Hindawi
Complexity
Volume 2020, Article ID 3853780, 8 pages
https://doi.org/10.1155/2020/3853780

Research Article

Complexity of Deep Convolutional Neural Networks in Mobile Computing

Saad Naeem,[1] Noreen Jamil,[1] Habib Ullah Khan,[2] and Shah Nazir[3]

[1]*Department of Computer Science, National University of Computer and Emerging Sciences, Islamabad, Pakistan*
[2]*Department of Accounting & Information Systems, College of Business & Economics, Qatar University, Doha, Qatar*
[3]*Department of Computer Science, University of Swabi, Swabi, Pakistan*

Correspondence should be addressed to Habib Ullah Khan; habib.khan@qu.edu.qa

Received 17 July 2020; Revised 2 September 2020; Accepted 6 September 2020; Published 17 September 2020

Academic Editor: Atif Khan

Neural networks employ massive interconnection of simple computing units called neurons to compute the problems that are highly nonlinear and could not be hard coded into a program. These neural networks are computation-intensive, and training them requires a lot of training data. Each training example requires heavy computations. We look at different ways in which we can reduce the heavy computation requirement and possibly make them work on mobile devices. In this paper, we survey various techniques that can be matched and combined in order to improve the training time of neural networks. Additionally, we also review some extra recommendations to make the process work for mobile devices as well. We finally survey deep compression technique that tries to solve the problem by network pruning, quantization, and encoding the network weights. Deep compression reduces the time required for training the network by first pruning the irrelevant connections, i.e., the pruning stage, which is then followed by quantizing the network weights via choosing centroids for each layer. Finally, at the third stage, it employs Huffman encoding algorithm to deal with the storage issue of the remaining weights.

1. Introduction

Neural networks, as the name suggests, are modeled after the human brain that has complex interconnections called synapses [1], and the human brain does not stay static as it learns from its environment and continuously updates its knowledge.

Neural network works at the same principles; it can be considered as a massively parallel distributed processor that is made up of simpler computing units called neurons that can store huge amounts of knowledge in the form of weights; it is similar to a human brain, in that it stores the knowledge gained from its environment and stores this knowledge via interneuron connection strength that is also called synaptic weights.

The most common applications of these networks are pattern recognition and object recognition; as their strength comes from their adaptive nature, they are able to change and adjust their synaptic weights as their surrounding environment changes. There are various types of networks and they employ different algorithms to match the problem statement; for example, CNNs also known as convolutional neural networks are better at image recognition whereas feed forward neural networks are better at predicting the results [2–4].

These neural networks employ a variety of algorithms and different network architectures like back propagation, feed forward, reinforcement learning, deterministic annealing, and hill climbing techniques that suit the problem scenario and learn the features from their environment and store this knowledge. Storing the knowledge and applying it require a large number of neurons and the connections between these neurons to be stored and called upon later. This is where the issue of training the networks comes into play.

Training the network requires the system to provide huge amounts of training data to the network; each training example requires the network to learn and adjust its weights

and its respective interneuron connections which is a time-consuming task [1, 5].

As we know that neural networks are computation-intensive and training them requires a lot of training data and each training example requires heavy computations, we look at the ways in which we can reduce the heavy computation requirement and possibly make them work on mobile devices.

There are lots of angles that can be looked at in order to tackle the above problem.

Almost all of the work that is being done can be categorized logically by their type, for example, model compression, compression, efficiency via proper parameters, knowledge transfer, and finally all these techniques combined in different ways plus extra consideration for mobile devices.

2. Literature Review

2.1. Related Work

2.1.1. Model Compression. Using model compression techniques [4], we can greatly reduce the number of parameters provided to the network; instead of a domain expert hand picking the features, we can use AutoML that uses reinforcement learning to search the design space and improve it.

2.1.2. Compression, Sparsity, and Redundant Calculations. Discrete cosine transform [6] is a mathematical technique that can be used for pattern recognition where instead of processing the whole image it recognizes the patterns in image by reducing the dimensionality of image. Deep compression [2] tackles this issue by using a combination of pruning the network, i.e., taking out the network branches that are not relevant to our decision making, quantization, i.e., limiting the number of weights needed to store by sharing the weights between different connections, and finally encoding these weights using Huffman encoding. Although neural network-based compression [3] on top of image and video compression using compression techniques such as JPEG and discrete cosine transform and HEVC for video compression solves the high storage cost, it still suffers from intensive computations needed to perform the compression.

2.1.3. Accuracy and Efficiency via Proper Parameters (YOLO). The issue of initializing the network with proper and accurate parameters that reduces the training time for high accuracy networks is obvious but critical [5]. Using the YOLO platform ("you only look once") that divides the images into regions, more accuracy (97.8%) can be achieved than the conventional networks in a relatively lesser time and is proven to be more efficient [7].

2.1.4. Knowledge Transfer. Tianqui Chen et al. worked on a technique called accelerating learning via knowledge transfer as shown in Figure 1. Instead of training a system from scratch, they transferred the knowledge from a previous network that was already trained for performing a similar task and added a layer on top of the new network and trained it. This saved extra training time and extensive computations required to train the new network. But in turn the resultant network got deeper and grew in number of layers [1].

2.1.5. Mobile Devices. With the advent of 5G, the latency and high bandwidth problem is solved up to some extent, which made it easier to send the data back to the cloud platform for computations. But training the network and running it locally still remains a big issue [8].

Distributed network architecture technique [9] sends the heavy computation load to a central server that performs the computations and sends the results back to the mobile device; this technique requires efficient workload distribution, but it is just sending away the load from mobile device and not really performing on-board computations; nonetheless, this technique is still deployed in many scenarios and does perform reliably.

Light-weight CNNS [10] leverages separable convolution concept to train the system faster where the amount of computations required reduces significantly. But the process still involves heavy amount of computation to be done on a mobile device so the network is still trained on a system that can handle this computation-hungry algorithm; then after the system is trained, it is deployed on a mobile edge device. ShuffleNet Architecture [11] designed for mobile devices uses channel shuffling and point-wise convolutions to reduce the number of computations to be performed; although the technique works better than the techniques in the category of mobile devices, it still remains applicable for problems of relatively lower complexity. Quantized convolutional networks [12] for mobile devices are most promising so far as they attempt to reduce both the computation cost and the storage cost by compression of parameters and using mathematical models for prediction results as shown in Figure 2. Minimizing the estimation error is the key driver behind this technique.

Figure 2 shows the impact of quantization process in terms of storage and time requirements where the quantized network requires significantly less storage for the weights and is faster to train (shown in blue).

2.2. Critical Review

2.2.1. Accelerated Learning via Knowledge Transfer. Accelerated learning via knowledge transfer technique comes under the category of network initialization techniques which attempts to cut off the network training time via initialization of a new network by a previously trained network doing a similar task. The trained network is named as teacher network and the new network which is being initialized is called the student network. This technique was named Net2Net by Chen et al. [1].

Although the Net2Net technique does reduce the network training time significantly and accelerates the learning

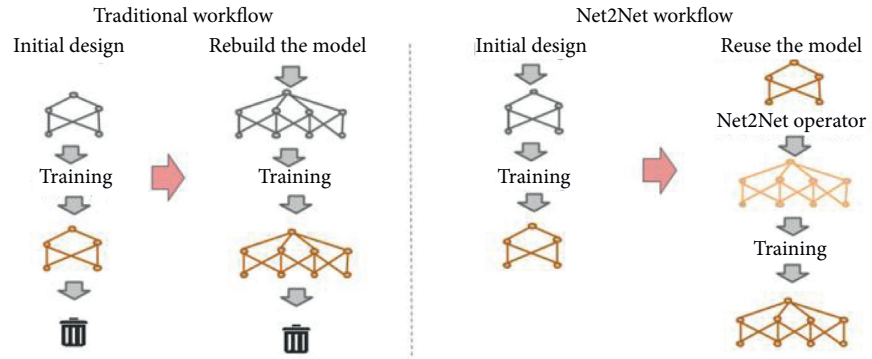

Figure 1: The traditional training process versus the Net2Net workflow where the training model is reused to train the student network.

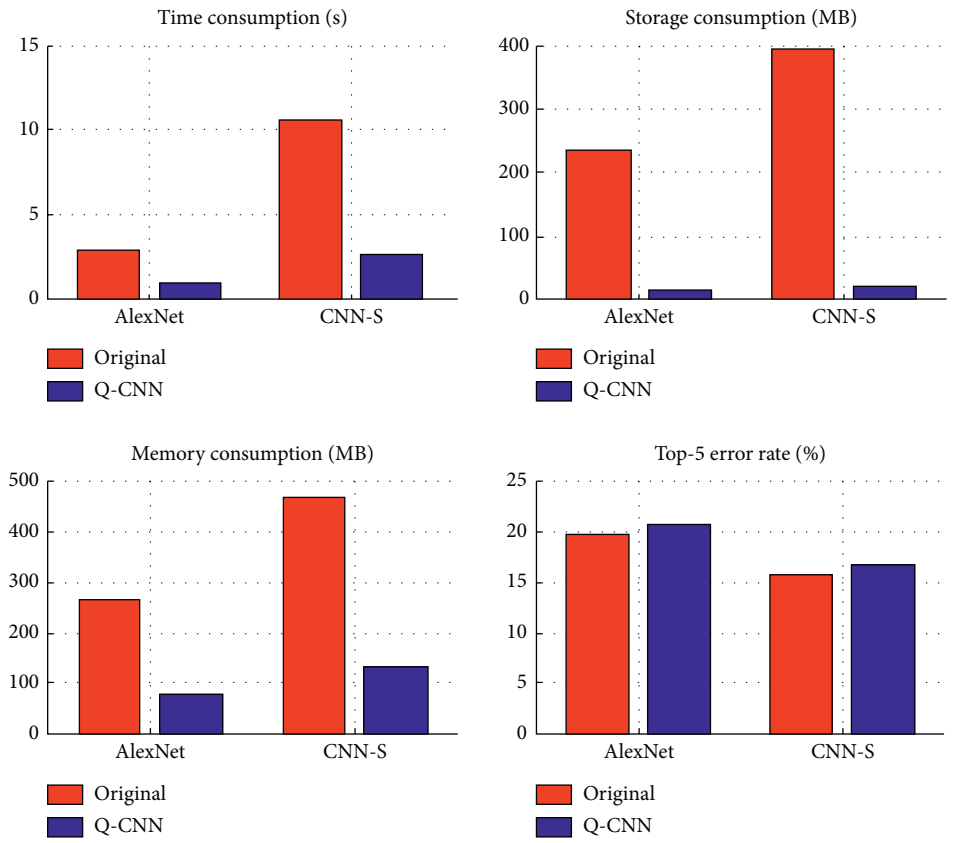

Figure 2: Alex Net versus the quantized convolutional network (blue) demonstrating the significant savings in storage and time complexity.

process, it also introduces additional layers in the network that are required to fine-tune the newly initialized network for doing the task specific to the student network; this makes the student network much deeper than the teacher network at least by a factor of 1.5 which in turn introduces redundant calculations at each layer. The authors demonstrated their work by training two networks side by side; one was trained from scratch and the second network was trained using Net2Net technique and the results were compared by clocking the training times for both techniques; the results showed that the training time was almost reduced to half.

2.2.2. Discrete Cosine Transform. Discrete cosine transform is a mathematical technique for pattern recognition that can be applied in signal processing as well that reduces the feature space and tries to solve the dimensionality reduction problems. Ahmed et al. [6] argue that for pattern recognition problems discrete cosine transform would lend itself better for signal processing than Fourier Transform; they demonstrated this via testing against the system applying Fourier Transform to the dimensionality reduction problem; the results showed that cosine transform extracts the feature space much better by extracting the most relevant patterns.

And the distortion in the results was much less as compared to Fourier Transform.

2.2.3. Deep Compression. Deep compression takes the whole development pipeline of neural network into consideration starting from pruning, then reducing the number of network weights, and finally encoding the weights using the process shown in Figure 3. Implementing these three-staged pipe lines requires significant amount of work and strictly follows the model specification, since missing any of which could lead to loss in accuracy.

Han et al. [2] were able to demonstrate the training speed-up using a smaller network efficiently but as the network size grows larger this technique starts suffering from scaling problem and the network accuracy is lost significantly.

2.2.4. Neural Network-Based Compression. Neural network-based compression is proposed to compress the audio-visual data using neural networks so that subsequent training time could be reduced using the compressed data. Neural network-based compression requires a trained network for a specific type of data to extract the important features but suffers from generalization problem; i.e., this technique does not generalize well when the data varies significantly in its features leading to inefficient compression and even missing important features that should have been considered. To tackle this problem, a domain expert is required to fine-tune the extraction process which could be considered a drawback of this technique. Using homogeneous data, Siwei Ma et al. [3] were able to compress the data using neural nets but failed to do so without any intervention from domain expert.

2.2.5. Mobile Devices. Mobile devices suffer from bandwidth and latency issues and are not able to handle large amount of neural network computations on board. Ahmed and Rehmani [8] tried to tackle this problem by sending the computation load to the servers via network and getting the results back from servers and displayed the results. Although they tackled the problem, this technique did not solve the problem of running the neural network on board chip. The major drawback of this technique is that it requires network connection and consumes significant amount of bandwidth. They demonstrated this by sending the computation load to the remote server and clocking the response time; the results showed a significant speed-up as compared to when they tried to run. The same computations on board resulted in system halt, therefore, proving that mobile devices cannot handle these kinds of computations when performed on board.

2.2.6. Model Compression. Model compression attempts to speed up the neural networks by determining the optimal compression policy; it does so by looking at the sparsity at each layer and outputs sparsity ratio which is then used as an input for compression but the problem is that every layer has different redundancies in it and is not constant.

Using reinforcement learning technique, a pertained network is required that scans the problem space and then only outputs an optimal compression ratio which is then used to perform network pruning. Another issue that has to be dealt with using AutoML technique is that in order to make the exploration process of the design space faster the final accuracy is tested without fine-tuning the reward accuracy for the agent and the argument being that this accuracy is an approximation to the final accuracy after the reward accuracy is fine-tuned.

Then, finally there is the learning agent itself which needs to be trained for different situations, i.e., whether the agent will get any reward for going below the budgeted constraints and what kind of tradeoffs the agent should balance, i.e., between time, space, and accuracy.

The reward function also needs to be tweaked manually in order to arrive at the compression ratio that suits the problem; for example, if there are no time constraints, then the reward function is adjusted to arrive at the optimal compression policy without any loss of accuracy but mostly this might not be the case as most of the times the agent is working under some sort of time or space constraints which usually results in the loss of accuracy.

2.2.7. Accuracy and Efficiency via Proper Parameters. Accuracy and efficiency via proper parameters is more of a heuristic than a technique that emphasizes initializing the network with appropriate parameters; doing so significantly reduces the training time and approaches the accuracy threshold value and converges faster than a network that is initialized with irrelevant features because the network training process spends significant amount of time in learning the important connection. Radovic et al. [5] demonstrated this by initializing the network with synthesized and relevant features and only then started the training process and the time log results showed that the properly initialized network converged faster than the network without proper initialization. And the resultant network recognized the objects using CNN with 98% accuracy which is an ideal case in object recognition problems. This heuristic appeals to the common sense of the developer so it does not really have a downside to it and the results showed a network trained in lesser time without any loss of accuracy.

2.2.8. YOLO Platform. YOLO platform implements a technique called "you only look once"; i.e., instead of few iterations, the network is shown the training data only once while extracting only the most relevant and important features and is expected to recognize the same objects accurately when shown back. This technique is suitable for real-time object detection; it looks at what objects are present in the image as well as where they are. It works in real time by dividing the whole image into different grids and looks at what objects are present in each grid as well as where they are. It does so via only single feed forward propagation pass on all the grids simultaneously, hence the name "only look once." After applying non-max suppression that deals with getting rid of multiple bounding boxes for a single

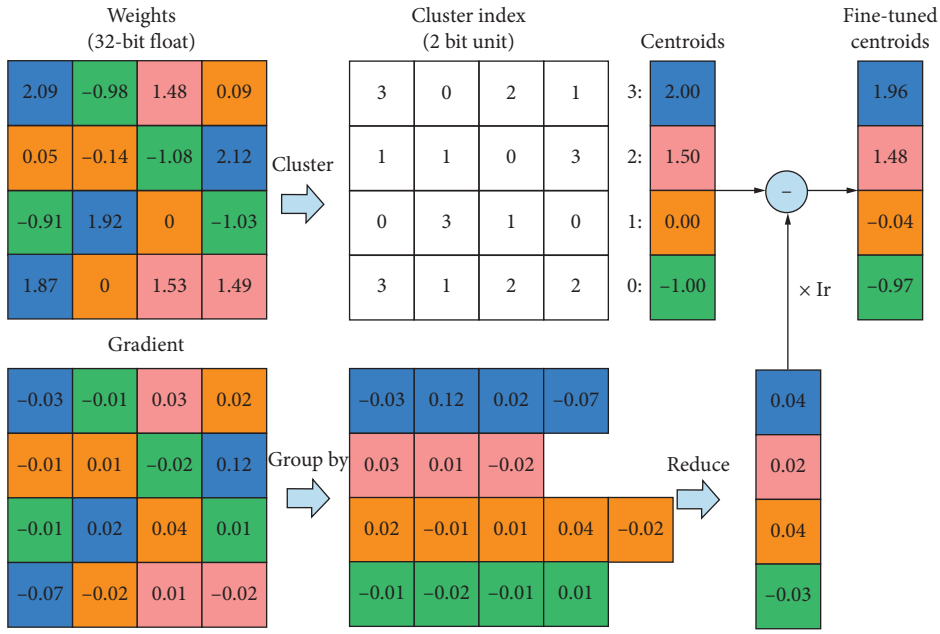

Figure 3: The quantization process where the network weights are compressed via centroid calculation.

object, it outputs the final prediction along with the box that shows the boundary around the detected object but as one might expect there is significant loss in accuracy because of less training time. Nonetheless, the technique works relatively well for situations where training time is a major constraint and the type of input data changes frequently.

2.2.9. Light-Weight CNNS. Light-weight CNNS leverages separable convolution concept to train the system faster where the amount of computation required reduces significantly. But the process still involves a heavy amount of computation to be done on a mobile device, so the network is still trained on a system that can handle this computation-hungry algorithm; i.e., on remote servers, after the system is trained it is deployed on a mobile edge device.

2.2.10. Distributed Network Architecture. Distributed network architecture is implemented to accurately perform video surveillance on mobile devices using edge computing. The multilayered architecture has to send the work load to the nearest server for feed analysis. This architecture requires significant amount of bandwidth and suffers from latency issues but Chen et al. [9] and Chen et al. [1] did solve the initial problem that involved performing real-time computations for facial recognition using mobile devices by sending the recognition query to the nearest server for analysis and got back the results over the network that were as accurate as a system running on full-fledged dedicated server.

2.2.11. Quantized Convolutional Networks. Wu et al. [12] reduced the network's weight storage cost computation overhead by quantizing the network weight allowing faster computation results. A major drawback comes in the form of 2.5 percent accuracy loss because of the weight quantization where the weights are quantized using clustering algorithm which takes all the weights of a single network layer and calculates their centroid, i.e., mean, and that value is stored in a separate matrix. Doing so also impacts the performance while computing the gradient descent. The authors argue that the 2.5% accuracy loss is less as compared to the amount of computational speed-up and the storage cost saved.

2.2.12. ShuffleNet Architecture. ShuffleNet architecture is a class of convolutional networks designed specifically for mobile devices like drones and robots that have constraints on power and computational power; this technique maintains its accuracy by cross-channel feature; sharing this technique increased the performance by 7.8% as compared to the state-of-the-art *Mobile-Net* architecture but all of this under the computation budget of 40 MFLOPs due to ARM mobile processor; this technique only relates to mobile processors like Snapdragon and ARM chips that are designed for mobile platforms and due to power constraints operate under different specifications than a traditional processor; this technique is directly tied with mobile hardware and can improve its performance with the improvement in 40 MFLOPs constraint.

2.3. Comparative Study. The common parameters that are available between different techniques are shown in the comparison table but other differences that are not common are discussed in the paragraph format after the comparison table:

Initialization means whether the technique is applied at initialization time

Compression means whether network compression was applied

Quantization means whether the weights were quantized

Speed-up refers to whether training time could be reduced or not

Onboard means whether the computations are being performed on the device

2.3.1. Other Differences between the Techniques. Other than the differences between common parameters in the techniques as shown in Table 1, there are some architectural differences as well which are given and compared below.

(1) Accelerated Learning via Knowledge Transfer. This technique is unique from others in a sense that it attempts to accelerate the learning process via knowledge transfer whereas other techniques like discrete cosine transform [6] are a mathematical based approach for pattern recognition and they attempt to solve the network speed-up problem from a different mathematical point of view.

(2) Discrete Cosine Transform. This technique comes under the category of mathematical techniques that can be used for pattern recognition where instead of processing the whole image it recognizes the patterns in image by reducing the dimensionality of image whereas other techniques like accelerated learning (1) attempt to tackle the speed-up and accuracy from purely a computer science perspective and devise a clever technique to reduce the training time via network initialization.

(3) Deep Compression. This technique stands out from other techniques by developing a complete pipeline from network initialization to training. It tackles this issue by using a combination of pruning the network, i.e., taking out the network branches that are not relevant to our decision making, quantization, i.e., limiting the number of weights needed to store by sharing the weights between different connections, and finally encoding these weights using Huffman encoding. This achieves relatively better speed-up than all the other techniques at the cost accuracy loss.

(4) Neural Network-Based Compression. This technique is used on top of image and video compression using compression techniques such as JPEG and discrete cosine transform and HEVC for video compression: although this technique solves high storage cost, it still suffers from intensive computations needed to perform the compression. This technique differs significantly from *deep compression (3)* because it attempts to perform automated compression using neural networks whereas deep compression performs the same process with the help of domain expert.

(5) Mobile Devices. This technique takes a different approach than all the other techniques; instead of performing the computations on board it sends the computation load to a remote server. With the advent of 5G, the latency and high bandwidth problem is solved to some extent, which made it easier to send the data back to the cloud platform for computations. But training the network and running it locally still remains a big issue. This technique employs the same architecture as that employed by *distributed network architecture (10)* that also sends its load to a remote server.

(6) Model Compression. This technique comes under network initialization category; in addition to this, an additional benefit of automated compression is there. It uses the same heuristics that are used by *neural network-based compression (4)*; i.e., we can greatly reduce the number of parameters provided to the network; instead of a domain expert handpicking the features, we can use AutoML that uses reinforcement learning to search the design space and improve it. This technique is different from other techniques in a sense that it tries to minimize the human factor that is involved in handpicking the features that are redundant and then prunes it. This technique like only few others sits in the category of initial optimization techniques that are applied before the network is even initialized like *(1) Net2Net* technique. It tries to solve the speed-up problem from a different angle instead of looking at avoiding redundant calculations at the run time like *deep compression (3)*; it tries to solve the problem by handling it at the initialization time via pruning the redundant channels in the network automatically without human intervention; this trait also sets this technique apart from other techniques which tries to handle the speed-up problem from different angles like compression and encoding.

(7) Accuracy and Efficiency via Proper Parameters. This technique comes under the category of heuristics. It is not really a technique but a common sense, although obvious but critical is the issue of initializing the network with proper and accurate parameters that reduces the training time for high accuracy networks; other techniques like *YOLO platform (8)* extensively employs this heuristic in its implementation. Z. Ullah et al. employed a similar technique in their networks to significantly reduce the training time.

(8) Using the YOLO Platform ("you only look once"). This technique divides the image into regions; an accuracy of 97.8% can be achieved which is less as compared to the conventional networks but in a relatively lesser time and is proven to be more efficient than other techniques in terms of the training time that it requires. The training time is lesser in orders of magnitude than the other techniques but still suffers from accuracy loss; like other techniques, it also balances between training time and accuracy; i.e., by getting faster training time, it compromises the accuracy of the network.

(9) Light-Weight CNNS. This technique tries to solve the same problem as attempted by *distributed network architecture (10)* and *mobile devices (8)*; it partially solves the problem via storing the network weights on the mobile device after training the network on remote servers. By employing this technique, the system does not suffer from latency and bandwidth issues like *distributed network*

Table 1: Comparison.

Ref.	Techniques/common parameters	Initialization	Compression	Quantization	Speed-up	Onboard
1(1)	Accelerated learning	✓	✗	✗	✓	✓
2(2)	Discrete cosine transform	✓	✓	✓	✗	✓
3(3)	Deep compression	✓	✓	✓	✓	✓
4(4)	Neural network compression	✗	✓	✗	✓	✓
5(5)	Mobile devices	✗	✗	✗	✓	✗
6(6)	Model compression	✓	✓	✓	✓	✓
7(7)	Proper parameters	✓	✗	✗	✓	✓
8(8)	YOLO platform	✓	✗	✗	✓	✓
9(9)	Light-weight CNNs	✗	✗	✗	✓	✓
10(10)	Distributed network architecture	✗	✗	✗	✓	✗
11(11)	Quantized CNNs	✗	✓	✓	✓	✓
12(12)	ShuffleNet architecture	✗	✗	✓	✓	✓

architecture (10) does. Uddin et al. [13] talked about deploying similar architecture for detecting terrorist activities in real time.

(10) Distributed Network Architecture. This technique sends the heavy computation load to a central server that performs the computations and sends the results back to the mobile device; this technique requires efficient workload distribution, but it is just sending away the load from a mobile device and not really performing onboard computations. *Light-weight CNNs (9)* solves the latency and bandwidth issue more efficiently; nonetheless, this technique is still deployed in many scenarios and performs reliably.

(11) Quantized Convolutional Networks. These techniques for mobile devices are most promising so far as they attempt to reduce both the computation cost and the storage cost by compression of network parameters and using mathematical models for prediction, where the mathematical modeling relates to defining some inherent properties in the data either statistically or by using a function adopted from classical Hamiltonian and Lagrangian systems that outputs optimal parameter quantization without much information loss. Minimizing the estimation error is the key driver behind this technique. This technique can be easily scaled to handle bigger computational loads as well. The quantization process used in this technique is the same as the one implemented by *deep compression (3)* technique.

(12) ShuffleNet Architecture. This technique designed for mobile devices uses channel shuffling and point-wise convolutions to reduce the number of computations to be performed although the technique works better than the techniques in the category of *mobile devices (5)*, but it still remains applicable for problems of relatively lower complexity.

3. Possible Ways of Extending the Work

3.1. Via Compression. Most of the storage that is used by a neural network is in the form of network weights that store the network knowledge. The current compression includes Huffman encoding for efficiently storing the network weights but the video compression like HDLC is still inefficient for compression problems. A possible extension to video compression could be via defining optimal compression ratio. Since a video feed is just a sequence of frames (like images) and the information between subsequent frames does not vary as much in its content, an optimal compression ratio could be computed via CNNs by object detection to look for variance in a video feed by CNNs; the frames where the informational content does not vary as much could be taken out in order to achieve better compression ratio.

3.2. Efficient RAM Storage. The policy of storing the network weights in the RAM also has a direct effect on the performance of the network; currently, there is no optimal policy defined for network weight storage in the RAM as they are brought into the RAM on the basis of usage from secondary storage. Similar to algorithms that are used for disk scheduling like FCFS (first come first serve) and SSTF (shortest seek time first) combined with disk read prediction, i.e., predicting the weights that could be accessed next based on the previous history of disk access could be brought into the RAM and the ones having prediction of above 98.5% could be brought into the cache for immediate access, these policies or algorithms could also be optimized using machine learning. As more time passes, the model becomes more and more accurate.

3.3. Better Quantization. Currently, the weights are quantized by calculating the weight centroids for each layer and storing them in a matrix that is half of the original layer size but there is no calculated relation between the subsequent layers; perhaps a better strategy would be to calculate the ratio between the centroids of two subsequent layers and storing the ratio factor as a weight for the second layer; this could reduce the storage cost even further by a factor of 2.5%, but one will need to verify whether there would be any loss in network accuracy.

3.4. Improving Neural Network-Based Compression. A specific network trained extensively just for the purpose of compression on massively varying and heterogeneous large data sets that could predict the optimal compression policy

just by looking at sample data while residing on cloud could significantly reduce the compression overhead. We think current *neural network-based compression (4)* was dismissed too quickly at not being accurate and having large computational overhead; the authors did not explore the possibility of training the network on cloud and exploring the possibility of getting the results from that network remotely.

4. Conclusions and Future Work

The main issue involved in running the neural networks is the time that is required for training, the storage space that is required for storing the network weights, and finally the accuracy that is given by the network as the output.

Most of the time while implementing the networks, the designers have to trade off between these three; most of the techniques attempt to tackle the problem at the initialization stage by devising clever ways to efficiently initialize the network by doing so; the network training time is reduced significantly.

Another set of techniques like *deep compression (3)* tries to tackle the problem by targeting the storage angle. This kind of techniques tries to minimize the storage space consumed by the network by efficiently storing those weights.

Finally, factoring in the scenario calls for tradeoffs in accuracy of the networks; i.e., if there is a constraint on training time, then accuracy is affected significantly which leads us to a final conclusion:

There is no silver bullet to solve this problem; a lot of work that is being done in this area is via considering different parameters and how to effectively utilize them.

The answer could be using all the combinations of the above techniques at each level in their most effective and optimized form. In the future, the proposed techniques will involve both the hardware-based and software-based solutions where different combinations of these two solutions are combined with different hyperparameters and experimenting with them to get an optimal training time while balancing all the three constraints, i.e., time, space, and accuracy.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Disclosure

The findings achieved herein are solely the responsibility of the authors.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This article was supported by Qatar University Internal Grant no. IRCC-2020-009.

References

[1] T. Chen, I. Goodfellow, and J. Shlens, "Accelerating learning via knowledge transfer," 2016, https://arxiv.org/abs/1511.05641.

[2] S. Han, H. Mao, and W. J. Dally, "Deep compression compressing deep neural networks with pruning, trained quantization and huffman coding," in *Proceedings of the Conference paper at ICLR*, San Juan, PR, USA, May 2016.

[3] S. Ma, X. Zhang, C. Jia, Z. Zhao, S. Wan, and S. Wang, "Image and video compression with neural networks," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 30, no. 6, pp. 1683–1698, 2019.

[4] Y. He, J. Lin, Z. Liu, W. Hanrui, Li-J. Li, and S. Han, "AMC: AutoML for model compression and acceleration on mobile devices," in *Proceedings of the European Conference Computer Vision Foundation ECCV*, Munich, Germany, September 2018.

[5] M. Radovic, O. Adarkwa, and Q. Wang, "Object recognition in aerial images using convolutional neural networks," *Journal of Imaging*, vol. 3, 2017.

[6] N. Ahmed, T. Natrajan, and K. R. Rao, "Discrete cosine transform," *IEEE Transactions on Computers*, vol. 23, no. 1, pp. 90–93, 1974.

[7] W. K. Pratt, J. Kane, and H. C. Andrews, "Hadamard transform image coding," *IEEE*, vol. 57, no. 1, 1969.

[8] E. Ahmed and M. H. Rehmani, "Mobile edge computing: opportunities, solutions, and challenges," *Journal of Future Generation Systems*, vol. 70, pp. 59–63, 2016.

[9] J. Chen, K. Li, Q. Deng, K. Li, and P. S. Yu, "Distributed deep learning model for intelligent video surveillance systems with edge computing," 2019, https://arxiv.org/abs/1904.06400.

[10] S. Y. Nikouei, Yu Chen, S. Song, R. Xu, B.-Y. Choi, and R. F. Timothy, "Smart surveillance reducing high computation cost for neural networks and mobile computing," 2018, https://arxiv.org/abs/1805.00331.

[11] X. Zhang, X. Zhou, M. Lin, and J. Sun, "ShuffleNet: an extremely efficient convolutional neural network for mobile devices," in *Proceedings of the 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 6848–6856, Salt Lake City, UT, USA, 2018.

[12] J. Wu, L. Cong, Y. Wang, Q. Hu, and J. Cheng, "Quantized convolutional neural networks for mobile devices," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4820–4828, Las Vegas, NV, USA, June 2018.

[13] M. I. Uddin, S. A. A. Shah, M. A. Al-Khasawneh et al., "A novel deep convolutional neural network model to monitor people following guidelines to avoid COVID-19," *Journal of Sensors*, vol. 2020, pp. 1–16, 2020.