



# Model Compression

*(A thesis submitted in fulfilment of the requirement  
for the degree of Master of Technology)*

***Presented by : Ankita Gupta***  
***Supervisor : Prof. Kolin Paul***

*(18 April 2021)*

# Introduction

- In recent years, deep neural networks have been applied to different applications and achieved drastic accuracy improvements in many tasks. 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.
- **Goal** : better performance, less memory and storage, deployment on devices with limited resources
- Model compression is a technique that shrinks trained neural networks while preserving its accuracy
- It is broadly divided into two categories - Architectural Compression (pruning) , Weight Compression (Quantization , clustering) .  
In this project we primarily focus on weight compression techniques.

# Overview

**Problem Statement : Design and build speech recognition model on Google speech dataset and then aim to compress it using weight compression techniques.**

- 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.
- The methodology for training deep neural networks using half-precision floating point numbers is covered. The aim is to do it without losing model accuracy. The concept of transfer learning is also introduced and how to use this concept in our way towards model compression is also discussed
- Model compression using post training Compression techniques which include clipping, clustering, fixed point Quantizers . The aim is to compress the model weights even more say 8 bits or less while preserving its accuracy.
- Judge the inference time speedup. This is done using tensorflow LITE toolkit.

# Models

Two models are built over the google speech dataset.

## CNN Model

- It consists of 4 convolution layers and 3 dense layers layered sequentially. Conv layers has dropout and maxpools layer incorporated. All layers uses relu activation except that of the last layer which uses softmax activation to finally output the class label. Model is built using keras API
- **Training** - Optimizer used is adam, loss function is categorical\_crossentropy
- **Accuracy** : Accuracy achieved over the test set is 88.20% (full precision training)

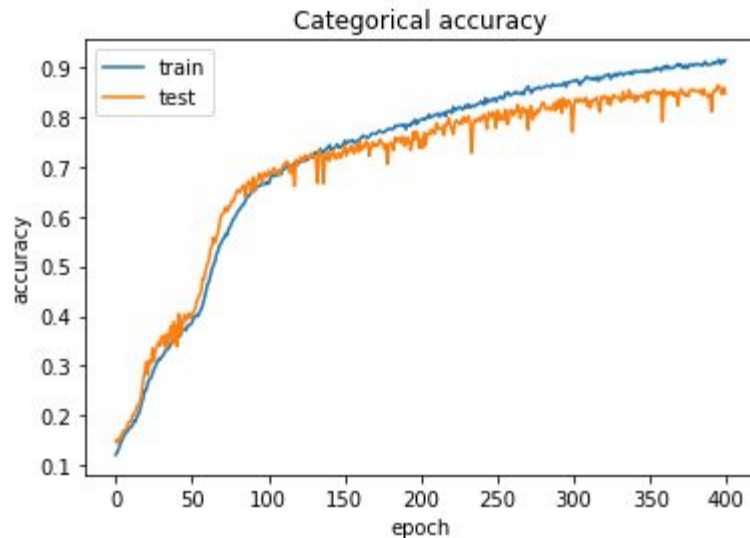
## LSTM Model

- It consists of LSTM layer and two dense layers. Last one with the softmax activation. It is smaller model as compared to the CNN model. It has much lesser number of trainable parameters.
- **Training** - Optimizer used is RMSprop, loss function is categorical\_crossentropy
- **Accuracy** : Accuracy achieved over the test set is 91.60%(full precision training)

# Lower Precision Training

The CNN model is trained on google speech dataset in half precision(FP16)

- **Advantages** : Decrease the required amount of memory, Shorten the training or inference time, Decreased memory bandwidth.
- **Methodology** :
  - Porting the model to use the FP16 data type.
  - Scaling constants so that their value does not drop below provided threshold.
  - Setting the optimizers: fixing optimizer to stochastic gradient descent and setting the lr to constant.
- **Result** : Test set accuracy of CNN model when trained in half precision(FP16) results in 86.12 % accuracy. which is ~ 2% accuracy drop as compared to full precision CNN model which is a marginal drop.



# In cooperating transfer learning

- One genuine thought arises that why to train the float16 model from scratch if we have already pre trained float32 model beforehand. So, this pushes us towards in cooperating another interesting methodology “**transfer learning**”.
- *Transfer learning is a machine learning technique where a model trained on one task is re-purposed on a second related task.*
- The general idea is to use the knowledge learned by the pretrained float32 CNN model in the process of training of float16 CNN model. so, training of the float16 is not done from scratch instead a knowledge base provided beforehand is used. It can be claimed that less amount of training 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.
- Based on the experiment conducted it is concluded that when retraining data less than 50% there is no accuracy increase and as the retraining data increases the model shows accuracy gain and more stable accuracy curve. Fairly for data greater than 50%.

# 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 compression 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
- The main advantages of quantization is as follows :
  - It is broadly applicable across a range of models
  - Smaller Model footprint
  - Faster computation
  - Lower Power

# Post Training Compression

- So the aim is to reduce the precision of the weights thereby reducing the model size post training while preserving the model accuracy.
- The most trivial way to compress the weights of the model is to directly cast the Float32 weights to lower precision say int8 or even less this quantization technique is called **“clipping”**. This undoubtedly reduces the model size by the factor 4X or more but is the accuracy of the model preserved?
- When the full precision(FP32) weights of the LSTM model built over google speech dataset are clipped to int8 values the model completely diverges. The accuracy dropped from 91.6% to 14.29% .
- 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.
  - Retraining a neural network for a target bitwidth is computationally expensive,
  - Retraining a neural network requires access to data, which may not be available.



# Fixed Point Quantizer

- 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 -
  - DNNs are known to be quite robust to noise and other perturbations once trained.
  - 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.

## Floating Point & Fixed Point

- 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 Whereas 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. Any value that is not an exact multiple of the constant will get rounded to the nearest point.

# Quantization Algorithm

1.  $a_q \leftarrow a \times 2^F$
2.  $a_q \leftarrow \text{Int}(a_q)$
3.  $\text{max\_bit} \leftarrow \max(\log_2 |a_q|)$
4.  $\text{int\_part} \leftarrow$  if  $W < \text{max\_bit}$  then  $[w_q \wedge (1 \ll i)$   
for  $i$  in  $\text{max\_bit}-1 \dots \text{max\_bit}-W+F$ ] else  $[w_q \wedge$   
 $(1 \ll i)$  for  $i$  in  $\text{max\_bit}-1 \dots F$ ]
5.  $\text{frac\_part} \leftarrow [w_q \wedge (1 \ll i)$  for  $i$  in  $F-1 \dots 0$ ]
6.  $\text{sign\_bit} = a \geq 0$
7. return  $\text{int\_part}, \text{frac\_part}, \text{sign\_bit}$

*Input :*

*a - Full precision weight value*

*W - Word Length of output ie. number of bits to quantize*

*F - hyper parameter representing fractional part*

*Output :*

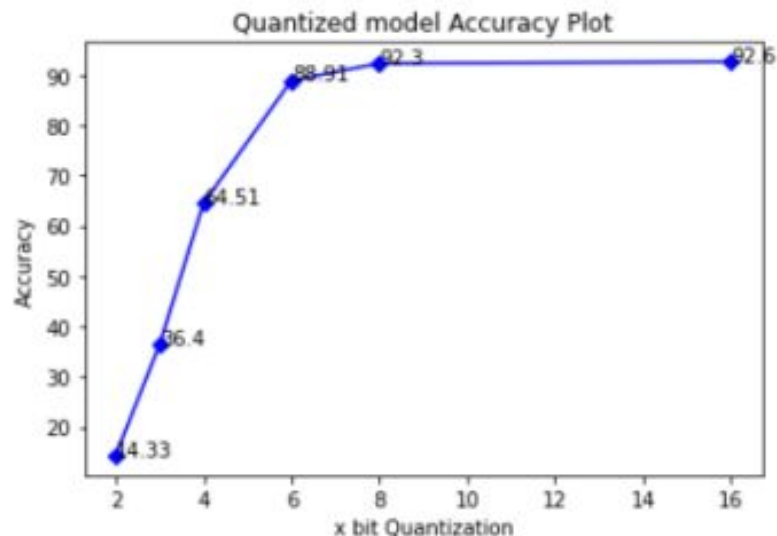
*Int\_part : List representing integer part of quantized output*

*Frac\_part : Fractional part of quantized output*

*Sign\_bit : Storing sign of the output*

# Result and Conclusion

- 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. The results obtained are as follows -
- A post quantization method used is able to preserve the model accuracy upto 6 bit compression which previously seems impossible.
- 16 bit quantization preserves the model accuracy completely.
- 8 bit quantization has marginal accuracy drop of 1%.
- 6 bit quantization incurs accuracy loss of 3 to 4%.



# Inference time speedup

- 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 we are able to handle the first case. Now its time to see the handling of the latter. This is done using tfLITE toolkit which is based on the concept of quantized inference.
- The main advantage of quantized inference is that the 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.
- The LSTM speech recognition model is compressed using tfLite to judge the inference speedup.
- Results :
  - No significant accuracy drop
  - 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.

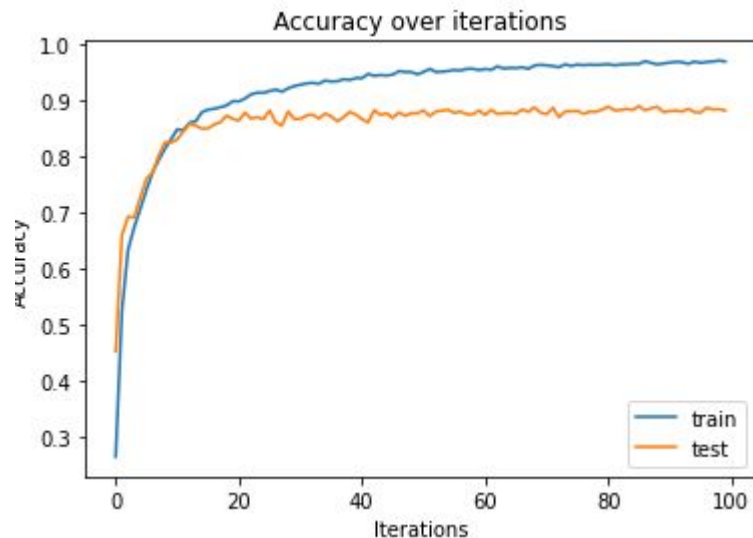
# References

- [1] Raghuraman Krishnamoorthi. Quantizing deep convolutional networks for efficient inference: A whitepaper. arXiv preprint arXiv:1806.08342, 2018..
- [2] 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).
- [3] 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..
- [4] Choukroun, Y., Kravchik, E., Yang, F. Kisilev, P. Low-bit Quantization of Neural Networks for Efficient Inference. arXiv:1902.06822 [cs, stat] (2019).
- [5] Krizhevsky, Alex, I.Sutskever and G. 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>
- [6]url: <https://docs.nvidia.com/deeplearning/performance/mixed-precision-training/index.html>
- [7] url : "<https://www.analyticsvidhya.com/blog/2019/07/learn-build-firstspeech-to-text-model-python/>" .
- [8] Lam, Maximilian et al. (Feb. 2020b). 'Quantized Neural Network Inference with Precision Batching'. In:arXiv e-prints, arXiv:2003.00822 [cs.LG].
- [9] "Tensorflow post training float16 quantization"  
url: " [https://www.tensorflow.org/lite/performance/post\\_training\\_float16\\_quant](https://www.tensorflow.org/lite/performance/post_training_float16_quant)"

**Thank you !**

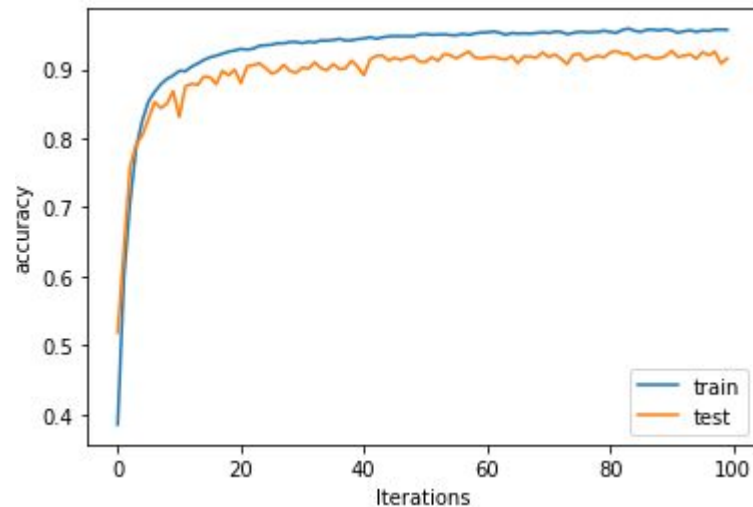
# Accuracy Plots

## CNN Model



Train Accuracy: 0.9295  
Test Accuracy: 0.8820

## LSTM Model

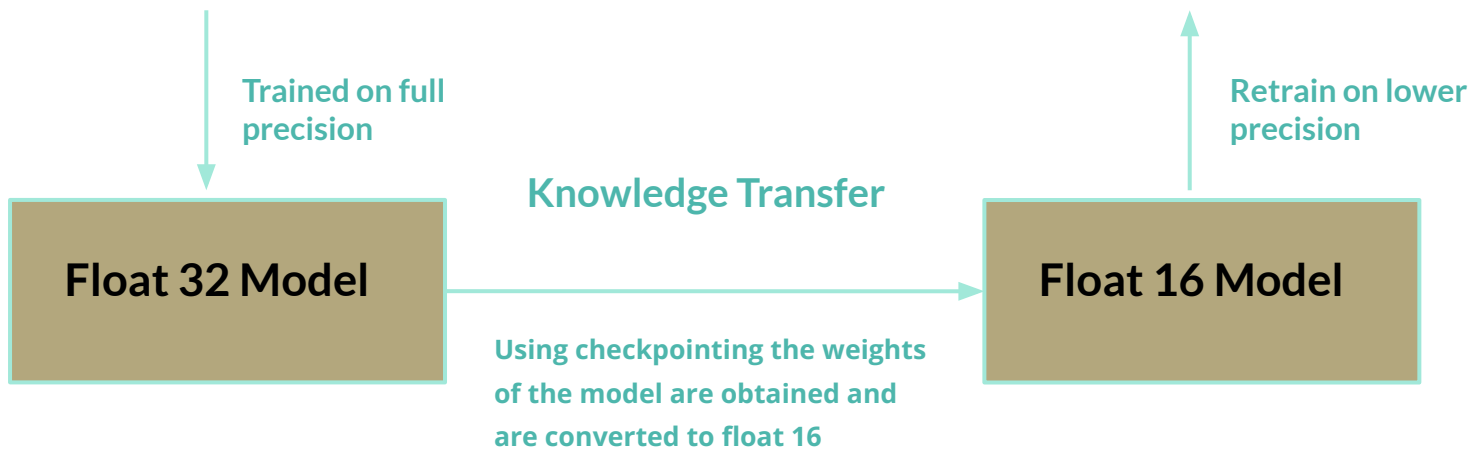


Train Accuracy: 0.9574  
Test Accuracy: 0.9160

# In cooperating transfer learning

- One genuine thought arises that why to train the float16 model from scratch if we have already pre trained float32 model beforehand. So, this pushes us towards in cooperating another interesting methodology “**transfer learning**”.
- *Transfer learning is a machine learning technique where a model trained on one task is re-purposed on a second related task.*
- 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. So, with transfer learning a solid machine learning model can be built with comparatively little training data .
- The general idea is to use the knowledge learned by the pretrained float32 CNN model in the process of training of float16 CNN model. so, training of the float16 is not done from scratch instead a knowledge base provided beforehand is used. It can be claimed that less amount of training 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.





# Results

- 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%.

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%

# 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”.
- 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 needed . This is going to lower the memory requirements. It's also going to lower the time needed for model's parameters to converge to ones that work well.

# Data Preprocessing



The Preprocessing applied is :

- Audio file is resampled to 8k hz. This is done using :
- 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.
- Handling shorter commands of less than 1 second - This is done by padding zeros.
- Output labels are converted to encoded integer and then further these integer encoded labels are converted to a one-hot vector as it is a multi-class classification problem.
- 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



# Models

In this section two different models on google speech dataset are built.

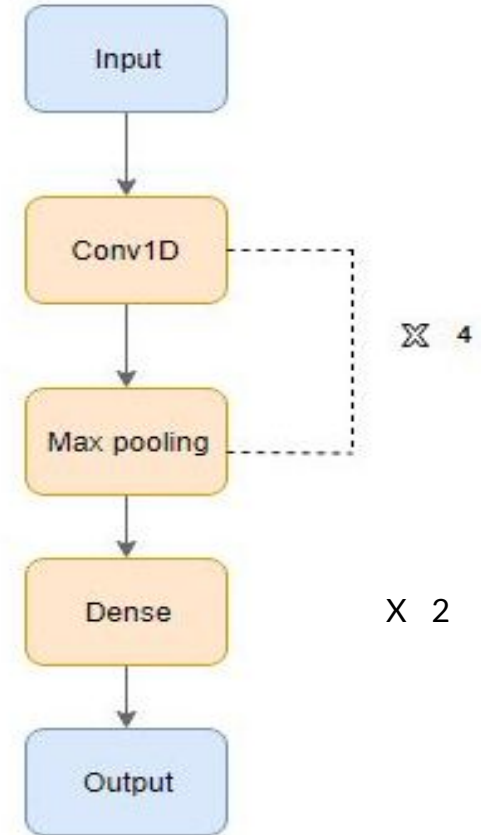
# CNN Model



- 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
- Keras API is being used to build the model. Model Used is the common CNN model .

# CNN Model Architecture

- The overview of architecture of the model used -
  - It consists of 4 convolution layers with maxpool and dropout layer with dropout 0.3 is incorporated. 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



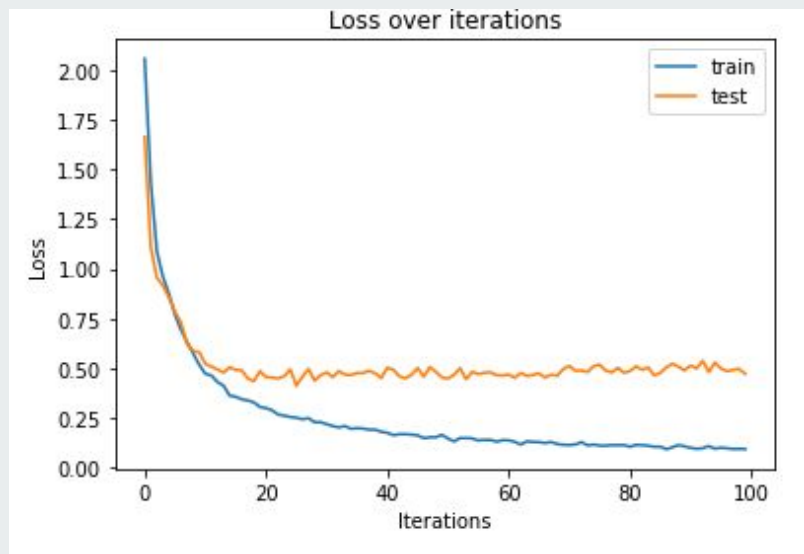
# CNN Model Training



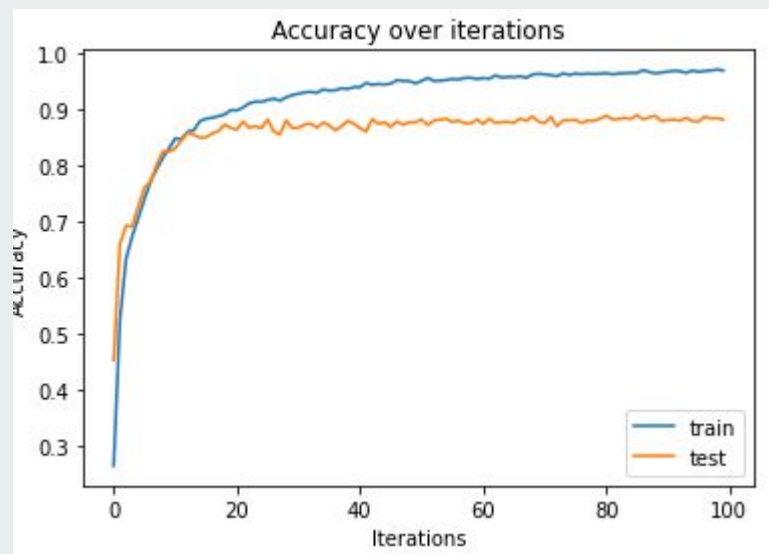
- The input feature vector is of dimension (8000,1)
- Total Trainable Parameters - 1,611,498
- Optimizer used - adam
  - Adam is an adaptive learning rate optimization algorithm that's been designed specifically for training deep neural networks. The algorithm leverages the power of adaptive learning 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



# Accuracy & Loss Plot - CNN Model



Train Loss: 0.093  
Test Loss: 0.4711



Train Accuracy: 0.9295  
Test Accuracy: 0.8820

# 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.

# LSTM Model Architecture



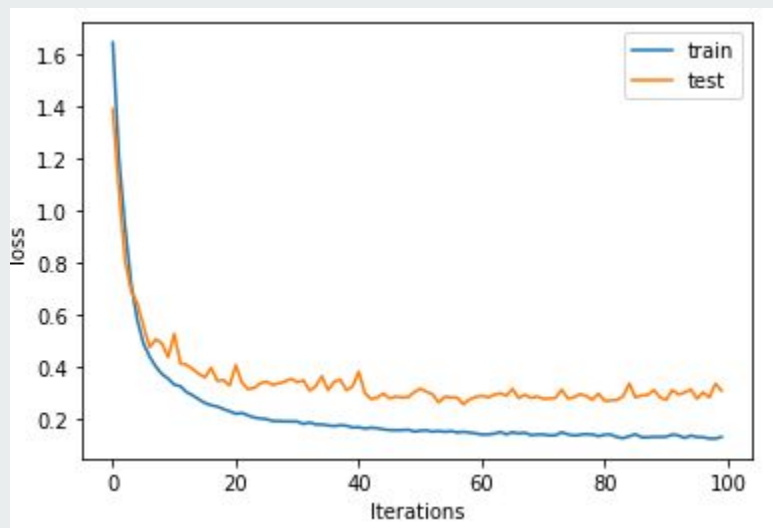
- It consists of a LSTM layer, hyper parameters of which are as follows-
  - hidden units = 12
  - Inner activation = 'sigmoid'
  - activation = 'tanh'.
- A Dense layer with relu as an activation function.
- And lastly a softmax layer to finally output the class label.

# LSTM Model Training

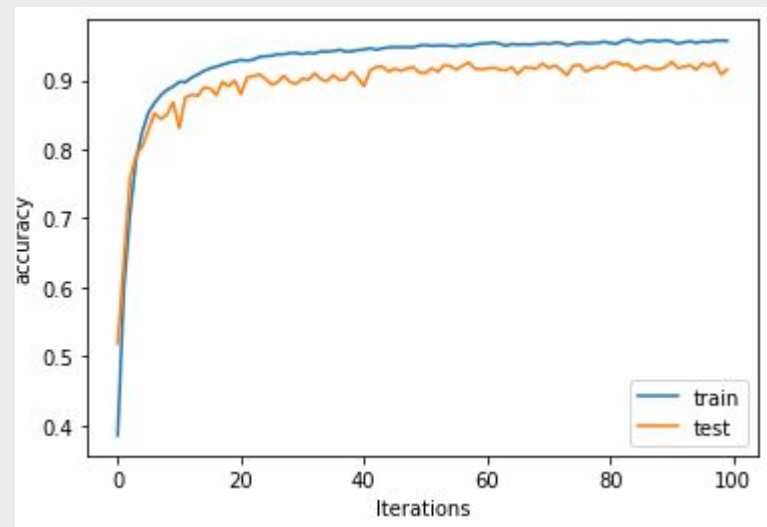


- Input feature vector is of dim (81, 12)
- Optimizer used - RMSprop with learning rate=0.01
- Loss function - categorical\_crossentropy
- Model is trained over the training set for 100 epochs with 100 batch size

# Accuracy & Loss Plot - LSTM Model



Train Loss: 0.1331  
Test Loss: 0.3103



Train Accuracy: 0.9574  
Test Accuracy: 0.9160

# Comparing the proposed model



<b>Models</b>	<b>Trainset Accuracy</b>	<b>Testset Accuracy</b>
<b>CNN Model</b>	92.95%	88.20%
<b>LSTM Model</b>	95.74%	91.60%

# Quantization Aware Training

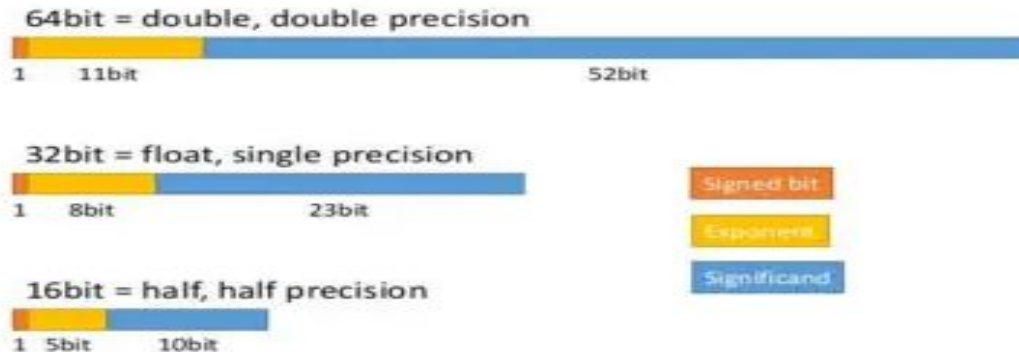


This section introduces the methodology for training our Convolution neural network model using half-precision floating point numbers and aim to do it without losing model accuracy. After this it introduces the concept of transfer learning and how to use this concept in our way towards model compression. 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

# Terminologies

- **Half precision** (also known as FP16)
- **Single precision** (also known as 32-bit) is a common/default floating point format (float in C-derived programming languages),
- **Double precision** also 64-bit, known as double precision (double).
- **Mixed precision** is as the name indicates is the combined use of different numerical precisions in a computational method.





# Lower precision training



In this section, we train our deep learning CNN model built over google speech dataset in half precision and will introduce some techniques on achieving good performance and accuracy and then finally compare the accuracies of both the models(ie. Float 32 and float16 models)

The main advantages of lower precision training are -

- Decrease the required amount of memory
  - Half-precision floating point format (FP16) uses 16 bits, compared to 32 bits for single precision (FP32). Lowering the required memory enables training of larger models or training with larger mini-batches.

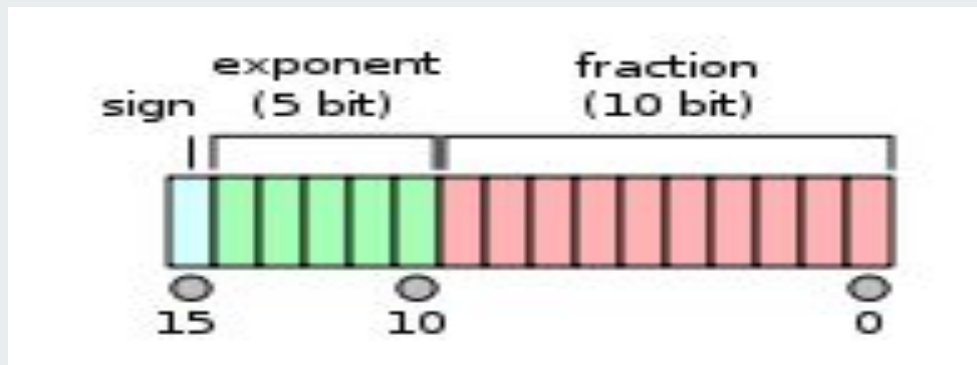
# Advantages of lower precision training



- 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.
- Decreased memory bandwidth
  - Requires less memory bandwidth thereby speeding up data transfer operations and lowering communication costs

# Challenges in half precision training

IEEE 754 standard defines the following 16-bit half-precision floating point format: 1 sign bit, 5 exponent bits and 10 fractional bits.



# Challenges in half precision training



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 your training.

- Your gradients can underflow
- Your activations or loss can overflow
- The weight update is imprecise.

# Solution



To address Above problems-

➤ **Porting the model to use the FP16 data type.**

This can be achieved in tensorflow using-  
`Tensorflow.keras.backend.set_floatx`  
("float16").


This enable all the backend calculations to be  
executed in float16

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

Converted to float 16

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

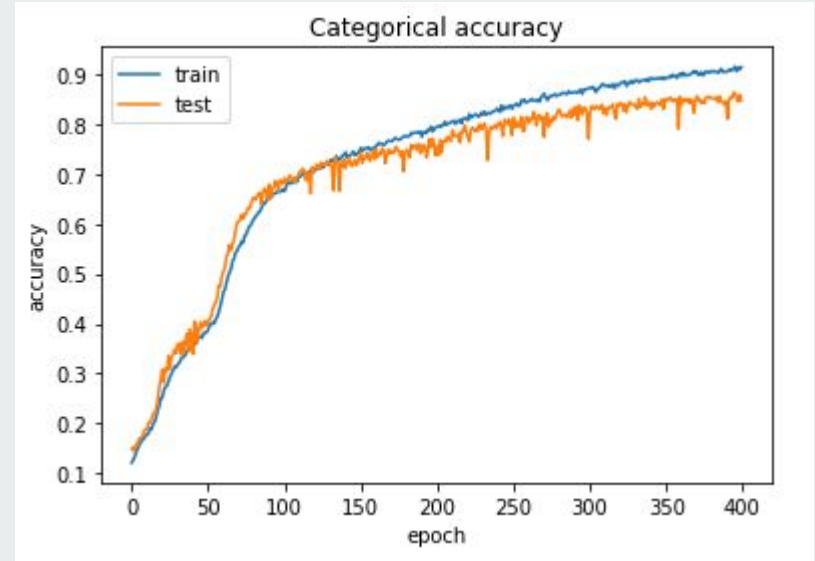
*To Be Continued...*

- 
- **Scaling constants** so that their value does not drop below provided threshold `K.set_epsilon(1e-4)`
  - **Setting the optimizers:** fixing optimizer to stochastic gradient descent and setting the learning rate to constant
  - Input and output shape for each conv layer is multiple of 8

# Results

We have carried out the above experiment on Convolution neural networks model built on google speech dataset.

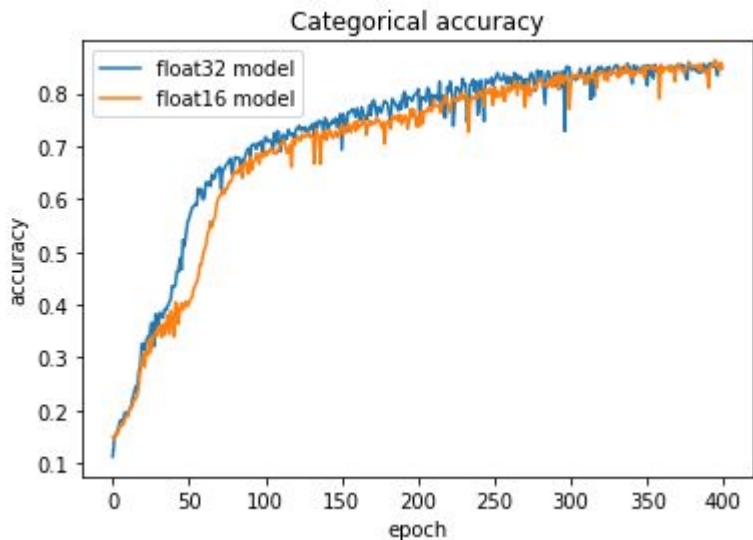
The accuracy plot for the model are as follows:-



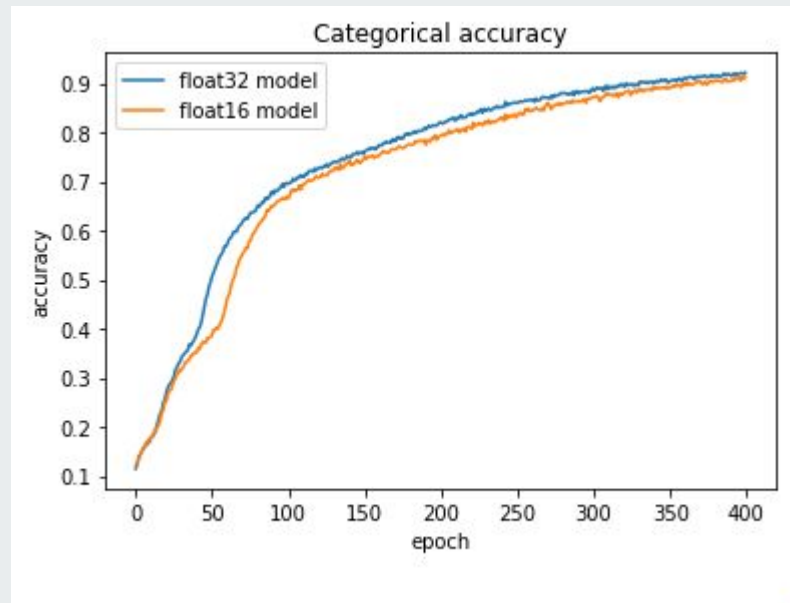
Train Accuracy: 0.9115

Test\_Accuracy: 0.8612

# Float32 model v/s float16 model



Train Accuracy of float32 model: 0.9295  
Train Accuracy of float16 model: 0.9115



Test Accuracy of float32 model: 0.8819  
Test Accuracy of float16 model: 0.8612



# Way to transfer learning



Now one genuine thought arises that why to train the float16 model from scratch if we have already pretrained float32 model beforehand. as using this will further reduce our training time.

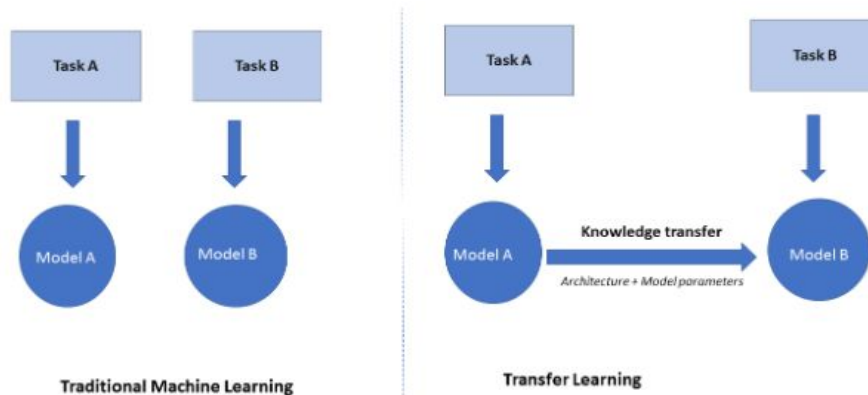
This push us towards incorporating another interesting methodology “transfer learning”. This section will find and analyse the effect of incorporating transfer learning in model compression. We will carry our analysis on Convolution neural networks(CNN) build on google speech data set.

# Transfer learning

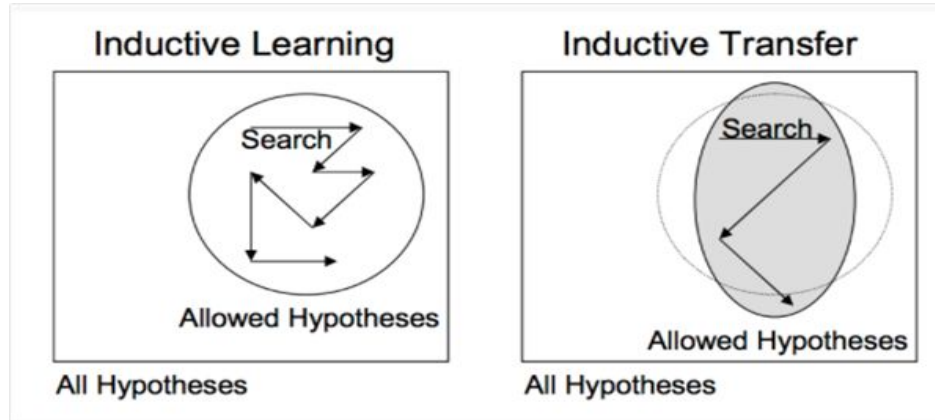
Transfer learning is a machine learning technique where a model trained on one task is re-purposed on a second related task.

Or

Transfer learning and domain adaptation refer to the situation where what has been learned in one setting ...is exploited to improve generalization in another setting



# Why transfer learning?



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

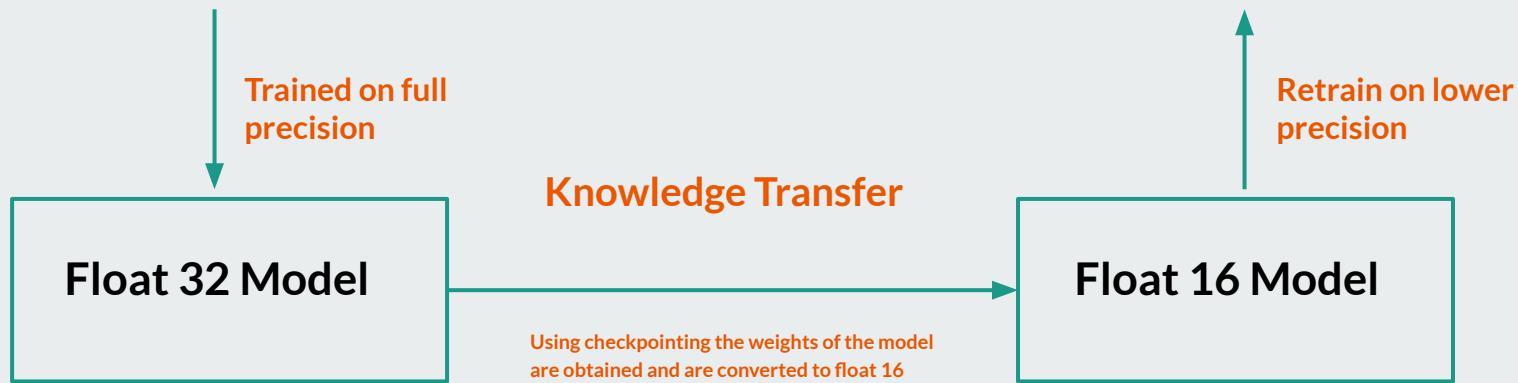
- Saving training time
- Better performance of neural networks (in most cases)
- Not needing a lot of data.

# Incorporating transfer learning



In this section we will apply transfer learning methodology in model comprehension . The general idea is to use the knowledge Learned by our pretrained float 32 model in the process of training of float16 model. As now we are not training the float16 from scratch instead we have a knowledge base provided beforehand so we can claim that the training time of the model going to get reduced and also we can need lesser data to reach the optimal accuracy. Hence we retrain our model on different portions of training data(10% 20% 70% etc) and the results are laid.

# Procedure/Implementation



# Procedure/Implementation




1) Consider pretrained float32 model

**Cnn model architecture :-**

-> Not using any custom layers because custom layers are to be treated differently while retraining in lower precision,

-> 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

```
data=models.layers[i].get_weights()    \\ weights of ith layer
data=data.astype(np.float16)           \\converting data to float16
data.save("data.npy")                  \\saving data
```

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

```
for i, layers in enumerate(modelfloat16.layers):
    if (len(layers.get_weights()))!=0:
        layers.set_weights(weightsfloat16[i])
```

4) Retrain the model over different portions of the training set to obtain the the behaviour of the model while retraining

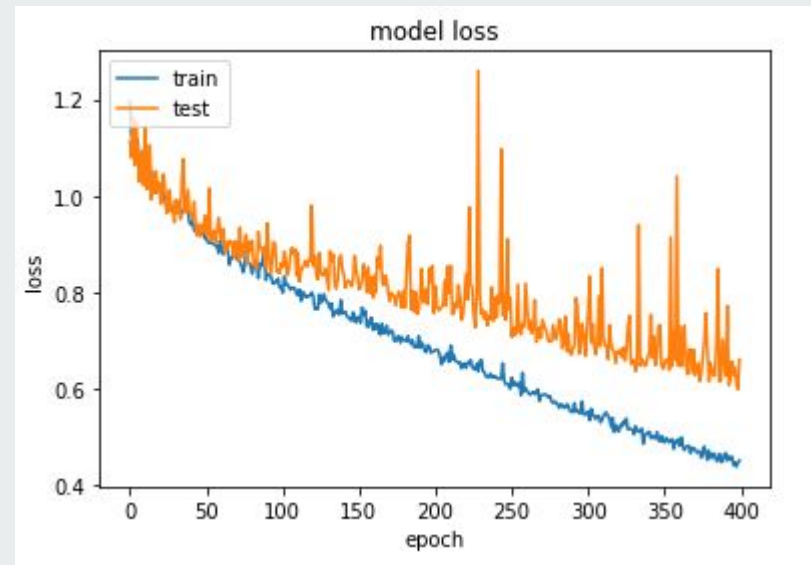
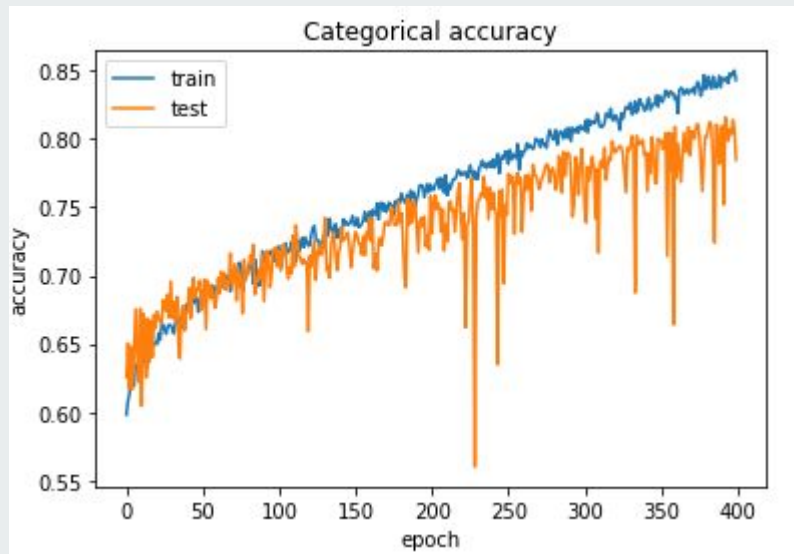
## Observation & Conclusion



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%

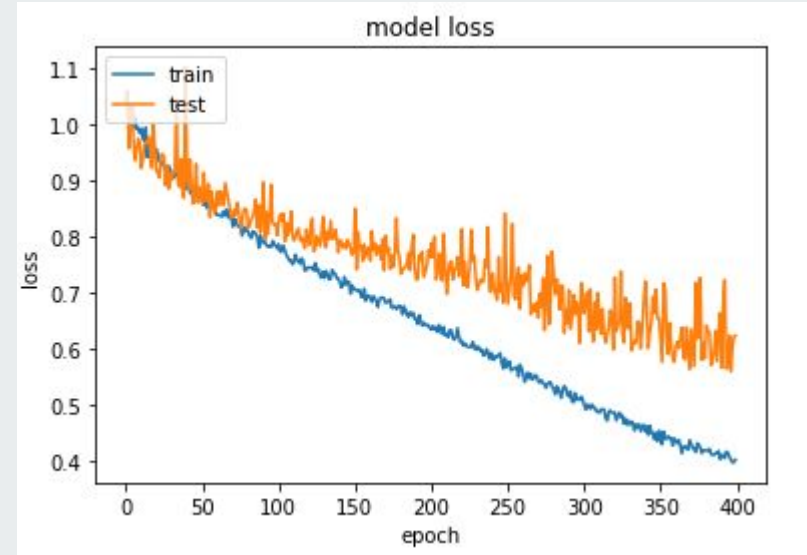
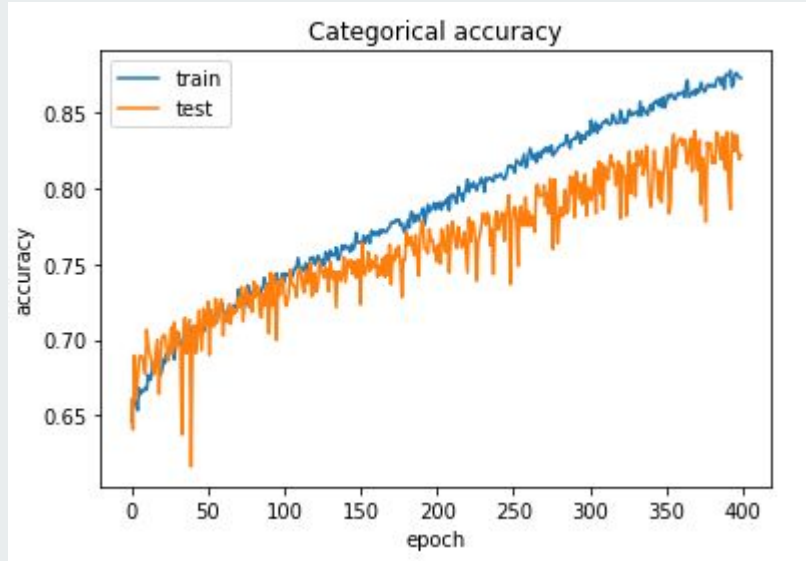


# Retraining over 50% of data



Retraining Float16 model completely over 50% of data while using FFloat32 as a knowledge base we get test set accuracy of 80% while the training accuracy reaches 86%

# Retraining over 70% of data



Retraining Float16 model completely over 70% of data while using FFloat32 as a knowledge base we get 2-3% accuracy gain as compared to when retraining is done over 50% of the data. Our test set accuracy reaches 82

# Conclusion




In this section 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 we conclude 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 our model shows accuracy gain and more stable accuracy curve. Fairly for data greater than 50%.

# Post -Training Compression



- One of the most intuitive way to compress any 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.
- This section works around some of the popular post training Compression techniques. The aim is to now compress the model weight even more say 8 bits 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 we can preserve the model accuracy 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.

# Clipping



- 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 two things -
  - The greater is the complexity of the model greater is the accuracy drop.
  - Larger is the precision drop greater will be the chances that the model will diverge.

# Procedure



1. Extracting the model weights post training. This is done using keras library function.
  - `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)
3. Now these new casted weights are then set back to the model layer i.
  - `model.layers[i].set_weights(modified_weights_int8)`

# Results



	<b>Float32 Model</b>	<b>Int8 weight Quantized Model</b>	<b>Observation</b>
<b>Model Size</b>	10.12%	2.53%	4X reduction
<b>Model Accuracy</b>	91.6%	14.29%	Drastic Accur- acy 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 change



# Quantizing weights - Float to fixed point conversion



# Algorithm



1.  $a_q \leftarrow a \times 2^F$
2.  $a_q \leftarrow \text{Int}(a \times 2^F)$
3.  $\text{max\_bit} \leftarrow \max(\log_2 |a_q|)$
4.  $\text{int\_part} \leftarrow$  if  $W < \text{max\_bit}$  then  $[w_q \wedge (1 \ll i) \text{ for } i \text{ in } \text{max\_bit}-W+F \dots \text{max\_bit}]$  else  $[w_q \wedge (1 \ll i) \text{ for } i \text{ in } \text{max\_bit}-F \dots \text{max\_bit}]$
5.  $\text{frac\_part} \leftarrow [w_q \wedge (1 \ll i) \text{ for } i \text{ in } 0 \dots \text{max\_bit}-F]$
6.  $\text{sign\_bit} = a \geq 0$
7.  $S \leftarrow [w_q \wedge (1 \ll i) \text{ for } i \text{ in } \text{max\_bit}-F \dots \text{max\_bit}]$
8. return  $\text{int\_part}, \text{frac\_part}, \text{sign\_bit}, \text{scaling factor}$

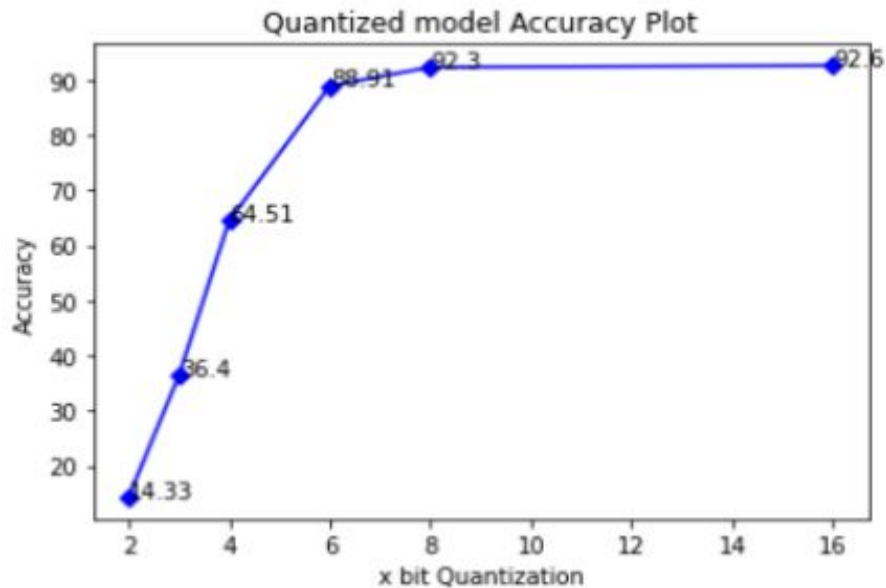
# Results



- 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

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

# Graph visualization of result





## observation

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%.



## Inference time speedup

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, clustering and float to fixed point conversion we30 5 Post-Training Compression 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.



# Understanding inference time speedup



## Results

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.





# Conclusion

This work covers the weight compression techniques of model Compression and how the speech recognitions 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 recogining model in half precision results in 86.12 % accuracy which is ~ 2% accuracy drop which is a marginal drop. In here the concept of transfer learning was also introduced and how to use this concept in our way towards model compression. 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 our 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 by which we can preserve the model accuracy to a great extent upto 8X model compression without in cooperating any retraining. Atlast we briefly talked about how these techniques can 326.1 Future outlook 33 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