Speech Recognition model compression

APPROVED BY

SUPERVISING COMMITTEE:

Ahmed Tewfik, Supervisor

Raymond J. Mooney

Speech Recognition model compression

by

Madhumitha Sakthi

REPORT

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science in Engineering

The University of Texas at Austin

May 2019

Dedicated to my Amma and Appa.

Acknowledgments

I would first of all want to thank my supervisor Dr. Ahmed Tewfik. He gave me the right combination of freedom to explore and independently work and at the same time guided me through whenever I was stuck. He guided me all through and it was great learning experience working with him. I am very grateful to be advised by him.

I would like to thank Prof. Raymond J. Mooney, for being the second reader for my report, giving me his valuable time and feedback.

I would also like to thank my friend Aastha Tripathi, for providing her comments, for the discussions on the topic and for the enormous moral support throughout my report work.

I would like to thank my parents for always being supportive of all my goals.

Finally, I would like to thank God for giving me an this opportunity

Abstract

Speech Recognition model compression

Madhumitha Sakthi, M.S.E.
The University of Texas at Austin, 2019

Supervisor: Ahmed Tewfik

Speech recognition models are widely deployed in mobile and embedded devices. However, the base architecture with which these models are developed is usually made of neural networks with bigger size and millions of model parameters. In this report, we investigate three compression schemes for these neural network architecture with a trade-off on accuracy and compressed model size. Also, we perform sensitivity analysis on the network parameters with known perturbations to determine the best compression scheme for a particular layer. The first compression scheme deployed is k-means clustering. This helps in generating clusters which are used for weight sharing and hence reduction in the total number of parameters required. Secondly, we employ svd based compression on various network layer parameters and achieve the best compression using svd in the case of a large vocabulary continuous speech recognition model. Finally, a two-stage compression scheme using k-means and

Huffman coding is proposed. We have investigated these compression schemes on keyword spotter speech recognition system and the Baidu's DeepSpeech large vocabulary continuous speech recognition model and have shown 58.3% reduction in size for only a 3.4% drop in accuracy and 45% reduction in size for only a 1.21% drop in accuracy respectively.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Recent advancements in deep learning have facilitated its usage in many day-to-day utilities. One such application is in Speech recognition. It is a problem where a user would give in voice commands or queries and the system would recognize the text. However, deploying such systems in embedded and mobile devices requires the models to be compact and yet accurate on various noise conditions. Hence, in this report, we have investigated the usage of three compression schemes on key-word spotter and DeepSpeech speech recognition architecture. The compression schemes proposed in this report does not require re-training. Therefore, accelerating the process of deployment of compressed models to mobile devices without much loss in accuracy.

In this report, the model parameters were compressed using k-means clustering and svd compression. As a two-stage compression scheme, we have evaluated k-means followed by Huffman coding as a third compression technique. Also, by performing sensitivity analysis on the model parameters we determined the level of perturbation the model could handle without much loss in accuracy. The parameters were perturbed with Gaussian noise, by a fraction of itself, followed by a value higher than the initialization values be-

fore training. Finally, based on sensitivity analysis, it is determined that the unique model parameter values could be sorted into bins of a histogram and the values in the bin, replaced with the mean value does not affect the model performance. We have shown that this scalar quantization method helps in representing the values with 2 bits instead of 32 bits to save memory requirement.

In chapter 2, we discuss related work. In chapter 3, we discuss the model architecture and the compression algorithms are explained. In chapter 4, we have shown the experimental results and discussed the limitations and future work in chapter 5. Finally, chapter 6 outlines the conclusions of the report.

Chapter 2

Related work

In recent years, with the development of deep learning, speech recognition architectures are deployed using deep learning models rather than Hidden Markov models [1]. Large, multi-layer deep learning architectures have been successful in computer vision, classification tasks [2]. These architectures typically have a few million parameters in a size range of a 200 to 300 MB. To deploy the compressed architectures with minimal decrease in accuracy, previous papers have investigated knowledge distillation thoroughly [3]. However, with knowledge distillation, it is imperative to train the models from scratch and determine the best parameter choices for the particular task. Also, it is a sequential training process where, the student neural network, learns to mimic the teacher neural network's output and hence would take a longer time to train the model. In [4], they proposed a compression scheme for RNN layers. Particularly, they propose a projection layer by compressing the recurrent layer using singular value decomposition and retaining only the top few singular values. This leads to size reduction at the projection layer and hence, they achieve parameter reduction. But, this method requires additional fine-tuning to achieve better performance with an increase in compression. In [5], they have proposed a multi-stage compression scheme, where they performed

singular value decomposition compression followed by quantization. Using quantization, they reduce the 32bit representation to 8-bit representation and achieve 4 times reduction in size. However, this method is prone to drastic changes in the model parameters. We have utilized scalar quantization based on the parameter distribution and retained the float values. This way, even though the float representation remains intact, the number of unique values is decreased and hence, they are mapped to uint8 integers for storage.

In the recent paper [6], using voice data of approximately 15000 hours, they performed three compression techniques. Knowledge distillation, low-rank matrix factorization and finally, pruning to LSTM layers while training. Pruning the LSTM layers gave the maximum compression for a word error rate of 6.4 %. Pruning is a method wherein while training the less salient connections in the neural network are removed and hence, it generated sparse weight matrices. Compression is achieved by reducing the model size by storing a sparse matrix. As an addition to the above methods, [7] has shown that knowledge distillation and pruning achieved 14.59× parameters reduction, 5× storage size reduction. Also, using layer normalization, they could accelerate convergence. When knowledge distillation is applied on a 6 layered, 1024 hidden size Deep Neural Networks, it was reduced to a 3 layered, 512 hidden size architecture with just 2% loss in accuracy. Therefore, with knowledge distillation, when the network parameters are jointly trained, it is possible to achieve more than 4x compression rate for a minimal loss in accuracy. So far, there have been various compression schemes on speech recognition models.

However, these methods require either model re-training or fine-tuning. In addition, there have been compression techniques introduced in the vision deep learning models. One such compression scheme[8] utilized k-means clustering on the convolutional kernels. Starting from a pre-trained model, representative 2D kernel centroids are extracted using k-means clustering. ResNet-18 even outperforms its uncompressed counterpart at ILSVRC 2012 classification task with over 10x compression ratio. Combined with pruning, the compressed VGG-16 achieves over 30x compression ratio with only 0.01% accuracy drop on the CIFAR-10 dataset. However, even in this method, the model is fine-tuned after applying k-means clustering to the convolutional kernels. In another Deep Neural Networks compression paper[9], they have performed a three-stage compression process of pruning, trained quantization ,and Huffman coding, and reduced the storage requirement of neural networks by 35× to 49× without affecting the model performance. Quantization was performed by clustering and weight sharing. On the ImageNet dataset, this method reduced the storage required of AlexNet by 35×, from 240MB to 6.9MB, without loss of accuracy.

In this report, to the best of our knowledge, this is the first method to have k-means clustering followed by Huffman coding implemented on speech recognition models. Also, we have implemented a k-means clustering based compression and singular value decomposition compression scheme. This is the first method to explore a thorough sensitivity analysis on the model parameters with various noises. We have also proposed a scalar quantization based

compression scheme based on the sensitivity analysis results. Also, in addition to the above, we have shown the change of memory requirement across the hardware platform when the model is compressed for storage reduction and decompressed for inference. Finally, the compression schemes proposed in this report work without model retraining or fine-tuning. Therefore, the best compression scheme is decided based on the trained models and the objective is to reduce the model size while the network remains robust.

Chapter 3

Model Architecture & compression algorithm

In this section, two speech recognition model architectures and their corresponding compression algorithms are explained. The first speech recognition model is the keyword spotter. It is a simple classifier network of 10 distinct speech sounds, each 1 second long. This is trained on the command and control dataset[10]. The second model is the large vocabulary continuous speech recognition system based on DeepSpeech[11] architecture. This model is capable of recognizing speech sentences.

As shown in figure 3.1, to train a speech recognition model, the input speech data is pre-processed as a spectrogram or MFCC features. These features are used to train the model based on a particular loss function. The last layer consists of a classifier layer with a softmax activation. This layer predicts the class probabilities. In the key-word spotter recognition system, the model is trained to directly predict the ten keywords along with unknown and silence. Whereas in the case of DeepSpeech model, it predicts the characters with blank and unknown characters. Therefore, the predicted characters are combined to words using the blank symbol and hence, words are decoded. Similar to a typical deep learning system, the model parameters such as the

number of layers and the number of units in each hidden layer is determined while training.

3.1 Keyword spotter

The dataset used for this architecture is Google's speech command dataset with over 60,000 instances of 30 words. The twenty core command words are spoken at least 5 times by each speaker. Whereas, the other 10 are auxiliary words spoken just once by each speaker. In this report, we trained the key-word spotter to recognize 10 core command words[10], 1 unknown symbol and 1 silence symbol. The total number of classes predicted by the model is 12.

The audio is sampled at 16000 kHz sampling rate. The pre-processing step extracts Mel-frequency cepstral coefficients(MFCC) from the audio data. To extract the MFCC features, a pre-emphasis filter is applied to the speech data, followed by framing and windowing. On these frames, Fourier transform is applied and Mel-scale filter banks are applied on the Fourier transformed signal. Finally, Discrete Fourier transform is applied to decorrelate the filter bank coefficients to obtain the MFCC features.

While training, the audio data is corrupted with the background noise of 10% volume. This helps the model to be robust in varying background noise conditions. The model architecture[12] consist of two convolutional layers followed by a feed forward layer and it is trained with a cross-entropy loss function. ReLU activation function is used as the activation function for the

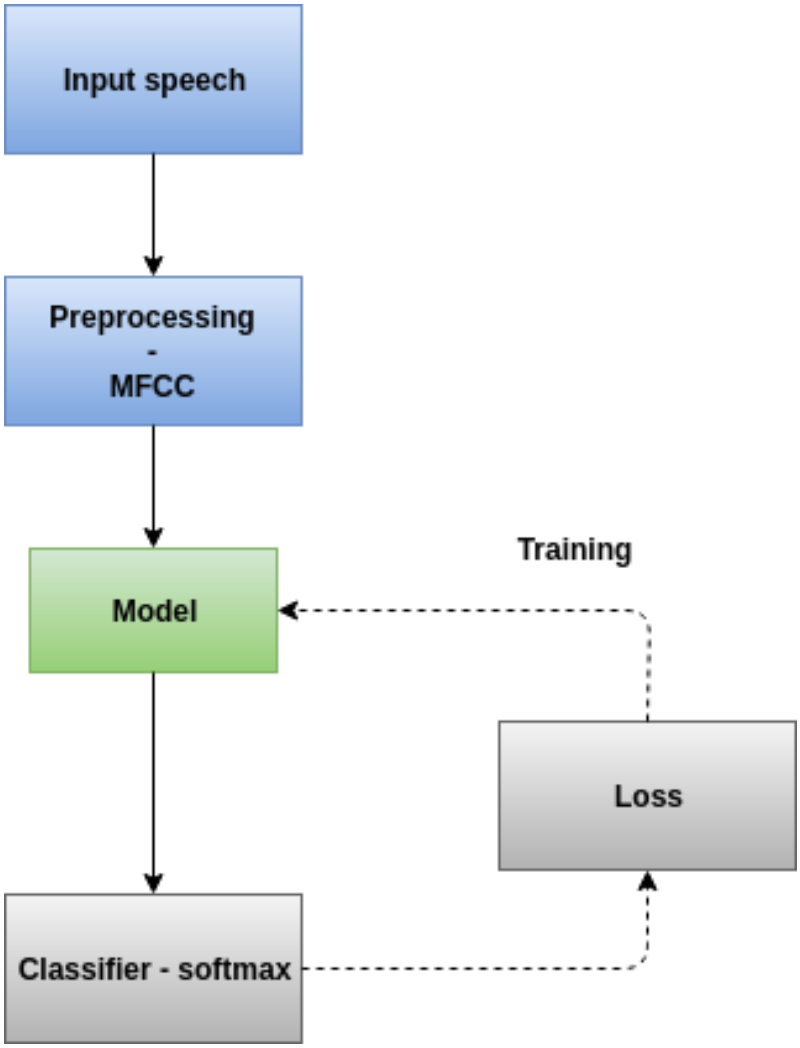

Figure 3.1: Speech recognition architecture

hidden layers. The MFCC features, extracted from the speech signal, frame the 1-D speech data into a 2-D frame data. Hence, similar to a vision application, CNN neural network architecture as the hidden layer. The feedforward layer contains the maximum number of model parameters which account for almost 87% of the model size. This model is trained for 18000 steps. The first 15000 steps are trained with a learning rate of 0.001 and the last 3000 steps are trained with a learning rate of 0.0001. At the end of the training, the model has a test accuracy of 87.5% and a model size of 3.7 MB with 926860 parameters.

As shown in figure 3.2, the input wav data is preprocessed in the first three nodes, the input is reshaped in such a way that the first convolutional layer can process it. Followed by that, a randomly initialized bias is added to the processed input and sent to the next convolutional layer. The feature_map size of the convolutional layers is 64. The output from the second convolutional layer is passed through the ReLU activation function and finally given to the linear layer. The linear layer has the output dimension of 12, which is predicted as class probabilities after a softmax layer. Hence, any given input frame is classified as one among the 12 output symbols.

3.2 LVCSR: DeepSpeech

The dataset used to train the model was Fisher [13], switchboard [14] and Librispeech [15]. The model is tested on the test set of Librispeech dataset. MFCC pre-processing was applied to the speech dataset. The LVCSR archi-

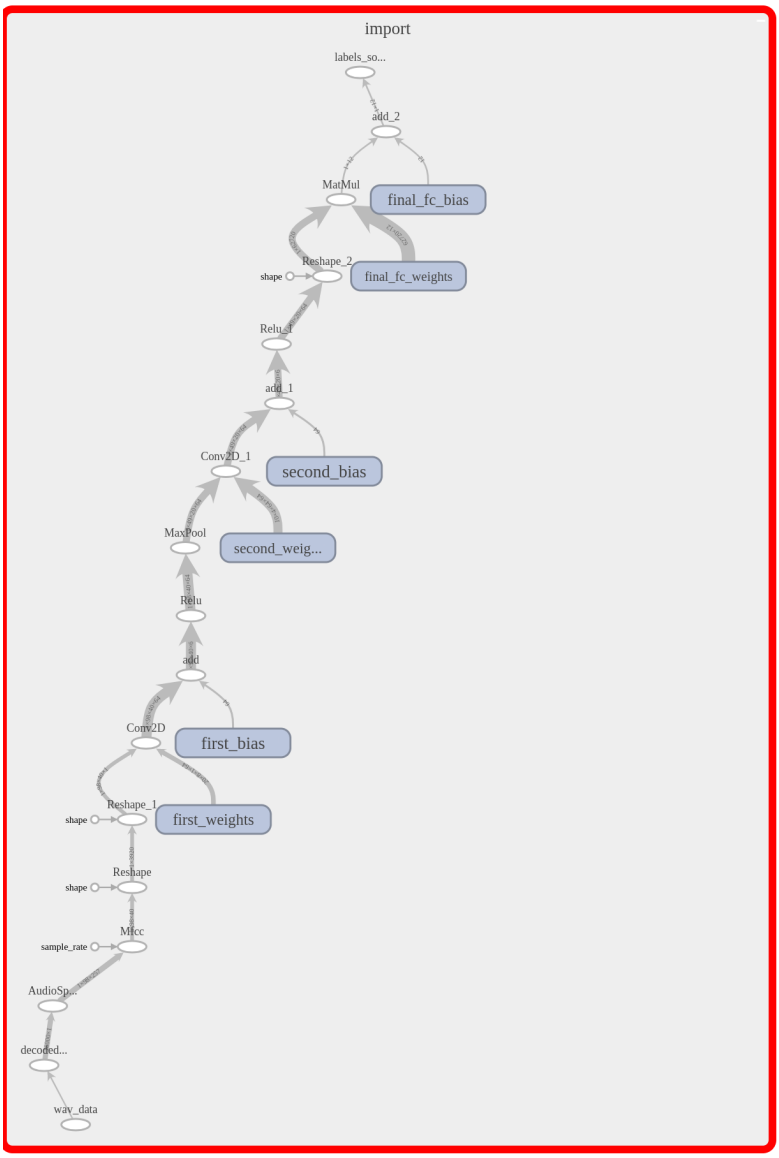

Figure 3.2: Key-word spotter graph generated from tensorboard

tecture[11] consist of 5 hidden layers. The first three layers are not recurrent layers. The fourth layer is a bi-directional LSTM which consists of forward and backward recurrent layers. The fifth non-recurrent layer takes input from both forward and backward units and the final layer is a softmax layer which predicts character probabilities.

The loss function is Connectionist Temporal Classification (CTC) loss function [16] [17]. Unlike the previous architectures which required phoneme level annotation of speech, using CTC loss function, the model can decode speech to characters directly. Initially, using a HMM[1][18] or a Deep Neural Networks architecture for a phoneme level classification required phoneme annotations. The loss functions used were typically cross-entropy loss. Also, the speech recognition architecture was divided into the acoustic model, pronunciation dictionary and a final language model as shown in figure 3.3 . Using the recent deep neural network architectures and loss functions, these three components are combined into a single architecture.

The Connectionist temporal classification loss function does require phoneme level or character level annotation of the speech data. This is because CTC loss directly computes the conditional probability of the output character given the input frame and this is done by marginalizing over all possible alignments of the output. Therefore, the CTC loss is capable of taking input frame length higher than the output character length to be predicted. These characters are later combined as words and hence the WER and Character error rate (CER) are calculated. The final WER for this model is 18%

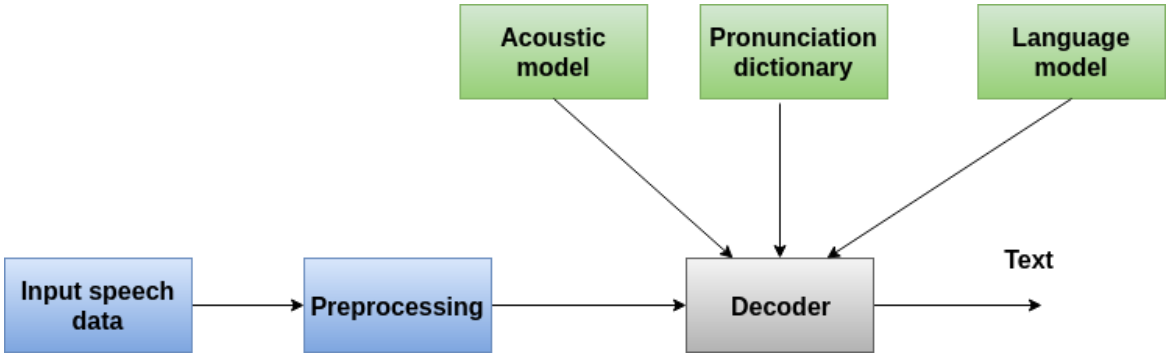

Figure 3.3: Traditional Speech recognition model architecture

and a CER of 9%. Also, with the CTC decoder, sometimes, an external language model is not utilized for fine-tuning the output word predictions. But, to achieve comparable performance without using the language model, the speech recognition model should be trained with large amounts of data. In which case, the model can be trained to directly predict word outputs using CTC [19].

As shown in figure 3.4, the input speech feature is passed through three linear layers, followed by a bi-directional LSTM layer. The bi-directional layer interacts within the layer itself and has information about the past and the future. The final layer takes input from both the forward layer and the backward layer and predicts the output from a final softmax classifier. However, the bi-directional recurrent neural networks are expensive to train.

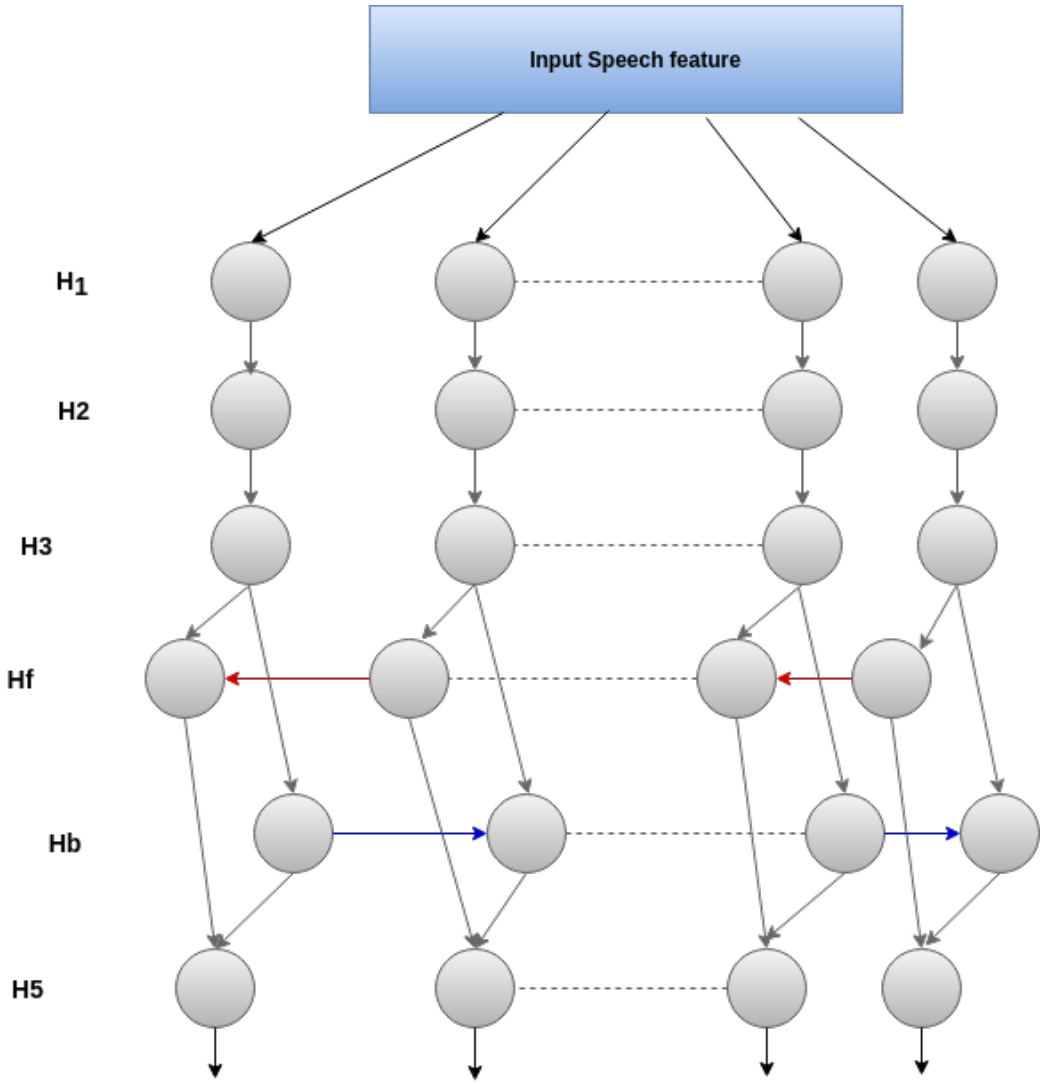

Figure 3.4: DeepSpeech model architecture

3.3 Compression schemes

In this section, three compression schemes are explained. The first compression scheme is a singular value decomposition. The second compression scheme is k-means clustering. Finally, as a two-stage compression scheme, Huffman coding is applied to the labels generated by the k-means clustering and the size if further reduced without loss in model performance.

The compression scheme can be chosen based on the sensitivity analysis and the dimension of the layer's parameter matrix. Mainly, in this report, we have chosen the layer with the maximum number of model parameters to achieve the maximum benefit of compression. As shown in figure 3.5, the layer with the maximum parameter is compressed based on the sensitivity of that layer in the model.

3.3.1 Singular value decomposition compression

The singular value decomposition involves decomposing the matrix

$$A = U\Sigma V^T \tag{3.1}$$

The eigenvectors of AA^T form the columns of U and the eigenvectors of A^TA forms the columns of V. The sigma values are the square root of the singular values of the above matrices. The singular values are all real numbers and are arranged in descending order. Using Singular value decomposition, an 8000 x 8000 matrix can be reduced to having only 400 components. Where, the decomposition would become 8000 x 400, 400x400, 400 x 8000 and hence

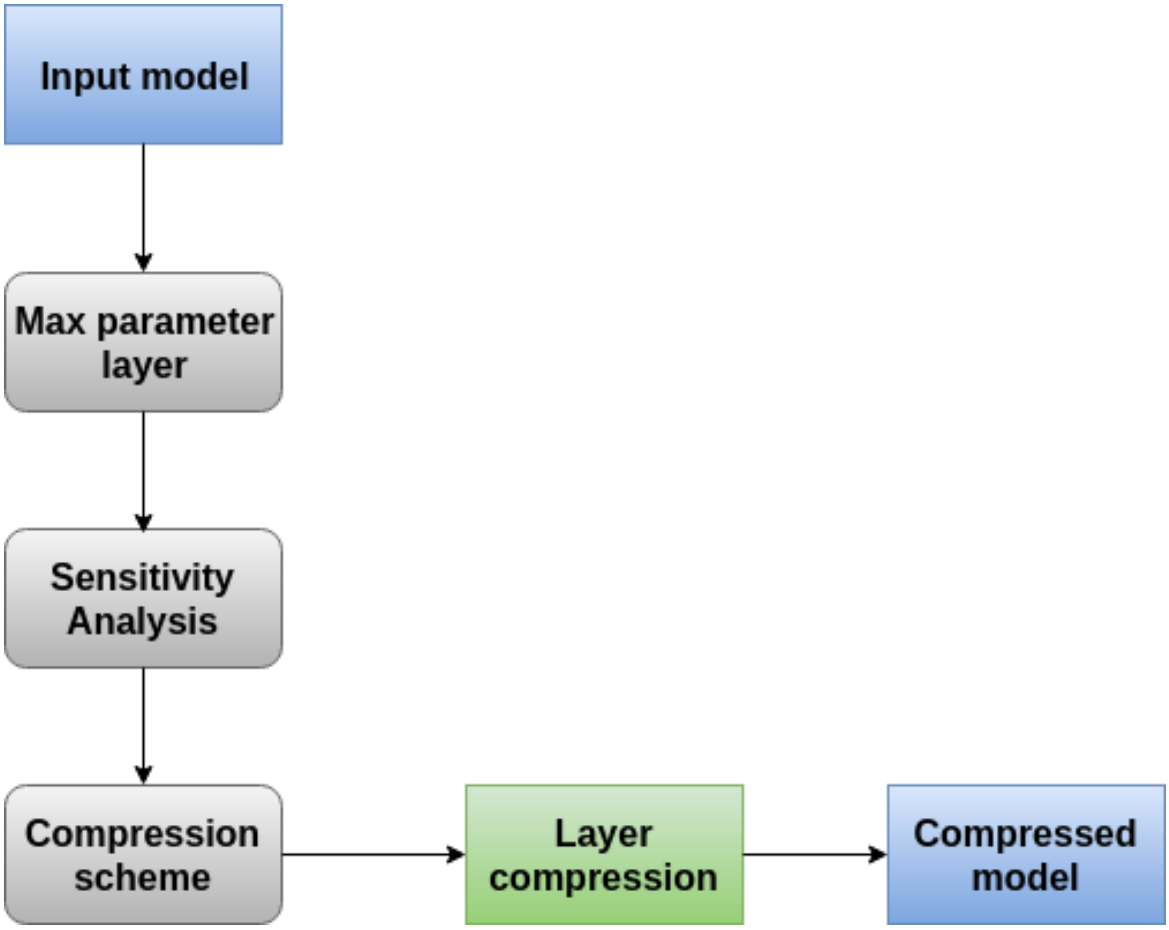

Figure 3.5: Compression algorithm

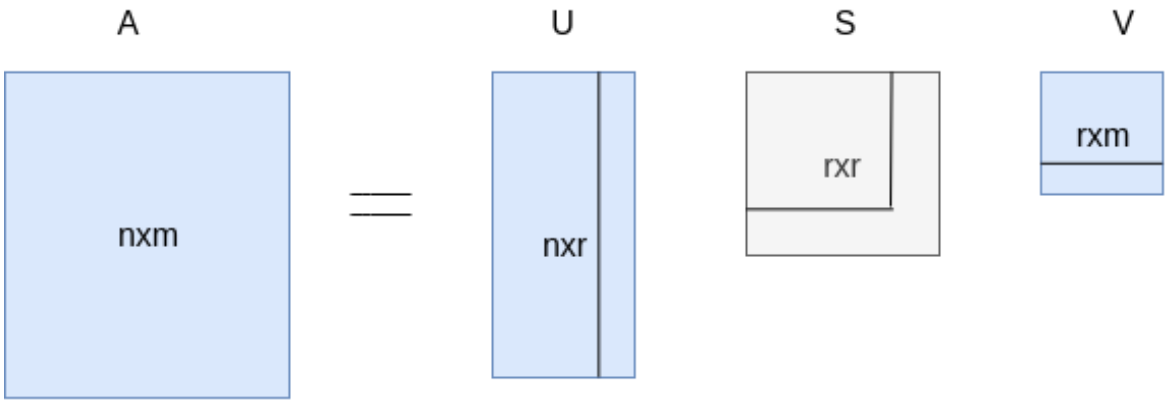

Figure 3.6: Singular value decomposition of a Matrix M

reduces the number of the parameter in the matrix. Therefore, Singular value decomposition is an effective compression scheme for a 2-D matrix. However, for a higher dimensional data, it can be reshaped to a 2-d matrix and then Singular value decomposition can be applied to that matrix. The compression based on the number of components is limited to the rank of the matrix.

As shown in figure 3.6, the matrix A is decomposed into three matrices. However, the number of parameters is reduced when the dimension of r is less than the dimension on the m. This way, the components that do not contribute to much information of the matrix are removed without much loss in the model performance.

3.3.2 k-means clustering

In k-means clustering, 'k' is determined by experiments. The algorithm takes k as input and the values to the cluster. Based on the value of 'k', k

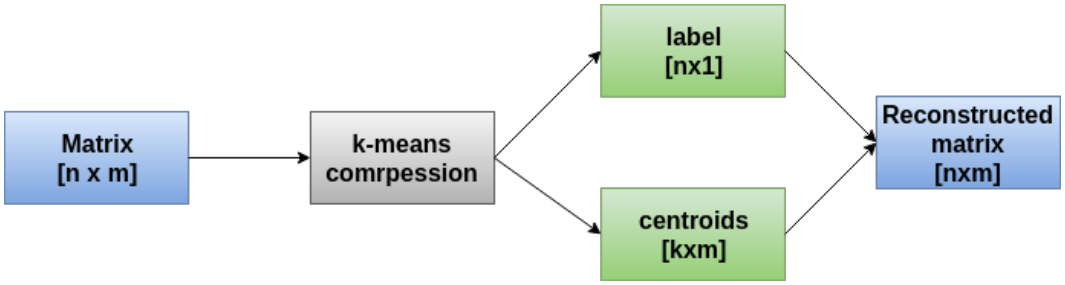

Figure 3.7: K-means compression-decompression of a matrix M

random cluster centroids are assigned. As an iterative step, the values are assigned to the nearest centroid based on Euclidean distance. For each centroid, the mean of the values of all the points belonging to it is calculated. The mean is the new centroid value. The above two steps are repeated until there is no change in the centroid value.

As shown in 3.7, matrix A, after model compression generates two matrices labels and centroids. These are combined again to form the reconstructed matrix of the same dimension as that of matrix A. However, the size of the storage is reduced by storing only the labels and centroids while the actual matrix is reconstructed on-the-fly for inference.

When k-means clustering is applied to a 2-d matrix, typically it is most effective for a nxm matrix where n¿m. For each row, k-cluster centroids are calculated. These cluster centroids are calculated for each row leading to a kxm matrix. A separate label matrix of size nx1 has the label information. Therefore, for a nxm matrix, the final model parameters to be stored are kxm + n parameters.

3.3.3 K-means & Huffman compression

After k-means clustering based compression, the label parameters are integers. For a large nx1 matrix, the elements are one among k label values. Therefore, this repetition of label information can be compressed using Huffman coding. In Huffman coding, the frequency of occurence of each number is obtained and sorted in descending order. These numbers are used to build the tree with the least frequent numbers being farthest from the node. Parsing through the tree, the encoding and decoding of these numbers are performed. This creates a bit code for each integer. Leading to the most frequent integer having the least number of bits and the least frequent number gets the longest bit encoding. The corresponding tree is saved for decoding the bits to numbers. Hence, this two-stage compression scheme helps in achieving maximum compression using k-means with no loss in accuracy after a Huffman compression stage.

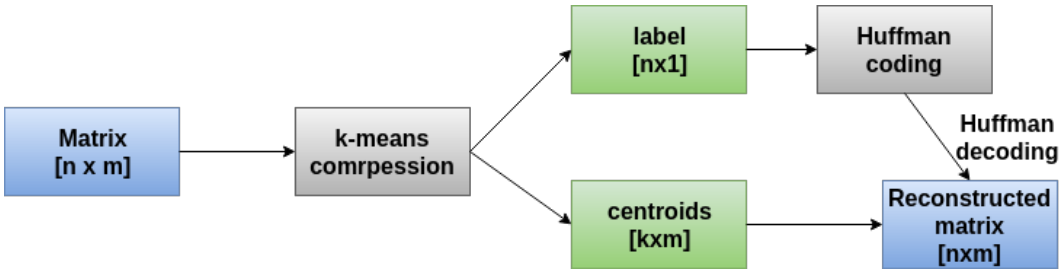

Figure 3.8: K-means & Huffman compression - decompression of matrix M

As shown in figure 3.8, matrix A is compressed using k-means compression. This generates labels and centroids matrix. The label matrix is encoded

using Huffman coding. While reconstructing the matrix, the Huffman encoded label matrix is decoded and then used for reconstructing the matrix A with the centroids information.

3.4 Model compression in hardware

The proposed compression schemes decreased the model size at the hardware flash level. Flash memory retains the data in the absence of power supply. Normally, in mobile and embedded devices, the model is stored in the flash memory. At the time of usage of the model as an application, the model is loaded from the flash memory to the DDR memory. The multi-core processor would fetch the model from the DDR memory to perform inference. The size of the model before and after compression is shown in figure 3.9

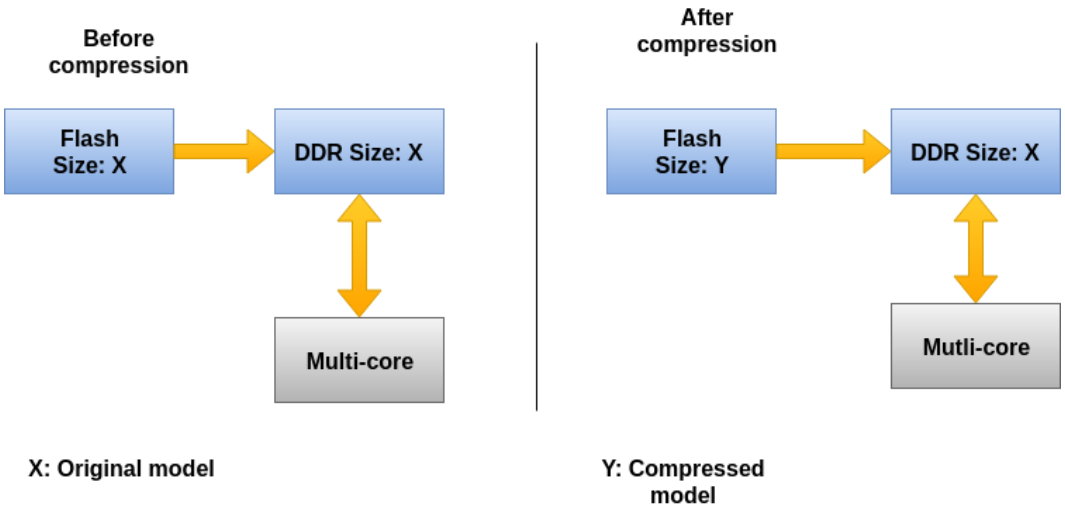

Figure 3.9: K-means & Huffman compression - decompression of matrix M

The before compression stage represents the original trained model of size X. This is usually in the order of 200 to 300 MB. However, after compression, this size is reduced to 50 to 150 MB using the above-mentioned compression schemes. The compressed model of size Y is decompressed at the DDR memory for inference.

Chapter 4

Sensitivity Analysis

The compression of layer parameters involves a change in the value of the parameter. Therefore, in this section, we analyze the sensitivity of the trained network parameters to known perturbations. The value of the network parameters was perturbed with a particular value (max_value, min_value), with Gaussian noise and with a percentage of the individual number itself on both key-word spotter and DeepSpeech model. Finally, based on the sensitivity analysis, the value of the parameters were binned based on the histogram plot and hence, a scalar quantization of the values leads to decrease in the number of bits required while losing minimal accuracy.

4.1 Key-word spotter

The key-word spotter consists of 2 hidden CNN layers and a final feed-forward layer. The feedforward layer is of [62720,12] dimension. Figure 4.1 shows the distribution of the model parameters after training. The original model accuracy is 87.5%. Even after training for 18000 iterations, the final model parameter distribution remains Gaussian in nature. However, the standard deviation of the parameter distribution is less and the values are skewed

towards zero. Therefore, we hypothesize that any compression scheme which can retain this distribution would not reduce the model accuracy.

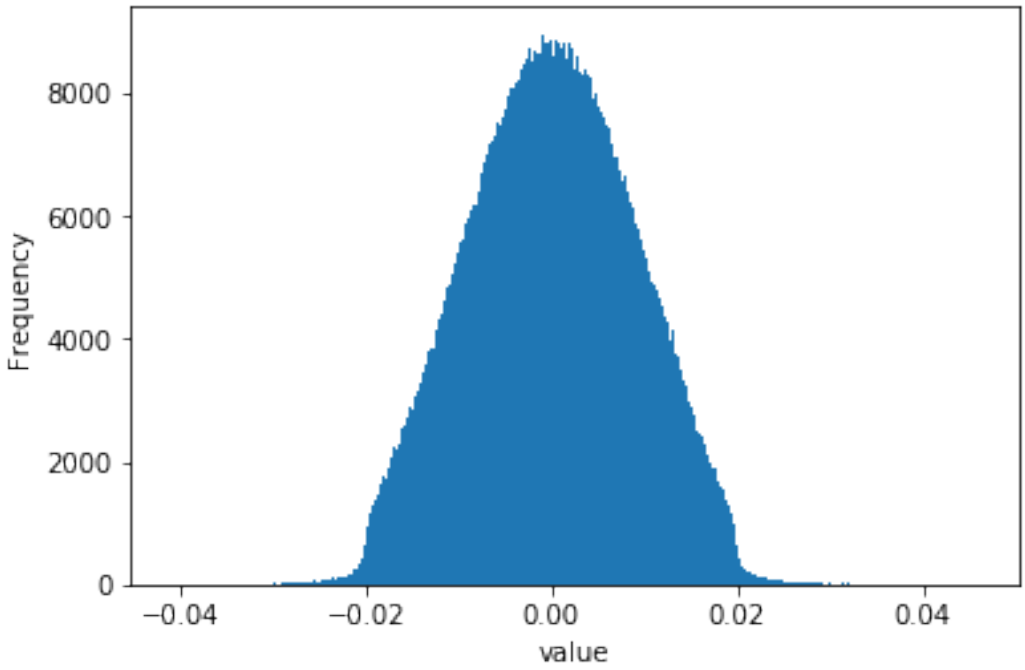

Figure 4.1: Histogram plot of final layer parameters

The maximum value of the above distribution is 0.045 and the minimum value is -0.044. The following perturbations as shown in table 4.1 were applied to the feedforward layer.

The weight values were perturbed with a percentage of itself. That is, $M = M + 0.1M$. So, 10% of its own value was added to the model parameters. However, the accuracy of the model was around 87.5%, comparable to the original model's accuracy. Therefore, this indicated that a perturba-

Variable	Perturbation	Accuracy [%]
All	±10,±20,±30,±40,±50 [%]	87.5%
All	±max_value, ±min_value	87.5%
All	Gaussian: (0,0.005)(0,0.01)(0,0.015)(0,0.02)	87.1%, 86.2%, 84.1%,82%
Row 1, Row2,..	1	8.3%

Table 4.1: The amount of perturbation vs. accuracy for feedforward layer parameters

tion value relative to the original value did not affect the performance of the model.Therefore, k-means compression technique would work the best for this model parameter.

The model parameters were also perturbed with the max_value and min_value. That is, $M = M$+max_value. Since the max_value and the min_value of the model was 0.045 and -0.044 respectively, this did not perturb the value of the model parameters by a huge volume. Therefore, the accuracy is still comparable to the original 87.5%.

Following this, the model was perturbed with gaussian of mean 0 and various standard deviations. As shown in table 4.1, with the increase in the standard deviation, the accuracy of the model decreased. As we had hypothesized initially, the training of the model decreased the standard deviation of the parameter values. The Gaussian noise with a higher standard deviation would distort the structure of the distribution and hence, the performance of the model decreases.

Finally, when a certain row of the matrix was set to 1, the accuracy

went down to 8.3%. However, most predictions were driven towards class 1 if the 1st row set to 1. Therefore, a significantly high value such as 1, relative to the other values in the model parameters drives the prediction towards that class achieving an accuracy of 8.3%.

Motivated by the fact that perturbation of the parameter by a fraction of itself did not lead to a drop in accuracy, we performed scalar quantization on the model parameters.

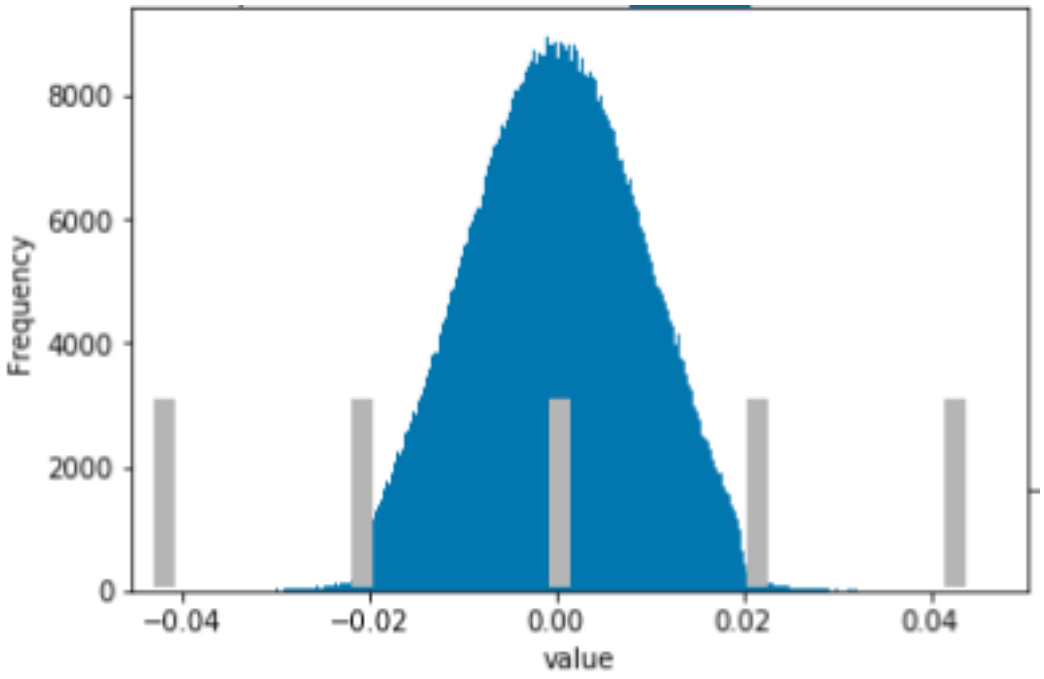

Figure 4.2: Histogram plot with 4 bins of final layer parameters

As shown in Figure 4.2, 4 bins were chosen from the parameter distribution. All the values in the distribution were replaced by the mean value of the

bin's starting and ending value. Therefore, 752640 parameters were replaced with 4 distinct values. The accuracy after binning quantization was 87.1%. Therefore, instead of having 32 bits to represent each value of the parameter, it can be represented with only 2 bits. However, these 4 unique values can be mapped to 2 bits. Also, we performed another scalar quantization with 6 bins, retaining 6 unique values. However, this was asymetric quantization, where, the bins were concentrated towards zero, with a finer representation. As shown in figure 4.3. This method gave an accuracy of 87.3%. Unlike the regular quantization technique of decreasing the number of bits and performing inference on the quantized parameter, with this technique, the floating point number can be stored as 2-bit values while, at inference, these can be mapped back to the floating point number using a dictionary mapping. Hence, this would retain a much higher accuracy than the quantization of the parameter values directly.

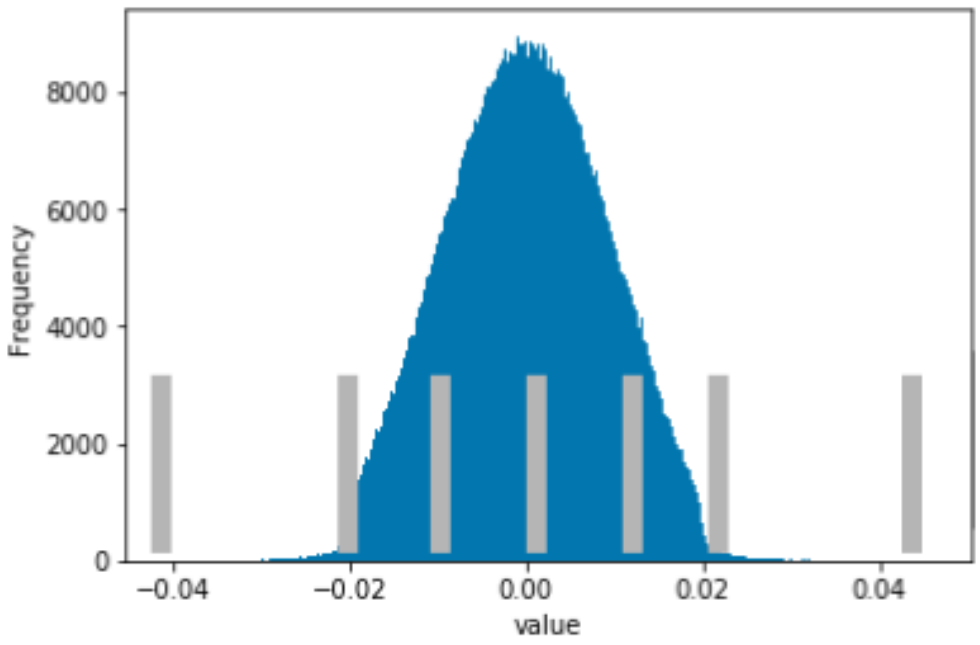

Figure 4.3: Histogram plot with 6 bins of final layer parameters

4.2 DeepSpeech

The DeepSpeech model consists of 3 linear layers followed by a LSTM forward, backward layer and then a linear layer. The LSTM consists of the maximum number of model parameters. The dimension of this layer's model parameter is [4096,8192]. The word error rate of the original model is 18% and the character error rate is 9%. The max_value is 3.24 and the min_value is -4.03. The distribution of the model's parameter value after training is Gaussian with a low standard deviation, as shown in figure 4.4. Similar to the key-word spotter distribution, most values in the layer are skewed towards 0.

Forming a very narrow distribution.

We hypothesize that such a network would be very sensitive to even a small perturbation of the values. Unlike the previous case, the perturbation of the values with the max_value and min_value would drastically shift the mean of the distribution and would affect the network performance.

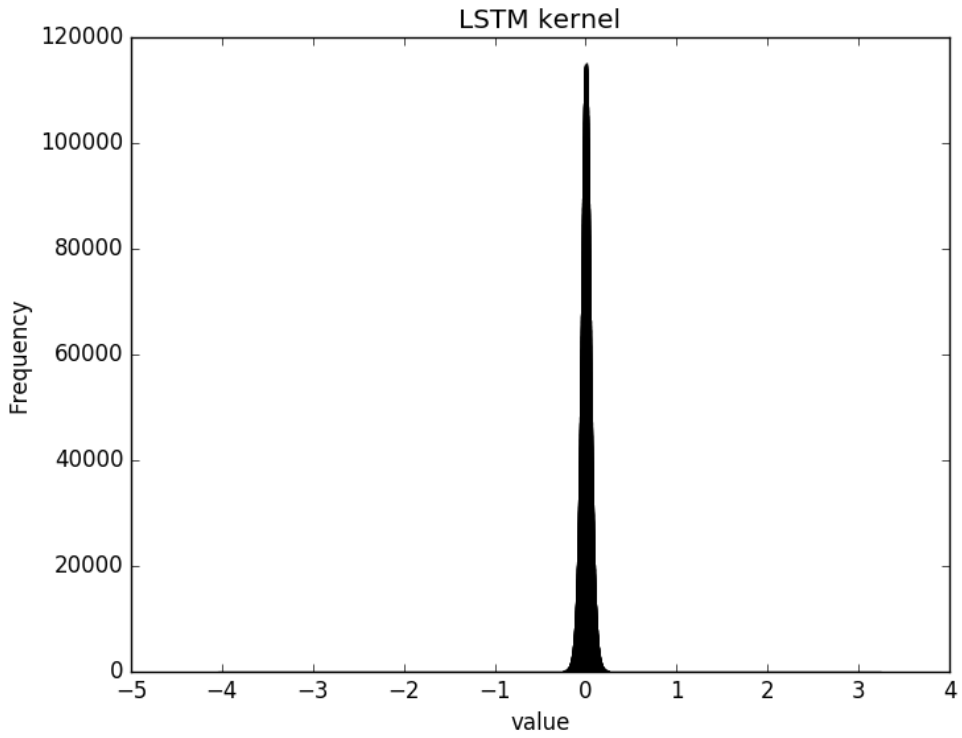

Figure 4.4: Histogram plot of LSTM layer parameters

Similar to the key-word spotter model, the parameters were perturbed with a known value(min and max value), a fraction of itself, followed by Gaus-

Variable	Perturbation	WER/CER [%]
All	±10 [%]	(19.84,9.78;18.57,9.35)
All	±20 [%]	(19.84,9.78;18.57,9.35)
All	±30 [%]	(24.05,11.91;33.53,22.15)
All	±40 [%]	(26.86,13.48;68.60,58.05)
All	±50[%]	(29.73,15.06;92.85,88.02)
All	±max_value, ±min_value	100, 100
All	Gaussian: (0,0.025)(0,0.075)	(23.51,12.53),(95.90,91.31)
All	Gaussian: (0,0.125)(0,0.25)	(98.17,98.97)(98.70,100)

Table 4.2: The amount of perturbation vs. accuracy for LSTM layer parameters

sian noise.

As shown in table 4.2, with an increase in the perturbation of the parameter value by itself, the word error rate and character error rate increases. Unlike the previous key-word spotter model, this model is not robust to most perturbations. A max_value and min_value perturbation lead to a character error rate and word error rate of 100%. As shown in the figure 4.5 and figure 4.6 , perturbation with the max_value or min_value perturbs the mean of the distribution by a large fraction. Therefore, the model is not robust to a perturbation of high value. As shown in the histogram plot of parameter distribution, the values are concentrated towards zero. The Gaussian perturbation with a standard deviation of 0.025 has an accuracy comparable to the original model accuracy since the parameter distribution is still skewed towards 0, as shown

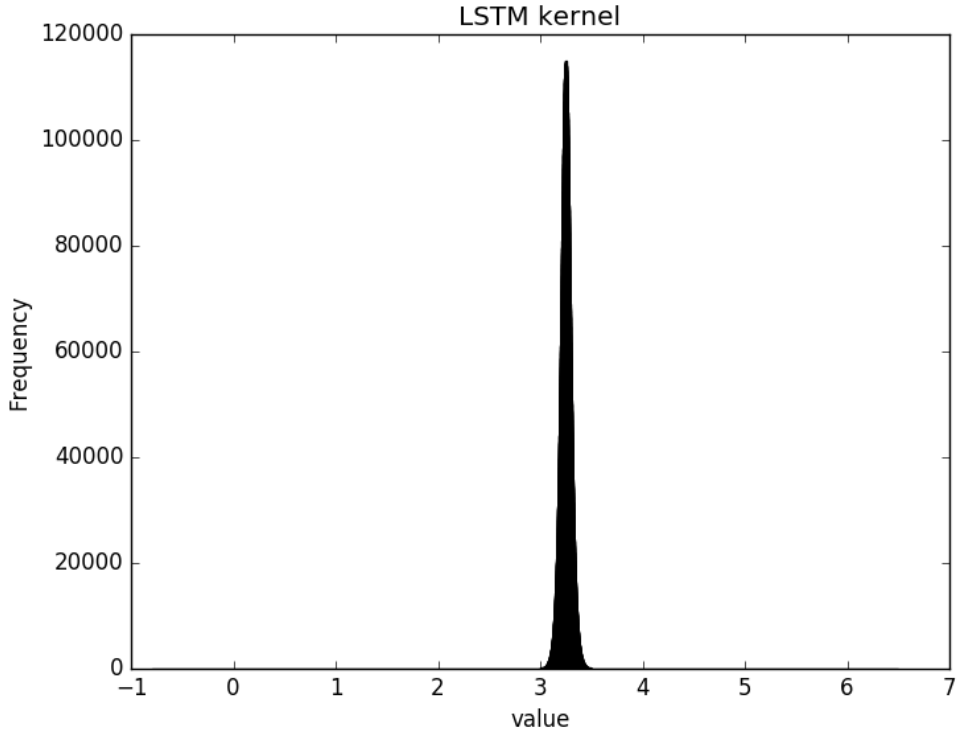

Figure 4.5: Histogram plot of perturbed LSTM layer parameters with max_value

in figure 4.7. Any perturbation above that distorts the value and the accuracy drops, increasing the character error rate and WER. The model parameters have low sensitivity because, Gaussian noise with a standard deviation of 0.75, distorts the confined standard deviation of the trained model, as shown in 4.8 and leads to a character error rate of 100% and word error rate of 100%.

Since the model parameters are very sensitive to the perturbation of a fraction of itself, a finer histogram binning was used for the scalar quantization

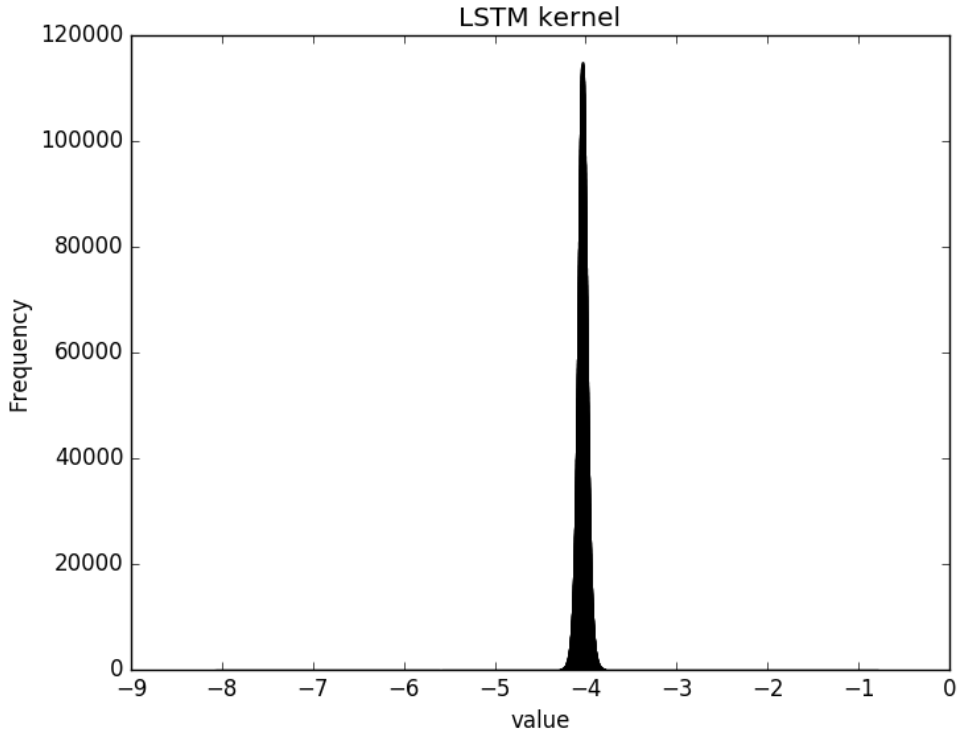

Figure 4.6: Histogram plot of perturbed LSTM layer parameters with min_value

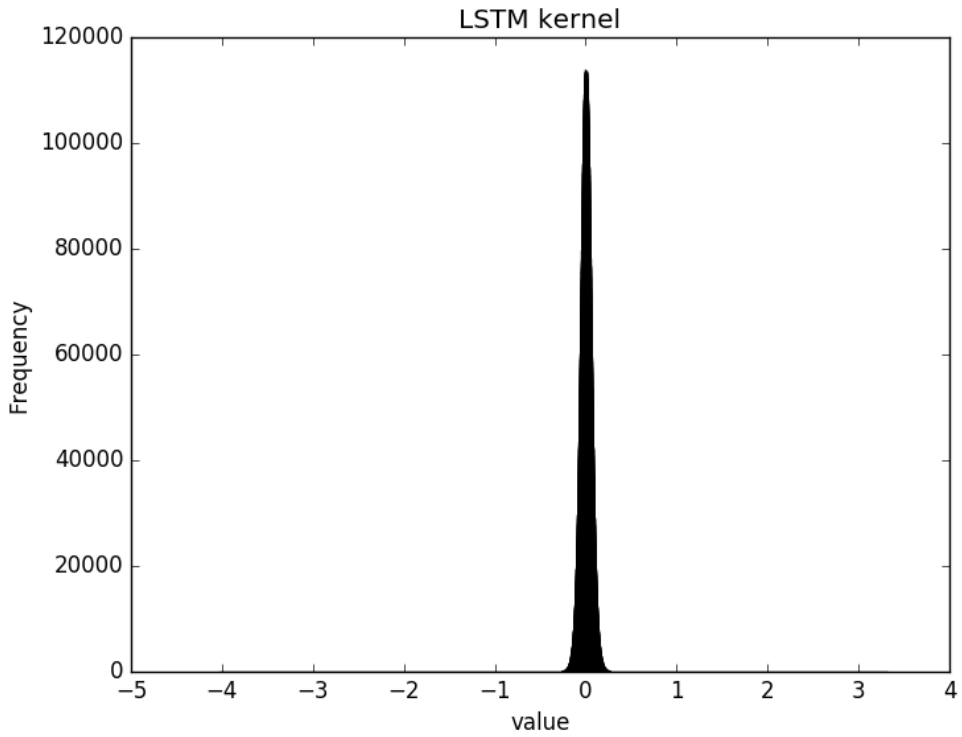

Figure 4.7: Histogram plot of perturbed LSTM layer parameters with gaussian noise of mean 0 and standard deviation 0.025

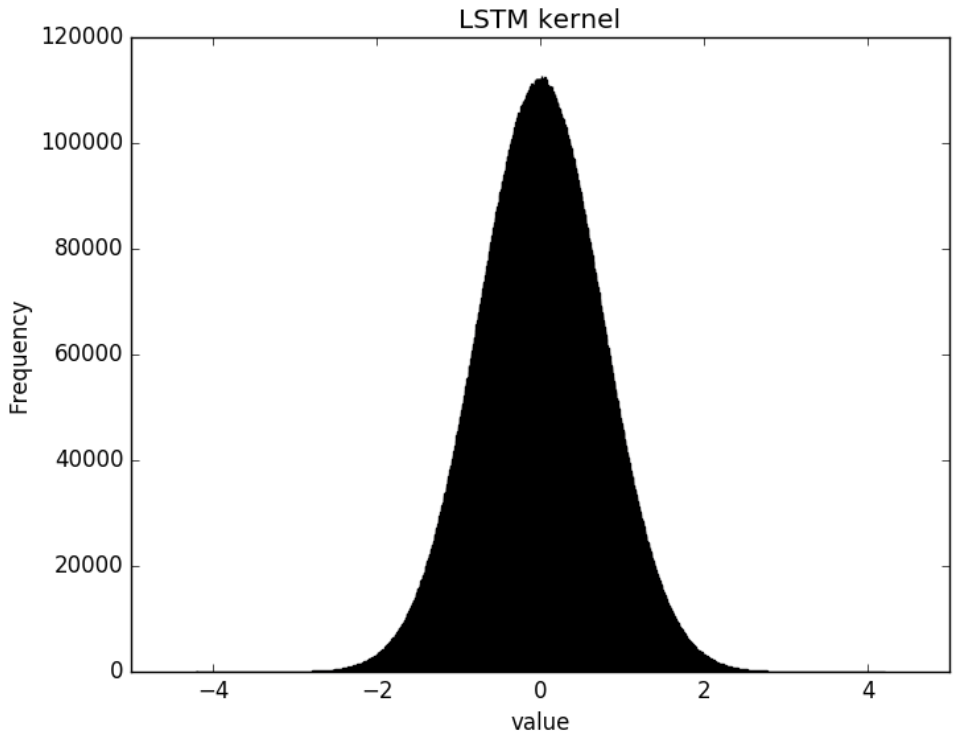

Figure 4.8: Histogram plot of perturbed LSTM layer parameters with gaussian noise of mean 0 and standard deviation of 0.75

of this model parameter. As shown in figure 4.9, after scalar quantization, the parameter values were replaced with 12 unique values. The word error rate for the quantized model is 28.34% and the character error rate is 15.27%. Therefore, using scalar quantization, without much loss in model performance, the size of the model can be reduced. The bins were chosen in such a way that at the concentrated portions of the distribution, finer quantization was performed. Whereas, at the tail of the distribution, only one bin was allocated. In this case, a bin was allocated for values between -4 and -1. Similarly, a bin was allocated between 1 and 3.5. Therefore, this manual allocation of bins based on distribution can be automated using the histogram plot of having finer bins at a region corresponding to 60 - 70% of the parameters.

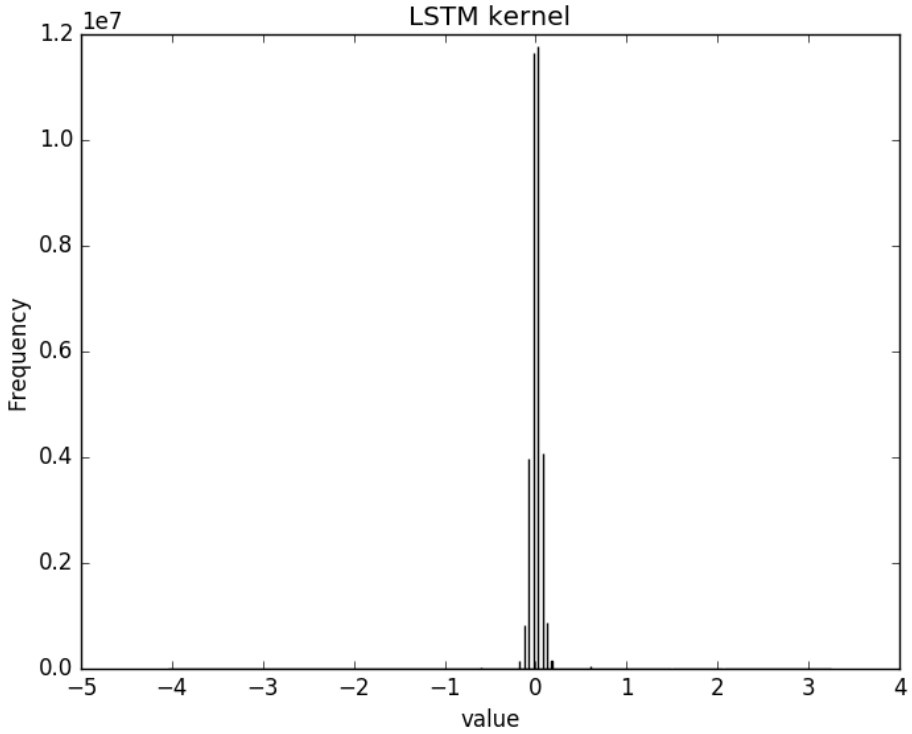

Figure 4.9: The quantized value for LSTM layer parameters

Therefore, to deploy our scalar quantization method to an arbitrary model, the distribution of the model parameter should be determined. Followed by that, at the most concentrated parts of the distribution, finer bins can be allocated. At the least concentrated parts of the distribution, such as the tail of the distribution, coarser bins can be allocated. This is an asymmetric scalar quantization which could be modified based on the network parameter being quantized. However, to determine the concentration of the distribution, the histogram of the values should be plotted. Also, this distribution should be

in such a way that it is concentrated around the center. If it is a multi-modal distribution, this can also be handled by having finer bins at the modes and coarser bins at the other parts of the distribution.

Chapter 5

Experiments

In this section, the experimental results with svd compression, k-means compression and a two-stage svd & k-means clustering is reported for keyword spotter and DeepSpeech speech recognition models.

5.1 Keyword spotter

The keyword spotter consist of two hidden layers followed by a final feed forward layer. The number of model parameters is shown in table 5.1. In this model architecture, the feedforward layer occupies the maximum memory. Therefore, the compression techniques are applied to the parameters of this layer. The original model accuracy is 87.5% and the model size of 3.7MB.

5.1.1 SVD compression

Singular value decomposition is applied to the feedfoward layer of the keyword spotter model. However, the number of components is limited 12, the rank of the matrix. Therefore, 5,6,7 were chosen to be the number of components for SVD. As the number of components is increased, the number of model parameters stored also increased. For 7 components, we obtain the

Variable	Dimension	Elements
First layer weights	[20,8,1,64]	10240
First layer bias	[64]	64
Second layer weights	[10,4,64,64]	163840
Second layer bias	[64]	64
Final layer weights	**[62720,12]**	**752640**
Final layer bias	[12]	12

Table 5.1: The number of parameters across the layers of a keyword spotter is shown in the above table

n-components	Parameters	Size benefit	Accuracy	reduction in size[%]	reduction in accuracy[%]
5	313665	1.97 MB	69.6%	46.7%	20.57%
6	376398	2.22 MB	76.3%	40%	12.8%
7	439131	2.45 MB	79.1%	33.7%	9.6%

Table 5.2: SVD compression on Keyword spotter model

maximum accuracy of 79.1% with a size of 2.45MB. When the number of components is decreased to 5, the model size is 1.97 MB, leading to a size reduction of 46.7%. Therefore, with the increase in n-components, accuracy increased. However, the reduction in size decreases. The number of parameters for 5 components is 313665 compared to the original 752640. This is more than 2 times reduction in the number of parameters of this layer. For 6 components, the size is decreased to 2.22 MB from 3.7 MB whereas, the final accuracy is 76.3%. Therefore, the best utilization of SVD is achieved for a number of components of 6 with a 40% reduction in model size and 76.3% accuracy.

5.1.2 K-means & Huffman compression

The number clusters for k-means clustering is chosen to be 128, 256 and 512. As the number of clusters increased, model accuracy increased along with model size. But, for a cluster size of 128, there is a size reduction benefit of 58.3% while the reduction in the size of only 3.4%. However, with 512 clusters, there is an incremental increase in accuracy while the reduction in size decreased 52.9%. Therefore, for k-means clustering, the optimal cluster size is 128. The increase in the number of clusters gave an incremental increase in the accuracy while leading to decrease in size reduction. For 512 clusters, the final model size is 1.74 MB compared to the original model size of 3.7 MB. Whereas, the model accuracy is 85.2% which is comparable to the original model accuracy of 87.5%.

The k-means clustering generated labels and cluster centroids. The labels are integers in the range of 1 to n clusters. Huffman compression is applied to the labels to further reduce the size of storage of these clusters and labels. After k-means clustering, as the second stage of compression, Huffman coding decreases the model size from 1.54MB to 1.07 MB. However, this two stage compression is effective only for a matrix of NXM, $N >> M$. For 128 clusters, this two-stage compression technique yields a model size of 1.07 MB while the accuracy is at 84.5%. This is a 0.43 MB drop in model size for no drop in accuracy using Huffman coding. Similarly, for 256 components, the two-stage compression technique yields a model size of 1.09 MB for accuracy of 85.1%. Therefore, the two-stage compression technique is most effective for

n-clusters	Size benefit	Accuracy	reduction in size[%]	reduction in accuracy[%]	Huffman size
128	1.54 MB	84.5 %	58.3%	3.4%	1.07 MB
256	1.63 MB	85.1 %	55.9%	2.7%	1.09 MB
512	1.74 MB	85.2%	52.9 %	2.6%	1.12 MB

Table 5.3: K-means & Huffman compression on keyword spotter model

Variable	Dimension	Elements
h1	[498,2048]	1019904
b1	[2048]	2048
h2,h3,h5	[2048,2048]	12582912
b2,b3,b5	[2048]	6144
LSTM_fused_cell-kernel	[4096,8192]	33554432
LSTM_fused_cell-bias	8192	8192
h6	[2048,29]	59392
b1	[29]	29

Table 5.4: The number of parameters across the layers of a DeepSpeech model is shown in the above table

the keyword spotter algorithm. Also, based on the analysis, we conclude that k-means clustering compression is a better technique for feedforward layer.

5.2 DeepSpeech

The Deepspeech model has 5 hidden layers and final feedforward classification layer. The first 4 hidden layers are linear layers and the fifth layer is an LSTM forward-backward layer with the maximum number of model parameters. The number of model parameters for each layer is shown in table 5.4. As shown in table 5.4, the LSTM forward-backward layer consists of a maximum

n-components	Parameters	Size benefit	WER,CER	reduction in size[%]	reduction in accuracy[%]
500	6144500	79 MB	25%,13%	58%	8.5%
750	9216750	91.5 MB	21%/10%	51%	3.65%
1000	12289000	103.8 MB	19%/9%	45%	1.21%

Table 5.5: SVD compression on DeepSpeech model model

number of model parameters. Therefore, compression techniques are applied to this layer. The model size is 188.89 MB. The character error rate (CER) is 18% on Librispeech[15] test dataset. The word error rate (WER) is 9%.

5.2.1 SVD compression

The Singular value decomposition compression is applied to the layer of size [4092,8192]. When the number of components is chosen as 500, the total number of parameter of this layer is reduced to 6144500 with a final size of 79 MB. This amounts to a 58% decrease in size for only 8.5% reduction in WER. As the number of components is increased, the size of the model increased while the reduction in accuracy decreases. For 750 components, the final model size is 91.5 MB with a reduction in accuracy of 3.65%. This accounts for a size reduction benefit of 51%. For DeepSpeech model, optimal compression using SVD is obtained using 1000 components. Whereas, the reduction in accuracy is only 1.21% whereas, the size is reduced by 45%. The number of components is restricted by the rank of the matrix.

5.2.2 K-means & Huffman compression

The two-stage compression scheme of K-means and Huffman compression is applied to the LSTM forward/backward layer of the Deepspeech model. As shown in table 5.6, the accuracy of the original model is maintained for a cluster size of 3500. However, there is only a 10.52% reduction in size. As the number of clusters is decreased to 2048, there is a size reduction benefit of 35.94%. However, there is huge reduction of accuracy of 70.73%. For a cluster size of 3000, the final model size is 153 MB, while the reduction in accuracy is only 9.75%. While the reduction in size is 19% which is twice the reduction in accuracy. The further Huffman compression of the labels generated by k-means compression decreased the size of the final model by a few thousand Bytes. Hence, the same model size is reported.

The k-means clustering is effective for a NXM matrix where the size of N is higher than M. Whereas, svd is most effective when the size of rows and columns of the matrix are comparable. Also, Huffman coding is most effective when the size of N is higher. In the case of keyword spotter mode, the size of N was 62720. Whereas, in the case of DeepSpeech model, the size of N is only 4096. Therefore, the benefit of Huffman coding is only incremental in saving storage requirements.

Therefore, for DeepSpeech model, the most effective compression technique is Singular value decomposition technique. This is mainly because the matrix is sensitive to perturbations more than the keyword spotter model and the size of the n,m dimensions of the matrix is almost similar compared to the

n-clusters	Size benefit	WER,CER	reduction in size[%]	reduction in accuracy[%]	Huffman size
2048	121 MB	76% ,58%	35.94%	70.73%	121 MB
3000	153 MB	26%, 14%	19%	9.75%	153 MB
3500	169 MB	20% ,10%	10.52%	2.43%	169 MB

Table 5.6: K-means & Huffman compression on Deep Speech model

feedforward layer of the keyword spotter of size [62740,12]. From the above analysis, we could conclude that, svd would perform better when the size of n,m of the matrix is almost the same. Whereas, k-means is most effective for single dimensional, long data, to identify the appropriate clusters for compression.

Chapter 6

Limitations & Future Improvements

In this report, we proposed three compression schemes for a key-word spotter speech recognition architecture and a large vocabulary speech recognition system architecture. However, the proposed k-means, svd and two-stage: k-means and Huffman compression schemes were not effective on both the models. In the case of key-word spotter architecture, the most effective method was k-means and Huffman compression scheme. Whereas, for DeepSpeech model, the most effective compression scheme was svd based compression.

Therefore, for an arbitrary model, it is imperative to apply all the compression schemes and then pick the best based on model robustness and size reduction. The proposed compression scheme, on the hardware level, helps in reducing the model size only at the flash memory level. The proposed compression schemes do not reduce the memory requirements at the DDR level. One potential future work is to perform a partial decompression at the DDR and save on the memory requirement at the DDR level. However, this would involve changes of the parameters in the previous layers which may require retraining.

The sensitivity analysis proposed in this report performs thorough anal-

ysis of the amount of perturbation for the layer being compressed. However, the scalar quantization based on sensitivity analysis required manual binning of the trained network parameter to achieve performance comparable to the original model, while decreasing the number of bits required for storage. As future work, the scalar quantization technique can be automated based on the concentration of the network parameters. That is, provide finer bins at the area of the distribution where 60-70% of the values are present and coarser bins for the rest of the distribution.

In addition to the scalar quantization, as future work, network pruning can be performed to sparsify the model and hence store a spare matrix. This would increase the model compression by multiple folds since the storage requirement of a sparse model is less compared to a full matrix.

Chapter 7

Conclusion

Speech recognition model compression is vital for deploying these models on mobile and embedded devices. The advancements in deep learning techniques have made the models more robust to various noise conditions and input conditions. Therefore, in this report, we have proposed three compression schemes. The first compression scheme is SVD, where the network parameter's layer matrix is decomposed into components. These components consume lesser space for storage compared to the original model. SVD compression scheme was most effective in the DeepSpeech mode, achieving a size reduction benefit of 45% while the performance drops only by 1.21%.

The second compression technique is k-means clustering. This method forms clusters on the input layer parameter which helps in weight sharing and hence reduction on the number of parameters. The k-means compression was most effective on the key-word spotter speech recognition model, achieving a size reduction of 58.3% while the reduction in accuracy is only 3.4%.

The third compression technique is a two-stage k-means and Huffman compression. The labels generated by k-means are further reduced in size using Huffman coding. This technique is most effective on the keyword spotter

recognition system with an accuracy of 84.5% compared to the original 87.5% while the final size of the model is 1.07 MB.

The sensitivity analysis helped in understanding the amount of perturbation the model parameters can handle while retaining the model performance. Based on the sensitivity analysis, a scalar quantization based compression scheme is proposed to reduce the number of bits required to represent the values from 32 to 4.

Therefore, to achieve the best compression while maintaining the model accuracy, the layer with the maximum number of parameters were chosen and the above-mentioned compression schemes were applied.

Bibliography

[1] K. Markov and T. Matsui, "Robust speech recognition using generalized distillation framework," pp. 2364–2368, 09 2016.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, pp. 84–90, 2017. [Online]. Available: http://doi.acm.org/10.1145/3065386

[3] J. D. Geoffrey Hinton, Oriol Vinyals, "Distilling the Knowledge in a Neural Network." NIPS 2014 Deep Learning Workshop, 2015.

[4] A. B. I. M. Rohit Prabhavalkar, Ouais Alsharif, "ON THE COMPRESSION OF RECURRENT NEURAL NETWORKS WITH AN APPLICATION TO LVCSR ACOUSTIC MODELING FOR EMBEDDED SPEECH RECOGNITION." ICASSP 2016, 2016.

[5] R. A. M. G. A. K. R. D. R. O. A. H. S. A. G. F. B. C. P. Ian McGraw, Rohit Prabhavalkar, "Personalized Speech recognition on mobile devices." ICASSP 2016, 2016.

[6] R. P. S. G. Y. W. S. Z. C.-c. C. Ruoming Pang, Tara N. Sainath, "Compression of End-to-End Models." Interspeech 2018, 2019.

[7] S. X. P. G. B. X. Chenxing Li, Lei Zhu, "Compression of Acoustic Model via Knowledge Distillation and Pruning." ICPR 2018, 2018.

[8] S. N. Sanghyun Son and K. M. Lee, "Clustering Convolutional Kernels to Compress Deep Neural Networks." ECCV 2018, 2018.

[9] W. J. D. Song Han, Huizi Mao, "DEEP COMPRESSION: COMPRESSING DEEP NEURAL NETWORKS WITH PRUNING, TRAINED QUANTIZATION AND HUFFMAN CODING." ICLR 2016, 2016.

[10] P. Warden, "Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition." arXiv, 2018. [Online]. Available: https://arxiv.org/abs/1804.03209

[11] J. C. B. C. G. D. E. E. R. P.-S. S. S. S. A. C. A. Y. N. Awni Hannun, Carl Case, "Deep speech: Scaling up end-to-end speech recognition." arXiv, 2014. [Online]. Available: https://arxiv.org/abs/1412.5567

[12] T. N. Sainath and C. Parada, "Convolutional neural networks for small-footprint keyword spotting," in *INTERSPEECH*, 2015.

[13] C. Cieri, D. Miller, and K. Walker, "The fisher corpus: a resource for the next generations of speech-to-text," in *LREC*, 2004.

[14] J. M. J.J. Godfrey, E.C. Holliman, "Switchboard: telephone speech corpus for research and development." IEEE, 1992.

[15] D. P. S. K. Vassil Panayotov, Guoguo Chen, "Librispeech: An asr corpus based on public domain audio books." IEEE, 2015.

[16] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber, "Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks," in *Proceedings of the 23rd international conference on Machine learning*. ACM, 2006, pp. 369–376.

[17] A. Graves and N. Jaitly, "Towards end-to-end speech recognition with recurrent neural networks," in *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ser. ICML'14. JMLR.org, 2014, pp. II–1764–II–1772. [Online]. Available: http://dl.acm.org/citation.cfm?id=3044805.3045089

[18] M. N. Bourlard H., "Connectionist speech recognition: A hybrid approach." 1993.

[19] H. Soltau, H. Liao, and H. Sak, "Neural speech recognizer: Acoustic-to-word LSTM model for large vocabulary speech recognition," *CoRR*, vol. abs/1610.09975, 2016. [Online]. Available: http://arxiv.org/abs/1610.09975