Springboard Data Science Career Track

Capstone 2

Traffic Sign Image Recognition - Classifying Traffic Sign Images using Convolutional Neural Network

Derek Samsom

March 2019

Introduction

Background

Reading traffic signs is an important component of autonomous driving. Autonomous cars need to be able to see the road signs just for the same reasons humans need them. They allow the driver to know the speed they should be traveling, where to stop, and what to be cautious of on the road ahead. Automotive companies could use traffic sign recognition as an aid to human-driven cars as well. Sometimes a human driver can miss a sign alerting them to an upcoming curve or a speed limit change, and the car could remind the driver if it doesn't sense proper responses to signage, helping to make driving safer.

In a fully autonomous vehicle world, traffic signs as we know them wouldn't be necessary, as they could be replaced with by other means such as communicating with the other cars on the road as well as the road itself electronically. However, since autonomous cars will be sharing the road with human drivers, and humans need traffic signs, autonomous vehicles need to be able to read them.

Problem Statement:

Autonomous driving requires the ability to read traffic signs to enable the software to learn from the immediate environment and respond accordingly to information such as "stop", "yield", and "speed limit 50". This requires a very high level of accuracy, as there could be severe consequences if, for example, a stop sign was misinterpreted as an increase in speed limit sign.

This is an image classification problem, and the data will be used to train a convolutional neural network to make the predictions.

Data

The data to be used for this project comes from the Institute for Neuroinformatik in Germany. This data can be accessed at the INI website: http://benchmark.ini.rub.de/?section=gtsrb&subsection=news.

The data set contains 35,000 training images and 12,000 test images across 43 classes of German traffic signs. The images are 32 x 32 pixels and have 3 color channels. Signs include speed limits, stop signs, and various caution signs.

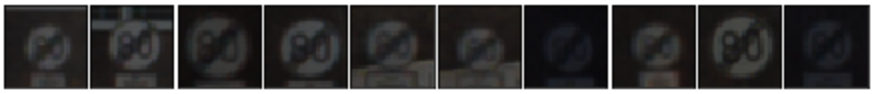
There is not a lot of data, especially given the number of classes.

Data Exploration

To get a feel for the image data, I will first plot some sample images. There are 43 classes, so I will exclude classes that are similar to other classes.

Class 0: Speed Limit 20 km/h

Class 6: End of Speed Limit 80 km/h

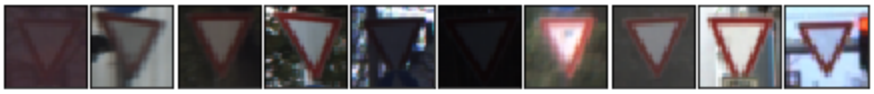


Class 9: No Passing

Class 11: Right-of-way at the next Intersection

Class 12: Priority Road

Class 13: Yield



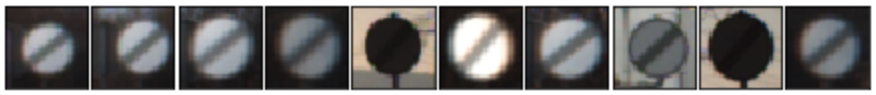
Class 14: Stop

Class 15: No Vehicles

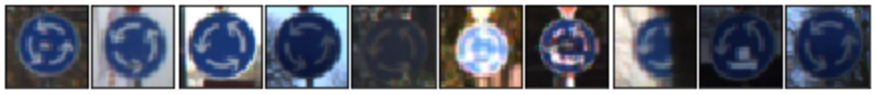
Class 17: No Entry

Class 18: General Caution

Class 32: End of all speed and passing limits



Class 40: Roundabout Mandatory



The images are small at only 32 x 32 pixels, and are fairly well centered with a small variation in rotation. The brightness varies considerably, with some images being almost too dark to see while others look washed out. The variation in brightness reflects how the signs would appear in the real world, since cars drive in conditions ranging from full sunlight to nighttime.

Class Imbalance

Next I want to plot the number of images per class see how well the classes are represented. Figure 1 shows the number of training images per class.

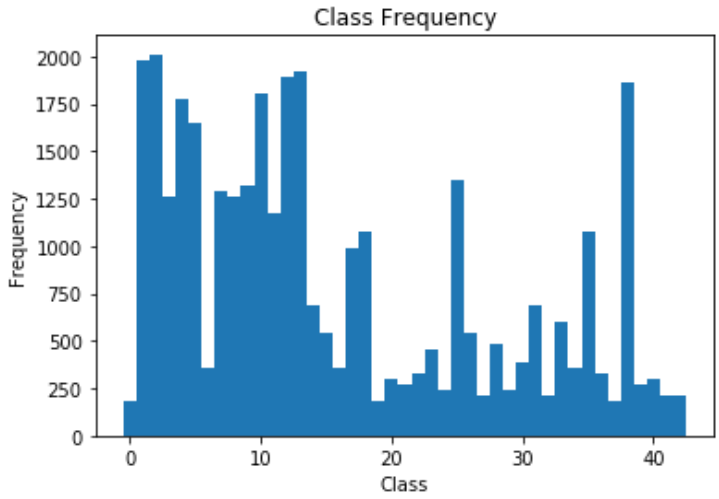


Figure 1: Class Frequency

There is a large range in the number of training samples per class, from as high as 2,010 images to as low as 180 images. Even the better represented classes with around 2,000

images do not provide a large amount of training data, and most have even less than that. I can upsample the classes to balance them out and give the model more training data.

Data Preprocessing

Neural Networks don't tend to need much preprocessing, however, there are a couple of steps I took. First, I converted the images to grayscale. There are several groups of signs that have the same color scheme, and the color would only help learn the type of sign rather than the specific sign. Also, much of the color variation is in the background, which adds some noise and won't be helpful in predicting the sign class.

The second preprocessing step taken was to create additional training images. I did this by sampling each class with replacement, and adding a random degree of rotation to the newly generated image between 5 and -5 degrees. The added rotation should help the model generalize better to unseen data, since the new images will mimic the expectation that if more data was gathered, it would differ slightly from the existing data.

There are more ways to slightly alter the newly generated images, such as changing the brightness and contrast, that could also help the model generalize better. It is also possible in some cases to flip images from one class vertically or horizontally, and use them for another class. For example, the left-turn sign images could be flipped across the vertical axis and would become training data for right-turn signage.

In upsampling the images, I will take more samples of the under-represented classes to balance them out. This should help the model learn the different classes at similar rates.

Model Architecture

I used three convolutional neural network architectures to classify images and built them using Tensorflow. Convolutional neural networks are commonly applied to imagery since they are able to extract features from the spatial relationships within an image. These features are generated using by moving small filters across an image to generate feature maps. This allows for pattern recognition, while also allowing some flexibility since the images in a class will have some variation, but similar patterns that can be learned.

Also adding to this flexibility the pooling method, which happens after the convolutions. Pooling shrinks down the images while preserving the important information. 2 x 2 pooling, for example, would reduce each dimension of the image by half, and each pixel in the new images are the maximum value of each 2x2 section of the image before pooling. This way, the model doesn't care exactly where the feature is fit, only that it is fit, and helps solve the problem of computers being overly literal.

Each convolutional layer should pick up higher level features than the last, and in each layer the features become more complex while the images become more compact.

After the convolutional layers, the output is flattened and passed through 1 or more fully connected layers, which look more like a traditional neural network. These layers typically use a rectified linear unit activation function, with the exception of the output layer in which no activation function is used.

The fully connected layers take the high-level filtered images, and translate them into votes for each class. These allow the model to learn more advanced combinations of features to help improve accuracy.

Architecture 1 - Setup

The initial model architecture I used is based on the LeNet-5 architecture, which was originated in the 1990s for recognizing handwritten digits. The architecture includes the following layers:

Layer 1: Convolutional Layer

Filter Size = 5 x 5
24 Filters
2 x 2 pooling
ReLU Activation

Layer 2: Convolutional Layer

Filter Size = 5 x 5
64 Filters
2 x 2 pooling
ReLU Activation

Layer 3: Fully Connected Layer

768 Outputs
ReLU Activation

Layer 4: Fully Connected Layer

512 Outputs
ReLU Activation

Layer 5: Fully Connected Layer - Output Layer

43 Outputs (1 for each class)
No Activation function used on output layer

The first convolutional layer is set to produce 24 images, or feature maps, and since 2x2 pooling is used, each of the 24 feature maps is 16 x 16 pixels. This results in 24 images of 16 x 16 images being passed to the 2nd convolutional layer.

The second convolutional layer is set up to produce 64 images, and since pooling is used, they are of size 8 x 8.

The flattening layer flattens the 64 images of 8 x 8 pixels into a 1 dimensional tensor of length 4096 (8X 8 X 64). These 1-dimensional arrays are passed through two fully connected layers of size 768 and 512, then finally pass to the output layer, which is size 43 (one for each class).

To update the weights during training, the Adam optimizer is used, using TensorFlows's Softmax cross entropy cost function. The neural network was trained using batches of 64 images.

Architecture 1 - Results

I ran 40,000 iterations with a batch size of 64 images per iteration. The overall accuracy on the test data after 40,000 iterations was 93.5%.

The confusion matrix in Figure 2 shows a strong diagonal line indicating the true-positive classifications. The confusion matrix is showing the number of classifications rather than the percentage. The test data is imbalanced like the training data, so some image classes have a higher number of classifications due to having a higher count of images.

Confusion Matrix - LeNet-5 Architecture

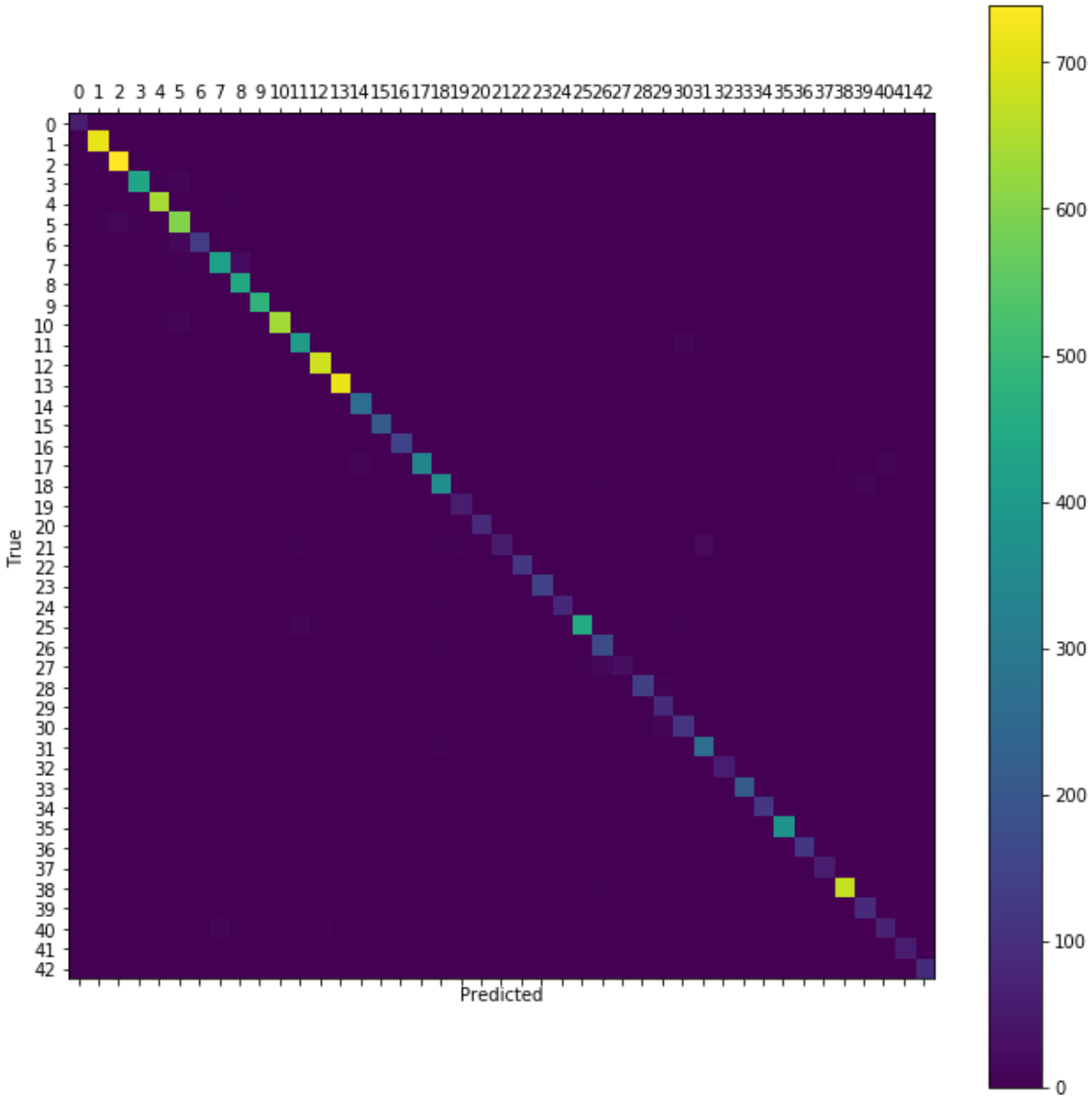


Figure 2: Confusion Matrix - Architecture 1

To get a better idea of how well the model is performing by class, the precision by class is shown in Figure 3, and the recall per class is shown in Figure 4.

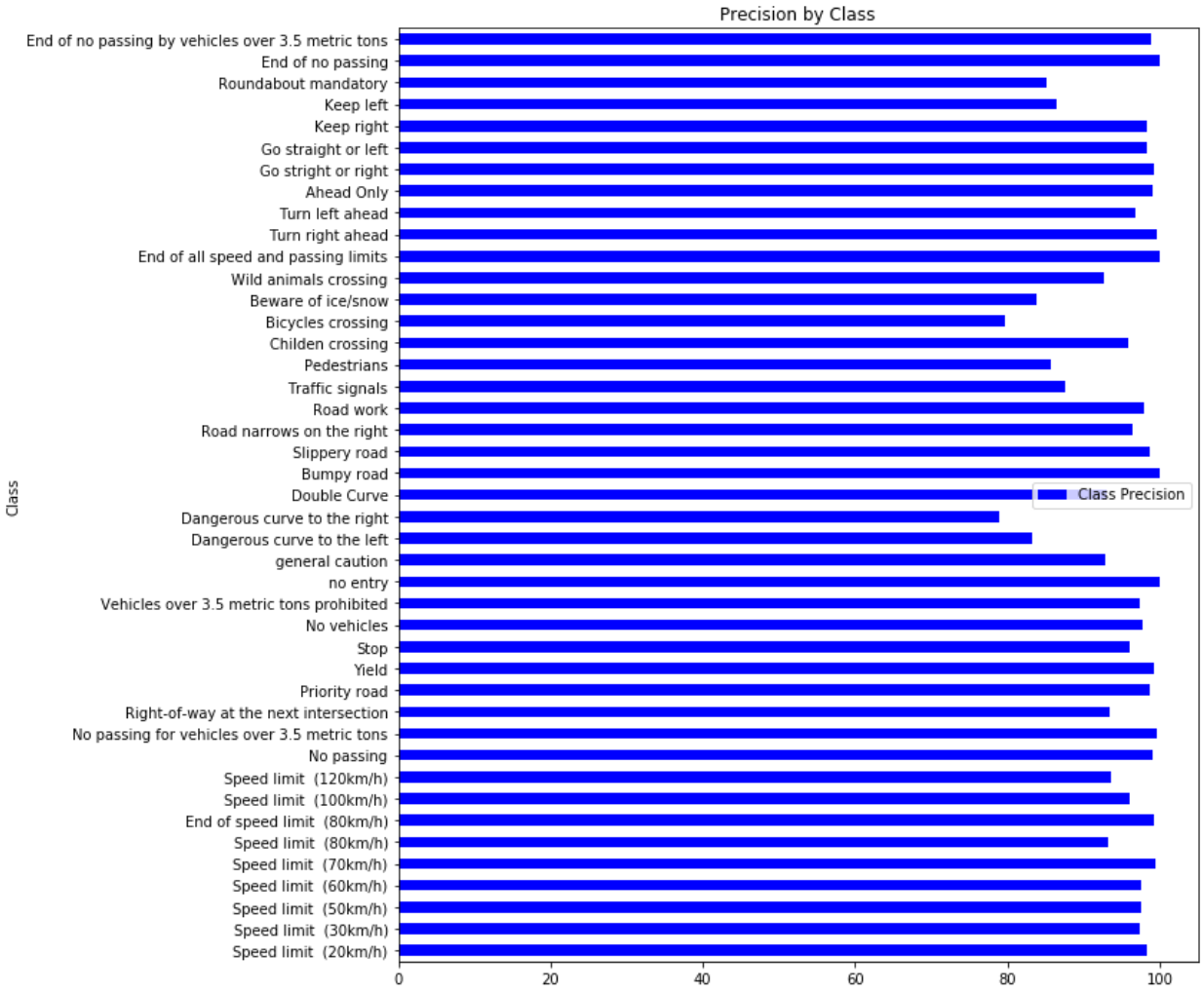


Figure 3: Precision by Class - Architecture 1

The signs “Dangerous Curve to the Right” and Bicycles Crossing” have the lowest precision at just under 80%, as shown in Figure 3. “Dangerous Curve to the Left” and “Beware of Ice/Snow” also have lower precision scores between 80% and 90%. The remaining classes have a precision score between 90% and 100%, with a few classes reaching 100%.

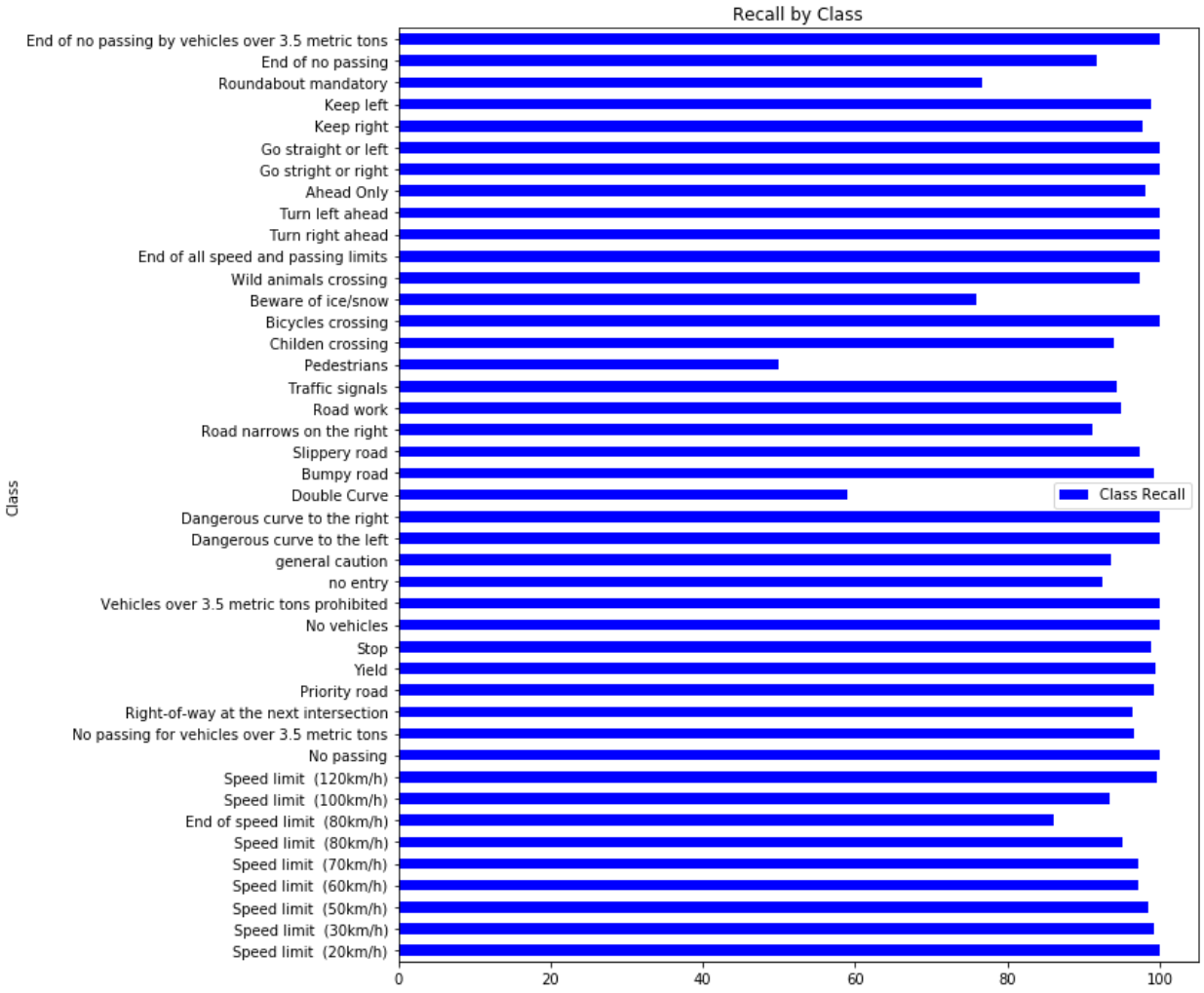


Figure 4: Recall by Class - Architecture 1

There are two classes, “Pedestrians” and “Double Curve”, that have a pretty poor recall between 40% and 60%. The model is also struggling, but to a lesser degree, with the “Roundabout Mandatory”, “Beware of Ice/Snow”, and “End of Speed Limit (80km/h)” signs having a recall between 70% and 90%. All other classes have a recall score above 90%, with a few reaching 100%.

Next I will try a new architecture and see if I can improve the results.

Architecture 2 - Setup

The second model architecture I used is based on the VGGNet architecture, which was first introduced in 2014. The VGGNet is much deeper than the LeNet-5 architecture, typically having 16-19 layers. VGGNet uses 3x3 filters, and 13 convolutional layers, with pooling occurring after

layers 2, 4, 7, 10, and 13. These layers are followed by 2 fully connected layers and the output layer. I adapted the architecture as follows:

Layer 1: Convolutional Layer

Filter Size = 3 x 3
64 Filters
No Pooling
ReLU Activation

Layer 2: Convolutional Layer

Filter Size = 3 x 3
64 Filters
2 x 2 pooling
ReLU Activation

Layer 3: Convolutional Layer

Filter Size = 3 x 3
128 Filters
No Pooling
ReLU Activation

Layer 4: Convolutional Layer

Filter Size = 3 x 3
128 Filters
2 x 2 pooling
ReLU Activation

Layer 5: Convolutional Layer

Filter Size = 3 x 3
256 Filters
No Pooling
ReLU Activation

Layer 6: Convolutional Layer

Filter Size = 3 x 3
256 Filters
2 x 2 pooling

ReLU Activation

Layer 7: Fully Connected Layer

2048 Outputs
ReLu Activation

Layer 8: Fully Connected Layer

1024 Outputs
ReLu Activation

Layer 9: Fully Connected Layer

512 Outputs
ReLu Activation

Layer 10: Fully Connected Layer - Output Layer

43 Outputs (1 for each class)
No Activation Function

Since I am working with smaller images, I was not able to make the network as deep due to the many pooling layers. Therefore, I simplified the VGGNet Architecture to use 3 x 3 filters, and 6 convolutional layers, with pooling occuring after the 2nd, 4th, and 6th and final convolutional layer.

The first and second convolutional layers have an output size of 64. This results in the second layer output being 64 filters of size 16 x 16, since the pooling on layer 2 reduces each dimension of the filtered images in half.

The 3rd and 4th convolutional layers have an output size of 128 images. With pooling used on the 4th layer, the output of layer 4 is 128 filtered images of size 8 x 8.

The 5th and 6th convolutional layers are the final convolutional layers, with pooling used after layer 6. These layers produce 256 filtered images of size 4 x 4. This 3rd pooling layer reducing the dimensions to 4 x 4 may be one too many, and for this reason I will not be able to add any more pooling layers and I will not add any more convolutional layers, so I will only have 6 convolutional layers rather that the 13 the VGGNet typically has.

The output of the 6th and final convolutional layer passes through a flattening layer. This layer is a 1 dimensional tensor of length 4096, which results from the input of 256 4 x 4 images (4 x 4 x 256).

The flattened layer passes through three fully connected layers of sizes 2018, 1096, and 512, followed by a final fully connected output layer of size 42 (one for each class). Except for the output layer, which should not use an activation function, rectified linear units are used on the fully connected layers.

As with the previous architecture, the Adam optimizer is used, using TensorFlows's Softmax cross entropy cost function. The neural network was trained using batches of 64 images.

Architecture 2 - Results

The second architecture, using the adapted VGGNet, had a final test accuracy of 96.0%, an improvement over the 93.5% reached using LeNet-5 in Architecture 1. The more complex architecture took a lot longer to train, but using an Amazon Web Services EC2 P2 instance helped speed things up.

The confusion matrix in Figure 5 shows that most of the predictions line up nicely in a diagonal line indicating correct predictions, with no obvious misclassifications.

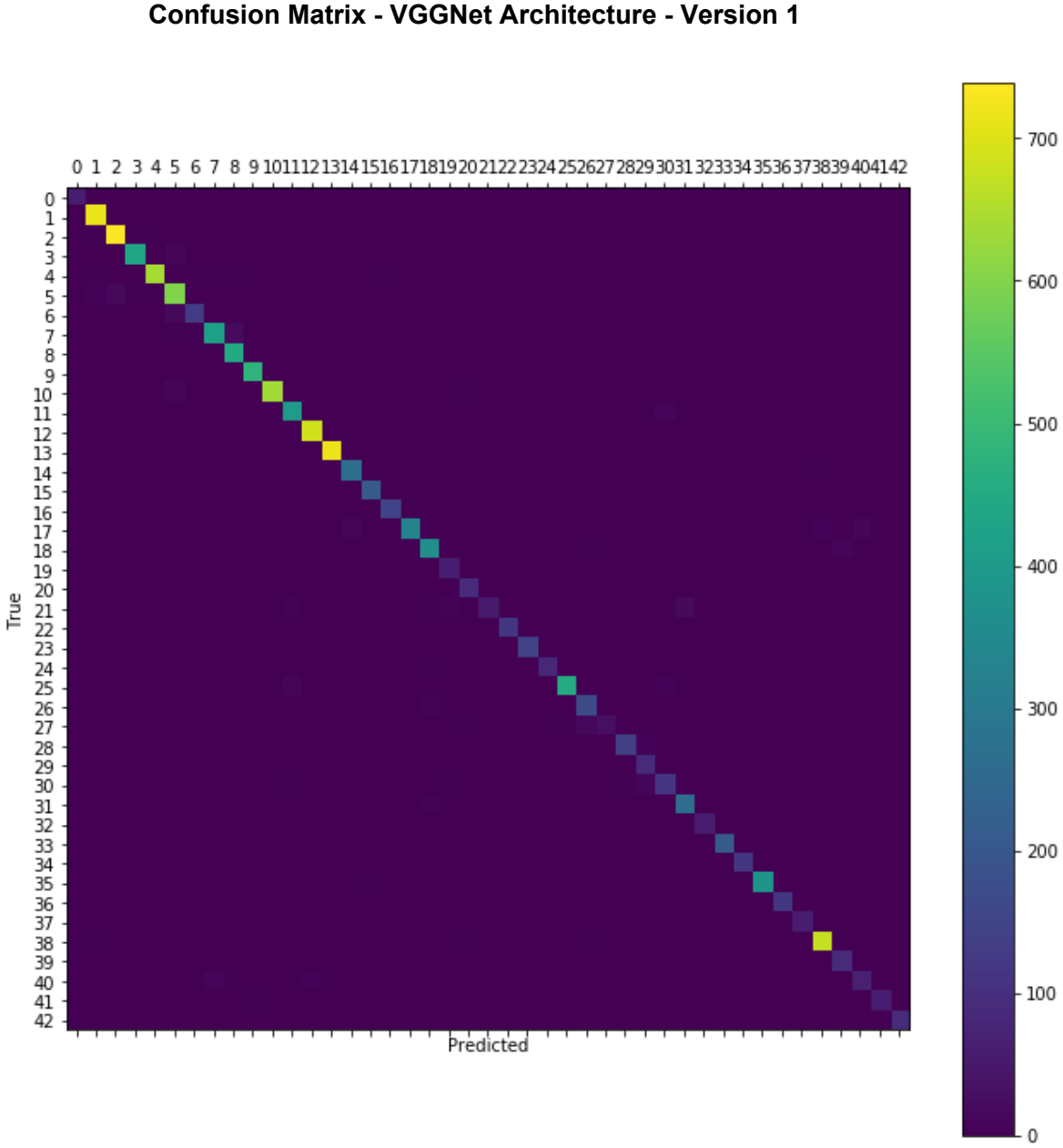


Figure 5: Confusion Matrix - Architecture 2

To get a better representation how well the model is doing for each class, I again calculated the precision and recall per class.

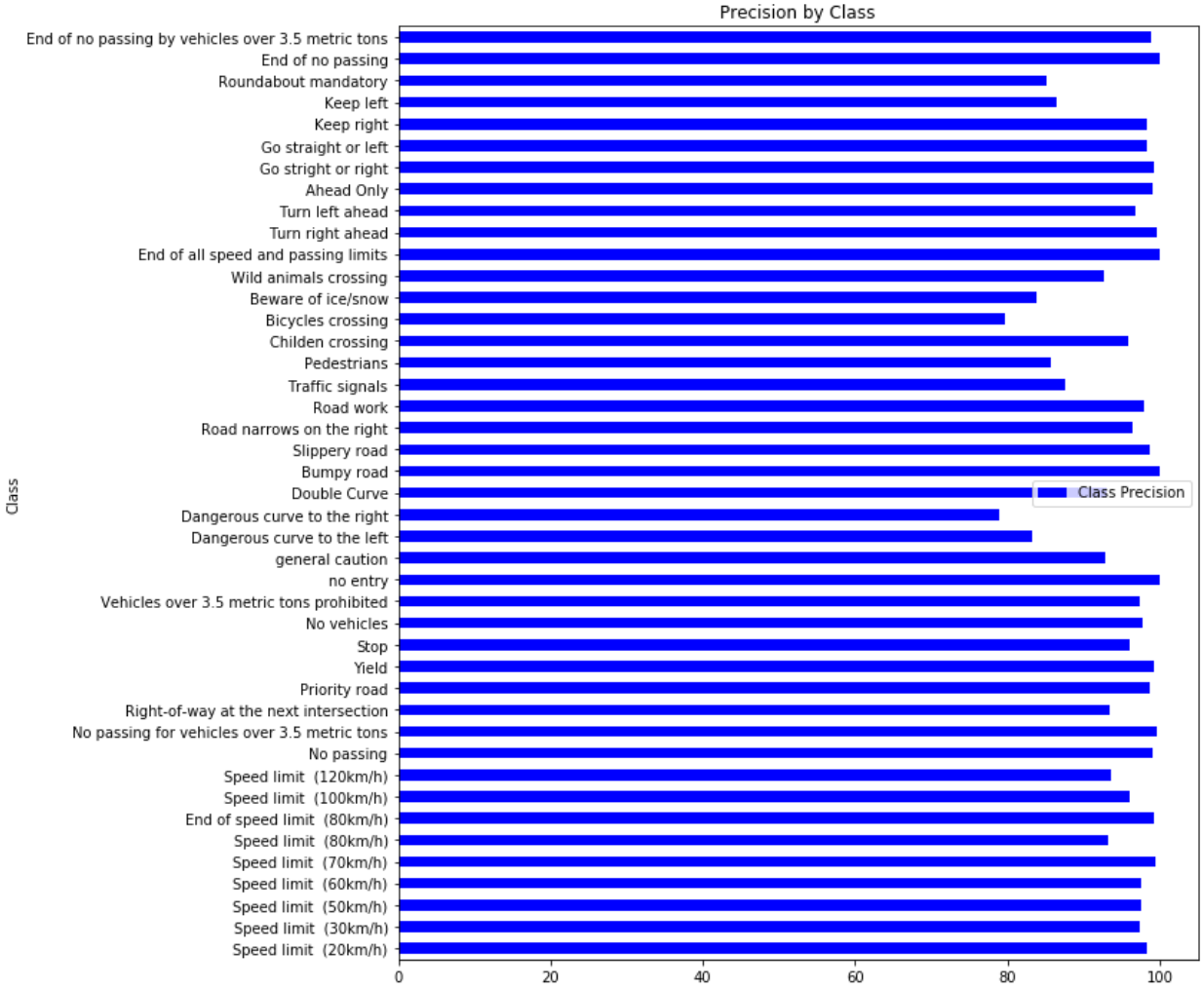


Figure 6: Precision by Class - Architecture 2

Figure 6 shows the second architecture, while improved over the first architecture, still had nearly the same issues with precision. The signs “Dangerous Curve to the Right” and Bicycles Crossing” have the lowest precision at just under 80%, just as in the first architecture. “Dangerous Curve to the Left” and “Beware of Ice/Snow” also have lower precision scores between 80% and 90%. The remaining classes have a precision score between 90% and 100%, with a few classes reaching 100%.

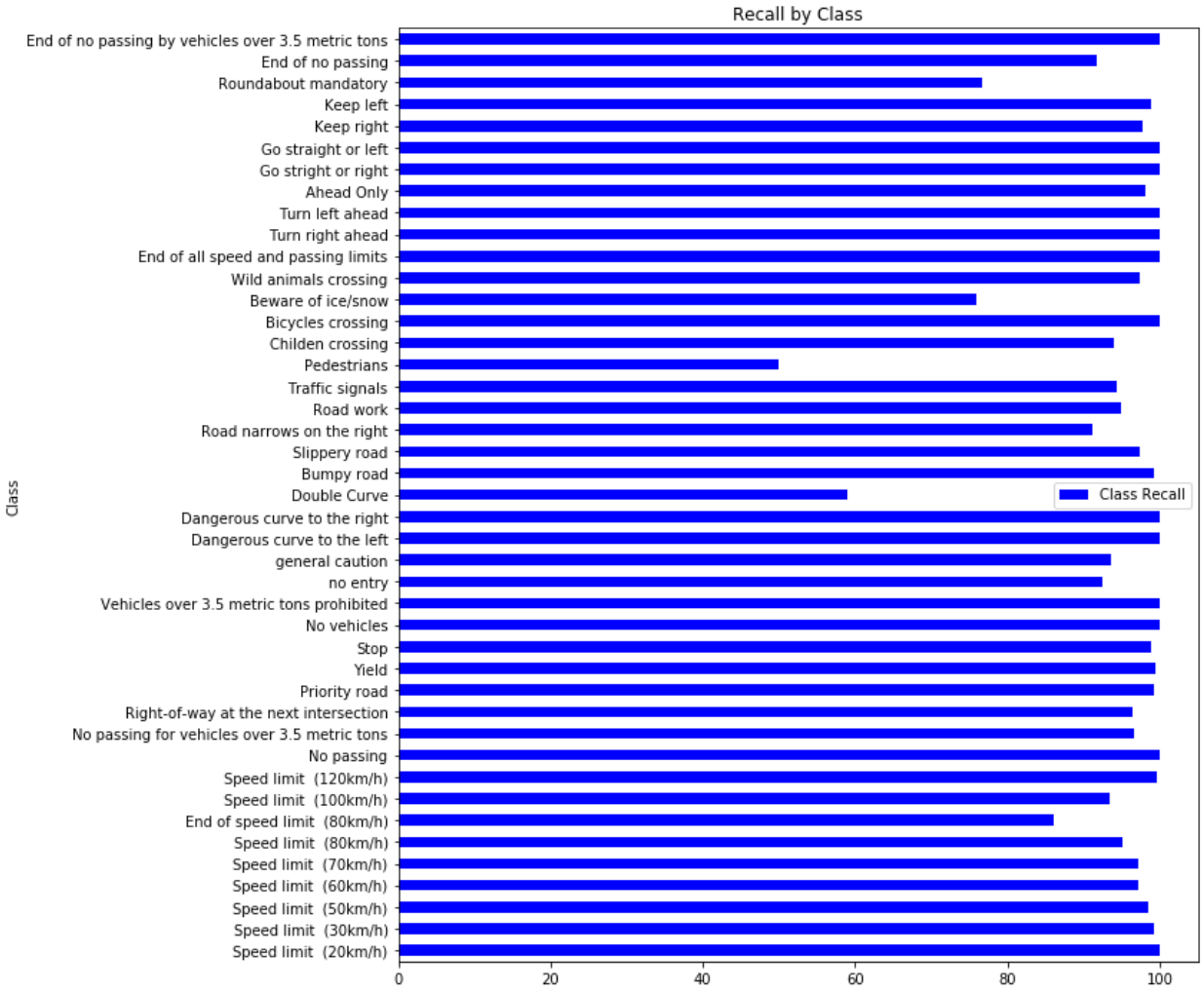


Figure 7: Recall by Class - Architecture 2

The recall per class shown in Figure 7 also shows a very similar pattern in the second architecture as was seen in the first. The recall was is below 60% for both “Pedestrians” and “Double Curve” signs, and there is still a low recall score for “Roundabout Mandatory”, “Beware of Ice/Snow”, and “End of Speed Limit (80km/h)” signs.

Architecture 3 - Setup

The 3rd and final architecture I used was meant to address the potential issue with the previous architecture with the third pooling layer reducing the filtered images to a size of 4 x 4. The third architecture is a VGGNet adaptation that is identical to the previous architecture, but I removed the 5th and 6th convolutional layers, which also removes the third pooling step on layer 6. This results in the output of the convolutional layers being 128 filtered images of size 8 x 8, rather than 256 images of size 4 x 4. The final architecture is as follows:

Layer 1: Convolutional Layer

Filter Size = 3 x 3
64 Filters
No Pooling
ReLU Activation

Layer 2: Convolutional Layer

Filter Size = 3 x 3
64 Filters
2 x 2 pooling
ReLU Activation

Layer 3: Convolutional Layer

Filter Size = 3 x 3
128 Filters
No Pooling
ReLU Activation

Layer 4: Convolutional Layer

Filter Size = 3 x 3
128 Filters
2 x 2 pooling
ReLU Activation

Layer 5: Fully Connected Layer

1024 Outputs
ReLu Activation

Layer 6: Fully Connected Layer

512 Outputs
ReLu Activation

Layer 7: Fully Connected Layer - Output Layer

43 Outputs (1 for each class)
No Activation Function

Again, the Adam optimizer was used, along with TensorFlows's Softmax cross entropy cost function. The neural network was trained using batches of 64 images.

Architecture 3 - Results

The third architecture was a small improvement over the 2nd, with an overall accuracy score of 97.1% after 40,000 training iterations, compared to 96.4 percent for the second architecture.

Again the confusion matrix for this architecture shown in Figure 8 does not show any surprises or obvious misclassifications.

Confusion Matrix - VGGNet Architecture - Version 2

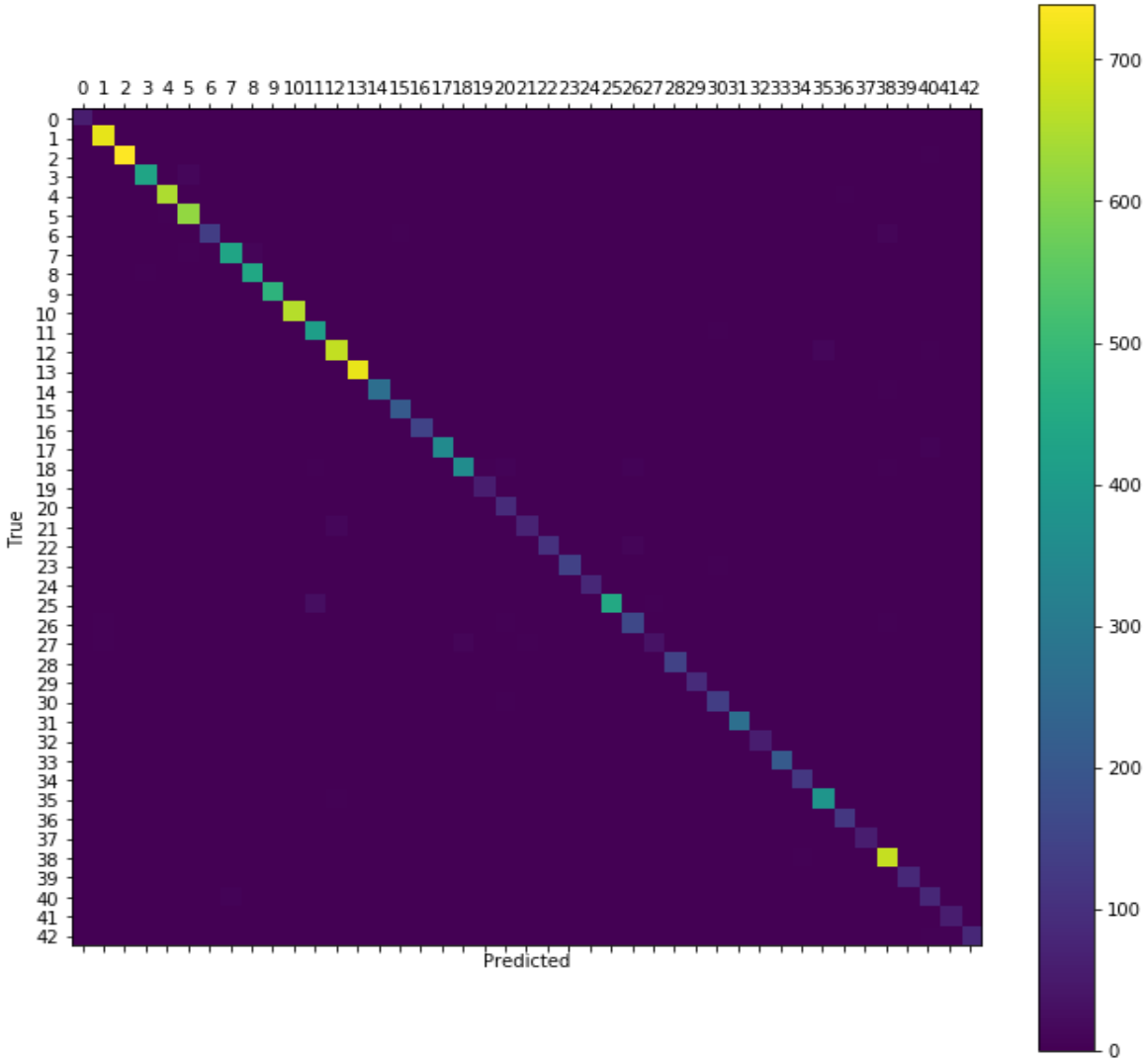


Figure 8: Confusion Matrix - Architecture 3

Next, I ‘ll look at the precision and recall by class for third architecture, and see if there were any improvements on the classes it struggled with on the first two models.

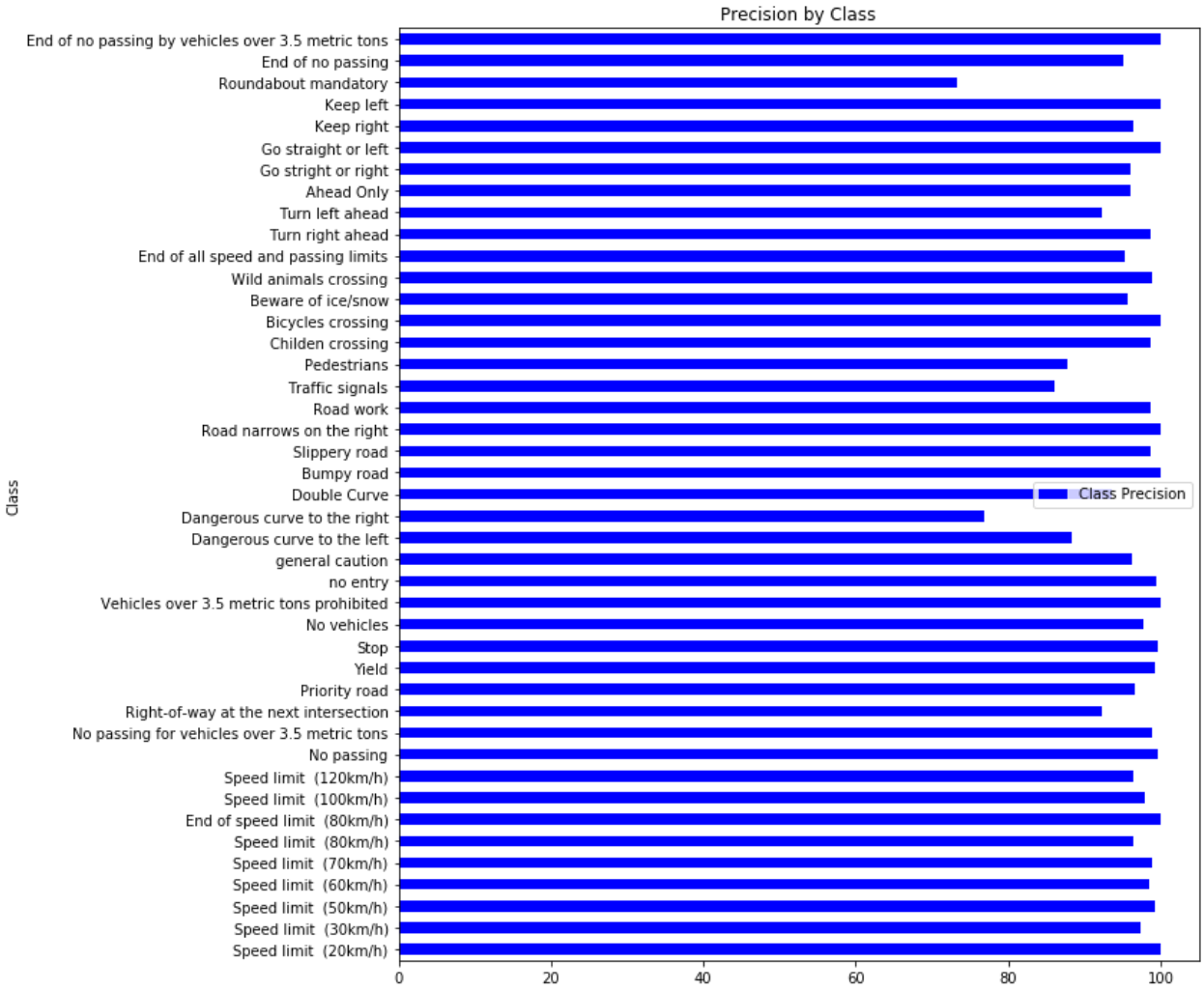


Figure 9: Precision by Class - Architecture 3

Figure 9 shows that the precision by class improved compared to the results of the previous model architectures. The previous architectures struggled most with the “Dangerous Curve to the Right” and “Bicycles Crossing” signs, resulting in a precision less than 80% for these classes. The third architecture is still struggling with precision on the “Dangerous Curve to the Right” signs, but improved to 100% precision on predicting “Bicycles Crossing” signs. Interestingly, this precision for “Roundabout Mandatory” dropped below 80% for this model, while it was higher in the previous models.

Overall, the third architecture has the highest average precision, but there are still a couple classes it isn’t able to do as well on.

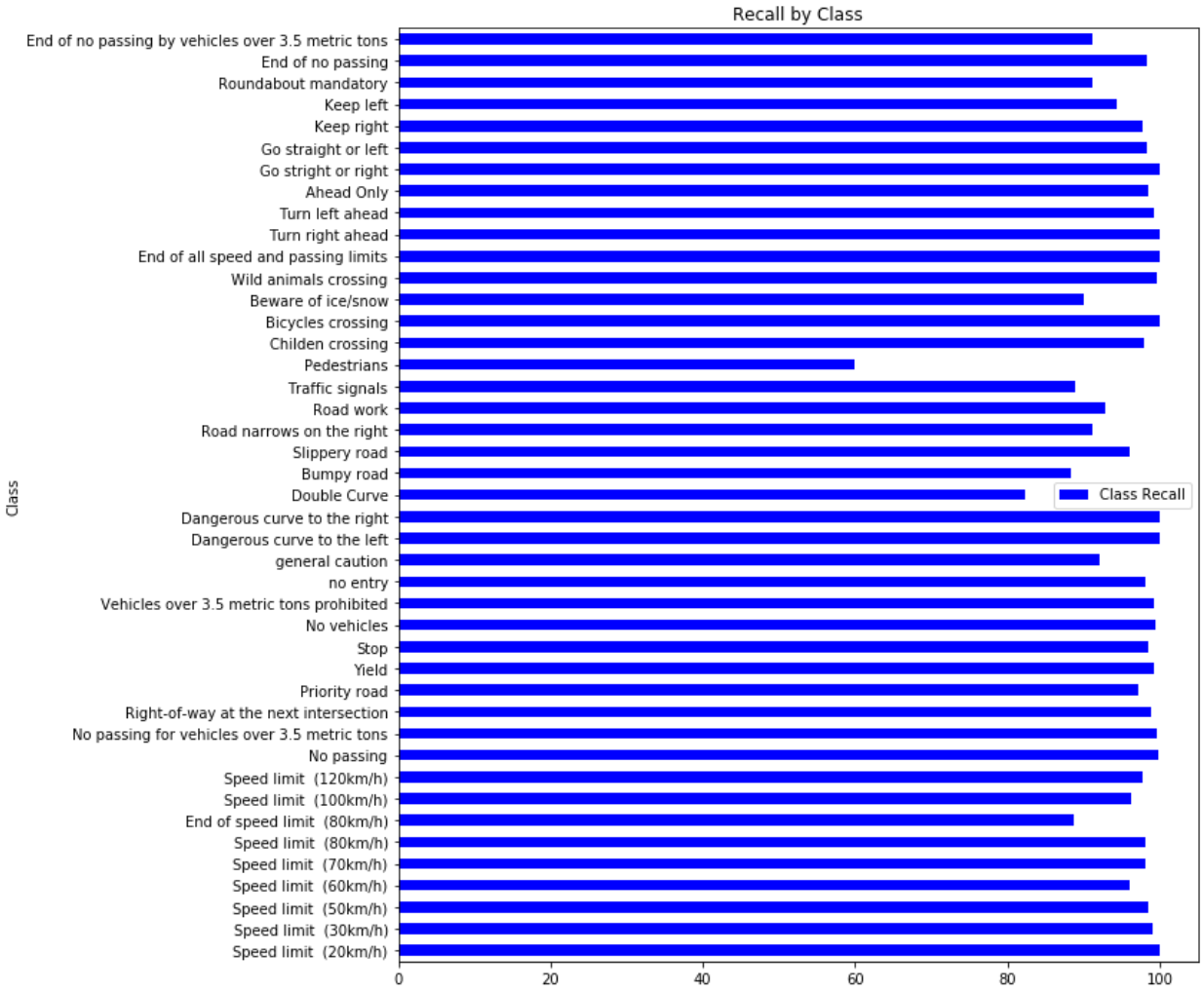


Figure 10: Recall By Class - Architecture 3

The recall per class shown in Figure 10 shows that the third architecture is still struggling with precision on “Pedestrians” signs, but there is a modest improvement with this class from around 50% precision to nearly 60% precision. Precision for “Double Curve” signs improved quite a bit from below 60% to just above 80%.

Overall, this is the best performing model on recall per class, but still concerning that only around 60% of “Pedestrian” signs are classified as such.

Summary

Overall, the VGGNet architecture is performing the best, and for the 32x32 images used in this project, there was a slight advantage in the second version only using 4 convolutional layers instead of 6 to limit the amount of pooling layers.

Table 1 summarizes the test results for the architectures used and the number of training iterations on the left hand side, using the original training images plus the additional images that were samples with random rotation applied. The VGGNet adaptations performed better than the simpler LeNet-5 architecture, with the 2nd version having the highest training accuracy I obtained at 97.1%.

I also experimented with training the same three models on just the original training images, without adding the sampled images with random rotation. The results of this training experiment is on the right side of Table 1. Using the more limited number of original images did surprisingly well, and resulted in higher performance on the LeNet-5 and the 6-convolutional-layer 1st version of the VGGNet architecture. The 4-convolutional-layer 2nd version of the VGGNet architecture is the only one that improved with the additional images, this was the best performing combination of all I tried.

Test Results

	With Additional Training Images				Original Images Only		
Training Iterations	**LeNet-5**	**VGGNet Version1**	**VGGNet Version2**		**LeNet-5**	**VGGNet Version1**	**VGGNet Version2**
100	18.3	22.0	24.9		31.9	30.5	40.1
1000	70.6	75.6	81.9		74.7	73.9	85.5
5000	86.7	92.0	88.2		88.2	90.6	90.9
10000	89.4	93.5	94.4		91.4	92.9	95.7
20000	93.2	95.9	96.5		94	96.4	96.8
30000	93.5	95.2	96.9		94	96.7	96.5
40000	93.5	96.1	97.1		94.4	96.7	96.5

Table 1: Test Results

Future Work

There are numerous ways to try to improve the results of the models. The main thing holding it back is training time, as it is part intuition and part trial and error in creating the best-performing model. The two major areas to try to get some performance improvements are in the data preprocessing and in the hyperparameter tuning.

For preprocessing, further work could be done to try to increase the amount and quality of the training data. Collecting more real data images would be a great help, but there are some additional steps that could be done with the existing data. First, some of these images can inherently double as an image of the same or another class just by flipping them vertically and/or horizontally. Flipping the “left turn ahead” sign on the vertical axis creates a “right turn ahead” training image, and flipping the “no vehicles” sign, which is just a white circular sign with a red border, both vertically and horizontally creates new images of the same class.

Another preprocessing step that I suspect would be helpful would be to remove the background of the images, since this is just noise that wastes training resources on information that doesn’t offer any predictive value.

Other pre-processing steps that may also help include histogram equalization to help even out the contrast and brightness, as well as adding more variance to any generated images in addition to the random rotation I applied. This can include introducing random brightening and darkening factors and adding a random skew to the image.

Further work on hyperparameter tuning would include trying out different combinations of filter size, number of filters, number of layers, as well as the learning rate and training batch size.

Recommendations

The convolutional networks are learning the images pretty well given the small size of the data set, however, autonomous cars need an even higher accuracy to be able to safely drive themselves.

For the 32 x 32 images, the adapted VGGNet works the best when it is trimmed down to 4 convolutional networks with pooling on layers 2 and 4. The VGGNet should work even better with larger images, since this would allow more convolutional layers without downsampling the images too far in the pooling layers.

To achieve the level of accuracy that autonomous driving would need to operate safely, I would recommend using larger images so that more convolutional layers can be used. This will allow for the VGGNet to go deeper than the 32 x 32 images allow, as it was designed to do. I would

also recommend obtaining more images, which will allow the network to train for all of the possibilities.