Experimentation and Comparison of Deep Learning based Hyperparameters for Traffic Sign Recognition

Adarsh Neema[1], Adarsh Bhandari[2]

[1,2]*Department of Computer Science and Engineering, Medi-Caps University, Indore*

---***---

Abstract- *With the rising number of drivers behind the wheel, the number of road accidents has thrived exponentially. One of the significant reasons behind these fatal misfortunes has been the high avertness of Traffic signals. With these, there is the utter need of a model that can guide the self-driving cars and even the cars with the drivers to help them perceive the signals and dwindle the road accidents. For this, it is required for a to perform real time analysis of traffic signs and alert the drivers. Following this, we have built a traffic sign image recognition system that can help driverless cars to interpret the traffic signals without any human intervention. Additionally, we have employed disparate optimizers while building up the model. Different activation functions are used with respect to optimizers to calculate the training loss and validation loss and hence draw tables and graphs to compare them in order to find the best fit for the model. Thus, this will clarify how a model behaves in multiple scenarios, with amendment in hyperparameters and time it takes to evaluate the result.*

Keywords- CNN, Traffic sign recognition, Image processing, Traffic signs.

Introduction

In recent times, the world saw behemoth traffic on the road, which became a key reason for the escalating number of road accidents. According to the research, the reason for such surgical incidents has been averting the traffic signals. Therefore, traffic sign detection and recognition has clinched importance and popularity in image processing which allows passengers to fully use cars for travelling, leading to gigantic transformation in the arena of electric self-driving cars.

An automated signal detection and recognition model will provide aptness for smart cars and driving to perceive the traffic signals and act accordingly. Not just for driverless cars, but also even with the driver behind the wheel, which may endorse with essential instructions, dwindling human fallacy that seeds accidents. With installation of such a system, it is anticipated the attrition of mishaps which will save a lot of human lives and capital associated with the car accidents.

The main goal behind developing such a system is clear because of the benefits the system provides in saving human lives and monetary as well. Hence, the objective of this project is to build a deep neural network model that can classify the traffic signs present in image into different categories. The suggested work has faculty to interpret the sign captured by the camera and further processing it by CNN network. Additionally, this paper carries major objectives of implementing distinct hyperparameters by using various optimizers and to conceive the best optimizer to improve the accuracy of the model above 95% such that the model functions well in a varied domain.

Literature Review

As the trend of machine learning is escalating at a rapid pace the number of hyperparameters and optimizers are also invented along the way to predict a deep classification among them and dig out the advancements made in them. With these advancements we are carrying out a comparative study of numerous activation functions under category image classification and how they perform under different circumstances. A lot of similar kind of work has been already been performed in this field, for instance: paper written by Zsolt T. Kharkov´acs, Zsombor Par´oczi and Endre Varga focused profoundly upon the problems, the system could face while analysis of real time traffic signals due to various important factors and proposed a novel system to capture image with high quality web camera with high resolution(1600x1200 frames) such that these images are clearly discernible in poor environment and hurdles and finally classifying the traffic signs. [1].

Another similar work in this field has been performed by Hui Guo and Zhongyu Wang, whose architecture was based on CNN and they introduced a novel, YOLO method for recognition which greatly improved accuracy of their model and provided faster results. [2]

Paper by Karthikeyan D, Enitha C, Bharathi S and Durkadevi K aimed at perceiving even the most perplexing images by learning in-depth features by rigorously learning from training samples by using YCBCR process. Their proposed work detects the image and compares it with images in their datasets to generate accurate prediction. Additionally, they have employed edge detection technique advanced GABOR filtering which is said to be as similar as human vision, which sections the traffic image from nearby images. [3]

In the paper by Danyah A. Alghmgham, Jaafar Alghazo and loy Alzubaidi, their CNN model produced an accuracy of 100%. They developed the model with 24 different traffic sign categories of Saudi Arabia with a total of 2718 images. Their proposed work detects the image and the captured image is processed by deep CNN. Moreover, they have defined the CNN model that they have implemented and published their results of accuracy and loss in tabular form differentiated by number of epochs and batch size. As a result, their final CNN model produced accuracy of 100% on 150 epochs. [4]

In the work done by Glory Reuben Maxwell and Dr. Dinesh D. Patil, they have described disparate methods for detection of traffic signals based on color, texture, and shape and explained them thoroughly. Furthermore, they have subdivided the color, texture and shape based method into subparts and defined them profoundly. [5]

After carefully scrutinizing the work performed by other authors, we tried to perform our project differently from the work discussed above. In our proposed system, we have employed different optimizers with different activation functions for parameter tuning, followed by comparison of their performance. The results are discussed at the end with tables and graphs.

Tools

In order to generate a highly efficient and smooth functioning system, we have worked with high level Machine Learning tools and technologies. These tools enabled us to import various foremost important libraries, allowed us to create graphs of accuracy and loss, helped us to find out the accuracy score etc. Tools used in this model are discussed below:

A. Jupyter Notebook: Jupyter notebook is a powerful tool that provides us the platform to write, share and visualize our code along with interactive development and presentation of our project. The coding of our machine learning model and implication of various hyperparameters took place in a jupyter notebook.

B. Python: Python is one of the most powerful programming languages majorly used in machine learning and data science projects. Due to the simplicity it provides while writing the code, it is the top preferred language for any programmer. The overall coding of our ML model is written in python. Additionally, numerous python libraries were imported to add functionality to the model.

C. Matplotlib: This is one of the most popular python libraries to visualize the data and know inner insights about it. It is frequently used with python mathematical library numpy and is free and open source. There are numerous types of graphs and complex plots that can be created with it like contour plot, polar plot, scatter plot, image plot, 3-D plots, histogram etc. Also in order to examine the accuracy and loss values in machine learning models to know the model behaviour in order to clarify them as overfitted or under fitted.

D. Scikit-Learn: This library is used to build the machine learning and deep learning models. It comprises supervised, unsupervised and reinforcement learning algorithms and efficient versions to develop the model. It is based on arithmetic python libraries like numpy, pandas, sympy, matplotlib, Scipy and ipython. The foundation of deep learning models is built on scikit learn which has feature selection, feature extraction and parameter tuning.

E. TensorFlow: It is an open source library for artificial intelligence for training and inference of deep neural networks. It is like a brain system which learns according to the data and performs under desired conditions. The novel ideas of image processing techniques specially used in diagnostic techniques for diseases like cancer, bone marrow, blood DNA analysis, genetic coding etc. are done with the help of this library. Also the latest advancements are done in this library to be used for solving AI related problems.

F. Tensor board: It is also a visualization tool with a deeper impact and insights about the deep learning model. It has a variety of methods to evaluate the epochs behaviour by depicting as histogram, time series analysis, graphs and scalars. It helps to compute the training and testing values under multiple hyperparameters to distinguish among each of them. It takes its values from the directory as specified during training of the machine learning model and can be used on various operating systems.

Optimizers

Over a period of time, machine learning and algorithms have exhibited certain advantages over human existence and transformed our lives with ease to make a certain decision about a problem. On top of machine learning, deep learning has offered higher flexibility and accuracy in disparate applications with increased precision and endorsement with its usage in almost every arena. With profound use of these algorithms, it becomes necessary to make use of these algorithms with minimal resources in order to reduce the monetary burden and ameliorate the performance by dwindling errors. This is where optimizer comes into play. Optimizers are the algorithms or the methods which transform our model into the most precise form by updating certain parameters such as weights, learning rate that minimizes the loss and improves the accuracy. In order to generate models with highest precision and minimum loss, we have employed different hyperparameters by making use of disparate optimizers. Below discussed are the optimizers that we have used in our project-

Table 1. Optimizers

Optimizers	Use
Adam	Adaptive moment system or Adam is a widely used optimizer designed specifically to train deep neural networks. Due to its competence with perplexed problems, minimal memory requirement, it is heavily preferred. While employing Adam optimizers it gave best working accuracy..
Adagrad	Adaptive gradient optimizer or Adagrad functions by thoroughly grasping the learning rate of the parameters by observing the frequency and behavior of the parameters (small updating for high frequency data and higher updating for low frequency data) and hence works best with scattered data.
RMSprop	Root mean square propagation or RMSprop functions by moving the average of the square of the gradients and dividing the gradient by the root of the average. It restricts oscillation in vertical directions and only supports in horizontal direction resulting in taking a large step which normalizes the gradient.
SGD	Stochastic Gradient Descent is an iterative method which works well on linear classifiers. It is one of the efficient ways to train the model with ease of implementation. The prime purpose of SGD is to find global optima by making amendments is learning rare alpha.
Adadelta	Adadelta or adaptive learning rate optimizers is a stochastic gradient descent method that is an extension of adagrad optimizer. Adadelta functions by restricting the fixed window size instead of summing all past gradients, leading to efficient updation of learning rate.

Activation Function

When input neurons are fed inside an artificial neural network, it is required to decide whether the particular neuron should be fired or not, which further helps to understand the complex pattern in the data, that decision parameters is known as activation function. Activation function maps the input value into a particular range, depending upon the type of function in use. In deep neural networks, weights and biases are applied to the data input and these input data are passed

through activation function. It is an activation function that decides whether this particular input data, called neuron, should be fired or not. In case the neuron is fired, the input data is fed to the next layer known, this process is known as forward propagation. In case, when the neuron is not fired (by checking the errors i.e. if the output generated is far from the actual value), it is fed to the previous layers and its weights and biases are updated, this process is known as back propagation. Depending upon the purpose, Activation function is broadly divided in two categories, they are discussed below:

A. Linear Activation Function:

As the name suggests, output in case of linear activation function will not be constricted within any range and therefore, it will produce continuous output. Because of continuous output, the range is (-infinity to +infinity). In practice, they are not much used in deep learning model due to following reasons-

a) Back propagation is not supported in linear activation function which further would not change the weights associated with the input because of the reason that the input value will always be same and the activation function will not limit the value in any range

b) Because of the fact that the last layer in a deep neural network is the linear function of the first layer (a linear combination of linear functions is still a linear function), so basically it converts the model into just a single layer.

B. Non Linear Activation Function:
Non-linear activation function as the name suggests is the function that confines the output within a particular range due to which output lies within a range. The range limit is dependent on the type of activation function in use. They help the model to learn all the involute patterns in the data and cover all the limitations of the linear activation function by supporting back propagation and by allowing multiple hidden layers to create the deep neural model. They learn advanced features in complex data such as audio, video, image, text and thus provide precise prediction. In our model, we have employed four non-linear activation functions. They are discussed below in the table-

Table 2. Activation Functions

Activation Function	Use
Sigmoid	Sigmoid function takes the input and converts it into range (0,1). Because of its range, it is mostly preferred in predicting the probability as an output. Sigmoid function is defined by the function- [6] $f(x) = 1/(1 + e - x)$ (1)
Relu	Relu activation function works quite differently from other functions. It checks the input value and produces output accordingly. if the input is positive, it will generate the output without changing the input value, but if the input is negative, it will map the output to zero. Its range is (0, infinity) therefore it is defined by the function- [7] $f(x) = max(0, x)$ (2)
Elu	Elu functions somewhat in similar fashion to Relu except for its behavior for the negative value. It is defined by the function- [8] $f(x) = x \; if \; x > 0$ (3) $\alpha(exp \; exp \; (x) \; - 1) if \; x \leq 0$ (4) it outputs the input if the value is positive but for negative value it produces a slightly smooth curve near the constant alpha. Elu has great advantages over other functions and most importantly it solves the problem of vanishing gradients (gradient approaches zero and becomes hard to train) and exploding gradients (gradient becomes too large, resulting in large update and unstable network) seen in sigmoid function.
Selu	Selu or self-exponential linear unit is a well-known action function for self-normalizing quality (meaning output of each layer will produce zero mean and unit variance). Moreover, Selu removed the vanishing gradient and exploding gradient problem. It is defined by the function- [9] $f(x) = \lambda x \; if \; x \geq 0$ (5) $f(x) = \lambda\alpha(exp(x) - 1) if \; x < 0$ (6) where the value of λ and α are predefined (λ=1.0507 and α=1.6733)

Proposed Work
This section consists of a series of steps undertaken, broadly, to build the deep CNN model. In recent years, a number of authors have worked in this field but no author has ever used different hyperparameters for comparison of the best optimizer for performance tuning in order to perceive the optimizer with greatest accuracy. Steps followed are as follows-

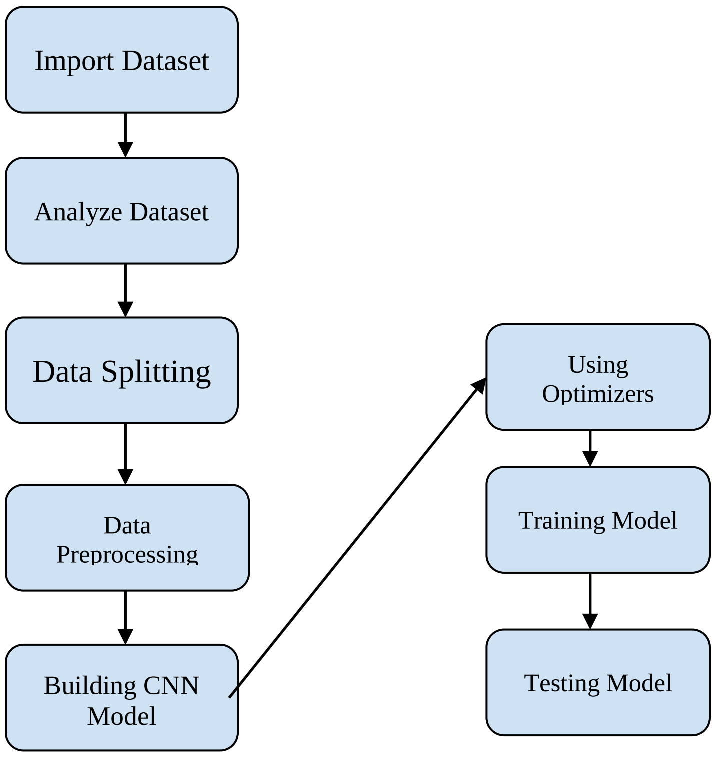


Fig 1. Steps Involved in building model

A. Import the dataset:
Primary step involved before developing the model is to load the dataset. In our model, we have imported a German traffic signals dataset from Kaggle in a jupyter notebook using python by defining the path of the data. Just by specifying the path, the data is imported from the specified location.

B. Analyzing the dataset:
After importing the dataset, another key process is to get familiar with the loaded data and explore it. In our proposed work, we have used a German traffic signals dataset with 41 different labels, ranging from 0 to 40. The shape of data is (39209,30,30,3), implying there are 39209 data with each image of size 30*30 pixels and 3 represents the colored image (RGB value). This part also involves feature engineering, meaning that the images are resized to appropriate size by specifying the dimensions of the image such that every image lies in the same scale.

C. Splitting the dataset:
Before moving further to develop the model, it's necessary to split the data into training and testing sets. In our model, we have used 80% of data as training data and remaining 20% of data as testing data. Splitting of data is necessary as we do not want our model to get familiar with every data present and want to save some percentage of data to test our model in the end in order to check accuracy.

D. Data preprocessing:
Data preprocessing is the process of transforming the raw datasets into interpretable form that could be well understood. Raw data is often imperfect having irregular matrix size, color correction. Before fitting the dataset into the model, it is required to convert the images into the same format with the same size and remove all the irregularities.

E. Building the CNN model:
For image classification into their respective categories, we build the CNN model, which is one of the best known methods for image classification. CNN is a deep learning algorithm that takes an image as an input, converts the input image in the form of pixels, assigns weights and biases to the input image and passes it to the next layer in the network. The network in CNN comprises different channels namely- CNN layer, pooling layer, Activation function layer and fully connected layer. Depending upon the model these layers can be repeated a different number of times. Architecture of our model is as follows-

F. Using Optimizers:
Optimizers are used in models to dwindle the loss factor by making amendments in weights, biases and learning rate and maximizing the efficiency of the system. After building the CNN model, we have used optimizers to calculate training accuracy, validation accuracy along with the training loss and validation loss. In order to work with the best optimizer, we have employed- ADAM optimizer, SGD optimizers, ADAGRAD optimizer, ADADELTA optimizer and RMSPROP optimizer. The result and performance of these models are discussed in the result and discussion section below.

G. Training the model:
After the model construction, we have trained our model using the function model.fit() with different batch sizes of 32 and 64. As a result our model performed efficiently with a batch size of 64. Approximately after 17 epochs, the accuracy was stable.

H. Testing the model:
In this final step, we finally allowed the machine to see the testing datasets and tested the working of the model by passing this to the machine learning model created to check the accuracy. In different scenarios, we got a

disparate percentage of accuracy. This is discussed in the result and discussion part, later in this paper.

Results and Discussion

In this section, we are going to talk about the result of our model. As conveyed above, we have employed various optimizers and activation in order to calculate the validation accuracy, training accuracy, validation loss. and training loss and their behavior is rigorously discussed in this part.

A. Selu as an Activation Function:

Optimizer	Validation Accuracy	Training Accuracy	Validation Loss	Training Loss
Adam	0.9779	0.9161	0.0756	0.3502
RMSprop	0.9802	0.9225	0.0824	0.4208
Adagrad	0.9415	0.8083	0.2019	0.6504

Table 3. Comparison between accuracies and losses keeping activation function as SELU

The above table illustrates the result obtained while using selu as an activation function and Adam, RMSprop and Adagrad as optimizers. Where Adam and RMSprop functioned as the best parameters, Adagrad gave relatively less accuracy. Although RMSprop has greater testing accuracy compared to Adam, the validation and training loss emerges Adam as the winner.

B. Sigmoid as an Activation Function:

Optimizer	Validation Accuracy	Training Accuracy	Validation Loss	Training Loss
Adam	0.9855	0.9718	0.0453	0.0941
RMSprop	0.9767	0.9596	0.1123	0.4306
Adagrad	0.575	0.593	0.3489	0.3525

Table 4. Comparison between accuracies and losses keeping activation function as Sigmoid

The above table illustrates the result obtained while using Sigmoid as an activation function and Adam, RMSprop and Adagrad as optimizers. The results show us that the Adam optimizers gave precise results as compared to its peers. Adagrad on the other hand gave aggravated results (accuracy less than 60%).

C. Relu as a Activation Function:

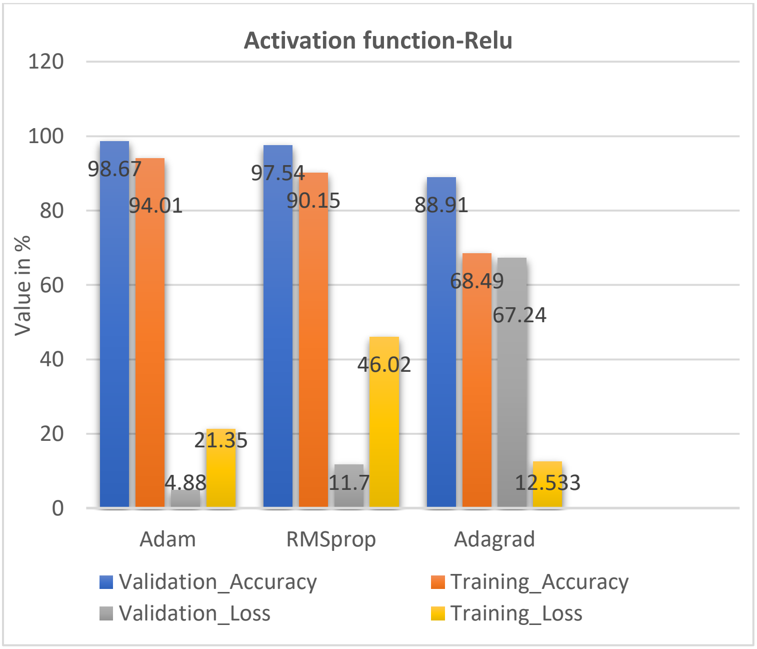


Fig 2. Comparison between accuracies and losses keeping activation function as Relu

In the figure above, we have used activation functions as Relu and optimizers as Adam, RMSprop and Adagrad. From the figure, it can be justified that Adam optimizers gave the highest validation and training accuracy followed by RMSprop.

D. Elu as an Activation Function:

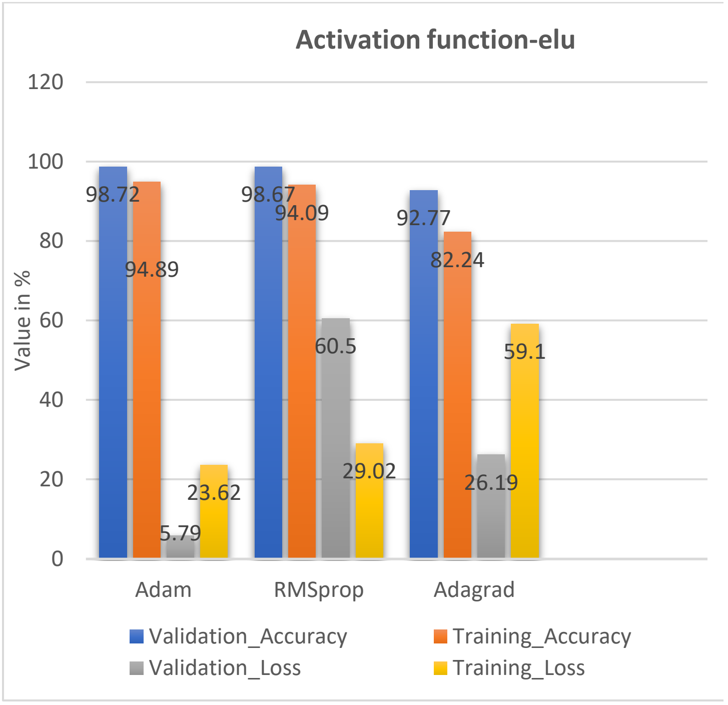


Fig 3. Comparison between accuracies and losses keeping activation function as Elu

Above figure depicts that Adam optimizers trained the model with highest precision and also performed well with training accuracy. Adagrad gave almost 60% of training loss resulting in poor performance.

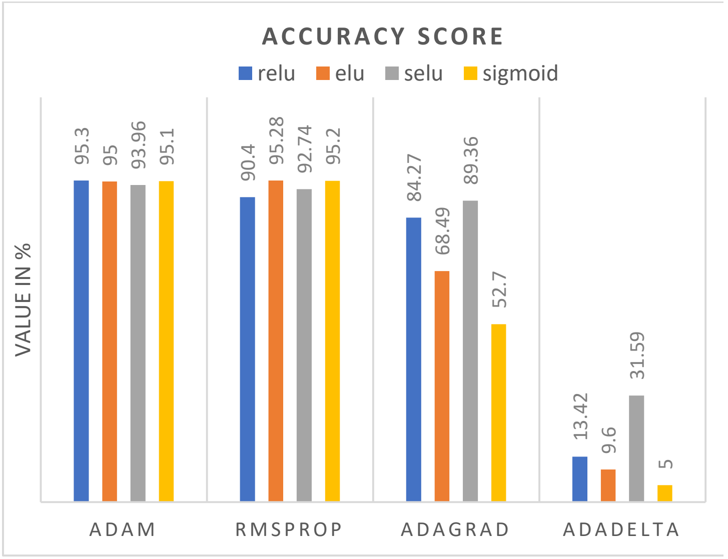


Fig 4. Comparison of the accuracy scores for each Activation Function with respect to each Optimizers.

In the figure above, we have calculated the final accuracy score of each activation function with respect to Adam, RMSprop, Adagrad, Adadelts optimizers.
It can be concluded, Adam optimizers performed well with all the activation functions, while the performance of Adadelts was the worst, with less than 50% accuracy with each activation function.

The reasoning behind the best performance of adam optimizers is due to the fact that it combines the benefits of both, RMSprop and Adagrad optimizer. Unlike RMSprop, which calculates the learning rate based on the average first moment (the mean) i.e despite of employing all the gradient for the momentum, it only makes use of most recent gradient, Adam optimizers on the other hand also takes into the account the average of second moment of the gradient as well, therefore it smoothly updates the learning rate, thus impacting the accuracy in positive manner. this could be understood with the mathematical formula-
Learning rate of Adagrad and RMSprop: [10]

$$\theta_{t+1}=\theta_t-(\eta/\sqrt{(E[g^2]_t+\epsilon)})*g_t \quad (7)$$

Learning rate of Adam:[11]

$$\hat{e}_t=e_t/(1-\beta_1^t) \quad (8)$$

$$\hat{u}_t=u_t/(1-\beta_2^t) \quad (9)$$

To update the parameter, we use:

$$\theta_{t+1}=\theta_t-(\eta/(\sqrt{\hat{e}_t}+\epsilon))\ \hat{e}_t \quad (10)$$

Taking into account the losses, Adam optimizers gave the lowest validation losses. Lower percentage of validation loss by adam optimizers can be now understood, due to smooth and efficient updation of the learning rate, training the model more perfectly and thus reducing the validation loss.

Using RMS prop as a optimizer using different activation functions:

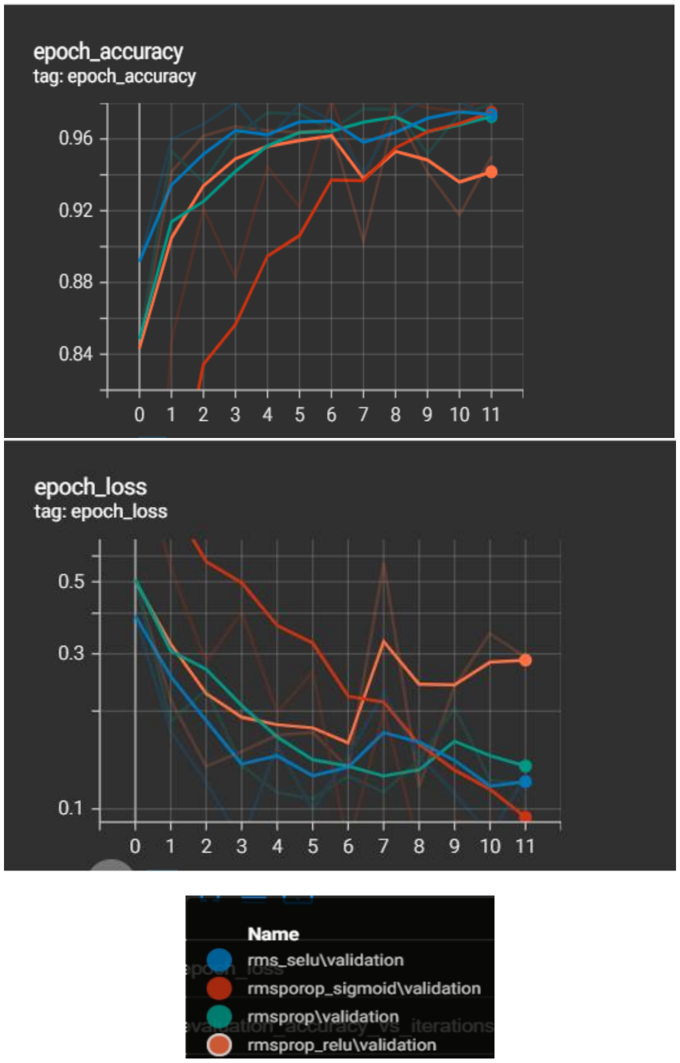


Fig 5. Accuracy and Loss graph of RMSprop under different activation functions.

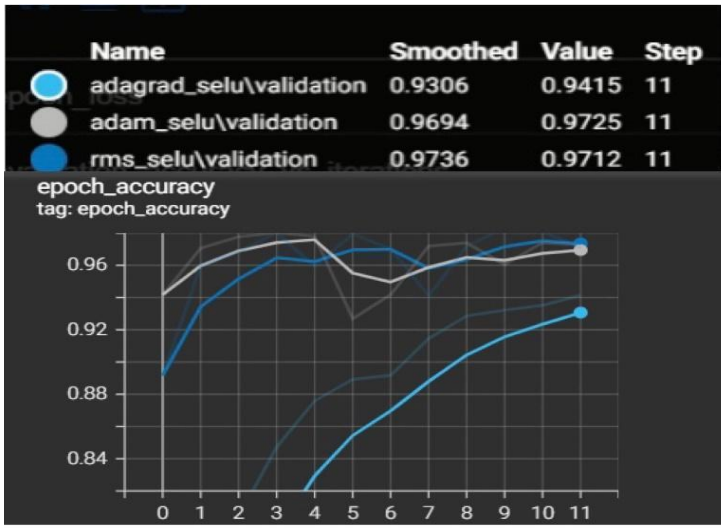
The diagram depicts the different activation function used with RMSprop optimizer under a similar number of epochs i.e.12. This study is done to carefully examine the validation accuracy and loss after training the model with desired number of polling layers and batch size. While relu, selu and elu showed a greater accuracy compared to sigmoid which works for the two-class logistic regression problems. In the two-class logistic regression, the predicted probabilities are as follows, using the sigmoid function: [12]

$$\Pr(Y_i=0) = e^{-\beta \cdot X_i}/(1+e^{-\beta \cdot X_i}) \qquad (11)$$

$$\Pr(Y_i=1) = 1-\Pr(Y_i=0) = 1/(1+e^{-\beta \cdot X_i}) \qquad (12)$$

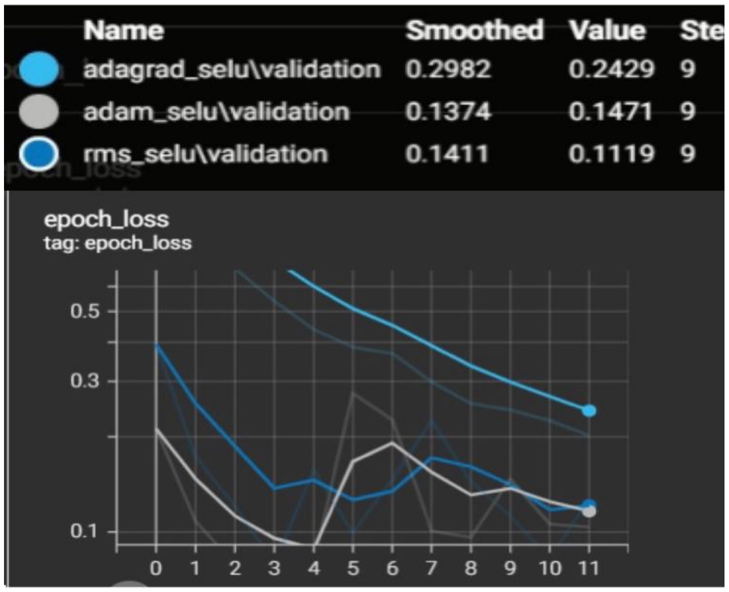
Following a steep curve in sigmoid we conclude that graphs for both the accuracy and loss are not following exponential curve and irregular pattern which disobeys the characteristics required for a proper training model curve. Choosing the right activation function involves which shows a greater accuracy and less loss where adam comes out as a clear winner. Additionally, it has displayed no overfitting and underfitting while carefully scrutinizing it under a greater number of epochs i.e 24. Selu and elu have also functioned well but they might do overfitting after a certain threshold and also they have discerned greater value loss as compared to adam. The two graphs significantly convey the difference while carrying out the research among activation functions under the same optimizer.

Study of selu as an activation function under different optimizers:



	Name	Smoothed	Value	Step
●	adagrad_selu\validation	0.9306	0.9415	11
●	adam_selu\validation	0.9694	0.9725	11
●	rms_selu\validation	0.9736	0.9712	11

Fig 6. Accuracy graph of SELU activation function using different optimizers.



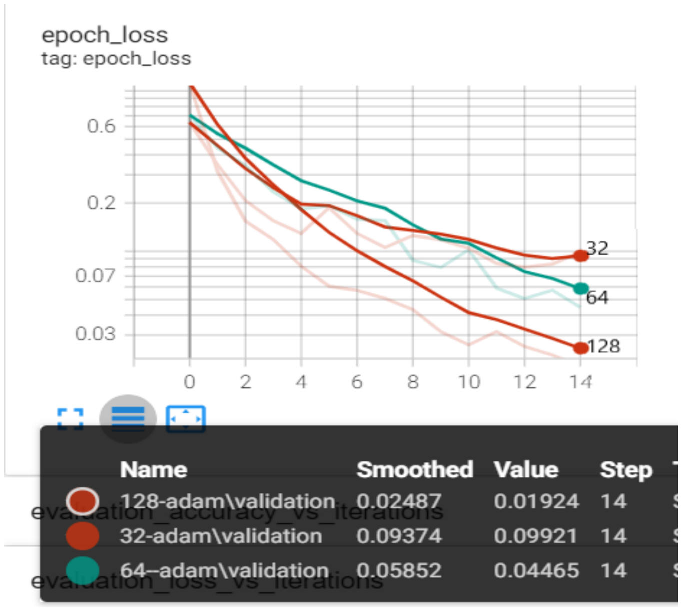
Name	Smoothed	Value	Ste
adagrad_selu\validation	0.2982	0.2429	9
adam_selu\validation	0.1374	0.1471	9
rms_selu\validation	0.1411	0.1119	9

Fig 7. Loss graph of SELU activation function using different optimizers.

The other point of difference and study lies here where we have the same activation function but different optimizers under similar epochs. This tuning of hyperparameters helps us to select among RMSprop and Adam because clearly adagrad has underperformed.

Adagrad's main downside is its accumulation of the squared gradients in the denominator: Since every added term is positive, the accumulated sum keeps growing during training. As mentioned previously in this research about the Adam and RMSprop functionality, where both projected well for image classification. Additionally, with selu there is greater accuracy (95%) but the difference lies at initial stages, where Adam outperforms RMSprop with higher training and testing values. Adam or Adaptive Moment Optimization algorithms combine the heuristics of both Momentum and RMSProp and so works better.

Changes with respect to different batch size:



Name	Smoothed	Value	Step
128-adam\validation	0.02487	0.01924	14
32-adam\validation	0.09374	0.09921	14
64-adam\validation	0.05852	0.04465	14

Fig 8. Validation accuracy and loss using Adam optimizer while training on different batch size

There is a significant impact while changing the batch size during compilation of the model. While the accuracy score remains significantly constant among the three scenarios not greater or less than 2% when compared (for 32 batch size: 96.30, for 64 batch size: 94.67, for 128 batch size:94.93), but the time to train and run the model varies according to the batch size. Although the number of epochs while running batch size 32 is 981 as compared to 246 epochs in batch size 128, the batch size of 128 took more time. Taking a too large batch size will indirectly lead to poor generalization and the desired model will have low bias and low variance. Thus, the inference can be drawn that always choosing the larger batch size is not the optimal solution as it either updates the gradient too large or small, since it all depends upon the sample drawn from the training data set.

Discrepancies while studying graph of various optimizers and activation function:

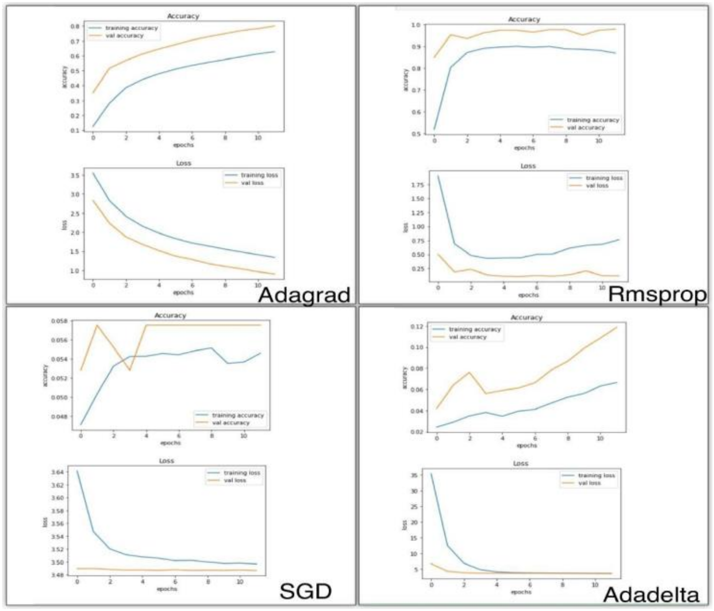


Fig 9. Comparison of Validation accuracy and loss of 4 different optimizers

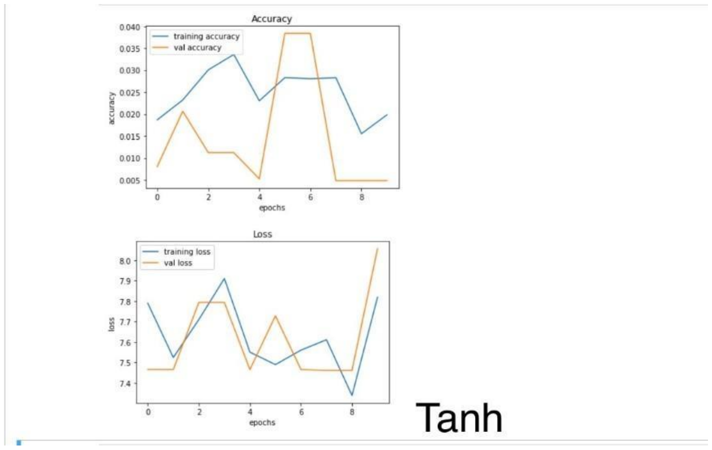


Fig 10. Discrepancy while using Tanh under similar epoch

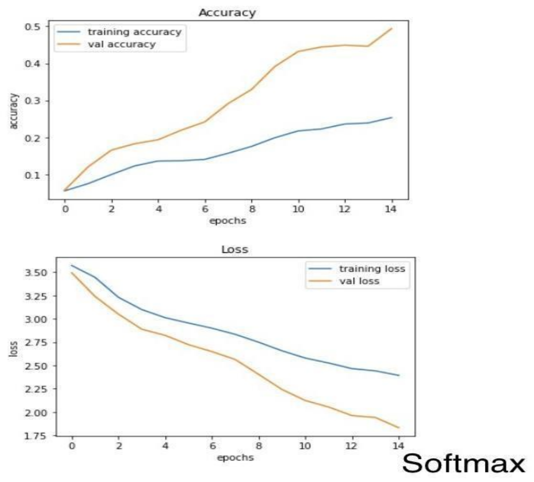


Fig 11. Discrepancy while using Softmax under similar epoch

The above two figures illustrate the disparate optimizers for this traffic sign recognition project to comprehend their behavior at the initial number of epochs and advocate us to understand why most of them cannot be used to train the Neural Network. One of the prime areas of our research is to find the action of the model while training on different optimizers of a similar dataset. After diligently studying the accuracy and loss graphs of every optimizer we found that:

1. The adagrad performs the underfitting and does not have an exponential curve with a certain threshold above which the accuracy and loss becomes stable.
2. The tanh has so much noise while running the epochs, therefore it cannot appropriately recognize the image and unable to do distinct accurately.
3. Both the sgd and adadelta have no improvement in accuracy values and show a constant value.
4. The RMSprop shows a better curve for both the loss and accuracy values while training for certain epochs. But due to advancement in adam, it clearly does the recognition more accurately and at a rapid rate.
5. The softmax function is used as a final activation function but not for all the layers in classification problems since it converts the final output into normalized form, but if used in all the layers, it will dwindle the values of vectors and hence, it will not train the neural network to certain characteristics. So using it at the last layer where the model has already learned from the features will generate a mean value which has a better impact on accuracy and classification.

CONCLUSION

In the research above, we saw how different optimizers are used with different activation functions to calculate the accuracy and losses of the dataset. After carefully inspecting the graphs and tables, it can be concluded that the Adam Optimizer is the best of all while Adadelta and Adagrad were relatively less useful. In addition, we carried out research in order to calculate the accuracy with respect to different batch size of 32,64 and 128, where batch size of 128 gave best results but it was prolonged in running as compared to batch size of 32 and 64. Additionally, although RMSprop too gave suitable results but since Adam being the advanced version of adagrad and rmsprop, it significantly performed proficiently and swiftly while training and testing of data. Furthermore, we carried on deeper analysis of numerous activation functions like relu, elu, selu, softmax, sigmoid to know the effect while training and testing on a traffic sign dataset. We also examined and scrutinized profoundly why certain activation and optimizers were a total fiasco because of their working and mathematical functions and why they behave differently seeing their testing and validation loss and accuracy. Finally, the in-depth research to understand the impact of hyperparameter tuning while making a deep learning model remarkably endorsed for subtle insights about the functioning of a neural network and how each layer i.e. polling relates with the activation function with the batch size.

REFERENCES

1. Kardkov´acs, Zsolt T. 2011. "Real-time Traffic Sign Recognition System." https://www.researchgate.net/publication/224255280_Real-time_traffic_sign_recognition_system/citations.
2. Hui Guo, and Zhongyu Wang Zhongyu Wang East China University of Science and Technology, Shanghai China View Profile. 2019. Research on Traffic Sign Detection Based on Convolutional Neural Network. https://dl.acm.org/doi/10.1145/3356422.3356457.
3. D, Karthikeyan, Enitha C, and Bharathi S. 2020. Traffic Sign Detection and Recognition using Image Processing. N.p.: IJERT. https://www.ijert.org/traffic-sign-detection-and-recognition-using-image-processing
4. Alghmgham, Danyah A., and Ghazanfar Latif. 2019. Autonomous Traffic Sign (ATSR) Detection and Recognition using deep CNN. https://pdf.sciencedirectassets.com/280203/1-s2.0-S1877050919X00198/1-s2.0-S1877050919321477/main.pdf?X-Amz-Security-Token=IQoJb3JpZ2luX2VjEH0aCXVzLWVhc3QtMSJGMEQCIF%2Bb%2FBRV%2FXzMhwDTwPH9m0lVk4Nyc6miH4A2fBYgsXTRAiBmZ6iRvuNt9WAlCbdHVgM%2FGVMH%2FvCV%2FoEGB.
5. Maxwell, Glory Reuben, and Dr. Dinesh D. Patil. 2020. A REVIEW ON TRAFFIC SIGN DETECTION AND RECOGNITION SYSTEM. https://www.irjet.net/archives/V7/i5/IRJET-V7I5314.pdf.
6. Towards Data Science and Sagar Sharma. n.d. Activation Functions in Neural Networks. Accessed 2017. https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6.
7. Great Learning and Hussain Mujtaba. 2020. What is Rectified Linear Unit (ReLU)? | Introduction to ReLU Activation Function. https://www.mygreatlearning.com/blog/relu-activation-function/.
8. HANSEN, CASPER. 2019. Activation Functions Explained - GELU, SELU, ELU, ReLU and more. https://mlfromscratch.com/activation-functions-explained/.
9. ML Glossary. n.d. Activation Functions. https://ml-cheatsheet.readthedocs.io/en/latest/activation_functions.html.
10. Towards Data Science and Vitaly Bushaev. 2018. Understanding RMSprop — faster neural network learning. https://towardsdatascience.com/understanding-rmsprop-faster-neural-network-learning-62e116fcf29a.
11. RUDER, SEBASTIAN. 2016. An overview of gradient descent optimization algorithms. https://ruder.io/optimizing-gradient-descent/.
12. Stack Exchange. Softmax vs Sigmoid function in Logistic classifier? https://stats.stackexchange.com/questions/233658/softmax-vs-sigmoid-function-in-logistic-classifier.