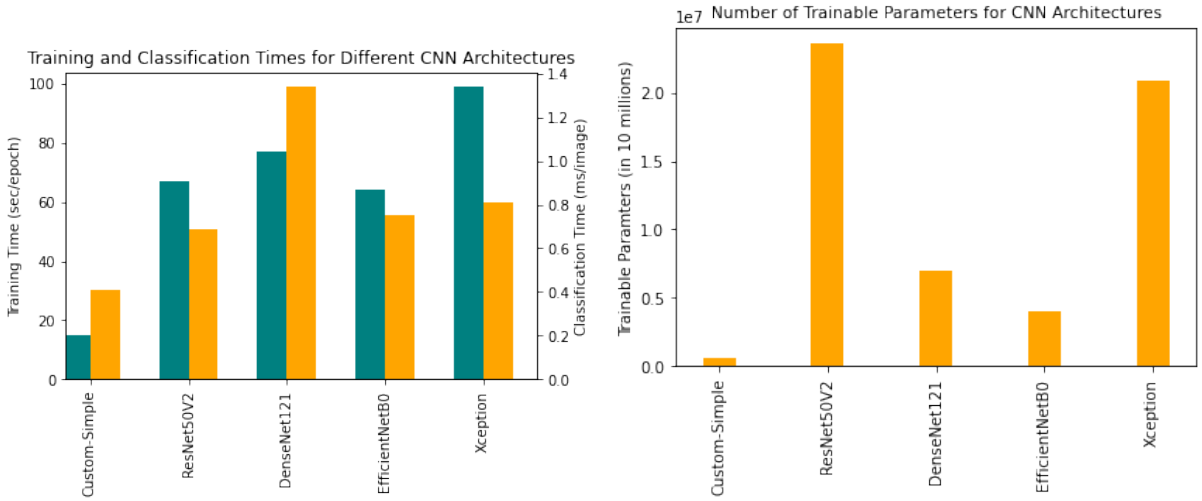

Figure 8: CNN complexity and speed comparisons

From this table and charts we have some takeaways. For the most part, all of the neural networks achieve similar training and validation accuracies. This is not too surprising since the flexibility of neural networks allow them to eventually detect patterns within the data to accurately classify the training data. Furthermore, the task itself is not a highly complicated one and the dataset is images are clean and cropped. None of the models demonstrate clear overfitting since their test and validation accuracies are close to each other. However, there are some gaps between accuracies predicting the test set. ResNet50V2 and EfficientNetB0 both have noticeably lower test accuracies than the rest of the models. Our model performs the best at 98.76% test accuracy which is a respectable accuracy that stands out over the rest. As a caveat, our custom model was more tuned than the other models since it was quicker to fit and optimize hyperparameters. This result shows how for this task, a simple architecture is able to achieve high test accuracies so we can pursue simple models to have the benefit of efficiency. Also, although CNNs do remove much of the prepreprocessing demands, even the most sophisticated architectures do not provide automatic results so there are still model tuning and data based decision making demands on the researcher.

One possible reason why some of these models may less effectively generalize to the test set is that most of these architectures were design for image sizes around 252×252 whereas most of the images in our dataset are under 60×60 dimensions. So for the larger models, we resized all images to 64×64 and 71×71 for Xception, but only used 32×32 images for our simpler custom model. We did not upscale our images all the way to the standard image sizes for these architectures since that much upscaling will likely just interpolate redudant information that will not improve model performance.

In addition to accuracy measures, training time and specifically classification time are important to practically deploying autnomous driving systems. As we can see from Table 2, our custom CNN trains and classifies images much faster than all of the other architectures. This is due to the simplicity and lower number of parameters of this model compared to the other ones. We can see that our model had far less parameters than the other models in Figure 1. We can also note that DenseNet121 had the second best accuracy, but it also has the longest classification time at more than three times more than our custom model. However, all of these classification times are not too slow for a single image which suggests that traffic sign classification will not be a major bottleneck in autonomous driving systems. On weaker hardware though, it will take longer to classify images so we still want as quick classification as possible so we scale to weaker hardware.

Our custom model achieves higher test accuracy to the other architectures, and is much simpler, less memory intensive, faster and more feasible to deploy onto autonomous vehicles for real time traffic sign classification. Furthermore, the much lower training time is useful since autonomous driving systems would need to occasionally retrain their traffic sign classification models once they acquire more data. In the actual classification of traffic signs, it seems like smaller images are enough to provide the necessary information to distinguish between traffic signs which is not surprising since traffic signs are designed to have simple

geometries to be easily recognized. This result suggests that optimal production level models need to consider time complexities, space complexities and accuracy measures.

7) Analysis and Diagnostics of Results from Our CNN Model

In this section, we further analyse the fit of the model and our classification results with this model to gain an understanding of what types of misclassifications it makes. We would like to find possible areas to improve the accuracy of this model while maintaining lower complexity. First, we will examine the training and validation accuracy curves to look for any prescence of overfitting then we will examine the data side itself. Since much of the performance of neural networks depends on the quantity and quality of the training data, much of this analysis will examine those factors.

To examine the fit of our model we examine the learning curve of our model through 30 epochs. As we can see in Figure 9, both the training and validation accuracies seem to settle at high values without a gap between them. This suggests that the model is able to optimize the predictions on the training data well which suggests the model is robust enough to and accurate enough to handle this image dataset. Since the validation accuracy also settles at a high value without a gap between this accuracy and the training accuracy we can also conclude that the model generalizes well to the validation data. So the steps taken to regularize our model has prevented overfitting as well. From this, we can conclude that the design choices we made allow for effective learning of the data with generalizability.

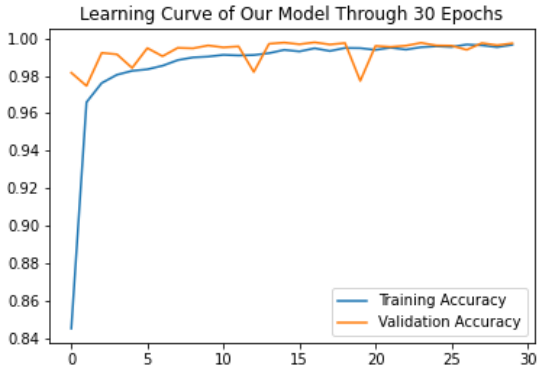

Figure 9: Training and test accuracies settle at high value without gap

Now, we examine the classification results of our model in relation to the amount of training data we have. Figure 3 plots the F1-score of each traffic sign class with the amount of training images in each class. The F1-score is a measure that captures the classification performance of a model for a specific class. It combines the information of the precision and recall for each class. Precision is a measure of the proportion of correct classifications for a class over the total amount of times the model predicted an image was in that class. This measures the true positives rate for a class. Recall is a measure of the proportion of images in a class that were classified correctly over the total amount of images in that class.

Figure 10 shows us that classes with lower amounts of training images tended to more often have lower F1-scores, sometimes dramatically so; where one class sits at less than 0.7 F1-score while most sit around the high 0.9 values. There are still many classes with low amounts of training data that still are classified well, but in the higher training data ranges past 1000 images none of these classes have F1-scores lower than around 0.97. This result suggests that to increase the training accuracy of our model to very precise production levels we may just need more training data for these classes with sparser data. This result also shows that although data augmentation may help produce a variety of more training images, it seems to not replace actually having more initial training data.

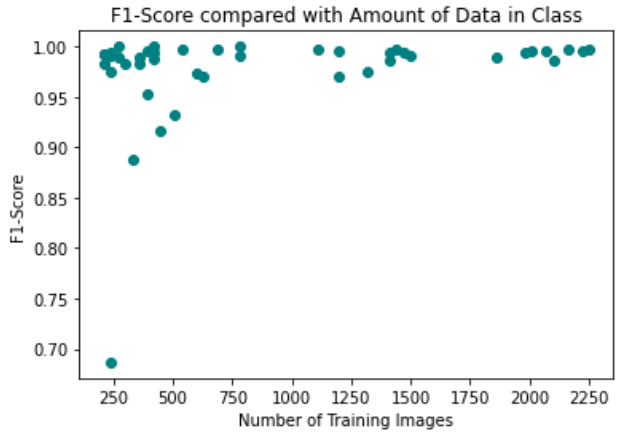

Figure 10: Less training images corresponded to typically lower accuracies

Now we look at the predictions made by the poorest performing class, class 27. Below, we visualize some misclassifications by our CNN regarding class 27. We also visualize some examples of images in class 11 and 21 which were the only misclassifications for class 27, with 20 classified as class 11 and 6 classified as class 21.

Figure 11: Misclassifications of class 27

As we can see from these images, the signs themselves in classes 11 and 21 look very similar to the ones in class 27. Also, the class 27 signs look to have been all be almost the same image taken simultaneously with a shadow obscuring half of the sign. This likely provides even less data and noisy data. Couple this with the fact that there were less training images in class 27 it seems like there was poor data quality for this one class which is why it has subpar results. Since class 27 is a pedestrians crossing sign it would be quite important for our model to recognize this sign for pedestrian safety. So to ease this issue, more a better quality data is needed for the model to learn the differentiating features of these images. It is able to do that most of the time since the overall accuracy for these classes is still pretty high, but it is not able to differentiate between these classes precisely enough to deploy safely.

From these observations and the trend of accuracy to increase with the number of training images in a class, we can conclude that just having more and higher quality data can likely lead to the accuracy improvements that we need to have extremely precise results. This sets a path for us to maintain a lightweight model while attaining the precision needed to deploy this techonology onto the roads.

8) Conclusion

Our results suggest promising possibilities for using simpler convolutional neural networks for real time image classification tasks in autonomous vehicles. Not only does this methodology allow the practitioner to not have to meticulously specify meaningful features for specific image classification tasks, but it achieves a respectable accuracy with quick classification times. In our model, we only use 483.7 thousand parameters, have a training time of only around 15 seconds per epoch using GPU boosting, and a classification time of around 0.41 ms/image. These times are fast enough to deploy into real time use and will likely scale well to weaker hardware. With our lighweight model, we are able to achieve 98.74% test accuracy which is a respectable, but maybe not quite precise enough number for full use in production.

In comparison to the traditional machine learning methods of random forests and support vector machines, convolutional neural networks were able to achieve much better test accuracy without needing as much data preprocessing. In comparision to more complicated, state-of-the-art architectures, our custom simpler model was able to achieve much faster training and classification times. This step is crucial for actual deployment of a model into autonomous cars. With optimizations and model tunings we were also able to achieve higher test accuracy than the more sophisticated models which suggests that for this case, a simple architecture is still strong enough to achieve respectable accuracy.

Although we were not able to achieve the very high level of precision needed to fully deploy this model into autonomous vehicles, we do achieve very strong results for most classes and we have a path forward to achieving higher accuracies. This solution is data based where we likely need more training data for certain classes to have better testing accuracy. Future endeavors should explore additional model optimizations to improve accuracy and the feasibility of collecting a higher variety of training data. Other techniques like encorporating GPS data, and aggregations of multiple predictions as safeguards may also be fruitful directions for achieving extremely precise traffic sign classifications.

Practical deep learning should consider not only accuracy achieved by extremely deep networks with abundant computing, but it should consider efficiency for deployment on weaker systems. Furthermore, a data centric and problem specific approach to machine learning allows one to reap the benefits of deep learning's impressive accuracy achievements while also balancing the tradeoffs that come with enormous models.

References

[1] History of Autonomous Cars. (2022, January 27). TOMORROW'S WORLD TODAY®. https://www.tomorrowsworldtoday.com/2021/08/09/history-of-autonomous-cars/. Accessed 5 June 2022

[2] Autonomous / Self-driving Cars Market. (2022, January). Market Research Firm. Accessed 5 June 2022

[3] Transitioning to Tesla Vision. (2022, May 17). Tesla. https://www.tesla.com/support/transitioning-tesla-vision. Accessed 5 June 2022

[4] The German Traffic Sign Recognition Benchmark. (2010, December 1). German Traffic Sign Benchmarks. https://benchmark.ini.rub.de/gtsrb_news.html. Accessed 5 June 2022

[5] Image classification using machine learning. (2022, March 15). Analytics Vidhya. https://www.analyticsvidhya.com/blog/2022/01/image-classification-using-machine-learning/#h2_4. Accessed 6 June 2022

[6] Random Forest: Introduction to random forest algorithm. (2021, June 24). Analytics Vidhya. https://www.analyticsvidhya.com/blog/2021/06/understanding-random-forest/. Accessed 6 June 2022

[7] Gandhi, R. (2018, July 5). Support Vector Machine - introduction to machine learning algorithms. Medium. Towards Data Science. https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47. Accessed 6 June 2022

[8] SVM: Support Vector Machine Algorithm in machine learning. (2021, August 26). Analytics Vidhya. https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/. Accessed 6 June 2022

[9] Brownlee, J. (2020, June 30). Introduction to dimensionality reduction for machine learning. Machine Learning Mastery. https://machinelearningmastery.com/dimensionality-reduction-for-machine-learning/. Accessed 6 June 2022

[10] Kanan C, Cottrell GW (2012) Color-to-Grayscale: Does the Method Matter in Image Recognition? PLoS ONE 7(1): e29740. https://doi.org/10.1371/journal.pone.0029740

[11] Mishra, B.K., Thakker, D., Mazumdar, S. et al. A novel application of deep learning with image cropping: a smart city use case for flood monitoring. J Reliable Intell Environ 6, 51–61 (2020). https://doi.org/10.1007/s40860-020-00099-x

[12] Advantages of deep learning, plus use cases and examples. Width.ai. https://www.width.ai/post/advantages-of-deep-learning. Accessed 6 June 2022

[13] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, L. D. Jackel; Backpropagation Applied to Handwritten Zip Code Recognition. Neural Comput 1989; 1 (4): 541–551. doi: https://doi.org/10.1162/neco.1989.1.4.541

[14] Goodfellow, Ian; Bengio, Yoshua; Courville, Aaron (2016). "6.5 Back-Propagation and Other Differentiation Algorithms". Deep Learning. MIT Press. pp. 200–220. ISBN 9780262035613.

[15] LeCun, Yann A., et al. "Efficient backprop." Neural networks: Tricks of the trade. Springer Berlin Heidelberg, 2012. 9-48

[16] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg and Li Fei-Fei. ImageNet Large

Scale Visual Recognition Challenge. IJCV, 2015.

[17] Chollet, François. Xception: Deep Learning with Depthwise Separable Convolutions. arXiv. 2016.

[18] He, Kaiming and Zhang, Xiangyu and Ren, Shaoqing and Sun, Jian. Deep Residual Learning for Image Recognition. arXiv. 2015.

[19] Huang, Gao and Liu, Zhuang and van der Maaten, Laurens and Weinberger, Kilian Q. Densely Connected Convolutional Networks. arXiv. 2016.

[20] Tan, Mingxing and Le, Quoc V. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. arXiv. 2019.

[21] J. Li and Z. Wang, "Real-Time Traffic Sign Recognition Based on Efficient CNNs in the Wild," in IEEE Transactions on Intelligent Transportation Systems, vol. 20, no. 3, pp. 975-984, March 2019, doi: 10.1109/TITS.2018.2843815.

[22] Ameur Zaibi, Anis Ladgham, Anis Sakly, "A Lightweight Model for Traffic Sign Classification Based on Enhanced LeNet-5 Network", Journal of Sensors, vol. 2021, Article ID 8870529, 13 pages, 2021. https://doi.org/10.1155/2021/8870529/

[23] Simonyan, Karen and Zisserman, Andrew. Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv. 2014.

[24] Krizhevsky, Alex; Sutskever, Ilya; Hinton, Geoffrey E. (2017-05-24). "ImageNet classification with deep convolutional neural networks". Communications of the ACM. 60 (6): 84–90. doi:10.1145/3065386.

[25] Ioffe, Sergey and Szegedy, Christian. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. arXiv. 2015.

Code Appendix

The code used for this analysis can be found at this repository:
https://github.com/dam-robert/STA208-Final