Model Compression

ANKITA GUPTA

M.Tech

Supervisor: Prof. Kolin Paul

A thesis submitted in fulfilment of
the requirements for the degree of
Master of Technology

Computer Science
Indian Institute Of Technology
Delhi

3 January 2025

Abstract

Deep neural networks (DNNs) have achieved great success in a number of areas. However, existing deep neural network models are computationally expensive and memory intensive, hindering their deployment in devices with low memory resources or in applications with strict latency requirements. Therefore, a natural thought is to perform model compression and acceleration in deep networks without significantly decreasing the model performance.

This project built & train speech recognition models on Google Speech dataset and then aim to compress it using several weight compression techniques. It covers two of the most popular weight compression techniques- Quantization Aware Training and Post Training quantization which involves clipping clustering and float to fixed point conversion. While performing Quantization aware training a concept called transfer learning is also introduced and how to use this concept in our way towards model compression is discussed . Moreover, All these schemes are applied on the built speech recognition model and the complete analysis regarding the performance, advantages, and drawbacks is provided.

Acknowledgements

Foremost, I would like to express my sincere gratitude to my supervisor Prof. Kolin Paul for his valuable advice and guidance throughout the project. It would be impossible to produce this work without his constant support. In addition, I would like to thank the college IIT Delhi for providing excellent facilities to support the project.
Lastly, I would like to thank my family and friends for their love and support.

Author's declaration

I declare that the work in this dissertation was carried out in accordance with the requirements of the University's Regulations and Code of Practice for Research Degree Programmes and that it has not been submitted for any other academic award. Except where indicated by specific reference in the text, the work is the candidate's own work. Work done in collaboration with, or with the assistance of, others, is indicated as such. Any views expressed in the dissertation are those of the author.

SIGNED: .... DATE: ....

Certificate

This is to certify that the thesis titled “Model Compression” being submitted by **Ankita Gupta** for the award of **Master of Technology in Computer Science and Engineering** is a record of bonafide work carried out by her under my guidance and supervision at the Department of Computer Science and Engineering. The work presented in this thesis has not been submitted elsewhere either in part or in full for the award of any other degree or diploma.

Prof. Kolin Paul
Department of Computer Science and Engineering
INDIAN INSTITUTE OF TECHNOLOGY, DELHI

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

1.1 Background

In recent years, deep neural networks have been applied to different applications and achieved drastic accuracy improvements in many tasks. These works rely on deep networks with millions or even billions of parameters, and the availability of GPUs with very high computation capability plays a key role in their success.
Existing DNN models are computationally expensive and memory intensive, hindering their deployment in devices with low memory resources or in applications with strict latency requirements. Therefore, a natural thought is to perform model compression without significantly decreasing the model performance.
By model Compression the aim is to reduce the size of the model and get inference time speedup and that too without compromising its accuracy. The approaches of the model Compression is divided into :

- Architectural Compression (Pruning)
- Weight Compression (Quantization, Clustering)

This project primarily focus on weight compression techniques.

1.2 Motivation

The main motivation behind Model Compression is:

- **Hardware Constraints** : Smaller models occupy less storage space. Also smaller models require less time and bandwidth to download the model.
- **Faster Execution** : Smaller models use less RAM when they are run, thus can translate to better performance and stability.

1.3 Thesis Overview

In this project two different types of Speech recognition models (CNN and LSTM) are built on Google Speech dataset and then the aim is to compress it using several weight compression techniques. The work covered is divided into three major sections. First section covers the methodology to train two different speech recognition models on google Speech dataset and then comparing accuracy of the two when trained in full precision. In the comming sections we aim to compress these models. In section 2, a methodology for training deep neural networks using half-precision floating point numbers is covered. The aim is to do it without losing model accuracy. In here, the concept of transfer learning is also introduced and how to use this concept in our way towards model compression is discussed. Last Section covers several post training Compression techniques which include clipping, clustering, algorithms to convert float point to fixed point. The aim is to now compress the model weights even more say 8bits or less while preserving its accuracy. Atlast this paper talks about how these techniques can result in inference time speedup. This is done using tensorflow LITE toolkit.
Moreover, A complete analysis of the performance of our speech recognition model on applying these techniques is done.

CHAPTER 2

Dataset

This section is divided into two subsections

- Understanding dataset
- Data Preprocessing

2.1 Understanding Dataset

The dataset used is Speech command dataset released by TensorFlow in 2017. It includes thousands of one-second long utterances of command words, by thousands of different people. These words are one syllable words. The dataset is designed to let one build basic but useful voice interfaces for applications, with common words which includes “Yes” ,”No”, ”Left” ,”Right” ,”Up” ,”Down” ,”Stop” ,”Go” ,”On” ,”Off”.
Dataset can be downloaded from :
http://download.tensorflow.org/data/speech_commands_v0.01.tar.gz

2.2 Data Preprocessing

The dataset consists of .wav audio files, the duration of a recordings is less than or equal to 1 second(mostly 1 sec) and the sampling rate is 16k Hz. In order to feed this data into a model some preprocessing is neeeded . This is going to lower the memory requirements. It's also going to lower the time needed for model's parameters to converge to the ones that work well. The Preprocessing applied is :

(1) Audio file is resampled to 8k hz. This is done using : `torchaudio.transforms.Resample(actual,desired).` So the input feature vector becomes of dim (1,8000)
(2) For LSTM model the further preprocessing of the input vector is done using MFCC feature extraction and is converted into (81, 12) dimension. This is done using pytorch library.
(3) Handling shorter commands of less than 1 second - This is done by padding zeros.
(4) Output labels are converted to encoded integers and then further these integer encoded labels are converted to a one-hot encoded vector as it is a multi-class classification problem.
(5) Atlast the dataset is splitted into training, test and validation set - data is split as 80% train set, 10% validation set and the rest 10% as test set.

CHAPTER 3

Models

In this chapter two different models on google speech dataset are built :

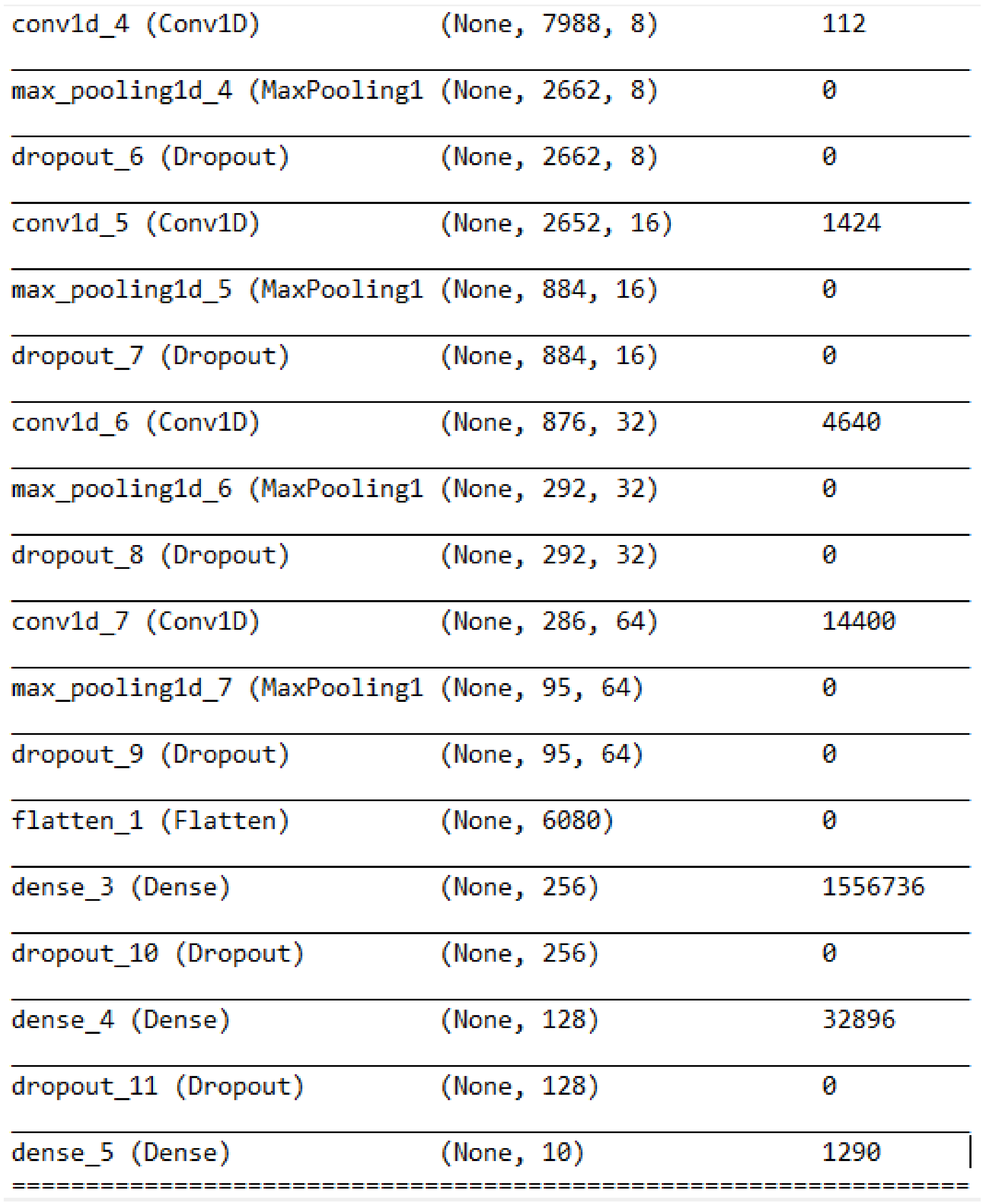
3.1 CNN Model

Convolutional neural networks (CNN) is a special architecture of artificial neural networks.These networks are formed by layers in which a bank of filters is convoluted with the input map to produce an output map which is further processed with a non-linear activation function. CNN uses some features of the visual cortex. One of the most popular uses of this architecture is image classification

3.1.1 Model Architecture

keras API is being used to build the model. Model Used is the common CNN model [9]. The overview of architecture of the model used is -

- It consists of 4 convolution layers with maxpool and dropout layer with dropout 0.3 . Activation function used is ‘relu’ .
- Atlast, it consists of 3 dense layers. Two of which having the ‘relu’ activation function and the last dense layer uses the ‘softmax’ activation function to finally output the class label.



```
conv1d_4 (Conv1D)            (None, 7988, 8)          112

max_pooling1d_4 (MaxPooling1 (None, 2662, 8)          0

dropout_6 (Dropout)          (None, 2662, 8)          0

conv1d_5 (Conv1D)            (None, 2652, 16)         1424

max_pooling1d_5 (MaxPooling1 (None, 884, 16)          0

dropout_7 (Dropout)          (None, 884, 16)          0

conv1d_6 (Conv1D)            (None, 876, 32)          4640

max_pooling1d_6 (MaxPooling1 (None, 292, 32)          0

dropout_8 (Dropout)          (None, 292, 32)          0

conv1d_7 (Conv1D)            (None, 286, 64)          14400

max_pooling1d_7 (MaxPooling1 (None, 95, 64)           0

dropout_9 (Dropout)          (None, 95, 64)           0

flatten_1 (Flatten)          (None, 6080)             0

dense_3 (Dense)              (None, 256)              1556736

dropout_10 (Dropout)         (None, 256)              0

dense_4 (Dense)              (None, 128)              32896

dropout_11 (Dropout)         (None, 128)              0

dense_5 (Dense)              (None, 10)               1290
=================================================================
```

FIGURE 3.1. CNN Model Architecture

3.1.2 Training

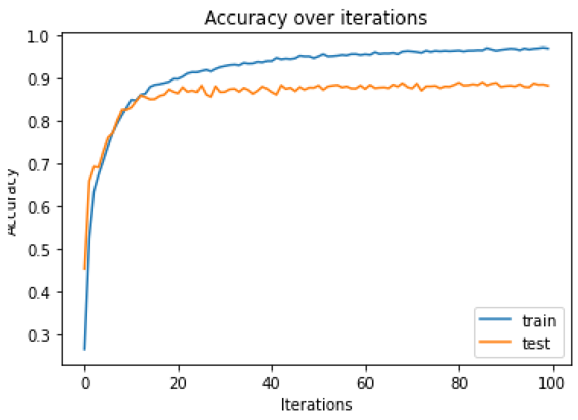
- The input feature vector is of dimension (8000,1)
- Total Trainable Parameters - 1,611,498
- Optimizer used - adam
  Adam[17] is an adaptive learning rate optimization algorithm that's been designed specifically for training deep neural networks. The algorithms leverages the power of adaptive learning rates methods to find individual learning rates for each parameter.
- Initial learning rate - 0.01
- Loss function - categorical_crossentropy
- Model is trained over the training set for 100 epochs with 100 batch size

3.1.3 Accuracy & Loss plot

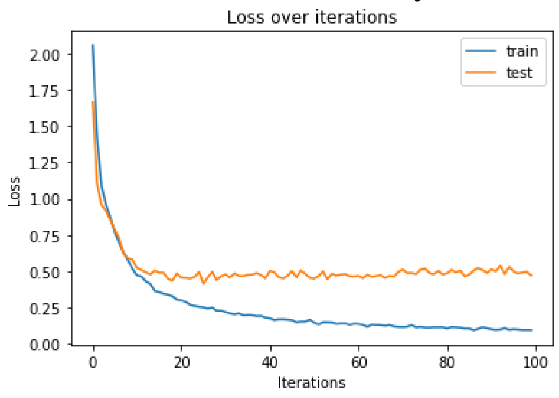
Accuracy loss statistics over training and test data is as follows -

Train Accuracy :	92.95%
Test Accuracy :	88.20%
Train Loss :	0.093
Test Loss :	0.471

The Accuracy and Loss plot of the proposed CNN model trained over Google speech dataset is shown in FIGURE 3.2



(a) CNN Model Accuracy Plot



(b) CNN Model Loss Plot

Figure 3.2. Accuracy/Loss Plot

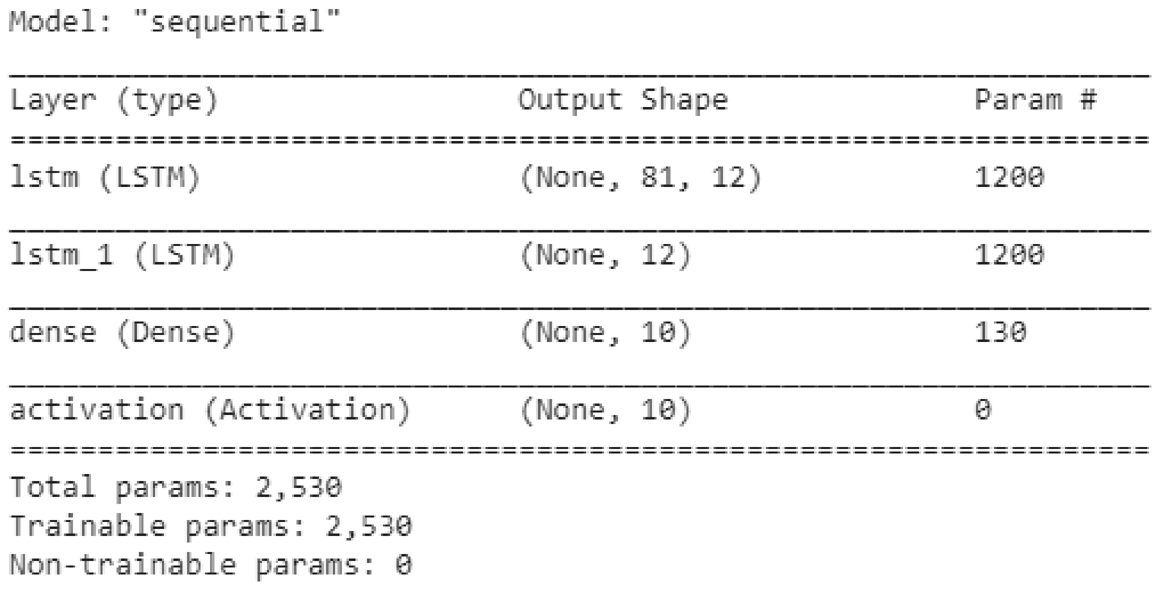
3.2 LSTM Model

LSTM stands for Long- short term memory usually just called "LSTMs" . These are special kind of RNN, capable of learning long-term dependencies. They work tremendously well on a large variety of problems especially the problem which has sequential data and speech data is one of those.

3.2.1 Model Architecture

Model is built using keras API. The overview of architecture of the model -

- It consists of LSTM layer, hyper parameters of which are as follows - hidden units = 12, Inner activation = ' sigmoid', activation = 'tanh'.
- And a Dense layer and lastly a softmax layer to finally output the class label.



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 81, 12)            1200
_________________________________________________________________
lstm_1 (LSTM)                (None, 12)                1200
_________________________________________________________________
dense (Dense)                (None, 10)                130
_________________________________________________________________
activation (Activation)      (None, 10)                0
=================================================================
Total params: 2,530
Trainable params: 2,530
Non-trainable params: 0
```

FIGURE 3.3. LSTM Model Architecture

3.2.2 Training

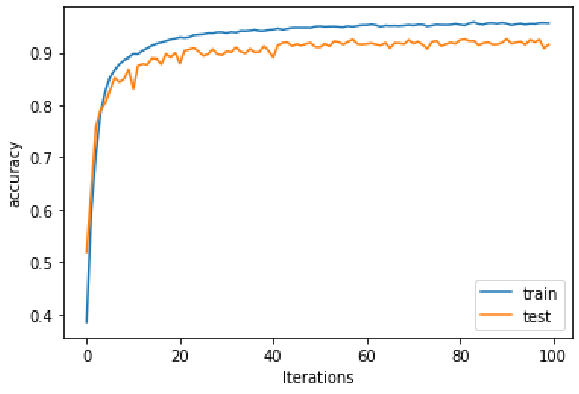
- Input feature vector is of dim (81, 12)
- Optimizer used - RMSprop with learning rate=0.01
- Loss function - categorical_crossentropy
- Model is trained over the training set for 100 epochs with 100 batch size

3.2.3 Accuracy/ Loss plot

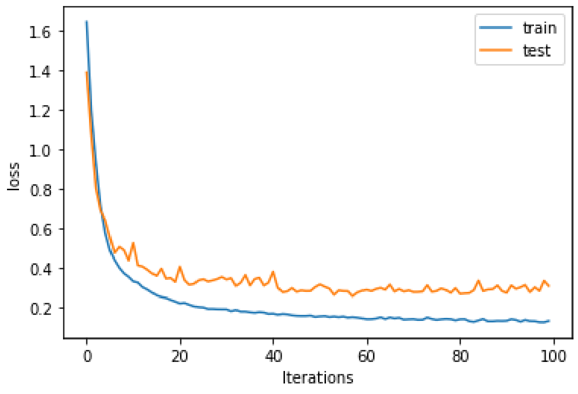
Accuracy loss statistics over training and test data is as follows -

Train Accuracy :	95.74%
Test Accuracy :	91.60%
Train Loss :	0.1331
Test Loss :	0.3101

The Accuracy and Loss plot of the LSTM model trained over Google speech dataset is shown in FIGURE 3.4



(A) LSTM Model Accuracy Plot



(B) LSTM Model Loss Plot

FIGURE 3.4. Accuracy/Loss Plot

3.3 Comparing the proposed models

LSTM model gives better accuracy as compared to CNN model. It gives 2-3% accuracy raise

Table 3.1. Model Accuracy overview

Models	**Trainset Accuracy**	**Testset Accuracy**
CNN Model	92.95%	88.20%
LSTM Model	95.74%	91.60%

CHAPTER 4

Quantization Aware Training

4.1 Overview

This chapter introduces the methodology for training our Convolution neural network model in lower precision using half-precision floating point numbers while preserving model accuracy. After this it introduces the concept of transfer learning and how to use this concept in our way towards model compression. For each scheme, insightful analysis regarding the performance, advantages, and drawbacks is done. So this section can be broadly divided into two subdomains-

- Lower Precision Training
- In cooperating Transfer Learning

Before moving further, lets visit some of the terminologies that are used in this section.

Terminologies used:-

- Half precision(FP16) - It is a binary floating-point representation format that occupies 16 bits.
- Single precision (FP32) - It occupies 32 bits and is a common/default floating point format (float in C-derived programming languages)
- Double precision(64-bit) - It occupies 64 bits.

- Mixed precision - As the name indicates it is the combined use of different numerical precisions in a computational method.

4.2 Lower Precision training

This section trains the deep learning CNN model built over google speech dataset in half precision and then introduces some techniques on achieving good performance and accuracy and then finally compare the accuracies of both the models (ie. full precision model and half precision model)

4.2.1 Advantages

Lower precision training has following advantages:-

- **Decrease the required amount of memory**
  Half-precision floating point format (FP16) uses 16 bits, compared to 32 bits for single precision (FP32). This Lowers the required memory enabling training of larger models or training with larger mini-batches[7, 8]
- **Shorten the training or inference time**
  Execution time can be sensitive to memory or arithmetic bandwidth. Half-precision halves the number of bytes accessed, thus reducing the time spent in memory-limited layers. NVIDIA GPUs offer up to 8x more half precision arithmetic throughput when compared to single precision(float32), thus speeding up math-limited layers.[8]
- **Decreased memory bandwidth**
  Requires less memory bandwidth thereby speeding up data transfer operations and lowering communication costs.

4.2.2 Training

Lower precision training offers significant computational speedup by performing operations in half-precision format, while storing minimal information in single-precision to retain as much information as possible in critical parts of the network. Since the introduction of Tensor Cores in the Volta and Turing architectures, significant training speedups are experienced by switching to mixed precision – up to 3x overall speedup on the most arithmetically intense model architectures.
The ability to train deep learning networks with lower precision was introduced in the Pascal architecture and first supported in CUDA® 8 in the NVIDIA Deep Learning SDK[7].
DNN training traditionally relied on IEEE single-precision format ie. FP32, here we will focus on training our CNN model with half precision while maintaining the network accuracy achieved with single precision.
Lower precision training requires two steps-

- Porting the model to use the FP16 data type.
- Scaling the constants to stop reaching to small values.

4.2.2.1 Challenges in Training in Lower Precision

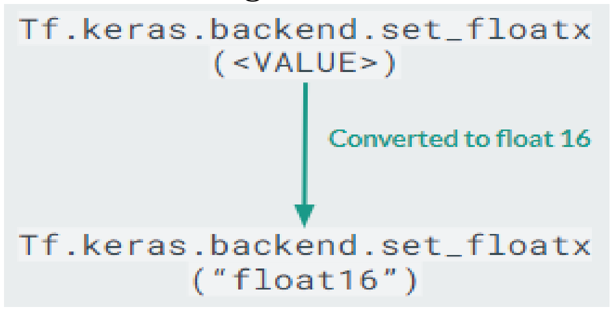
IEEE 754 standard defines the following 16-bit half-precision floating point format: 1 sign bit, 5 exponent bits, and 10 fractional bits.
Exponent is encoded with 15 as the bias, resulting [-14, 15] exponent range (two exponent values, 0 and 31, are reserved for special values). An implicit lead bit 1 is assumed for normalized values, just like in other IEEE floating point formats.
So, Compared to FP32, we have a smaller range of possible values (2e-14 to 2e15 roughly, compared to 2e-126 to 2e127 for FP32) but also a smaller offset.

For instance, between 1 and 2, the FP16 format only represents the number 1, 1+2e-10, 1+2*2e-10... which means that 1 + 0.0001 = 1 in half precision. That's what will cause a certain numbers of problems, specifically three that can occur and mess up the training.

(1) The weight update is imprecise: Inside the optimizer, we basically do w = w - lr * w.grad for each weight of our network. The problem in performing this operation in half precision is that very often, w.grad is several orders of magnitude below w and the learning rate is also small. The situation where w=1 and lr*w.grad is 0.0001 (or lower) is therefore very common, but the update doesn't do anything in those cases.
(2) Gradients can underflow. In FP16, gradients can easily be replaced by 0 because they are too low.
(3) Activations or loss can overflow. The opposite problem from the gradients: it's easier to hit nan (or infinity) in FP16 precision, and our training might more easily diverge.

Addressing the above challenges in tensorflow:-

(1) Porting the model to use the FP16 data type. This can be achieved in tensorflow using:-



```
Tf.keras.backend.set_floatx
(<VALUE>)
```

```
Tf.keras.backend.set_floatx
("float16")
```

This enable all the backend calculations to be exected in float16.
(2) Scaling constants so that their value doesnot drop below provided threshold

```
import tensorflow.keras.backend as K
K.set_epsilon(1e-4)
```

(3) Setting the optimizers: fixing optimizer to stochastic gradient descent and setting the lrate to constant.

(4) Its better to keep Input and output shape for each conv layer in multiple of 8.

4.2.3 Results

The accuracy plot for the CNN model obtained when trained in half precision is as follows:-

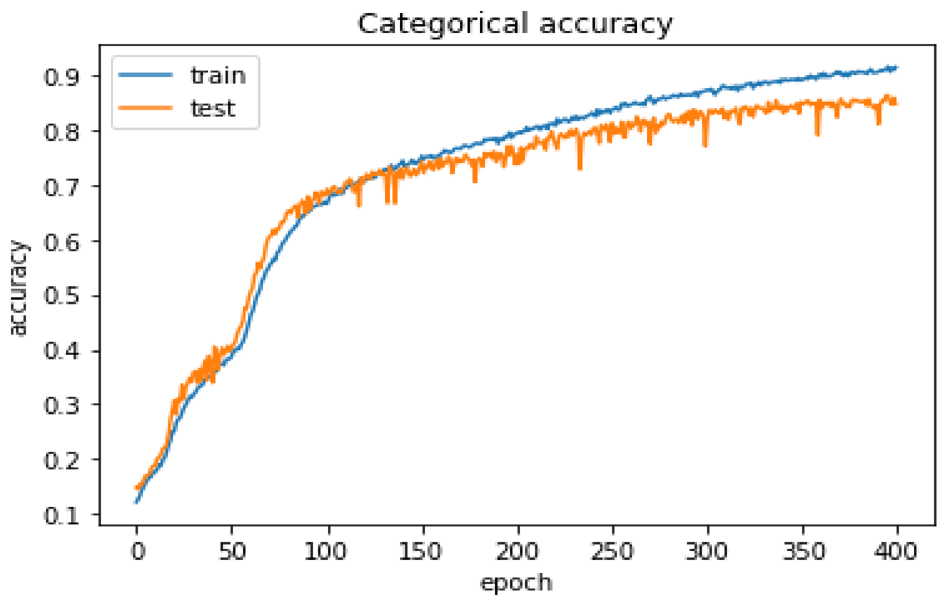


FIGURE 4.1. Accuracy Plot of Float16 model

Train set accuracy = 91.15 %

Test set accuracy = 86.12 %

4.2.4 Conclusion

Test set accuracy of full precision CNN model trained over Google speech dataset is 88.19. Training this model in half precision results in 86.12 % accuracy. which is $\sim$ 2% accuracy drop which is a marginal drop.

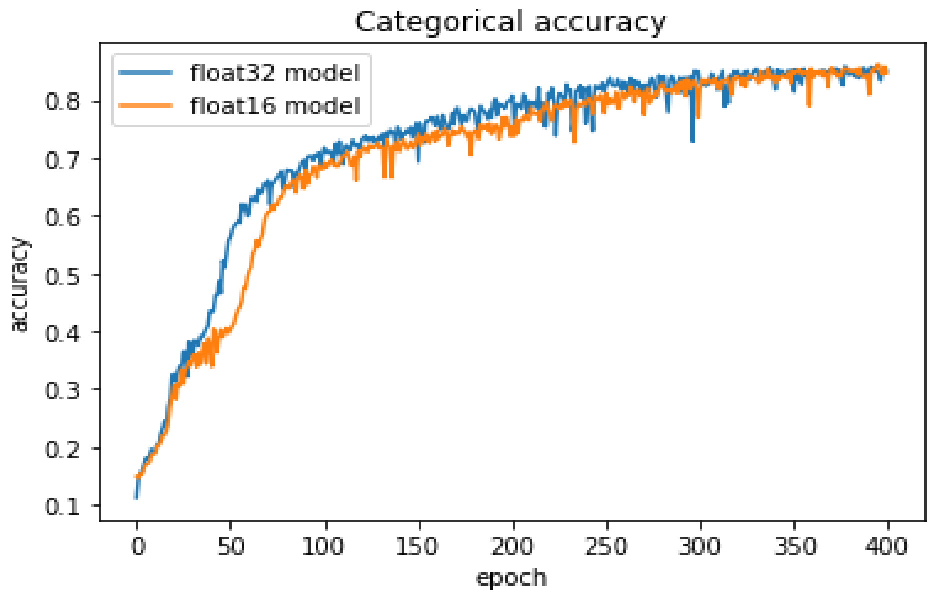


Figure 4.2. Accuracy Comparison of Float32 and Float16 model

4.3 In-cooperating transfer learning

Now, one genuine thought arises that why to train the float16 model from scratch if we have already pre trained float32 model beforehand. This will reduce the training time further. So, this pushes us towards in cooperating another interesting methodology “transfer learning”. This section finds and analyse the effect of in cooperating transfer learning in model compression. The analysis is carried on Convolution neural networks(CNN) build on google speech data set.

4.3.1 Advantages

Transfer learning is a machine learning technique where a model trained on one task is re-purposed on a second related task.

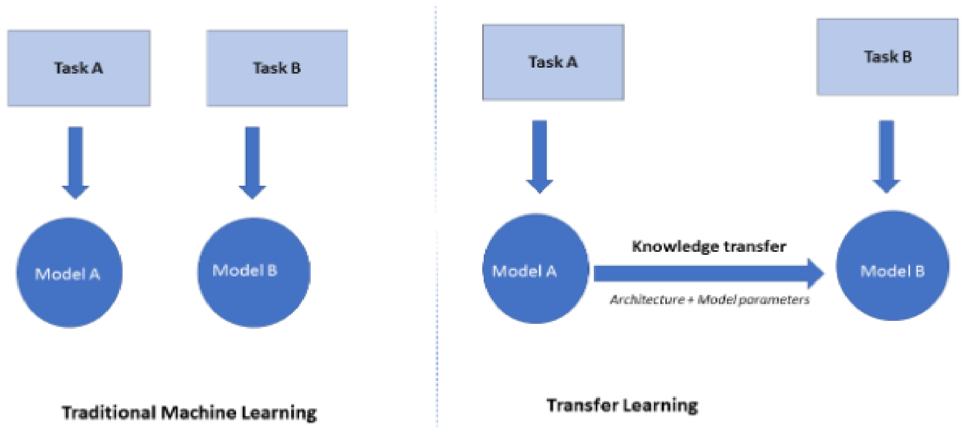


FIGURE 4.3. Transfer Learning

Transfer learning has several benefits, but the main advantages are:-

- saving training time.
- Better performance
- Not needing a lot of data.

Usually, a lot of data is needed to train a neural network from scratch but access to that data isn't always available — this is where transfer learning comes in handy. With transfer learning a solid machine learning model can be built with comparatively little training data because the model is already pre-trained. This is especially valuable in natural language processing because mostly expert knowledge is required to create large labeled datasets. Additionally, training time is reduced because it can sometimes take days or even weeks to train a deep neural network from scratch on a complex task.

4.3.2 Implementation

In here, transfer learning is in-cooperated. The general idea is to use the knowledge learned by our pretrained float 32 CNN model in the process of training of float16 CNN model. so, training of the float16 is not done from scratch instead a knoweledge base provided beforehand is used. hence, we can claim that the training time of the model going to get reduced and also lesser data is needed to reach the optimal accuracy. Hence the half precision model is retrained on different portions of training data(10% 20% 70% etc) and the results are laid.

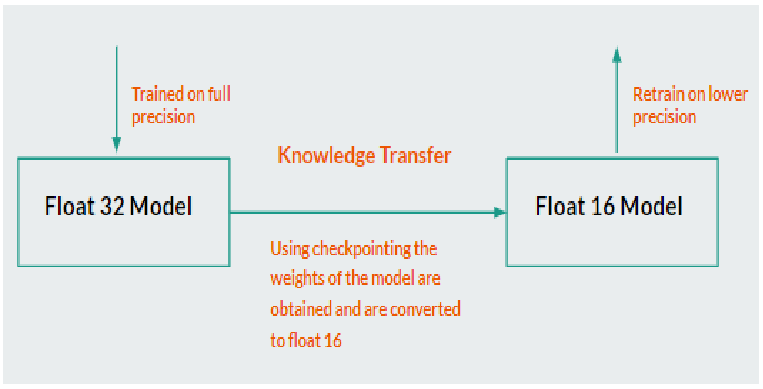

Trained on full precision
Retrain on lower precision
Knowledge Transfer
Float 32 Model
Float 16 Model
Using checkpointing the weights of the model are obtained and are converted to float 16

Figure 4.4. Complete illustration of the transfer learning procedure

(1) Consider pretrained CNN float32 model on google speech dataset.(Optimizer-SGD, Loss function- categorical_crossentropy)
(2) Convert the weights(60%) of float 32 model which are in float32 to float16 value and save them for further usage as a knowledge base for untrained float16 model.
(3) Now consider untrained float16 model with same architecture as that of a pretrained model considered in step 1 set weights those are obtained in step 2
(4) Retrain the model in half precision over different portions of the training set to obtain the the behaviour of the model while retraining.

4.3.3 Result/Observation

The results/observations are listed in table 4.1 when the half precision CNN model is retrained using full precision CNN model as a knowledge base over different percentages of the training data say 10%, 20%, 50%, 70% 80%. Figure 4.5 & Figure 4.6 shows the accuracy curve obtained when the float16 CNN model is trained completely over 50% and 70% of the data using float32 model as a knowledge base.

TABLE 4.1. Accuracy and observation when the lower precision model after using full precision model as knowledge base is retrained over different percentages of data

Retraining model over x% of data	**Accuracy**	**Observation**
10%	64.38%	Unstable Accuracy curve No convergence No improvement in accuracy
20%	67.24%	Unstable Accuracy curve No convergence No improvement in accuracy
50%	80%	Slow convergence Accuracy gained from 61% to 70%
70%	82%	Moderate convergence Accuracy gained from 61% to 82%
80%	85%	Moderate convergence Accuracy gained from 61% to 85%

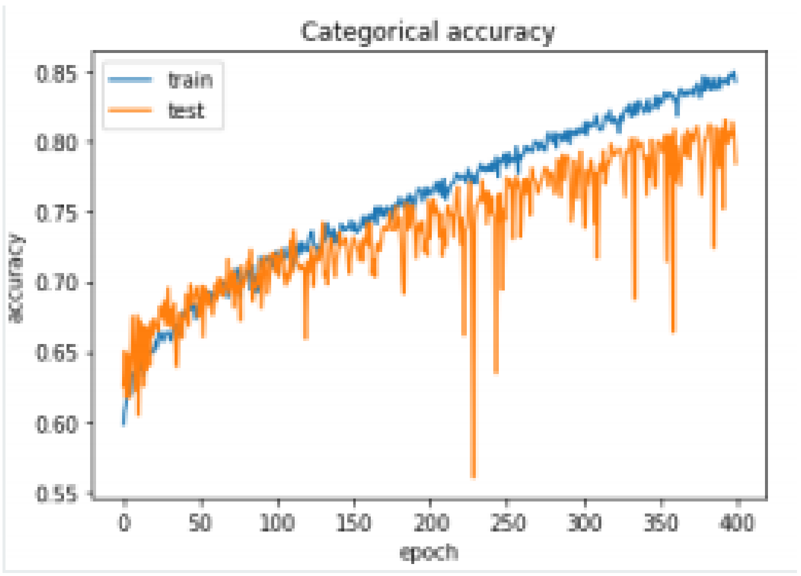


Figure 4.5. Retraining Float16 model completely over 50% of data while using FLoat32 as a knoweledge base the model get test set accuracy of 80% while the training accuracy reaches 86%

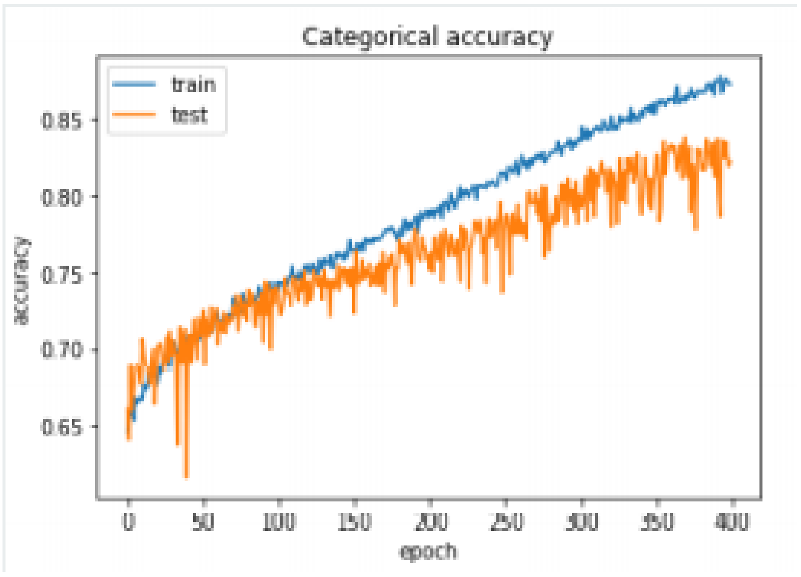


Figure 4.6. Retraining Float16 model completely over 70% of data while using FLoat32 as a knoweledge base the model get 2-3% accuracy gain as compared to when retraining is done over 50% of the data. The test set accuracy reaches 82%

4.4 Conclusion

In this chapter a method to train our google speech CNN model in lower precision(FP 16) is introduced . It is being found that trained Float16 CNN speech recognition model gives the marginal accuracy drop. Further, the concept of transfer learning in model compression is in cooperated . Based on the experiment conducted it can be concluded that when retraining of the lower precision(Float 16) CNN model while using Float32 model as the knowledge base is done with Training data less than 50% there is no accuracy increase and as the training data increases the model shows accuracy gain and more stable accuracy curve. Fairly for data greater than 50%

CHAPTER 5

Post-Training Compression

One of the most intuitive way to compress any machine learning model is to reduce the precision requirements for the weights learned by the model post training. This is what post training quantization techniques does.
Quantization actually reduces the precision of the numbers used to represent the model's parameters, which by default are 32-bit floating point numbers. This results in a smaller model size and faster computation.
The main advantages of quantization is as follows[1] -

- It is broadly applicable across a range of models. There is no need for developing a new model architecture.
- Faster computation: Most processors allow for faster processing of lower bit data.
- Smaller Model footprint : With lower bit quantization (eg 8, 6,4 bits), one can reduce the model size to a great extent. This leads to faster download times for model updates.
- Having lower precision weights and activations allows for better cache reuse.
- Lower Power: Moving 8-bit data is 4 times more efficient than moving 32-bit floating point data. Therefore reduction in amount of data movement can have a significant impact on the power consumption.

Smaller models are more adaptive towards post training compression as these model has lesser layers which means lesser number of parameters to train hence lesser probability to diverge. In our case we have inhand two trained

models out of which LSTM model is a model which have lesser number of trainable parameters.
This chapter works around some of the popular post training Compression techniques. The aim is to now compress the model weight even more say 8bits or less while preserving its accuracy. At first the weights of our model are directly casted to the lower precision to judge the accuracy drop. For obvious reasons we can say that by straightforwardly casting these weights to lower precision will result in a significant accuracy drop. This accuracy drop is directly proportional to the complexity of the model ie. greater the complexity of the model greater is the accuracy drop. Later in here, other weight compression techniques are applied. By which the model accuracy can be preserved to a great extent. Finally, a detailed analysis of the performance of our speech recognition model on applying these techniques is done. Also we understand how some of these techniques can result in inference time speedup.
Moreover, an insightful analysis regarding the performance, advantages, and drawbacks when these methods are applied on our LSTM speech recognition model built over google speech dataset is provided.

5.1 Clipping

In this section the weights of our LSTM model are directly casted to the lower precision to judge the accuracy drop. For obvious reasons we can say that by straightforwardly casting these weights to lower precision will result in a significant accuracy drop. This accuracy drop is directly proportional to the complexity of the model ie. greater the complexity of the model greater is the accuracy drop.

5.1.1 Illustration of procedure

The LSTM model is quantized by casting its weights to lower precision say int8. This casting is simply done by -

(1) Extracting the model weights post training. This is done using keras library function available.
`model.layers[i].get_weights()`
This line gives the weight list of layer i of the model.
(2) After extracting weights, these weights are casted to int 8 by rounding it to nearest integer value. eg - 3.43 :-> 3
(3) Now these new casted weights are then set back to the model layer i.
`model.layers[i].set_weights(modified_weights_int8)`

5.1.2 Results

when the LSTM model is quantized by casting its weights to lower precision say int8, the results are as follows -
Below table illustrates the behaviour of our LSTM model when its weights are casted to int 8 quantization.

TABLE 5.1. Behaviour of LSTM speech recognition model when its weights are casted to int 8 quantization.

	Float32 Model	**Int8 weight Quantized Model**	**Observation**
Model Size	10.12%	2.53%	4X reduction
Model Accuracy	91.6%	14.29%	Drastic Accuracy Drop

As we can see that the model completely diverges, there is a need of retraining this model from here to restore its accuracy. This actually add to more work. In general, post training quantization methods apply following techniques to reduce the accuracy penalty:

(1) Retraining/recalibration
(2) Architectural changes

But the problem with these methods are[10]:

- Retraining a neural network for a target bitwidth is computationally expensive, often taking much longer to train than its full precision counterpart, requiring a separate hyperparameter tuning process for convergence and best results
- Retraining/recalibrating a neural network requires access to data, which may not be available.
- Becomes hardware dependent to get inference time speed up.

So there is a need to find the way to quantize the model achieving >=4X reduction without in-cooperating retraining and achieving the similar accuracy as that of full precision inference on the traditional hardware platforms. The comming section will revolve around this.

5.2 Quantizing the weights : Floating to fixed Point conversion

5.2.1 Introduction

The main principle behind compressing the model weights value is that the weight value is quantized (i.e. discretized) to some specific values, which then can be represented using integers. precisely, this is actually fixed-point representation. This method work due to two reasons -

(1) First, DNNs are known to be quite robust to noise and other small perturbations once trained. This means even if we subtly round off numbers, we can still expect a reasonably accurate answer.
(2) The weights of the model often tend to lie in a small range, allowing us to concentrate our precious fewer bits within a smaller range.

5.2.1.1 Floating-point representation & Fixed-point representation

In floating point representation mantissa and an exponent are used to represent a real number and both can vary. Also the decimal is not fixed and can appear anywhere relative to the digits.
In fixed point the exponent is replaced by fixed scaling factor so that integer could be use to represent the number. The decimal point's position is now "fixed" by the scaling factor. The value of the scaling factor is actually the smallest distance between 2 ticks on the line and the number of such ticks are the number of bits used to represent the integer . Any value that is not an exact multiple of the constant will get rounded to the nearest point.

This section starts with defining the algorithm used for quantizing weight and then we experiment with the tradeoff between range and precision to get the best accuracy.

5.2.2 Algorithm

Weight matrix values is effectively converted into fixed point format[10]. The algorithm used to convert full precision weights to desired reduced precision is listed as algorithm 1 whereas the algorithm used to convert back these reduced precision value to original full precision is listed as algorithm 2

Algorithm 1: Quantization Algorithm

Input :

a	Full precision weight value
W	Word Length of output ie. number of bits to quantize
F	hyper parameter representing fractional part

Output :

int_part	List representing integer part of quantized output
Frac_part	Fractional part of quantized output
sign_bit	Storing sign of the output
S	scaling factor

- -

```
1.    $a_q \leftarrow$ a X $2^F$
2.    $a_q \leftarrow$ Int( a X $2^F$)
3.    max_bit $\leftarrow max(\log_2 |a_q|)$
4.    int_part $\leftarrow$ if W < max_bit then $[w_q \wedge (1 << i)$ for i in max_bit-W+F .. .. max_bit ] else $[w_q \wedge (1 << i)$ for i in max_bit-F ... max_bit ]
5.    frac_part $\leftarrow [w_q \wedge (1 << i)$ for i in 0 ... max_bit-F ]
6.    sign_bit = $a \geq 0$
7.    S $\leftarrow [w_q \wedge (1 << i)$ for i in max_bit-F ... max_bit ]
8.    return int_part, frac_part, sign_bit, scaling factor
```

Algorithm 2: De-Quantization Algorithm

Input :

int_part	List representing integer part of quantized output
Frac_part	Fractional part of quantized output

sign_bit	Storing sign of the output
F	hyper parameter representing fractional part

Output :

num_	De-quantized output

- -

```
1.    num_ ← concat(int_part, frac_part)
2.    num_ ← Int(num_)
3.    num_ ← num_ « F
4.    num_ ← if (! sign_bit) then (-1) X num_
5.    return num_
```

5.2.3 Implementation

The proposed algorithm for floating to fixed point conversion is applied to each weight value of our LSTM model to reduce its precision.These modified reduced precision weights are then set back to the model layers. This conversion technique is worked out for different values of W F. W is the reduced precision bits(16,8,6..) and F is the fractional part associated with that W.

```
For W in [16,8,6,4,3,2] and F ranges from W-2 to 1(dropping trivial cases)
$w_{old}$ ← get_weights()
$w_{new}$ ← map(Quantization($w_{old}$))
model.setweights($w_{new}$)
```

Procedure

(1) Consider the full precision float32 model

(2) Extract the weights of the model and quantize them using the stated algo.
(3) Set these modified low precision weights to the model
(4) Evaluate the model in full precision to obtain the behaviour of the speech recognition model.

5.2.4 Results

After quantizing our LSTM model weights by using the stated methodology the results obtained are provided in table 5.2.

Table 5.2. Behaviour of LSTM speech recognition model when its weights are quantized to lower precision using the stated algo

x bit quantization	**F value which gives best Accuracy**	**Accuracy obtained**
16	8	92.6%
8	6	92.3%
6	3	88.91%
4	2	64.51%
3	1	36.4%
2	1	14.33%

Graph visualization of the results obtained -

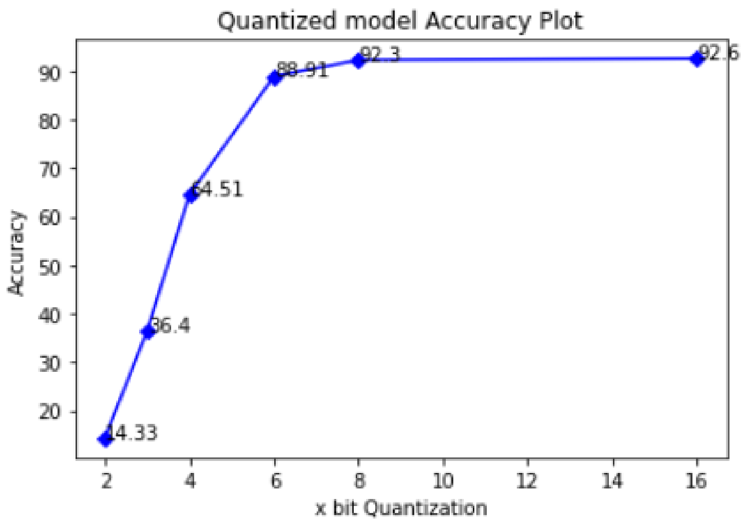


FIGURE 5.1. Accuracy of the Quantized LSTM model when its weight are quantized to 16,8,6,4,3,2 bits

5.2.5 Conclusion

(1) A post quantization method used is able to preserve the model accuracy upto 6bit compression which previously seems impossible.
(2) 16 bit quantization preserves the model accuracy completely.
(3) 8 bit quantization has marginal accuracy drop of 1%.
(4) 6 bit quantization incurs accuracy loss of 3 to 4%.

5.3 Inference time speedup

5.3.1 Introduction

Model compression has major two benefits first is smaller model size which reduces the model storage size and makes model easier and faster to download and the second is speeding up the inference which results in lesser RAM requirement to run the model.
Uptil now, using weight compression techniques like quantization and float to

fixed point conversion we are able to handle the first case. In this section we understand the handling of the latter using tfLITE toolkit which is based on the concept of quantized inference. The main advantage of quantized inference is that integer arithmetic is faster than floating point arithmetic. Thus making the deployment of the model possible on ultra low-power embedded devices, such as watches, drones or other IoT devices.
In here, LSTM speech recognition model is compressed using tfLite to judge the inference speedup.

5.3.2 procedure

(1) Convert our full precision trained LSTM speech recognition model into a float16 quantized TensorFlow Lite model using TFLiteConverter

   (a) Load the LSTM model using the TFLiteConverter.

```
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()
```

   (b) Quantize the model to float16 by first setting the optimizations flag as default then specify float16 as the supported type.

```
converter.optimizations = [tf.lite.Optimize.DEFAULT]
converter.target_spec.supported_types = [tf.float16]
tflite_fp16_model = converter.convert()
```

(2) Evaluate the Quantized float16 TensorFlow Lite LSTM model using the Python TensorFlow Lite Interpreter.

   (a) Load the Quantized model into the interpreters.

```
interpreter = tf.lite.Interpreter(model_content =
tflite_model)
interpreter.allocate_tensors()
```

   (b) For evaluating the model over the test set the one-hot encoded test set vector is converted to categorical vector.

```
testy = np.argmax(test_y,axis = 1)
```

(c) Run inference and calculate the inference time using time library in python.

(d) Find the accuracy percentage by comparing results with ground truth labels

5.3.3 Result

(1) No significant accuracy drop

(2) Inference time of the full precision LSTM model for predicting one test data label is 0.054 sec

(3) Inference time of the float16 Quantized TensorFlow Lite LSTM model for predicting one test data label is 0.0026, which is nearly a 20X speedup.

Chapter 6

Conclusion

This work covers the weight compression techniques of model Compression and how the speech recognition model respond to those techniques. It started by training two different speech recognition model on google Speech dataset - CNN model and LSTM model and then aim to compress these models starting from 2X compression to finding a way to do upto >8X compression.
At first, Quantization aware training technique is used and using this the model get reduced to half the original size. This includes a methodology for training deep neural networks using half-precision floating point numbers, while preserving model accuracy. Training the CNN speech recognition model in half precision results in 86.12 % accuracy which is $\sim$ 2% accuracy drop which is a marginal drop. In here the concept of transfer learning is introduced and how to use this concept in our way towards model compression is discussed. Based on the experiment conducted it is concluded that when retraining of the lower precision CNN model while using Float32 model as the knowledge base is done with training data less than 50% there is no accuracy increase and as the training data increases the model shows accuracy gain and more stable accuracy curve. Fairly for data greater than 50%.
Then we move forward towards the way to further compress model upto 8X reduction while preserving the model accuracy. For this, post training Compression techniques was used. It includes clipping, clustering, algorithms to convert float point to fixed point which can preserve the model accuracy to a great extent uptil 8X model compression without in cooperating any retraining. Atlast we briefly talked about how these techniques can result in

inference time speedup using tensorflow tfLITE toolkit. Moreover, an insightful analysis regarding the performance, advantages, and drawbacks when these methods are applied on our LSTM speech recognition model built over google speech dataset is being provided.

6.1 Future outlook

In future this work can be extended in multiple ways. These techniques of model compression can be applied to larger and more complex models to find out how the complex model respond to these techniques. It will be exciting to know more about these compression techniques. Also there is wide scope in further compressing the model till binary weights ie less than 4bit quantization.

References

[1] Raghuraman Krishnamoorthi. Quantizing deep convolutional networks for efficient inference: A whitepaper. *arXiv preprint arXiv:1806.08342, 2018..*

[2] Stanford University Song Han CVA group. "Deep Neural Network Model-Compression and Efficient Inference Engine". *In : arXiv, 2015.*

[3] Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. 2017. A survey of model compression and acceleration for deep neural networks. *arXiv preprint arXiv:1710.09282 (2017).*

[4] Jeffrey L McKinstry, Steven K Esser, Rathinakumar Appuswamy, Deepika Bablani, John V Arthur, Izzet B Yildiz, and Dharmendra S Modha. Discovering low-precision networks close to full-precision networks for efficient embedded inference. *arXiv preprint arXiv:1809.04191, 2018..*

[5] Choukroun, Y., Kravchik, E., Yang, F. Kisilev, P. Low-bit Quantization of Neural Networks for Efficient Inference. *arXiv:1902.06822 [cs, stat] (2019).*

[6] Krizhevsky, Alex, Ilya Sutskever and Geoffrey E. Hinton (May 2017). 'ImageNetClassification with Deep Convolutional Neural Networks'. In: *Commun.ACM* 60.6, pp. 84–90.issn: 0001-0782.doi:10.1145/3065386. url : https://doi.org/10.1145/3065386.36

[7] url: *"https://docs.nvidia.com/deeplearning/performance/mixed-precision training/index.html"*

[8] url :*"https://developer.nvidia.com/blog/mixed-precision-training-deepneural-networks/"*

[9] url : *" https://www.analyticsvidhya.com/blog/2019/07/learn-build-first-speech-to-text-model-python/"* .

[10] Lam, Maximilian et al. (Feb. 2020b). 'Quantized Neural Network Inference with Precision Batching'. In:arXiv e-prints, arXiv:2003.00822 [cs.LG].

[11] "Tensorflow post training float16 quantization" url: *https://www.tensorflow.org/lite/performance/post_training_float16_quant.*

[12] Choi, Yoojin, Mostafa El-Khamy and Jungwon Lee (Dec. 2016). 'Towards the Limit of Network Quantization'. In: *arXiv e-prints*, arXiv:1612.01543.

[13] Han, Song, Huizi Mao and William J. Dally (Oct. 2015). 'Deep Compression:Compressing Deep Neural Networks with Pruning, Trained Quantizationand Huffman Coding'. In: *arXiv e-prints*, arXiv:1510.00149

[14] Choi, Jungwook et al. (2018).PACT: Parameterized Clipping Activation forQuantized Neural Networks. arXiv : 1805.06085 [cs.CV].

[15] Zhang, Dongqing et al. (2018).LQ-Nets: Learned Quantization for Highly-Accurate and Compact Deep Neural Networks. arXiv:1807.10029 [cs.CV].

[16] Zhou, Shuchang et al. (2018).DoReFa-Net: Training Low Bitwidth Convolu-tional Neural Networks with Low Bitwidth Gradients. arXiv:1606.06160[cs.NE].

[17] Diederik P. Kingma and Jimmy Lei Ba. Adam : A method for stochastic optimization. 2014. arXiv:1412.6980v9