**ROAD SAFETY**

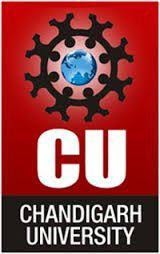
**Final Project Report**

Submitted in partial fulfillment of the requirements for the award of degree of

**BACHELOR OF ENGINEERING**

**IN**

**COMPUTER SCIENCE & ENGINEERING**

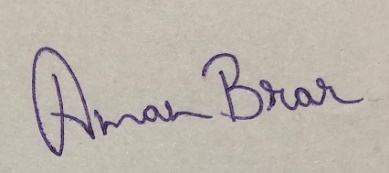


**Submitted to:**

**Dr. Shanky Goel**

**Mentor:**

**Er. Amandeep Kaur**

****

**Submitted By:**

**Aditya**

**UID- 18BCS6656**

**Shivam**

**UID- 18BCS3054**

**Kanishka Singh Rathore**

**UID- 18BCS6634**

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

**CERTIFICATE**

This is to certify that the work embodied in this Project Report entitled “Road Safety” being submitted by  “  Aditya, Shivam, Kanishka”  - UID “18BCS6656,18BCS3054,18BCS6634 respectively ” , of  6th Semester for partial fulfillment of the requirement for the degree of  “ Bachelor of Engineering in Computer Science & Engineering ” discipline in “ Chandigarh University ” during the academic session July-December 2021  is a record of bonafide piece of work, carried out by student under my supervision and guidance in the “ Department of Computer Science & Engineering ”, Chandigarh University**.**

**DECLARATION**

I, student of Bachelor of Engineering in Computer Science & Engineering, 6th Semester , session: Jan – April 2021, Chandigarh University, hereby declare that the work presented in this Project Report entitled “Road safety” is the outcome of my own work and team work, is bonafide and correct to the best of my knowledge and this work has been carried out taking care of Engineering Ethics. The work presented does not infringe any patented work and has not been submitted to any other university or anywhere else for the award of any degree or any professional diploma.

**Student details and Signature**

Aditya              UID: 18bcs6656

     Kanishka                   UID: 18bcs6634

     Shivam               UID: 18bcs3054

**GUIDED BY: ER. Amandeep Kaur**

**INDEX**

1. **INTRODUCTION**
2. **PROJECT SETUP**
3. **CONVOLUTIONAL NEURAL NETWORK**
4. **TRAFFIC SIGN RECOGNIZATION**
5. **CONCLUSIONS**
6. **DISCUSSION**
7. **DIFFICUTIES**
8. **FUTURE WORK**
9. **BIBLOGRAPHY**

**INTRODUCTION**

In today’s world as the number of vehicles are increasing so are the road accidents and according to reports, India is on 1st spot in most number of accidents in a country. This is caused due to many reasons such as poor enforcement of laws, carelessness etc. One of the reason is that people don’t recognize or follow traffic sign boards. So we have made a traffic sign recognizer which can inform the driver of the vehicle about the traffic sign coming ahead and to follow it. This can reduce the road accidents.

Traffic signs are an integral part of our road infrastructure. They provide critical information, sometimes compelling recommendations, for road users, which in turn requires them to adjust their driving behaviour to make sure they adhere with whatever road regulation currently enforced. Without such useful signs, we would most likely be faced with more accidents, as drivers would not be given critical feedback on how fast they could safely go, or informed about road works, sharp turn, or school crossings ahead. In our modern age, around [1.3M people](http://asirt.org/initiatives/informing-road-users/road-safety-facts/road-crash-statistics) die on roads each year. This number would be much higher without our road signs.  
Naturally, autonomous vehicles must also abide by road legislation and therefore recognize and understand traffic signs.

Traditionally, standard [computer vision](https://en.wikipedia.org/wiki/Computer_vision) methods were employed to detect and classify traffic signs, but these required considerable and time consuming manual work to handcraft important features in images. Instead, by applying deep learning to this problem, we create a model that reliably classifies traffic signs, learning to identify the most appropriate features for this problem by itself.



**Project Setup**

The dataset is plit into training, test and validation sets, with the following characteristics:

* Images are 32 (width) x 32 (height) x 3 (RGB color channels)
* Training set is composed of 34799 images
* Validation set is composed of 4410 images
* Test set is composed of 12630 images
* There are 43 classes (e.g. Speed Limit 20km/h, No entry, Bumpy road, etc.)

Moreover, we will be using Python 3.5 with Tensorflow to write our code.

## Images and Distribution

You can see below a sample of the images from the dataset, with labels displayed above the row of corresponding images. Some of them are quite dark so we will look to improve contrast a bit later.

# Convolutional Neural Networks

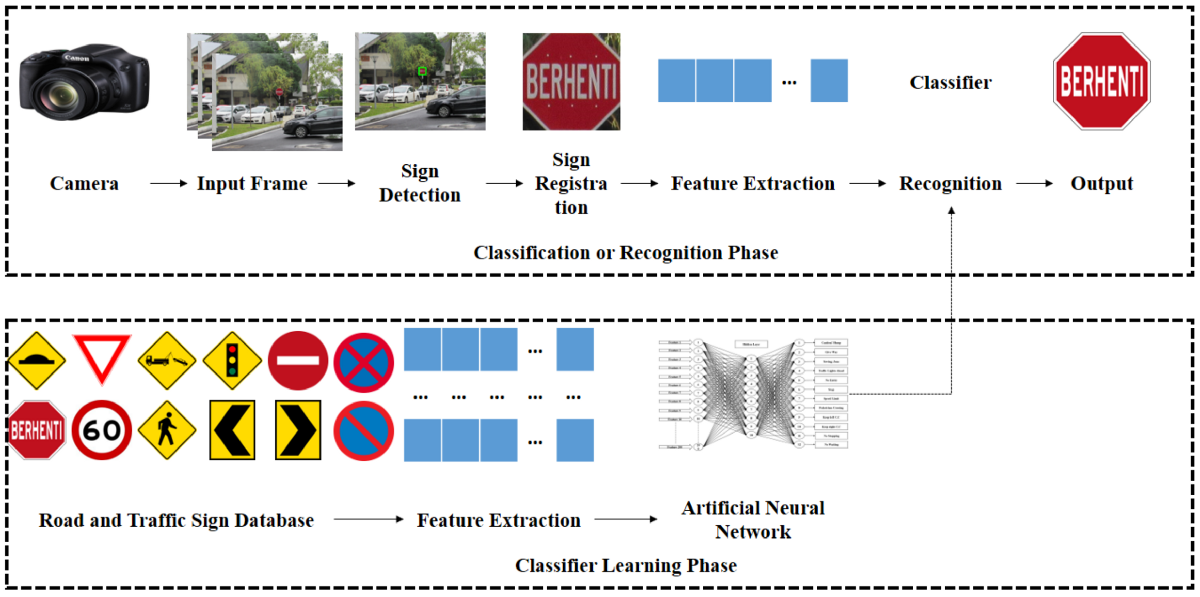
Convolutional neural networks are a part of deep learning and extensively used in image recogntion. These convolutional neural networks consists of several layers. First a Conv2D layer is used for feature extraction with the help of filters. Number of filters are generally in power of 2 like 32, 64 or 128. An activation function is used in this layer. Generally ReLU(Rectified Linear Unit) activation function is used. ReLU function is defined as maximum(0, x).

Next is the max pooling layer which is used reduce the dimensions of the image. This is done to reduce the computation power required for processing the image. Third is dropout layer. This dropout layer is used to prevent overfitting and to reduce the complexity of the model. In this layer some neurons are removed randomly.

The combination of first 3 layers is called feature learning phase. These 3 layers are used multiple times to improve the training.

Fourth is the flatten layer which converts the 2-D data into a long 1-D vector of features for a fully connected layer that can be fed into the neural network.

The last layer is the dense layer which is used as a output layer. The last layer has number of nodes same as the number of classes. The last dense layer uses softmax activation function. Softmax function gives the probability value (between 0 and 1) so that the model can predict which class has the highest probability.



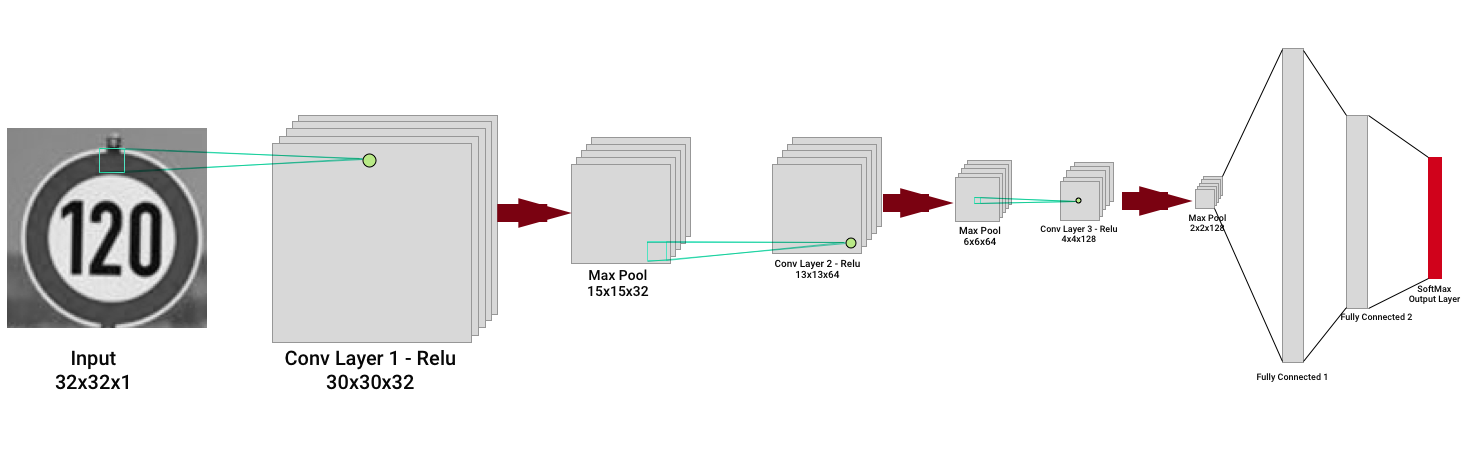
# Traffic sign recognition

Road traffic accidents are primarily caused by drivers error. Safer roads infrastructure and facilities like traffic signs and signals are built to aid drivers on the road. But several factors affect the awareness of drivers to traffic signs including visual complexity, environmental condition, and poor drivers education. This led to the development of different ADAs like TSDR that enhances vehicle system. More complex algorithms are implemented for improvement but this affects the performance of a real-time system. This study implements a real-time traffic sign detection and recognition system with voice alert using Python. It aims to establish the proper trade-off between accuracy and speed in the design of the system. Four pre-processing and object detection methods in different color spaces are evaluated for efficient, accurate, and fast segmentation of the region of interest. In the recognition phase, ten classification algorithms are implemented and evaluated to determine which will provide the best performance in both accuracy and processing speed for traffic sign recognition. This study has determined that Shadow and Highlight Invariant Method for the pre-processing and color segmentation stage provided the best trade-off between detection success rate (77.05%) and processing speed (31.2ms). Convolutional Neural Network for the recognition stage not only provided the best trade-off between classification accuracy (92.97%) and processing speed (7.81ms) but also has the best performance even with lesser number of training data. Embedded system implementation utilized Nvidia Jetson Nano with interface Waveshare IMX219-77 camera, Nvidia 7” LCD and generic speaker and programmed in Python with OpenCV, sci-kit learn and Pytorch libraries. It is capable of running at an adaptive frame rate from 8-12 frames per second with no detection and down to approximately 1 frame per second when there is a traffic sign detected.

# Model Architecture

The architecture proposed is inspired from Yann Le Cun’s [paper](http://yann.lecun.com/exdb/publis/pdf/sermanet-ijcnn-11.pdf) on classification of traffic signs. We added a few tweaks and created a modular codebase which allows us to try out different filter sizes, depth, and number of convolution layers, as well as the dimensions of fully connected layers. In homage to Le Cun, and with a touch of cheekiness, we called such network **EdLeNet** :).

We mainly tried 5x5 and 3x3 filter (aka kernel) sizes, and start with depth of 32 for our first convolutional layer. EdLeNet’s 3x3 architecture is shown below:



EdLeNet 3x3 Architecture

The network is composed of 3 convolutional layers — kernel size is 3x3, with depth doubling at next layer — using [ReLU](https://en.wikipedia.org/wiki/Rectifier_(neural_networks)) as the activation function, each followed by a 2x2 max pooling operation. The last 3 layers are fully connected, with the final layer producing 43 results (the total number of possible labels) computed using the [SoftMax](https://www.quora.com/Why-is-softmax-activate-function-called-softmax) activation function. The network is trained using mini-batch stochastic gradient descent with the [Adam](https://www.quora.com/Can-you-explain-basic-intuition-behind-ADAM-a-method-for-stochastic-optimization) optimizer. We build a highly modular coding infrastructure that enables us to dynamically create our models like in the following snippets:

mc\_3x3 = ModelConfig(EdLeNet, "EdLeNet\_Norm\_Grayscale\_3x3\_Dropout\_0.50", [32, 32, 1], [3, 32, 3], [120, 84], n\_classes, [0.75, 0.5])  
mc\_5x5 = ModelConfig(EdLeNet, "EdLeNet\_Norm\_Grayscale\_5x5\_Dropout\_0.50", [32, 32, 1], [5, 32, 2], [120, 84], n\_classes, [0.75, 0.5])  
  
me\_g\_norm\_drpt\_0\_50\_3x3 = ModelExecutor(mc\_3x3)  
me\_g\_norm\_drpt\_0\_50\_5x5 = ModelExecutor(mc\_5x5)

The ModelConfig contains information about the model such as:

* The model function (e.g. EdLeNet)
* the model name
* input format (e.g. [32, 32, 1] for grayscale),
* convolutional layers config [filter size, start depth, number of layers],
* fully connected layers dimensions (e.g. [120, 84])
* number of classes
* dropout keep percentage values [p-conv, p-fc]

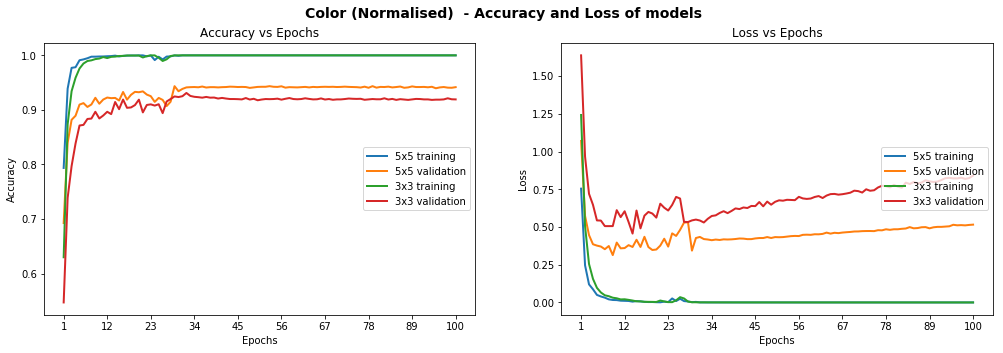
The ModelExecutor is reponsible for training, evaluating, predicting, and producing visualizations of our activation maps.

To better isolate our models and make sure they do not all exist under the same Tensorflow graph, we use the following useful construct:

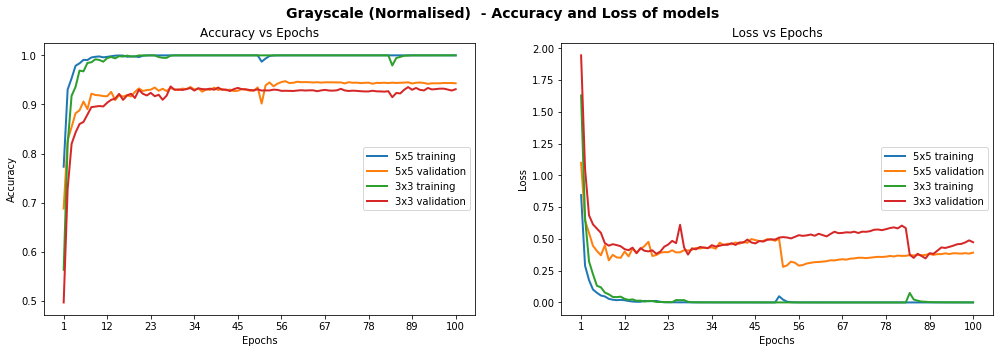
self.graph = tf.Graph()  
with self.graph.as\_default() as g:  
 with g.name\_scope( self.model\_config.name ) as scope:  
  
...  
  
with tf.Session(graph = self.graph) as sess:

This way, we create separate graphs for every model, making sure there is no mixing of our variables, placeholders etc. It’s saved me a lot of headaches.

We actually started with a convolutional depth of 16, but obtained better results with 32 so settled on this value. We also compared color vs grayscale, standard and normalised images, and saw that grayscale tended to outperform color. Unfortunately, we barely scratched 93% test set accuracy on 3x3 or 5x5 models, not consistently reaching this milestone. Moreover, we observed some erratic loss behaviour on the validation set after a given number of epochs, which actually meant our model was overfitting on the training set and not generalising. You can see below some of our metric graphs for different model configurations.



Models Performance on Color Normalised Images



Models Performance On Grayscale Normalised Images

# Dropout

In order to improve the model reliability, we turned to dropout, which is a form of regularisation where weights are kept with a probability p: the unkept weights are thus “dropped”. This prevents the model from overfitting. Dropout was introduced by Geoffrey Hinton, a pioneer in the deep learning space. His group’s [paper](http://jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf) on this topic is a must read to better understand the motivations behind the authors. There’s also a fascinating parallel with biology and evolution.  
In the paper, the authors apply varying degrees of dropout, depending on the type of layer. I therefore decided to adopt a similar approach, defining two levels of dropout, one for convolutional layers, the other for fully connected layers:

p-conv: probability of keeping weight in convolutional layer  
p-fc: probability of keeping weight in fully connected layer

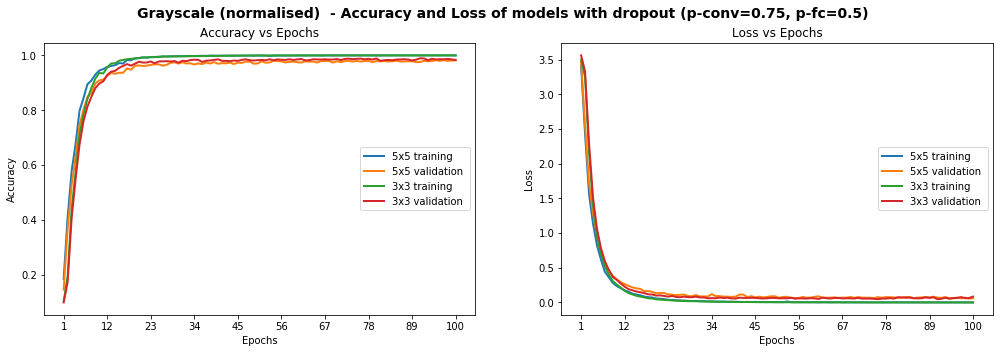
Moreover, the authors gradually adopted more aggressive (i.e. lower) values of dropout as they go deeper in the network. Therefore I also decided:

p-conv >= p-fc

that is, we will keep weights with a greater than or equal probability in the convolutional than fully connected layers. The way to reason about this is that we treat the network as a funnel and therefore want to gradually tighten it as we move deeper into the layers: we don’t want to discard too much information at the start as some of it would be extremely valuable. Besides, as we apply [MaxPooling](https://www.quora.com/What-is-max-pooling-in-convolutional-neural-networks) in the convolutional layers, we are already losing a bit of information.

We tried different paratemers but ultimately settled on p-conv=0.75 and p-fc=0.5, which enabled us to achieve a test set accuracy of 97.55% on normalised grayscale images with the 3x3 model. Interestingly, we achieved over 98.3% accuracy on the validation set:

Training EdLeNet\_Norm\_Grayscale\_3x3\_Dropout\_0.50 [epochs=100, batch\_size=512]...  
  
[1] total=5.222s | train: time=3.139s, loss=3.4993, acc=0.1047 | val: time=2.083s, loss=3.5613, acc=0.1007  
[10] total=5.190s | train: time=3.122s, loss=0.2589, acc=0.9360 | val: time=2.067s, loss=0.3260, acc=0.8973  
...  
[90] total=5.193s | train: time=3.120s, loss=0.0006, acc=0.9999 | val: time=2.074s, loss=0.0747, acc=0.9841  
[100] total=5.191s | train: time=3.123s, loss=0.0004, acc=1.0000 | val: time=2.068s, loss=0.0849, acc=0.9832  
Model ./models/EdLeNet\_Norm\_Grayscale\_3x3\_Dropout\_0.50.chkpt saved  
[EdLeNet\_Norm\_Grayscale\_3x3\_Dropout\_0.50 - Test Set] time=0.686s, loss=0.1119, acc=0.9755



Models Performance on Grayscale Normalised Images, After The Introduction Of Dropout

The graphs above show that the model is smooth, unlike some of the graphs higher up. We have already achieved the objective of scoring over 93% accuracy on the test set, but can we do better? Remember that some of the images were blurry and the distribution of images per class was very uneven. We explore below additional techniques we used to tackle each point.

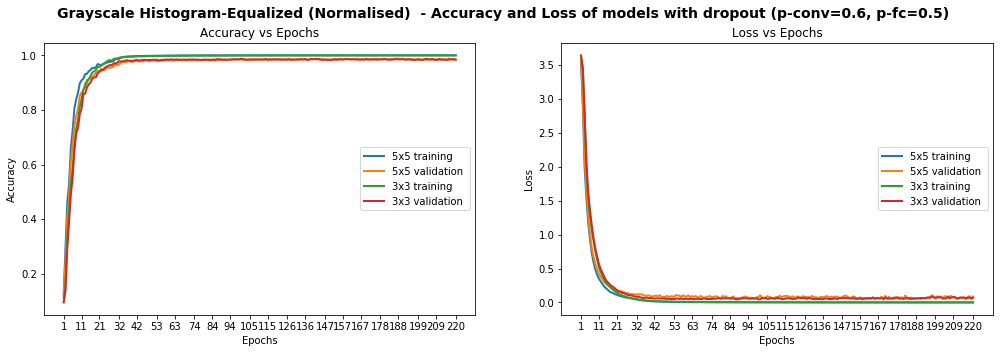
# Histogram Equalization

[Histogram Equalization](https://en.wikipedia.org/wiki/Histogram_equalization) is a computer vision technique used to increase the [contrast](https://en.wikipedia.org/wiki/Contrast_(vision)) in images. As some of our images suffer from low contrast (blurry, dark), we will improve visibility by applying OpenCV’s [Contrast Limiting Adaptive Histogram Equalization](http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_histograms/py_histogram_equalization/py_histogram_equalization.html) (aka CLAHE) function.

We once again try various configurations, and find the best results, with **test accuracy of 97.75%**, on the 3x3 model using the following dropout values: p-conv=0.6, p-fc=0.5 .

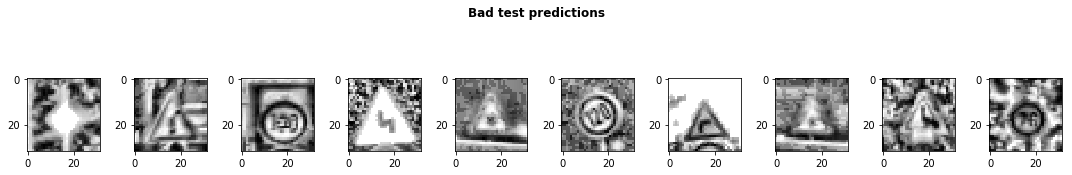
Training EdLeNet\_Grayscale\_CLAHE\_Norm\_Take-2\_3x3\_Dropout\_0.50 [epochs=500, batch\_size=512]...[1] total=5.194s | train: time=3.137s, loss=3.6254, acc=0.0662 | val: time=2.058s, loss=3.6405, acc=0.0655  
[10] total=5.155s | train: time=3.115s, loss=0.8645, acc=0.7121 | val: time=2.040s, loss=0.9159, acc=0.6819  
...  
[480] total=5.149s | train: time=3.106s, loss=0.0009, acc=0.9998 | val: time=2.042s, loss=0.0355, acc=0.9884  
[490] total=5.148s | train: time=3.106s, loss=0.0007, acc=0.9998 | val: time=2.042s, loss=0.0390, acc=0.9884  
[500] total=5.148s | train: time=3.104s, loss=0.0006, acc=0.9999 | val: time=2.044s, loss=0.0420, acc=0.9862  
Model ./models/EdLeNet\_Grayscale\_CLAHE\_Norm\_Take-2\_3x3\_Dropout\_0.50.chkpt saved  
[EdLeNet\_Grayscale\_CLAHE\_Norm\_Take-2\_3x3\_Dropout\_0.50 - Test Set] time=0.675s, loss=0.0890, acc=0.9775

We show below graphs of previous runs where we tested the 5x5 model as well, over 220 epochs. We can see a much smoother curve here, reinforcing our intuition that the model we have is more stable.



Models Performance On Grayscale Equalized Images, With Dropout

We identified 269 images that are model could not identify correctly. We display 10 of them below, chosen randomly, to conjecture why the model was wrong.



Sample of 10 images where our model got the predictions wrong

Some of the images are very blurry, despite our histogram equalization, while others seem distorted. We probably don’t have enough examples of such images in our test set for our model’s predictions to improve. Additionally, while 97.75% test accuracy is very good, we still one more ace up our sleeve: data augmentation.

# Data Augmentation

We observed earlier that the data presented glaring imbalance across the 43 classes. Yet it does not seem to be a crippling problem as we are able to reach very high accuracy despite the class imbalance. We also noticed that some images in the test set are distorted. We are therefore going to use data augmentation techniques in an attempt to:

1. Extend dataset and provide additional pictures in different lighting settings and orientations
2. Improve model’s ability to become more generic
3. Improve test and validation accuracy, especially on distorted images

We use a nifty library called [imgaug](https://github.com/aleju/imgaug) to create our augmentations. We mainly apply [affine transformations](https://en.wikipedia.org/wiki/Affine_transformation) to augment the images. Our code looks as follows:

def augment\_imgs(imgs, p):  
 """  
 Performs a set of augmentations with with a probability p  
 """  
 augs = iaa.SomeOf((2, 4),  
 [  
 iaa.Crop(px=(0, 4)), # crop images from each side by 0 to 4px (randomly chosen)  
 iaa.Affine(scale={"x": (0.8, 1.2), "y": (0.8, 1.2)}),  
 iaa.Affine(translate\_percent={"x": (-0.2, 0.2), "y": (-0.2, 0.2)}),  
 iaa.Affine(rotate=(-45, 45)), # rotate by -45 to +45 degrees)  
 iaa.Affine(shear=(-10, 10)) # shear by -10 to +10 degrees  
 ])   
 seq = iaa.Sequential([iaa.Sometimes(p, augs)])  
   
 return seq.augment\_images(imgs)

While the class imbalance probably causes some bias in the model, we have decided not to address it at this stage as it would cause our dataset to swell significantly and lengthen our training time (we don’t have a lot of time to spend on training at this stage). Instead, we decided to augment each class by 10%. Our new dataset looks as follows.



Sample Of Augmented Images

The distribution of images does not change significantly of course, but we do apply grayscale, histogram equalization and normalisation pre-processing steps to our images. We train for 2000 epochs with dropout (p-conv=0.6, p-fc=0.5) and achieve**97.86% accuracy on the test set:**

[EdLeNet] Building neural network [conv layers=3, conv filter size=3, conv start depth=32, fc layers=2]  
Training EdLeNet\_Augs\_Grayscale\_CLAHE\_Norm\_Take4\_Bis\_3x3\_Dropout\_0.50 [epochs=2000, batch\_size=512]...  
  
[1] total=5.824s | train: time=3.594s, loss=3.6283, acc=0.0797 | val: time=2.231s, loss=3.6463, acc=0.0687  
...  
[1970] total=5.627s | train: time=3.408s, loss=0.0525, acc=0.9870 | val: time=2.219s, loss=0.0315, acc=0.9914  
[1980] total=5.627s | train: time=3.409s, loss=0.0530, acc=0.9862 | val: time=2.218s, loss=0.0309, acc=0.9902  
[1990] total=5.628s | train: time=3.412s, loss=0.0521, acc=0.9869 | val: time=2.216s, loss=0.0302, acc=0.9900  
[2000] total=5.632s | train: time=3.415s, loss=0.0521, acc=0.9869 | val: time=2.217s, loss=0.0311, acc=0.9902  
Model ./models/EdLeNet\_Augs\_Grayscale\_CLAHE\_Norm\_Take4\_Bis\_3x3\_Dropout\_0.50.chkpt saved[EdLeNet\_Augs\_Grayscale\_CLAHE\_Norm\_Take4\_Bis\_3x3\_Dropout\_0.50 - Test Set] time=0.678s, loss=0.0842, acc=0.9786

## ****This is our best performance so far!!!****



Neural Network Celebration

**But**… if you look at the loss metric on the training set, you can see that at 0.0521, we most likely still have some wiggle room. We are planning to train for more epochs and will report on our new results in the future.

# Testing On New Images

We decided to test our model on new images as well, to make sure that it’s indeed generalised to more than the traffic signs in our original dataset. We therefore downloaded five new images and submitted them to our model for predictions.



Download 5 new traffic signs — color

The ground truth for the images is as follows:

['Speed limit (120km/h)',  
 'Priority road',  
 'No vehicles',  
 'Road work',  
 'Vehicles over 3.5 metric tons prohibited']

The Images were chosen because of the following:

* They represent different traffic signs that we currently classify
* They vary in shape and color
* They are under different lighting conditions (the 4th one has sunlight reflection)
* They are under different orientations (the 3rd one is slanted)
* They have different background
* The last image is actually a design, not a real picture, and we wanted to test the model against it
* Some of them are in under-represented classes

The first step we took was to apply the same CLAHE to those new images, resulting in the following:

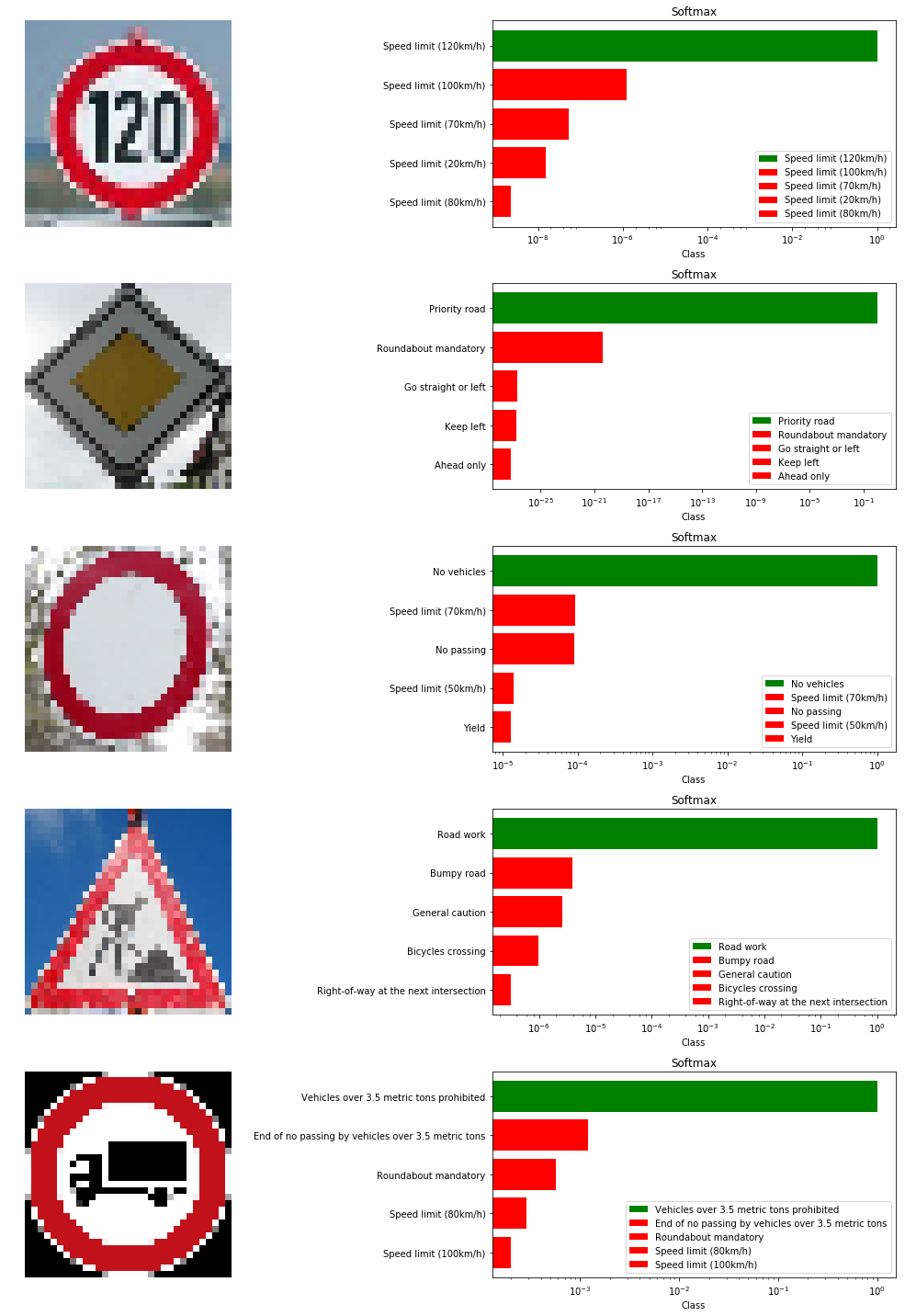


Download 5 new traffic signs — grayscale CLAHE

We achieve perfect accuracy of 100% on the new images. On the original test set, we achieved 97.86% accuracy. We could explore blurring/distorting our new images or modifying contrast to see how the model handles those changes in the future.

new\_img\_grayscale\_norm\_pred\_acc = np.sum(new\_img\_lbs == preds) / len(preds)  
print("[Grayscale Normalised] Predictional accuracy on new images: {0}%".format(new\_img\_grayscale\_norm\_pred\_acc \* 100))  
...  
[Grayscale Normalised] Predictional accuracy on new images: 100.0%

We also show the top 5 SoftMax probabilities computed for each image, with the green bar showing the ground truth. We can clearly see that our model is quite confident in its predictions. In the worst case (last image), the 2nd most likely prediction has a probability of around 0.1% . In fact our model struggles most on the last image, which I believe is actually a design and not even a real picture. Overall, we have developed a strong model!

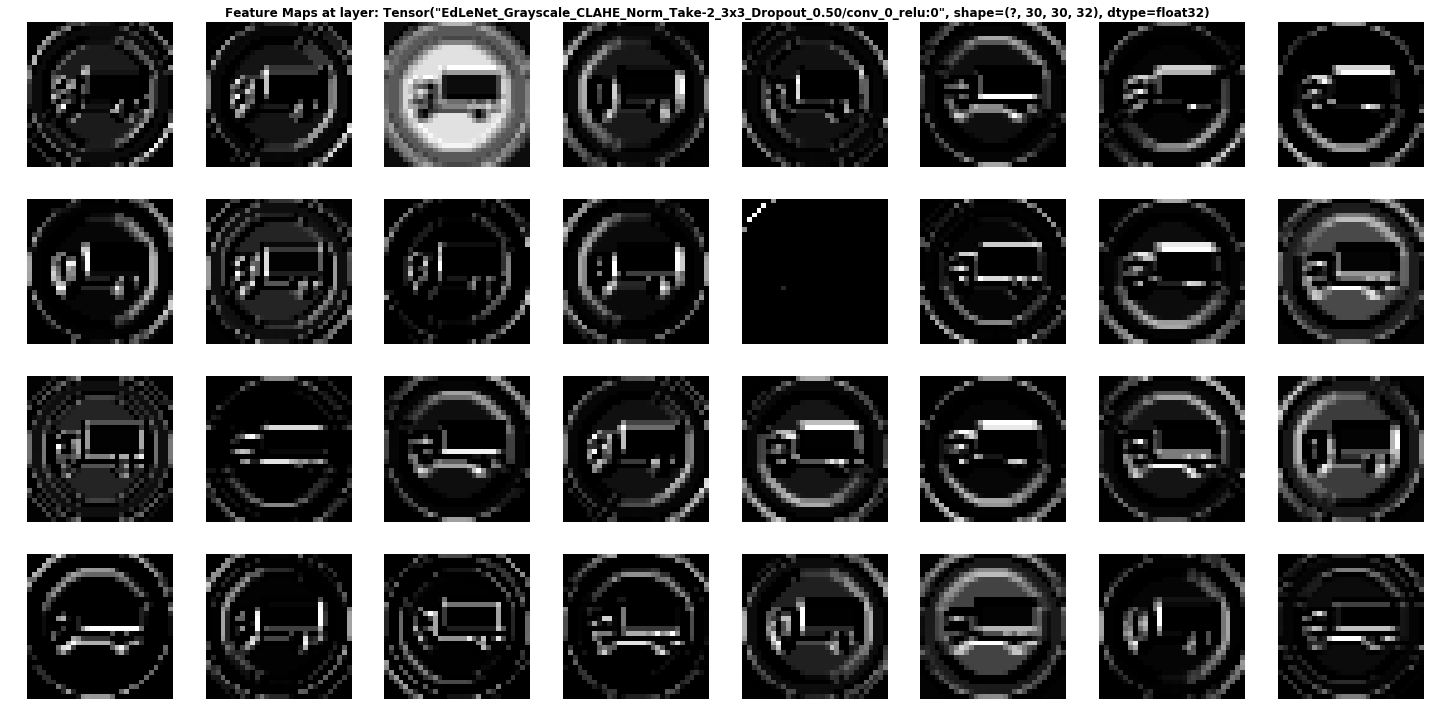


Visualizations of The Model’s Top 5 Predictions

# Visualizing Our Activation Maps

We show below the results produced by each convolutional layer (before max pooling), resulting in 3 [activation maps](https://www.quora.com/What-is-meant-by-feature-maps-in-convolutional-neural-networks).

## ****Layer 1****



We can see that the network is focusing a lot on the edges of the circle and somehow on the truck. The background is mostly ignored.

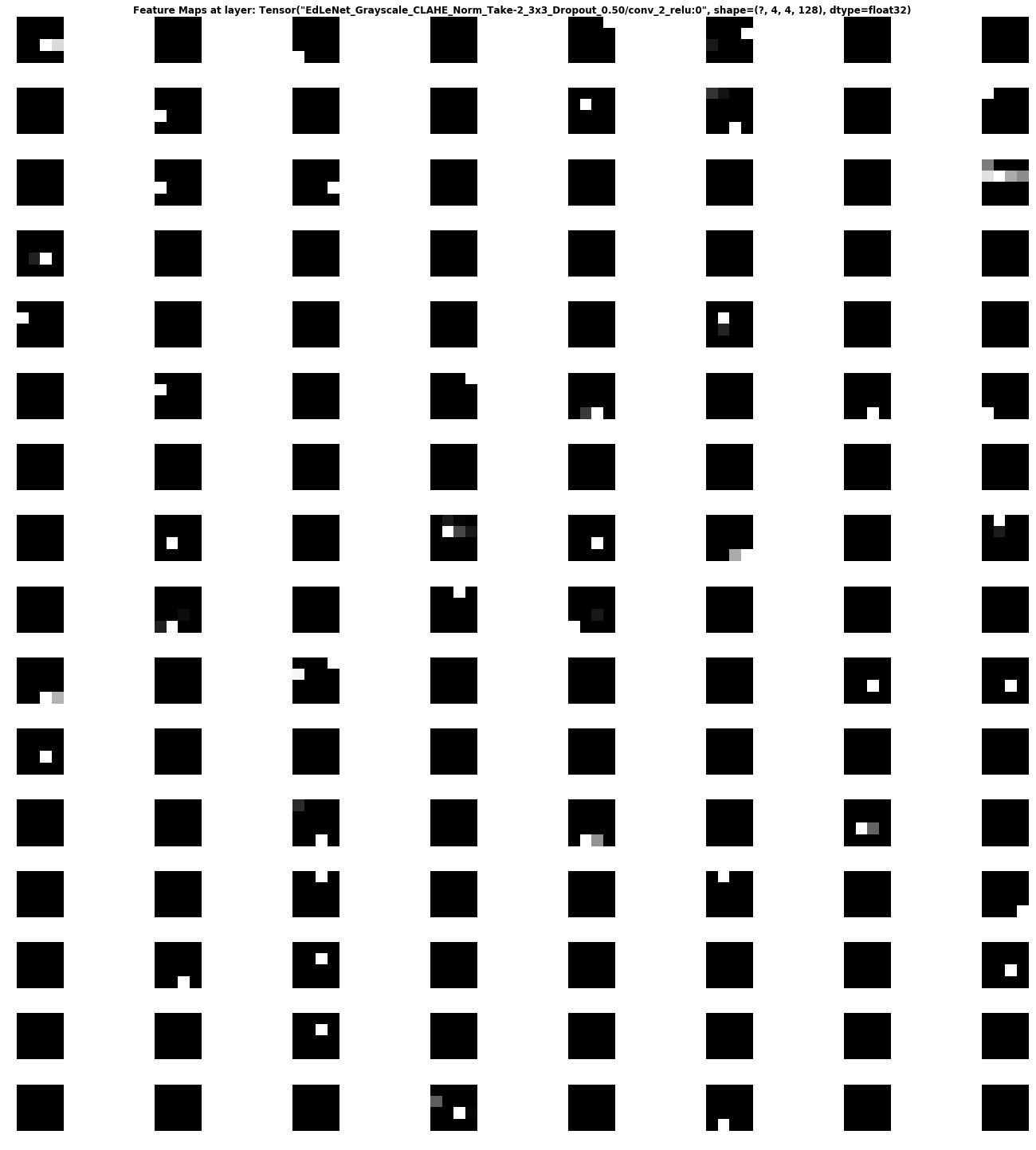
## Layer 2



Activation Map Of Second Convolutional Layer

It is rather hard to determine what the network is focusing on in layer 2, but it seems to “activate” around the edges of the circle and in the middle, where the truck appears.

**Layer 3**



This activation map is also hard to decipher… But it seems the network reacts to stimuli on the edges and in the middle once again.

# Conclusion

We covered how deep learning can be used to classify traffic signs with high accuracy, employing a variety of pre-processing and regularization techniques (e.g. dropout), and trying different model architectures. We built highly configurable code and developed a flexible way of evaluating multiple architectures. Our model reached close to close to 98% accuracy on the test set, achieving 99% on the validation set.

Personally, I thoroughly enjoyed this project and gained practical experience using Tensorflow, matplotlib and investigating artificial neural network architectures. Moreover, I delved into some seminal papers in this field, which reinforced my understanding and more importantly refined my intuition about deep learning.

In the future, I believe higher accuracy can be achieved by applying further regularization techniques such as [batch normalization](https://www.quora.com/In-layman%E2%80%99s-terms-what-is-batch-normalisation-what-does-it-do-and-why-does-it-work-so-well) and also by adopting more modern architectures such as [GoogLeNet’s Inception Module](https://arxiv.org/abs/1409.4842), [ResNet](https://arxiv.org/abs/1512.03385), or [Xception](https://arxiv.org/abs/1610.02357).

**Discussion:**

When we decide to do this project we really don’t know much more about how to develop apps in android studio and what is the principal behind it. Thanking to my this semester where we will learn Android App Development on regular basic and keep practicing and traying and thanks to our mentors who cleared every doubt and help our team to figure it out in making our working

Participation of members:

 Aditya- Coding and development of project, testing of project.

 Shivam- Basic research for the game and logic behind app and

animation ideas.

 Kanishka- All the hardware and software requirement and installation

was fulfilled and provide details through IEEE papers.

**Diﬃculties:**

When we start working on this project we face lot of problems like code is not running properly,

**Future Work:**

In future we would like to continue on this Projects next version which is more reliable andadding more features in it.

**BIBLOGRAPHY**

<https://towardsdatascience.com/recognizing-traffic-signs-with-over-98-accuracy-using-deep-learning-86737aedc2ab>

<https://medium.com/dataflair/class-data-science-project-for-2020-traffic-signs-recognition-12b09c131742>

<https://data-flair.training/blogs/python-project-traffic-signs-recognition/>