Definition

(approximately 1 - 2 pages)

Project Overview

In this section, look to provide a high-level overview of the project in layman's terms. Questions to ask yourself when writing this section:

- *Has an overview of the project been provided, such as the problem domain, project origin, and related datasets or input data?*
- *Has enough background information been given so that an uninformed reader would understand the problem domain and following problem statement?*

In this project I will develop a proof of concept solution that can be used in building an automatic train control system. Although there are solutions that are developed in the rail industry for self-driving vehicles, the existing solutions for driverless trains and driver assistance usually requires dedicated infrastructure, signalling systems, and specific rolling stock, all of which requires significant investment. The idea behind this project is to build a proof of concept solution that would use image recognition with relatively cheap hardware and open source software to recognise signals along the train track, and to use it to control the behaviour of a train set. The input data for this project will be collected by a camera that will be mounted on top of a train set. Images will first be used to train an artificial neural network to recognize images, and in a later phase of this project real-time data will be fed to the neural network to do image recognition on the fly with the trained model.

Problem Statement

In this section, you will want to clearly define the problem that you are trying to solve, including the strategy (outline of tasks) you will use to achieve the desired solution. You should also thoroughly discuss what the intended solution will be for this problem. Questions to ask yourself when writing this section:

- *Is the problem statement clearly defined? Will the reader understand what you are expecting to solve?*
- *Have you thoroughly discussed how you will attempt to solve the problem?*
- *Is an anticipated solution clearly defined? Will the reader understand what results you are looking for?*

For this project I will use a toy train set and a Raspberry Pi with a camera. The toy train will use image recognition to recognize a stop signal along the train track, which can then be used to automatically stop the train. I will use the Raspberry Pi module mounted on the train to feed images to a computer. A neural network will be developed to do the image recognition. The network will predict if there is a stop sign on the image seen. This prediction then can be used by the Raspberry Pi to stop the train, or to start it again if the stop sign disappears.

Metrics

In this section, you will need to clearly define the metrics or calculations you will use to measure performance of a model or result in your project. These calculations and metrics should be justified based on the characteristics of the problem and problem domain. Questions to ask yourself when writing this section:

- *Are the metrics you've chosen to measure the performance of your models clearly discussed and defined?*
- *Have you provided reasonable justification for the metrics chosen based on the problem and solution?*

For the training and for the validation data I have used the accuracy for measuring performance. Accuracy is simply the percentage of correctly identified images. Given that I had an equal amount of examples in both categories accuracy gives a good measure of model performance.[1] However for the final evaluation on the test set I have used an alternative metric called Recall. Recall is the number of correctly identified stop signs divided by the number of correctly identified stop signs and the number of misclassified stop signs. In other words the Recall score captures the ability of the model to detect all of the stop signals. Using the Recall score for evaluating the final model was important because from the practical point of view we prefer to detect all of the stop signs, while misclassifying non-stop as a stop is less crucial (i.e. we want the train to stop at all instances when there is a stop sign to prevent accidents, whereas stopping 'accidentally' by detecting a stop sign where there is none is not such a big problem).

Analysis

(approximately 2 - 4 pages)

Data Exploration

In this section, you will be expected to analyze the data you are using for the problem. This data can either be in the form of a dataset (or datasets), input data (or input files), or even an environment. The type of data should be thoroughly described and, if possible, have basic statistics and information presented (such as discussion of input features or defining characteristics about the input or environment). Any abnormalities or interesting qualities about the data that may need to be addressed have been identified (such as features that need to be transformed or the possibility of outliers). Questions to ask yourself when writing this section:

- *If a dataset is present for this problem, have you thoroughly discussed certain features about the dataset? Has a data sample been provided to the reader?*
- *If a dataset is present for this problem, are statistics about the dataset calculated and reported? Have any relevant results from this calculation been discussed?*
- *If a dataset is* ***not*** *present for this problem, has discussion been made about the input space or input data for your problem?*
- *Are there any abnormalities or characteristics about the input space or dataset that need to be addressed? (categorical variables, missing values, outliers, etc.)*

The data for this project was collected by myself. I have used the Raspberry Pi's camera module to take images. There were two categories I needed to collect images for. One category contained a stop sign, the other category did not. The challenge for the data collection was to have images that are similar to each other except for one thing, the presence or absence of the stop sign. I solved this by moving the Raspberry Pi along a fixed path on a platform, simulating how the train would actually move, while capturing images. The platform was extending out in front of the camera and I could mount the stop sign on it when capturing images. In the other condition I left the platform empty. In this way everything was constant except for the presence or absence of the stop sign.

[1] Also, for the time being Keras does not support other metrics than accuracy, and it would have required significant work-around to calculate the Recall score during training and validation.

I have collected 400 images for each category and later used 200 images for training, 100 images for validation, and 100 for testing for each category. The Raspberry Pi camera had a 5 MP resolution camera, but I decided to capture images at 320x160 pixels, as this size would allow for online streaming and processing in later development phases of the project. Image resolution could be set easily on the Raspberry Pi, so my initial guess was to use this resolution, as I have seen in a similar application[2] that this size allows for low latency video streaming. For additional image pre-processing I have used a convenient built in function in Keras, which I will discuss later.

Given that for recognizing stop signs the colour of the sign is crucial I have used RGB images (3 channels).

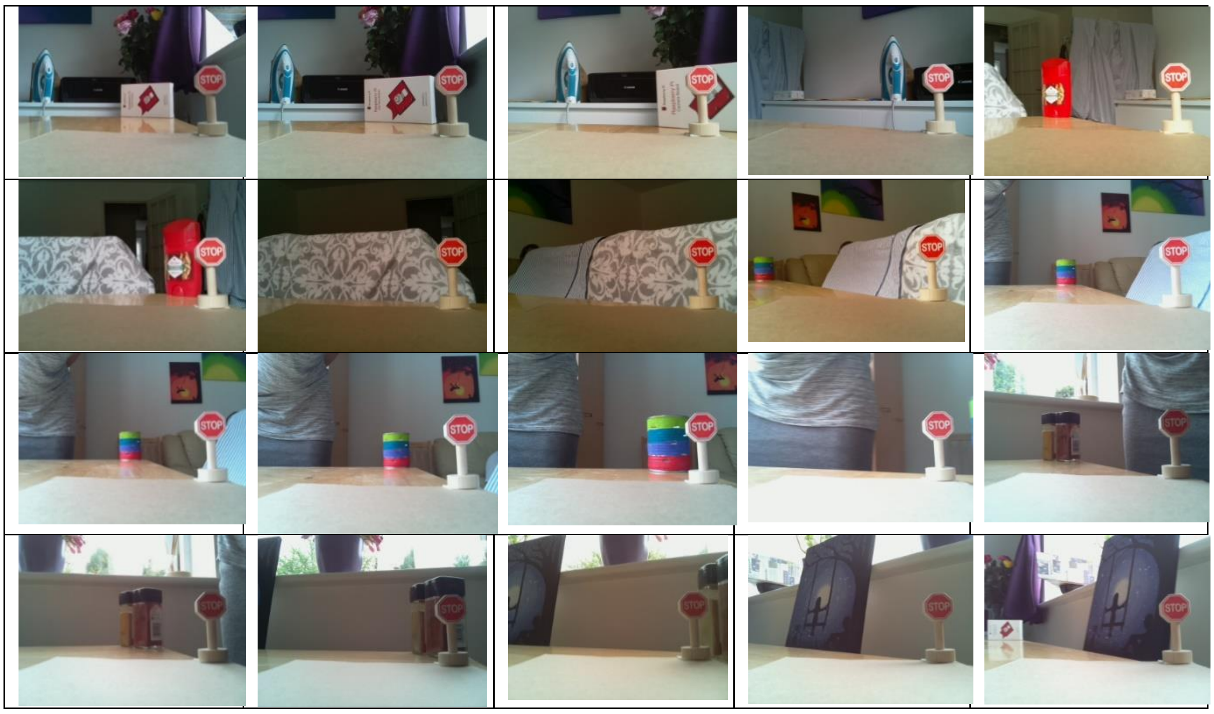
Exploratory Visualization

In this section, you will need to provide some form of visualization that summarizes or extracts a relevant characteristic or feature about the data. The visualization should adequately support the data being used. Discuss why this visualization was chosen and how it is relevant. Questions to ask yourself when writing this section:

- *Have you visualized a relevant characteristic or feature about the dataset or input data?*
- *Is the visualization thoroughly analyzed and discussed?*
- *If a plot is provided, are the axes, title, and datum clearly defined?*

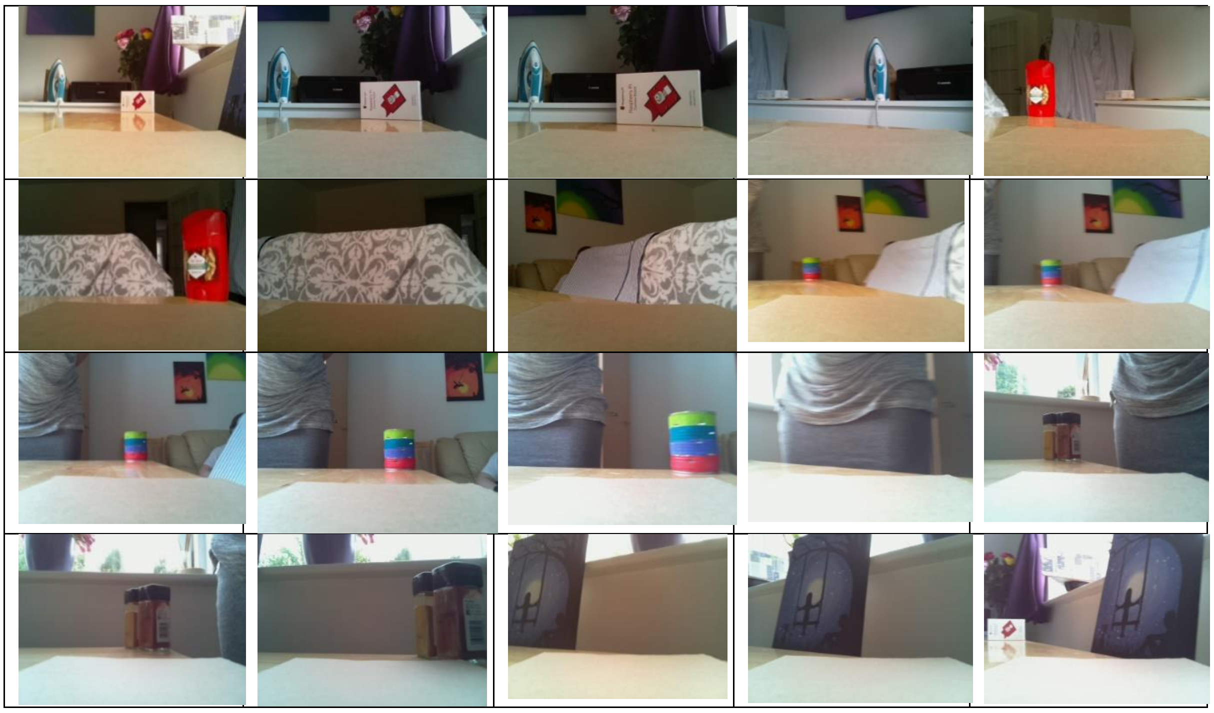
Below I have highlighted some example images, covering the path that the camera was moving on in both conditions.

Stop:

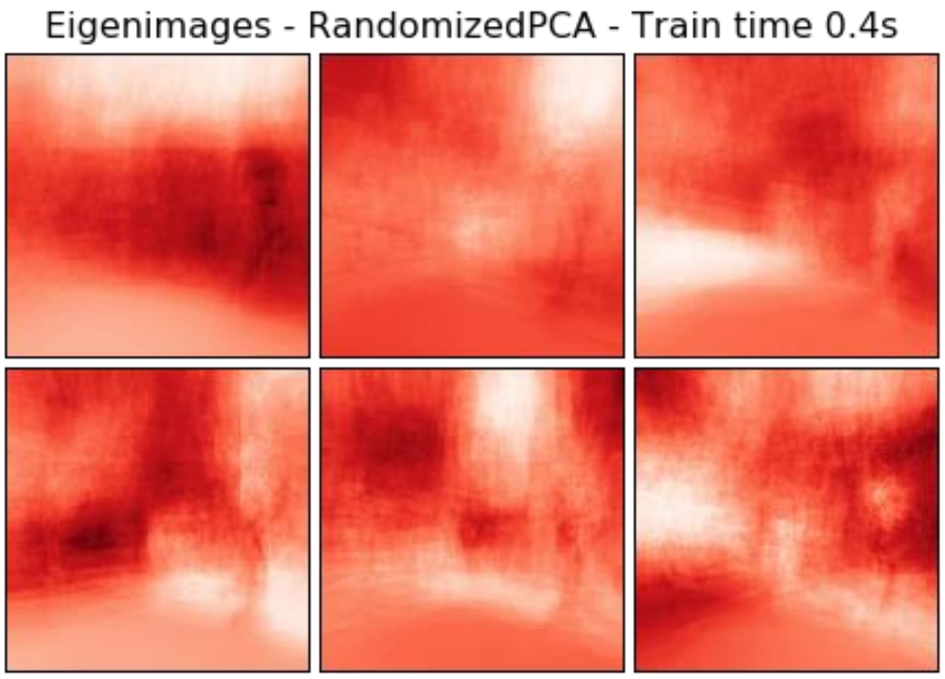


Non-stop:

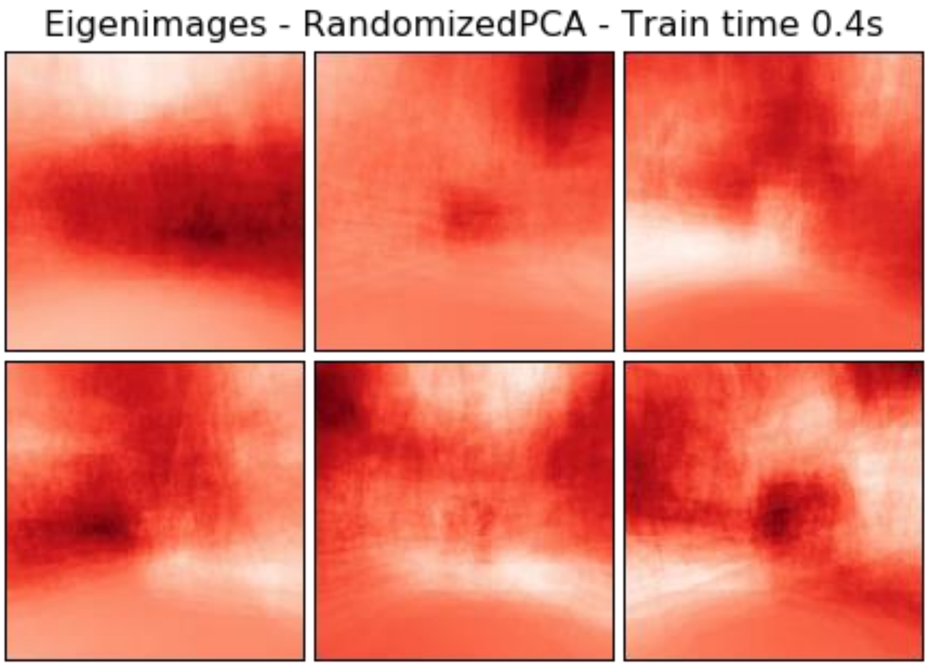
[2] https://zhengludwig.wordpress.com/projects/self-driving-rc-car/



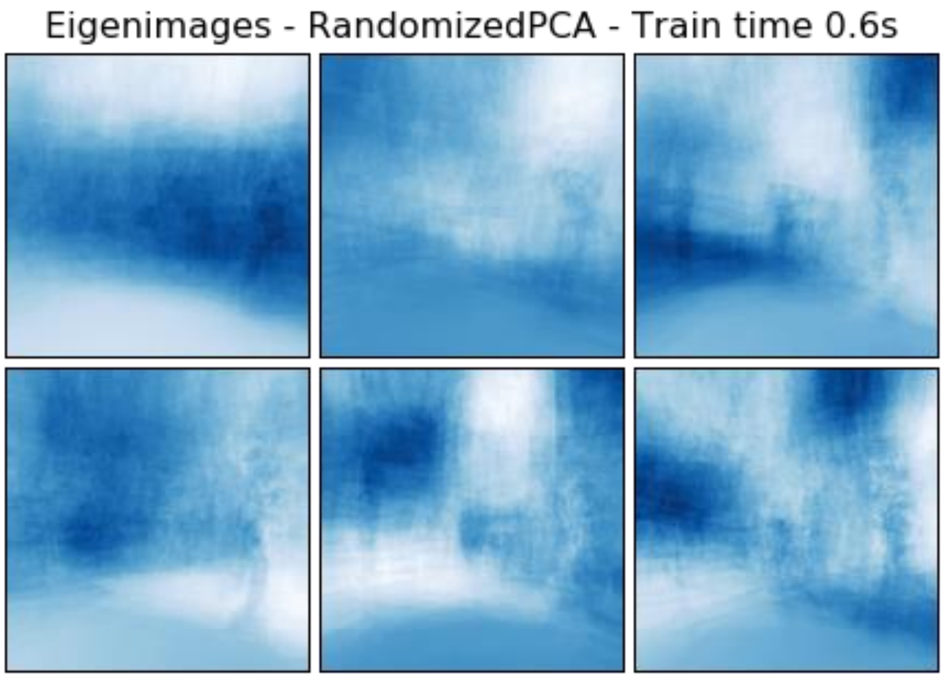
In addition, I run a principal component analysis (PCA) on the images to visualize a decomposed representation of the images in the two conditions. I did it separately for the three channels (RBG). Below you can see the first six components for the two conditions and three channels respectively.



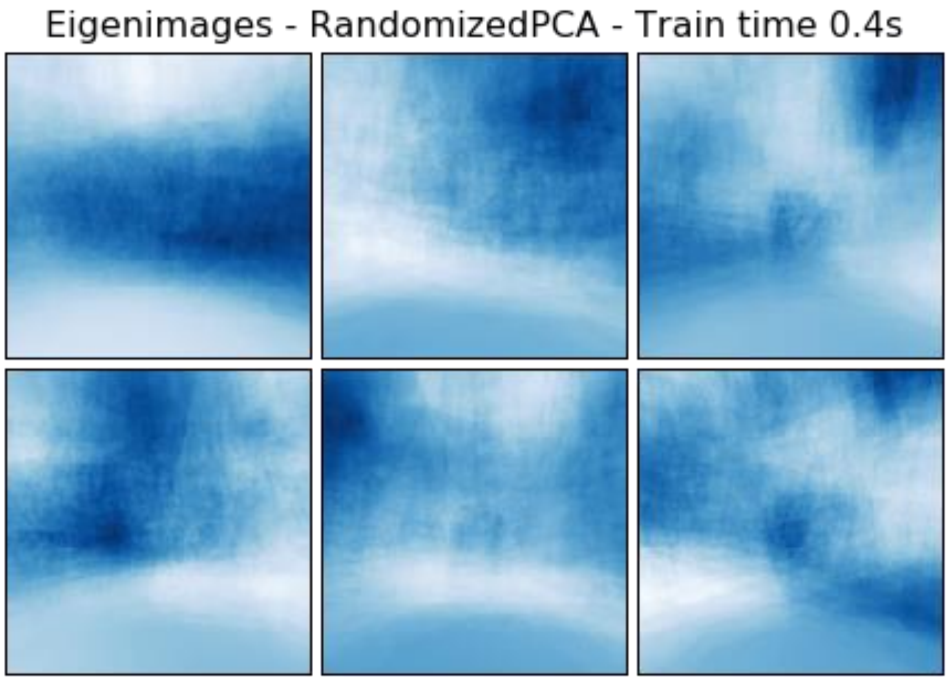
Stop



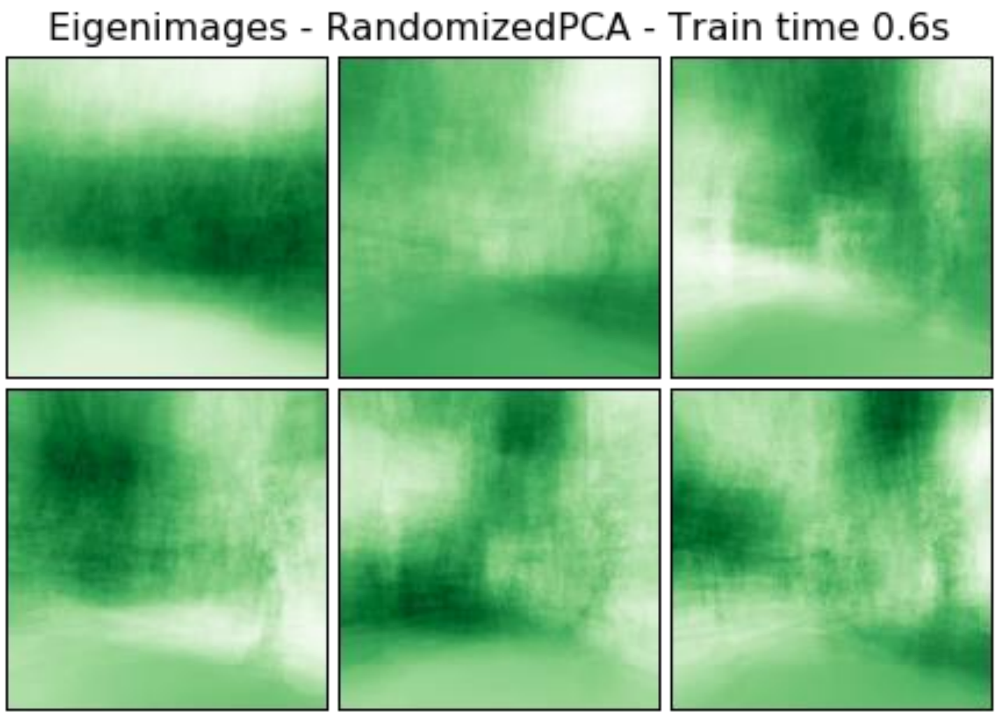
Non-stop



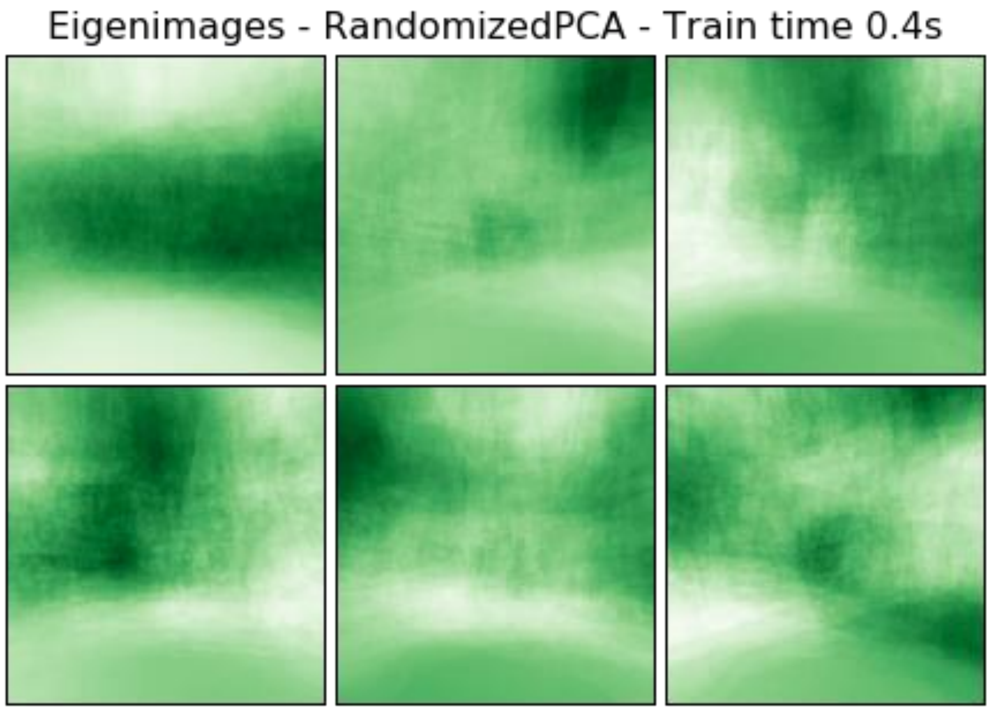
Stop



Non-stop



Stop



Non-stop

As we can see the principal components look very similar in both categories, which is important as it indicates that I have managed to gather a balanced dataset. This means that images in both categories were very similar to each other except for the presence or absence of the crucial feature that needed to be identified. I believe that the PCA on the Red channel for the 'stop category' has highlighted it in the shape of a red circular patch where the stop sign usually occurred. We can see this on the first component in the upper left corner of the Red images of the 'stop category'.

Algorithms and Techniques

In this section, you will need to discuss the algorithms and techniques you intend to use for solving the problem. You should justify the use of each one based on the characteristics of the problem and the problem domain. Questions to ask yourself when writing this section:

- *Are the algorithms you will use, including any default variables/parameters in the project clearly defined?*
- *Are the techniques to be used thoroughly discussed and justified?*
- *Is it made clear how the input data or datasets will be handled by the algorithms and techniques chosen?*

I used a deep convolutional neural net to solve this problem as this is the best approach known today to do image recognition.

Categorizing images is a difficult task for computers as the input space is generally large. For instance, a greyscale image (1 channel) of size 150x150 would be transformed into a vector of size 150x150=22500 for a fully connected neural network.

In my implementation I initially wanted to use my 320x160 colour (3 channels) images. However with image size 320x160 my model was not compiling. I have figured out that if I rescale the image to have equal height and width the model compiles[3], so I specified in my data augmentation method to rescale the images to 150x150 (3 channels).

In my initial model I have applied 3 convolutional layers, with 3x3 convolutions and 32 output filters applied to each followed by pooling layers of 2x2. This was followed by flattening the last pooling layer followed by a fully connected layer, dropout, and a final single output neuron to perform the binary classification task.

With regards to optimizers, I have tried Adam and RMSPROP, both of which are using adaptive learning-rate methods, which ensures fast convergence[4]. In comparison to SGD they have the benefit that one does not have to worry about tuning the learning rate, and one can likely achieve best results with default values.

I took advantage of a built in feature of Keras for generating batches of augmented data from a directory. I only had to copy the images for the two categories into different directories, and Keras took care of data augmentation and applying labels for the binary classification.

Data augmentation was performed for both categories for the training set, but not for the validation or test sets. The specifics of the data augmentation will be described in the Methodology/Data Preprocessing section.

[3] I coudn't figure out why this is so, and it was not crucial to stick with that particular resolution, so I went on with applying the rescaling.
[4] http://sebastianruder.com/optimizing-gradient-descent/index.html#whichoptimizertochoose

Data augmentation is useful when one has only a small sample of images. Augmentation consists of small transformations applied to the image, which helps prevent overfitting, as we have a larger training set to learn from. The specific transformations I have applied to the images were: rescaling, rotation, shifting width and height, shearing, and zooming. Below you can see some examples:

I had to fine tune with how much to transform, so that the stop sign remains always on the image. If too much width/height shift, or zoom is applied, than it can happen that the stop sign disappears from the picture.

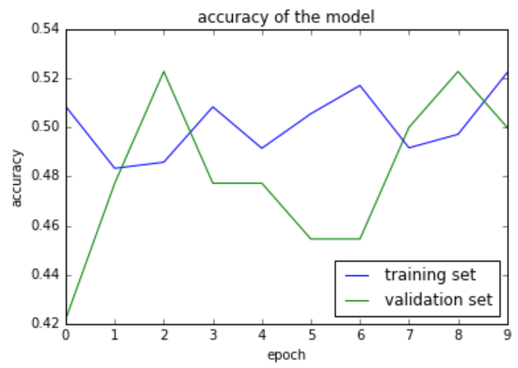
Benchmark

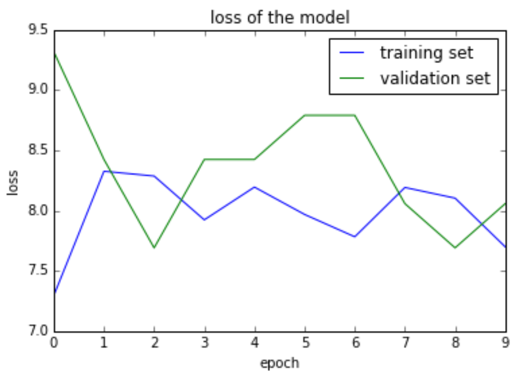
In this section, you will need to provide a clearly defined benchmark result or threshold for comparing across performances obtained by your solution. The reasoning behind the benchmark (in the case where it is not an established result) should be discussed. Questions to ask yourself when writing this section:

- *Has some result or value been provided that acts as a benchmark for measuring performance?*
- *Is it clear how this result or value was obtained (whether by data or by hypothesis)?*

For this task the benchmark was accuracy, and I aimed for an accuracy of at least 95%, as I have found this level of performance to be achievable even in multi category classification for traffic signs (e.g. http://yann.lecun.com/exdb/publis/pdf/sermanet-ijcnn-11.pdf).

I have also run a baseline model with a minimalistic architecture using a single convolutional layer, and an additional hidden layer with 32 neurons, without regularization (i.e. dropout). The performance of this baseline model looked like the following:





Methodology

(approximately 3 - 5 pages)

Data Preprocessing

In this section, all of your preprocessing steps will need to be clearly documented, if any were necessary. From the previous section, any of the abnormalities or characteristics that you identified about the dataset will be addressed and corrected here. Questions to ask yourself when writing this section:

- *If the algorithms chosen require preprocessing steps like feature selection or feature transformations, have they been properly documented?*
- *Based on the **Data Exploration** section, if there were abnormalities or characteristics that needed to be addressed, have they been properly corrected?*
- *If no preprocessing is needed, has it been made clear why?*

Data preprocessing consisted in reducing the resolution of the images (such that it is manageable to train a neural net in reasonable time even on a simple laptop with CPU, and to allow for online streaming and processing in the future), and in image transformations. Images were captured with size 320x240 pixels. These were further scaled to 150x150 pixels by Keras. Transformation of images were done with the following parameters:

- rescale=1./255: as the images taken by Raspberry Pi's camera come with RGB coefficients in the range of 0-255 I had to normalize the values to span from 0 to 1., which was achieved by this scaling
- rotation_range=20: images were rotated randomly by 0-20 degrees
- width_shift_range=0.01: range in which image was randomly translated vertically
- height_shift_range=0.1: range in which image was randomly translated horizontally
- shear_range=0.05: range in which shearing transformations were applied randomly
- zoom_range=0.1: range in which image was zoomed at randomly
- fill_mode='nearest': this was the method with which newly introduced pixels were filled out

Data augmentation was carried out only for the training images. I have used a function from Keras ('ImageDataGenerator'), which takes a path to a directory, and generates batches of augmented data indefinitely in an infinite loop.

Implementation

In this section, the process for which metrics, algorithms, and techniques that you implemented for the given data will need to be clearly documented. It should be abundantly clear how the implementation was carried out, and discussion should be made regarding any complications that occurred during this process. Questions to ask yourself when writing this section:

- *Is it made clear how the algorithms and techniques were implemented with the given datasets or input data?*
- *Were there any complications with the original metrics or techniques that required changing prior to acquiring a solution?*
- *Was there any part of the coding process (e.g., writing complicated functions) that should be documented?*

For the implementation I have chosen Keras. Keras is a neural network library for Theano and TensorFlow written in Python. As I have already mentioned, a convolutional neural net architecture was applied for the task. The network consisted of an input layer, 3 convolutional layers, a fully connected layer, and an output layer. The convolutional layers used 3x3 convolutions and 32 output filters followed by pooling layers of 2x2. For the activation functions I have chosen rectified linear units, except for the final output neuron which was sigmoid. After the fully connected layer a dropout of 0.5 was applied (this helps to prevent overfitting). For the loss function I have used logloss (binary crossentropy). The initial optimizer was rmsprop.

With regards to the difficulties encountered in the process, the first difficulty was to understand how data augmentation should be carried out. It needed a bit of trial and error to figure out how much I can distort the images, such that the stop sign remains on the images all the time in the positive category.

With regards to building the model the main difficulty was to figure out how I can use alternative error metrics such as Recall. It proved to be difficult, because of the specific way I implemented the training phase. I did not have my training set compiled in a fixed array and my labels in a similar fixed array. Instead, as I mentioned previously, I have used a function from Keras ('ImageDataGenerator'), which takes a path to a directory, and generates batches of augmented data indefinitely in an infinite loop. I had to write additional code to have the data in a format in which I could use error metrics imported from sklearn.

A further difficult was to figure out how to visualize the filters. Although I had a solution to start from[5], the code needed some workaround, such as referencing my convolutional layers, writing a function to draw the images for all filters, and to understand how it works in general, as I haven't done anything like this previously [which made the process exciting as well].

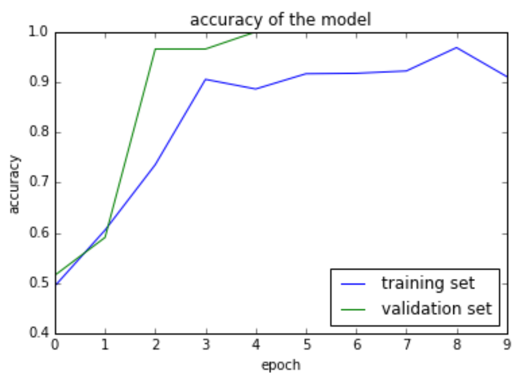
Refinement

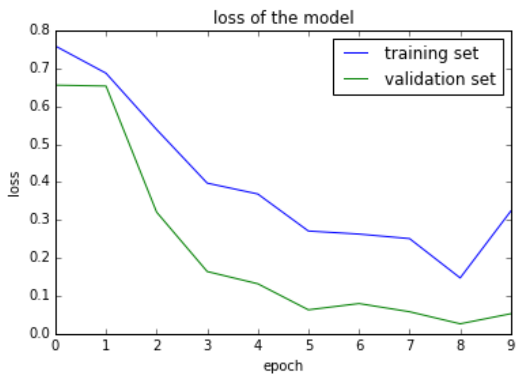
In this section, you will need to discuss the process of improvement you made upon the algorithms and techniques you used in your implementation. For example, adjusting parameters for certain models to acquire improved solutions would fall under the refinement category. Your initial and final solutions should be reported, as well as any significant intermediate results as necessary. Questions to ask yourself when writing this section:

- *Has an initial solution been found and clearly reported?*
- *Is the process of improvement clearly documented, such as what techniques were used?*
- *Are intermediate and final solutions clearly reported as the process is improved?*

As even the first implementation gave an accuracy of >98% on the validation set, I only tried minor variations, including a change in the optimizer (change rmsprop to adam). Below are the initial results followed by the results with the modified parameter:

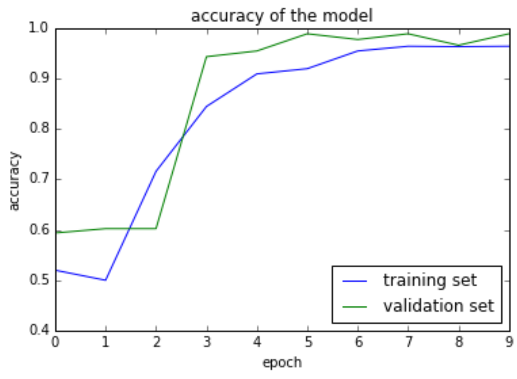
RMSPROP:

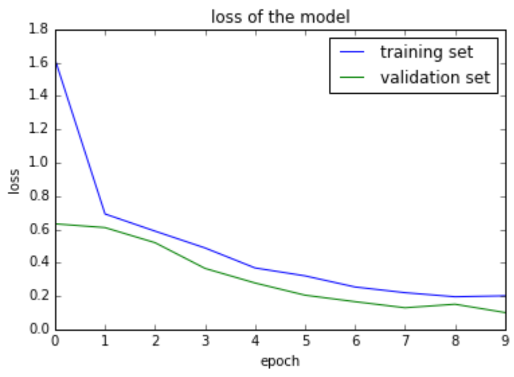




ADAM:

[5] https://blog.keras.io/how-convolutional-neural-networks-see-the-world.html





Results

(approximately 2 - 3 pages)

Model Evaluation and Validation

In this section, the final model and any supporting qualities should be evaluated in detail. It should be clear how the final model was derived and why this model was chosen. In addition, some type of analysis should be used to validate the robustness of this model and its solution, such as manipulating the input data or environment to see how the model's solution is affected (this is called *sensitivity analysis*). Questions to ask yourself when writing this section:

- *Is the final model reasonable and aligning with solution expectations? Are the final parameters of the model appropriate?*
- *Has the final model been tested with various inputs to evaluate whether the model generalizes well to unseen data?*
- *Is the model robust enough for the problem? Do small perturbations (changes) in training data or the input space greatly affect the results?*
- *Can results found from the model be trusted?*

I have used a validation set for model selection, and a separate test set for the final evaluation of the model. The final model reached close to 100% accuracy on the validation set, and a Recall score of 1 on the test set. Such a high level of accuracy is very important given the nature of the application (train control).

As I have used an optimizer with adaptive learning rates, and it has given great results with my initial model setup, I did not need to tweak other hyperparameters [e.g.: activation functions, number of layers, number of neurons per layer, etc.] further. If there were a need to find the most minimalistic model which still solves the problem I would have used GridSearch to find optimal parameters, but it was not necessary in the present case.

As I have applied both dropout and data augmentation, I believe the model can be trusted, at least as long as the train is moving along the same path where the training data was gathered. It would be good to test how robust the model is if we change the path, and introduce a very different environment.

Justification

In this section, your model's final solution and its results should be compared to the benchmark you established earlier in the project using some type of statistical analysis. You should also justify whether these results and the solution are significant enough to have solved the problem posed in the project. Questions to ask yourself when writing this section:

- *Are the final results found stronger than the benchmark result reported earlier?*
- *Have you thoroughly analyzed and discussed the final solution?*
- *Is the final solution significant enough to have solved the problem?*

The result obtained with the final model was higher than I initially expected. I thought that an accuracy of around 95% would be impressive enough for this particular task. The initial benchmark with the minimalistic architecture had an accuracy of about 50% percent, which means that it was basically just randomly guessing. Importantly for the final model, which was tested on a separate test set, I have used the Recall score for evaluation. As I mentioned earlier, for this particular problem the ideal error metric is Recall, as the most important thing is that we would like detect each stop signs, otherwise the train would 'cause accidents'. Given that my final model reached a Recall score of 1, which means that it has correctly identified all of the stop signs, in the specific context it seems the model have solved the problem.

Conclusion

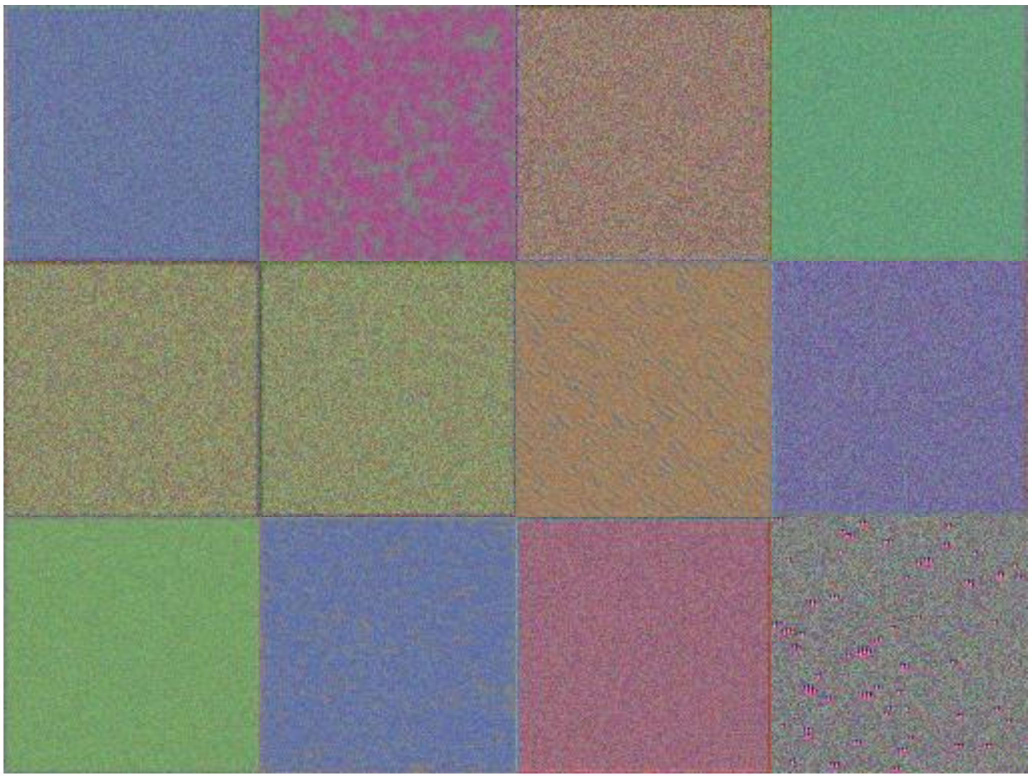
(approximately 1 - 2 pages)

Free-Form Visualization

In this section, you will need to provide some form of visualization that emphasizes an important quality about the project. It is much more free-form, but should reasonably support a significant result or characteristic about the problem that you want to discuss. Questions to ask yourself when writing this section:

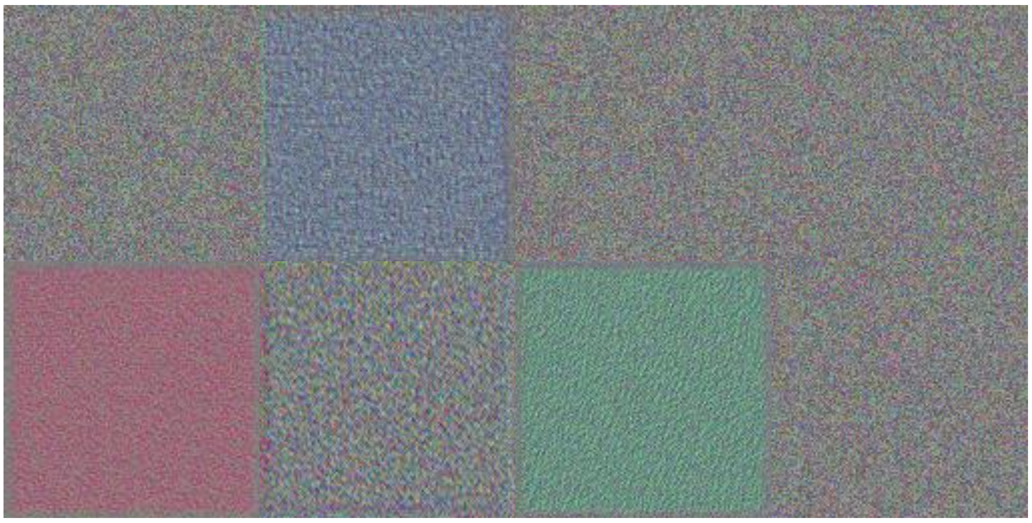
- *Have you visualized a relevant or important quality about the problem, dataset, input data, or results?*
- *Is the visualization thoroughly analyzed and discussed?*
- *If a plot is provided, are the axes, title, and datum clearly defined?*

In this section I have decided to visualize what the neural network learned, by visualizing what inputs maximize the filters in different layers. This gives us an idea of how the neural net decomposes the visual space.



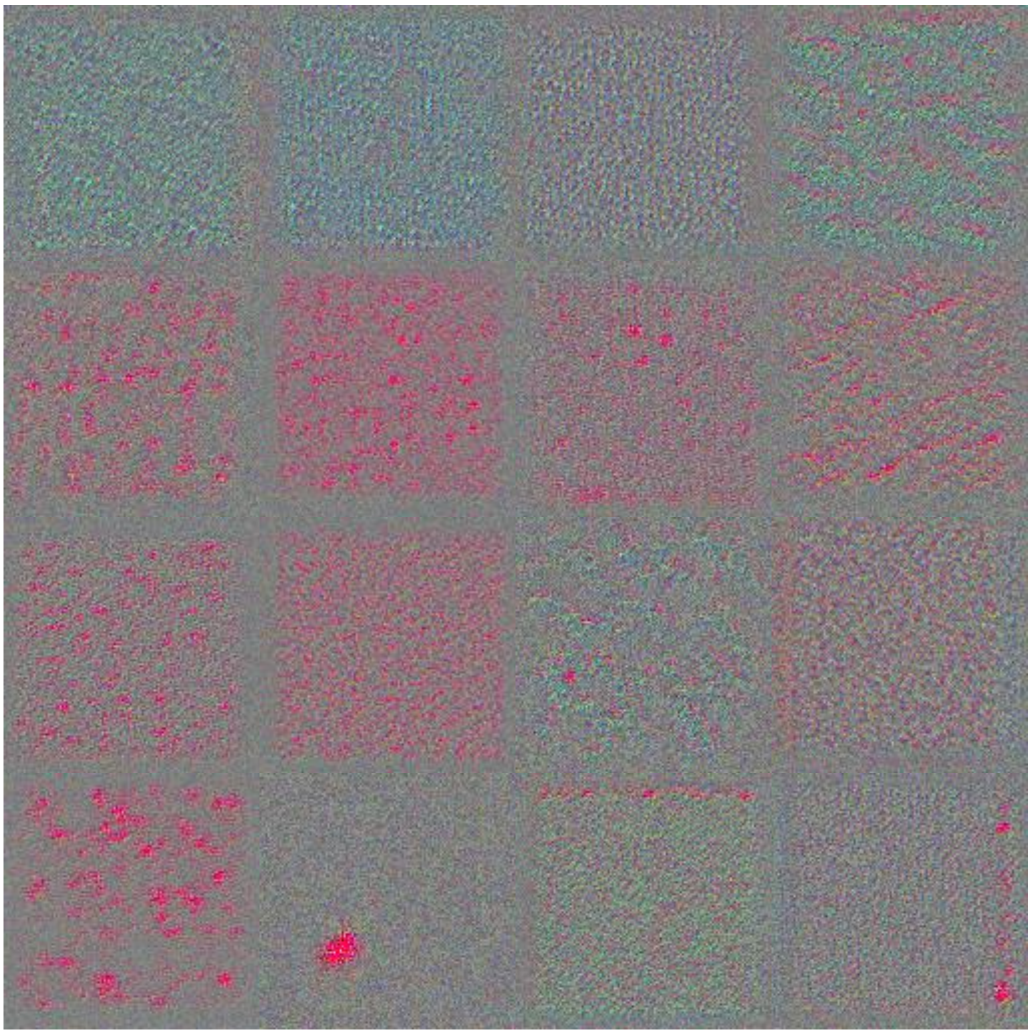
CONVOLUTIONAL LAYER 1 (a few selected filters)

As we can see from the output the first layer is mainly encoding color. We can see however that one of the filters already picks up small “redish” patches.



CONVOLUTIONAL LAYER 2 (a few selected filters)

In the second convolutional layer are rather noisy, but we have three filters for the three basic colors of red, blue and green.



CONVOLUTIONAL LAYER 3 (a few selected filters)

The third layer however contains something very interesting, and something that I hoped I will see. Many of the filters seem to be sensitive to red patches, and we have three particularly interesting filters, one which is sensitive to red in the right side of the image, one that is sensitive to red in the upper half of the image, and one that is sensitive to a red circle.

Reflection

In this section, you will summarize the entire end-to-end problem solution and discuss one or two particular aspects of the project you found interesting or difficult. You are expected to reflect on the project as a whole to show that you have a firm understanding of the entire process employed in your work. Questions to ask yourself when writing this section:

- *Have you thoroughly summarized the entire process you used for this project?*
- *Were there any interesting aspects of the project?*
- *Were there any difficult aspects of the project?*
- *Does the final model and solution fit your expectations for the problem, and should it be used in a general setting to solve these types of problems?*

In this project I have provided a solution for an aspect of automatic train control using image recognition. Image recognition used to be a difficult task historically, however for the last few years we have discovered efficient methods to approach these kinds of problems. Deep multi-layer neural networks are capable of building up a hierarchy of abstractions that makes it possible to identify complex inputs (i.e. images), and in this project I have utilized such an approach.

There were two major areas for the project. The first was data collection, the second was model building. Before I could actually build an accurate model, it was important to have a sufficient amount of data. Given that I was not working with an externally provided dataset a major difficulty for this project was to figure out an efficient method with which I could collect enough data. The reason why it is good to collect a large amount of data is that it helps to prevent overfitting and improves generalization. I think I have managed to collect a sufficient amount of data for this particular problem, and with data augmentation I had introduced quite a bit of variation, which can help in generalization.

After this, the actual model building was not that difficult. Although there are a large number hyperparameters I could experiment with (such as the number and type of layers, number of neurons per layers, the type of activation functions, regularization methods, loss functions, error metrics, optimizers), I could start from good architectures that were already discovered and published by others[6], and build from there. For this particular problem, although I had already extensive experience with using neural nets for demand forecasting, I had no prior experience with using neural nets for image recognition. It was amazing to see how efficient this method is, and how fast I can set up an architecture that is performing really well on the task.

Although the final method fits my expectation for the problem, I suspect that the specific model would need additional testing if it were to be used in an environment that is much different than the one in which I have developed it. The bottleneck here was the difficulty around data collection. However additional data could be gathered in case the model does not generalize well enough in largely different environments.

Improvement

In this section, you will need to provide discussion as to how one aspect of the implementation you designed could be improved. As an example, consider ways your implementation can be made more general, and what would need to be modified. You do not need to make this improvement, but the potential solutions resulting from these changes are considered and compared/contrasted to your current solution. Questions to ask yourself when writing this section:

- *Are there further improvements that could be made on the algorithms or techniques you used in this project?*
- *Were there algorithms or techniques you researched that you did not know how to implement, but would consider using if you knew how?*
- *If you used your final solution as the new benchmark, do you think an even better solution exists?*

With regards to improvements, I have already mentioned that gathering additional data would help in generalization. A further area in which I would like expand the project, is to expand it to a multiclass classification project, such that the model not only recognizes stop signs, but many other traffic signs as well, such as speed limits, etc., which could then be used to control the train in more ways. Considering that, it would potentially be necessary to expand the model architecture by adding more layers and neurons to it, such that the model is expressive enough to accommodate the additional complexity.

Another area of improvement that I have realized after visualizing what inputs maximize the activation of the filters, is that it will most likely not detect stop signs as such, but rather red blobs

[6] https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html

(probably of a particular size). If I want it to discriminate between stop signs, and similar traffic signs, which have red background, I would have to incorporate such traffic signs into the training, such that it learns how to make that finer discrimination.

Furthermore, an area that would probably require optimisation is prediction time, given that ideally the predictions would happen real time on streaming images. This is an area that would possibly need some tweaking if the model is to be put in practical use. However it becomes more of an issue if one does not have fast enough hardware to run the models on.