





Optimized Frequency Regularization: Decrease the Usage of Random Access Memory on Android Devices^{*}

Wenhao You¹, Leo Chang², Guanfang Dong³, and Anup Basu⁴

University of Alberta, Edmonton, Canada
{wyou1, chewei}@ualberta.ca

Abstract. We implemented a compressed algorithm frequency regularization (FR) on Android devices successfully and also packaged frequency regularization into a Python library, being deployed on Android devices. Moreover, we proposed **few optimized methods of frequency regularization algorithm to reduce the usage of random access memory (RAM)**. Furthermore, we developed a software on Android, implementing frequency regularization and its optimized versions. And we utilized Termux for deploying the frequency regularization Python library on mobile devices. Additionally, we adopted a layer-by-layer memory-freeing approach to optimize the neural network process in the optimized version. We also divided a complete image into four different parts for separated neural network processing. Our experiments in this paper illustrated that **the layer-by-layer memory-freeing method** works, reducing the usage of random access memory on Android devices significantly. It makes an increasingly possible to deploy certain useful and complex neural networks for image segmentation on limited hardware devices such as mobile phones.

Keywords: Frequency regularization · Neural network optimization · Python library · Random access memory · Android devices.

1 Introduction

Currently, people cannot live without mobile devices. They are not only for communication and entertainment, the information transformation is also a significant feature. Their portability and versatility make them play an important role in our daily lives. Meanwhile, convolutional neural networks (CNNs) are also vital in some research areas such as computer vision and other interdisciplinary data analysis. However, these neural networks are usually implemented on high-end hardware because of their huge amount of parameters and layers. It means that only a few researchers work on the deployment of neural networks on mobile. There exists several advantages of running convolutional neural networks on mobile devices: privacy, internet, and runtime. First, personal information

^{*} Supported by University of Alberta.

does not need to be uploaded or transmitted to the cloud servers which improves the privacy enhancement. Second, the functionality on local devices can replace some internet services. Therefore, it is independent from the internet connection. Last, in some applications that need real-time feedback, without connecting to the cloud server can shorten the processing time which decrease the runtime. In all, convolutional neural networks can totally replace the usage of many applications on mobile devices, ensuring personal data security.

According to the popularity of mobile devices and the benefits of convolutional neural networks, we want to find a way to deploy some large and complex convolutional neural networks on mobile devices, leading to the question: “How can we deploy large convolutional neural networks on mobile devices?”

We found five methods to achieve our goal: upgrade the hardware to a high-specification mobile device; implement Extreme Learning Machine (ELM) [1] to allocate the weight of hidden layers randomly in order to train large models on mobile devices faster; implement NestDNN [11] dynamically adjusts the size and computational complexity of the network based on available resources on mobile devices; implement “One-shot Whole Network Compression” [18] to prune, quantize, and compress the neural networks; implement frequency regularization (FR) [32] algorithm to reduce parameters by removing high-frequency component. We make a more detailed introduction to their drawbacks and limitations in Section 2.

After conducting a thorough literature review, considering all the limitations, accuracy, complexity, and future potential, we choose frequency regularization (FR) as our target algorithm. We deployed it on the Android-based device which is one of the most popular operating system on mobiles. Our main idea is to implement FR to compress a U-Net convolutional neural network and then decompresses the uploaded compressed model on an Android mobile device in a short time. After that, take the decompressed model to do image segmentation for the Carvana Image Masking Challenge Dataset [4].

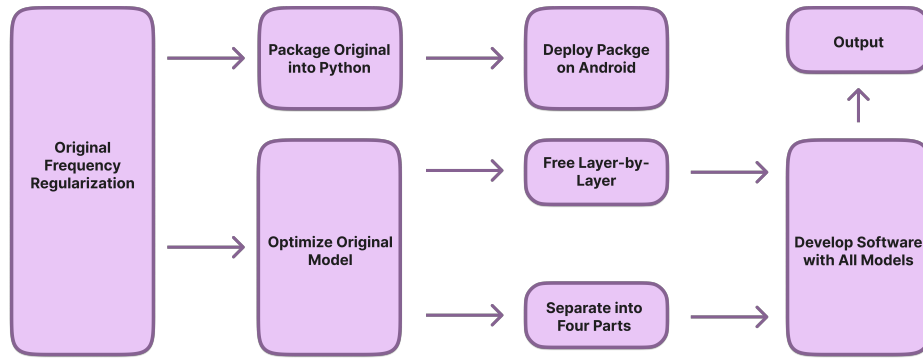


Fig. 1. General workflow of deploying frequency regularization and its optimized versions on Android devices.

In our work, before optimizing the existing FR code [10], we have two different directions of implementing the program on a mobile device. One is to install the Linux virtual environment by using Termux [25], [21], [26] on the Android system and to deploy the source code implementation. Another one is to develop an Android application which contains the code of frequency regularization by using Android Studio. As shown in Fig. 1, our workflow also contains the optimization/improvement of existing frequency regularization [32], [10]. The approach of freeing RAM layer-by-layer decreases the total usage of RAM on Android phones significantly. Moreover, the output quality of image segmentation does not have any side effects. Overall, our main contributions are:

1. Packing the frequency regularization source code into the Python library and installing it on the Android system successfully.
2. Optimizing and improving the existing frequency regularization algorithm to decrease the RAM usage on mobile devices.
3. Developing Android application containing original frequency regularization algorithm and all the other optimized versions to apply image segmentation.

Our optimized models achieve the RAM usage around 1.932 GB whereas the RAM usage of original frequency regularization algorithm is around 3.270 GB on Android devices. This means the optimized models are 40.9% relative improvements over previous algorithm [32], [10] specially for Android devices.

2 Related Work

There are four main related works in this Section: Extreme Learning Machine (ELM) [1], [23], [17], [8], [16], NestDNN [11], “One-shot Whole Network Compression” [18], and Frequency Regularization [32], [10].

Extreme learning machine (ELM) has been widely used in artificial intelligence field over the last decades [1], [23], [17], [8], [16]. Although this algorithm has seen significant development, it also contains several drawbacks. It is also the reason that we do not use ELM in our work:

- Poor tunability: It has poor controllability of the model since ELM calculates the least squares solution directly, and users cannot analyze the characteristics of the datasets to fine-tune. Adjusting models based on the specific performance of mobile devices is important to mobile development.
- Lack of robustness: The performance of the model can be affected significantly while including certain outliers in different datasets, indicating poor robustness. Deployment on mobile devices needs to handle various inputs, including every potential outlier. Although there are many advanced versions of ELM [13], [31], [33], [24] they lack universality and are not as easy as other algorithms to deploy.
- Overfitting issues: While deploying large convolutional neural networks on mobile devices, model generalization is crucial since overfitting can result in poor performance on unseen data. ELM easily leads to overfitting issues because it is based on empirical risk minimization without considering

structural risk. Xue et al. [29] pointed to a regularization strategy to solve this problem by feature selection.

NestDNN is a framework that takes the dynamics of runtime resources into account [11]. The experiment of Fang et al. [11] achieves as much as 4.2% increase in inference accuracy, $2.0\times$ increase in video frame processing rate and $1.7\times$ reduction in energy consumption. However, NestDNN also comes with some limitations. Its computational cost is significantly higher by using filter pruning method Triplet Response Residual (TRR). The high computational cost could probably exceed the processing capabilities of existing mobile devices and the runtime of model generation may be too long, which is not suitable for our deployment.

“One-shot Whole Network Compression” [18] includes removing certain parameters or structures, which is irreversible. Moreover, by using this compression method, the accuracy is too low. For example, in the experiment of Kim et al., by using AlexNet, the accuracy of the compressed model can drop by more than 50%. In order to increase its accuracy, we have to fine-tune the compressed model. Increasing accuracy requires at least more than 10 training epochs, which takes too much time. In our work, to deploy on mobile devices, this algorithm cannot be chosen obviously.

Frequency regularization (FR) [32], [10] works by restricting the non-zero elements of network parameters in the frequency domain, thus reducing information redundancy. Table 1 illustrates the evaluation of the proposed frequency regularization algorithm on UNet, according to compression rate, number of parameters, and dice score. Dice score is a metric for assessing the quality of image segmentation and ranges from 0 to 1, where 0 indicates no overlap and 1 indicates perfect overlap. The data under the dashed-line represents the result under the most extreme condition in which only 759 float16 parameters are kept in UNet-v4. Thus, according to the surprising and satisfying experiment outcomes, we chose frequency regularization as our compression method to do further implementation, to deploy it on mobile devices (i.e. Android system).

Table 1. Evaluation of the proposed frequency regularization algorithm on UNet for image segmentation tasks using Carvana Image Masking Challenge Dataset [32], [4].

	Dice Score Compression Rate # of Parameters		
UNet-ref	99.13%	100%(1 \times)	31,043,586
UNet-v1	99.51%	1%(100 \times)	310,964
UNet-v2	99.37%	0.1%(1000 \times)	31,096
UNet-v3	98.86%	0.0094%(10573 \times)	2,936
UNet-v4	97.19%	0.0012%(81801 \times)	759(float16)

3 Methodology

3.1 Problem Formulation

To implement frequency regularization algorithm [32], [10] on Android devices effectively, our primary goal is to optimize the algorithm in order to reduce the usage of random access memory (RAM). In our work, we chose image segmentation to test the efficiency of the methods. The optimization we want to achieve is ensuring that the devices run the algorithm efficiently without compromising lots of available memory resources. Simultaneously, there is another problem we need to solve: maintain the quality of results as much as possible while optimizing the algorithm. The balance between memory efficiency and the quality of images is the key to deliver an optimized user experience on mobile devices. The challenges in our work involve adjusting the algorithm to minimize the RAM usage while preserving the integrity and clarity of the processed images (i.e. image segmentation in our work).

3.2 Original and Optimized Frequency Regularization

To carry out the experiment of image segmentation, we designed two creative methods based on frequency regularization algorithm [32], [10] to optimize the usage of memory and its efficiency. We also want to make sure that the accuracy is still effective after the optimization.

- Original frequency regularization: In Section 2, Zhao et al. designed this method and we will use their source code [10] to implement it on the Android system.
- Free random access memory layer-by-layer based of the neural network: In this method, we stored the tensor information of the network layers in the local storage and free the random access memory (RAM) while each layer ends.
- Free random access memory layer-by-layer of the neural network and separate one image into four parts: The main idea is similar to the previous method. We planned to separate an image into four different parts and implement the previous method on each four parts independently. After that, we merge the four parts together to get the result.

3.3 Deployment Tools

To deploy all the frequency regularization related methods in Section 3.2 on Android devices, we have two different implementations. In other words, we have to use two different tools. One is Termux, which can implement a Linux virtual environment on an Android system. The other is Android Studio, which can help to develop Application with FR image segmentation.

- Termux [25], [21], [26] is a terminal application so that we can run the Linux virtual environment. It only requires some minimum setups. However, to run Termux, the system needs to meet some minimal requirements: Android 5.0 to 12.0; CPU: AArch64, ARM, i686, x86_64; at least 300 MB of disk space. It is open source and its documentation can be accessed at the link here. The instruction of installing Termux on Android devices is available at here.
- Android Studio is an Integrated Development Environment (IDE) designed specifically for developing applications for the Android platform. Moreover, this platform supports the PyTorch Mobile Library, which makes it to be more simple to deal with neural networks on Android system.

3.4 Qualitative and Quantitative Metrics

To analyze the results of our experiment, we have three qualitative and quantitative metrics: usage of random access memory, dice score, and visual perception. In our work, we need to analyze these three metrics collectively rather than making judgments on them individually.

- Usage of Random Access Memory (RAM): The RAM usage is a key metric when evaluating the performance of any software application. RAM usage usually refers to the amount of memory that the system allocates to a particular task or application while it is running. This quantitative metric reflects the demand for computing resources, as well as its efficiency. The lower usage of RAM is used, the higher efficiency of the application. In Section 4, we tried to decrease the RAM usage in order to avoid unnecessary waste of resources and allow more lower-end Android devices to deploy the neural networks.
- Dice Score: It is also known as the Dice Similarity Coefficient. It is a measure of the similarity between two sets of data, usually represented as binary arrays [20]. For example, in the image segmentation of Section 4, the dice score can be used to evaluate the similarity between a predicted segmentation mask and the ground truth segmentation mask [20]. Its range is between 0 and 1, representing no overlap to perfect overlap. The higher value the more closer to the ground truth. We used this quantitative method to evaluate the performance of the algorithms. Equation (1) shows the formula of dice score [28].

$$\text{Dice Score} = \frac{2 \times |X \cap Y|}{|X| + |Y|} \quad (1)$$

where X is the predicted set of pixels and Y is the ground truth.

- Visual Perception: Visual perception as a metric in image quality assessment involves evaluating images based on how well they align with human visual characteristics [14]. Though numerous image quality measures have been proposed, human visual perception is still a good way to evaluate the quality

of images [27]. In all, we used our own perception as a qualitative metric and combined it with the other two quantitative metrics to decide the best outcomes in our experiment.

4 Experiments

4.1 Experimental Settings

We utilized an Android device that ran version 12.0.1 and 8 GB RAM for this section. In order to facilitate the installment of Ubuntu operating system within the Android system, we downloaded Termux [25], [21], [26], which version is v0.118.0. Upon accessing Termux, we employed a suite of basic tools, such as wget, proot, and git, to establish the Ubuntu. The Ubuntu package [2] we used is quite different from the conventional Ubuntu installations on normal personal computers. For more detailed steps of setting up the environment, please check our source code repository [30]. For developing the Android Application, we utilized the same version of the emulator (i.e. Android Virtual Device) and made sure that the Android Studio we chose is in version 2023.1.1. The detailed steps can be checked on our source code repository as well [30].

4.2 Package Frequency Regularization Source Code

As of the current and future plans of Zhao et al. [10], [32], we have accomplished **the development of a pip repository for their frequency regularization technique** and committed to their original repository [10]. We can now integrate frequency regularization algorithm into our project by simply running the command line in the Linux virtual environment: `$ pip install frereg`. This step is instrumental in simplifying the deployment of condensed yet potent models in pragmatic applications. We will use this Python library in Section 4.3.

4.3 Build Linux Virtual Environment and Install the Frequency Regularization Library

After initiating the Ubuntu operating system [2] and installing both Python and the Python-pip tool, we used the command line mentioned in Section 4.2 and installed the frequency regularization library successfully. From our own perspectives, this method is not that innovative since it is based on the Termux, which is a mature Linux virtual environment for Android system. This part of our experiment aims to **prove the possibility of running a compressed model by building a Linux environment**, that only requires enough random access memory (RAM).

4.4 Develop an Application with Optimized Methods

As mentioned in Section 3, we utilize Android Studio to develop an application with our methods. In order to run the Python script on the Android Studio, there is a library called Chaquopy [6] that can help us. Our Android application can upload the chosen images from local storage and implement the original frequency regularization or the other optimized methods respectively. We use this application to generate all the data we need in this research, such as the usage of RAM. We also use this application to obtain the outputs from the algorithm and its optimized version running on an Android phone. The feature of the application provides currently include importing an image from the local machine directory and running a bulk of images in one run. After that, we get the output from the compressed algorithm and store it in the local device automatically.

4.5 Analyze Results

Fig. 4.5 illustrates the results among three ways of image segmentation. The left-hand column of the Fig. 4.5 shows the original images we want to implement the image segmentation. From left to right, the three columns represent three outputs generated by an Android phone respectively: output from non-optimized code, output from optimized code, output from optimized code and separate the original image into 4 parts.



Fig. 2. Comparison between Original Images and Outputs from Image Segmentation for Carvana Image Masking Challenge Dataset [4] on an Android Phone: Uncompressed U-Net, FR Compressed and Decompressed U-Net, and FR Compressed and Decompressed U-Net Processing Four Parts of an Image Simultaneously.

Table 2 shows that the average RAM usages among three experiments are 3.2687 GB, 1.9318 GB, and 1.3063 GB respectively. It is obvious that the optimized frequency regularization algorithm **decreases the RAM usage on Android devices significantly**. Because of this, we will say that more Android phones equips lower-end hardware can probably implement these two algorithms. To analyze Table 3, the table illustrates the average dice scores among three experiments. For using the compressed model (i.e. FR) directly, the dice score is 0.9718 which is identical as using the original non-compressed model. Moreover, for using the compressed model on four separated parts of the image, the dice score shows a small number of 0.7567, which means that the image segmentation under this method does not output a good quality. Combining these two tables: Table 2 and Table 3, and the visual perception in Fig. 4.5, using FR directly on Android is a better algorithm to implement on the U-Net. Obviously, the original compressed model and optimized compressed model by FR have almost the same quality of image segmentation. By visual perception, we are more certain that **optimized FR is a highly efficient algorithm** even if the system is Android.

Table 2. Comparison of Usages of RAM for Image Segmentation for Carvana Image Masking Challenge Dataset [4] on an Android Phone: Original FR, Free RAM Layer-by-Layer of neural network, and Free RAM Layer-by-Layer of neural network with Four Separate Parts in Section 3.

	Original	Optimized	Optimized & 4 parts
1 st Avg.	3.4088 GB	2.0217 GB	1.2684 GB
2 nd Avg.	3.1013 GB	1.8294 GB	1.2602 GB
3 rd Avg.	3.2959 GB	1.9444 GB	1.3823 GB
Total Avg.	3.2687 GB	1.9318 GB	1.3036 GB

Table 3. Comparison of Dice Scores for Image Segmentation for Carvana Image Masking Challenge Dataset [4] on an Android Phone: Original FR, Free RAM Layer-by-Layer of neural network, and Free RAM Layer-by-Layer of neural network with Four Separate Parts in Section 3.

	Original	Optimized	Optimized & 4 parts
1 st Avg.	0.9718	0.9718	0.7567
2 nd Avg.	0.9718	0.9718	0.7567
3 rd Avg.	0.9718	0.9718	0.7567
Total Avg.	0.9718	0.9718	0.7567

5 Limitation

5.1 Incomplete Mobile PyTorch

In some situations, some PyTorch features may still not work when running the code because of the difference in hardware structure between the computer and mobile devices [12]. For instance, the Discrete Fourier Transforms or the Fast Fourier Transforms [22], [7], [19], [5], mobile devices do not support this operation due to the requirement of GPU involvement. Therefore, an alternative solution is to convert the tensor data to the NumPy array, which runs through the CPU. After the transformation, we then convert back to the PyTorch tensor array for further steps. Many functions require conversion to make it work in the frequency regularization algorithm. But this approach will decrease the efficiency too much.

5.2 Unknown Trend

Fig. 3 shows the trend of RAM usages by using the optimized frequency regularization (i.e. free RAM layer-by-layer). For the three experiments by using the same method, we get the similar trend. It is unexpected since we thought the usage of RAM should keep decreasing while the output of each layers are saved in the local storage.

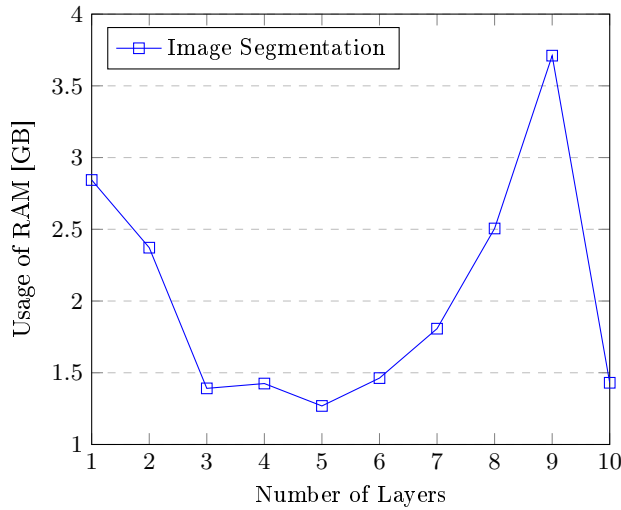


Fig. 3. Trend of One of the Three Experiments about Optimized Frequency Regularization (i.e. free RAM layer-by-layer) on RAM Usage for Image Segmentation.

6 Future Work

The main purpose of our future work is to prove the general applicability of frequency regularization algorithm [32]. We plan to expand more neural network models like ResU-Net, SegNet, X-Net, and so on [9], [3],[15]. We also plan to try figuring out the cause of the trend to be not monotonic. Moreover, adjusting and improving frequency regularization to fit more tasks with not only image segmentation is the direction we want to explore. We are also interested in continuing the development of our Android application. Our plan is to enhance this software by adding more models and features, optimizing algorithms, and aiming to make it a more efficient and user-friendly patent.

7 Conclusion

We created **the first Python library of frequency regularization** and tested it in good condition on the Android system. We also proposed **few methods of optimizing frequency regularization that helps decreasing the RAM usage without any significant loss**. This creative approach based on Zhao et al. 's frequency regularization algorithm [32] solved some limitations and improved their work a lot. Moreover, this approach helped us **deploy it on Android devices** successfully and quickly. To some extent, by using the approach proposed by us, the popularization of using large and complex neural networks on mobile devices will become possible.

References

1. Anton Akusok, Leonardo Espinosa Leal, Kaj-Mikael Björk, and Amaury Lendasse. High-performance elm for memory constrained edge computing devices with metal performance shaders. In Jiuwen Cao, Chi Man Vong, Yoan Miche, and Amaury Lendasse, editors, *Proceedings of ELM2019*, Proceedings in Adaptation, Learning and Optimization, pages 79–88, International, 2021. Springer.
2. Alexander Argentak. ubuntu-in-termux, 2023.
3. Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(12):2481–2495, 2017.
4. Maggie Mark McDonald Patricia Will Cukierski Brian Shaler, DanGill. Carvana image masking challenge, 2017.
5. E. Oran Brigham. *The Fast Fourier Transform and Its Applications*. Prentice-Hall, Inc., USA, 1988.
6. Chaquopy. Chaquopy 14.0 documentation. <https://chaquo.com/chaquopy/doc/current/>, 2023. Accessed: 2023-12-05.
7. W.T. Cochran, J.W. Cooley, D.L. Favon, H.D. Helms, R.A. Kaenel, W.W. Lang, G.C. Maling, D.E. Nelson, C.M. Rader, and P.D. Welch. What is the fast fourier transform? *Proceedings of the IEEE*, 55(10):1664–1674, 1967.
8. ChenWei Deng, GuangBin Huang, Jia Xu, and JieXiong Tang. Extreme learning machines: new trends and applications. *Science China Information Sciences*, 58(2):1–16, 2015.
9. Foivos I. Diakogiannis, François Waldner, Peter Caccetta, and Chen Wu. Resunet-a: A deep learning framework for semantic segmentation of remotely sensed data. *ISPRS Journal of Photogrammetry and Remote Sensing*, 162:94–114, 2020.
10. Guanfang Dong. Frequency regularization. <https://github.com/guanfangdong/pytorch-frequency-regularization>, 2023.
11. Biyi Fang, Xiao Zeng, and Mi Zhang. Nestdnn: Resource-aware multi-tenant on-device deep learning for continuous mobile vision. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking, MobiCom '18*. ACM, October 2018.
12. Rostislav Fojtik. New processor architecture and its use in mobile application development. In Tatiana Antipova, editor, *Digital Science*, pages 545–556, Cham, 2022. Springer International Publishing.
13. John M. Fossaceca, Thomas A. Mazzuchi, and Shahram Sarkani. Mark-elm: Application of a novel multiple kernel learning framework for improving the robustness of network intrusion detection. *Expert Systems with Applications*, 42(8):4062–4080, 2015.
14. Yan Fu and Shengchun Wang. A no reference image quality assessment metric based on visual perception. *Algorithms*, 9(4), 2016.
15. Haruki Fujii, Hayato Tanaka, Momoko Ikeuchi, and Kazuhiro Hotta. X-net with different loss functions for cell image segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pages 3793–3800, June 2021.
16. Guang-Bin Huang, Qin-Yu Zhu, and Chee-Kheong Siew. Extreme learning machine: Theory and applications. *Neurocomputing*, 70(1):489–501, 2006. Neural Networks.
17. Shui-Hua Wang Yu-Dong Zhang Jian Wang, Siyuan Lu. A review on extreme learning machine. *Multimedia Tools and Applications*, 81(29):41611–41660, 2022.

18. Yong-Deok Kim, Eunhyeok Park, Sungjoo Yoo, Taelim Choi, Lu Yang, and Dongjun Shin. Compression of deep convolutional neural networks for fast and low power mobile applications, 2016.
19. Henri J. Nussbaumer. *The Fast Fourier Transform*, pages 80–111. Springer Berlin Heidelberg, Berlin, Heidelberg, 1982.
20. OECD.AI. Dice score, 2023. Catalogue of Tools & Metrics for Trustworthy AI.
21. Masud Rana. Open and accessible education with virtual reality, 2022.
22. James C. Schatzman. Accuracy of the discrete fourier transform and the fast fourier transform. *SIAM Journal on Scientific Computing*, 17(5):1150–1166, 1996.
23. Ru Nie Shifei Ding, Xinzheng Xu. Extreme learning machine and its applications. *Neural Computing and Applications*, 25(3):549–556, 2014.
24. Kai Sun, Jianshe Zhang, Chunxia Zhang, and Junying Hu. Generalized extreme learning machine autoencoder and a new deep neural network. *Neurocomputing*, 230:374–381, 2017.
25. termux. Termux application, 2023.
26. termux. The termux wiki, 2023.
27. Rameez Wajid, Atif Bin Mansoor, and Marius Pedersen. A human perception based performance evaluation of image quality metrics. In George Bebis, Richard Boyle, Bahram Parvin, Darko Koracin, Ryan McMahan, Jason Jerald, Hui Zhang, Steven M. Drucker, Chandra Kambhamettu, Maha El Choubassi, Zhigang Deng, and Mark Carlson, editors, *Advances in Visual Computing*, pages 303–312, Cham, 2014. Springer International Publishing.
28. Varun Yerram. Understanding dice coefficient, 2020.
29. Xue Ying. An overview of overfitting and its solutions. *Journal of Physics: Conference Series*, 1168(2):022022, feb 2019.
30. Wenhao You and Leo Chang. Nerual networks on mobile devices, 2023.
31. Kai Zhang and Minxia Luo. Outlier-robust extreme learning machine for regression problems. *Neurocomputing*, 151:1519–1527, 2015.
32. Chenqiu Zhao, Guanfang Dong, Shupeizhang, Zijie Tan, and Anup Basu. Frequency regularization: Reducing information redundancy in convolutional neural networks. *IEEE Access*, 11:106793–106802, 2023.
33. Qin-Yu Zhu, A.K. Qin, P.N. Suganthan, and Guang-Bin Huang. Evolutionary extreme learning machine. *Pattern Recognition*, 38(10):1759–1763, 2005.