VARACNet: An Ensemble of Convolutional Neural Networks for Traffic Lights Recognition

Aditya Verma
A53219148
arv018@ucsd.edu

Akshaya Purohit
A53215013
akpurohi@ucsd.edu

Chetan Gandotra
A53210397
cgandotr@ucsd.edu

Rishabh Misra
A53205530
r1misra@ucsd.edu

Vamshi Gudavarthi
A53216604
vgudavar@ucsd.edu

Abstract

We describe VARACNet, an ensemble of multiple Convolutional Neural Networks (CNN) where each CNN is built with the motive of giving superior performance while keeping the model size small. The sub-models have no more than 490k parameters but each achieves an accuracy greater than 87%. Models are tested and trained on the Nexar traffic lights challenge dataset with the aim of correctly recognizing the presence and state of traffic lights in images taken by the drivers using the Nexar app. We show that minimizing the number of parameters in each of the models allows quick training even when computational resources are not abundant. We also perform localization experiments using Faster R-CNN on a separate annotated dataset (UCSD traffic lights dataset) to demonstrate high performance for classification and detection tasks. The output is highly desirable and useful for deployment onto actual vehicles where the memory is limited and traffic light inferences have to be made in real-time.

1 Introduction

We have all been there - a light turns green and the car in front of you doesn't budge. No one likes to get stuck behind a vehicle that doesn't notice when a light changes. This is where the Nexar Traffic Lights Recognition Challenge comes into the picture. This challenge demanded to develop a Convolutional Neural Network (CNN) model to recognize traffic-light state in the driving direction of the car. In any given image, the classifier needs to output whether there is a traffic light in the scene and whether it was red or green. More specifically, it should only identify traffic lights in the driving direction.

Nexar builds the world's largest open vehicle-to-vehicle (V2V) network by turning smartphones into connected AI dash-cams. Nexar's technology provides a new, safer driving experience with the potential of saving the lives of 1.3 million people who die on the road every year. This was the motivation behind Nexar's Challenge and this paper. This paper discusses the basic ideas to build a good model for the Nexar Traffic Lights Challenge and compare it to the top ranked solutions. The outcome will be highly beneficial in scenarios related to traffic lights and signs wherein the models will be able to predict traffic lights quickly and potentially avoid accidents.

An important aspect of this paper is keeping the model size small. The motivation for this is drawn from the fact that Nexar evaluates the network submitted to them using two aspects - (1) Classification Accuracy, and (2) Model Size. Hence, we discuss the approaches to maximize the challenge score by minimizing the model size score while attaining a high accuracy. At the naive implementation, our mini-models even beat SqueezeNet[9] in terms of both accuracy and model size for this task.

2 Background and Literature Review

With the great motivation to solve this problem, we investigated the previous work done in this area. Until recently, similar problems were solved by extracting features using standard computer vision techniques described in [5] and [6]. Some of these techniques include generating a candidate set of traffic lights using color and shape information. This candidate set is further pruned using shape and spatial information. Given the candidate set, researchers adopted machine learning algorithms to detect and classify the traffic lights. In [7], the detection was done by obtaining the candidate set of gray scale images by first identifying bright spot lights and then used an adaptive template matching scheme to detect the traffic lights. In these methods, we encode our prior belief in extracting features and hence, these methods lack the robustness. However, with Convolutional Neural Networks (CNNs) performing extremely well on classification tasks(such as ImageNet) in the recent years, the first usage of CNN's to solve the traffic lights problem is presented in [8]. The results are highly encouraging and this may be due to the fact that the convolutional layers extract features while the fully connected layers classify. This gives CNNs added advantage over other methods and allows them to achieve higher accuracy and robustness. Also, CNNs do so by looking at a lot of input-output labels and encoding their representations. Encouraging results using CNNs in similar kind of problem - Traffic Signs detection and classification - is presented in [11] by Yann LeCun.

As mentioned earlier, the Nexar traffic lights challenge also carries points for model size. Due to this, we are restricted in terms of model selection. The models which achieved high accuracy on ImageNet tasks like AlexNet, VGGNet and GoogLeNet can't be directly used due to the large model size. Hence, we started looking for an alternate simple model. This is when we encountered SqueezeNet[9], a model that has 50 times fewer parameters than AlexNet and yet it is able to achieve accuracy similar to AlexNet on ImageNet. Small models like SqueezeNet offer at least 3 advantages as described in [9] - (1) training heavy neural networks like AlexNet require cross-communication across many systems during training if implemented efficiently and these communications come with a cost, (2) these models require less bandwidth to export the model from cloud to autonomous vehicle and hence, this model is very practical in these tasks where the ultimate aim is to assist in the perception of autonomous vehicles in real-time, and (3) these smaller CNNs are more feasible to deploy on FPGAs (Field-Programmable Gate Array) and other hardware with limited memory.

After training a SqueezeNet network on the Nexar traffic lights dataset, we decided to improve the accuracy through an ensemble of various CNNs as described in [10]. Even though the idea of employing an ensemble of models for classification task has been prevalent in the Machine Learning community for a long time, our inspiration was primarily derived from the GoogLeNet architecture, specifically the inception module. In an inception module, any particular convolutional layer output is taken and passed in parallel through different filter sizes and max-pooling layers, and later all the outputs are combined through depth wise concatenation. This is similar to constructing hybrid feature maps at many places in the network. This is the reason we believed that an ensemble of many models will be better at the classification task. The detailed descriptions of all the ideas presented in this section will be elaborated in the coming sections.

2.1 Dataset

The Nexar training dataset consist of 18,659 labeled images, which we propose to split in appropriate proportions for training and validation sets. The data was augmented using various techniques which will be discussed in the section 2.2. The augmented data size exceeds 100,000 images.The Nexar dataset is labelled in the following format:

- 0 = No traffic light in the driving direction
- 1 = Red traffic light in the driving direction
- 2 = Green traffic light in the driving direction

In addition to the Nexar dataset, we used UC San Diego Traffic Lights dataset [18] for Faster R-CNN [15] approach. There are 5403 training images with classes 'green', 'red' and 'yellow', and 899 validation images in this dataset.

2.1.1 Sample images and evaluations

Some sample images are shown below whose original resolution is 455×256:

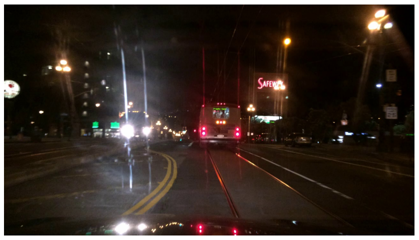
Figure 1: No Traffic Lights

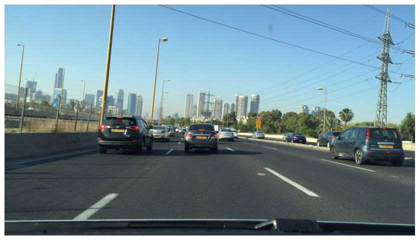
Figure 2: No Traffic Lights

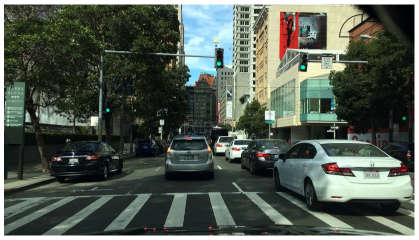
Figure 3: Green Traffic Light

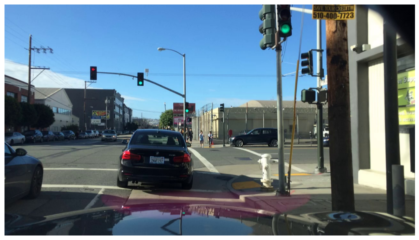
Figure 4: Green Traffic Light

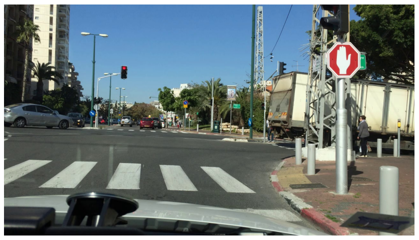
Figure 5: Red Traffic Light

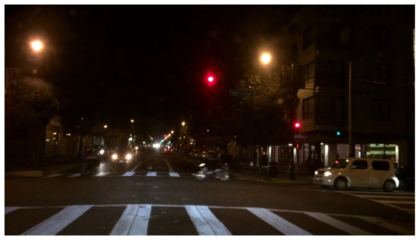
Figure 6: Red Traffic Light

By looking at the average pixel densities of images, we can easily distinguish them as day or night images. The images with low average pixel density values correspond to dark images - taken at night-time, and bright images had high average pixel density values - taken at daytime, as shown in figure 7.

As stated earlier, an important aspect of this paper is keeping the model size small. This is reiterated from the fact that Nexar evaluates the network submitted to them using following two aspects. First is the classification accuracy (CA) which is calculated as follows:

$$CA = \frac{m}{n}$$

where m is the number of predictions that our model makes correctly, while n is the total number of predictions made. The second aspect is the normalized model size (NMS) score. If w is the model size in megabytes, NMS can be calculated as:

$$NMS = e^{-\frac{w}{100}}$$

Combining both, we have:

$$Challenge\ Score = CA \times NMS$$

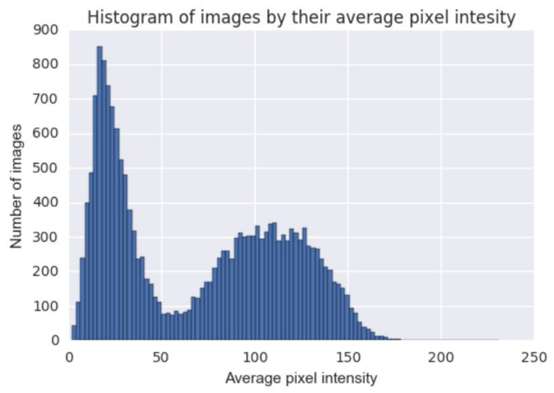

Figure 7: Average Pixel Densities[4]

2.1.2 Object Detection

The R-CNN method [12] proposed by Ross Girshick *et al.* was the state of the art method for Object Detection in 2014. It computed region proposals of the input image using Selective Search [13] and then passed each region proposal through AlexNet and generated class scores. Selective search is a fast object segmenting method for images which captures objects along different scales and over diverse range (texture, light,etc.). R-CNN makes use of these object regions and classifies them. This method gave good results but took a lot of time during both training and testing. Then came along the Fast R-CNN [14] model which instead of passing each region proposal through network passes the full image. After the convolutional layers, a region of interest (RoI) pooling was performed by projecting the region proposals onto the feature map. This reduced the training time by a big amount, but it still computed all the region proposals using Selective Search. This was later updated to the Faster R-CNN [15], which computed the region proposals using a Region Proposal Network (RPN), described in section 3.5. This reduced both training and testing times and gave real time object detection. Another state of the art method for real-time object detection, YOLO [17], was also explored. YOLO tackles the detection task as a regression problem. The image is divided into an even grid, and then the bounding boxes, confidence of those boxes and their class probabilities are predicted simultaneously. However, YOLO is documented to give lower accuracy for smaller objects. Since traffic lights are not expected to occupy a larger image area, Faster R-CNN method was used instead of YOLO. In the later parts of this paper, we discuss how both the methods (CNNs and Faster R-CNN) fare in terms of accuracy while detecting the presence and state of traffic lights in a given image.

2.2 Data Augmentation

Since deep networks need to be trained on a huge number of training images to achieve satisfactory performance and the original image dataset provided by Nexar contained only around 18k training images, we decided to augment the data to boost performance.

Most of the images were horizontal (i.e. normal orientation), but about 2.4% were vertical with different orientations. Although it's not a big part of the dataset but we still want our model to classify these rotated images correctly. Thus, training on rotated images was an absolute necessity. The images were flipped horizontally, zoomed, sheered by upto 20° and rotated in all possible combinations to create a new, more comprehensive dataset. All the models mentioned below were trained on this new augmented dataset.

3 Models

To solve the problem at hand, multiple CNN models were created, trained and tried. The process involved trial-and-error with both the model architectures as well as specifics like learning rates, optimizers etc. We now discuss the various models used and the techniques undertaken to train the VARACNet.

3.1 VGG-16 Transfer Learning

VGG-16[16] is a popular model which performs very well on the ImageNet dataset. Even though this model has a very high model size and number of parameters, we decided to use this model and train it on the Nexar traffic lights dataset using transfer learning.

3.2 SqueezeNet

SqueezeNet[9] is a Convolutional Neural Network architecture with accuracy comparable to that of AlexNet yet has 50 times fewer parameters. We used the SqueezeNet architecture and trained it from scratch on Nexar dataset which provided us a baseline. The model architecture is shown in Fig 14.

3.3 Custom Architectures

Since the final score also gives importance to the model size, we decided to build custom architectures which have fewer layers and relatively less parameters than SqueezeNet but give comparable performance. Further, we try to ensemble few of these architectures to see if they have synergical effect and report their results.

3.3.1 Custom Model 1

In the first model (figure 11), we introduce four convolutional layers with increasing number of feature maps (specifically 16, 32, 64 and 128). By increasing the number of feature maps in each convolutional layer, we are allowing our network to gradually learn more and more about the input as we go deeper into the network. For the first convolution layer, we use a stride of 2 in horizontal and vertical direction (found empirically) in order to control the number of parameters. Rest of the convolution layers have a stride of 1. After each of the convolutional layers, we have a max-pool layer which is responsible for providing translational invariance. After much experimentation, we found that both the kernel size and pooling size of 3×3 work best with our data. After these, we have introduced two dense layers with 128 neurons each which allows our model to take non-linear combination of the features produced by the last convolutional layer. We had also introduced dropout which makes the model more robust and allows it to generalize well on the unseen data by preventing overfitting. First three layers have 0.2 dropout fraction and the rest have the value of 0.3. ReLU activations were used in the network. Number of parameters in this model is 261,923. Architecture 2 and 3, which are described in the following sections are simple variations of this architecture.

3.3.2 Custom Model 2

In this model (figure 12), we increase the number of feature maps for the last convolutional layer from 128 to 256. We also tweak the dropout fraction of the aforementioned layer and the first dense layer to 0.2. These tweaks were done in order to see if increasing the number of parameters towards the end of the network improves the performance. Rest of the architecture of this model is same as that of the first model.

3.3.3 Custom Model 3

In this model (figure 13), we add another convolutional layer and max pool layer after the fourth layer in model 1. This model is developed by taking inspiration from both Model 1 and Model 2. The fifth convolutional layer has 256 feature maps and has a dropout of 0.2. We also tweak the dropout fraction of the dense layers to 0.5. The idea behind this model is to see if we can combine the architecture of Model 1 and Model 2 in some way to make better predictions.

3.3.4 Custom Model 4

In this model (figure 10), we tried something entirely different. We took inspiration from GoogLeNet and made use of few naive inceptions layers. In total, we have 3 convolutional layers, 2 inception modules, 4 max pooling layers, 1 average pooling layer, and 1 dense layer. First, we start with a convolutional layer which has 64 feature maps, kernel size of 3×3 and stride of 2 in horizontal as well as vertical direction. Next, we introduce a max pool layer with pool size of 3×3 and a stride of 2 in both the directions. This is followed by an inception module with three convolutional layers where each of the convolutional layers has 48 feature maps. First convolutional layer has the kernel size of 7×7, second has 3×3, and the third has 5×5. We then merge these convolutional layers into one and do the average pooling with pool size 3×3 and a stride of 2 in both the directions. We again introduce an inception module with 3 convolutional layers each with 64 feature maps. First convolutional layer has the kernel size of 1×1, second has 3×3, and the third has 5×5. We again merge these three convolutional layers and introduce a dropout of 0.3. Subsequently, we do max pooling with pooling size of 3×3 with stride of 2 in both the directions. Next, we add convolution layer with 128 feature maps and kernel size of 1×1. We max pool the output of these layers with pool size of 3×3 and stride of 2. Subsequently, we add another convolution layer with 256 feature maps and kernel size of 3×3. We max pool the output of these layers with pool size of 3×3 and stride of 2. Finally, we flatten the output from the max pool layer using a dense layer of 128 units and introduce a dropout of 0.3. Lastly, we feed the output from this layer to the softmax layer which provides the probability of input being in one of three classes.

3.4 VARACNet

In the effort of improving the performance, we decided to build an ensemble of various model we proposed. We term this ensemble model as **VARACNet**, as shown in figure 15. Since the aim of the paper is to build an architecture that has fewer parameters but gives superior performance, we experiment with ensemble of 4 different models we proposed as they all have relatively less parameters than the SqueezeNet. To that end, we first trained all the custom models on the whole dataset and saved the final parameter values. Next, we introduced a dense softmax layer with 3 units (corresponding to 3 classes) and fed the corresponding output of the candidate models to them. Afterwards, we train this ensemble of models to learn the weights of the newly introduced dense layer. These weights would decide how much say each of the individual models would get in deciding the final output. Since this ensemble model tries to combine the best of all the models, it is expected to perform better. This expected behavior is reaffirmed by our results as shown in section 4.

3.5 Faster R-CNN

Figure 8 shows the model of Faster R-CNN as proposed in the paper[15]. The model first passes the input image through a series of convolutional layers and generates a feature map. The feature map is then passed through a Region Proposal Network (RPN) which generates different region proposals. This is followed by a Fast R-CNN network which gives two outputs: bounding box coordinates and probability of each class.

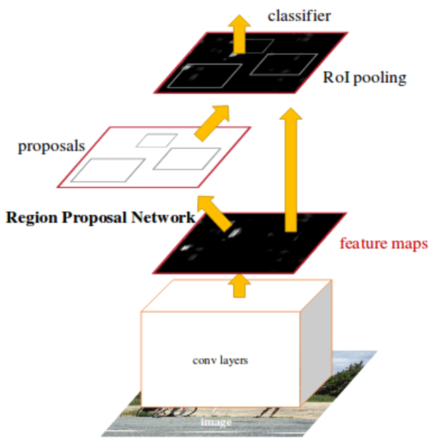

Figure 8: Faster R-CNN Model. source: [15]

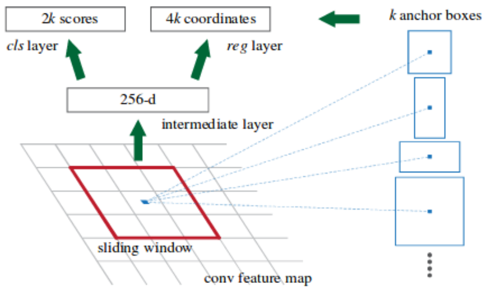

Figure 9: Region Proposal Network. source: [15]

Figure 9 shows how the RPN generates region proposal networks. It takes a 3×3 window and traverses it through whole feature map. For each window is takes k anchor boxes (boxes with varying scale and aspect ratio) centered at the window center. Here 3 scales and 3 aspect ratios are used, leading to a total of 9 anchor boxes per window ($k = 9$). The window output is then attached to an intermediate layer, followed by two fully connected layers for class scores and region box coordinates of length $2k$ and $4k$ respectively.

An RoI pooling layer is applied to the region proposals which generates fixed size pooled layers from differently sized region proposals. These are then passed to two separate fully connected layers which generate model class scores and bounding boxes.

The model was trained on the UC San Diego Traffic Light data set[18]. The Python-Caffe Faster R-CNN implementation by Ross Girshick was used. The available model was being trained for 21 classes on PASCAL data set. The code was updated to meet our requirement of 3 output classes. The existing weights were fine tuned to detect traffic lights. This model gives a mean average precision of 0.54.

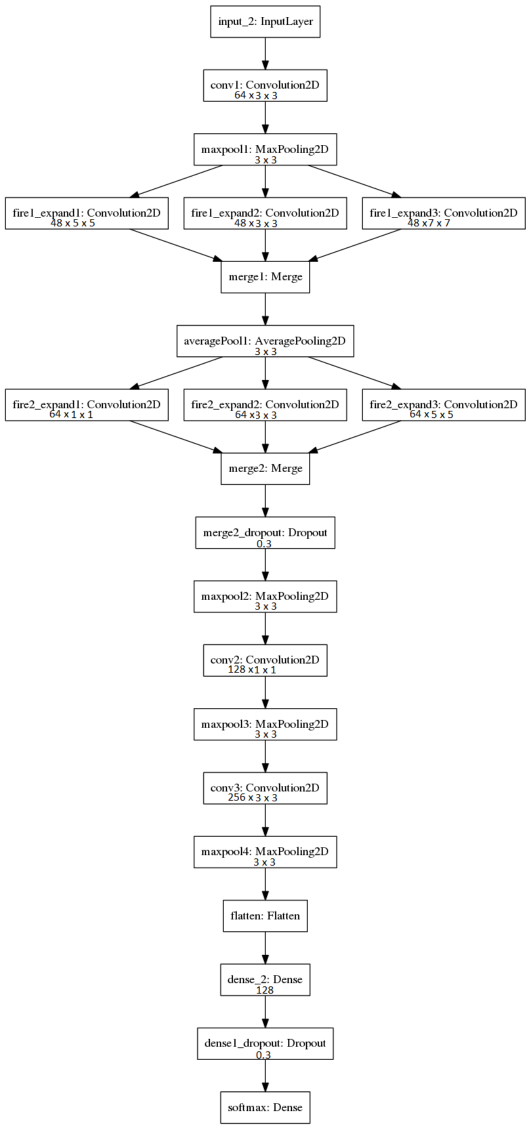

Figure 10: Model 4

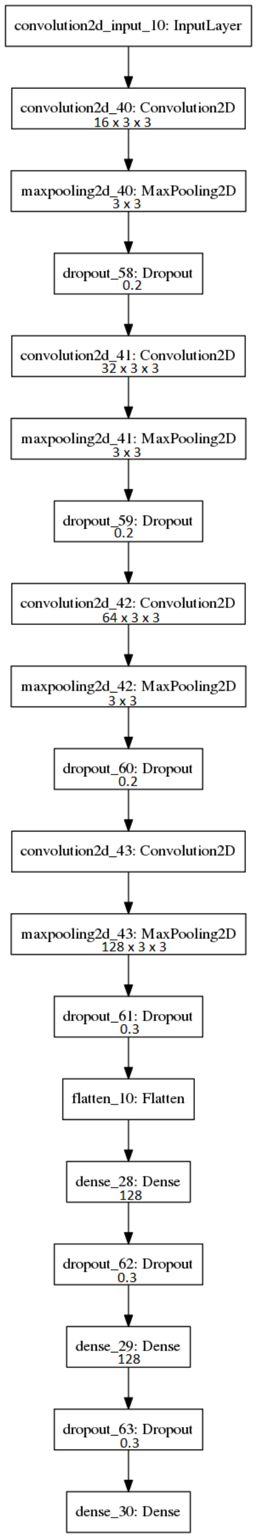
convolution2d_input_10: InputLayer
convolution2d_40: Convolution2D
16 x 3 x 3
maxpooling2d_40: MaxPooling2D
3 x 3
dropout_58: Dropout
0.2
convolution2d_41: Convolution2D
32 x 3 x 3
maxpooling2d_41: MaxPooling2D
3 x 3
dropout_59: Dropout
0.2
convolution2d_42: Convolution2D
64 x 3 x 3
maxpooling2d_42: MaxPooling2D
3 x 3
dropout_60: Dropout
0.2
convolution2d_43: Convolution2D
maxpooling2d_43: MaxPooling2D
128 x 3 x 3
dropout_61: Dropout
0.3
flatten_10: Flatten
dense_28: Dense
128
dropout_62: Dropout
0.3
dense_29: Dense
128
dropout_63: Dropout
0.3
dense_30: Dense

Figure 11: Model 1

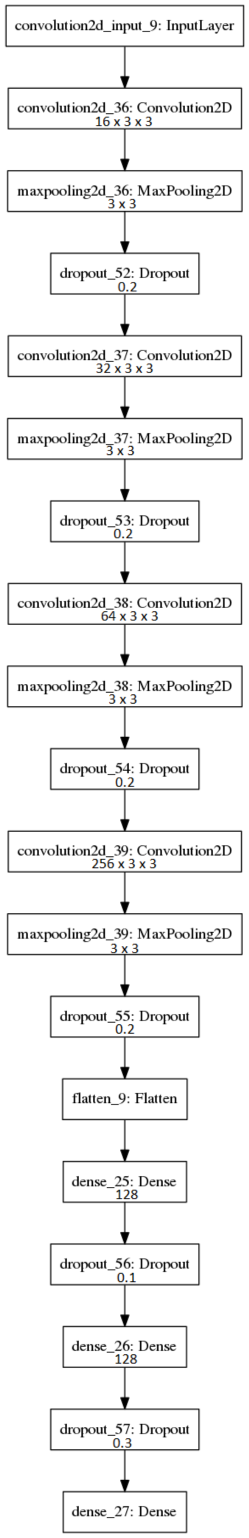
convolution2d_input_9: InputLayer
convolution2d_36: Convolution2D
16 x 3 x 3
maxpooling2d_36: MaxPooling2D
3 x 3
dropout_52: Dropout
0.2
convolution2d_37: Convolution2D
32 x 3 x 3
maxpooling2d_37: MaxPooling2D
3 x 3
dropout_53: Dropout
0.2
convolution2d_38: Convolution2D
64 x 3 x 3
maxpooling2d_38: MaxPooling2D
3 x 3
dropout_54: Dropout
0.2
convolution2d_39: Convolution2D
256 x 3 x 3
maxpooling2d_39: MaxPooling2D
3 x 3
dropout_55: Dropout
0.2
flatten_9: Flatten
dense_25: Dense
128
dropout_56: Dropout
0.1
dense_26: Dense
128
dropout_57: Dropout
0.3
dense_27: Dense

Figure 12: Model 2

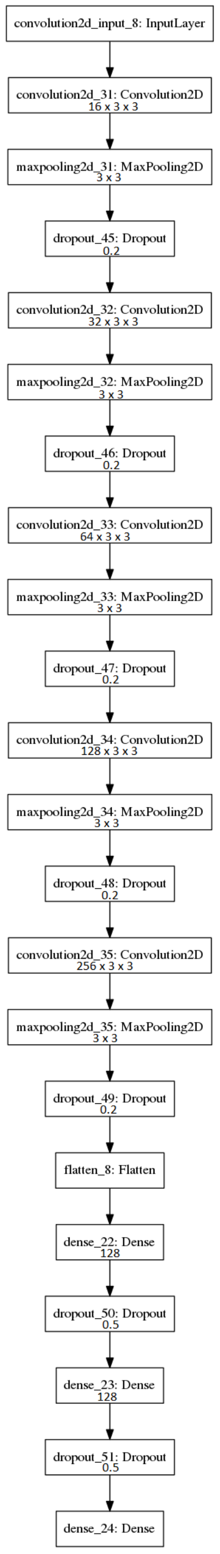
convolution2d_input_8: InputLayer
convolution2d_31: Convolution2D
16 x 3 x 3
maxpooling2d_31: MaxPooling2D
3 x 3
dropout_45: Dropout
0.2
convolution2d_32: Convolution2D
32 x 3 x 3
maxpooling2d_32: MaxPooling2D
3 x 3
dropout_46: Dropout
0.2
convolution2d_33: Convolution2D
64 x 3 x 3
maxpooling2d_33: MaxPooling2D
3 x 3
dropout_47: Dropout
0.2
convolution2d_34: Convolution2D
128 x 3 x 3
maxpooling2d_34: MaxPooling2D
3 x 3
dropout_48: Dropout
0.2
convolution2d_35: Convolution2D
256 x 3 x 3
maxpooling2d_35: MaxPooling2D
3 x 3
dropout_49: Dropout
0.2
flatten_8: Flatten
dense_22: Dense
128
dropout_50: Dropout
0.5
dense_23: Dense
128
dropout_51: Dropout
0.5
dense_24: Dense

Figure 13: Model3

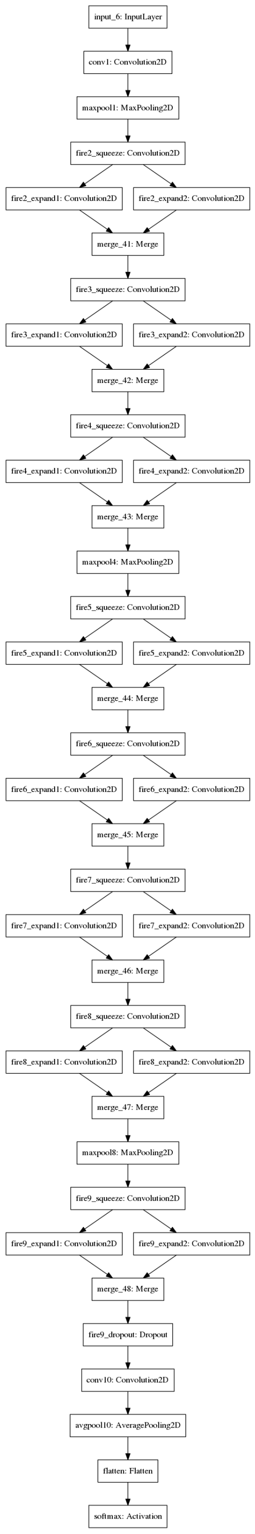
input_6: InputLayer
conv1: Convolution2D
maxpool1: MaxPooling2D
fire2_squeeze: Convolution2D
fire2_expand1: Convolution2D
fire2_expand2: Convolution2D
merge_41: Merge
fire3_squeeze: Convolution2D
fire3_expand1: Convolution2D
fire3_expand2: Convolution2D
merge_42: Merge
fire4_squeeze: Convolution2D
fire4_expand1: Convolution2D
fire4_expand2: Convolution2D
merge_43: Merge
maxpool4: MaxPooling2D
fire5_squeeze: Convolution2D
fire5_expand1: Convolution2D
fire5_expand2: Convolution2D
merge_44: Merge
fire6_squeeze: Convolution2D
fire6_expand1: Convolution2D
fire6_expand2: Convolution2D
merge_45: Merge
fire7_squeeze: Convolution2D
fire7_expand1: Convolution2D
fire7_expand2: Convolution2D
merge_46: Merge
fire8_squeeze: Convolution2D
fire8_expand1: Convolution2D
fire8_expand2: Convolution2D
merge_47: Merge
maxpool8: MaxPooling2D
fire9_squeeze: Convolution2D
fire9_expand1: Convolution2D
fire9_expand2: Convolution2D
merge_48: Merge
fire9_dropout: Dropout
conv10: Convolution2D
avgpool10: AveragePooling2D
flatten: Flatten
softmax: Activation

Figure 14: SqueezeNet

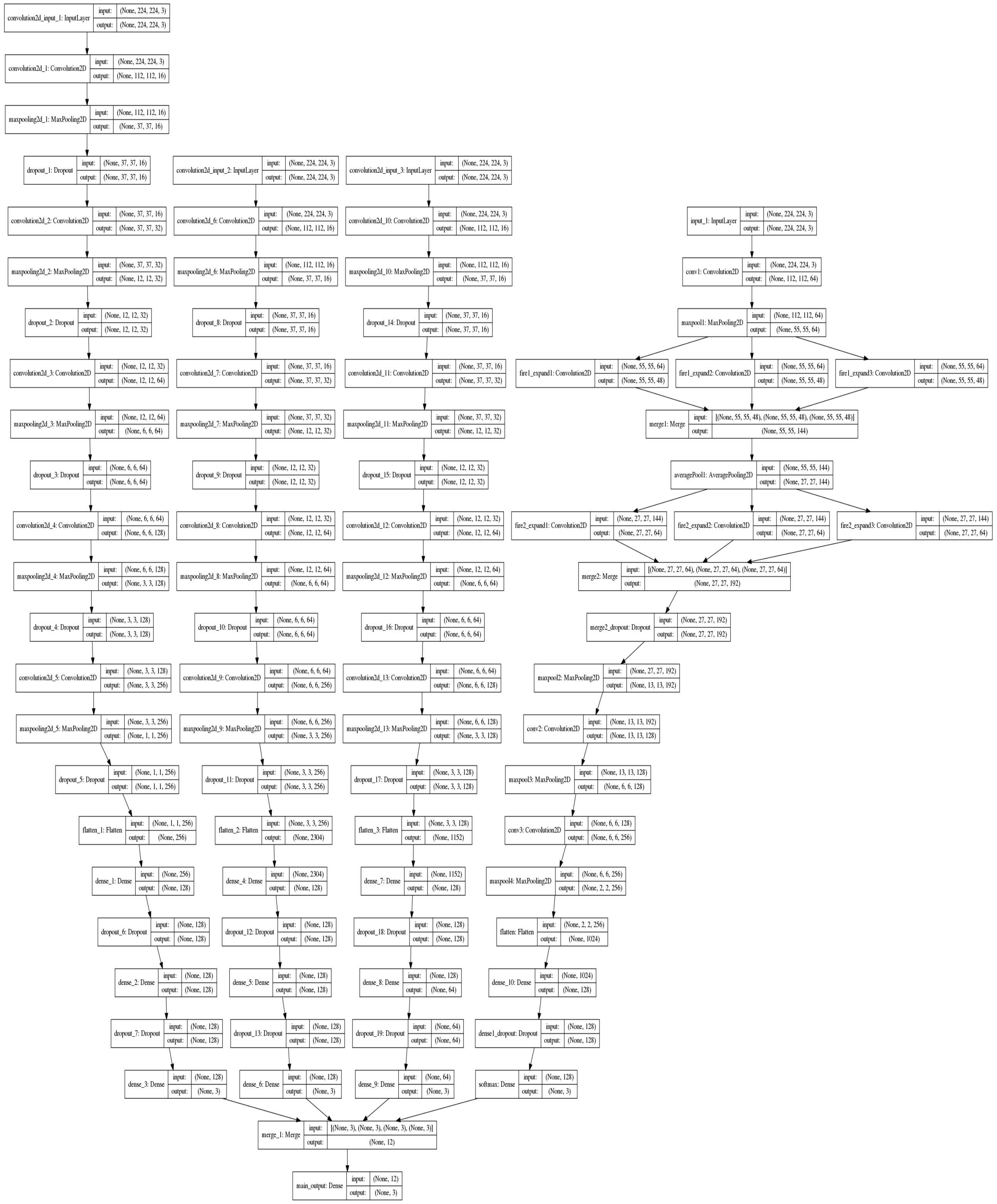
convolution2d_input_1: InputLayer
convolution2d_1: Convolution2D
maxpooling2d_1: MaxPooling2D
dropout_1: Dropout
convolution2d_2: Convolution2D
maxpooling2d_2: MaxPooling2D
dropout_2: Dropout
convolution2d_3: Convolution2D
maxpooling2d_3: MaxPooling2D
dropout_3: Dropout
convolution2d_4: Convolution2D
maxpooling2d_4: MaxPooling2D
dropout_4: Dropout
convolution2d_5: Convolution2D
maxpooling2d_5: MaxPooling2D
dropout_5: Dropout
flatten_1: Flatten
dense_1: Dense
dropout_6: Dropout
dense_2: Dense
dropout_7: Dropout
dense_3: Dense
convolution2d_input_2: InputLayer
convolution2d_6: Convolution2D
maxpooling2d_6: MaxPooling2D
dropout_8: Dropout
convolution2d_7: Convolution2D
maxpooling2d_7: MaxPooling2D
dropout_9: Dropout
convolution2d_8: Convolution2D
maxpooling2d_8: MaxPooling2D
dropout_10: Dropout
convolution2d_9: Convolution2D
maxpooling2d_9: MaxPooling2D
dropout_11: Dropout
flatten_2: Flatten
dense_4: Dense
dropout_12: Dropout
dense_5: Dense
dropout_13: Dropout
dense_6: Dense
convolution2d_input_3: InputLayer
convolution2d_10: Convolution2D
maxpooling2d_10: MaxPooling2D
dropout_14: Dropout
convolution2d_11: Convolution2D
maxpooling2d_11: MaxPooling2D
dropout_15: Dropout
convolution2d_12: Convolution2D
maxpooling2d_12: MaxPooling2D
dropout_16: Dropout
convolution2d_13: Convolution2D
maxpooling2d_13: MaxPooling2D
dropout_17: Dropout
flatten_3: Flatten
dense_7: Dense
dropout_18: Dropout
dense_8: Dense
dropout_19: Dropout
dense_9: Dense
input_1: InputLayer
conv1: Convolution2D
maxpool1: MaxPooling2D
fire1_expand1: Convolution2D
fire1_expand2: Convolution2D
fire1_expand3: Convolution2D
merge1: Merge
averagePool1: AveragePooling2D
fire2_expand1: Convolution2D
fire2_expand2: Convolution2D
fire2_expand3: Convolution2D
merge2: Merge
merge2_dropout: Dropout
maxpool2: MaxPooling2D
conv2: Convolution2D
maxpool3: MaxPooling2D
conv3: Convolution2D
maxpool4: MaxPooling2D
flatten: Flatten
dense_10: Dense
dense1_dropout: Dropout
softmax: Dense
merge_1: Merge
main_output: Dense

Figure 15: VARACnet

4 Experiments and Results

4.1 State of the art results

Table 1 shows state of the art results from the leaderboard of Nexar Challenge[2]

Table 1: Leadership Board and Challenge Results[2]

Rank	Classification Accuracy(%)	Model Size Score	Challenge Score
1	94.41	0.9741	0.9197
2	93.62	0.9716	0.9097
3	90.80	0.9886	0.8977

4.2 Accuracy and performance of different models

Although the model size of VGG-16 is almost 528 MB, we wanted to investigate how it performs through transfer learning on the Nexar dataset. We trained only the last two dense layers of VGG-16[16] network and unfortunately, the accuracy was just 56%, which would have not yielded a great challenge score. We also performed transfer learning by fine tuning a pre-trained SqueezeNet model - trained on ImageNet dataset, to classify the traffic light images. We trained only the last softmax layer of the model. Adam optimizer with the hyper parameters $\eta = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $decay = 0.01$ was used for training. However, even this yielded an accuracy of only around 50%. Hence, we decided not to move forward with doing transfer learning on VGG-16 or related architectures like AlexNet/GoogLeNet. Given that the nature of images is very different and owing to the low accuracy percentage, we decide to train our models from scratch.

Table 2 shows the classification accuracies and challenge scores of the models designed.

Table 2: Parameters and accuracies of different models

Model Name	Classification Accuracy(%)	Number of Parameters	Challenge Score
Model 1	88.54	261,923	0.8838
Model 2	89.90	483,135	0.894
Model 3	89.35	442,403	0.8907
Model 4	88.1	640,163	0.8771
SqueezeNet	87.7	712,697	0.8726
VARACNet	91.7	1,827,624	0.9053

SqueezeNet model was fully trained on the Nexar dataset. The hyper parameters were kept same as ones used during fine tuning. With this we were able to achieve an accuracy of 87% (fig. 16). Figure 17 shows that the validation loss begins to converge at around 40 epochs. It actually started to increase after 60 epochs and since early stopping was in place training had stopped.

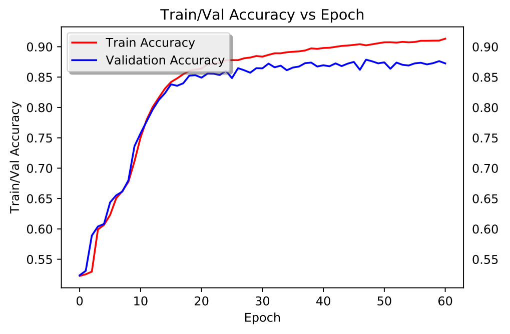

Figure 16: SqueezeNet Accuracy

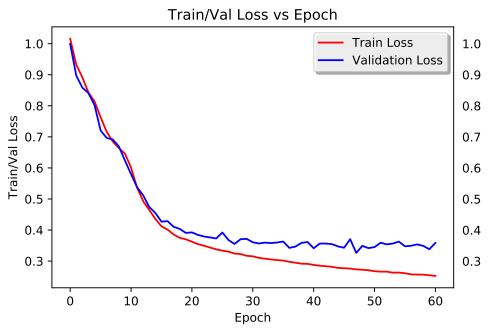

Figure 17: SqueezeNet Error

Figure 18 show that Model 1 achieves validation accuracy of 88.54% and training accuracy of 86.76%. From the Figure 19 we can see that the validation loss converges to the value around 0.31 and training loss converges around 0.35 around 10 iterations. These results show that our model performs better than the Squeeze-net model.

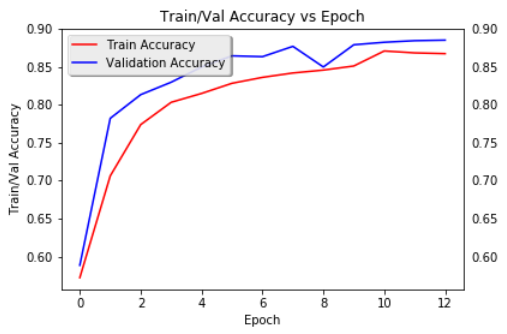

Figure 18: Model 1 Accuracy Plots

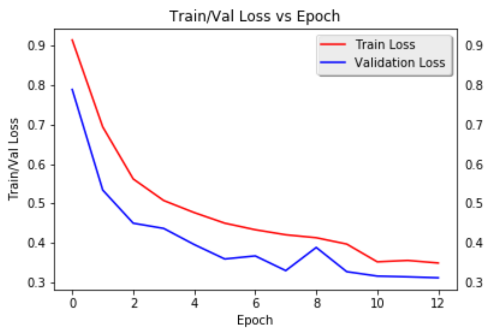

Figure 19: Model 1 Error plots

Figure 20 shows that Model 2 achieves validation and training accuracies which are actually better than model 1 and SqueezeNet. From the Figure 21 we can see that the validation loss converges to the value around 0.31 and training loss converges around 0.37 in about ten iterations.

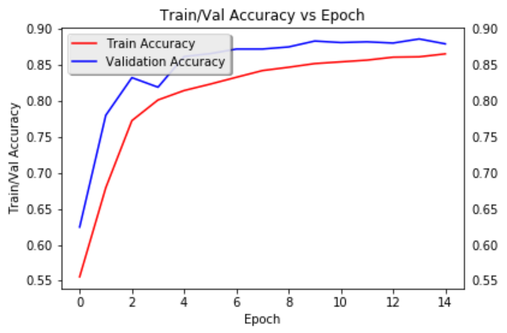

Figure 20: Model 2 Accuracy Plots

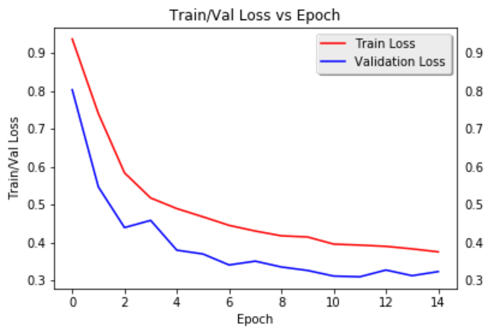

Figure 21: Model 2 Error plots

Figure 22 shows that Model 3 achieves validation and training accuracies which are better than model 1 and SqueezeNet but worse than Model 2. From the Figure 23 we can see that the validation loss converges to the value around 0.33 and training loss converges around 0.38 in about fifteen iterations.

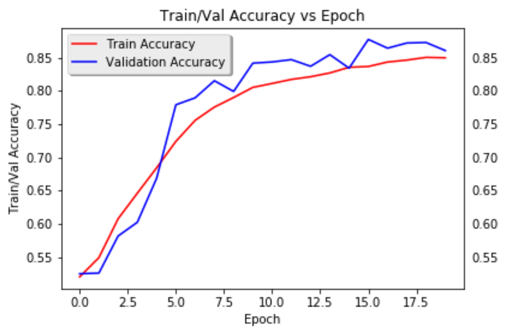

Figure 22: Model 3 Accuracy Plots

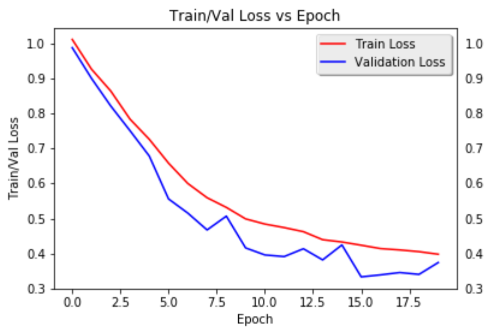

Figure 23: Model 3 Error plots

Figure 24 shows the validation and training accuracies on the inception based architecture (Model 4). From the Figure 25 we can see that the validation loss converges to the value around 0.32 and training loss converges around 0.33 in about twenty iterations.

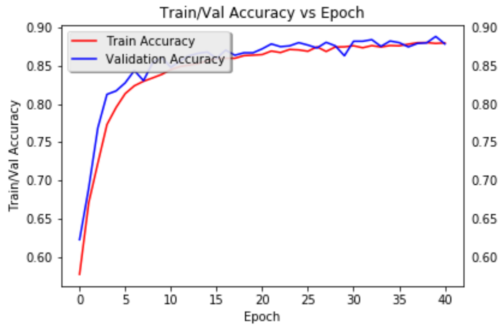

Figure 24: Model 4 Accuracy Plots

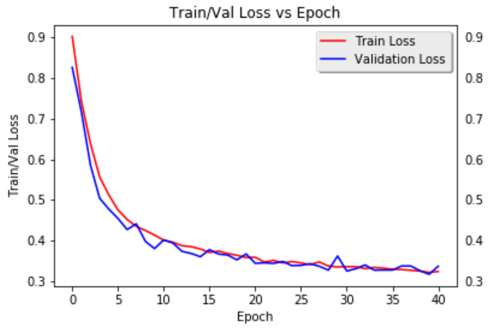

Figure 25: Model 4 Error plots

Figure 26 shows the error during training the Faster R-CNN model. The validation loss begins to converge in about sixty epochs. Figure 27 shows the accuracy plot of the ensemble model against number of training iterations. We observe that the learned weights increase the accuracy by almost 2% over uniformly weighted average of all the model outputs.

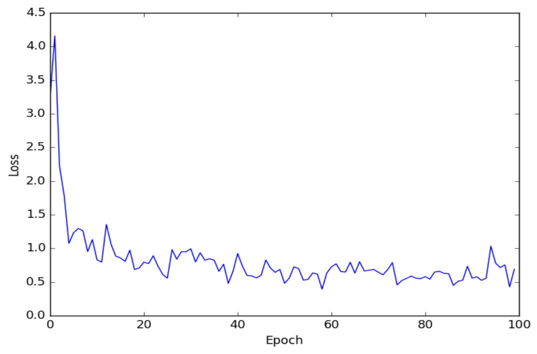

Figure 26: Faster R-CNN Error

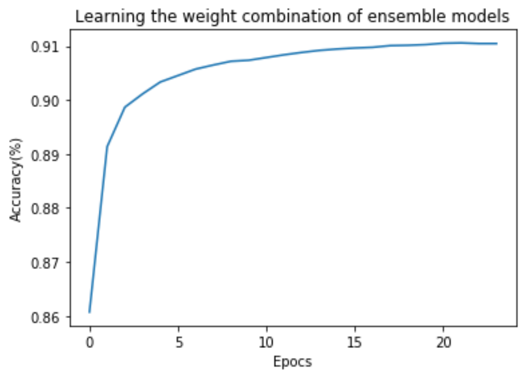

Figure 27: Plot of Ensemble accuracy vs Epocs

Figure 28, 29, and 30 show the output of the localization task. From the results, we can see that our model is able to localize the traffic lights far far away and even under variety of background conditions.

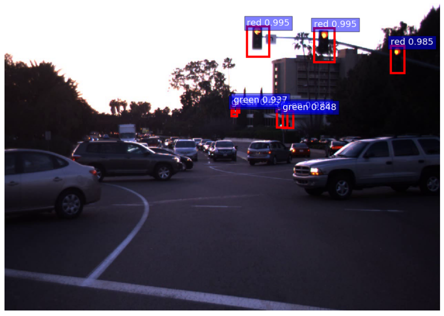

Figure 28: UCSD Dataset Day Image

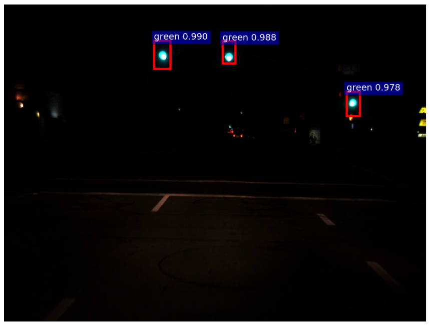

Figure 29: UCSD Dataset Night Image

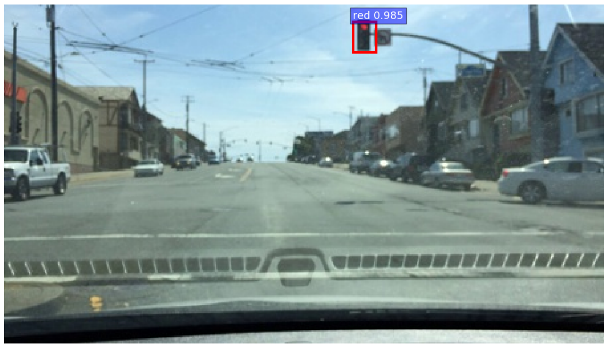

Figure 30: Nexar dataset Classification

4.3 Robustness of the ensemble model

To test the robustness of our model against traffic light images, the team captured photographs of streets around UC San Diego. Predictions on these custom images with different image definitions, without proper street view, and some with more than one traffic light, were satisfactory. If the number of traffic lights in an image is greater than one, then the prediction is based on the traffic light in the direction of driving. Figure 31 - 35 are the images which were correctly classified.

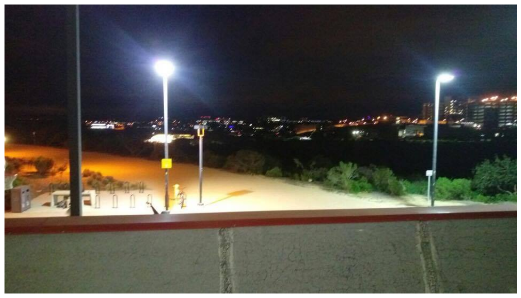

Figure 31: Image with no traffic light

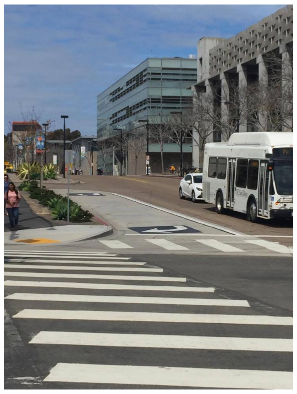

Figure 32: Images with no traffic light

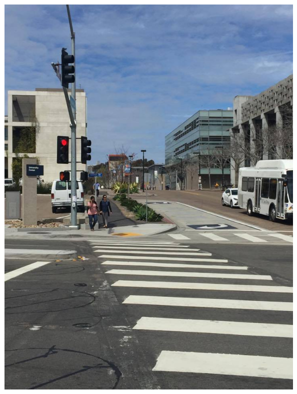

Figure 33: Image with red traffic light

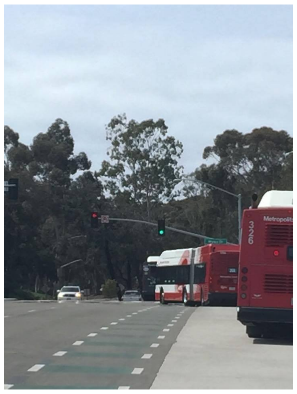

Figure 34: Image with green traffic light

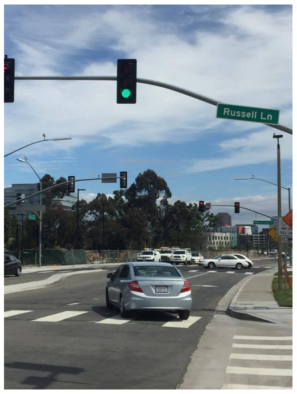

Figure 35: Image with green traffic lights

4.4 Experiments that did not work

Many of our experiments did not succeed the first time we performed them. There were plenty of experiments that we tried but they did not work out too well. This included separating the images into day and night images and training separate CNNs on each of them. Also, we tried to merge the small custom CNN models by using a functional model rather than taking their weighted average. Even this did not improve the accuracy further than what we had achieved using a weighted average. Apart from these, we thought that cropping the image and retaining the top half of it would be helpful as the top part is where the traffic light would be. According to this logic, the lower part should add noise to the dataset. However, cropping images reduced the validation accuracy and we did not pursue this further. Batch normalization didn't seem to work either, given the fact that we had dropout after each maxpool layer.

5 Discussion and Conclusion

The first place solution to Nexar Challenge[4] had a model score of 0.919, whereas we managed a challenge score of 0.9053. It is worthwhile to notice that the person who stood first in this competition manually modified the labels of about 700 images in the training data whereas we did not modify any labels. Correcting the wrongly labelled images helped increase his model's accuracy by 0.6%. Also, VARACNet was able to beat the challenge score of third rank solution and stood behind the second rank solution by a margin of only 0.0044. This clearly indicates that VARACNet is an efficient ensemble of multiple Convolutional Neural Networks built and trained from scratch. Each of these sub-CNNs are small in terms of number of parameters contained in them, which results in a total of around 1.8 million parameters for VARACNet. The performance of most of these mini-CNNs is better than that of a SqueezeNet, which has comparatively larger number of parameters. Since SqueezeNet's performance is comparable to that of AlexNet as per [9], we can safely infer that our models perform as well as AlexNet at least on this dataset and for this problem. Traffic lights in Nexar dataset and the objects in ImageNet are two very different types of images and hence, a transfer learning mechanism wherein we train only the last few layers is unlikely to work here. This is evident from the fact that when VGG-16 and SqueezeNet were transfer learned on the Nexar dataset, the performance was very unsatisfactory.

It is interesting to note that VARACNet has a dropout layer after almost every max-pool/full connected layer (in its sub-CNNs). This was done for regularization purposes and to avoid over-fitting on the training dataset, whose number exceeded 100k after data augmentation. Also, the smaller size of the network makes it less prone to over-fitting as compared to bigger networks like AlexNet and VGGNet, for smaller dataset. Both these features combined makes our network roboust to the over-fitting issue.

The idea of having exponentially increasing the number of feature maps in our models was inspired from SqueezeNet. SqueezeNet has feature maps which exponentially increase and then decrease in number, and this is followed in cycle, without use of fully connected layer. We decided to make use of this unique property of SqueezeNet and combine it with some traditional fully connected layers and the results are encouraging. From this we can hypothesise that the model is first learning abstract level of features and as we go deep into the network finer details are extracted. This leads to good learning with comparably fewer parameters.

In Model 4, we have used the inception module and this model gives a pretty good accuracy even with its small size. In an inception module, we have varying size filters extracting features simultaneously from the same input in parallel. These feature maps are combined through depth-wise concatenations. An advantage of building such hybrid feature maps would be having the possibility of looking at both finer and coarse level features at the same time.In our problem set, the objects of interest (traffic light boundaries) are of regular shape (rectangular). This gives additional advantage to the model where smaller kernel will be able to identify corner/line region with good precision and the larger kernels will be able to combine this information with the features it extracts.

We see that models with less parameters and depth can perform as well as SqueezeNet, a comparatively deeper network. A possible reason for these comparatively shallow networks being able to recognize the traffic lights could be the fact that the number of classes and the features to learn are less for the problem at hand. To correctly classify traffic lights, a network needs filters to be trained to identify the traffic light boundary and then fire neurons to detect either of green or red light. In case of non-identification of a traffic light boundary, it can safely predict that no traffic light exists in the current image.

Note that Model 2 performs marginally better than Model 3, even though Model 3 has greater depth and more parameters. This tells us that deeper networks are not necessarily better[19] in terms of performance when it comes to classification using CNNs. Model 2 performs better than Model 3 probably because it extracts more information at in the earlier stages of the network (closer to the actual image).

From above discussions we can conclude that deep networks are not always the best option to go for. If the number of classes to classify is less, it is computationally efficient to create small networks which can learn reasonably well and combine them. Also, having increasing number of feature maps is a good way to extract information from image.

6 Concept and Innovation

The value provided by these kind of experiments is immense, especially with increasing number of attempts being made to build driver-less cars which can sense their environment and navigate safely. Although to build such autonomous vehicles many different disciplines have to join hands, visual perception is a very important aspect to look at. Thus, this project is of high practical relevance in the current world. Also, the approach we used to solve this problem through relatively simple models makes it possible for deploying them on low memory machines/cars and using them for real-time detection.

SqueezeNet was chosen initially because of its small size (50 times fewer parameters than AlexNet) and yet AlexNet level accuracy. However, by trying a simple custom CNN of three to four layers, we found good accuracy on the validation set. This made us believe that since the problem at hand is simpler than the object classification problem as in ImageNet, a comparatively shallow network should be able to perform well. This is where we started to experiment with various model architectures and came up with multiple models that gave more than 88% accuracy on both training and validation sets. Since this accuracy percentage was greater than that of SqueezeNet, we shifted our focus from SqueezeNet ensembles to custom architecture ensembles.

We designed four custom models. Each model had variation in terms of depth of network, receptive field, dropout percentage, inception modules, training methodology, number of filters etc. We experimented with a naive inception type architecture in Model 4, which had three parallel convolution filters of size 3x3, 5x5 and 7x7, and merged them in the later layers. As we flowed down the network, we gradually increased the number of filter maps from $64 \rightarrow (48, 48, 48) \rightarrow 96$ which performs better as compared to equal number of filter across all the layers. Also, having three convolution filter maps in parallel had better performance as compared to two filter maps in parallel.

Compared to any of the standard CNN models like SqueezeNet, VGGNet, AlexNet, our model could give comparable classification accuracy. The models could be trained on our personal computers with training time close to 3-4 hours. The small size of our models enables it to be incorporated in any low memory device, without effecting the system's performance. Further, this would help the system operate in real-time without any evident lag or having significant processing time.

Since the Nexar dataset images did not provide annotations of traffic lights, manually annotating all the traffic lights would have been an impractical task, even by using tools like Sloth[3]. This made us approach the problem in a different way. UC San Diego CVRR lab has a traffic lights dataset that provides annotations for all the traffic lights in a given image. For object localization, we used Faster R-CNN model on this dataset. We used the Faster R-CNN model and trained it from scratch on the UCSD CVRR dataset. The model was validated on the test set and we found that the model correctly located the traffic lights with high confidence level. Now with this trained model, we tested it against the Nexar dataset and observed that the localization model did not perform as competitively as it performed on UCSD CVRR dataset. The model was able to localize the traffic lights but with a lower confidence level. This could be attributed to the fact that the UCSD CVRR and Nexar datasets had different image resolutions and aspect ratios.

7 Individual Contributions

This project is, in true sense, a combined team effort wherein everyone dedicated equal amount of effort and hours. The team sat down together for long hours every day working on the project and helped each other whenever someone got stuck. All the team members were involved in all the aspects of the project (data processing, coding, debugging, analysis of generated samples, experimentation, writing report etc.) and added their perspective which led the to successful completion of the project. At a high level, each of us implemented one of the sub-models of the VARACNet. Every person was responsible for the fine tuning of his own model and for plotting the graphs. This was preceded by transfer learning of VGG-16 and SqueezeNet, and full training of SqueezeNet, which was carried out by Vamshi, Aditya and Chetan. Faster R-CNN was handled primarily by Akshaya and Rishabh. The ensemble part was undertaken by everyone together by projecting the code on to a screen and providing inputs while one person handled the code.

References

[1] *Using Deep Learning For Traffic Light Recognition, The Nexar Challenge #1*

[2] *Nexar Challenge Results*

[3] *Sloth*

[4] *First Place Solution Blog*

[5] Nathaniel Fairfield and Chris Urmson *Traffic Light Mapping and Detection*, ICRA 2011

[6] J. Gong, Y. Jiang, G. Xiong,C. Guan, G. Tao and Huiyan Chen *The Recognition and Tracking of Traffic Lights Based on Color Segmentation and CAMSHIFT for Intelligent Vehicles*, In proceedings of IV, 2010

[7] R de Charette, F Nashashibi *Real time visual traffic lights recognition based on spot light detection and adaptive traffic lights templates*, In proceedings of IV, 2009

[8] V. John, K. Yoneda, B. Qi, Z. Liu, and S. Mita *Traffic Light Recognition in Varying Illumination using Deep Learning and Saliency Map*, ITSC 2014

[9] Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, Kurt Keutzer *SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size*.

[10] Xavier Frazao and Luis A. Alexandre *Weighted Convolutional Neural Network Ensemble*, CIARP 2014

[11] Pierre Sermanet and Yann LeCun *Traffic Sign Recognition with Multi-Scale Convolutional Networks*, IJCNN 2011

[12] Ross Girshick, Jeff Donahue, Trevor Darrell, Jitendra Malik *Rich feature hierarchies for accurate object detection and semantic segmentation Tech report (v5)*

[13] Selective Search for Object Recognition J.R.R. Uijlings, K.E.A. van de Sande,T. Gevers and A.W.M. Smeulders *Selective Search for Object Recognition*

[14] Ross Girshick *Fast R-CNN*

[15] Shaoqing Ren, Kaiming He, Ross Girshick and Jian Sun *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*

[16] Simonyan Ken and Zisserman Andrew *Very Deep Convolutional Networks for Large-Scale Image Recognition*

[17] Joseph Redmon, Ali Farhadi *YOLO9000:Better, Faster, Stronger*

[18] *VIVA CVRR UC San Diego Traffic Lights data set*

[19] Jimmy Ba Lei and Caruana Rich *Do Deep Nets Really Need to Be Deep*