Stop Sign Detection:

A Comparison of Methods Using
Deep Learning & Logistic Regression

Statistical Data Science
github.com/audreychu/stop-sign-detection

Austin Chi: 999100655
Audrey Chu: 999138148
Jeremy Weidner: 999073290

June 2017, Department of Statistics
University of California, Davis

Abstract

In this project, we examine methods for detecting the presence of stop signs in photographs including

1. Transfer Model with Google's Inception V3
2. Mini-Inception Block Neural Network
3. Convolutional Neural Network
4. Logistic Regression using HOG

As stated in class, tackling the case of machine learning in autonomous cars in ten weeks would be infeasible, and so we opted for a related but simpler goal. We hoped to achieve a deeper understanding of machine learning for self-driving cars. At the end of our complete analysis, we found that Mini-Inception Block Neural Network achieved the best accuracy at about 89%, 3% higher the fully retrained Inception V3 model. Retrained Inception managed an accuracy of 86% but was not robust to red non-stop-sign objects. Logistic Regression achieved 73% accuracy by identifying roads instead of stop signs, while the Convolutional Feed Forward Neural Network was the worst performing at 71%.

Contents

1 Introduction

Image detection and machine/computer vision have many applications. For example, they can be used to substitute labor and more efficiently complete a job. In our case, we chose to look at stop sign recognition which serves as an introductory foundation for the field of autonomous cars. This report analyzes two methods for image classification of stop sign detection which are neural networks and logistic regression. Neural networks are a broad subject with many variations. In this report, we focus on three types. First, we use an existing model and retrain its features to fit our classification task. This represents the least computationally intensive method to train a network on this classification problem. The next method is a simple convolutional feed forward network, representing a naive approach. The final model is the Mini-Inception Net, which represents a complex model cut down to be trainable on a consumer-grade computer. We finish our analysis with logistic regression on histogram of orientated gradients in order to establish a baseline for non-neural net performance.

2 Data Collection

With python and the Google StreetView API, we gathered and cleaned data for stop sign locations in San Francisco. DataSF[1] provided all sign latitude and longitude coordinates. We then manipulated the camera orientation to obtain a diverse dataset containing various angles and lighting. After obtaining these images, we manually sorted the images based on the presence or absence of stop signs. Our final and cleaned dataset contains 789 images containing stop signs and 789 images containing no stop signs.

In terms of data limitations, we spent more time than expected on gathering and exploring our data. Initially, we hoped to combine a pre-established dataset from a Github repository as well as a dataset from the Laboratory for Intelligent and Safe Automobiles (LISA). Both datasets did not provide a diverse enough collection and thus we opted to create our own. Our final dataset is limited to a smaller than the at minimum several thousand images that would be ideal for our purposes.

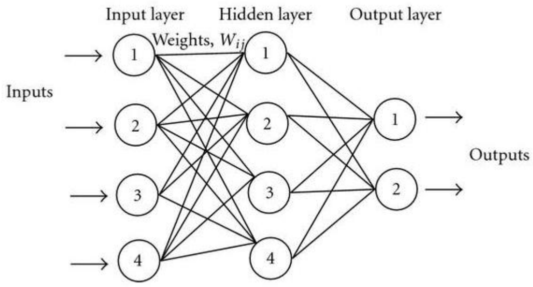
3 Neural Networks

There are many approaches for object recognition in computer vision. However, recent developments in neural networks using convolutional layers are making rapid gains compared to older methods of image recognition and so this was chosen as our first method to evaluate.

Inspired by the biological nervous system's way of processing information, a neural network is a processing paradigm composed of a large number of highly interconnected elements. A neural network learns by many iterations through a structure built to mimic the brain's learning process.

Neural Networks can be thought of as a collection of layers, made up of nodes, with weights connecting the nodes from different layers. The network structure at most basic form is made up of an input layer, a variable number of hidden layers, which are fully connected to a sequence of weights with biases, and an output layer. The input layer represents the information given to the network, which are stop sign images and their corresponding labels of yes/no in our case given to be used for training. The hidden layers act by being shown large numbers of labeled examples and adjusting the weights and biases through backpropogation to do the actual learning and the fully connected layers put together that information to be able to output a value. The output layer is typically a prediction probability and format depends on the type of input and nature of the network.

[1] https://data.sfgov.org/Transportation/Map-of-Stop-Signs/ekxq-2t5t/data



[2]

Figure 1: A Neural Network with a single hidden layer

Unlike other methods, neural networks have have the ability to handle feature extraction governed by just a few hyperparameters. A trained neural network is able to perceive patterns in the data that the scientist may not have noticed. While this makes the Neural Network hard to interpret, it can yield performance that outperforms humans in certain tasks. It has the remarkable ability to derive meaning and extract patterns to detect trends.

3.1 Feed Forward Networks

Neural networks where the output from one layer is fed as the input to the following layer are known as *feed forward neural networks*[3]. This is the most basic form of neural network and may not handle complex problems as well as other types of networks which can be thought of as variations of feed forward networks built to handle specific problems.[4]

3.1.1 Activation Functions

Activation functions are fundamental to neural networks, as they allow the neural network to fit non-linear models. The functions act as gates in the node regulating the type of output the node can produce. This helps the network mimic the firing of neuron in the brain.

Sigmoid

$$S(x) = \frac{1}{1+e^{-x}}$$

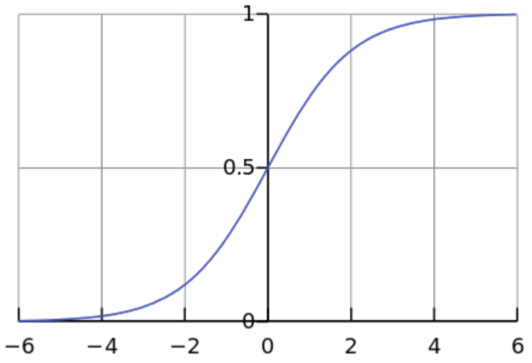


Figure 2: A Sigmoid Function

Sigmoid functions were one of the earliest functions used for neural networks. However, there are a few reasons why Sigmoid functions are no longer the preferred activation function for hidden layers. Sigmoid functions are flat at very low and very high values. Flat areas are much harder to "descend" from during gradient descent resulting in *saturated* neurons, slowing down the learning speed of the network. While training with an intermediate number of hidden layers can allow a network to escape saturation, this process is slow and the solution that the network arrives at is of poorer quality than other activation functions [GB10]. Sigmoid functions also require unsupervised pre-training of layers in order to achieve optimum performance [ECV10].

Rectified Linear Units

$$f(x) = max(0, x)$$

[3] Also known as Multilayer Perceptrons

[4] This structure also precludes the network from incorporating time series data. However, a more complex Neural Network, called a Recurrent Neural Network, can accomplish those tasks.

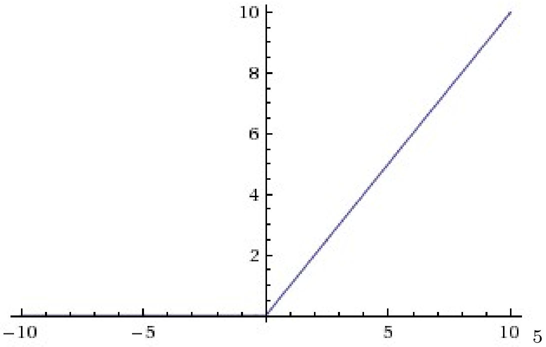


Figure 3: A Rectified Linear Unit

Rectified Linear Units (ReLU) have become very popular recently for a number of reasons.[6] First, they address the saturation issue at large values that affect Sigmoid functions. ReLU are also easily differentiable making it less computationally taxing to compute the gradient of networks that utilize them[NH10][KSH12]. Empirically, rectified linear units do a good job fitting non-linear functions without any unsupervised pre-training, making the task of creating a well fitting neural network much more straightforward [GBB11].

Softmax

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

The Softmax activation function is unique in that it is not traditionally used as an activation function for hidden layers. Rather, it is typically used in the output layer to normalize the outputs so that they sum to 1, giving a probabilistic interpretation to the output. For classification problems with mutually exclusive labels (both binary[7] and multinomial), this output is usually paired with a categorical cross entropy regression[8] layer for the final output [Gra12][BCV13][Sze+15b].

3.1.2 Backpropagation

Backpropogation is the process of adjusting the weights throughout the layers of a neural network based off the gradient of a loss function. A gradient descent optimizer is then used in order to find approximate minima.

Stochastic Gradient Descent Stochastic gradient descent (SGD) is a method of optimization that has become common in machine learning. Essentially, it is an iterative process in which a starting point is chosen; the gradient of the loss function is calculated; and is taken away from the gradient. The gradient is calculated by taking partial derivatives of the loss function.

$$\nabla F(x,y,z) = (\frac{\partial F}{\partial x}, \frac{\partial F}{\partial y}, \frac{\partial F}{\partial z})$$

This process is repeated until reaching a point that is an approximate minimum on the loss function. Stochastic gradient descent differs from traditional gradient descent in that only a subsection of the data[9] is used when calculating the gradient. This is necessary because the full dataset may be too large to process in the model at once.

Adaptive Moment Estimation Adaptive moment estimation more commonly known as ADAM is an optimization technique built on top of SGD calculation to converge faster and more accurately. It improves on the original algorithm by implementing the idea of momentum based on adaptive estimates of lower-order moments.[KB15] This momentum serves the purpose of helping the process of gradient descent converge faster and get past poorly performing

[6] While ReLU is very popular, it also has some problems, namely that nodes can ”die” if they receive a large negative update to its biases. This can be fixed by the LeakyReLU or Exponential Linear Unit(ELU)[Xu+15][CUH15]

[7] Binary categorical cross entropy is also known as logistic loss.

[8] Note that this is not a regression model, rather an algorithm that returns a score. An algorithm that returns a class label is known as a machine i.e. *support vector machine.*

[9] More commonly known as a batch.

local minima. The momentum factor is added to the gradient during the process of gradient descent.[10]

3.2 Convolutional Networks

Convolutional Neural Networks have been a breakthrough in the object recognition field. The fundamental feature of these types of networks are the convolutional layers.

3.2.1 Convolutional Layers

A convolutional layer is a hidden layer of a network made up of convolutional kernels, otherwise known as filters. Convolutional kernels are $n \times n$ windows that slide or *convolve* around the image. Each area of the window has a weight that is multiplied with the pixel value of that area. These weights are effectively the same as weights in a traditional feed forward network, in that error is back propagated to them as the network trains. This process results in two useful features.

1. Convolutional Neural Networks are area agnostic. Because traditional object recognition methods look at pixel values individually, the item must be in a specific position for the algorithm to detect it. This is usually solved with a sliding image, where a window is fed to the algorithm at a time, similar to how a convolutional filter works. However, convolutional networks are position agnostic while potentially being computationally less expensive than a more simple algorithm computing n times over a single image[11].

2. Convolutional Neural Networks deepen the feature set. One can think of a picture as a $n \times m \times 3$ array, where n and m are dimensions and 3 stands for the three color channels (RGB in this case). A convolutional layer can have an arbitrary window size, stride size, and filter number, expanding or decreasing the number of features based on these user defined hyper-parameters. In other words, the network does the feature engineering that would have traditionally been left to the user.

3.2.2 Max Pooling

Max pooling is a way to lower the computational cost of a convolutional neural network by effectively downsampling. An $n \times n$ window is used similarly to a convolutional layer, but instead of assigning different weights into each of the areas within the window, the max pooling filter selects the largest[12] value in said window and uses that as its value. While stride size is variable, like convolutional layers, max pooling layers usually use a stride size of n so as to not have overlap over the image; however empirical evidence has suggested that some amount of overlap of max pooling layers can lead to less overfitting [KSH12]. The amount downsampled by a max pooling layer is a function of the window size and the size of stride size. Max pooling has the added benefit of generalizing a convolutional kernel, making it scale and orientation invariant.

3.3 Transfer Learning

Deep neural networks are computationally expensive during training, especially when built from scratch. Modern deep networks exhibit a curious phenomenon where the first-layer features are general enough to apply to any training objective, including differing natural image

[10] Other optimizers also take into account momentum, but ADAM has a more effective method of calculating the momentum. More information can be found in Kingma's *Adam: A Method for Stochastic Optimization*[KB15]

[11] Note that this may be more computationally expensive than one would think. Usually an image must be iterated over multiple times with different sized windows in order to find objects that may be scaled differently

[12] Another popular pooling technique is average pooling. Similar to max pooling, instead of choosing the largest value, it averages all the values in the window and assumes that value.

datasets, supervised image classification, and unsupervised density learning [Yos+14]. Transfer learning takes advantage of this phenomenon by using a base network pretrained on a dataset, and then repurposing the learned features to a second target network to be trained on a target dataset and task.

Vanishing Gradients With very deep networks, a large concern is propagating the error back to the earliest layers[Sch15]. This is because, gradient descent algorithms distribute the error from the end of the net back toward the beginning, making the weights at the beginning of the net difficult to update. With transfer learning, we assume that many of these early factors are shared across classification tasks, getting rid of the need to propagate the error far back into the net [Ben12].

3.3.1 Google's Inception Net

Inception Net is a convolutional neural net framework currently in its 4th version. Each iteration is developed by Google for a large scale image recognition competition called ImageNet where teams compete to get the highest accuracy on a testing set with 1000 classes based on a training set of 1.2 million images [Sze+16]. Models are judged based on a metric called a top 5 error rate which is the rate at which a network guesses the correct label for a picture outside of the top 5 guesses for that picture. Google's Inception Net v3, a 48 layer Convolutional Neural Net built of Inception blocks had a top 5 error rate of 3.46% in 2015. It is notable that humans are expected to have a top 5 error rate of roughly 4% which means that this network is performing better than humans at the task of object identification.

Inception Blocks In order to reduce computational cost, as well as balance the depth and width of the network, Inception net uses Inception blocks [Sze+15b]. Unlike traditional feed forward layers, that can get deep very quickly, Inception blocks are wider. Each block splits the base input into 4 separate layers, a 5x5 convolution(approximated by two 3x3 convolutions in order to reduce computation cost), a 3x3 convolution, a max pooling layer, and a 1x1 convolution, before being concatenated.[Sze+15b] This is effectively trading depth for width, balancing the two in order to achieve the optimal network.[Sze+15b]

Auxiliary Outputs As touched on previously, the problem of vanishing gradients arises when networks are very deep. A way to combat this problem is with auxiliary outputs. Auxiliary outputs are small convolutional networks added on top of intermediate layers in the network. During training, their loss is added to the total loss with a discounted weight [Sze+15a]. The extra loss allows error to be more easily propagated to the earlier layers. In the case of Inception Model V3, the Auxiliary outputs are place on the 4a and 4d blocks.

4 Histogram of Oriented Gradients (HOG)

For our fourth model of evaluation, we used histogram of oriented gradients (HOG) feature descriptor with logistic regression. A feature descriptor is a simplified version of a image that only contains useful information.

HOG divides an image into many small spatial cells, all of which accumulate a weighted local 1-D histogram of gradient directors over the pixels. The feature descriptor converts an image of size $width \times height \times 3$ *color channels* to a feature vector or array of length n. It analyzes each block of an image, with overlap, to take edge information. All feature vectors are then concatenated into one giant vector that then works with logistic regression to determine a final classification.

4.1 Feature Vector Calculation

HOG descriptors are computed on a dense grid of uniformly spaced cells [DT05]. The feature vector $\mathbf{x}$ is calculated by dividing a subwindow by to $m \times n$ cells and calculating the histogram of oriented gradients for each cell. The feature vector consists of cell-level features

$$x = [x_1; x_2; ...; x_m n] \in \mathbb{R}^{mnd}$$

$$where\ \forall c \in \{1, ..., mn\}, x_c \in \mathbb{R}^d$$

and d is the dimension of the cell-level features [ASM14].

Finally, the HOG descriptors from all blocks of a dense overlapping grid of blocks are combined to be used in a window classifier. Below is an example of HOG, showing what features are most important for object detection. The edge detection shows how street location and shape are strong factors.

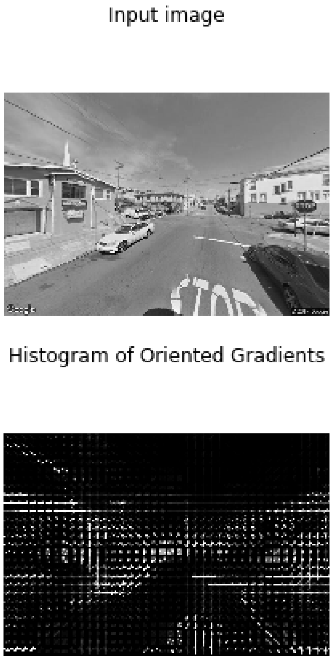


Figure 4: A picture before and after it has undergone HOG feature extraction

4.2 Logistic Regression

The logistic regression method uses the sigmoid[13] function by maximizing the log-likelihood probability. The calculated HOG parameters are then fed into the function to classify whether an image contains a stop sign based on its probability [SBY].

5 Empirical Findings

In comparing the four models across two methods, we considered their respective accuracy rates as well as feature maps. In machine learning, 2-dimensional feature maps help visualize the probability of object detection by iterating over regions of an image. A test image is systematically covered up with a gray square where a respected model is run on each image with a differing gray square position [13]. In our case, when a stop sign is obscured, the probability for *stop sign present* drops significantly to the color blue. Red represents areas of the photograph where the model does a predicts the high probability that there is no stop sign.

5.1 Transfer Learning with Google's Inception V3

The model we decided to use for our transfer learning implementation is the Inception V3 Network from Google. The Inception network was trained on 1000 classes from the Image Net Large Scale Visual Recognition Challenge from 2012. After removing the last layers, and retraining them on our classification tasks we were able to achieve 86.00% accuracy in only 4000 steps. This took roughly 30 minutes on the CPU, much less computation than the simple convolutional model or the Mini-Inception Net.

[13] This is the same as binary softmax

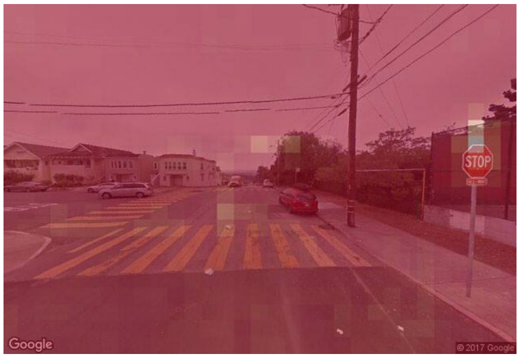


Figure 5: A Feature Map Using the Retrained Inception Neural Network

The feature map shows that the network has managed to identify the stop sign from the rest of the image. However, we had suspicions that the network was looking for red rather than for the actual stop signs. We tested this by passing 2 unrelated pictures to the network that contained red, a red shoe and a red fish[14]. The model identified both of those images as stop signs, failing the robustness checks.

5.2 Convolutional Feed Forward Network

The Convolutional Feed Forward Network was trained 100 epochs, considerably less than the Mini-Inception Model. As expected from such minimal training, its accuracy suffered. It is by far our worst performing model at 70.89% accuracy. Shortcomings with regards to the number of iterations can be attributed to a series of unforeseen technical complications which led to running out of time to train the model. We had originally anticipated this model to do better than logistic regression based off of structure alone but it fell short. We believe that this is due in part to the previously mentioned lower number of training epochs but also because we did not have the time to identify and sort by hand the thousands of images that networks in the feed forward family would require to train from scratch.

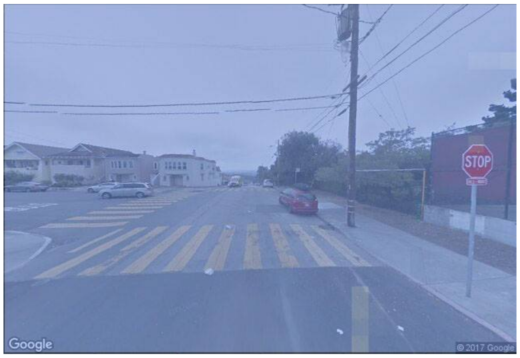


Figure 6: A Feature Map Using the Convolutional Feed Forward Neural Network

5.3 HOG with Logistic Regression

Our logistic regression model was trained on a dataset containing our HOG feature values. The model split the dataset into 80% training data, 10% validation, and 10% testing data. After predicting values from our model, we used cross-validation to evaluate the estimator performance. In order to prevent overfitting, the test set is held out as a part of the available dataset. Our final results showed an accuracy rate of 74%.

This 73.93% is an indication of poor performance in the model. This might be because the model is predicting stop signs based on location or type of street angle. Specifically, we found a pattern in which images where the street view is entirely facing a row of homes, the model is simply guessing at a 50% rate. For the case of the image below that contains a stop sign at a differing angle, the logistic regression model predicted sign absent with 49.23%.

[14] These pictures are included in the appendix

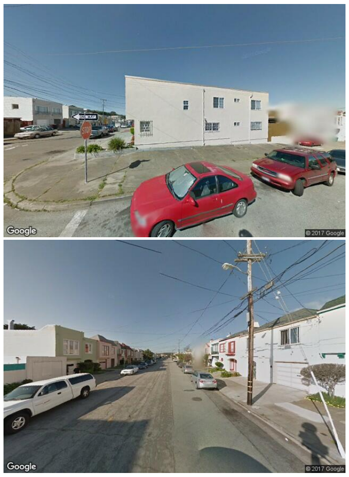


Figure 7: Comparison of images with the same angle, similar prediction rate, but differing presence of a stop sign.

In contrast, for images that contain a straight street view as shown in the figure straight above, the model is simply guessing the presence of a stop sign. The above image does not contain a stop sign; however, the model predicts the present of a stop sign with a probability of 79.60%. HOG using logistic regression is thus not the better method of detection in that it is largely dependent on surrounding factors.

As shown below, the feature map using HOG logistic regression does a unsatisfactory job at predicting the probability of a stop sign. Specifically, the area on top of the stop sign is the same hue as regions that do not contain a stop sign.

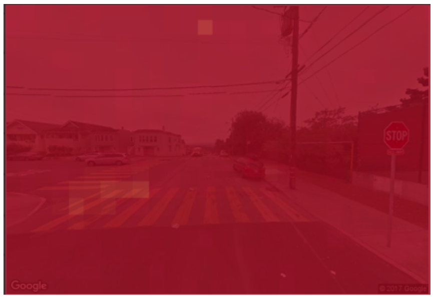


Figure 8: A feature map using the HOG Logistic Regression method

5.4 Mini-Inception Net

We found great success with a cut down version of the Inception V3 model. After training the model for 900 epochs[15], the model achieved an accuracy of 89.24%.[16]

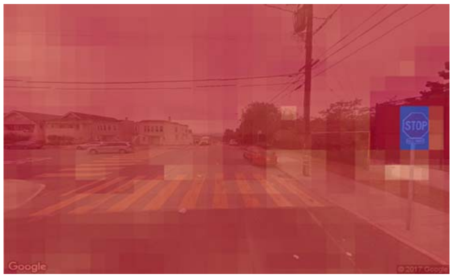


Figure 9: A Feature Map Using The Mini-Inception Net

The feature map shows that the model is indeed finding the stop sign. In addition to cutting down the network to a single inception block, the number of nodes and kernels were quartered in order to fit onto our hardware. Additionally, localized area normalization was omitted due to the inclusion of batch normalization.[SS15]

[15] 9 hours on a GTX 780 GPU using batch sizes of 50.

[16] Our initial suspicions were that this model was also just detecting red objects, however feeding the network an image of a red fish and a red shoe yielded the correct low probability we expected. Those pictures are in the appendix.

6 Conclusions

Table 1: Accuracy of the different models.

Model	Accuracy
Mini-Inception	89%
Transfer	86%
HOG Logistic	74%
Convolutional Feed Forward[17]	71%

In this project, we analyzed four models across two methods for detecting stop signs. We had initially anticipated the higher performance of neural networks when compared to logistic regression. To our surprise, this was not entirely the case. While our best neural network did outperform logistic regression by a respectable amount, our poorly constructed and minimally trained convolutional feed forward network managed to under-perform logistic regression with HOG.

Next Steps Further progress in this field can be made on multiple fronts. Moving from photo data to video data presents a unique challenge in trying to incorporate time series data in a convolutional network. Object locating is another area that can be improved upon, finding exactly where the stop sign is in the image.[18] Speed of training as well as computational speed are also subjects of interest.

[17] The Convolutional Net was only trained on 100 epochs

[18] Existing models on this topic include faster R-CNN models[Ren+15]

References

[13] rob fergus matthew zeiler rob fergus. “visualizing and understanding convolutional networks”. In: *arxiv* (Nov. 2013). ISSN: 978-3-319-10589-5. DOI: 10.1007/978-3-319-10590-1{_}53. URL: http://arxiv.org/abs/1311.2901%20https://arxiv.org/pdf/1311.2901v3.pdf.

[ASM14] Ejaz Ahmed, Gregory Shakhnarovich, and Subhransu Maji. “Knowing a good HOG filter when you see it: Efficient selection of filters for detection”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics).* Vol. 8689 LNCS. PART 1. 2014, pp. 80–94. ISBN: 9783319105895. DOI: 10.1007/978-3-319-10590-1{_}6. URL: http://ttic.uchicago.edu/~gregory/papers/goodParts-cameraReady.pdf.

[BCV13] Yoshua Bengio, Aaron Courville, and Pascal Vincent. “Representation Learning: A Review and New Perspectives”. In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 35.8 (June 2013), pp. 1798–1828. ISSN: 1939-3539. DOI: 10.1109/TPAMI.2013.50. URL: http://arxiv.org/abs/1206.5538.

[Ben12] Yoshua Bengio. “Practical Recommendations for Gradient-Based Training of Deep Architectures”. In: (June 2012). ISSN: 18727913. DOI: 10.1016/j.ijantimicag.2014.12.030. URL: http://arxiv.org/abs/1206.5533.

[CUH15] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. “Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)”. In: (Nov. 2015). URL: http://arxiv.org/abs/1511.07289.

[DT05] Navneet Dalal and Bill Triggs. “Histograms of oriented gradients for human detection”. In: *Proceedings - 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 2005.* Vol. I. 2005, pp. 886–893. ISBN: 0769523722. DOI: 10.1109/CVPR.2005.177. URL: http://vc.cs.nthu.edu.tw/home/paper/codfiles/hkchiu/201205170946/Histograms%20of%20Oriented%20Gradients%20for%20Human%20Detection.pdf.

[ECV10] Dumitru Erhan, Aaron Courville, and Pascal Vincent. “Why Does Unsupervised Pre-training Help Deep Learning ?” In: *Journal of Machine Learning Research* 11 (2010), pp. 625–660. ISSN: 15324435. DOI: 10.1145/1756006.1756025. URL: http://www.jmlr.org/papers/volume11/erhan10a/erhan10a.pdf%20http://portal.acm.org/citation.cfm?id=1756025.

[GB10] Xavier Glorot and Yoshua Bengio. “Understanding the difficulty of training deep feedforward neural networks”. In: *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS)* 9 (2010), pp. 249–256. ISSN: 15324435. DOI: 10.1.1.207.2059. URL: http://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf%20http://machinelearning.wustl.edu/mlpapers/paper_files/AISTATS2010_GlorotB10.pdf.

[GBB11] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. “Deep sparse rectifier neural networks”. In: *AISTATS ’11: Proceedings of the 14th International Conference on Artificial Intelligence and Statistics* 15 (2011), pp. 315–323. ISSN: 15324435. DOI: 10.1.1.208.6449. URL: http://proceedings.mlr.press/v15/glorot11a/glorot11a.pdf.

[Gra12] Alex Graves. *Supervised Sequence Labelling with Recurrent Neural Networks.* Vol. 385. 2012. ISBN: 978-3-642-24796-5. DOI: 10.1007/978-3-642-24797-2. URL: http://www.cs.toronto.edu/~graves/phd.pdf%20http://link.springer.com/10.1007/978-3-642-24797-2.

[KB15] Diederik P. Kingma and Jimmy Lei Ba. "Adam:" in: (Dec. 2015), pp. 1–15. ISSN: 09252312. DOI: http://doi.acm.org.ezproxy.lib.ucf.edu/10.1145/1830483.1830503. URL: http://arxiv.org/abs/1412.6980.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances In Neural Information Processing Systems* (2012), pp. 1–9. ISSN: 10495258. DOI: http://dx.doi.org/10.1016/j.protcy.2014.09.007. URL: https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.

[NH10] Vinod Nair and Geoffrey E Hinton. "Rectified Linear Units Improve Restricted Boltzmann Machines". In: *Proceedings of the 27th International Conference on Machine Learning* 3 (2010), pp. 807–814. ISSN: 1935-8237. DOI: 10.1.1.165.6419. URL: http://machinelearning.wustl.edu/mlpapers/paper_files/icml2010_NairH10.pdf.

[Ren+15] Shaoqing Ren et al. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". In: (2015). ISSN: 0162-8828. DOI: 10.1109/TPAMI.2016.2577031. URL: http://arxiv.org/abs/1506.01497.

[SBY] Hyun Oh Song, Sara Bolouki, and Andrew Yen. "Detection of Stop Signs". In: (). URL: http://ai.stanford.edu/~kosecka/FinalReport_5_T2.pdf.

[Sch15] Jurgen Schmidhuber. *Deep Learning in neural networks: An overview.* Apr. 2015. DOI: 10.1016/j.neunet.2014.09.003. URL: http://arxiv.org/abs/1404.7828%20http://dx.doi.org/10.1016/j.neunet.2014.09.003.

[SS15] Christian Szegedy and Ioffe Sergey. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: (2015). URL: https://arxiv.org/pdf/1502.03167.pdf.

[Sze+15a] Christian Szegedy et al. "Going deeper with convolutions". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition.* Vol. 07-12-June. Sept. 2015, pp. 1–9. ISBN: 9781467369640. DOI: 10.1109/CVPR.2015.7298594. URL: http://arxiv.org/abs/1409.4842.

[Sze+15b] Christian Szegedy et al. "Rethinking the Inception Architecture for Computer Vision". In: (Dec. 2015). ISSN: 08866236. DOI: 10.1109/CVPR.2016.308. URL: http://arxiv.org/abs/1512.00567.

[Sze+16] Christian Szegedy et al. "Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning". In: (2016). ISSN: 01678655. DOI: 10.1016/j.patrec.2014.01.008. URL: https://arxiv.org/pdf/1602.07261.pdf%20http://arxiv.org/abs/1602.07261.

[Xu+15] Bing Xu et al. "Empirical Evaluation of Rectified Activations in Convolution Network". In: *ICML Deep Learning Workshop* (May 2015), pp. 1–5. URL: http://arxiv.org/abs/1505.00853.

[Yos+14] Jason Yosinski et al. “How transferable are features in deep neural networks?” In: (2014). ISSN: 10495258. URL: https://arxiv.org/pdf/1411.1792v1.pdf%20http://arxiv.org/abs/1411.1792.

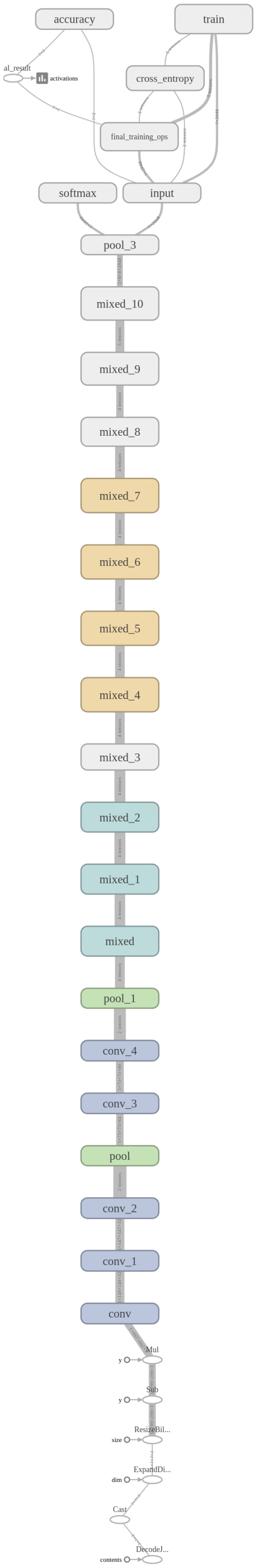
Appendices

A Reflection

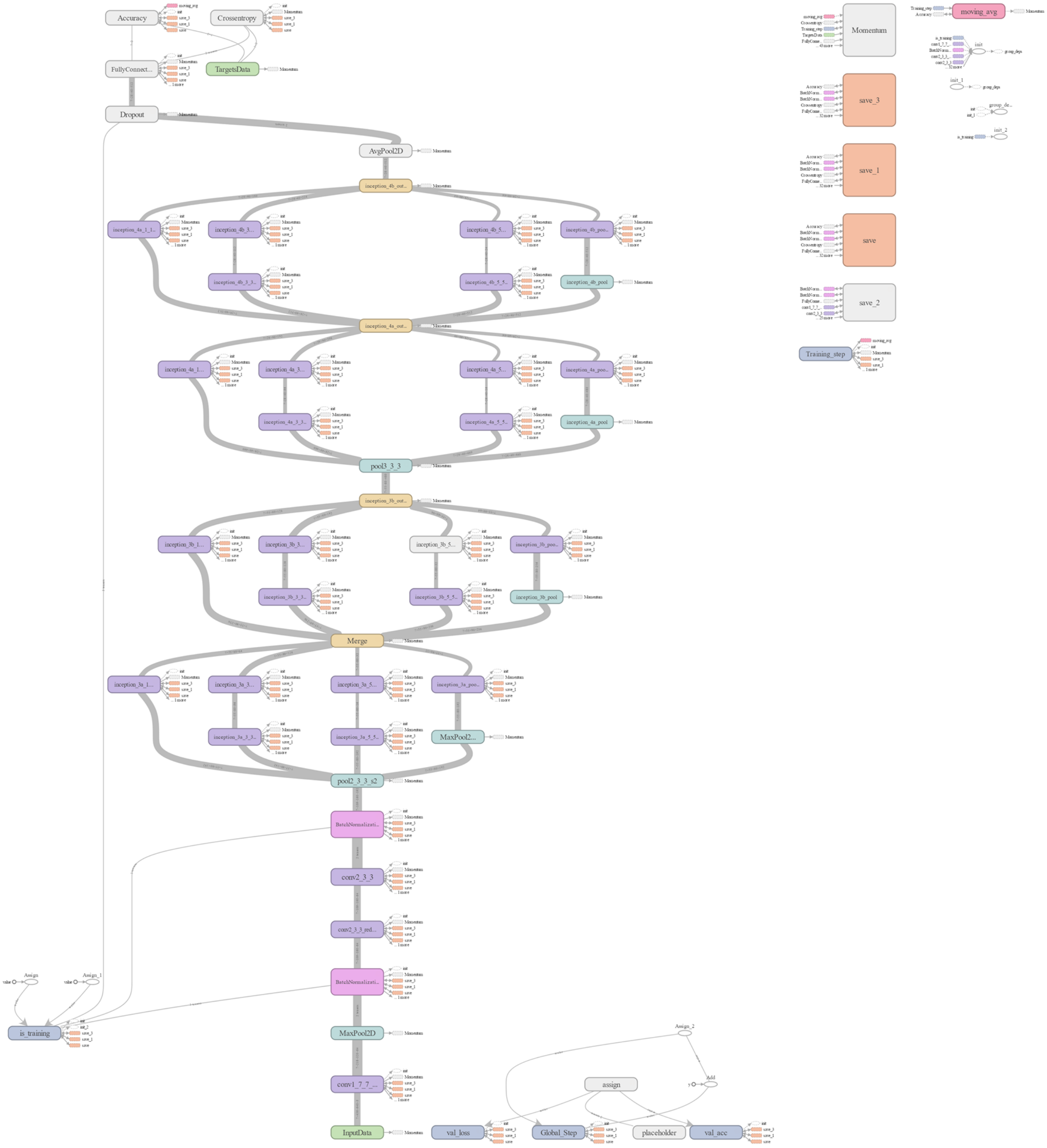
Our project timeline and goals have dramatically changed since the beginning of the spring quarter. Our initial timeline had underestimated the amount of time needed to not only gather, but also clean a near-perfect dataset. In terms of data extraction, we were substantially far from a correct time estimation. Our biggest challenge for this project was learning the theory and process of each method. Aside from logistic regression, we had to spend a great deal of energy to understand the inner-workings of neural networks and why they are better than other methods. In the case of logistic regression, we had to learn how histogram of oriented gradients work. We also had challenges regarding computational power, meaning a large portion of our time was spent on finding a common ground so that all of our of processes and methods would be in-sync. This included working with the correct versions of python for respective tasks, learning docker, TensorFlow, TFlearn, sklearn, PIL, and OpenCV.

Our project goals also evolved from the initial idea of using neural networks to detect stop signs to a comparison across several methods. This created a far more challenging goal that we think we adequately accomplished. We worked together to help each other answer questions in what we already knew or had recently learned. Online resources as well as answers from faculty, including Professor Cho-Ju Hsieh and graduate student Huan Zhang, were of great help. Tasks were shared across all team members. We split methods among members; however, still worked together to solve issues we could not by ourselves. We learned to collaborate on projects virtually as well as in person. Patience and the ability compromise were successful skills that were not only strengthened but also contributing factors to completing a cohesive report. Aside from the knowledge we gained from the technical portion of our analysis, our main takeaway is not to underestimate potential pitfalls and their ramifications with respect to a project’s timeline.

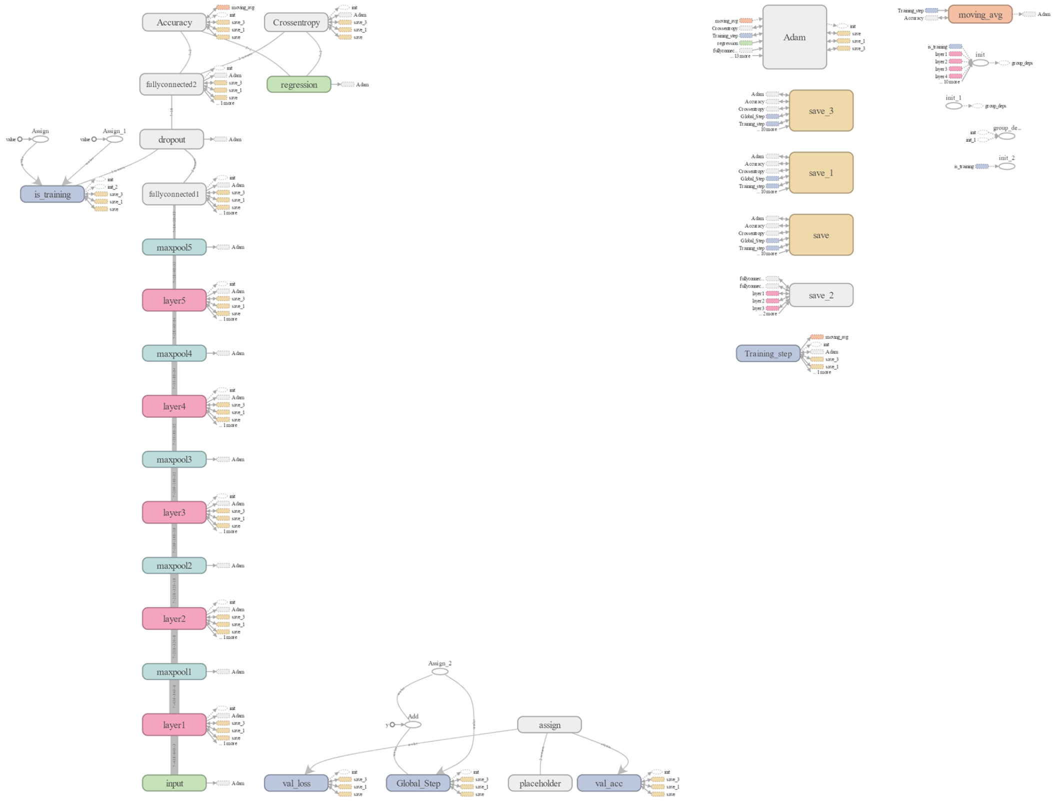
B Additional Diagnostics



Full Inception Graph



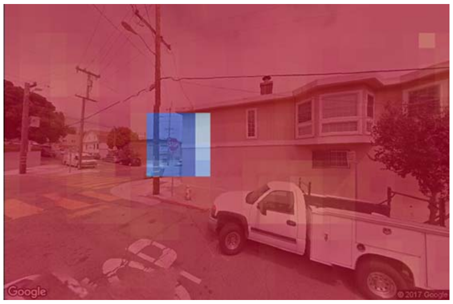
Mini Inception Graph



Convolutional Graph

Our Red Fish Robustness Test

Our Red Shoe Robustness Test



Another Feature Map Produced from the Mini-Inception Model

C Essential Code

Excerpts of code. All code can we found on our github repository.

C.1 Retraining Google's Inception Model

```
#THIS IS TO LAUNCH DOCKER
# run with sudo
docker run -it \
  --publish 6006:6006 \
  --volume ${HOME}/tf_files:/tf_files \
  --workdir /tf_files \
  tensorflow/tensorflow:1.1.0 bash

#POINT THE ABOVE AT WHATEVER DIRECTORY YOU WANT TO USE

#THIS IS TO LAUNCH THE MODEL
python retrain.py \
  --bottleneck_dir=bottlenecks \
  --how_many_training_steps=40000 \
  --model_dir=inception \
  --summaries_dir=training_summaries/try4\
  --output_graph=retrained_graph.pb \
  --output_labels=retrained_labels.txt \
  --image_dir=finalstops \
  --print_misclassified_test_images

```

Listing 1: `runmodel.txt`

C.2 The Mini-Inception Network

```
import pandas as pd
import numpy as np
import random
import sklearn
from sklearn.model_selection import train_test_split
import os
from PIL import Image
import glob

import tflearn
from tflearn.layers.normalization import batch_normalization
from tflearn.layers.conv import conv_2d, conv_2d_transpose, max_pool_2d, avg_pool_2d, upsample_2d, conv_1d, max_pool_1d, avg_pool_1d, residual_block, residual_bottleneck, conv_3d, max_pool_3d, avg_pool_3d, highway_conv_1d, highway_conv_2d, global_avg_pool, global_max_pool
from tflearn.layers.core import input_data, dropout, fully_connected
from tflearn.layers.estimator import regression
from tflearn.layers.merge_ops import merge

label = str(random.randint(0,100000000))
MODEL_NAME ="INCPT"
PWD = "C:\\Users\\austi\\Dropbox\\STA 160\\stop-sign-detection\\Mini_Inception\\Mini_Inception"
PATH = "C:\\Users\\austi\\Desktop\\"
```

```
    #os.chdir(”C://Users/Jeremy/Desktop/School/160”)
    os.chdir(”C:\\Users\\Austin Chi\\Google Drive\\Other\\misc\\testbatch26k”)

    yespathlist = []
    for p in glob.glob('Yes/*.jpg'):
        yespathlist.append(p)

    nopathlist = []
    for p in glob.glob('No/*.jpg'):
        nopathlist.append(p)
    #nopathlist=random.shuffle(nopathlist)
    nopathlist = nopathlist[0:len(yespathlist)]

    allpath = yespathlist + nopathlist
    print(len(yespathlist))
    print(len(nopathlist))
    print(len(allpath))
    labellist = []
    image_list = []
    for path in allpath:
        for filename in glob.glob(path):
                if len(image_list) < len(yespathlist):
                    label = 1
                else:
                    label = 0
                im=Image.open(filename)
                c = im.copy()
                image_list.append(np.array(c))
                labellist.append(label)
                im.close()
    labellist = np.array(labellist)
    labellist = labellist.reshape(-1,1)
    enc = sklearn.preprocessing.OneHotEncoder(sparse = False)
    enc.fit(labellist)
    labellist = enc.transform(labellist)

    #789 are yes

    ###END DATA PROCESSING###
    #Split train test validate
    xtrain, x2, ytrain, y2 = train_test_split(image_list, labellist, test_size = 0.2,random_state = 1)
    xvalid, xtest, yvalid, ytest = train_test_split(x2, y2, test_size = 0.5, random_state=1)

    #start of building net
    network = input_data(shape=[None, 436, 640, 3])
    conv1_7_7 = conv_2d(network, 16, 7, strides=2, activation='relu', name = 'conv1_7_7_s2')
    pool1_3_3 = max_pool_2d(conv1_7_7, 3,strides=2)
    pool1_3_3 = batch_normalization(pool1_3_3)
    conv2_3_3_reduce = conv_2d(pool1_3_3, 16,1, activation='relu',name = 'conv2_3_3_reduce')
    conv2_3_3 = conv_2d(conv2_3_3_reduce, 48,3, activation='relu', name='conv2_3_3')
    conv2_3_3 = batch_normalization(conv2_3_3)
    pool2_3_3 = max_pool_2d(conv2_3_3, kernel_size=3, strides=2, name='pool2_3_3_s2')
    inception_3a_1_1 = conv_2d(pool2_3_3, 16, 1, activation='relu', name='
    inception_3a_1_1')
    inception_3a_3_3_reduce = conv_2d(pool2_3_3, 32,1, activation='relu', name='inception_3a_3_3_reduce')
    inception_3a_3_3 = conv_2d(inception_3a_3_3_reduce, 48,filter_size=3, activation='relu', name = 'inception_3a_3_3')
    inception_3a_5_5_reduce = conv_2d(pool2_3_3,4, filter_size=1,activation='relu', name ='inception_3a_5_5_reduce' )
    inception_3a_5_5 = conv_2d(inception_3a_5_5_reduce, 8, filter_size=5, activation='relu', name= 'inception_3a_5_5')
    inception_3a_pool = max_pool_2d(pool2_3_3, kernel_size=3, strides=1, )
    inception_3a_pool_1_1 = conv_2d(inception_3a_pool, 8, filter_size=1, activation='relu', name='inception_3a_pool_1_1')

    # merge the inception_3a__
    inception_3a_output = merge([inception_3a_1_1, inception_3a_3_3, inception_3a_5_5, inception_3a_pool_1_1], mode='concat', axis=3)

    pool5_7_7 = avg_pool_2d(inception_3a_output, kernel_size=7, strides=1)
    pool5_7_7 = dropout(pool5_7_7, 0.4)
    loss = fully_connected(pool5_7_7, 2,activation='softmax')
    network = regression(loss, optimizer='momentum',
                         loss='categorical_crossentropy',
                         learning_rate=0.001)

    model = tflearn.DNN(network, tensorboard_verbose = 0,best_checkpoint_path = PWD, tensorboard_dir = "C:\\tmp\\tflearn_logs")

    if os.path.exists('Mini_Inception8861.meta'.format(MODEL_NAME)):
        model.load(MODEL_NAME)
        print("Model Loaded")
    else:
        model.fit(xtrain,ytrain, n_epoch = 900, snapshot_epoch = True, validation_set=(xvalid,yvalid),snapshot_step = 500, show_metric = True, run_id = MODEL_NAME, batch_size=1)
        model.save(PATH + MODEL_NAME)

```

Listing 2: `Incpt.py`

C.3 Convolutional Neutral Network

```
import tflearn
from tflearn.layers.conv import conv_2d, max_pool_2d
from tflearn.layers.core import input_data, dropout, fully_connected
from tflearn.layers.estimator import regression

convnet = input_data(shape = [None, 640, 436, 3], name = 'input')

convnet = conv_2D(incoming = convnet, nb_filter = 32, filter_size =  2, activation = 'relu', name = 'layer1')
convnet = max_pool_2D(convnet, 2, name = 'maxpool1')

convnet = conv_2D(incoming = convnet, nb_filter = 64, filter_size =  2, activation = 'relu', name = 'layer2')
convnet = max_pool_2D(convnet, 2, name = 'maxpool2')

convnet = conv_2D(incoming = convnet, nb_filter = 128, filter_size =  2, activation = 'relu', name = 'layer3')
convnet = max_pool_2D(convnet, 2, name = 'maxpool3')

convnet = conv_2D(incoming = convnet, nb_filter = 256, filter_size =  2, activation = 'relu', name = 'layer4')
convnet = max_pool_2D(convnet, 2, name = 'maxpool4')

convnet = conv_2D(incoming = convnet, nb_filter = 512, filter_size =  2, activation = 'relu', name = 'layer5')
convnet = max_pool_2D(convnet, 2, name = 'maxpool5')

convnet = fully_connected(convnet, n_units = 1024,activation = 'relu', name = 'fullyconnected1')
convnet = dropout(convnet, keep_prob = 0.8, name = 'dropout')

convnet = fully_connected(convnet, n_units = 2,activation = 'softmax', name = 'fullyconnected2')
convnet = regression(convnet, optimizer = 'adam', loss='categorical_crossentropy', learning_rate=0.001, name = 'regression')

model = tflearn.DNN(convnet, tensorboard_dir = '/tmp/tflearn_logs/')
model.fit({'input': X}, {'targets': Y}, n_epoch = 10, snapshot_epoch = True, run_id = 'stopsign',
          snapshot_steps = '200', validation_set=({'input': test_x}, {'targets': test_y}))

model.save('tflearnsimpleCNN.model')

```

Listing 3: `simpleCNN.ipynb`

C.4 HOG Calculation

```
import cv2
import glob as glob
import matplotlib.pyplot as plt
from skimage.feature import hog
from skimage import data, color, exposure


# Compute gradient for Stop Present (Logical = 1), same process for Absent
df = pd.DataFrame()
for name in glob.glob('/Users/audreychu/Documents/4th Year/STA160/StopYes/*jpg'):
    jpg = cv2.imread(name)
    image = color.rgb2gray(jpg)
    fd, hog_image = hog(image, orientations=8, pixels_per_cell=(16, 16),
                        cells_per_block=(1, 1), visualise=True)
    start = '/Users/audreychu/Documents/4th Year/STA160/StopYes/'
    end = '.jpg'
    name = name[len(start):-len(end)]
    fd = np.append(fd, name)
    fd = pd.DataFrame(fd).T
    df = df.append(fd)

df['Stop'] = [1] * len(df)

```

Listing 4: Remaining code under `HOG_Feature.py`

C.5 Logistic Regression Model

```
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split,cross_val_score
import numpy as np

data = pd.read_csv(”/Users/audreychu/Documents/4th Year/STA160/hog_data2.csv”)
del data['index']
Y = data['Stop']
X = data.drop('Stop',axis=1)
# del X['level_0']


Xtrain,Xtest,Ytrain,Ytest = train_test_split(X,Y,test_size = .2, random_state=1)

model = LogisticRegression()
model.fit(Xtrain[Xtrain.columns[:-1]],Ytrain)
test = model.predict(Xtest[Xtest.columns[:-1]])
scores = cross_val_score(model,Xtrain[Xtrain.columns[:-1]],Ytrain,cv=5)


y = list(Ytest)
yhat = list(test)
count = 0
for i in xrange(len(test)):
    if y[i]==yhat[i]:
        count += 1.0

print float(count/len(test))


# Evaluate score by cross validation
scores = cross_val_score(model,Xtrain[Xtrain.columns[:-1]], Ytrain, cv=100)
print(np.average(scores))

```

Listing 5: `LogHog.py`

C.6 Feature Map

```

Xlist = []
for p in sorted(glob.glob('/Users/audreychu/Documents/4th Year/STA160/stop-sign-detection/Heatmap_LogReg/New_Images/*png'), key=os.path.getmtime):
    Xlist.append(p)

X = []
for path in Xlist:
    im=Image.open(path)
    print(path)
    c = im.copy()
#     a = np.array(c)
    im.close()
#     hog1 = cal_HOG(path)
    print(hog1.shape)
    probs = model.predict_proba(hog1.T)
    X.append(probs[0][1])
```

```python
    print(min(X))
    with open("heatmaparray.pickle",'wb') as mat:
        pickle.dump(X,mat)

```

Listing 6: General prediction script. Varies across models which can be found at `Pred.py`, `Pred_LogHog.py`, `simplepred.py` and `transfermapping.py`

```python
from PIL import Image
import numpy as np
from itertools import chain

def flatten(listOfLists):
    "Flatten one level of nesting"
    return list(chain.from_iterable(listOfLists))

X = range(0,601,20)
i = 0

lists = [[(num,y) for num in X] for y in range(0,401,20)]

lol = flatten(lists)
i = 0
for el in lol:
    img = Image.open('Picture.jpg', 'r')
    background = Image.new('RGBA', (80, 80), (150, 150, 150, 255))
    bg_w, bg_h = background.size
    img.paste(background, el)
    img.save('out' +str(i) +'.png')
    i+=1

print(len(range(0,401,20)))
print(len(range(0,601,20)))

```

Listing 7: `Image Process.py`

```python
import pickle
import numpy as np

with open("heatmaparray.pickle",'rb') as mat:
    lister = pickle.load(mat)

lister = np.array(lister)
lister = lister.reshape((21,31))
print(lister)

from matplotlib import pyplot
import matplotlib as mpl
# make a color map of fixed colors

#bounds=[-6,-2,2,6]
#norm = mpl.colors.BoundaryNorm(bounds, cmap.N)

# tell imshow about color map so that only set colors are used
img = pyplot.imshow(lister ,cmap = 'coolwarm',vmin=0,vmax=1)

# make a color bar
```

```
    pyplot.colorbar(img,cmap='coolwarm')

    pyplot.show()

```

Listing 8: `Heatmapper.py`