Kernel-Controlled DQN Based CNN Pruning for Model Compression and Acceleration

Romancha Khatri
South Dakota State University

KERNEL-CONTROLLED DQN BASED CNN PRUNING FOR MODEL COMPRESSION AND ACCELERATION

BY

ROMANCHA KHATRI

A thesis submitted in partial fulfillment of the requirements for the

Master of Science

Major in Computer Science

South Dakota State University

2020

THESIS ACCEPTANCE PAGE

Romancha Khatri

This thesis is approved as a creditable and independent investigation by a candidate for the master's degree and is acceptable for meeting the thesis requirements for this degree. Acceptance of this does not imply that the conclusions reached by the candidate are necessarily the conclusions of the major department.

Kwanghee Won Advisor	Date
Siddharth Suryanarayanan Department Head	Date
Dean, Graduate School	Date

DEDICATION

I dedicate this thesis to my beloved parents, little sister and my wife. First, I dedicate this to my mother, I would not be the person what I am without her guidance, encouragement, and the most prominent thing is her willingness in my interest. Thank you so much for befriending me with the machine at my premature age and enrolling me in the College of Engineering; finally, a dream comes true. Second, I dedicate this to my wife for her patience, support and encouragement amid of my hectic schedules. I hope she will understand why I spent more time on my computer than with her during my research work. Appreciate you all. Love you. Thanks god.

ACKNOWLEDGEMENTS

First and foremost, I would like to express my sincere gratitude to the College of South Dakota State University and advisor Dr. Sung Y. Shin for letting me fulfil my dream of pursuing master's degree in Computer Science.

I would like to express my deep and sincere gratitude to my research advisor Dr. Kwanghee Won for giving me the opportunity to do research and providing priceless guidance throughout this research. His sincerity, diligence, vision, and motivation have deeply inspired me. I am extremely grateful for your assistance, directions and suggestions throughout my project. The author is pretty much thankful for having such a good advisor like you.

I would like to thank my committee members, Dr. George Hamer, CS Graduate Coordinator Dr. Sung Y. Shin, research advisor Dr. Kwanghee Won as well as the graduate faculty representative Dr. Christine Larson for providing necessary information in completing this endeavor.

Last but not the least, I would like to thank my family; my parents Mr. Khum Bahadur Khatri and Mrs. Geeta Devi Khatri, for prayers, caring and supporting me spiritually and financially throughout my life and sending me far away for a higher degree of education, without you none of this would indeed be possible. I am very much thankful to my wife Anasuya K.C. for her love, understanding, patience and continuing support to complete this research work.

TABLE OF CONTENTS

ABBREVIATIONS

CNN: Convolutional Neural Network

DQN: Deep Q Network

DNN: Deep Neural Network

RNN: Recurrent Neural Network

RL: Reinforcement Learning

AEC: Accuracy Ensured Compression

CECA: Compression Ensured Considering Accuracy

ReLU: Rectifier Layer Unit

FC: Fully Connected

Conv2d: Convolutional Layer, 2 dimensions

MaxPool2d: Max Pooling Layer, 2 dimensions

LIST OF TABLES

LIST OF FIGURES

LIST OF EQUATIONS

KEYWORDS

Deep Neural Network Compression, Convolutional Neural Network, Accuracy Ensured Compression, Compression Ensured Model, Deep Reinforcement Learning, Kernel Pruning, Neural Network Model Compression

ABSTRACT

KERNEL-CONTROLLED DQN BASED CNN PRUNING FOR MODEL COMPRESSION AND ACCELERATION

ROMANCHA KHATRI

2020

Apart from the accuracy, the size of convolutional neural networks (CNN) models is another principal factor for facilitating the deployment of models on memory, power and budget constrained devices. However, conventional model compression techniques require human experts to setup parameters to explore the design space which is sub-optimal and time consuming. Various pruning techniques are implemented to gain compression, trading off speed and accuracy. Given a CNN model [11], we propose an automated deep reinforcement learning [9] based model compression technique that can effectively turned off kernels on each layer by observing its significance on decision making. By observing accuracy, compression ratio and convergence rate, our model can automatically re-activate (turned on) the healthiest(fittest) kernels to train it again which greatly ameliorate the model compression quality. Experimented results on MNIST dataset [7], the proposed method reduces the size of convolution layers for VGG-like model [9] up to 60% with 0.5% increase in test accuracy within less than a half the number of initial amount of training (speed-up up to 2.5×), state-of-the-art results of dropping 80% kernels (86% parameters compressed) with increase in accuracy by 0.14%. Further dropping 84% kernels (94% parameters compressed) with the drop of test accuracy 0.40%. The first proposed Auto-AEC (Accuracy-Ensured Compression) model can compress the network by preserving original accuracy or increase in accuracy of the model, whereas, the second proposed Auto-CECA (Compression-Ensured Considering the Accuracy) model can compress to the maximum by preserving original accuracy or minimal drop of accuracy. Based on experiments, further analyzed effectiveness of kernels on different layers based on how proposed model explores & exploits in various stages of training.

1. INTRODUCTION

Over the last few years, deep learning models have been continuously achieving state-of-the-art performance in a variety of computer vision task, ranging from Visual Data Processing- Image Classification [6], Segmentation [6], Object Detection and Object Recognition [16][14] , Natural Language Processing- (Natural Language Generation - Image Captioning, Question Answering, Market Intelligence, Sentiment Analysis, Natural Language Understanding - Language Translator, Speech & Text Recognition), Bio-medicine, Autonomous driving, and so on. Most of the applications require high volumes of information for training the model that result in the requirement of high memory and powerful computational hardware to achieve state-of-art performance. For deploying the model in a constraint environment (memory, power, computation) locally (on smart-phones, drones, autonomous vehicles, glasses, smart watches) and globally on cloud is painful and insecure.

Furthermore, running large networks requires a huge memory bandwidth to fetch enormous weights and tremendous amounts of matrix multiplications, FLOPs, which consumes considerable amounts of energy. Therefore, it is essential to reduce the size of models which has relatively low computational cost, low memory footprint but high accuracy and high convergence in real- world applications.

Various approaches have been proposed and tested in Convolution Neural Network for dimensionality reduction in both convolution layers (dropout, pooling, point-wise convolution) and fully-connected layers (replacing with Global Average Pooling Layer), computation reduction (Group convolution - for data & model paralyze , point-wise convolution, Depth-wise Separable convolution [5] , Fast Fourier Transform(FFT)

& Inverse Fast Fourier Transform (IFFT) convolution, and Winograd Convolution [12]. Although, the above methodologies worked for few CNN models, they also suffered from various application criteria. For example, group-wise convolution solved computation overload by sub-sampling inputs and processed independently, but the number of channels increased as we progressed through the layers because of concatenation applied after inception model. FFT & IFFT only worked well for large filters, whereas, Winograd Convolution solved the problem working with small filters but only applicable where there is overlapping for example, it would not be feasible to apply for 1×1 convolution or 2×2 convolution with stride 2 and so on.

Pruning techniques solved the problem by reducing the memory footprint and can be implemented in low computing smart devices, but the major challenges - pruned channels & weights are irreversible, in other words, once they are pruned, they will not be recovered. During the training process, the significance of kernels may vary. For instance, a #2 kernel for convolution layer-1 could play an important role in identifying a high-level feature for image classification at the end of training, but it could be pruned in the earlier part of training and never be recovered. Many works have focused on pruning methodologies, but they did not study the significance of each weight / kernels in different stages of training and once the weights/filters are selected & pruned, it will not get recovered.

In this work, we propose to develop a novel Auto-AEC (Automatic Accuracy Ensured Compression) model for deep compression by addressing problems with a hand-crafted compression rate which is sub optimal & time consuming, slow convergence [iterative pruning], unrecoverable pruned weights/kernels. We have used a Deep

Reinforcement Learning algorithm (Deep Q Learning) for selecting appropriate kernels and perform binary operations (active and inactive) for kernels in convolution layers that improves the model compression quality. Inactive operation is used to turn off kernel/s whereas active operation is used to turn on inactive kernels if needed. Turning on and off operations are automatized by using Deep Q Learning to select kernels based on kernel's efficiency and training stage.

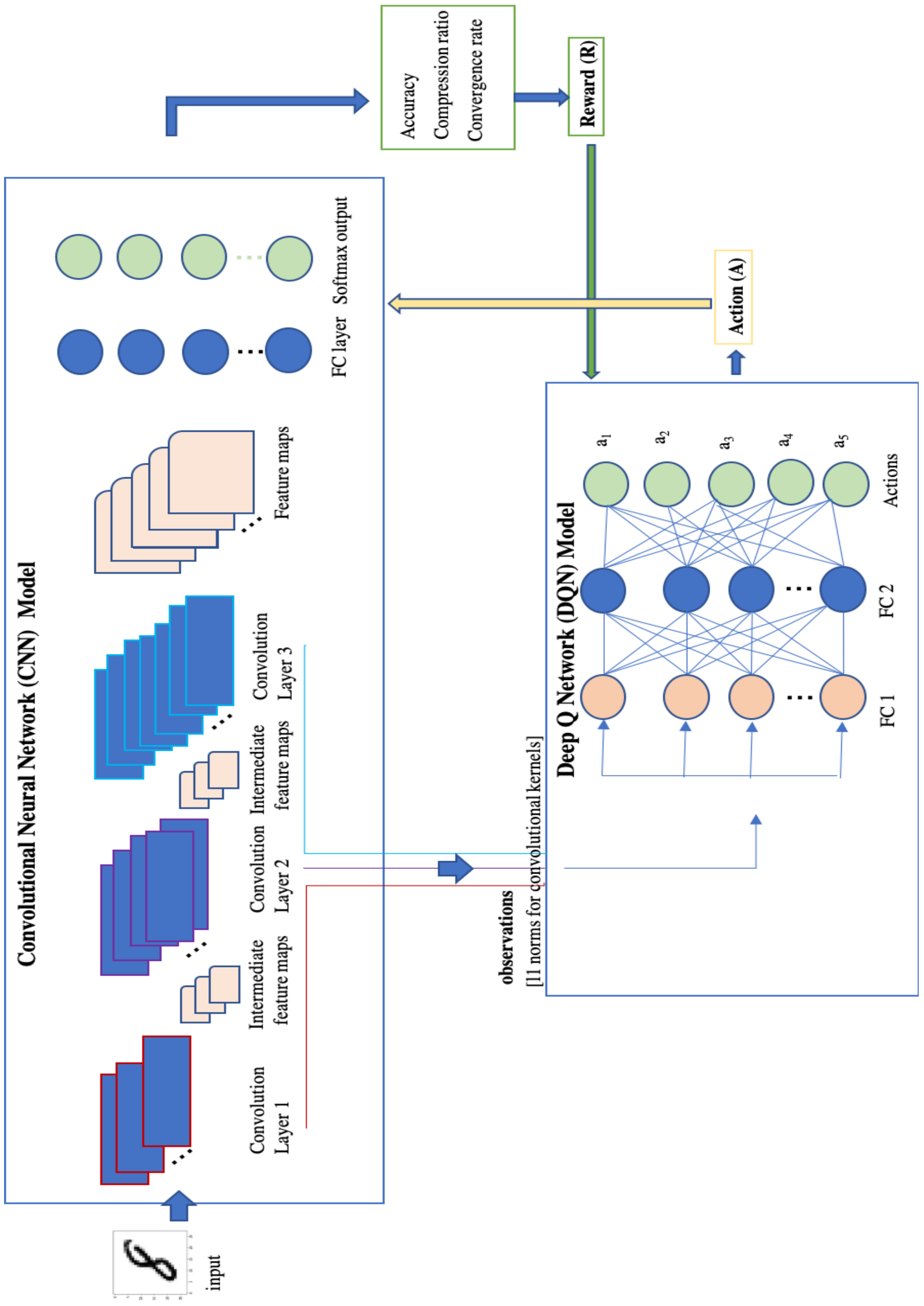


Figure 1: Architecture of Deep-Q compression model

2. RELATED WORK

As per a survey conducted by Yu Cheng [4], different approaches for model compression and acceleration are listed below :

i. Parameter pruning & sharing (quantization, block coding & encoding)

ii. Low-rank factorization (low rank filter, low rank approximation)

iii. Knowledge distillation (teacher-student model)

iv. Transferred/compact convolution filters (filter/channel pruning)

By exploring the above approaches, although each methodology has surpassed their performance on different types of network, human expertise was required to set up hyper-parameters, design space before implementing it. Method (i) & (ii) outperformed in many applications as they could support both train from scratch & pre-trained models. The rest of the approaches could only support training from scratch so they would be more challenging for implementation on Knowledge Transfer. Furthermore, method (iv) channel/filter pruning could only be applied in Convolution layers, so we could not deal with the parameters in FC layers. Song Han [8] also concluded that FC layers have more redundancy compared to convolution layers. They could prune up to 90% parameters on FC layers and up to 70% parameters on convolution layers without or with negligible accuracy drop.

Pruning channels/filters could be more sensitive as an individual filter has high responsibility on generating a feature map results in loss of prominent features. The problems with method (i) are to identify the appropriate threshold parameter, and manual setup for the sensitivity of each layer's threshold for parameters pruning. Manual setup for threshold will lead to inconsistency in training and another important aspect is retraining, fine tuning the pruned network after each pruning operations to compensate for pruned weights. Many works focus on compressing existing networks by setting up hand-crafted compression ratio for each layer, which requires a human expert. To sum up, it will elongate training time for the model.

Therefore, to address aforementioned problems, proposed Deep Compression neural networks with pruning [8], trained quantization & Huffman coding and achieved state-of-art performances reducing the model size on AlexNet by 35×, from 240MB to 6.9MB, and on VGG-16 by 49× from 552MB to 11.3MB, with no loss of accuracy with pruning reduced the number of connections by 9× to 13× and quantization reduced the number of bits representation for each connection from 32bits to 5 bits. The size reduction of the model resulted in its possibility of implementation on SRAM which requires 120× fewer energy for computation rather than DRAM memory. In the last few years, different algorithms have been proposed for deep neural compression based on different methodologies.

To replace conventional techniques of modifying the architecture manually or using pre-defined heuristic, [2] proposed policy-gradient RL for Network to Network Compression which compressed teacher model in two stages (Layer removal & Layer shrinkage). Removing layers aggressively reduced the size of the architecture but required heavy training & large computation in second stage for fine tuning the model. Yihui He and Song Han [10] proposed ADC: Automated Deep Compression and Acceleration with Reinforcement Learning for channel pruning. Work focused on layer wise channel pruning and did not study channel dependencies. An output channel is only dependent on input channels in the same group, and we can have convolution kernels and channel dependency as well. So channel pruning without studying their dependencies might affect other channels and kernels. Once a channel is pruned, it cannot be recovered so the study of channel dependencies is important before pruning it.

As discussed earlier, once we trained the model for few epochs, some kernels may have less significance than it was at the beginning of training. So the importance of individual kernel varies on different stages of training. And mostly, when the kernels are used to train model for sufficient number of epochs, turning off the kernels would have minimal impact than if they were turned off at early stages of training. To address this problem, we used epoch-based reward (E_{reward}). When the model proceeds toward higher number of epochs, negative reward is given to the model as a punishment, then model will learn to converge sooner with the predefined constrained. This epoch-based reward helps the model to converge sooner and if model takes long amount of training, DQN force the model to prune more kernels at latter part of the training. Pruning kernels in latter part of training is safe because most of the weights have been trained well. In this way, DQN force the model to accelerate.

$E_{reward} = \{-1 \text{ if } \#epochs \uparrow\}$, $score_e = score_e + E_{reward}$ where, $score_e$ is accumulated score at episode e.

For resurrections of kernels, the model will learn the importance of that particular kernel before it was dropped out or deactivated and assign the fit value. During the resurrection process, the most fit (healthiest) kernel/s among deactivated ones will be reactivated and initialized. There could be a chance of them getting turned back off again immediately for newly reactivated kernels, so we modified weights for reactivated kernel/s after its initialization.

$\boldsymbol{w_{lk}} = \boldsymbol{init}(\boldsymbol{w_{lk}}) * \ |\boldsymbol{w_{lk}}|$ ....eq i Modified weight

2.1. ACCURACY-ENSURED COMPRESSION (AEC)

We deployed Deep Q Network to learn the importance of kernels in reduced VGG-16 like models and compressed the model by turning off/on kernels. We were successfully able to turn-off up to 73% of the kernels with increasing the accuracy and also able to accelerate the model by 1.5 times which resulted in faster convergence. As the training elongated, DQN will enforce the model to deactivate more kernels because most of the weights are already learned, which will lead to increase compression ratio considering the accuracy. If the accuracy drops, instead of using iterative based training used by many other researches, this proposed model used a naive concept of reactivating inactive kernels to fine tune the model. We are making sure that the model will select kernel/s with higher fit (F_{lk} – fit value for k's kernel in l-layer) value. While turning off kernels, the model will evaluate all those kernels and assign fit value for each deactivated kernel based on their significance. So, during the re-activation cycle model will select prominent kernel/s so we can fine tune the model with the fewer number of kernel/s reactivation.

Algorithm 1: Finding the best policy based on Accuracy Ensured Compression

```
Goal: Find the optimal policy
Target policies: Accuracy (= | ↑) and Compression rate (↑↑↑)
Candidate optimal policies = [Target policy and fast convergence]
For every episode:
    1) Initialize observation (start state)
```

– $S_0 \epsilon (l_{11}, l_{12}, l_{13} \ldots \ldots l_{21} l_{22} \ldots \ldots l_{n1} l_{n1} .. l_{nm})$, l1 norm / Manhattan Distance (l), number of layers n and list of kernels each layer m $\epsilon [m_1, m_2, \ldots.. m_n]$

2) Exploration:

   a. Taking random actions from action space $A_s \epsilon (a_1 \ldots. a_5)$:

   b. Observe kernel's significance & apply actions

   c. Observe accuracy & compression changes

   d. Save states on Replay memory

      R = [S_t, a_t, r_t, S_{t+1}, acc, compress ………..]

   e. Explore few iterations at the beginning of training

   Exploitation:

   f. Learn and choose the best action from replay memory [Using past experience from R]

   Q_{target} = reward $_{batch}$ + $\boldsymbol{\gamma}$ * max(Q_{next_state}).............eq ii. Target Q value

   g. Random explore to make the model stochastic

3) Repeat until current state (S_t) is the final state (S_f) or max number of training (exploration)

2.2. COMPRESSION-ENSURED CONSIDERING ACCURACY (CECA)

The second proposed model is CECA, which compresses the big model by maintaining original accuracy in most of the cases, but a negligible drop of accuracy is acceptable up to -0.5% loss if the compression rate is relatively high, in other words able to generate high weight sparsity in the network. The sparser the network is, the less training and testing time is required for the network and also will fit the model in small memory and computation constrained devices like smart phone, smart watch, smart car, drone, surveillance camera, and even in the cloud. The purpose of this model is to compress the model by only preserving the minimal required number of kernels in convolutional layers of CNN. We have used the same algorithm as we did use for the accuracy ensured compression model with a change in Target policy.

Target policy for CECA model = Compression rate (↑↑↑↑) and Accuracy (= | ↑↑↑ | ↓)

We have successfully obtained a model with a high compression rate, an increase in accuracy and a convergence rate. The model achieved state of the art performance by compressing big ratios for both the number of kernel's required in each layer and the number of pruned/inactive parameters in the network.

3. METHODOLOGY

Figure **1** illustrates the proposed Automated DQN based Deep Compression model. We aim to automatically find the importance of kernels in each layer of the network to improve the quality of compression algorithms. Motivated from [15] kernel-wise variational pruning to remove unimportant kernels from the input channel dimensions, we propose to search for unimportant kernels based on accuracy, compression ratio & convergence rate and deactivate kernels instead of pruning so that we could recover the kernels (turn- on /activate) if required.

The goal of the proposed methodology is to improve the quality of compression in terms of accuracy, compression ratio and convergence rate. Accuracy is the test accuracy performed after training; compression rate is defined in terms of kernels dropout [turned off kernels] which will result in more inactive parameters. Convergence rate is the total number of epochs required to train the model to reach target policy. Considering these three factors, the goal of proposed DQN model to two different findings:

1. Appropriate number of kernels in each convolution layer
2. Appropriate time to turned off / turn on kernels

Simply learning which kernels to turn-off and when (at what training stages) to turn-off. To achieve appropriate number of kernels and appropriate time for any kernels to turn-off, actual human learning approach is implemented. Depicting into the model how human progress through learning, how human can skip redundant and unimportant information. For example, human is asked to read a book for 100 times, human may read thoroughly for the first time but after second attempt human speed up learning by skipping unnecessary details, redundant information and also some small information may be learned within a few readings. This concept is translated into machine by turning off kernels from convolution layers. But what if some really important information is missed while skipping learning? To cover this, human can reread it. This idea is depicted in the machine by turning on kernels which are already turned off so machine can learn missed information on few epochs of training

3.1. PROBLEM DEFINITION

Model compression is achieved by deactivating the number of kernels from k to k'.

k = [1,1,1,1,1,1,1... 1], k' = [0,0,0,1,1,0,1,0…0] for n kernels, 1 is active kernels and 0 is inactive kernels. k' is sparsity of kernels. The sparsity ratio can radically affect the compressed model's performance, so we propose to find sparsity ratios in fine-grained with reinforcement learning.

Observation is a series of l1 norms, $S_0 \epsilon (l_{11}, l_{12}, l_{13} \ldots \ldots l_{21} l_{22} \ldots \ldots l_{n1} l_{n1} .. l_{nm})$, proposed reinforcement model always starts with same initial state values computed by l1 norm on kernel's weight. [13] We used l1 norms because of it has a lower computation cost as compared to l2 norm. Successive states are determined based on action values and random exploration. When the model learned from first batch of observations, it will exploit more from past experience and explore less often in order to deal with changing states and makes model stochastic. Two different experiments are done with different size of experience 64 and 150. This numbers suggest that 64 past experience was used to exploit the learning for Deep Q Network. It's better to have more experience to train the model so model learns better.

Two run time memories for storing accuracy and compression ratio for n episodes are $A_{cc} = \{a_{original}, a_{e1}, a_{e2}, a_{e3}, \dots, a_{en}\}$, $C_{ratio} = \{0, c_{e1}, c_{e2}, c_{e3}, \dots, c_{en}$, and $A_s \epsilon (a_1 \dots a_5)$ action space.

3.2. PROBLEM FORMULATION

Action: An action is an operation to be performed in the big model based on exploration and exploitation. Exploration is done by picking a random action from actions available in action space (A_s), whereas, an exploitation is done based on experience (selecting the best action so far based on sequence of observation, action, reward and new observation from replay memory). An action can be exploration or exploitation.

Action Space (A_s): All the available actions are stored in action space, so that Deep Q Network performs one of these actions at a time). Proposed model has action space A_s = [0, 1, 2, 3, 4] having five actions.

Table 1: Action table for DQN Model

Action No.	Actions	Action based on
0	Turn off (3%) kernels	kernels with minimum l1 norm
1	Turn off (5%) kernels	kernels with minimum l1 norm
2	Turn on (5%) kernels	kernels with higher fit value
3	Turn on (10%) kernels	kernels with higher fit value
4	Epoch-based, Turn off (3%) kernels	kernels with minimum l1 norm
	Epoch-based, Turn off (5%) kernels	kernels with minimum l1 norm

State Space (S_s): All the states/ observations are stored in the State Space, i.e. the possible observations. In our case, the state space is continuous.

Episode: An episode is a sequence of observations/states that start with the initial state to the final state. DQN model learns better as the number of episodes progresses. The number of states in each episode can be different which is known as step-size in Q-learning. The goal is to minimize the step size while reaching the terminal state.

Step-size (Convergence rate): The total number of epochs required for training the model in a particular episode is considered as a step-size in the proposed model. With the requirement of a minimal number of epochs in training will increase the training efficiency and require less training time.

Reward: A reward is what we give feedback to the model on its action on particular observation. A positive reward will motivate the model, whereas, a negative reward will deviate the model to take another action. In the proposed CECA model, the goal is to compress the model as much as we can. Therefore, the reward for turning off kernels is +1, whereas the reward for turning on kernel is -1. We have further classified rewards below:

a. Activation/Deactivation based rewards: The rewards for turning-off and turning-on kernels.
b. Epoch/Step-size based rewards: The rewards given based on current step-size. The goal is to reduce the number of epochs/step-size so as the

number of epochs increases, we give negative rewards to the model to push the model for early convergence with a quality result in terms of accuracy & compression.

Table 2: Reward table for DQN Model

Action value	Reward value
0	+1
1	+1
2	-1 [Punish]
3	-1 [Punish]
4	+1
Epoch based reward	Reward value
Early convergence	+10
Accuracy (= or >)	+20
Compression (>)	+10
Full epoch of training	-2 on each epoch [Punish]

The goal is to compress the CNN Model, so a positive reward (+1) is given when the model chose the actions that resulted in a compressed model. But it does not mean that compressed model is always good if the accuracy is dropped too much. So, to maintain the accuracy a good reward (+20) is given to the model so it will try to maintain the original accuracy or better accuracy. Similarly, obtaining a more compressed model balancing the accuracy is most important, so, for compression a reward (+10) is given. To obtain the optimal result with fast convergence, a negative reward/penalty (-2) is given as model training progress further.

Exploration: At the beginning of learning for the DQN model, we forced the model to explore until the replay memory consisted of the total number of sequences as the batch size. The other way to explore was controlled using an Epsilon value in the model. The exploration happened a lot at the beginning with some occasional exploration throughout the learning. The purpose of the exploration in the proposed model was to address the following issues:

a. The problem of Q-learning is over-exploitation, narrow the search space so does not deal well with changing observations.
b. To prevent model from premature convergence i.e. we do not want the model to get converged with bad performance (accuracy & compression).

Exploitation: Learning from past experience. We have used Replay Memory that can memorize past experience so that the model can re-use those experiences to learn in a more efficient way. We have conducted experiments with different sizes of Replay memory [64, 150]. The bigger the size of replay memory, the model has more past experience to learn from therefore the model can learn quicker and better.

3.3. MODEL STRUCTURES

Table 3: DQN Model Structure

Layer #	Layer type	Activation function	No. of inputs	No. of outputs
1	Linear / FC	ReLU	224	256
2	Linear / FC	ReLU	256	256
3	Linear / FC	softmax	256	5 (actions)

Table 4: Tested CNN Model

Index: [- : The layer does not have that feature]

Layer #	Layer type	Activation Function	No. of inputs	No. of outputs	Kernel size/padding/stride
1	Conv2D	ReLU	1	32	3 / 1 / 1
2	MaxPool2D	-	-	-	2 / - / 1
3	Conv2D	ReLU	32	64	3 / 1 / 1
4	MaxPool2D	-	-	-	2 / - / 1
5	Conv2D	ReLU	64	128	3 / 1 / 1
6	FC	ReLU	7 * 7 * 128	1024	-
7	FC	Softmax	1024	10	-

4. EXPERIMENTS

In all experiments, the MNIST [3] dataset contains 70000 grayscale images which is divided by 60000 samples for training, 10000 samples for evaluation. We adopt a VGG like network as a base model with 3 convolution layers and 2 dense layers. The first 2 convolution layers used max pool operations for down-sampling with stride 2 and kernel size 3×3. We used ReLU (Rectified Linear Unit) as the activation function.

$f(x) = \max(0, x)$ ....eq iii ReLU Activation Function [1].

For Deep Q Network, we used 3 linear layers and ReLU [1] as the activation function except last layer. The discounting factor is selected as $\gamma = 0.99$ and epsilon as $\epsilon = 1.0$ and final epsilon as 0.01 with a decay of 5e-2. Replay memory of size 64. The learning rate for Deep Q Network is 0.003 and Adam is used as an optimizer. The loss function used for training is Cross Entropy Loss. All implementation has been done on a top python-based machine learning framework called 'PyTorch', developed by Facebook AI team.

4.1. ACCURACY-ENSURED COMPRESSION

The Figure 2 Network is trained for 50 Epochs and 50 Episodes for accuracy constrained compression. Each episode begins with the same initial weight as the original model used for its training. The result shows that for compressing the original model using proposed Auto-AEC, 80% of kernels can be turned-off without dropping the accuracy or with a slight increase in accuracy, whereas, with 0.5% loss of accuracy, 84% of kernels can be turned-off. With the compression rate more than 55%, the network converged early on 80% of trainings.

Figure **1** shows that the proposed Auto-AEC model learnt well with accumulating rewards and reducing number of steps to achieve the target. Fluctuating curve for step size means randomly at certain iterations we forced the model to explore and learn so the model trained for full epoch of 50 for episode 20, 29, 37, 44, 50.

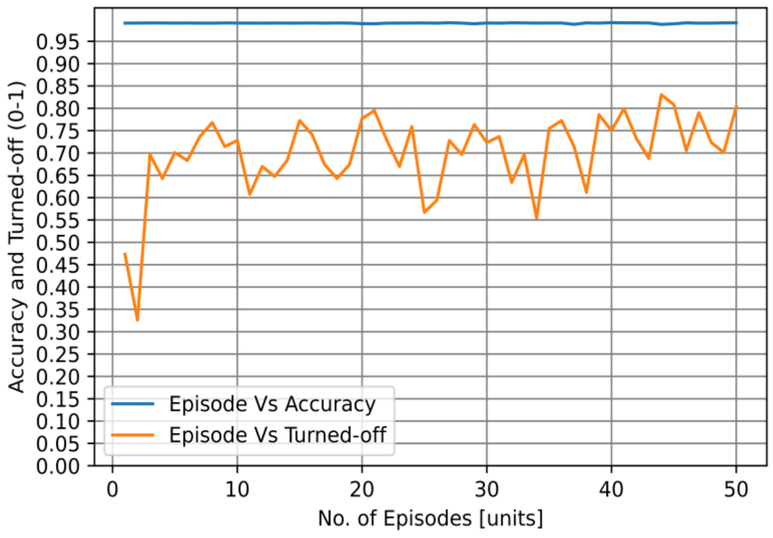


Figure 2: Accuracy and Compression rate/turned off rate 50epochs/50episodes

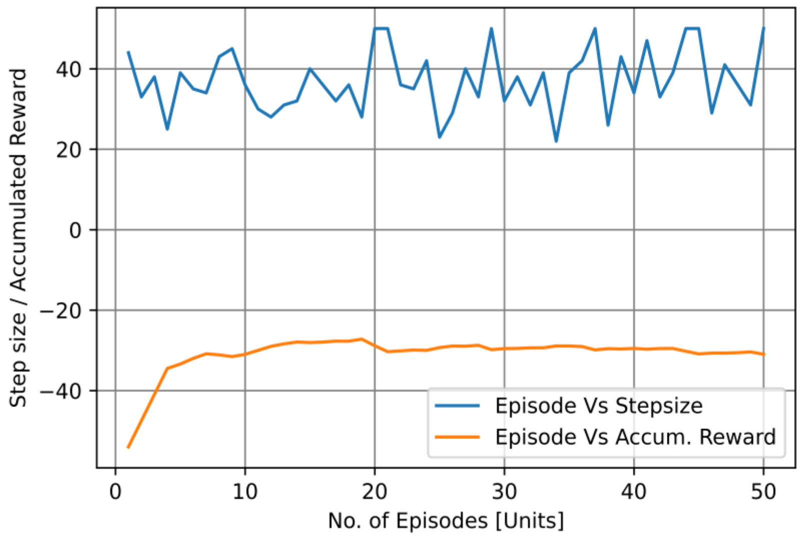


Figure 3: Accumulated rewards and Convergence rate 50epochs/50episode

Table 5: Average accuracy & convergence rate variance with compression rate (kernel turned-off %)

Compression rate (%)	Accuracy changes (%)	Converge rate (times)
50-59 %	+ 0.5	+2.3
60-69%	+ 0.08	+ 1.7
70-75%	+ 0.15	+ 1.6
75-80%	+0.12	+ 1.25
>80%	- 0.40	+ 1.15

The optimal policy [based on accuracy increase] leant by the proposed model for the experiment above is (i) in Table **5** with an increase in accuracy, compressed kernels to 50-60% and an increase in convergence rate by 1.9 × to 2.5×. Similarly, the optimal policy [based on compression rate] is (iii) in Table **5** with increase in accuracy, compressed kernels to 75% and increase in convergence rate by 1.6 ×.

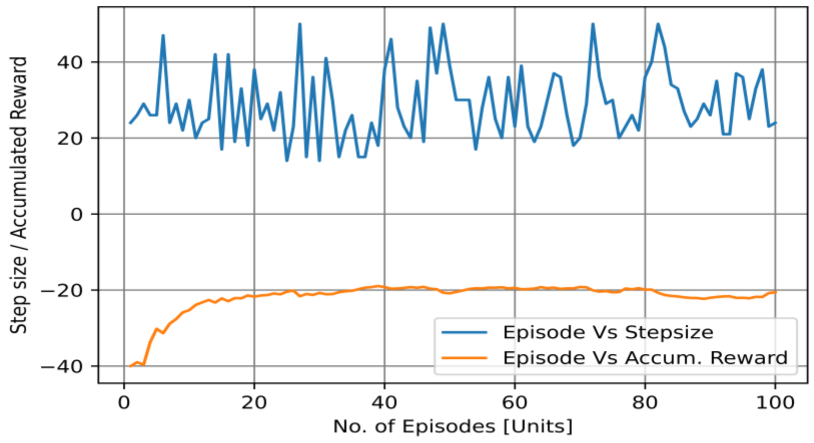


Figure 4: Accumulated reward and convergence rate for 50 epochs 100 episodes of training AEC

While ensuring the accuracy, the compressed model can converge even more than two times faster.

4.2. COMPRESSION-ENSURED CONSIDERING THE ACCURACY

Based on the test conducted with 50 epochs, 50 episodes compression ratio can be guaranteed more than 50% with an increase in accuracy up to 1% or a loss of accuracy less than 0.05% and convergence rate raised to 2.5×. A smaller than half sized model achieved the accuracy milestone within 20 epochs of training.

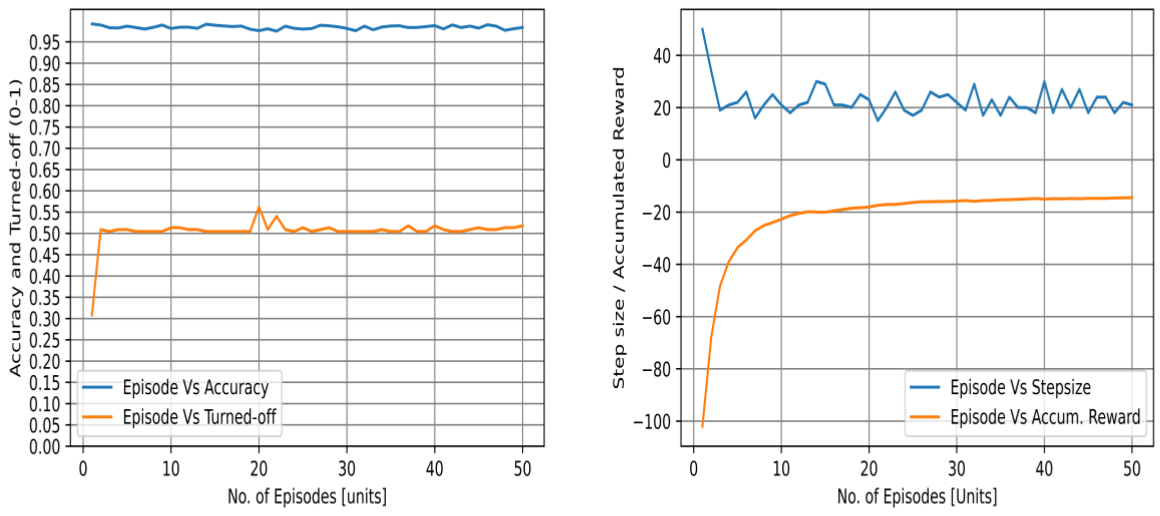


Figure 5: Accuracy Vs. Compressed Rate and Accumulated Rewards Vs. Convergence Rate 50 epochs 50 episodes

With another experiment with 100 episodes of training we have obtained the following results. The results show that the performance of the model is increasing as training episodes increases.

As the episodes progress, the model learned well and converged very well with optimal results (both accuracy & compression rate).

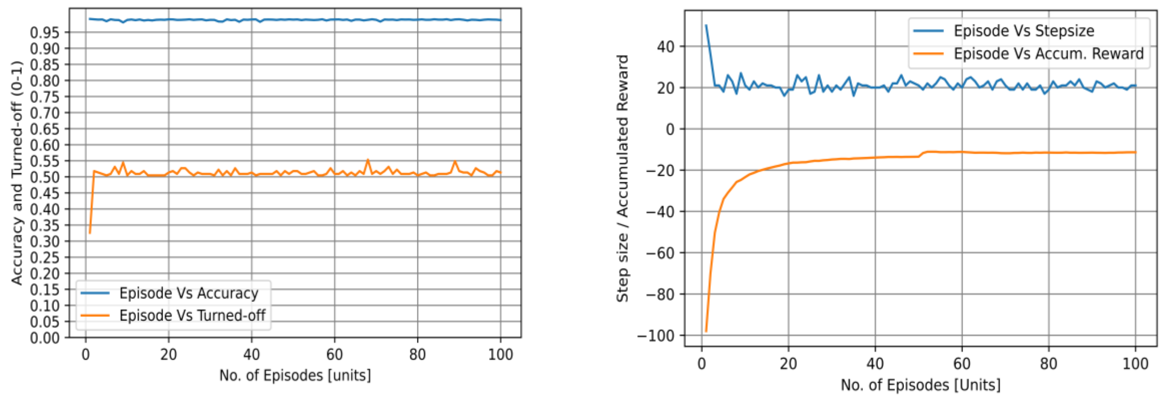


Figure 6: Accuracy Vs. compressed rate and accumulated rewards Vs. convergence rate 50 epochs 100 episodes

While exploring CECA training of 100 episodes, the model explores until 17 episodes and then starts exploiting often and exploring less often until 50 episodes. After 50 episodes, the model leant very well and starts converging sooner. The proposed model can find a good model very quickly. The state-of-the-art results of compression of 80% kernels (86% parameters) with increase in accuracy by 0.14% and compression of 84% kernels (94% parameters) with the drop of minimal test accuracy 0.4% is shown below.

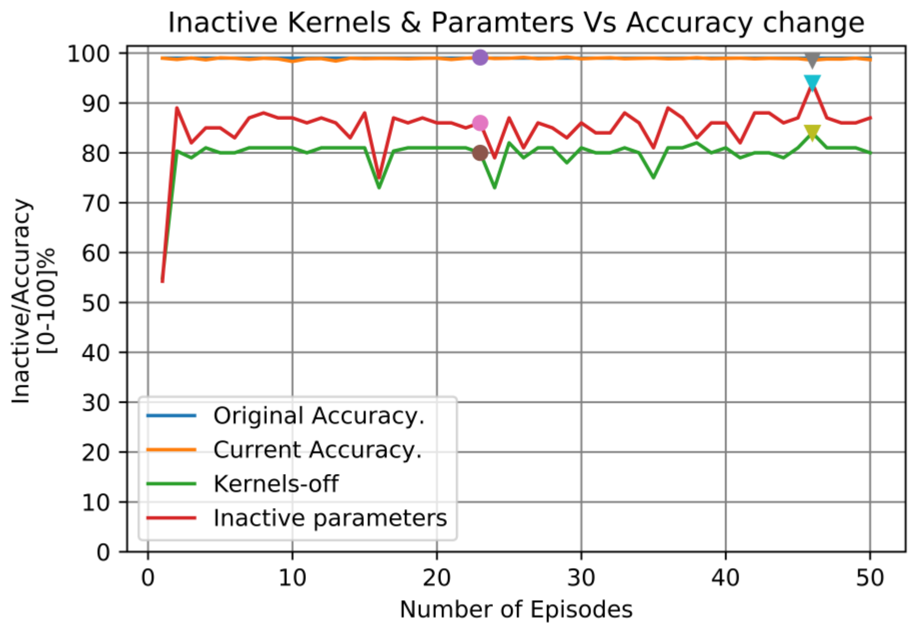


Figure 7: Inactive kernels and parameters versus accuracy change

Dropping off 80% of kernels resulting 86% inactive/sparse parameters from different convolution layers with improve on accuracy +0.14%. As experiment conducted on 3-layer Convolution Neural Network with kernels (32, 64 and 128) respectively starting from first layer, we can make conclusion that kernels at latter layer have more significance on performance factor of Convolution Neural Network for both accuracy and compression performance. Figure **8** below shows 64% of parameters are pruned from the first layer, whereas, 68% parameters are inactive in the second layer and the highest percentage 91% parameters are pruned from the third convolution layer. Out of 128 kernels in the third layer, DQN just retained 12 kernels as active kernel to perform classification and outperformed on both accuracy and compression performance.

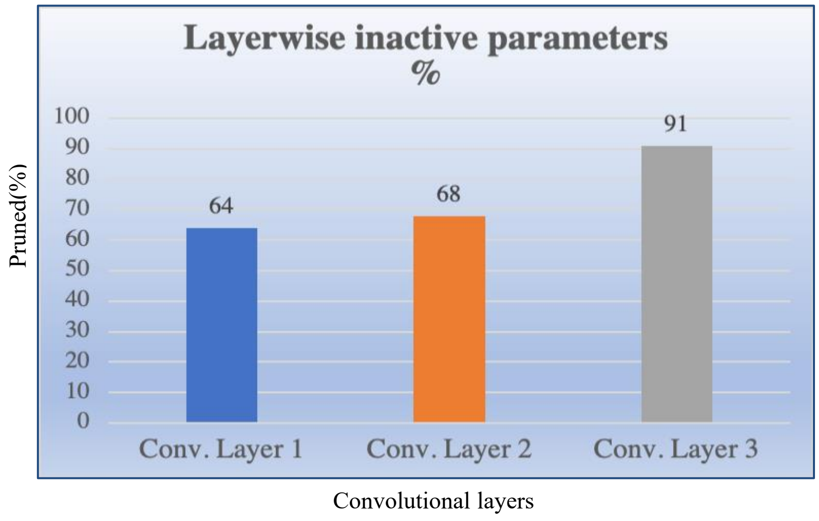


Figure 8: Layer wise percentage of inactive parameters for CNN

To ensure the consistency of our model's performance, we have conducted a test combining both previous model's policies. The Accuracy-Ensured Model and the Compression-Ensured Model considering Accuracy are merged to see the performance differences. We have conducted an experiment for 50 epochs and 70 episodes of training with a slight adjustment on size of replay memory [150]. At total of 150 best action batches are stored in replay memory which means with having more experience, the model learns better. The results are shown below.

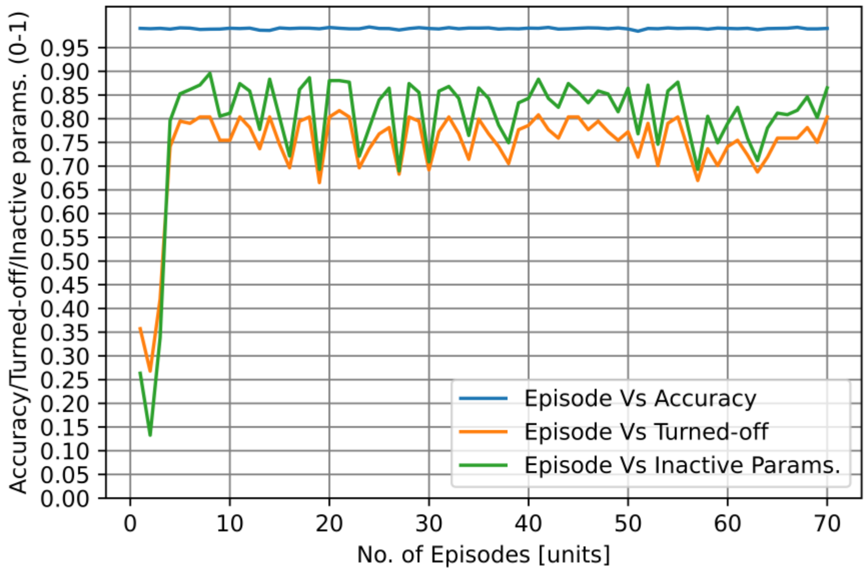


Figure 9: Episode Vs. accuracy and layer-wise turned-off kernels, and inactive parameters

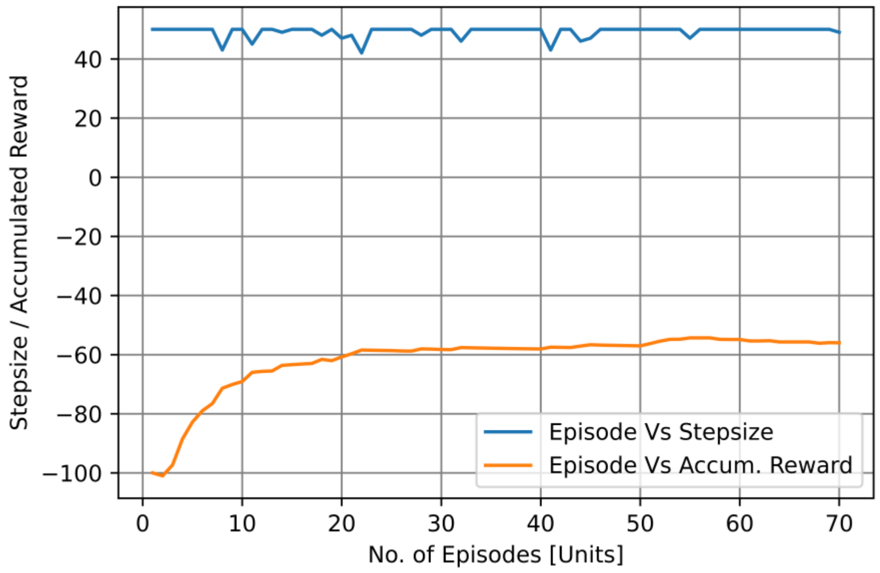


Figure 10: Episode Vs. Step-size and Accumulated Reward for 150 sized replay memory

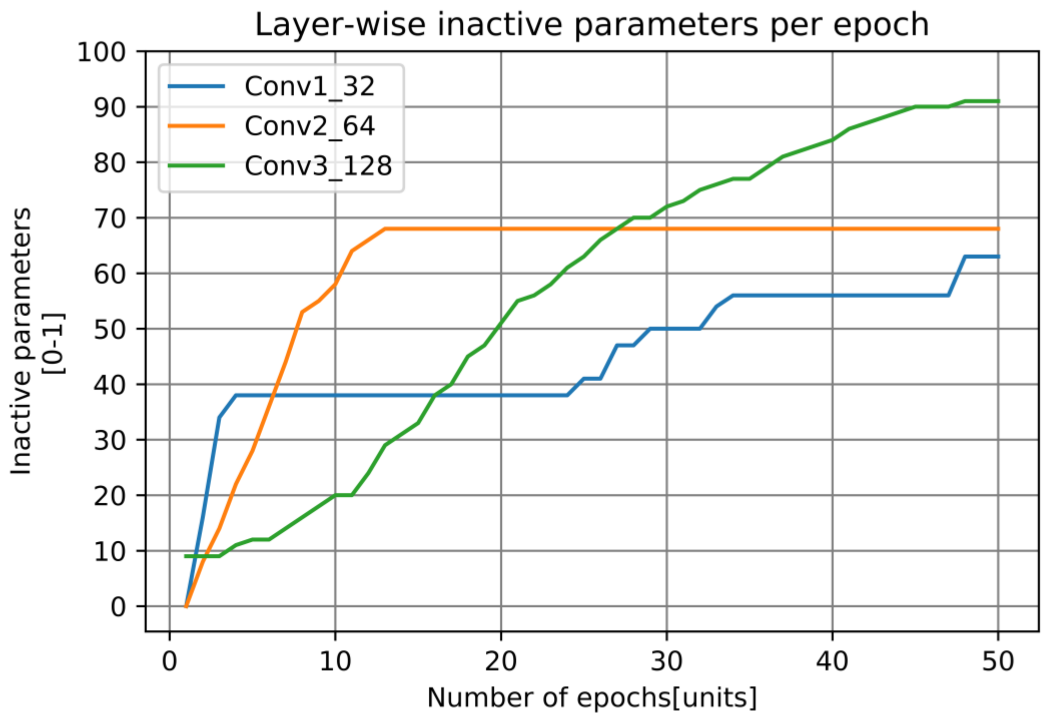


Figure 11: Layer-wise inactive parameters per epoch [average of last five episodes]

By observing Figure **11**, regarding both accuracy and compression, layer 3 kernels have a significant role. As layer 3 has 64 input channels and 128 output channels, turning off a single kernel in layer 3 will result in (64 × kernel-size) parameters to be zero. When you have a target policy to compress high, the DQN model will select kernels from layer 3 at most but turning off many kernels at this layer would cause a significant drop in accuracy so the model has to sensibly turn off kernels in different convolution layers. As results show, layer 3 kernels are sensibly turned off in a linear way as the number of training epochs increase with balancing kernels from previous layers [turning off or saturate]. After 13 epochs of training, kernels turned off for layer 1 is saturated at 68% but we can see a variance in kernels turned off rate for layer 2. From this experiment we still found state-of-art result compressing convolution layers by 89% (overall 80% of kernels are turned off) with accuracy increase in +0.31%.

4.3. LAYER-WISE KERNEL’S EFFECTIVENESS ANALYSIS

According to the proposed DQN model, while analyzing the importance of kernel’s on each layer, on different stages of training on both experiments, we found that kernel’s at earlier convolution layers are safe to be turned off. But to obtain more compressed network, kernel’s at latter parts of convolution layers are turned off. Maintaining the balance between accuracy and compression rate, the proposed model determines which layer’s kernels should be turned off or turned on at what stage of training. For an instance, we have 3 convolution layers, for most of the episodes of training the model turned off kernels from first & second convolution layers considering accuracy. But when we trained the model based on experiment 2, trying to increase the compression rate, the model turned off kernels from the third layer. For regaining accuracy, the most important kernels from layer three are turned back on.

For Accuracy Ensured Compression (AEC) training, kernels at the earlier layers are turned off whereas, for Compression Ensured Considering the Accuracy (CECA) training, kernels at latter layers are turned off. In both the training, the proposed model gained high compression rate with an increase in accuracy or negligible drop on accuracy in average or less than a half percentage for model compressed ratio greater than 80%.

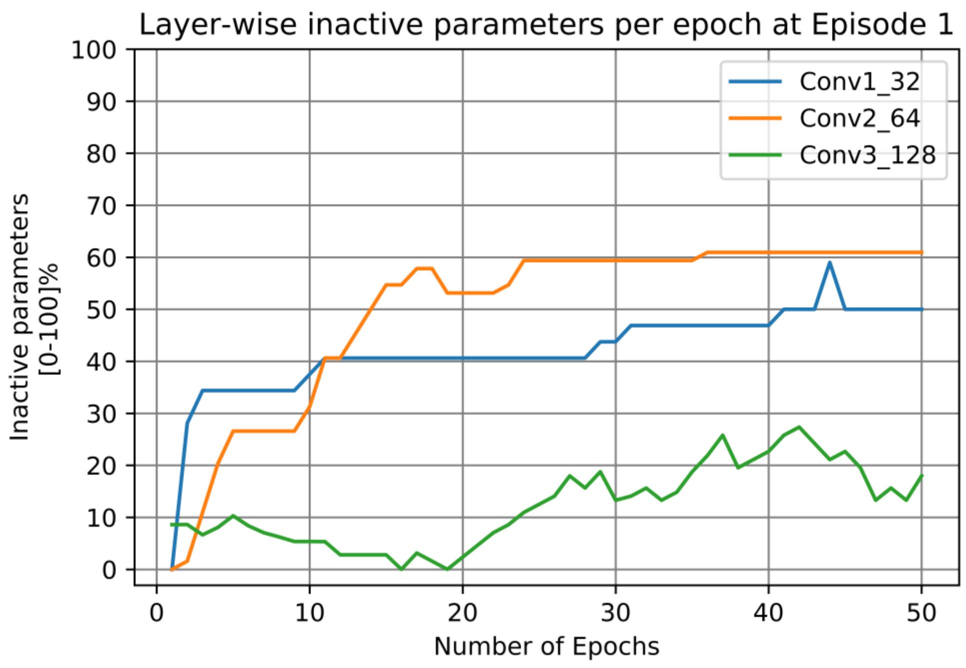


Figure 12: Inactive parameters at episode 1

Convolution layer 1 has 32 output channels, layer 2 has 64 output channels and layer 3 has 128 output channels. Turning off kernels at the first layer results in less sparse network, whereas, turning off kernels from layer 3 results in sparser network (high compression rate). Kernels at layer 3 have a connection with more parameters so turning off kernels at layer 3 is highly sensitive as compared to previous layers regarding both accuracy and compression rate. Turning off kernels from layer 3 will result in high compression but on the other hand, the accuracy might get affected when more weights are pruned at once. So the goal of model is to learn the significance of kernels at each layer and at different stages of training such that how many and when the kernels should be turned off/turned on. Turning off will definitely result in compression network but that is not the primary issue if we lose accuracy by a large margin, so it is important to know when to turn off the kernels.

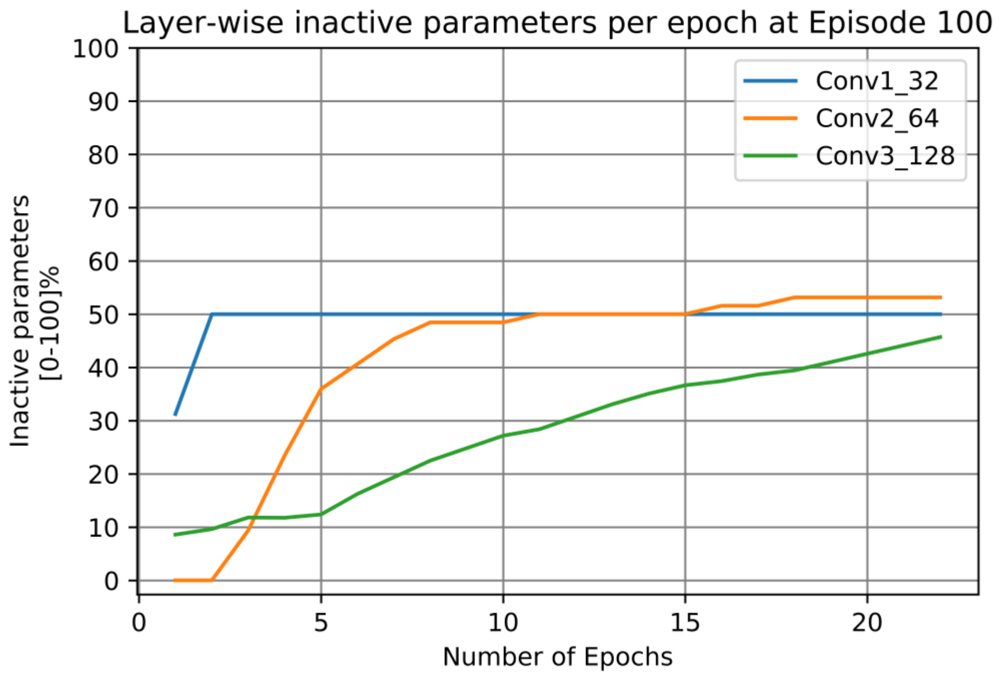
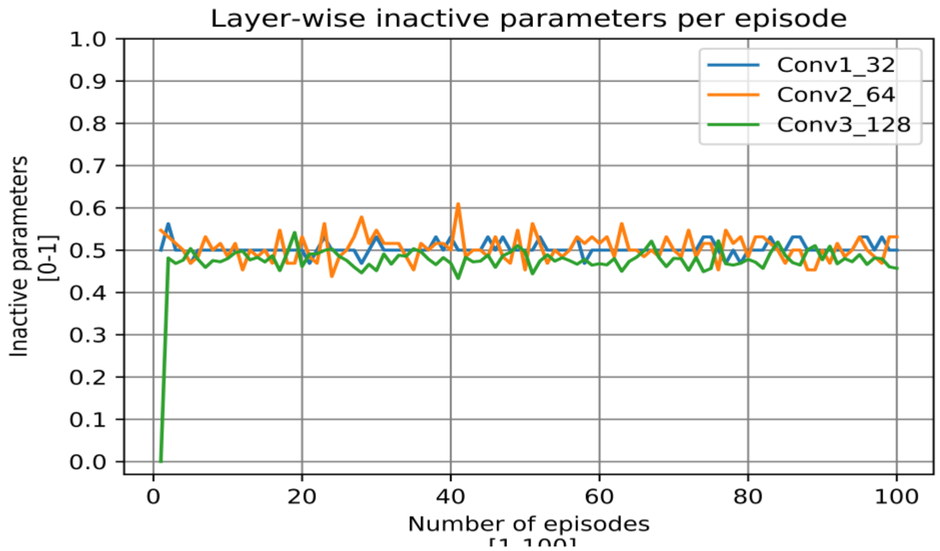
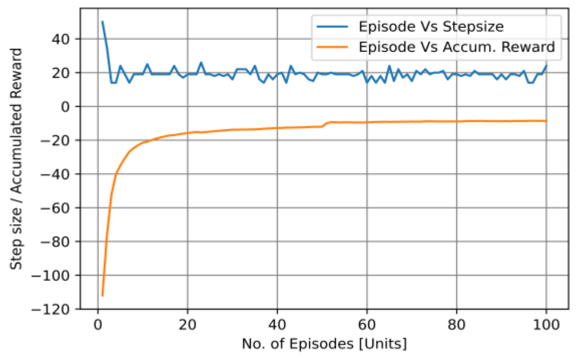


Figure 13: Inactive parameters at episode 100

Figure **12** is the result obtained at the beginning of learning when applying actions of turning off and turning on kernels to maintain accuracy, compression ratio and convergence rate. As the learning progresses, Figure **13** describes the model learnt importance of various kernels at different convolution layers. Layer 3 with 128 output channels, kernels at that layer play the most significant role to maintain both accuracy & compression ratio. So, the model cautiously deactivating kernels as epochs of training progress considering accuracy and compression ratio also results in fast convergence. Figure 7(b) shows that Accuracy-Ensured Compression training is able to compress (>50%) with few epochs of training just (23 epochs) as compared to 50 epochs of training for original model. After training for several episodes, the model has learnt kernel's efficiency and as the epochs progress the model force to converge early by deactivating kernel’s that lead to increase compression ratio with guaranteed accuracy increase.

Figure **13** shows that turning off kernels from the first layer is saturated after 2 epochs of training, similary, turning off kernels from second layer is increasing as training progress until 11 epoch and saturated for a while with balancing kernels turning off from third convolution layer until 23 epochs of training.





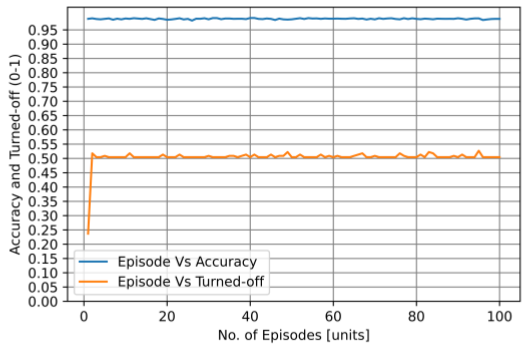


Figure 14: Layer wise performance analysis and respective accuracy, compression ratio and convergence rate

4.4. RESULTS COMPARISON

The accuracy of the small model is lagging in all three cases for (44 , 40, 43)% kernels turned-off percentage by -0.04%, -0.12%, -0.17% respectively with compared to big model. The accuracy of the model using DQN model is higher as compared to the original model average accuracy for running 50 different testing (+0.09%, +0.15%, +0.13%) respectively. Comparing the model with DQN outperforms in terms of accuracy with smaller model as well by (+0.09%, +0.16%, +0.14%) respectively. The DQN compressed model outperforms in terms of accuracy, compression ratio & convergence rate compared to the original model, and in terms of accuracy compared to new designed small model. Therefore, the compressed network gererated by DQN model performs better than the equal sized small network.

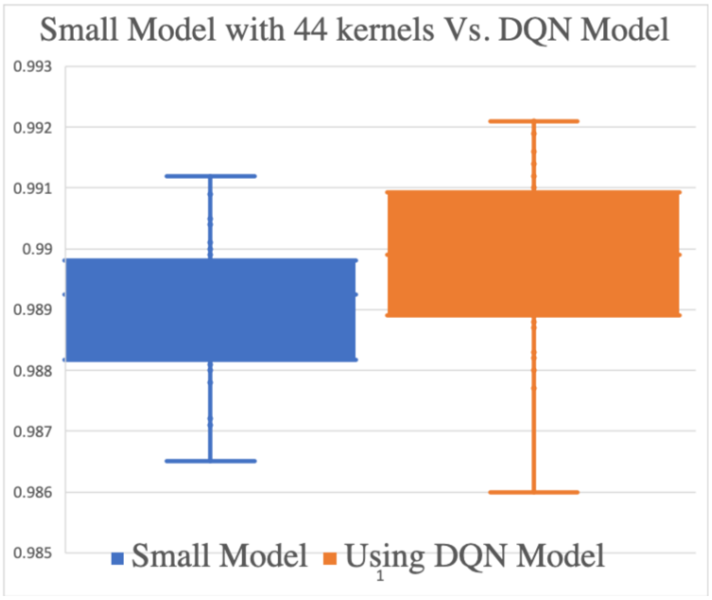


Figure 15: Accuracy graph for small model with 44 kernels Vs. model with DQN model

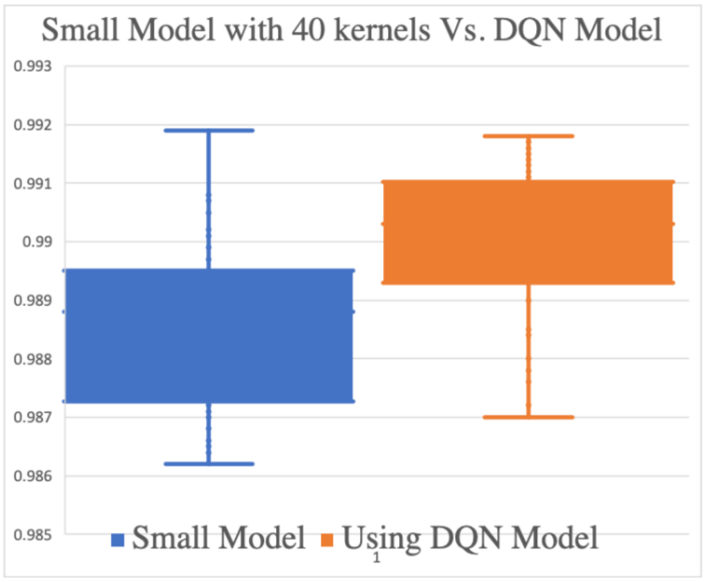


Figure 16: Accuracy graph for small model with 40 kernels Vs. model with DQN model

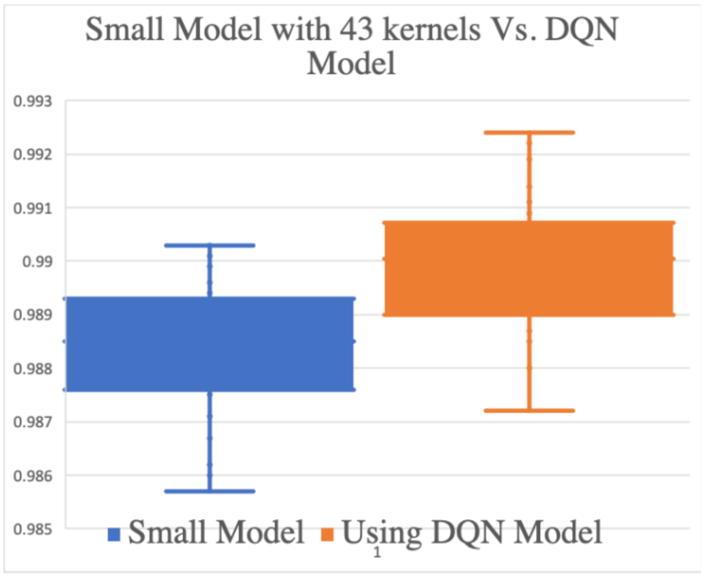


Figure 17: Accuracy graph for small model with 43 kernels Vs. model with DQN model

To find the appropriately sized small model for the given problem a brute force approach is tedious and time consuming. The DQN model helped to find the small sized model for the given problem so that we can construct a small model base on the number of active kernels remaining in DQN compressed model. The results showed that the performance of the DQN compressed model surpassed in every aspect to the original model and small sized model. Training for 50 different randomly initialized weights, DQN based compressed model showed consistence performance with gain in test accuracy.

Table 6: Overall kernels and parameters comparison

Model	Convolution Layers			
	Active parameters (#)	Kernels		Inactive parameters (# / %)
		Inactive	Active	
Original Model	92448	0 (0%)	224	0 (0%)
DQN compressed Model (Better accuracy)- AEC	12943	179 (80%)	45	79505 (86%)
DQN compressed Model (Better compression)- CECA	5547	188 (84%)	36	86901 (94%)

5. CONCLUSION

The proposed DQN model, an automated deep reinforcement-learning based model can effectively turn off kernels on each layer by observing the kernel's significance on decision making. Unimportant kernels are deactivated and assigned a fit value based upon its current importance. We addressed the problem of traditional pruning approaches which required more training in order to compensate pruned weights. The proposed model simply turned on kernels if required and immediately regained the accuracy with minimal training required. By observing accuracy, compression ratio and convergence rate, the DQN model can automatically renovate (turn on) the healthiest(fittest) kernels to train it again which greatly improves the model compression quality. Instead of doing premature pruning, the DQN model meticulously decides the appropriate time for kernels to deactivate without causing performance degradation. Furthermore, conducting experiment results on MNIST dataset, the proposed method can reduce the size of the convolution layers for VGG-like model up to 60% with 0.5% increase in test accuracy within less than a half the number of initial amount of training (speed-up up to 2.5×), state-of-the-art results of 80% kernels (86% parameters) with increase in accuracy by 0.02%. Further compression up to 84% kernels (94% parameters) with the drop of minimal test accuracy 0.4%. The DQN model successfully learned to find the appropriate number of kernels required in each convolution layers and what is the appropriate time to turn off a particular kernel. Two different models are proposed for model compression and acceleration, Auto-AEC (Accuracy-Ensured Compression) model can compress the network by preserving original accuracy or increase in accuracy of the model, whereas, Auto-CECA (Compression-Ensured Considering the Accuracy) model can compress to the maximum by preserving original accuracy or minimal drop of accuracy.

6. REFERENCES

[1] Abien Fred Agarap. 2018. Deep Learning using Rectified Linear Units (ReLU). 1 (2018), 2–8. Retrieved from http://arxiv.org/abs/1803.08375

[2] Anubhav Ashok, Nicholas Rhinehart, Kris M Kitani, and Fares Beainy. 2017. N2N LEARNING:NETWORK TO NETWORK COM- PRESSION VIA POLICY GRADIENT REINFORCEMENT LEARNING. (2017), 1–20.

[3] Hossein Baktash, Emanuele Natale, and Laurent Viennot. 2019. Compression. (2019), 1–18.

[4] Yu Cheng, Duo Wang, Pan Zhou, Tao Zhang, and Senior Member. 2019. A Survey of Model Compression and Acceleration for Deep Neural Networks. (2019), 1–10.

[5] François Chollet. 2017. Xception: Deep learning with depthwise separable convolutions. *Proc. - 30th IEEE Conf. Comput. Vis. Pattern Recognition, CVPR 2017* 2017-Janua, (2017), 1800–1807. DOI:https://doi.org/10.1109/CVPR.2017.195

[6] R Deepa. 2019. Image Classification and Text Extraction using Machine Learning. *2019 3rd Int. Conf. Electron. Commun. Aerosp. Technol.* (2019), 680–684.

[7] Li Deng. 2012. The MNIST database of handwritten digit images for machine learning research. *IEEE Signal Process. Mag.* 29, 6 (2012), 141–142. DOI:https://doi.org/10.1109/MSP.2012.2211477

[8] Song Han, Huizi Mao, and William J. Dally. 2016. DEEP COMPRESSION: COMPRESSING DEEP NEURAL NETWORKS WITH PRUNING, TRAINED QUANTIZATION AND HUFFMAN CODING. (2016), 1–14. Retrieved from

https://arxiv.org/abs/1510.00149

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep residual learning for image recognition. *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.* (2015), 770–778. DOI:https://doi.org/10.1109/CVPR.2016.90

[10] Yihui He and Song Han. 2018. ADC : Automated Deep Compression and Acceleration with Reinforcement Learning. (2018), 1–12. Retrieved from https://arxiv.org/abs/1802.03494

[11] Hinton E. Geoffrey Krizhevsky Alex, Sutskever Ilya. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *Neural Information Processing Systems (NIPS)*, 1–1432. DOI:https://doi.org/10.1201/9781420010749

[12] Andrew Lavin and Scott Gray. 2015. Fast Algorithms for Convolutional Neural Networks. (2015). Retrieved from https://arxiv.org/abs/1509.09308

[13] Andrew Y Ng. 2004. L1 and L2 regularisation comparisation. *Proc. 21 st Int. Conf. Mach. Learn.* (2004).

[14] Kwang Hee Won, Sisay Gurmu, and Soon Ki Jung. 2013. Pedestrian detection using labeled depth data. *FCV 2013 - Proc. 19th Korea-Japan Jt. Work. Front. Comput. Vis.* (2013), 117–120. DOI:https://doi.org/10.1109/FCV.2013.6485472

[15] Huixin Zhan and Yongcan Cao. 2019. Deep Model Compression via Deep Reinforcement Learning. (2019). Retrieved from https://arxiv.org/abs/1912.02254

[16] Zhong Qiu Zhao, Peng Zheng, Shou Tao Xu, and Xindong Wu. 2019. Object Detection with Deep Learning: A Review. *IEEE Trans. Neural Networks Learn. Syst.* 30, 11 (2019), 3212–3232. DOI:https://doi.org/10.1109/TNNLS.2018.287686