ISSN 2278-3091
Volume 13, No.2, March - April 2024
International Journal of Advanced Trends in Computer Science and Engineering
Available Online at http://www.warse.org/IJATCSE/static/pdf/file/ijatcse051322024.pdf
https://doi.org/10.30534/ijatcse/2024/051322024

Evaluation of the effects of K-Means Clustered-Based Weight Quantization on a Keras Library Based Convolutional Neural Network for Hand Written Digit Image Recognition

Roderick Yap[1], Goldwin Giron[2], Leonard Miguel Lanto[3], Lorenzo Garcia[4]
[1]De La Salle University, Manila, Philippines, roderick.yap@dlsu.edu.ph
[2] De La Salle University, Manila, Philippines, goldwin_giron@dlsu.edu.ph
[3] De La Salle University, Manila, Philippines, miguel_lanto@dlsu.edu.ph
[4]De La Salle University, Manila, Philippines, lorenzo_garcia@dlsu.edu.ph

Received Date : February 26, 2024 Accepted Date: March 22, 2024 Published Date: April 06, 2024

ABSTRACT

A lot of Convolutional Neural Networks (CNNs) have been implemented using FPGAs for the past years. Subsequently, memory saving features were added to the CNN through weight quantization using K-means clustering. A future goal on an ASIC design, involving CNN and weight quantization working together in one chip, can give way to an automated procedure of memory-saving CNN design. In this paper an evaluation was done on the effect of quantizing the weights of a Keras library-based CNN using K means clustering. Various values of K in K-means clustering were tested to see its effects on the CNN accuracy performance. This paper presents first the design approach of a Keras library based Convolutional Neural Network (CNN) for hand-written digit images. It then presents a hardware model design of K-Means clustering algorithm using VHDL. The performance of CNN for image recognition was then tested for various levels of weight quantization using K-means clustering algorithm. Simulation results showed a compression of weights as high as 60% resulted to less than 1% reduction in CNN's accuracy. The findings in this paper will serve as guide in determining the relevant values of K i.e. the compression ratio, for future ASIC design on this topic.

Key words: Convolutional Neural Network; K-Means Clustering; Hardware Model; VHDL; Weight Compression; Field Programmable Gate Array

1. INTRODUCTION

Convolutional Neural Network (CNN) is a popular machine learning tool used for recognizing images. It is popular in many research focused on various character recognition. Through CNN, computers can have the chance to recognize an image. In [1], Deep Neural Network, Deep belief Network and Convolutional Neural Network performances on handwritten digit recognition, are compared in terms of accuracy and performance. A handwritten digital recognition using CNN, was done in [2] and [3]. The paper made use of the MNIST database. In [4], Ensemble Learning was the focus for hand-written digit recognition. The paper mentioned CNN as one powerful algorithm for image detection and classification with high accuracy when compared to other machine learning tools. In [5], Java based Deep Learning, known as Deep Learning4j was used for recognition of handwritten digits from MNIST database. In [6], CNN was used for Bangla handwritten digit recognition. Fingertip written digit recognition using CNN, trained on MNIST data, was mentioned in [7]. The framework has hardware implementation using Verilog. KNN and CNN were compared for its performance on handwritten digit recognition in [8]. In [9], CNN was used for Malayalam digit recognition. While many research for CNN are done using the software approach that is on a computer, some would implement it on hardware like a Field Programmable gate Array (FPGA). For instance, [10] implemented a Deep CNN's hardware accelerator on a XILINX Kintex-7 FPGA. The paper proposed an optimized streaming method proposed for hardware accelerators on embedded platforms for a deep CNN. Data concatenation and data reused were adopted as means of increasing speed. CNN is used by [11] to implement a real time super-resolution system on an FPGA. The system highlighted processing moving images in real time.

Like the CNN, clustering is another tool in artificial intelligence that has found many applications. Some of these were focused on genes classification[12], fast determination of cluster center[13], cluster evolution analysis[14] and collaborative clustering[15]. CNN and Clustering crossed path in [16]. It was mentioned here that K-means clustering can be used for compressing the weights of the deep neural network.

By quantizing the weights to enforce weight sharing, usage of memory can be reduced.

2. SIGNIFICANCE OF THE STUDY

The findings in this research provide significant information needed for future researches on design and implementation of CNN using the hardware approach instead of software approach. The CNN adopted for this research makes use of MNIST data for handwritten digit recognition. Considering the usage of handwritten digit recognition in many real-life applications, the effect of various levels of weight compression can give idea to future designers on how much memory can be saved in many future hardware implementations. The Hardware approach design is good for embedded systems application. The findings in this research can guide the hardware designers on weight compression as well as determining the circuit size needed for its implementation. This is because the size of the weight compression circuit depends on the compression ratio. Using various values of compression ratio, one can search for the highest compression that will still maintain good accuracy on image recognition of the CNN. A lot of compression ratio values, however, can also lead to a large circuit size when considering hardware modeling. The findings in this research can serve as guide for designers on the limits of K values when implementing the hardware circuit for the clustering algorithm.

3. DESIGN CONSIDERATION

3.1 The CNN

Keras is an open source library used for building CNN. It is an Application Program Interface. It provides convenience in its accessibility for fast CNN implementation on a computer. In Keras, a user can implement a CNN using ready-made library modules also known as configurable building blocks. Commonly needed layers in implementing neural networks are made available to users. Keras supports not only neural networks but also convolutional neural networks and recurrent neural networks. Needless to say, writing a code for deep neural networks becomes easier through the use of available building blocks. The code in Keras is written using Python. Using the Keras library, a software algorithm for the CNN was implemented. Figure 1 shows the block diagram of the adopted CNN in this study. Two Convolutional Layers, two Pooling Layers, and two Fully Connected Layers were used to achieve the best accuracy while trying to reduce the number of weights to cluster. A study by [17] essentially provided the means for the basis of choosing the number of convolutional layers. The modelling of the design of the CNN model was also based on the same study and [18]. [17] provides the guide on the design of the pooling layer and remaining layers.

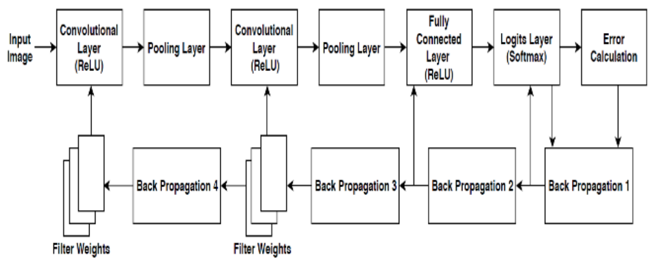

Figure 1: CNN Block Diagram

3.2 Input Image

The images used in training the CNN model were taken from the MNIST database which consists of a training set of 60,000 samples of handwritten digits, as inspired by a study by [19]. Using the “model.fit” command from the Keras library used in the constructing the CNN, the model will be able to train all 60,000 images. The speed of training is dependent on the processing power of the computer being used. Such processes were based on KERAS and NUMPY libraries. All the images in the MNIST database are of the size 28x28 pixels. The pixels in each image have a value ranging between 0-255 where 0 is the darkest and 255 is the lightest. For CNN to be able to process this data, the values are normalized into the range of 0-1 by dividing the values by 255, as per stated by a study conducted by [20]. The size of the input images was used in constructing the optimal CNN model consisting of minimal weights while still achieving the target accuracy.

3.3 Filter

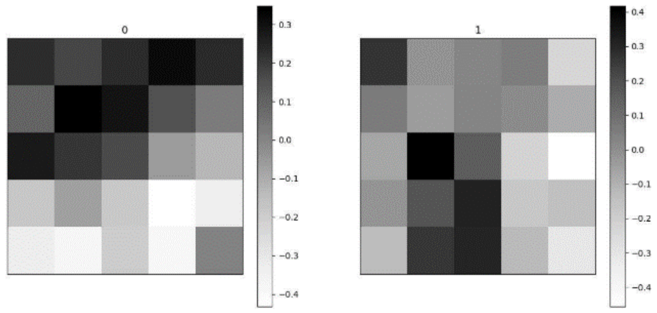

Figure 2 : Layer 1 Filter After Training

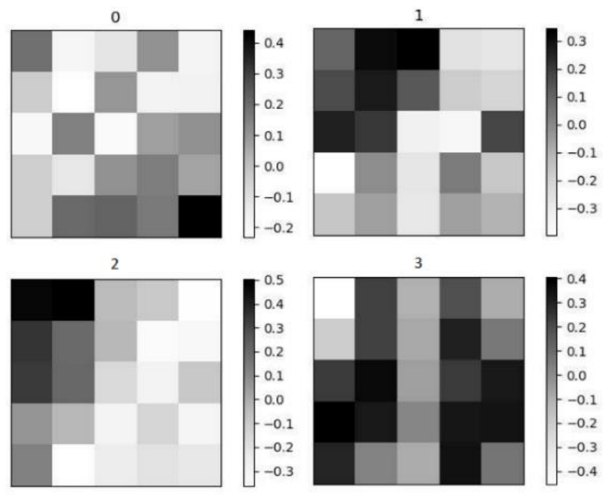

Figure 3 : Layer 2 Filter After Training

To maintain the lowest number of weights to be clustered, while being able to achieve the target, the Convolutional Neural Network was reduced as much as possible. 5x5 filters were used for the 1st and 2nd convolutional layers. Padding was utilized to maintain the output shape of the weights while still maintaining accurate results. Two 5 x 5 filters were used for the first convolution layer while four 5x5 filters were used for the second convolution layer. Figures 2 and 3 show the filter layer after the training.

3.4 Pooling

Two pooling layers were used to reduce the size of the image for it to be processed in the fully connected layers. By using pooling filters which have a size of 2x2 and applying it over strides of 2, the 28x28 image would be shrunk into 14x14 after the first pooling layer and 7x7 after the second pooling layer. This is done to minimize the number of nodes that will be connected to the Fully Connected Layer which would have drastically increase the number of weights present in the CNN.

3.5 Fully Connected Layer

Two Fully Connected layers are used in this design. The first Fully Connected Layer acts as a hidden layer in between the convolution layers and the output layer which takes in a certain set of input weights and outputs a value accordingly through an activation function. In this case 'ReLU' or Rectified Linear Units was used here as the activation function. To keep the weights to a minimum, only 16 nodes are included in this hidden layer. The layer before the hidden layer is the second pooling layer which has an output shape of 2x7x7 nodes. All of these nodes would be fully connected to the 16 nodes in the hidden layer and would produce one weight each which, when multiplied, would amount to 1568 weights.

The second Fully Connected layer is called the Logits layer and it is the layer responsible for predicting the digit displayed. Because it is a Fully Connected layer, all 16 nodes from the hidden layer are connected to the 10 nodes used for classifying the digit from 0-9, which would produce a total of 160 weights. The activation used in this layer is 'softmax' where it would only output a value from 0-1 and the total of all its outputs would be equal to 1. There are 10 outputs in this layer, each predicting the probability of the number 0-9 to be the one being displayed or tested. Table 1 summarizes the specifications for the CNN design.

Table 1: CNN Specification

Input Image Size	28 x 28
Filter Matrix Size	5 x 5
Number of Filter Weights	150
Size of Fully Connected Layer	16 Nodes
Number of Fully Connected Layer Weights	1568 Weights
Size of Logit Layer	10 nodes
Number of Logit Layer Weights	160 Weights
Overall Accuracy	93.32%

3.6 Fully Connected Layer

The weights used in the CNN were represented as the multiplicative factor of the filters. They were extracted using the \model.get_weights() command in the Keras library. Weights were used as the inputs for both C++ and VHDL implementations of K-Means Clustering. The clustered weights present at the output of the K-Means clustering implementation were then fed back to the CNN using the \textit{model.set_weights()} command and the accuracy was evaluated using the \textit{model.evaluate} command. A total of 1878 weights were extracted and clustered using K-Means Clustering. Before clustering, the CNN already achieved an overall accuracy of 93.32%.

3.7 K-means Clustering

With a future goal of moving into ASIC, a VHDL code is used. Writing a software code like C language for the clustering algorithm, makes the processing sequential. It also takes considerable waiting time before the clustering finishes most especially if there are many input weights to compress. This paper proposes the use VHDL can deliver the results faster. This is because in VHDL, several variables can be updated all the same time. Although the code is not meant for synthesis, the VHDL model written still adopted a sequential circuit setup. Clock and reset input are still used in the code. The whole system is still made synchronous with the clock input. The completed model comes one step closer to realizing the synthesizable hardware model for future ASIC implementation. Shown in Figure 4 is the block diagram of the K-means Clustering hardware model. The inputs to this module are the weight values coming from the CNN. The number of groups or K is decided by the user. The value of K will dictate the number of centroid values.

Figure 4 shows the block diagram. The inputs undergo Euclidean distance calculation with each centroid value through the EUDIST module. The initial centroid values are chosen randomly from the list of inputs. For every input, the number of Euclidean distance values computed is equal to K. A search for the minimum from among the values in every set will determine to which group does an input belong to. The search is done by the Findmin module. The output of the Findmin modules determines the groupings of the inputs. The average of every group determines the new set of centroid values. The averaging procedure is done by the sumave module. In sumave module, a sum accumulator works by searching for every input belonging to one cluster and accumulates the sum. For every input searched for a group, not only is the sum accumulated, but a counter is also incremented. Equations 1 and 2 shows the behavior of the sum accumulator and the counter respectively. Sum S_N denotes the variable that accumulates the sum of all inputs belonging to a particular cluster, n. In is the input belonging to the aforementioned cluster. C_N counts the number of inputs belonging to the same cluster. Dividing S_N by C_N yields the centroid value of the given cluster after every iteration. With the new set of centroid values, the entire operation starting from the Euclidean distance calculation is repeated to regroup the inputs. The operation only stops when the current set of centroid values is exactly the same as the set of previous values. In the FINALANS module, every input is represented by the average value of the group to which it belongs. The MUX module is an option for releasing every encoded input.

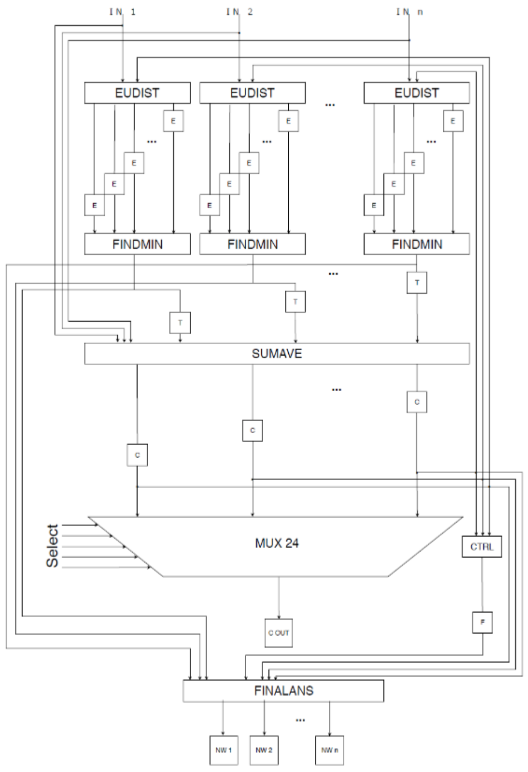

Figure 4 : K-Means Clustering Block Diagram

$$S_N = S_N + I_N \quad (1)$$
$$C_N = C_N + 1 \quad (2)$$

4. DATA AND RESULTS

The goal of the compression is to reduce the CNN weights memory usage through weight quantization using K-means clustering algorithm. This effect is weight sharing. The higher the compression ratio, the better is the memory savings. However, weight quantization means changing the original value of the CNN weights after training is completed. Changing the weight values after the training can obviously affect the very purpose of the CNN training i.e. image recognition. In this section, the weights were compressed using various ratios and then the CNN is subsequently evaluated to see if its image recognition ability is still working. The data in gathered in this section is an analysis of up to how much compression, which also denotes up to how much memory savings, can be carried out without sacrificing the CNN's performance when it comes to image recognition. Table 2 shows how much memory is occupied by the CNN on the Random Access Memory of the computer. This representation is widely used for computing how much a CNN implementation would take up in the RAM of the computer and was seen in part in [19]. Before clustering is applied, 166.2MB of the RAM is occupied by the CNN since the model has been fitted with 60000 images. Applying K-means clustering to the CNN model will linearly decrease the memory used by the CNN since clustering applies quantization to the weights. This can be seen also in another study in [21] where they applied binary quantization by changing all the weights into binary form which gave them a very big advantage in terms of memory without suffering a major loss in accuracy.

Table 2: Memory Occupied By CNN

Layer	Output Size	Memory
Input	28x28x1	784B
Conv2d.1	28x28x2	1568B
Conv2d.2	14x14x2	392B
Dense.1	1x1x16	16B
Dense,2	1x10	10B
Total Memory per Image		2.77KB
Total Memory(6000 images)		166.2MB

Figure 5 represents the memory usage of the CNN for every weight compression applied. As seen from the figure, the memory of the CNN has been reduced from 166.2MB up to 16.62MB when 90% clustering is applied.

The memory savings can increase up to around 150 MB but the accuracy will noticeably drop. A study by [22] states that applying K-means clustering to weights can lead to very good balance between memory of the CNN and accuracy of recognition which is evident in the compression of the CNN.

A 100% compression means all weights of the CNN will be represented by just 1 value. This level of compression however shows the CNN failing in its recognition ability.

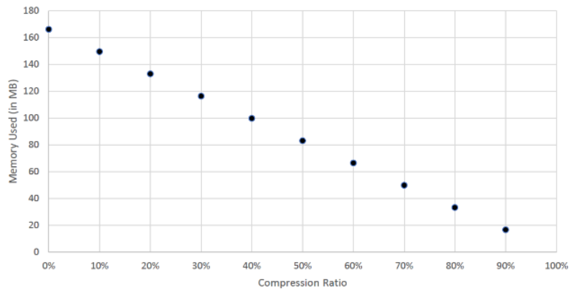

Figure 5 : Compression vs Memory Usage

Figure 6 portrays the amount of resource units in the Random Access Memory of the computer saved with respect to the compression ratio of the CNN's weight values. There is an evident trend in resource unit savings as far as the increase in compression ratio is concerned, although it is not as large and not as linear.

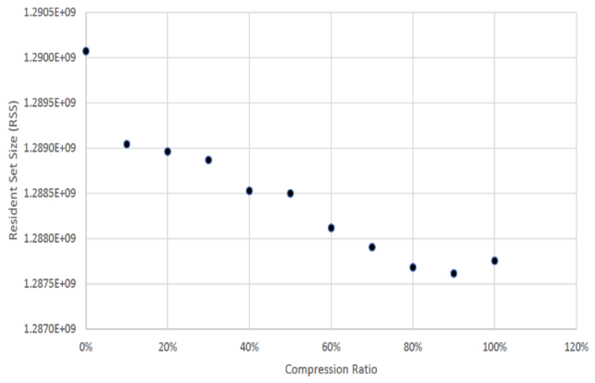

Figure 6 : Resource Unit vs Compression

Table 3: Compression vs Accuracy

% Clustering	Accuracy
0	93.32 %
10	93.28%
20	93.20%
30	93.35%
40	93.45%
50	93.35%
60	93.29%
70	92.90%
80	91.30%
90	89.81%
100	11.35%

The Keras based CNN was trained using training images obtained from MNIST until the target outputs are reached. After the training, test images from MNIST were subsequently fed to the CNN to test its performance. Due to the limitation imposed on the size of filters, the accuracy obtained after testing, resulted to 93.32%. The weights of the CNN were then compressed through weight quantization using K-means clustering. Table 3 shows the data gathered from testing the accuracy of the CNN for various compression rate obtained from the VHDL simulation results. It can be seen from the table that the system accuracy hardly deviated from the no compression performance even when the compression has reached 60%. As seen from Table 3, depending on the requirement of the designer, the CNN's accuracy can be considered high even when compression has reached as high as 90%.

5. CONCLUSION AND RECOMMENDATION

In this paper, a Keras based CNN was developed and subsequently used for training using MNIST images of hand-written digits. After the training, the CNN was tested using test images, also provided by MNIST. Due to the reduction in filter size to 5x5, the accuracy obtained was placed at around 93.32%. Applying weight compression through weight quantization using K-means clustering algorithm, the CNN's accuracy stays at above 93%, even when compression has reached 60%. This is considered very close or almost the same as the accuracy level without compression applied. At 70% compression ratio, accuracy can still be considered good at 92.9 %. Also shown in this paper is the CNN's reduction in memory usage as the compression ratio increases. From 166.2MB, the memory usage can go down to as low as 16.62MB at 90% compression ratio.

It is recommended for further study, to design and implement a CNN architecture with weight compression feature using the hardware approach. The hardware design approach can be realized through FPGA or ASIC implementation. To avoid consuming too much hardware for implementing the K-means clustering algorithm, the designer may opt to limit the minimum compression ratio to 60%. At this value, there should be significant savings in memory usage while maintaining a good accuracy performance of the CNN when compared to the no compression scenario.

REFERENCES

1. Mahmoud M. Abu Ghosh, Ashraf Y. Maghari. **A Comparative Study on Handwriting Digit Recognition Using Neural Networks**, *2017 International Conference on Promising Electronic Technologies (ICPET)*, Deir El-Balah, October, 2017, pp. 77-81.
2. Mayank Jain , Gagandeep Kaur, Muhammad Parvez Quamar et. al, **Handwritten Digit Recognition Using CNN**, *2021 International Conference on Innovative*

Practices in Technology and Management (ICIPTM), Noida, February, 2021, pp. 211-215.

3. Elizabeth Rani. G, Sakthimohan. M, Abhigna Reddy. G, et. al, **MNIST Handwritten Digit Recognition using Machine Learning**, *2022 2nd International Conference on Advance Computing and Innovative Technologies in Engineering (ICACITE)*, Greater Noida, July, 2022, pp.768-772.
4. Kuppa Venkata Padmanabha Nandan, Manoj Panda and S Veni, **Handwritten Digit Recognition Using Ensemble Learning**, *2020 5th International Conference on Communication and Electronics Systems (ICCES)*, Coimbatore, July 2020, pp. 1008-1013.
5. Saqib Ali, Zareen Sakhawat et. al, **A robust CNN model for handwritten digits recognition and classification**, *2020 IEEE International Conference on Advances in Electrical Engineering and Computer Applications(AEECA)*,Dailan, August 25-27, 2020, pp. 261-265.
6. Ashadullah Shawon , Md. Jamil-Ur Rahman et. al., **Bangla Handwritten Digit Recognition Using Deep CNN for Large and Unbiased Dataset**, *International Conference on Bangla Speech and Language Processing (ICBSLP)*, Sylhet, September, 2018.
7. MD Shahbaz Khan, Niharika, Priya Yadav et. al., **FPGA Simuation of Fingertip Digit Recognition Using CNN**, *7th International Conference on Signal Processing and Integrated Networks (SPIN)*, Noida, February 27-28, 2020. Pp. 1072-1077
8. Taher Mostafa El-Sahhar, Mohamed A.Wahby Shalaby A. Rahman, **KNN and the CNN for Handwritten Digit Recognition: A comparative study**, *2023 5th Novel Intelligent and Leading Emerging Sciences Conference (NILES)*, Giza, October 21-23, 2023, pp. 339-342
9. Divya Konikkara , Mrs Usha K , **MALAYALAM Digit Recognition using CNN**, *2022 Third International Conference on Intelligent Computing Instrumentation and Control Technologies (ICICICT)*, Kannur, August 11-12, 2022.
10. A. Dundar, J. Jin, B. Martini, and E. Culurciello, **Embedded streaming deep neural networks accelerator with applications**, *IEEE Transactions on Neural Networks and Learning Systems*, Vol 28, Issue 7, pp. 1572–1583, July 2017.
11. T. Manabe, Y. Shibata, and K. Oguri, **FPGA implementation of a real-time super-resolution system using a convolutional neural network**, in 2016 *International Conference on Field-Programmable Technology (FPT)*, Xi'an, December 7-9, 2016, pp. 249–252.
12. I. A. Pagnuco, J. I. Pastore, G. Abras, M. Brun, and V. L. Ballarin, **Analysis of genetic association using hierarchical clustering and cluster validation indices**, *Genomics*, Vol 109, pp. 438-445, 2017.
13. C. Jinyin, L. Xiang, Z. Haibing, and B. Xintong, **A novel cluster center fast determination clustering algorithm**, *Applied Soft Computing*, vol. 57, pp. 539 – 555, 2017.
14. R. Ramon-Gonen and R. Gelbard, **Cluster evolution analysis: Identification and detection of similar clusters and migration patterns**, *Expert Systems with Applications*, vol. 83, pp. 363 –378, 2017.
15. A. Cornujols, C. Wemmert, P. Ganarski, and Y. Bennani, **Collaborative clustering: Why,when, what and how**, *Information Fusion*, Vol. 39, pp. 81 – 95, 2018.
16. H. M. .W. D. Song Han, **Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding**, Available: *https://arxiv.org/pdf/1510.00149.pdf*, 2016.
17. Keras: **The Python Deep Learning Library**, Available: *https://keras.io/?fbclid=IwAR1y9rZ5VcDoME55wIhTn6mlwfaUXIYI0w4ZncOqK-HZcOjZJbXGggVguDg*
18. **CS231n Convolutional Neural Networks for Visual Recognition**, Available: : *http://cs231n.github.io/convolutional-networks/?fbclid=IwAR3oxTeP-IDOUMb7l4e1BO_Fd5TWUIkvI84pKVGJUnMcLFpdhz-KO-imLzQ*
19. Brownlee, Jason, **Handwritten Digit Recognition using Convolutional Neural Networks in Python with Keras**, Available : *https://machinelearningmastery.com/handwritten-digit-recognition-using-convolutional-neural-networks-python-keras/?fbclid=IwAR15cEz6U0YXgQouVjGNdd0bkY-DXRQTUlxfqGzNorPG8jvtVAAk2lmcBDw*
20. Tom Hope, Yehezkel S. Resheff, Itay Lieder, **Learning TensorFlow: A guide to building deep learning systems**, Available: *https://lucien-labadie.firebaseapp.com/aa431/learning-tensorflow-a-guide-to-building-deep-learning-systems-by-tom-hope-yehezkel-s-resheff-itay-lieder-1491978511pdf*
21. Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, Ali Farhadi, **XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks**, Available: https://arxiv.org/pdf/1603.05279.pdf
22. Yunchao Gong, Liu Liu and Lubomir D. Bourdev, **Compressing Deep Convolutional Networks using Vector Quantization**, Available: https://arxiv.org/pdf/1412.6115.pdf