PERFORMANCE OF DIFFERENT OPTIMIZERS FOR TRAFFIC SIGN CLASSIFICATION

Ishita Joshi
Student Member, IEEE
Electronics and Communication Dept.
Sarvajanik College of Engineering and Technology
Surat, India
ishitahjoshi@gmail.com

Milauni Desai
Electronics and Communication Dept.
Sarvajanik College of Engineering and Technology
Surat, India
desai.milauni23@gmail.com

Sannidhi Bookseller
Electronics and Communication Dept.
Sarvajanik College of Engineering and Technology
Surat, India
sannidhibookseller@gmail.com

Rucha Mohod
Electronics and Communication Dept.
Sarvajanik College of Engineering and Technology
Surat, India
ruchavmohod01@gmail.com

Chirag N. Paunwala
SMIEEE
Electronics and Communication Dept.
Sarvajanik College of Engineering and Technology
Surat, India
chirag.paunwala@scet.ac.in

Bhaumik Vaidya
Electronics and Communication Dept.
Sarvajanik College of Engineering and Technology
Surat, India
bhaumik.vaidya@scet.ac.in

Abstract*— The use of traffic signs is vital especially when travelling in the highways or the hills in adverse environmental conditions. The pace of advancements in the field of Machine learning has opened doors for scopes of improvement in the performance of Convolutional Neural Network Architectures dedicated to classification of traffic signs. Speed is as important as accuracy for such problems in the field of Advance Driver Assistance Systems and the use of GPU instead of CPU gives the benefit of parallel processing. Gradient Descent helps navigating towards the minima of the loss function. Purpose of various gradient descent optimizing algorithms is to help in quicker convergence. This proposed algorithm comprises of a compact Convolutional Neural Network architecture that was trained on GPU using RMSProp, Adam and Nadam optimizers on the BelgiumTS dataset. RMSProp and Adam caused either under-fitting or over-fitting that was resolved by Nadam used with an appropriate dropout with 97.51 training accuracy and 96.78 testing accuracy. The predictions on test images convey that the architecture trained using Nadam works perfectly for blurry images, positionally challenging images and images with uneven illumination.

***Keywords*— convolutional neural networks, GPU, machine learning, optimizers, Traffic sign classification**

I. INTRODUCTION

Traffic signs serve the purpose to maintain traffic discipline. But the use of traffic signs becomes more crucial on the highways, especially during night; on hilly regions with sharp curves; during uneven visual conditions as driving at night or during foggy climate; to indicate construction areas or diversions on expressways. In these discussed cases, traffic signs do more than just maintaining discipline and correct interpretation of traffic sign is extremely important. Quick identification of traffic signs is vital as it will give more time for taking required action which was the motivation for implementing the proposed CNN Architecture on a multi-core GPU. The aim is to accomplish traffic sign classification and to train the proposed architecture using three optimizers. There are many challenges in the identification of traffic signs such as position, occlusion, obstacles, illumination and low image quality at high speed.

In the domain of traffic sign detection and classification, the research began with use of conventional algorithms derived using the concept of image processing. Lately, because of the easy availability of data, the research has inclined more towards the use of Convolutional Neural Networks derived using the concepts of Machine Learning that focuses greatly on building an architecture, training it using a huge collection of data and optimizing it so as to obtain desired results. Software implementation might have been done a lot many times but implementation on a hardware component that is highly computationally powerful is not undertaken as frequently.

Initial section includes brief overview of the optimizers used. The result section includes an outline of the datasets available and feature description of hardware used. Experimentation results comprise of accuracy plots and a summarized table that help to interpret and compare the performance of the three optimizers in a better way.

II. RELATED WORK

Researches on traffic sign detection and recognition are being continued from last decade has achieved ample amount of theoretical achievements; to put these achievements in actual practice seems quite difficult. Most accidents occur due to the avoidance (unintentionally or otherwise) or false interpretation of traffic signs in various environments.

A. Traffic Sign Detection and Recognition

If a part of an object is somehow covered (not properly visible) then successful detection and classification requires proper recognition task. Traffic sign recognition is an active research area in computer vision, color codes, shapes and pictograms used for traffic signs are different for countries. So, well defined color and shapes are main cues for detection of traffic signs. Also to attract the driver's attention easily, physical properties of traffic signs are so important. Due to these reasons the characteristics can be divided into 3 sub-classes:

1) Color based detection and recognition:

Color can be easily affected by lightning, as we discussed above different colors are used as background in different signs in different countries. Due to these differences, color spaces used nowadays are RGB (Red Green Blue), HSI (Hue Saturation and Intensity), HSV (Hue, Saturation, and Value), CIELAB, YUV [1]. The advantage of using color based approach is that it has fast computing speed. Whether HSL or RGB are better suited for traffic sign recognition has no hard proof. Both have their own pros and cons [2]. In HSV color of area proportional to brightness intuitive; color based on artist idea of tint, saturation and tone. Hue in HSV and CIELAB color space depends on distance and weather condition of traffic signs [3]. The most common approaches are using color and space based filtering to select the region of interest (ROIs).

2) Shape based detection and recognition:

Shape features plays key role for detection as shape is an important attribute for traffic sign detection. Reliability of shape detection depends on boundary detection or matching algorithm. As we discussed earlier shape based detection is important in a condition i.e. In Germany 'one-way street' represented by Long, Laying blue rectangular box whereas in US it is represented by white rectangular box. It is more robust to changes in illumination conditions as it depends on shape, which is based on edge or boundary of traffic signs. Imperfect shape of signs, occlusions of the object may cause the task somewhat challenging.

3) Both colour and shape based detection and recognition:

In order to improve the performance of detection process, a joint implementation of shape and color based algorithm is utilized.

B. Identification of Traffic Signs

One should have to set features properly in order to get good results. Identification of traffic signs consist of two steps; Pre-processing and classification. Well known classifiers named MLPs (Multi Layer Perceptions), SVMs [20] (Support Vector Machine), Radial basis functions [21], k nearest neighbors [22] are very sensitive to 2D input data. Normalized input data is required which makes it important to use feature extraction techniques. Feature extraction is a crucial part to distinguish multiclass probabilities as correctness of algorithm depends on feature vectors.

C. Motivation

We desire to reach the minima of the loss function in order to gain maximum accuracy. So, the intent is to identify which Gradient Descent optimizing algorithm out of RMSProp, Adam and Nadam converges to the local minima the earliest.

As far as Traffic sign detection and recognition is concerned, most of the earlier studies comprise of experimenting with different architectures. Whereas this paper is centered on a compact Convolutional Neural Network architecture that remains unchanged throughout the study and instead compares three optimizers used to train the architecture.

III. Overview of Optimizers

A convolutional neural network architecture consists of convolution layers designed for feature extraction, pooling layers that reduce the size of feature maps, flattening layers for converting 2-dimensional feature maps into 1-dimensional arrays followed by fully connected layers dedicated to *make sense* from the extracted feature maps, i.e. the decision making classifiers, activation functions used to maintain non-linearity and evaluating probabilities of the output classes. Dropout layers are used to randomly drop-off certain neurons in order to reduce the co-dependency among neurons in order to make the algorithm more robust by resolving the issue of over-fitting (a state where the algorithm works good for training set but fails to perform well on the test set). There exists no assured method to figure if a specific architecture will perfectly work for the given set of data until and unless practical experimentation is carried out. Hence we need to build our architectures intuitively which makes it very important to understand the backend mathematics.

Once we have designed our neural network architecture, the most important step is optimization. Sometimes a change as simple as switching to the correct optimizer can give improved accuracy instead of adding layers, increasing layer size and making the model bulky.

Understanding of Gradient Descent is essential to interpret the theory of optimizers. Gradient descent serves the purpose to minimize the loss function *J(θ)*. In other words, Gradient Descent helps us navigate towards the local minima. It gives the gradient and then takes a descent i.e. moves in opposite direction of the gradient. Hence it was named as Gradient Descent. The batch gradient descent [6] or vanilla gradient descent computes the gradient of the loss function,

$$\theta = \theta - \eta \nabla_{\theta} J(\theta) \qquad (1)$$

Choosing a learning rate is quite a task which motivates to optimize the Gradient Descent.

Vanilla Gradient considered present gradients to evaluate the next step towards local minima. It faces problems navigating ravines that occur frequently around the local minima. Momentum [7] considers a fraction of update vector of the past time step along with the current update vector [8]

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta) \quad (2)$$

$$\theta = \theta - v_t \quad (3)$$

Momentum increases dimensions of gradients in same directions and reduces updates in all other directions. This results into faster convergence.

RMSProp [9] i.e. Root Mean Square Propagation is a method for adaptive optimization. It divides the learning rate by exponential decaying average of squared gradients [6]

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t \quad (4)$$

Adaptive moment estimation [10] (Adam) is said to converge more quickly. By quick convergence, it means that it will require less number of epochs to reach the local minima of the los function. RMSProp considers exponentially decaying averages of past squared gradients and momentum considers exponentially squared averages of past gradients. Adam can be interpreted as a combination of RMSProp and Momentum. [6]

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \; ; \quad (5)$$

Exponentially decaying averages of past gradients

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \; ; \quad (6)$$

Exponentially decaying averages of past squared gradients

Bias correction $\widehat{m}_t = \frac{m_t}{1-\beta_1^t}$ and $\hat{v}_t = \frac{v_t}{1-\beta_2^t}$ and then,

update parameters using Adam's update rule:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}} \widehat{m}_t \quad (7)$$

with default values β_1 =0.9 and β_2=0.999 as proposed by authors.

Nesterov Accelerated Gradient [11] (NAG) calculates the gradient not w.r.t. the current position of parameters like momentum did, but w.r.t. approximate future position of the parameters:

$$v_t = \gamma v_{t-1} + \eta \, \nabla_\theta \, J(\theta - \gamma v_{t-1}) \quad (8)$$

Where $\theta = \theta - v_t$

Nesterov-accelerated Adaptive Moment Estimation (Nadam)[12] combines Adam and NAG by updating the momentum term, m_t. Recalling the momentum update rule: $\theta_{t+1} = \theta_t - (\gamma m_{t-1} + \eta g_t)$. It illustrates that momentum takes a step in direction of previous momentum vector and a step in the direction of the current gradient [6]

Modifying the gradient term as per NAG: [6]

$$g_t = \nabla_{\theta_t} J(\theta_t - \gamma m_{t-1}) \quad (9)$$

$$m_t = \gamma m_{t-1} + \eta g_t \quad (10)$$

$$\theta_{t+1} = \theta_t - m_t \quad (11)$$

As we have seen above, NAG is advanced than vanilla momentum so Nadam is expected to work out better than Adam, as per theory. Again, the results may seem otherwise, as the type of data plays a major factor. In this paper, we have compared the performance of RMSProp, Adam and Nadam on the BelgiumTS dataset for classification.

IV. Results

A. Benchmarks/ Datasets

Plenty of benchmarks are available for traffic sign detection and recognition algorithms like United states traffic sign dataset [25], Italy traffic sign dataset, Croatia traffic sign dataset, Chinese traffic sign dataset, BTSD[4] (Belgium traffic sign dataset), GTSRB[23] (German traffic sign recognition benchmark), STSD[24] (Swedish traffic sign dataset) etc. There are multiple reasons to choose one dataset over another, includes various facts depending on requirements. The reduced BelgiumTS Dataset including 62 classes comprised of warning, priority, prohibitory, mandatory, parking and direction signs has been used to implement the proposed CNN architecture. If the provided dataset is not sufficient to train the models, then to increase the volume of images some images can be created synthetically by changing contrast and by adding noise using image editors.

B. NVIDIA Jetson Tx1/Tx2

A GPU can be termed as a co-processor that performs requested operations by CPU [5]. We are fulfilling our purpose of traffic sign classification using Convolutional Neural Networks that mainly demands to perform operations on matrices. For the feature extracting convolutional layers, the task is to convolve feature maps inputted by previous layers with the weight matrices, and to do it repeatedly for each layer. CPUs are efficient for complex mathematical tasks whereas GPUs are proficient in carrying out repetitive tasks.

The NVIDIA Jetson Tx1 Development board (pre-flashed with a linux environment) that contains 256 CUDA (Computer Unified Device Architecture) Maxwell cores at 998MHz, a quad-core 64-bit ARM Cortex–A57 at 1.73GHz CPU. Jetson Tx2 is twice as energy efficient for deep learning inference than Tx1. 256 CUDA cores Pascal GPU at 1300MHz. Quad-core ARM Cortex-A57 at 2GHz and dual-core NVIDIA Denver2 at 2 GHz. Both the kits have 5 MP Fixed Focus MIPI CSI Camera that captures 30 fps, 4 power buttons(power, reset, force recovery and user defined), port connectivity (USB 3.0, USB 2.0 Micro AB, HDMI, M.2 Key E, PCI-E x4, Gigabit Ethernet), connectivity to 802.11ac Wi-Fi and Bluetooth enabled devices.[19]

C. Experimentation Results

This section experimentally compares the performance of RMSProp, Adam and Nadam optimizers.

```
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_10 (Conv2D)           (None, 62, 62, 64)        1792      
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 31, 31, 64)        0         
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 29, 29, 128)       73856     
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 14, 14, 128)       0         
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 12, 12, 128)       147584    
_________________________________________________________________
max_pooling2d_12 (MaxPooling (None, 6, 6, 128)         0         
_________________________________________________________________
flatten_4 (Flatten)          (None, 4608)              0         
_________________________________________________________________
dense_13 (Dense)             (None, 256)               1179904   
_________________________________________________________________
dropout_7 (Dropout)          (None, 256)               0         
_________________________________________________________________
dense_14 (Dense)             (None, 256)               65792     
_________________________________________________________________
dropout_8 (Dropout)          (None, 256)               0         
_________________________________________________________________
dense_15 (Dense)             (None, 128)               32896     
_________________________________________________________________
dense_16 (Dense)             (None, 62)                7998      
=================================================================
Total params: 1,509,822
Trainable params: 1,509,822
Non-trainable params: 0
_________________________________________________________________
```

Table 1 : Summary of proposed CNN architecture

Table 1 depicts the proposed CNN architecture that was built for the classification of traffic signs on the BelgiumTS classification dataset. The aim here was to make the architecture as compact as possible. The architecture contains three convolutional layers and their respective pooling layers, flattening layer, three fully connected layers followed by dropout layer after first and second fully connected layer and an output layer with dimensions equal to the number of output classes. The performance of RMSProp, Adam and Nadam using this architecture on the BelgiumTS classification dataset is portrayed by figure 1 to figure 6.

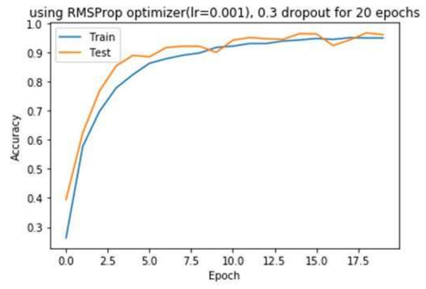


Figure 1 : Accuracy plot for RMSProp optimizer for dropout 0.3

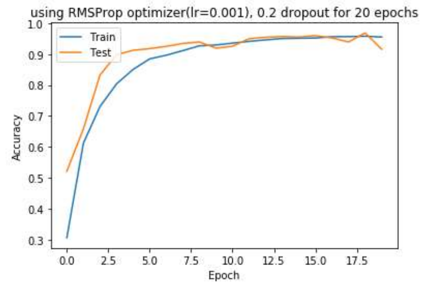


Figure 2 : Accuracy plot for RMSProp optimizer for dropout 0.2

Using RMSProp optimizer with dropout of 0.3 gave 94.91% training accuracy and 96.07 testing accuracy at 20 epochs. For majority of the training period, under-fitting can be observed from figure 2. Using the dropout of 0.2 gave comparatively better graphs. As per figure 2, under-fitting is observed initially, which was minimized after 10 epochs. With accuracies 95.98% and 91.63% for train and test respectively, over-fitting is observed towards the end.

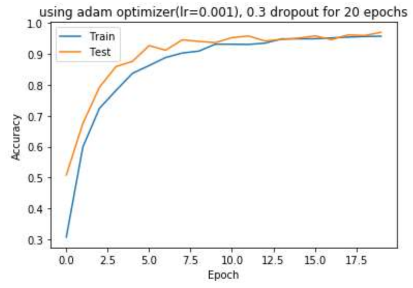


Figure 3 : Accuracy plot for Adam optimizer for dropout 0.3

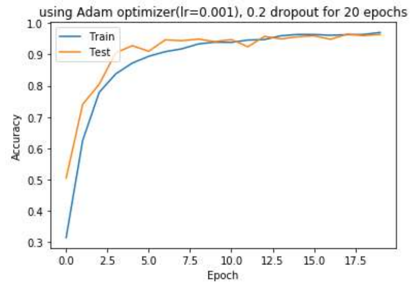


Figure 4 : Accuracy plot for Adam optimizer for dropout 0.2

Even after using Adam, under-fitting was observed for dropout of 0.3 accuracies 95.73% and 96.99% for train and test respectively were noted. Dropout of 0.2 gave better results compared to 0.3 as per the accuracy graphs. The training and testing accuracies were 97.05% and 96.43% respectively.

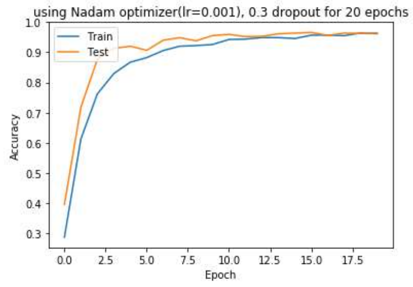


Figure 5 : Accuracy plot for Nadam optimizer for dropout 0.3

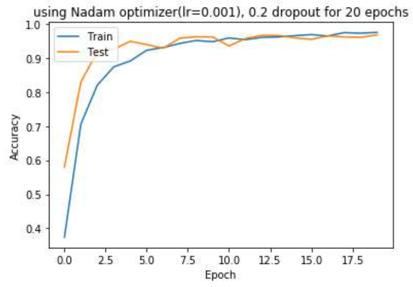


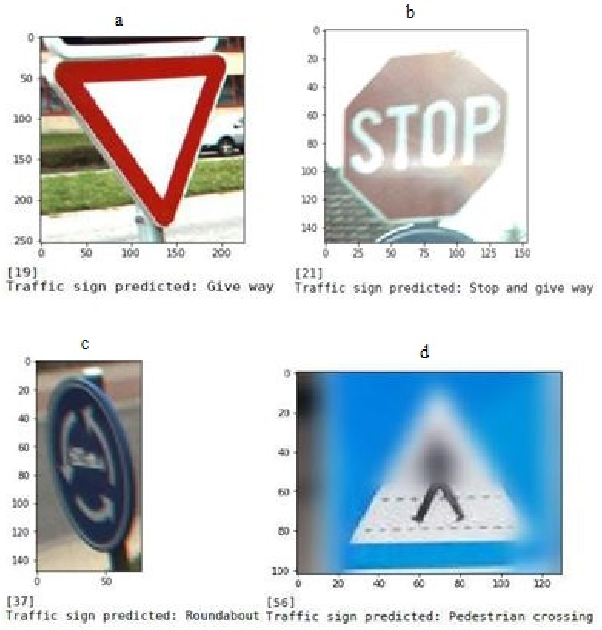
Figure 6 : Accuracy plot for Nadam optimizer for dropout 0.2

Using the Nadam optimizer for dropout of 0.3 boosted the training accuracy to 99.19%, but the testing accuracy was 97.62%. It can be seen that even though over-fitting is observed, Nadam improved the overall accuracy of the model. Using the dropout 0.2 with Nadam improved the training and testing accuracies to 97.51% and 96.78% respectively. The learning rate α was set to 0.001 for training all the models.

Clearly, the Nadam optimizer when used with the dropout of 0.2 was the best out of all the combinations stated above and predictions on test images were carried out using it.

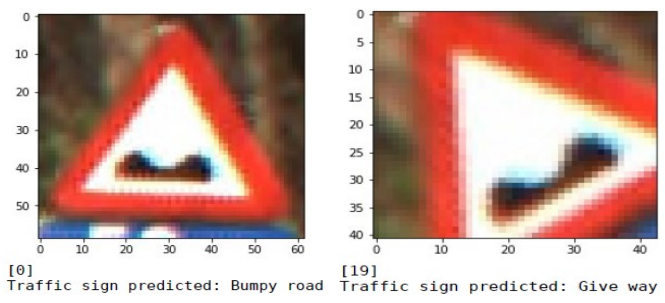
Figure 7 shows our predictions on test images as implemented on the NVIDIA Jetson Tx1/Tx2. Figure 7(a) is a prediction of a visually clear traffic sign. Figure 7(b) and 7(c) are unevenly illuminated and improperly positioned signs that were predicted correctly by our model. Parts of the image in Figure 7(d) were purposely blurred and that too, was predicted correctly.

Identifying incorrect predictions is more important than the correct ones as it hints about further improvements. Figure 8 (left image) is a correctly predicted sign named 'Bumpy road'. Figure 8 (right) was intentionally tilt using editing tools and the architecture gave an incorrect prediction.



```
[19]
Traffic sign predicted: Give way
[21]
Traffic sign predicted: Stop and give way
[37]
Traffic sign predicted: Roundabout
[56]
Traffic sign predicted: Pedestrian crossing
```

Figure 7



```
[0]
Traffic sign predicted: Bumpy road
[19]
Traffic sign predicted: Give way
```

Figure 8

V. Summary

DROPOUT	OPTIMIZER	EPOCHS	TRAIN	TEST
0.3	RMSProp	20	94.91	96.07
	Adam		95.31	95.75
	Nadam		96.19	96.43
0.2	RMSProp	20	95.98	91.63
	Adam		97.05	96.43
	Nadam		97.51	96.78

Table 2 : Accuracies for different optimizers on BelgiumTS dataset

As stated above, we have used dropout regularization of 0.3 and 0.2 and compiled the model using RMSProp, Adam and Nadam optimizers.

As to compare the optimizers, RMSProp is the *slowest* to converge. When used with momentum i.e. Adam, it converged *quicker*. When Adam used with nestrov momentum i.e. Nadam, it gave *quickest convergence* and hence best results. When dropout of 0.2 used with Nadam optimizer, it gave the best accuracy; Training accuracy = 97.51, Testing accuracy = 96.78

VI. Conclusion

As per theory, the Nestrov momentum makes Nadam intuitive and suggests better performance compared to other optimizers. We conclude with that Nadam works the best with Training accuracy = 97.51, Testing accuracy = 96.78 and converges quicker and better than RMSProp and Adam for classification on the BelgiumTS dataset.

Upon experimenting with many images, it can be concluded that the proposed architecture can predict blurred images well but it fails to predict tilted images. The scope of improvement is in expanding the training set and including misaligned images in order to make the algorithm more robust.

VII. Acknowledgements

The research was financially supported by the Student Start-up and Innovation Policy (SSIP), an initiative by the Government of Gujarat. After clearing the screenings conducted by GTU Innovation Council, we were eligible for the financial grant for Jetson Tx2, Display device (MarQ monitor) and other peripheral devices (keyboard, mouse, SSD drive, etc.)

VIII. References

[1] W.G. Shadeed, D.I. Abu-Al-Nadi, and M.J. Mismar, "Road traffic sign detection in color images", Proceedings of the 2003 10th IEEE International Conference on Electronics, Circuits and Systems (ICECS). IEEE, 2003

[2] Kardkovács, Zsolt & Paróczi, Zsombor & Varga, E & Siegler, Adam & Lucz, P. (2011), "Real-time traffic sign recognition system".

[3] Emre Ulay, "Color and Shape Based Traffic Sign Detection". A thesis submitted to the Graduate School of Natural and Applied Sciences, 2008

[4] M. Mathias, R. Timofte, R. Benenson, L.J.V. Gool, "Traffic sign recognition - how far are we from the solution?" in: International Joint Conference on Neural Networks (IJCNN), 2013.

[5] Nathan Otterness[1] , Ming Yang[1] , Sarah Rust[1] , Eunbyung Park[1] , James H. Anderson[1] , F. Donelson Smith[1] , Alex Berg[1] , and Shige Wang[2], "An Evaluation of the NVIDIA TX1 for Supporting Real-time Computer-Vision Workloads", [1]Department of Computer Science, University of North Carolina at Chapel Hill [2]General Motors Research.

[6] Sebastian Ruder, "An overview of gradient descent optimization algorithms" arXiv:1609.04747

[7] Ning Qian, "On the momentum term in gradient descent learning algorithms", Neural networks : the official journal of the International Neural Network Society, 12(1):145–151, 1999.

[8] John Duchi, Elad Hazan, and Yoram Singer, "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization", Journal of Machine Learning Research, 12:2121–2159, 2011.

[9] H. Brendan Mcmahan and Matthew Streeter. "Delay-Tolerant Algorithms for Asynchronous Distributed Online Learning", Advances in Neural Information Processing Systems (Proceedings of NIPS), pages 1–9, 2014.

[10] Diederik P. Kingma and Jimmy Lei Ba., "Adam: a Method for Stochastic Optimization". International Conference on Learning Representations, pages 1–13, 2015.

[11] Yurii Nesterov, "A method for unconstrained convex minimization problem with the rate of convergence o(1/k2)", Doklady ANSSSR (translated as Soviet.Math.Docl.), 269:543–547.

[12] Timothy Dozat, "Incorporating Nesterov Momentum into Adam", ICLR Workshop, (1):2013–2016, 2016.

[13] François Chollet, "Keras - https://keras.io/", Neural Networks platform written in python, stable release: 2.2.4 / 3 October 2018.

[14] John D. Hunter, "Matplotlib: A 2D Graphics Environment", Computing in Science & Engineering, **9**, 90-95 (2007), DOI:10.1109/MCSE.2007.55 (publisher link)

[15] Travis E, Oliphant, "A guide to NumPy", USA: Trelgol Publishing, (2006).

[16] Stéfan van der Walt, S. Chris Colbert and Gaël Varoquaux, "The NumPy Array: A Structure for Efficient Numerical Computation", Computing in Science & Engineering, 13, 22-30 (2011), DOI:10.1109/MCSE.2011.37

[17] Python Software Foundation, "Python Language Reference, version 3.6.8" Available at http://www.python.org

[18] A. Collette, "HDF5 for Python", 2008 (http://h5py.alfven.org)

[19] Dustin Franklin, "NVIDIA Jetson TX2 Delivers Twice The Intelligence To the Edge", March 7, 2017 NVIDIA Developer Blog.

[20] Muthukumaresan.T, Kirubakaran.B, Kumaresan.D, Akhil Satheesan, Jaya Prakash.A., "Recognition of Traffic Sign using Support Vector Machine and Fuzzy Cluster", IJSTE - International Journal of Science Technology & Engineering, Volume 2, Issue 10, April 2016.

[21] D.M. Gavrila, "Traffic Sign Recognition Revisited", Proc. of the 21st DAGM Symposium für Mustererkennung, pp. 86-93, Springer Verlag, 1999.

[22] Fatin Zaklouta, Bogdan Stanciulescu and Omar Hamdoun, "Traffic Sign Classification using K-d trees and Random Forests".

[23] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel, "The German Traffic Sign Recognition Benchmark: A multi-class classification competition", submitted to International Joint Conference on Neural Networks, 2011.

[24]Fredrik Larsson and Michael Felsberg, **"**Using Fourier Descriptors and Spatial Models for Traffic Sign Recognition"**,** In Proceedings of the 17th Scandinavian Conference on Image Analysis, SCIA 2011, LNCS 6688, pp. 238-249. doi:10.1007/978-3-642-21227-7_23

[25] A. Møgelmose, M. M. Trivedi, and T. B. Moeslund, "Vision based traffic sign detection and analysis for intelligent driver assistance systems: Perspectives and survey", IEEE Trans. Intell. Transp. Syst., vol. 13, no. 4, pp. 1484–1497, Dec. 2012.

[26] Felipe Sisido[1], Jonas Goya[2], Guilherme S. Bastos[3] and Audeliano W. Li[4] , "Traffic Signs Recognition System with Convolutional Neural Networks", 2018 Latin American Robotic Symposium, 2018 Brazilian Symposium on Robotics (SBR) and 2018 Workshop on Robotics in Education (WRE)

[27] Liu Wei, Lu Runge and Liu Xiaolei, "Traffic Sign Detection and Recognition via Transfer Learning", Chinese Control and decision Conference (CCDC) 9-11 June 2018, Shenyang, China, doi: 10.1109/CCDC.2018.8408160

[28] Yan Han and Erdal Oruklu, "Traffic Sign Recognition Based on the NVIDIA Jetson TX1 Embedded System using Convolutional Neural Networks", IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS), 6-9 Aug. 2017, Boston, MA, USA.

[29] Chen, Z., Huang, X., Ni, Z., & He, H. (2014) "A GPU-based real-time traffic sign detection and recognition system" 2014 IEEE Symposium on Computational Intelligence in

Vehicles and Transportation Systems *(CIVTS)*.doi:10.1109/civts.2014.7009470

[30] Schaul, T.; Zhang, S.; and LeCun, Y. 2013, "No More Pesky Learning Rates", International Conference on Machine Learning (ICML).

[31] Tieleman, T., sand Hinton, G. 2012. Lecture 6.5 "RMSprop: Divide the Gradient by The Running Average of its recent magnitude", In COURSERA: Neural Networks for Machine Learning.