

Optimizing neural radiance field: a comprehensive review of the impact of different optimizers on neural radiance fields

Latika Pinjarkar¹, Aditya Nittala¹, Mahantesh P. Mattada², Vedant Pinjarkar³, Bhumika Neole⁴,
Manisha Kogundi Math⁵

¹Symbiosis Institute of Technology Nagpur Campus, Symbiosis International (Deemed University), Pune, India

²Department of Electronics and Communication, PES Institute of Technology and Management, Shivamogga, India

³Department of Electronics and Communication Engineering, Shri Ramadeobaba College of Engineering and Management, Nagpur, India

⁴Department of Electronics and Communication Engineering, Ramdeobaba University, Nagpur, India

⁵Department of Information Science and Engineering, JNN College of Engineering, Shivamogga, India

Article Info

Article history:

Received Feb 15, 2024

Revised Aug 20, 2024

Accepted Sep 4, 2024

Keywords:

Adagrad

Adam

AdamW

Deep learning

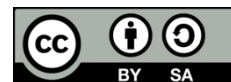
NeRF

RMSProp

ABSTRACT

Neural radiance field (NeRF) is a form of deep learning model that may be used to depict 3D scenes from a collection of photos. It has been demonstrated that NeRF can produce photorealistic photographs of fresh perspectives on a scene even from a small number of input images. However, the optimizer that is employed can have a significant impact on the quality of the final reconstruction. Finding an effective optimizer is one of the biggest challenges while learning NeRF models. The optimizer is responsible for making changes to the model's parameters to minimize the discrepancy between the model's predictions and the actual data. We cover the many optimizers that have been used to train NeRF models in this study. We present research results contrasting the effectiveness of multiple optimizers and examine the benefits and drawbacks of each optimizer. For training NeRF models, four different optimizers viz. Adaptive moment estimation (Adam), AdamW, root mean square propagation (RMSProp), and adaptive gradient (Adagrad) are trained. The most effective optimizer for a given assignment will vary depending on a variety of elements, including the size of the dataset, the complexity of the scene, and the level of accuracy that is required.

This is an open access article under the [CC BY-SA](https://creativecommons.org/licenses/by-sa/4.0/) license.



Corresponding Author:

Latika Pinjarkar

Symbiosis Institute of Technology Nagpur Campus, Symbiosis International (Deemed University)

Pune, Maharashtra, India

Email: latika.pinjarkar@sitnagpur.siu.edu.in

1. INTRODUCTION

A 3D scene representation technique called neural radiance fields (NeRF), learns to represent a scene as a continuous volumetric function. This function converts a 5D point, which has 3D spatial coordinates and a 2D viewing direction, into a radiance value, which denotes the amount of light that point in the scene emits or reflects [1]. NeRF models are developed using a series of input photos that serve as training data. Pairs of 5D points and radiance values that are produced by raytracing the scene from various angles make up the training data. To reduce the error between the produced images from the model and the original photos, the NeRF model is then trained.

It has been demonstrated that NeRF models are particularly good at producing photorealistic photographs of complicated settings. They have been employed to produce models of a wide range of objects, including both synthetic and real-world settings. One of the key features of NeRF is continuous

volumetric representation, which allows for smooth interpolation between different viewpoints. Other features are the ability to render photorealistic images of complex scenes and quick training even with large datasets.

To establish the minimum value of a function, an optimizer is employed. In the context of NeRF, the function whose minimization we seek is the loss function. The loss function calculates the variation between the original photos and the rendered images produced by the NeRF model. To decrease the loss function, the optimizer is used to adjust the NeRF model's parameters. An optimizer is used to determine a function's minimum value. The loss function is the function that we are attempting to minimize in the context of NeRF. The difference between the rendered images produced by the NeRF model and the original images is measured by the loss function. The NeRF model's parameters are updated using the optimizer to reduce the loss function. Optimizers can make training substantially faster, optimizers are crucial for NeRF. Since a good optimizer can locate the loss function's minimum more rapidly, the NeRF model can be trained using fewer input photos.

Earlier studies tried to compare the performance of different optimizers [2], [3]. The ultimate results of machine learning model training are greatly influenced by the optimizer selection. It's critical to understand that, even when used on the same dataset, different optimizers might produce different outcomes. Additionally, the dataset's properties have a significant influence on the model's performance. Specifically, larger datasets tend to provide clearer and more robust results, as they offer a broader spectrum of examples for the model to learn from and generalize.

Some of the recent trends in NeRF include generalizable NeRF, adapting to new scenes instantly by integrating multi-view stereo with differentiable volume rendering [4]. NeRF-visual odometry (NeRF-VO) [5] combines learning-based sparse visual odometry (VO) for fast camera tracking with a neural radiance scene representation, enabling advanced dense reconstruction and novel view synthesis. Recursive-NeRF [6] is an efficient and dynamically growing NeRF that enables a balance between efficiency and quality by reducing computational time. With shadow NeRF and Sat-NeRF, it's possible to consider the solar angle in a NeRF-based framework for rendering scenes from new viewpoints using satellite images for training. One of the recent works [7] builds on these contributions by demonstrating how to make the renderings season-specific. Another application of NeRF is audio-driven talking head animation. An entirely end-to-end talking head animation method that inherently captures 3D structures through learning a conditional NeRF [8].

In this work, we will delve into the critical consideration of selecting the most suitable optimization technique based on the specific dataset at hand. This decision is instrumental in harnessing the full potential of the NeRF model. Understanding the interplay between optimizer selection and dataset characteristics empowers practitioners to make informed choices, optimizing model training for superior performance. In essence, current work aims to guide how to leverage the NeRF model to its utmost capabilities. By tailoring the optimization approach to the dataset's size, complexity, and inherent characteristics, one can unlock the model's potential to generate accurate and meaningful insights or reconstructions, whether in computer vision, 3D scene rendering, or other applications where NeRF excels.

The rest of the paper is organized as follows. The method section provides a brief overview of the dataset used for these tests. Section 3 reviews the different optimizers that have been used for training this NeRF model. Section 4 showcases the results of individual optimizers and a comparison of their performance. Section 5 includes concluding remarks of the work.

2. METHOD

The training of machine learning models, especially neural networks, requires the use of an optimizer. Its main purpose is to repeatedly modify the model's weights and biases during the training phase to minimize a predetermined loss function. The parameters of the model are initialized before training begins. This initialization may entail giving the parameters random values or, in some situations, starting with values that have already been learned using a different model. A batch of training data is transmitted through the neural network in what is referred to as the feed-forward throughout each iteration or epoch of training. Predictions or outputs from this method are compared to the actual target values. A loss is calculated as a result of this comparison, which expresses the degree to which the predictions and the actual objectives differ [9].

The optimizer then performs a step known as backpropagation, sometimes known as the backward pass. Here, the gradients of the loss for each model parameter are calculated. This stage applies the chain rule to analyze how changes in each parameter affect the overall loss. It is based on the calculus concepts [10]. The next step for the optimizer is to use these gradients to modify the model's parameters. The objective is to consistently lower the loss by changing the parameters. Depending on the optimization technique being used, the specifics of this parameter updating mechanism can change.

The entire training dataset is covered by this iterative procedure, which goes on for several iterations. Training typically lasts until a predetermined ending criterion is reached. This could include doing so after a specific number of training iterations or when the loss converges to an acceptable level. The convergence of this process ultimately indicates that the parameters of the model have been adjusted iteratively to minimize the loss function. As a result, the model improves over time at making precise predictions using the training set of data. The optimizer selected as well as the hyperparameters linked to it can have a considerable impact on the training process and, as a result, the final performance of the machine learning model. Therefore, choosing the right optimizer and maybe adjusting its hyperparameters are crucial factors in successfully training a model for a particular task.

2.1. Overview of the dataset

The dataset can contain images and data that are real and fake. Synthetic data is sometimes created using computer graphics techniques like 3D modeling and rendering programs like blender or unity. Synthetic environments can be created by adjusting several variables, such as lighting, camera settings, and object placement. This makes it possible to manipulate data accurately and create many scenarios that adhere to accepted reality. A set of 2D photos of Lego toys together with the associated camera angles make up the NeRF dataset. The NeRF model needs to be trained with this dataset to learn a continuous representation of the underlying 3D scene and produce unique views from various angles.

Real-world pictures are taken using cameras and depict real-world scenes from various angles. Structure-from-motion (SfM) and simultaneous localization and mapping (SLAM) techniques, as well as intrinsic camera parameters, are used to extract the camera poses, which comprise location and orientation information. The dataset may include depth data that was gathered using stereo matching, depth sensors, or depth estimating methods [11].

To guarantee that the NeRF model's input criteria are met, the dataset is carefully selected and preprocessed. To ensure effective training, this may entail resizing photos, lining up camera angles, and structuring the data into the right forms. All things considered, the NeRF dataset provides the basis for training the model to understand 3D scene representation and allows the creation of realistic views from original points of view. Figure 1 shows a typical dataset showing details of the images captured from various angles and processed.



Figure 1. A typical dataset showing details of the images captured from various angles and processed

2.2. Peak signal-to-noise ratio calculation

The peak signal-to-noise ratio (PSNR) between two pictures is measured in decibels. It is computed using the PSNR block. The original and compressed images' quality is compared using this ratio. The quality of the compressed or rebuilt image improves with increasing PSNR.

The mean-square error (MSE) and PSNR are used to compare the quality of image compression. While the MSE shows the cumulative squared error between the original and compressed image, the PSNR measures the peak error. The MSE value has an inverse relationship with the error. To compute the PSNR, the block first calculates the mean-squared error using (1):

$$MSE = \sum_{M,N} [I_1(m, n) - I_2(m, n)]^2 \div (M * N) \quad (1)$$

The rows and columns of the input images are denoted by M and N in the preceding equation. Next, the block uses (2) to calculate the PSNR. In (2), R is the maximum fluctuation in the input image data type. R

is 1, for instance, if the data type of the input image is double-precision floating-point. R is 255, for example, if the data type is an 8-bit unsigned integer.

$$PSNR = 10 \log_{10}(R^2/MSE) \quad (2)$$

The following steps are carried out by the designed source code to calculate PSNR:

- It calculates camera-originating rays (lines) at a given point and orientation (as determined by testpose). In a 3D scene, these rays are utilized to replicate the camera's perspective.
- The rendering function in the code uses these rays to generate a picture (RGB), a depth map (depth), and accumulation information. The rendering function considers the number of samples per ray as well as the depth range (near and far).
- After rendering, it compares the RGB to a ground truth image to determine the loss value (loss). This loss measures how closely the rendered image resembles the original.
- Based on the calculated loss, it calculates the PSNR. Better image quality is indicated by higher PSNR values.
- The PSNR values and iteration counts are recorded in the code for subsequent analysis or logging. The iteration numbers aid in tracking progress during training or evaluation, while the PSNR values offer insights into how closely the produced images correspond to the actual scene.

In conclusion, this code segment generates images, computes a loss comparing them to the ground truth, computes PSNR for evaluating quality, and tracks these metrics over numerous iterations, probably to evaluate and enhance the efficiency of the rendering model.

3. OVERVIEW OF DIFFERENT OPTIMIZERS

The story of NeRF started with a simple light field rendering experiment performed by Levoy and Hanrahan [12]. They provided the bases required for the development of NeRF by Mildenhall *et al.* [13]. The work presented in [14] provided the mathematical infrastructure needed in the paper "a volumetric method for building complex models from range images". Since then gradient descent algorithms have evolved at a much faster rate. NeRF has attracted attention ever since it emerged as the quickest method for rendering a variety of images [15].

3.1. Adaptive moment estimation

Adaptive moment estimation (Adam) is a well-established optimization technique that is applied to the training of deep neural networks and other machine learning models. It is a well-known optimization technique that is an extension of stochastic gradient descent (SGD) [16]-[20] and is useful for optimizing complicated models. Adam integrates concepts from Momentum and root mean square propagation (RMSProp), two different optimization techniques [21]. Figure 2 depicts the results of Adam optimizer for 1,000 iterations along with PSNR.

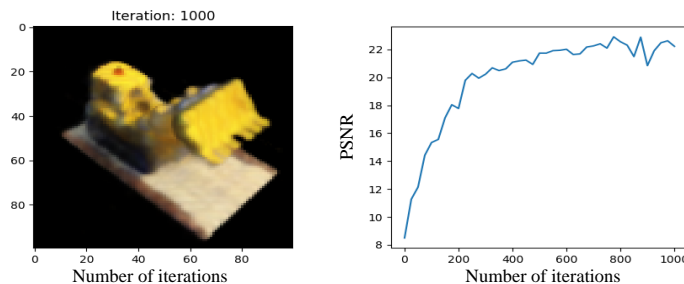


Figure 2. Results of Adam for 1,000 iterations along with PSNR

3.2. AdamW

Weight decay regularization is used by the Adam optimizer variation known as AdamW. Most people agree that AdamW is the best optimizer for training NeRF models. It has been demonstrated to generate high-quality NeRF models on a range of datasets, and it is reasonably quick and simple to use [22]. The results of AdamW performance for 1,000 iterations along with PSNR as shown in Figure 3.

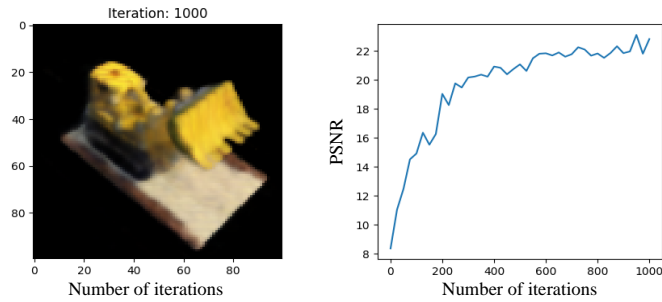


Figure 3. Results of AdamW for 1,000 iterations along with PSNR

3.3. Root mean square propagation

A moving average of the squared gradients is used by RMSProp, a stochastic gradient descent optimizer, to determine the learning rate. In general, AdamW is thought to be more effective than RMSProp when it comes to training NeRF models. On very big datasets, it can be more efficient than AdamW for training NeRF models [23]. Figure 4 depicts the results of RMSProp optimizer for 1,000 iterations along with PSNR.

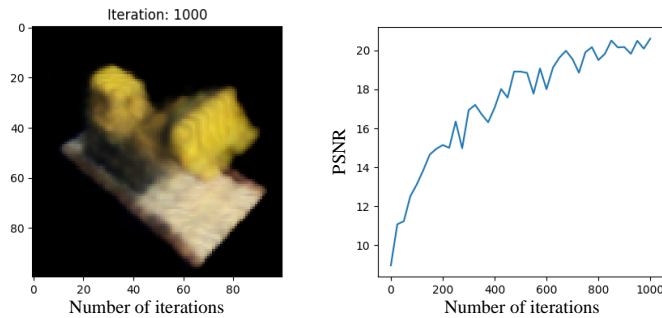


Figure 4. Results of RMSProp for 1,000 iterations along with PSNR

3.4. Adaptive gradient

Adaptive learning rate optimizer adaptive gradient (Adagrad) gives parameters with infrequent updates and higher learning rates. When it comes to training NeRF models on datasets with a lot of parameters, Adagrad can be superior to AdamW. Adagrad may, however, also be more susceptible to hyperparameter adjustments [24], [25]. The Adagrad technique known as Adagrad, dynamically modifies learning rates for individual model parameters based on historical gradient amplitudes. Particularly for models trained on sparse data or with features of different scales, this adaptability is advantageous. However, Adagrad has drawbacks, including the potential for learning rates to become exceedingly small over time, causing sluggish convergence. Hyperparameter tuning is crucial for optimizing the system's performance, but it adds up to the slow convergence. The results of Adagrad are shown in Figure 5.

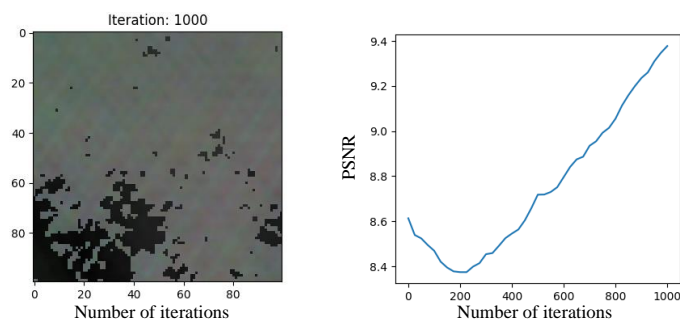


Figure 5. Results of Adagrad for 1,000 iterations along with PSNR

4. PERFORMANCE COMPARISON OF DIFFERENT OPTIMIZERS

As a part of this work, we evaluated PSNR results for all 4 optimizers. The steps involved in calculation are mentioned in the section 2.2. The subsequent paragraphs depict details of each optimizer's behavior and crucial results.

NeRF's major option and the standard training approach is the Adam optimization algorithm. After some time of training and deeper inspection, a clear and consistent upward trajectory in terms of model performance becomes apparent [26]. According to the PSNR metric, model performance during this training phase specifically shows a steady rising tendency. In quantitative terms, the PSNR value is recorded at an initial 7.883498, reflecting the model's performance at the onset of this time frame. However, it is noteworthy that this value demonstrates an inclination toward improvement throughout training. The training process culminates in the attainment of a peak PSNR value, reaching a commendable 23.12753 and ultimately ending at 22.83474. This trend of development underlines the Adam optimizer's efficiency in situations where training time is limited and the dataset being studied doesn't contain a sizable amount of data. With its momentum-based methodology and adjustable learning rate, Adam excels at rapidly improving rendering quality and fine-tuning model parameters throughout training comparatively brief training intervals.

The unique qualities of the particular challenge at hand and the precise architectural setup of the neural network model being used are what determine which optimization technique is used. It is essential to do empirical experimentation and evaluation to determine the best optimizer for a particular task. In this process, AdamW's relative performance against other optimization techniques is carefully analyzed and quantitatively measured. The initial PSNR measurement, which represents the model's performance at the start of this timeframe, registers at 7.7753005 in terms of numerical metrics. It is important to note that this value exhibits a tendency to improve over the course of training. The PSNR value increases noticeably during the training, peaking at 22.86007 before ending at 23.336285, which is a significant improvement. In the current scenario, considering the dataset utilized in conjunction with the applied model, it is observed that the AdamW optimization algorithm emerges as the superior choice among the various optimizer options. This choice is substantiated by its demonstrably superior performance in comparison to alternative optimization techniques.

All three adaptive optimization algorithms-RMSProp, Adam, and AdamW-adjust learning rates during training; however, their approaches to handling weight decay, accumulating moments, and implementing bias correction are different. Which of these optimizers is better for a given task may require empirical experimentation because the choice between them frequently depends on the particular problem, dataset, and model design. The model's performance at the beginning of this temporal interval is indicated by the initial measurement of PSNR, which is measured as 7.853294 in numerical terms. It is crucial to stress that this indicator shows a clear inclination toward improvement over the course of training. Throughout the training, the PSNR metric gradually and steadily increases, reaching a peak at 20.292233, an increase that, while indicating improvement, can be categorized as somewhat positive. In contrast to Adam and AdamW within the specific context under consideration, it becomes evident that this particular optimizer falls short in terms of surpassing their performance levels. However, it is noteworthy that this optimizer demonstrates a greater aptitude when applied to datasets characterized by substantial volume. Nevertheless, it successfully accomplishes the optimization task at hand.

In Adagrad, PSNR quantification at the beginning of this time period, which registered at 8.077606 in numerical terms, provides a clear indication of the model's performance over this time period. It is crucial to stress that, this statistic shows a steady and gentle inclination for improvement throughout the course of training. The PSNR metric shows a smooth and consistent increase during the training period, reaching its peak at 9.510035 with the least observed increment among all the other optimizers. Adagrad exhibits optimal compatibility when applied to extensive datasets, extended time frames, and ample computational resources. However, its performance in the present context is notably deficient, as it fails to generate a discernible image for us. Figure 6 depicts the PSNR values of all four optimizers. The summary of all four optimizers is mentioned in Table 1.

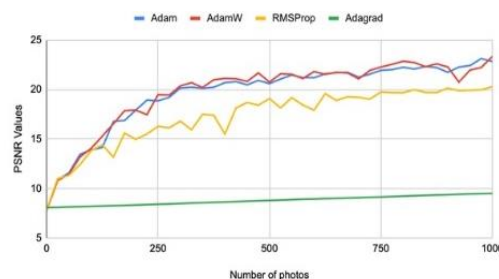


Figure 6. Comparison graph showing PSNR values of different optimizers

Table 1. Comparison matrix of all 4 optimization techniques

Optimization algorithm	Performance trajectory	Initial PSNR	Peak PSNR	Notable features	Suitability
Adam	Demonstrates a clear and consistent upward performance trajectory during training, efficiently improving model performance.	7.883498	23.12753	Momentum-based with adjustable learning rate. Effective for rapid quality enhancement in shorter training intervals.	Well-suited for limited training time and smaller datasets.
AdamW	Relative performance must be assessed through empirical experimentation. Significantly improves PSNR from 7.7753005 to 23.336285.	7.775300	23.33628	Effective optimization, particularly when applied to specific datasets and models.	Demonstrates superiority in certain contexts.
RMSProp	Displays a clear inclination for performance improvement, albeit with a modest increase from 7.853294 to 20.29223.	7.853294	20.29223	Adaptive learning rates.	May fall short compared to Adam and AdamW but performs better with substantial datasets.
Adagrad	Fails to produce a discernible image in the current context despite a gradual increase in PSNR from 8.077606 to 9.510035.	8.077606	9.510035	Adaptive learning rates based on gradient magnitudes.	Best suited for large datasets, extended training, and ample resources. May suffer from slow convergence.

5. CONCLUSION

This review study has explored four optimization algorithms in depth, specifically Adam, AdamW, RMSProp, and Adagrad, in the context of NeRF model training. We have identified distinct performance trajectories and characteristics for each optimizer through a thorough analysis. The choice of the best optimizer is dependent on many variables, including the size of the dataset, the resources that may be used for computing, and the particulars of the model at hand. The best optimizer for a specific task can only be found by empirical experimentation.

Ultimately, this research deepens our understanding of how optimization algorithms behave during NeRF model training and emphasizes the significance of customized selection depending on specific needs. For training NeRF models, we discover that the Adam optimizer is typically the most successful optimizer. However, in rare circumstances, alternative optimizers, like RMSProp, may also be useful. According to our research, selecting an optimizer is a crucial decision to make while training NeRF models. The most effective optimizer for a given assignment will vary depending on a variety of elements, including the amount of the dataset, the complexity of the scene, and the level of accuracy that is required. We hope that our results will help researchers and practitioners choose the best optimizer for their NeRF applications. Future studies may look towards hybrid strategies or cutting-edge optimization methods to improve the training of neural models like NeRF.

ACKNOWLEDGEMENTS

Authors would like to thank Symbiosis Institute of Technology, Nagpur for providing necessary facility and funding for this research work.





REFERENCES

- [1] K. Rematas, C. H. Nguyen, T. Ritschel, M. Fritz, and T. Tuytelaars, "Novel views of objects from a single image," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 8, pp. 1576–1590, Aug. 2017, doi: 10.1109/TPAMI.2016.2601093.
- [2] A. Mustapha, L. Mohamed, and K. Ali, "Comparative study of optimization techniques in deep learning: application in the ophthalmology field," *Journal of Physics: Conference Series*, vol. 1743, no. 1, 2021, doi: 10.1088/1742-6596/1743/1/012002.
- [3] F. Martínez, H. Montiel, and F. Martínez, "Comparative study of optimization algorithms on convolutional network for autonomous driving," *International Journal of Electrical and Computer Engineering*, vol. 12, no. 6, pp. 6363–6372, Dec. 2022, doi: 10.11591/IJECE.V12I6.PP6363-6372.
- [4] D. Lee and K. M. Lee, "Dense depth-guided generalizable NeRF," *IEEE Signal Processing Letters*, vol. 30, pp. 75–79, 2023, doi: 10.1109/LSP.2023.3240370.
- [5] J. Naumann, B. Xu, S. Leutenegger, and X. Zuo, "NeRF-VO: real-time sparse visual odometry with neural radiance fields," *IEEE Robotics and Automation Letters*, vol. 9, no. 8, pp. 7278–7285, Dec. 2023, doi: 10.1109/LRA.2024.3421192.
- [6] G. W. Yang, W. Y. Zhou, H. Y. Peng, D. Liang, T. J. Mu, and S. M. Hu, "Recursive-NeRF: an efficient and dynamically growing NeRF," *IEEE Transactions on Visualization and Computer Graphics*, vol. 29, no. 12, pp. 5124–5136, Dec. 2023, doi: 10.1109/TVCG.2022.3204608.
- [7] M. Gableman and A. Kak, "Incorporating season and solar specificity into renderings made by a NeRF architecture using satellite images," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 46, no. 6, pp. 4348–4365, Jun. 2024, doi: 10.1109/TPAMI.2024.3421192.





- 10.1109/TPAMI.2024.3355069.
- [8] S. Shen, W. Li, X. Huang, Z. Zhu, J. Zhou, and J. Lu, "SD-NeRF: towards lifelike talking head animation via spatially-adaptive dual-driven NeRFs," *IEEE Transactions on Multimedia*, vol. 26, pp. 3221–3234, 2024, doi: 10.1109/TMM.2023.3308441.
- [9] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics. JMLR Workshop and Conference Proceedings*, 2010, pp. 249–256.
- [10] N. Qian, "On the momentum term in gradient descent learning algorithms," *Neural Networks*, vol. 12, no. 1, pp. 145–151, Jan. 1999, doi: 10.1016/S0893-6080(98)00116-6.
- [11] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Springer Verlag, 2016, pp. 630–645, doi: 10.1007/978-3-319-46493-0_38/TABLES/5.
- [12] M. Levoy and P. Hanrahan, "Light field rendering" *Seminal Graphics Papers: Pushing the Boundaries*, vol. 2, pp. 441–452, Aug. 2023, doi: 10.1145/3596711.3596759.
- [13] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, "NeRF: representing scenes as neural radiance fields for view synthesis," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Springer Science and Business Media Deutschland GmbH, 2020, pp. 405–421, doi: 10.1007/978-3-030-58452-8_24.
- [14] B. Curless and M. Levoy, "A volumetric method for building complex models from range images," in *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1996*, Association for Computing Machinery, Inc, Aug. 1996, pp. 303–312, doi: 10.1145/237170.237269.
- [15] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv*, 2016, doi: 10.48550/arXiv.1609.04747.
- [16] S. Zhang, A. E. Choromanska, and Y. LeCun "Deep learning with elastic averaging SGD," in *Advances in Neural Information Processing Systems 28 (NIPS)*, 2015, pp. 685–693.
- [17] S. Mandt, M. D. Hoffman, and D. M. Blei, "Stochastic gradient descent as approximate Bayesian inference," *Journal of Machine Learning Research*, vol. 18, pp. 1–35, 2017.
- [18] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, pp. 2121–2159, 2011.
- [19] J. Schmidhuber, "Deep learning in neural networks: an overview," *Neural Networks*, vol. 61, pp. 85–117, Jan. 2015, doi: 10.1016/J.NEUNET.2014.09.003.
- [20] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Dec. 2016, pp. 770–778, doi: 10.1109/CVPR.2016.90.
- [21] D. P. Kingma and J. L. Ba, "ADAM: A method for stochastic optimization," *arXiv*, pp. 1-15, 2014, doi: 10.48550/arXiv.1412.6980.
- [22] H. Zhong *et al.*, "Adam revisited: a weighted past gradients perspective," *Frontiers of Computer Science*, vol. 14, no. 5, pp. 1–16, Oct. 2020, doi: 10.1007/S11704-019-8457-X/METRICS.
- [23] T. Kurbiel and S. Khaleghian, "Training of deep neural networks based on distance measures using RMSProp," *arXiv*, pp. 1–6, 2017, doi: 10.48550/arXiv.1708.01911.
- [24] A. T. Hadgu, A. Nigam, and E. Diaz-Aviles, "Large-scale learning with AdaGrad on Spark," in *Proceedings - 2015 IEEE International Conference on Big Data, IEEE Big Data 2015*, pp. 2828–2830, Dec. 2015, doi: 10.1109/BIGDATA.2015.7364091.
- [25] N. Zhang, D. Lei, and J. F. Zhao, "An improved adagrad gradient descent optimization algorithm," in *Proceedings 2018 Chinese Automation Congress, CAC 2018*, pp. 2359–2362, Jul. 2018, doi: 10.1109/CAC.2018.8623271.
- [26] A. Neelakantan *et al.*, "Adding gradient noise improves learning for very deep networks," *arXiv*, 2015, doi: 10.48550/arXiv.1511.06807.

BIOGRAPHIES OF AUTHORS






Latika Pinjarkar     obtained her Ph.D. in Computer Science and Engineering from CSVTU CG, India in 2019. She completed an M.Tech. degree in Computer Technology and Application from CSVTU, CG, India in 2008 and a B.E. degree in Computer Technology from Nagpur University, Maharashtra in 1999. She has been engaged in research and teaching for more than 22 years. At present, she is an Associate Professor and Academic Head in the Department of Computer Science and Engineering at Symbiosis Institute of Technology Nagpur, Symbiosis International (Deemed University) Pune, Maharashtra, India. She has presented more than 40 papers in International/National Journals/Conferences and has 05 Patents to her credit. She has completed one research project sponsored by TEQIP-III. Her research interests include image processing, computer vision, content-based image retrieval, and machine learning. She can be contacted at email: latika.pinjarkar@sitnagpur.siu.edu.in.






Aditya Nittala     is a final year B.Tech. student in Computer Science and Engineering at Symbiosis Institute of Technology, Nagpur. His area of research and projects lie in the intersection of deep learning, machine learning, and artificial intelligence. He can be contacted at email: aditya.nittala.batch2021@sitnagpur.siu.edu.in.






Mahantesh P. Mattada    holds a Ph.D. in time mode data converters from Visvesvaraya Technological University, Belagavi, India. He is currently working as an Associate Professor at the Department of ECE, PES Institute of Technology and Management, Shivamogga, India. His research interest includes time to digital converters, CMOS VLSI design, and digital circuit design on FPGA. He has 20 publications in reputed International Journals and Conferences. He is a reviewer for well-known IEEE conferences and Journals. He is a life member of ISTE and IAENG. He can be contacted at email: mpmathad@gmail.com or mahanthesh@pestrust.edu.in.






Vedant Pinjarkar    is currently a final year student at Shri Ramdeobaba College of Engineering and Management where he's completing his Bachelor of Technology degree in Electronics and Communication Engineering. His research interests include computer vision, machine learning, and embedded systems. He has worked on projects such as driver drowsiness detection using computer vision and crop defects detection using CNN. He aspires to continue advancing and working on the combined field of embedded systems and machine learning through impactful research and collaboration. He can be contacted at email: pinjarkarvs@rknc.edu.



Bhumika Neole    received her Bachelor of Engineering (Electronics Engineering) in 2001 from Priyadarshini College of Engineering and Architecture, RTM Nagpur University, India and M.Tech. (VLSI design) in 2007 from RCOEM, Nagpur, India. She has been awarded Ph.D. from VNIT, Nagpur, India in 2018. She has published 25 research papers in International Journals and Conferences, 1 Copyright and 1 Patent granted. Since 2004, she has been working as an Assistant Professor at Ramdeobaba University, Nagpur, India. Her current research interest is embedded and VLSI design, digital image processing, and signal processing. She is member of IEEE and a life member of ISTE. She can be contacted at email: neoleba@rknc.edu.



Manisha Kogundi Math    holds an M.Tech. degree from Visvesvaraya Technological University, Belagavi, India. She is currently working as an Assistant Professor at the Department of Information Science and Engineering, JNN College of Engineering, Shivamogga, India. Her research interest includes image processing and machine learning. She is a life member of ISTE and IAENG. She can be contacted at email: km.manisha294@gmail.com.