# *Title:- Low bit-width quantization scheme for LSTM inference which gives predictable degradation for ASR (Automatic Speech Recognition) models.*

## Step 1: Setup (Import Libraries)

```
import os
import pathlib

import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import tensorflow as tf
print(tf.version.VERSION)
from tensorflow.keras.layers.experimental import preprocessing
from tensorflow.keras import layers
from tensorflow.keras import models
from IPython import display

# Set seed for experiment reproducibility
seed = 42
tf.random.set_seed(seed)
np.random.seed(seed)
```

```
     2.9.2
```

## Step 2:- Import the Google Speech Commands dataset

```
data_dir = pathlib.Path('data/mini_speech_commands')
if not data_dir.exists():
  tf.keras.utils.get_file(
      'mini_speech_commands.zip',
      origin="http://storage.googleapis.com/download.tensorflow.org/data/mini_speech_comma
      extract=True,
      cache_dir='.', cache_subdir='data')
```

***## Moving wav files from command directories to unknown sub-directory (Factory Reset to reset data directory)***

```
# comment out below line if "unknown" directory already exists
!mkdir /content/data/mini_speech_commands/unknown
```

```
# moves files from their specific commands directory to the "unknown" directory (replaces
!mv /content/data/mini_speech_commands/down/* /content/data/mini_speech_commands/unknown
!rm -d /content/data/mini_speech_commands/down
!mv /content/data/mini_speech_commands/go/* /content/data/mini_speech_commands/unknown
!rm -d /content/data/mini_speech_commands/go
!mv /content/data/mini_speech_commands/left/* /content/data/mini_speech_commands/unknown
!rm -d /content/data/mini_speech_commands/left
!mv /content/data/mini_speech_commands/right/* /content/data/mini_speech_commands/unknown
!rm -d /content/data/mini_speech_commands/right
!mv /content/data/mini_speech_commands/stop/* /content/data/mini_speech_commands/unknown
!rm -d /content/data/mini_speech_commands/stop
!mv /content/data/mini_speech_commands/up/* /content/data/mini_speech_commands/unknown
!rm -d /content/data/mini_speech_commands/up
!rm /content/data/mini_speech_commands/README.md
!ls /content/data/mini_speech_commands/unknown | wc -l
```

```
mkdir: cannot create directory ‘/content/data/mini_speech_commands/unknown’: File exi
mv: cannot stat '/content/data/mini_speech_commands/down/*': No such file or director
rm: cannot remove '/content/data/mini_speech_commands/down': No such file or director
mv: cannot stat '/content/data/mini_speech_commands/go/*': No such file or directory
rm: cannot remove '/content/data/mini_speech_commands/go': No such file or directory
mv: cannot stat '/content/data/mini_speech_commands/left/*': No such file or director
rm: cannot remove '/content/data/mini_speech_commands/left': No such file or director
mv: cannot stat '/content/data/mini_speech_commands/right/*': No such file or directo
rm: cannot remove '/content/data/mini_speech_commands/right': No such file or directo
mv: cannot stat '/content/data/mini_speech_commands/stop/*': No such file or director
rm: cannot remove '/content/data/mini_speech_commands/stop': No such file or director
mv: cannot stat '/content/data/mini_speech_commands/up/*': No such file or directory
rm: cannot remove '/content/data/mini_speech_commands/up': No such file or directory
rm: cannot remove '/content/data/mini_speech_commands/README.md': No such file or dir
3311
```

***## Sets wanted commands for training (Available commands: Down, Go, Left, No, Right, Stop, Up, Yes, and Unknown for commands that are not to be tested***

```
commands = np.array(tf.io.gfile.listdir(str(data_dir)))
commands = ["yes","no", "unknown"]
print('Commands:', commands)
```

```
Commands: ['yes', 'no', 'unknown']
```

***## Extract the audio files into a list and shuffle it.***

```
filenames = tf.io.gfile.glob(str(data_dir) + '/*/*')
filenames = tf.random.shuffle(filenames)
num_samples = len(filenames)
print('Number of total examples:', num_samples)
print('Number of examples per label:',
      len(tf.io.gfile.listdir(str(data_dir/commands[0]))))
print('Example file tensor:', filenames[0])
```

```
Number of total examples: 5311
Number of examples per label: 1000
Example file tensor: tf.Tensor(b'data/mini_speech_commands/unknown/5c8af87a_nohash_2
```

## Step 3:- Split the files into training, validation and test sets using a 80:10:10 ratio, respectively.

```
# Take 80% of total number examples for training set files
train_files = filenames[:4249]
# Take 10% of total number examples adding to 80% of total examples for validation set fil
val_files = filenames[4249: 4249 + 531]
# Take -10% of total number examples for test set files
test_files = filenames[-531:]

print('Training set size', len(train_files))
print('Validation set size', len(val_files))
print('Test set size', len(test_files))
```

```
Training set size 4249
Validation set size 531
Test set size 531
```

## Step 4:- Reading audio files and their labels

***## The audio file will initially be read as a binary file, which you'll want to convert into a numerical tensor.***

***## To load an audio file, you will use `tf.audio.decode_wav`, which returns the WAV-encoded audio as a Tensor and the sample rate.***

***## A WAV file contains time series data with a set number of samples per second.Each sample represents the amplitude of the audio signal at that specific time. In a 16-bit system, like the files in `mini_speech_commands`, the values range from -32768 to 32767.***

***## The sample rate for this dataset is 16kHz. Note that `tf.audio.decode_wav` will normalize the values to the range [-1.0, 1.0]. ***

```
def decode_audio(audio_binary):
  audio, _ = tf.audio.decode_wav(audio_binary)
  return tf.squeeze(audio, axis=-1)
```

**## The label for each WAV file is its parent directory.**

```
def get_label(file_path):
  parts = tf.strings.split(file_path, os.path.sep)

  # Note: You'll use indexing here instead of tuple unpacking to enable this
  # to work in a TensorFlow graph.
  return parts[-2]
```

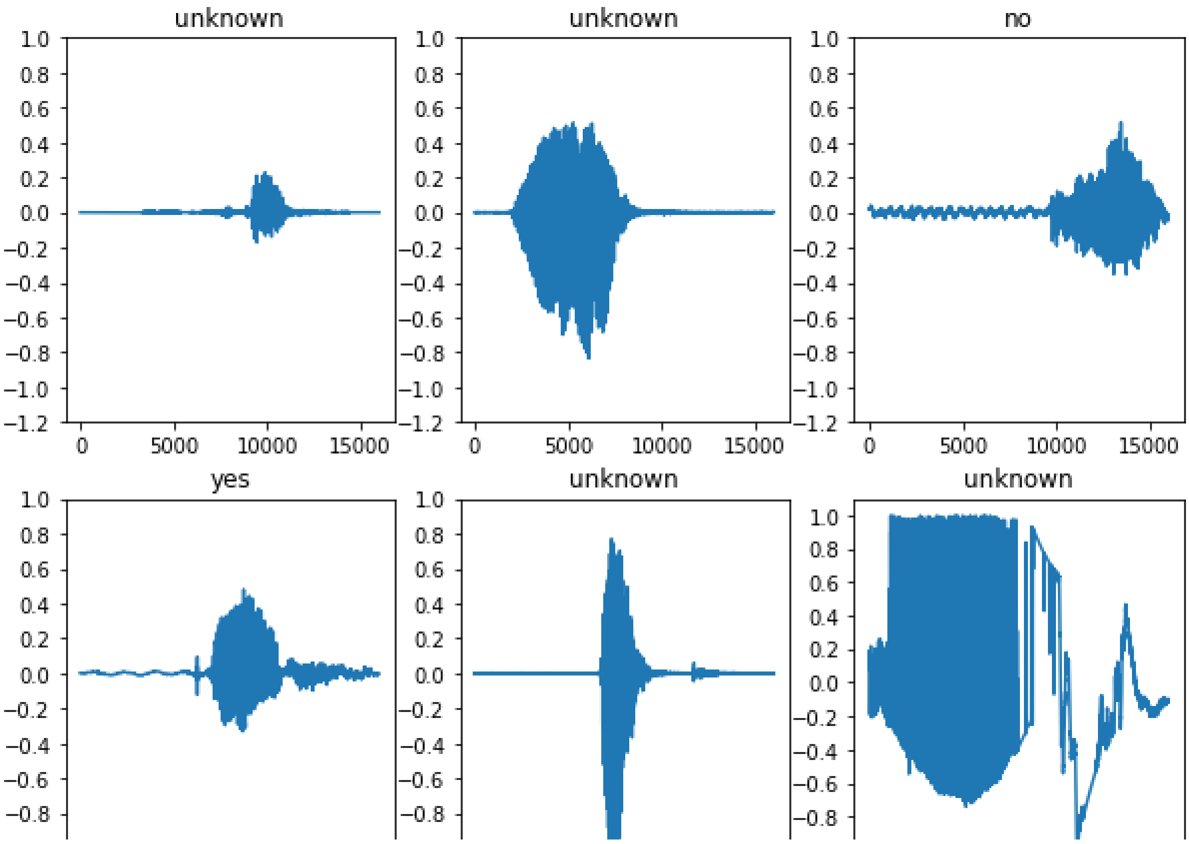
**## Let's define a method that will take in the filename of the WAV file and output a tuple containing the audio and labels for supervised training.**

```
def get_waveform_and_label(file_path):
  label = get_label(file_path)
  audio_binary = tf.io.read_file(file_path)
  waveform = decode_audio(audio_binary)
  return waveform, label
```

```
AUTOTUNE = tf.data.AUTOTUNE
files_ds = tf.data.Dataset.from_tensor_slices(train_files)
waveform_ds = files_ds.map(get_waveform_and_label, num_parallel_calls=AUTOTUNE)
```

***## Let's examine a few audio waveforms with their corresponding labels.***

```
rows = 3
cols = 3
n = rows*cols
fig, axes = plt.subplots(rows, cols, figsize=(10, 12))
for i, (audio, label) in enumerate(waveform_ds.take(n)):
  r = i // cols
  c = i % cols
  ax = axes[r][c]
  ax.plot(audio.numpy())
  ax.set_yticks(np.arange(-1.2, 1.2, 0.2))
  label = label.numpy().decode('utf-8')
  ax.set_title(label)

plt.show()
```



# Step 5:- Spectrogram

*## We converted the waveform into a spectrogram, which shows frequency changes over time and can be represented as a 2D image. This can be done by applying the short-time Fourier transform (STFT) to convert the audio into the time-frequency domain.*

*## A Fourier transform ( `tf.signal.fft` ) converts a signal to its component frequencies, but loses all time information. The STFT ( `tf.signal.stft` ) splits the signal into windows of time and runs a Fourier transform on each window, preserving some time information, and returning a 2D tensor that you can run standard convolutions on.*

*## STFT produces an array of complex numbers representing magnitude and phase. However, you'll only need the magnitude for this tutorial, which can be derived by applying `tf.abs` on the output of `tf.signal.stft`.*

*## We can Choose `frame_length` and `frame_step` parameters such that the generated spectrogram "image" is almost square. For more information on STFT parameters choice, we can refer to this video on audio signal processing.*

*## We also want the waveforms to have the same length, so that when we convert it to a spectrogram image, the results will have similar dimensions. This can be done by simply zero padding the audio clips that are shorter than one second.*

```
def get_spectrogram(waveform):
  # Padding for files with less than 16000 samples
  zero_padding = tf.zeros([16000] - tf.shape(waveform), dtype=tf.float32)
  # Concatenate audio with padding so that all audio clips will be of the
  # same length
  waveform = tf.cast(waveform, tf.float32)
  equal_length = tf.concat([waveform, zero_padding], 0)
  spectrogram = tf.signal.stft(
      equal_length, frame_length=480, frame_step=320, fft_length=512)

  spectrogram = tf.abs(spectrogram)

  return spectrogram
```

***## Next, we will explore the data. Compare the waveform, the spectrogram and the actual audio of one example from the dataset.***

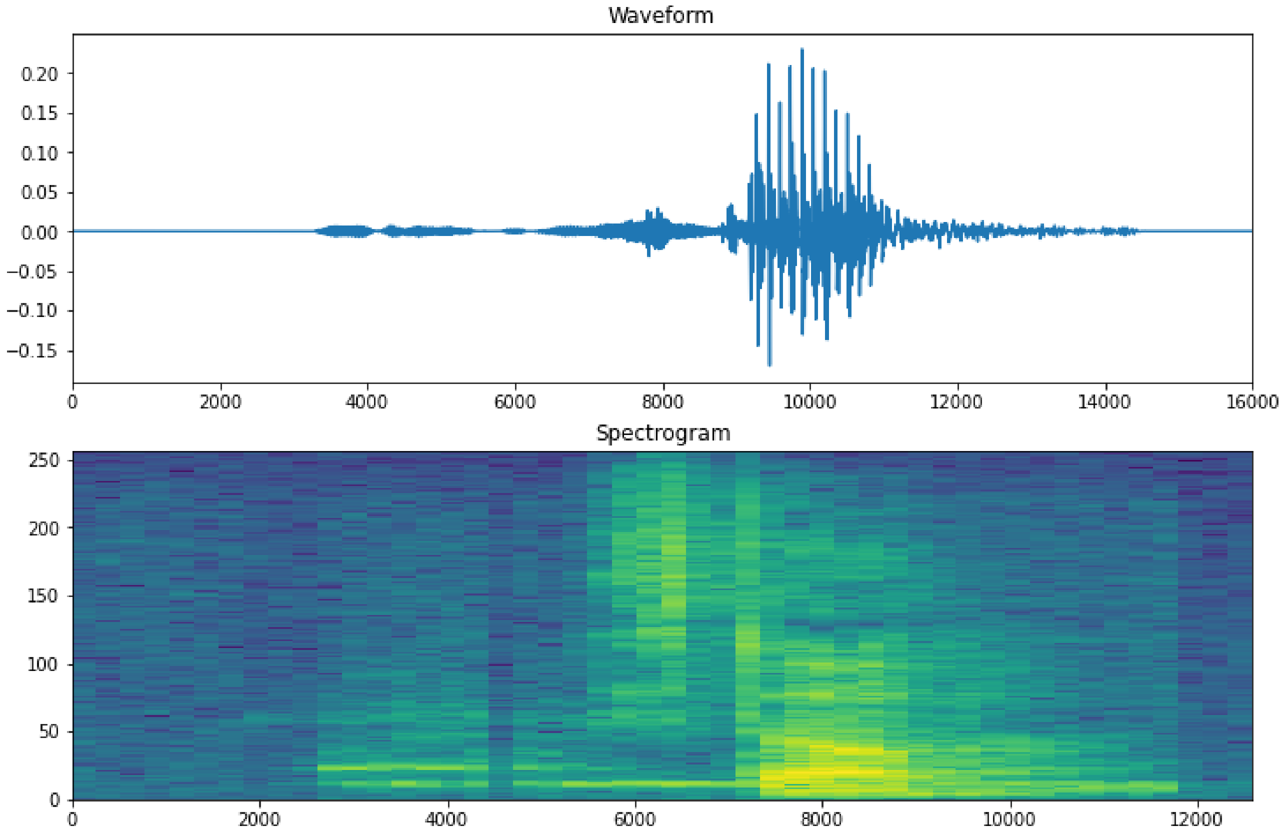
```
for waveform, label in waveform_ds.take(1):
  label = label.numpy().decode('utf-8')
  spectrogram = get_spectrogram(waveform)

print('Label:', label)
print('Waveform shape:', waveform.shape)
print('Spectrogram shape:', spectrogram.shape)
print('Audio playback')
display.display(display.Audio(waveform, rate=16000))
```

```
Label: unknown
Waveform shape: (16000,)
Spectrogram shape: (49, 257)
Audio playback
```

0:00 / 0:01

```
def plot_spectrogram(spectrogram, ax):
  # Convert to frequencies to log scale and transpose so that the time is
  # represented in the x-axis (columns).
  log_spec = np.log(spectrogram.T)
  height = log_spec.shape[0]
  width = log_spec.shape[1]
  X = np.linspace(0, np.size(spectrogram), num=width, dtype=int)
  Y = range(height)
  ax.pcolormesh(X, Y, log_spec)
```

```
fig, axes = plt.subplots(2, figsize=(12, 8))
timescale = np.arange(waveform.shape[0])
axes[0].plot(timescale, waveform.numpy())
axes[0].set_title('Waveform')
axes[0].set_xlim([0, 16000])
plot_spectrogram(spectrogram.numpy(), axes[1])
axes[1].set_title('Spectrogram')
plt.show()
```

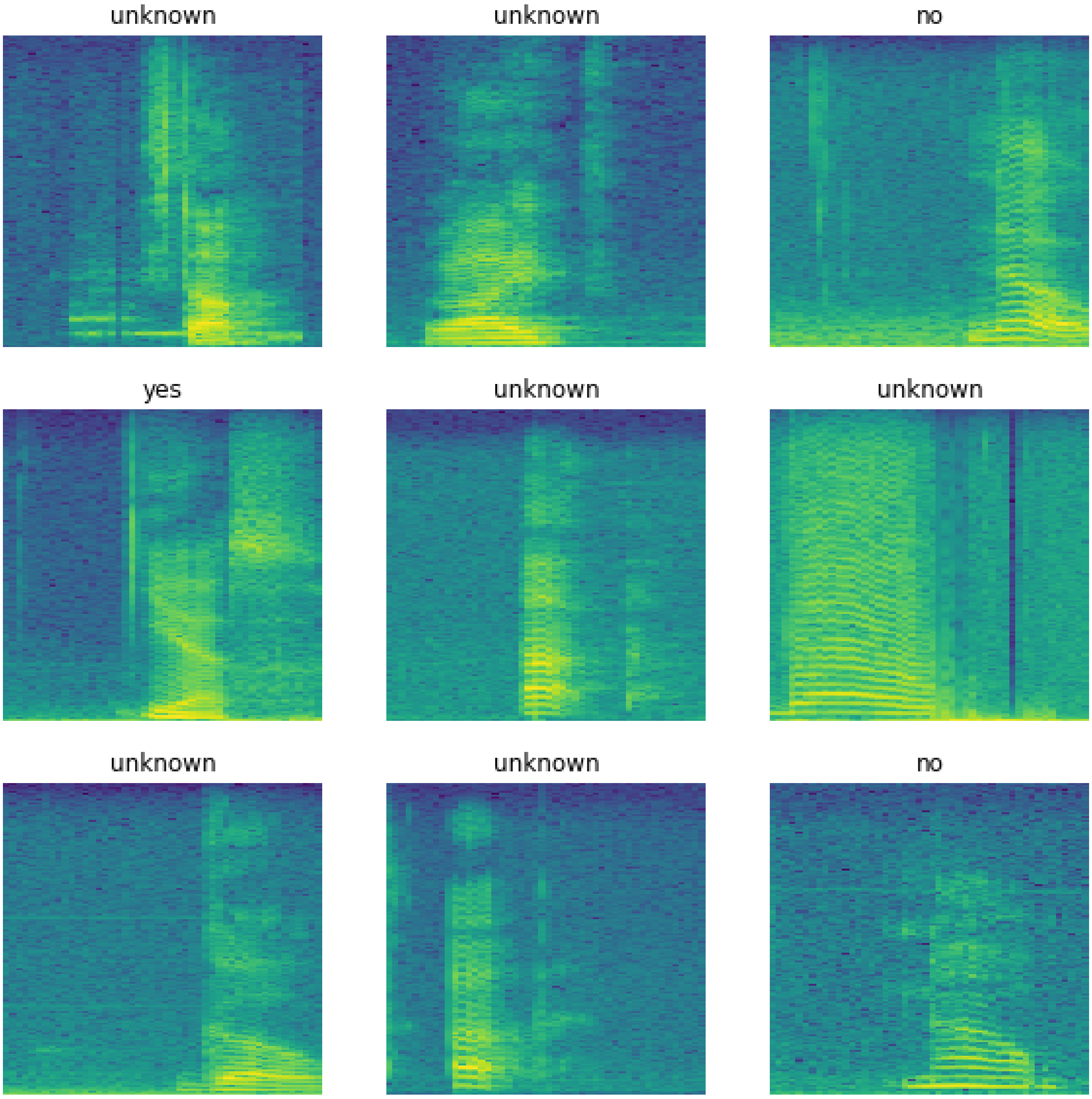


*## Now transform the waveform dataset to have spectrogram images and their corresponding labels as integer IDs.*

```
def get_spectrogram_and_label_id(audio, label):
  spectrogram = get_spectrogram(audio)
  spectrogram = tf.expand_dims(spectrogram, -1)
  label_id = tf.argmax(label == commands)
  return spectrogram, label_id


spectrogram_ds = waveform_ds.map(
    get_spectrogram_and_label_id, num_parallel_calls=AUTOTUNE)
```

*## Examine the spectrogram "images" for different samples of the dataset.*

```
rows = 3
cols = 3
n = rows*cols
fig, axes = plt.subplots(rows, cols, figsize=(10, 10))
for i, (spectrogram, label_id) in enumerate(spectrogram_ds.take(n)):
  r = i // cols
  c = i % cols
  ax = axes[r][c]
  plot_spectrogram(np.squeeze(spectrogram.numpy()), ax)
  ax.set_title(commands[label_id.numpy()])
  ax.axis('off')

plt.show()
```



# Step:-6 Build and train the model

***## Now we can build and train our model. But before we do that, we'll need to repeat the training set preprocessing on the validation and test sets.***

```
def preprocess_dataset(files):
  files_ds = tf.data.Dataset.from_tensor_slices(files)
  output_ds = files_ds.map(get_waveform_and_label, num_parallel_calls=AUTOTUNE)
  output_ds = output_ds.map(
      get_spectrogram_and_label_id,  num_parallel_calls=AUTOTUNE)
  return output_ds


train_ds = spectrogram_ds
val_ds = preprocess_dataset(val_files)
test_ds = preprocess_dataset(test_files)
```

```
print(val_ds)
print(test_ds)
```

```
<ParallelMapDataset element_spec=(TensorSpec(shape=(None, 257, 1), dtype=tf.float32,
<ParallelMapDataset element_spec=(TensorSpec(shape=(None, 257, 1), dtype=tf.float32,
```

***## Batch the training and validation sets for model training.***

```
batch_size = 64
train_ds = train_ds.batch(batch_size)
val_ds = val_ds.batch(batch_size)
```

***## Add dataset `cache()` and `prefetch()` operations to reduce read latency while training the model.***

```
train_ds = train_ds.cache().prefetch(AUTOTUNE)
val_ds = val_ds.cache().prefetch(AUTOTUNE)
```

***## For the model, we'll use a simple LSTM network, since we have transformed the audio files into spectrogram images.***

***## The model also has the following additional preprocessing layers:***

- A `Resizing` layer to downsample the input to enable the model to train faster.
- A `Normalization` layer to normalize each pixel in the image based on its mean and standard deviation.

***## For the `Normalization` layer, its `adapt` method would first need to be called on the training data in order to compute aggregate statistics (i.e. mean and standard deviation).***

```
for spectrogram, _ in spectrogram_ds.take(1):
  input_shape = spectrogram.shape
print('Input shape:', input_shape)
num_labels = len(commands)
print('num_labels:', num_labels)

model = models.Sequential([
    layers.Input(shape=(49, 257), name='input'),
    layers.Reshape(target_shape=(49, 257)),
    layers.LSTM(80, time_major=False, return_sequences=True),
    layers.Flatten(),
    layers.Dense(3, activation=tf.nn.softmax, name='output')
])
model.summary()
```

```
Input shape: (49, 257, 1)
```

```
num_labels: 3
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 reshape_1 (Reshape)         (None, 49, 257)           0         
                                                                 
 lstm_1 (LSTM)               (None, 49, 80)            108160    
                                                                 
 flatten_1 (Flatten)         (None, 3920)              0         
                                                                 
 output (Dense)              (None, 3)                 11763     
                                                                 
=================================================================
Total params: 119,923
Trainable params: 119,923
Non-trainable params: 0
_________________________________________________________________
```

## ## Loss function and Optimizer used

```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

## ## Model fit

```
EPOCHS = 100
history = model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=EPOCHS,
   # callbacks=tf.keras.callbacks.EarlyStopping(verbose=1, patience=2),
)
```

```
67/67 [==============================] - 0s 6ms/step - loss: 0.0061 - accuracy: 0.9
Epoch 73/100
67/67 [==============================] - 0s 6ms/step - loss: 0.0059 - accuracy: 0.9
Epoch 74/100
67/67 [==============================] - 0s 6ms/step - loss: 0.0058 - accuracy: 0.9
Epoch 75/100
67/67 [==============================] - 0s 6ms/step - loss: 0.0057 - accuracy: 0.9
Epoch 76/100
67/67 [==============================] - 0s 6ms/step - loss: 0.0053 - accuracy: 0.9
Epoch 77/100
67/67 [==============================] - 0s 6ms/step - loss: 0.0054 - accuracy: 0.9
Epoch 78/100
67/67 [==============================] - 0s 6ms/step - loss: 0.0049 - accuracy: 0.9
Epoch 79/100
67/67 [==============================] - 0s 6ms/step - loss: 0.0048 - accuracy: 0.9
Epoch 80/100
67/67 [==============================] - 0s 6ms/step - loss: 0.0053 - accuracy: 0.9
Epoch 81/100
67/67 [==============================] - 0s 6ms/step - loss: 0.0185 - accuracy: 0.9
Epoch 82/100
67/67 [==============================] - 0s 6ms/step - loss: 0.0114 - accuracy: 0.9
```

```
Epoch 83/100
67/67 [==============================] - 0s 6ms/step - loss: 0.0072 - accuracy: 0.
Epoch 84/100
67/67 [==============================] - 0s 6ms/step - loss: 0.0058 - accuracy: 0.
Epoch 85/100
67/67 [==============================] - 0s 6ms/step - loss: 0.0053 - accuracy: 0.
Epoch 86/100
67/67 [==============================] - 0s 6ms/step - loss: 0.0051 - accuracy: 0.
Epoch 87/100
67/67 [==============================] - 0s 6ms/step - loss: 0.0049 - accuracy: 0.
Epoch 88/100
67/67 [==============================] - 0s 6ms/step - loss: 0.0047 - accuracy: 0.
Epoch 89/100
67/67 [==============================] - 0s 6ms/step - loss: 0.0047 - accuracy: 0.
Epoch 90/100
67/67 [==============================] - 0s 6ms/step - loss: 0.0116 - accuracy: 0.
Epoch 91/100
67/67 [==============================] - 0s 6ms/step - loss: 0.0310 - accuracy: 0.
Epoch 92/100
67/67 [==============================] - 0s 6ms/step - loss: 0.0345 - accuracy: 0.
Epoch 93/100
67/67 [==============================] - 0s 6ms/step - loss: 0.0479 - accuracy: 0.
Epoch 94/100
67/67 [==============================] - 0s 6ms/step - loss: 0.0230 - accuracy: 0.
Epoch 95/100
67/67 [==============================] - 0s 6ms/step - loss: 0.0161 - accuracy: 0.
Epoch 96/100
67/67 [==============================] - 0s 6ms/step - loss: 0.0113 - accuracy: 0.
Epoch 97/100
67/67 [==============================] - 0s 6ms/step - loss: 0.0076 - accuracy: 0.
Epoch 98/100
67/67 [==============================] - 0s 7ms/step - loss: 0.0051 - accuracy: 0.
Epoch 99/100
67/67 [==============================] - 0s 6ms/step - loss: 0.0047 - accuracy: 0.
Epoch 100/100
67/67 [==============================] - 0s 6ms/step - loss: 0.0045 - accuracy: 0.
```

```
# Save the entire model as a SavedModel.
!mkdir -p saved_model
model.save('saved_model/my_model')
```

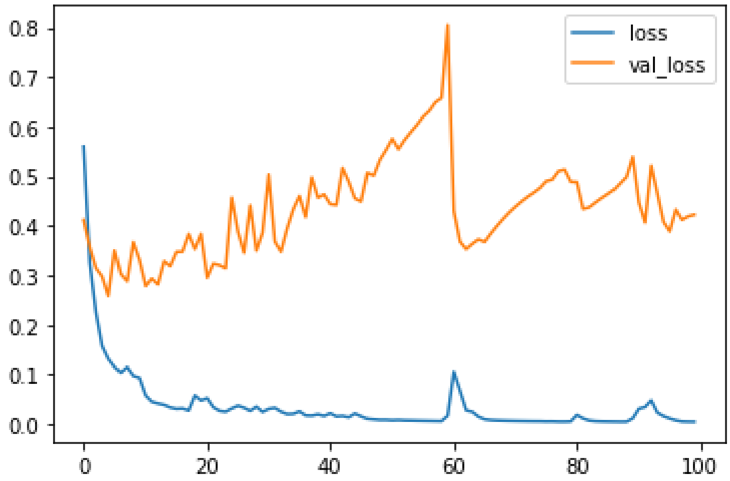
```
WARNING:absl:Found untraced functions such as lstm_cell_1_layer_call_fn, lstm_cell_1_
```

```
print("Model:")
```

```
Model:
```

## *Let's check the training and validation loss & accuracy-val_accuracy curves to see how our model has improved during training.*

```
metrics = history.history
plt.plot(history.epoch, metrics['loss'], metrics['val_loss'])
plt.legend(['loss', 'val_loss'])
plt.show()
```



# Step 7:- Evaluate test set performance

***## Let's run the model on the test set and check performance.***

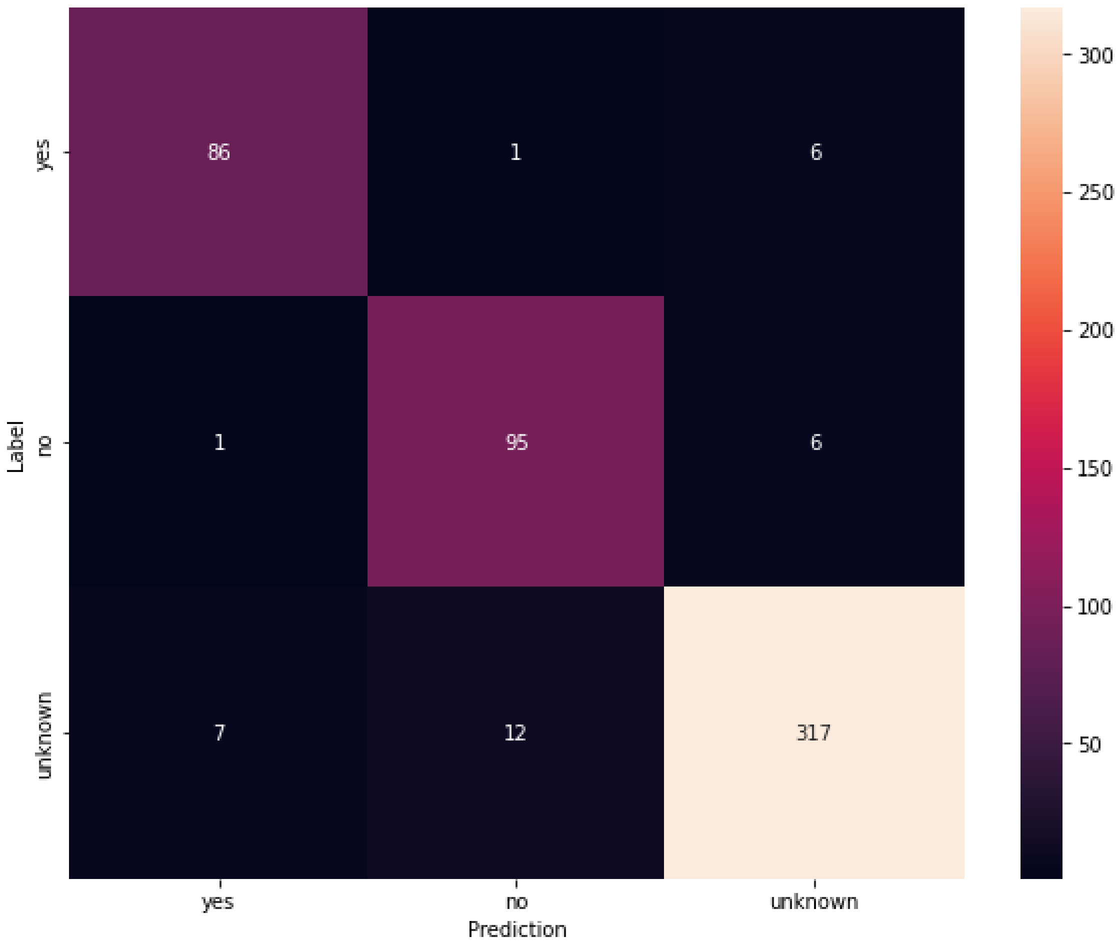
```
test_audio = []
test_labels = []

for audio, label in test_ds:
  test_audio.append(audio.numpy())
  test_labels.append(label.numpy())

test_audio = np.array(test_audio)
test_labels = np.array(test_labels)


y_pred = np.argmax(model.predict(test_audio), axis=1)
y_true = test_labels

test_acc = sum(y_pred == y_true) / len(y_true)
print(f'Test set accuracy: {test_acc:.0%}')
```

```
17/17 [==============================] - 0s 3ms/step
Test set accuracy: 94%
```

# Step 8:- Display a confusion matrix

***## A confusion matrix is helpful to see how well the model did on each of the commands in the test set.***

```
confusion_mtx = tf.math.confusion_matrix(y_true, y_pred)
plt.figure(figsize=(10, 8))
sns.heatmap(confusion_mtx, xticklabels=commands, yticklabels=commands,
            annot=True, fmt='g')
plt.xlabel('Prediction')
plt.ylabel('Label')
plt.show()
```

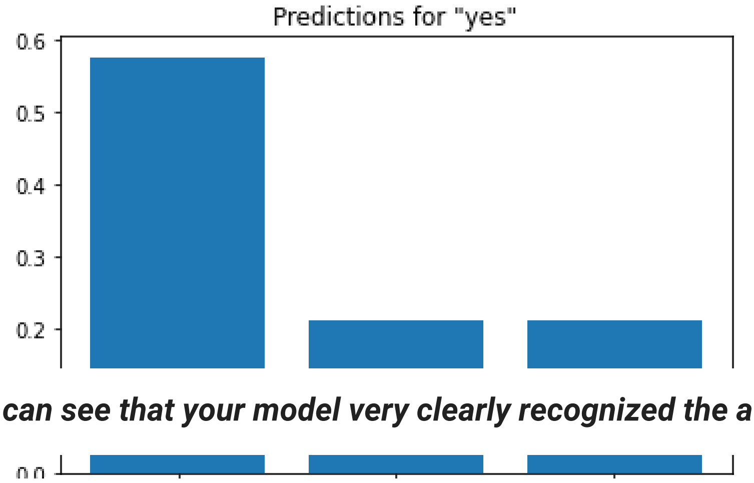


## Step 9:- Run inference on an audio file

***## Finally, verify the model's prediction output using an input audio file of someone saying "yes." How well does your model perform?***

```
sample_file = '/content/data/mini_speech_commands/yes/0132a06d_nohash_1.wav'

sample_ds = preprocess_dataset([str(sample_file)])
for spectrogram, label in sample_ds.batch(1):
  prediction = model(spectrogram)
  print(len(commands))
  print(tf.nn.softmax(prediction[0]))
  print(tf.nn.softmax(prediction))
  plt.bar(commands, tf.nn.softmax(prediction[0]))
  plt.title(f'Predictions for "{commands[label[0]]}"')
  plt.show()
```

```
3
tf.Tensor([0.57611686 0.21194158 0.2119416 ], shape=(3,), dtype=float32)
tf.Tensor([[0.57611686 0.21194158 0.2119416 ]], shape=(1, 3), dtype=float32)
```



***--We can see that your model very clearly recognized the audio command as "yes."***

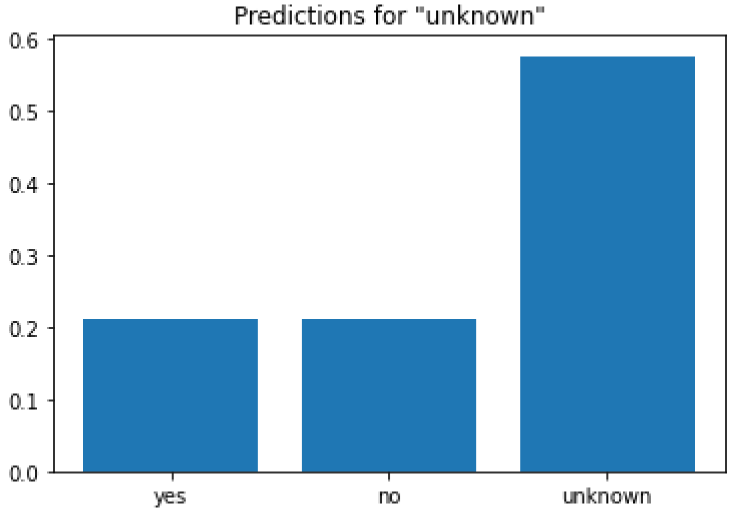
```
model.save("./saved_model.h5/")
```

```
WARNING:absl:Found untraced functions such as lstm_cell_1_layer_call_fn, lstm_cell_1_
```

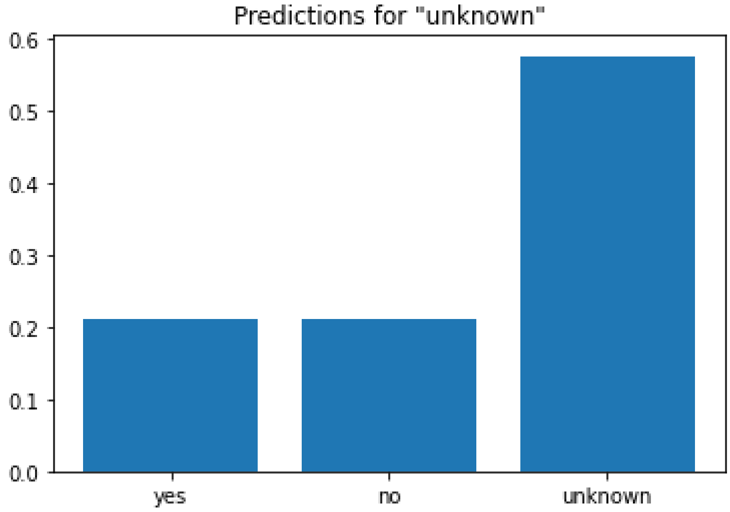
## Step 10:- Run TF (Tensorflow) inference on multiple audio files

```
import glob
import pandas as pd
pd.set_option("display.precision", 2)

txtfiles = []
for file in glob.glob("/content/data/mini_speech_commands/unknown/*.wav"):
    txtfiles.append(file)

for i in range(25):
  print(txtfiles[i])
  sample_ds = preprocess_dataset([str(txtfiles[i])])
  for spectrogram, label in sample_ds.batch(1):
    prediction = model(spectrogram)
    plt.bar(commands, tf.nn.softmax(prediction[0]))
    plt.title(f'Predictions for "{commands[label[0]]}"')
    plt.show()
```

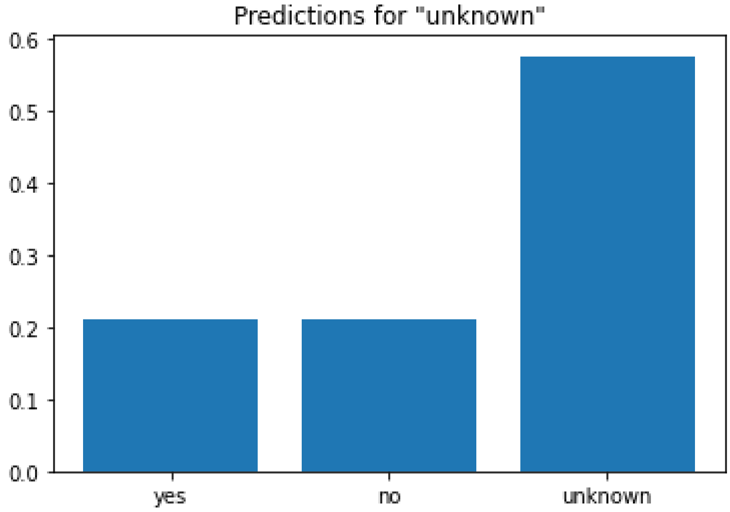
```
/content/data/mini_speech_commands/unknown/48a9f771_nohash_0.wav
```



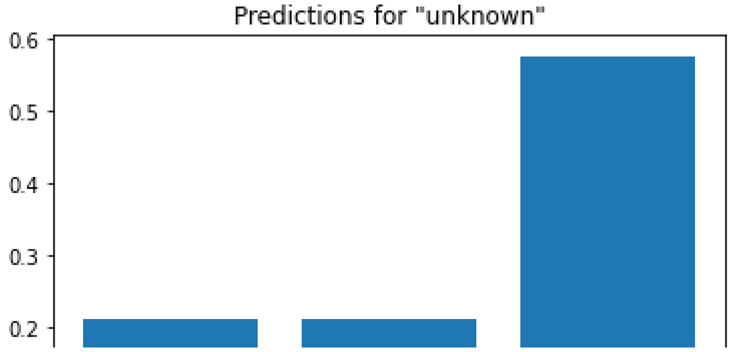
```
/content/data/mini_speech_commands/unknown/26e573a9_nohash_1.wav
```

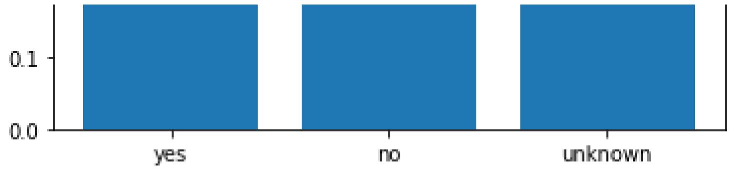


```
/content/data/mini_speech_commands/unknown/1657c9fa_nohash_1.wav
```

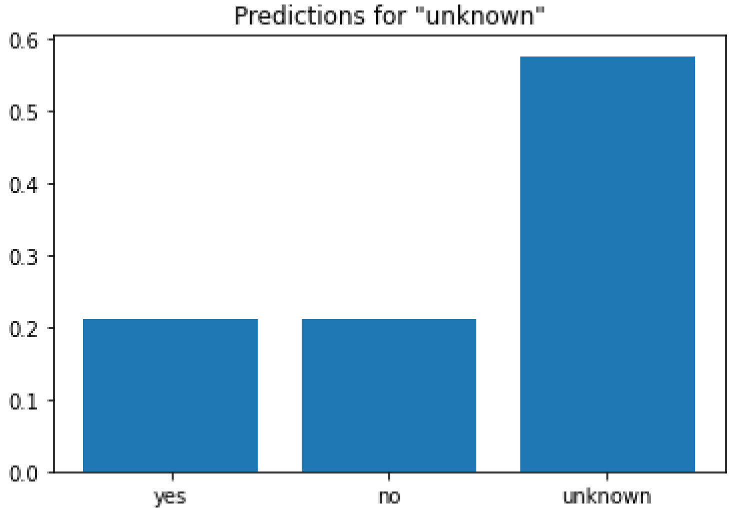


```
/content/data/mini_speech_commands/unknown/3a33d3a4_nohash_1.wav
```

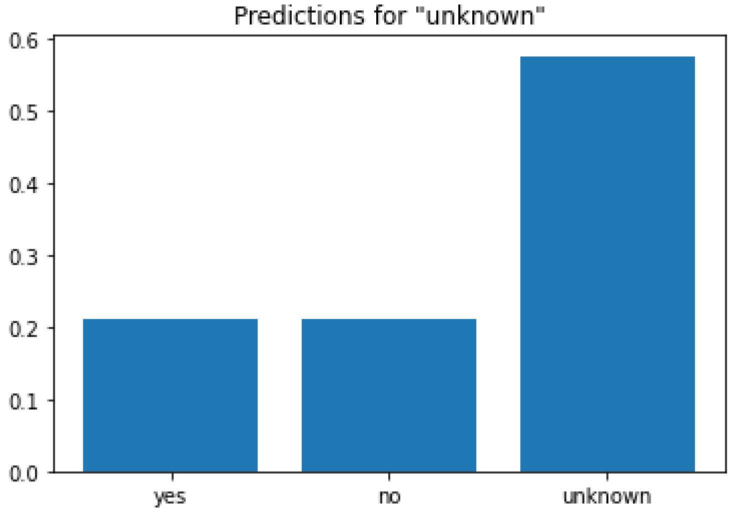




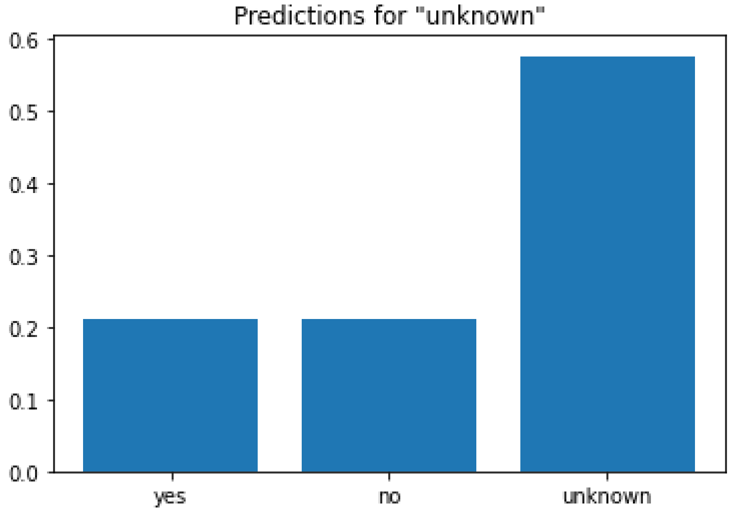
```
/content/data/mini_speech_commands/unknown/bf5d409d_nohash_1.wav
```



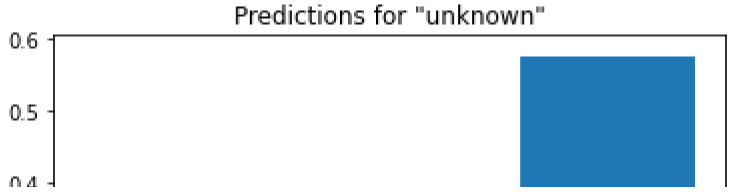
```
/content/data/mini_speech_commands/unknown/784e281a_nohash_0.wav
```

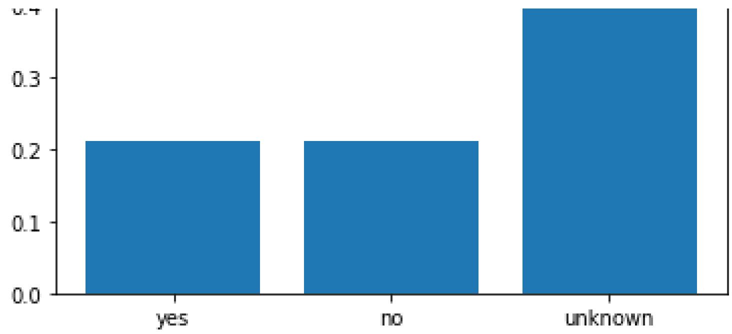


```
/content/data/mini_speech_commands/unknown/f6617a86_nohash_1.wav
```

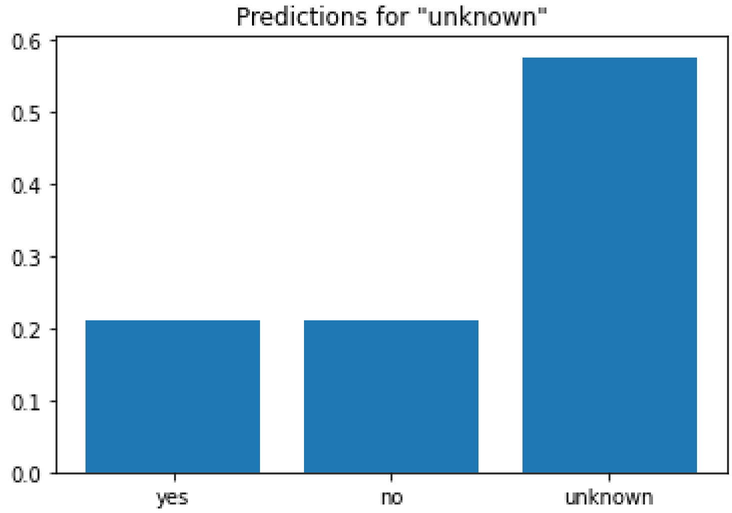


```
/content/data/mini_speech_commands/unknown/a2fefcb4_nohash_0.wav
```

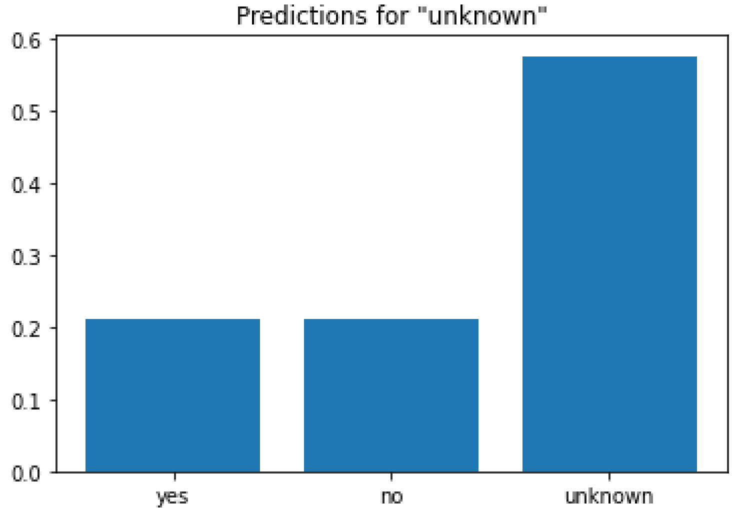




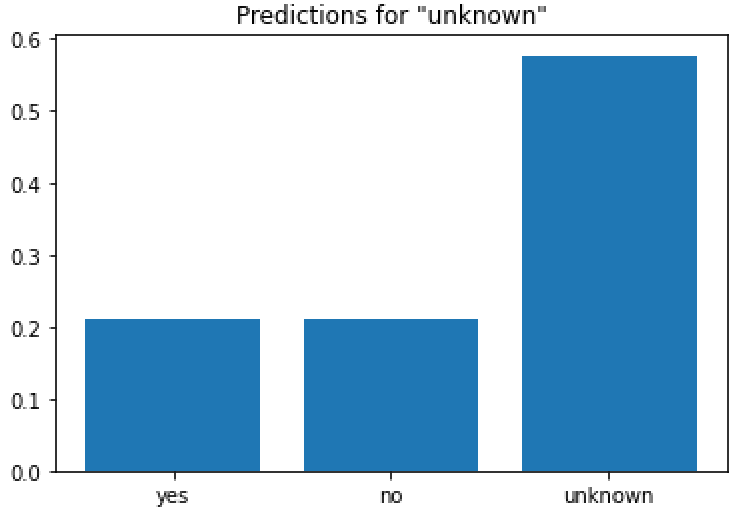
```
/content/data/mini_speech_commands/unknown/3e2ba5f7_nohash_1.wav
```



```
/content/data/mini_speech_commands/unknown/0474c92a_nohash_0.wav
```

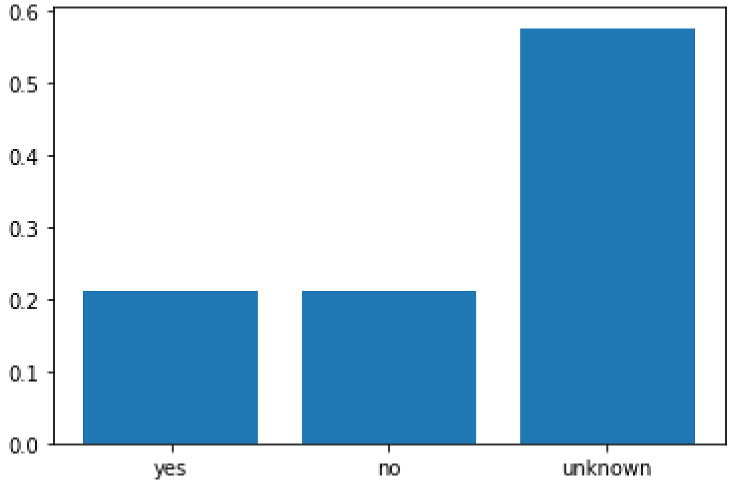


```
/content/data/mini_speech_commands/unknown/da1d320c_nohash_1.wav
```

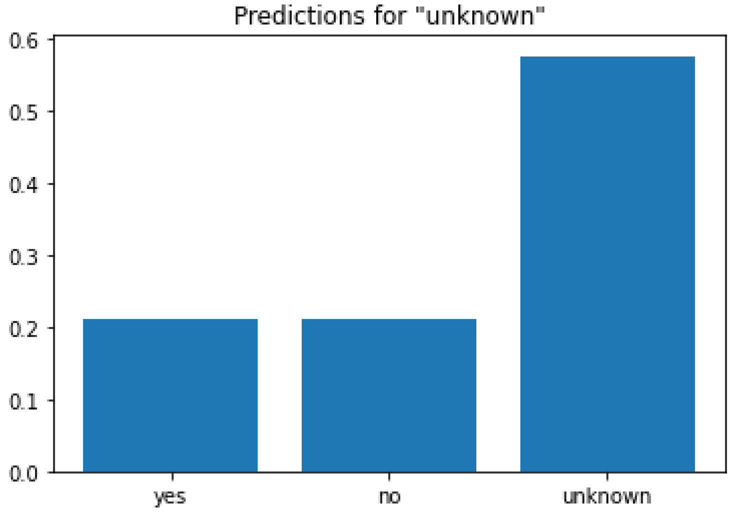


```
/content/data/mini_speech_commands/unknown/edd8bfe3_nohash_1.wav
```

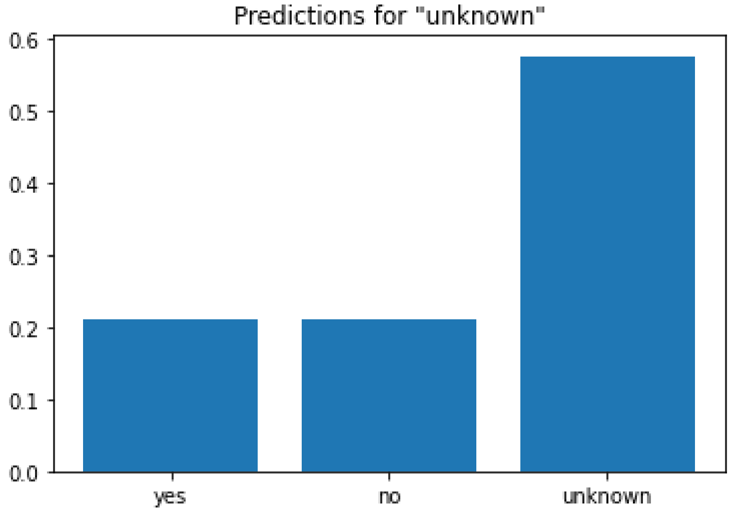
Predictions for "unknown"



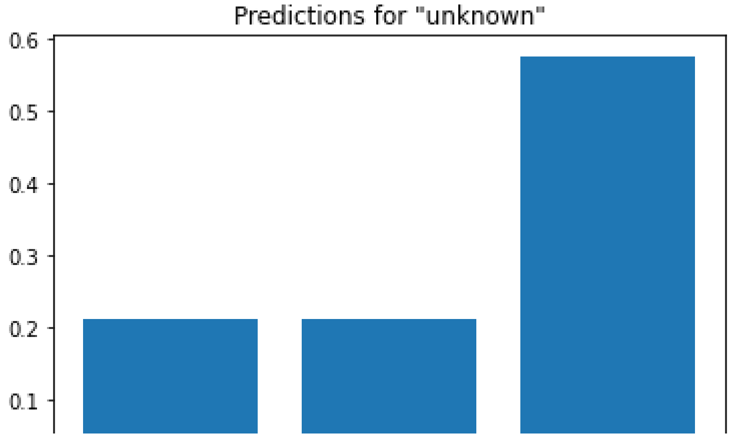
```
/content/data/mini_speech_commands/unknown/0ff728b5_nohash_1.wav
```

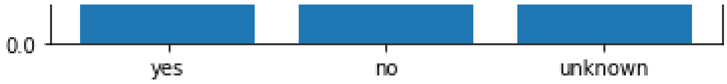

Predictions for "unknown"

```
/content/data/mini_speech_commands/unknown/5b26c81b_nohash_0.wav
```

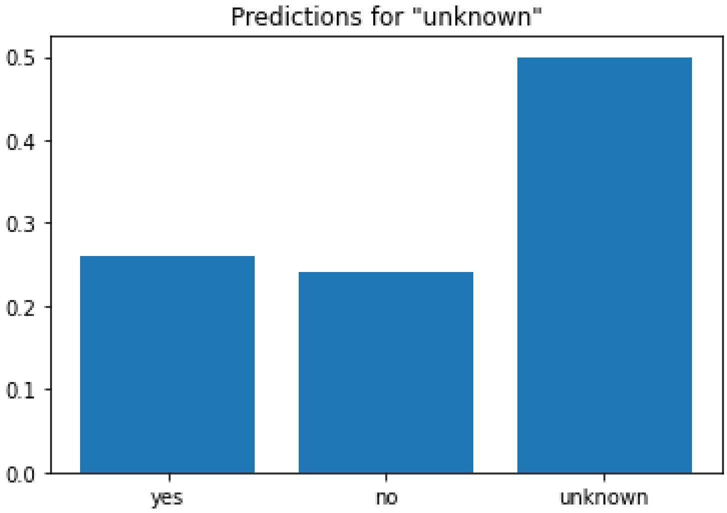

Predictions for "unknown"

```
/content/data/mini_speech_commands/unknown/28e47b1a_nohash_3.wav
```

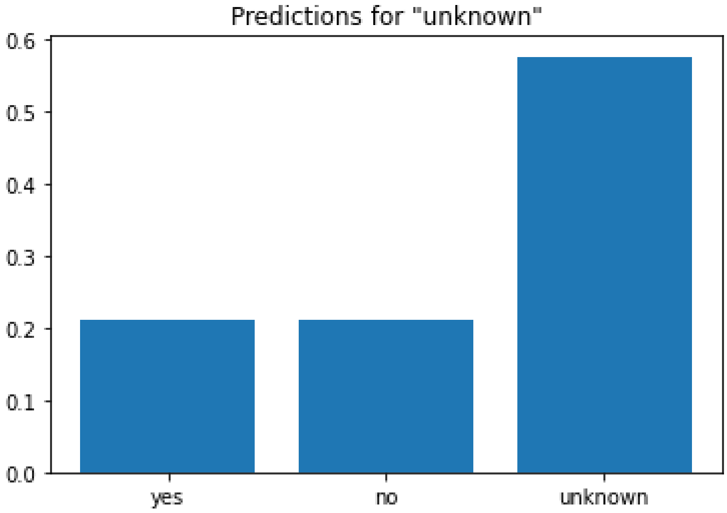

Predictions for "unknown"



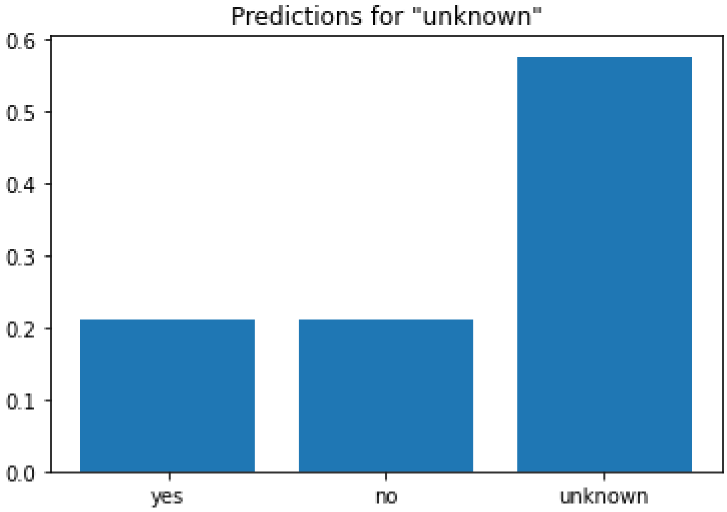
```
/content/data/mini_speech_commands/unknown/ced835d3_nohash_3.wav
```


Predictions for "unknown"

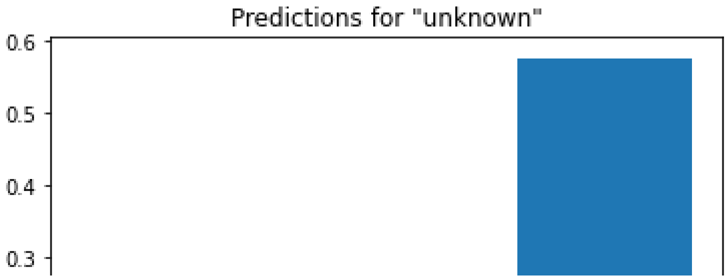
```
/content/data/mini_speech_commands/unknown/a1cff772_nohash_0.wav
```

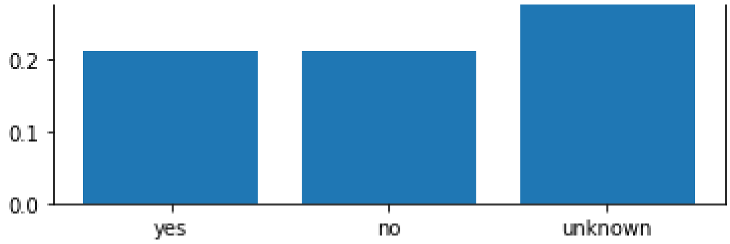

Predictions for "unknown"

```
/content/data/mini_speech_commands/unknown/9beccfc8_nohash_1.wav
```

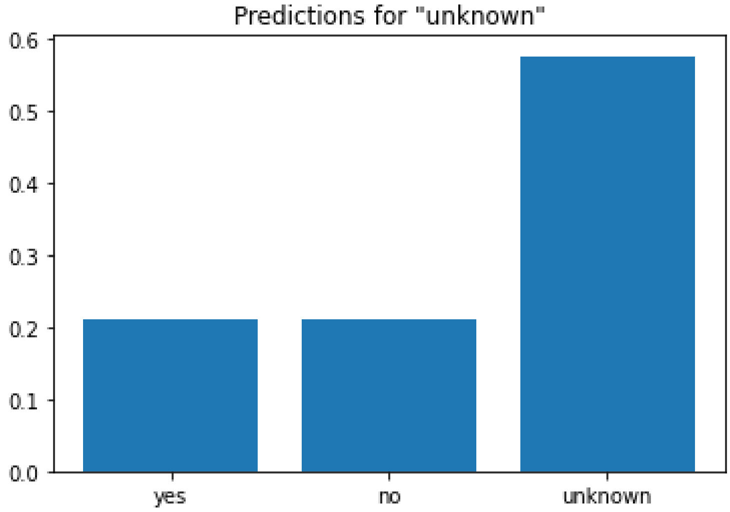

Predictions for "unknown"

```
/content/data/mini_speech_commands/unknown/6cf5459b_nohash_1.wav
```

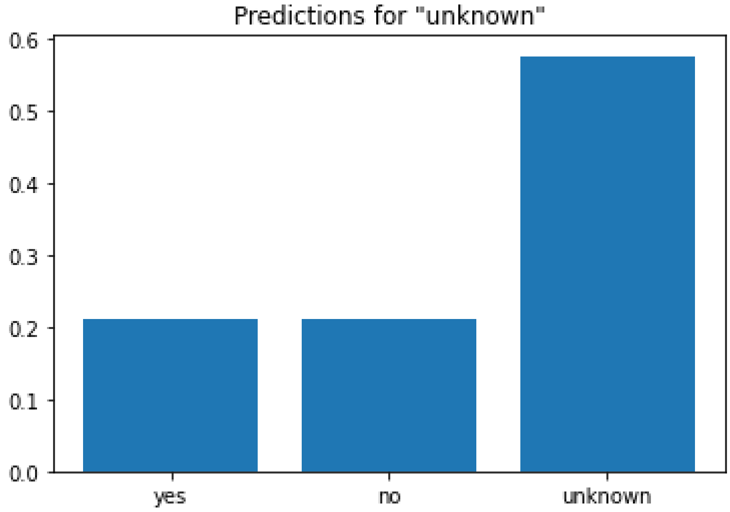

Predictions for "unknown"



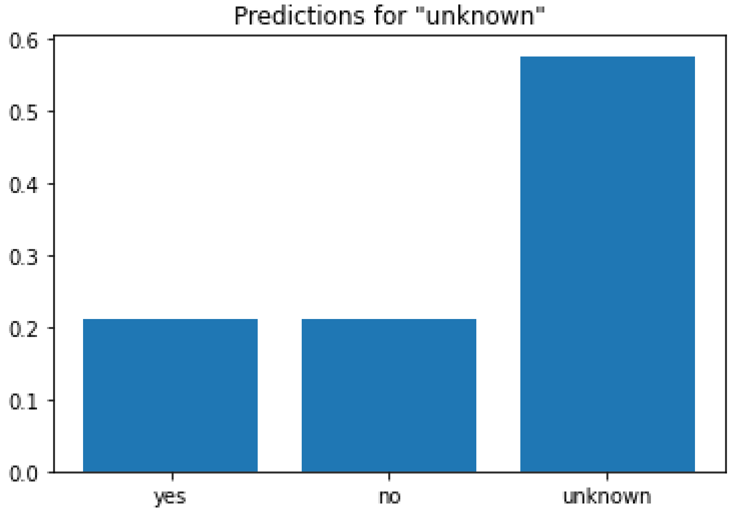
```
/content/data/mini_speech_commands/unknown/edd8bfe3_nohash_0.wav
```



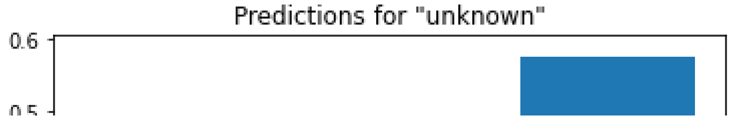
```
/content/data/mini_speech_commands/unknown/96ab6565_nohash_2.wav
```

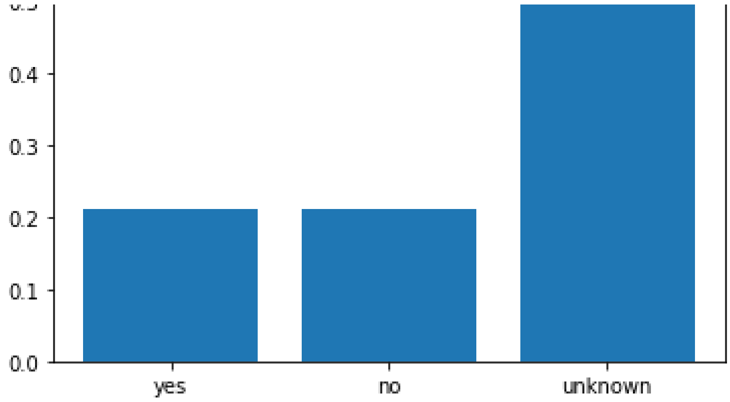


```
/content/data/mini_speech_commands/unknown/90b0b91a_nohash_1.wav
```

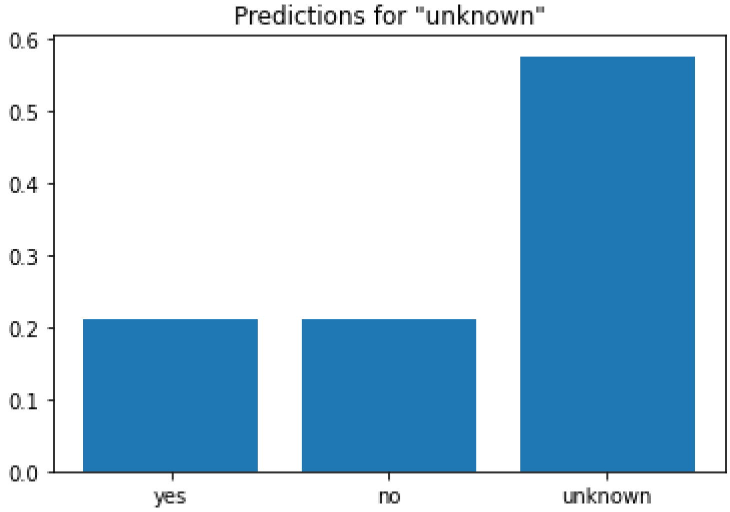


```
/content/data/mini_speech_commands/unknown/dedc7fab_nohash_1.wav
```

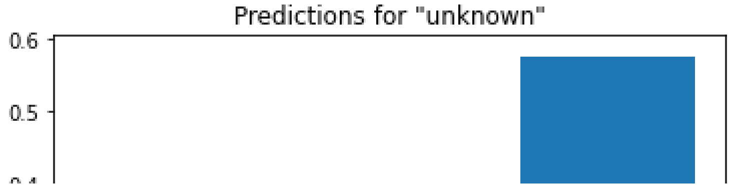




```
/content/data/mini_speech_commands/unknown/ca58a8c6_nohash_0.wav
```



```
/content/data/mini_speech_commands/unknown/fb7cfe0e_nohash_0.wav
```



## Step 11:- Trained model weights visualizations layerwise

### Step 11.1:- Weights for Layer [0]

```
model.get_weights()[0]
```

```
array([[-0.00563288,  0.02722069,  0.02178445, ...,  0.11146132,
        -0.04921982,  0.03150596],
       [-0.01186763,  0.04691156, -0.13949604, ...,  0.05764583,
        -0.04431278, -0.03428394],
       [-0.14132024,  0.00147425, -0.11869487, ...,  0.18961735,
         0.02061029, -0.2226579 ],
       ...,
       [ 0.09847562, -0.04395953,  0.05241163, ..., -0.09560476,
         0.05311788, -0.03395343],
       [ 0.04786448,  0.05726088,  0.03543131, ..., -0.1665768 ,
         0.1144514 , -0.06121617],
       [ 0.05060462,  0.04284142,  0.08905675, ..., -0.10912168,
        -0.01330861, -0.12079769]], dtype=float32)
```

### *Step 11.2:- Weights for Layer [1]*

```
model.get_weights()[1]
```

```
    array([[ 0.06777401,  0.06759165, -0.11989291, ..., -0.01104744,
            -0.04181176, -0.12480928],
           [-0.11874168,  0.01187701,  0.00720231, ...,  0.08293021,
            -0.0663819 , -0.03104337],
           [ 0.15734038, -0.1954211 , -0.03614312, ..., -0.04541576,
             0.01764638, -0.07291853],
           ...,
           [-0.09940577,  0.12286972, -0.30298692, ...,  0.08476383,
             0.01410181, -0.05769118],
           [-0.03236272,  0.04511095, -0.00150096, ...,  0.03450888,
            -0.00148458,  0.07267053],
           [ 0.09258701,  0.06765915,  0.10189385, ...,  0.07742485,
            -0.19675745,  0.0231267 ]], dtype=float32)
```

### *Step 11.3:- Weights for Layer [2]*

```
model.get_weights()[2]
```

```
            9.57526326e-01,  9.97431695e-01,  8.66218805e-01,  1.02473211e+00,
            1.10267115e+00,  1.00714552e+00,  9.51363444e-01,  9.29126859e-01,
            9.79502380e-01,  1.10982239e+00,  1.15015268e+00,  1.06933308e+00,
            9.71925318e-01,  1.01608205e+00,  9.85301197e-01,  1.11632621e+00,
            1.08904934e+00,  9.98482883e-01,  1.00662923e+00,  8.89398873e-01,
            9.66598511e-01,  1.00164521e+00,  9.69219387e-01,  9.81615841e-01,
            9.78496075e-01,  9.30068433e-01,  1.08858550e+00,  1.04268372e+00,
            1.03943658e+00,  9.98060167e-01,  1.03112936e+00,  9.04340446e-01,
            1.01481199e+00,  1.01915789e+00,  9.85788584e-01,  1.07223654e+00,
            9.64191139e-01,  9.75218356e-01,  9.54456747e-01,  9.18242931e-01,
            9.21438098e-01,  1.02076674e+00,  1.02725565e+00,  9.94487286e-01,
            9.68944848e-01,  1.01321733e+00,  1.07815278e+00,  9.77605164e-01,
            9.06899571e-01,  1.11797512e+00,  1.01943362e+00,  1.00336015e+00,
            1.04066145e+00,  9.65861440e-01,  9.83944893e-01,  1.14388287e+00,
            1.03661418e+00,  9.12949681e-01,  1.04882288e+00,  9.84841287e-01,
            1.03915215e+00,  1.12184322e+00,  9.94718552e-01,  1.02396071e+00,
            9.55791354e-01,  1.07630527e+00,  8.85838568e-01,  1.01847661e+00,
           -6.87905448e-03,  2.26764567e-02,  1.61300693e-02, -4.78645694e-03,
           -3.10650449e-02, -3.40941325e-02,  1.86896343e-02,  8.92082788e-03,
           -1.12632727e-02, -4.27742349e-03, -1.09403403e-02, -1.16341393e-02,
           -1.27771422e-02,  5.80965132e-02,  3.48129161e-02, -5.40827110e-04,
            9.86322854e-03,  1.41829094e-02,  1.41390378e-03, -1.77349355e-02,
           -4.02648561e-02, -2.20553763e-02, -1.13608371e-02,  1.75506119e-02,
           -2.14500669e-02, -3.69772837e-02,  2.29502581e-02, -1.24770729e-02,
           -5.55435419e-02,  1.58325657e-02,  1.15060564e-02, -3.48312296e-02,
           -1.16121890e-02, -5.87414484e-03, -4.99279574e-02,  1.39855286e-02,
            5.39994389e-02, -1.30007407e-02, -3.35767567e-02, -4.65596244e-02,
            5.79028251e-03,  4.01817597e-02, -1.38439750e-02, -2.52826065e-02,
            2.35471502e-03, -1.32176848e-02,  1.31066060e-02, -1.73702072e-02,
            2.99341208e-03,  3.02875526e-02,  6.13649329e-03, -3.80107835e-02,
            8.92498344e-02,  3.36106718e-02,  1.37243960e-02,  2.57514641e-02,
           -1.37995910e-02,  2.07069535e-02, -1.94300748e-02,  7.80594209e-03,
            1.69660319e-02, -2.07808986e-02,  1.87570509e-02,  4.19962918e-03,
            1.31348753e-02,  2.73659378e-02, -6.34227600e-03,  2.08946858e-02,
            2.76989844e-02,  1.24754542e-02,  1.24331750e-03,  3.50610279e-02
```

```
        2.76989844e-02,  1.24754542e-02,  1.24331750e-03,  3.50610279e-02,
        8.38968065e-03,  1.73458811e-02,  3.79903913e-02,  3.86711061e-02,
       -3.02877147e-02, -6.84603080e-02,  6.03204546e-03,  5.75914122e-02,
        7.32691810e-02, -1.00181192e-01, -8.78253428e-04, -3.94931762e-03,
        1.24641173e-02,  3.15759927e-02, -1.63893700e-02,  5.58563136e-02,
        8.36112946e-02, -9.35319439e-02, -8.45441967e-02, -4.42930311e-02,
       -7.39627182e-02, -3.11709289e-03,  3.02279051e-02, -1.27332797e-03,
        5.26266769e-02, -8.00020546e-02, -1.78254209e-02, -8.73361826e-02,
       -3.86155471e-02,  2.00129122e-01,  4.14125435e-02, -3.53166834e-02,
        9.49775502e-02, -4.39387601e-04, -3.78712825e-02,  5.03545478e-02,
        7.97494203e-02, -1.86536964e-02,  7.75164180e-03, -3.42939422e-02,
       -8.76416042e-02, -1.15766712e-02, -3.89277488e-02,  1.36305951e-02,
       -3.55004035e-02, -2.46527586e-02,  5.07961847e-02,  4.12495323e-02,
        4.08685356e-02, -2.98484452e-02,  7.19511136e-02,  6.64878637e-03,
        3.72174121e-02,  1.29167000e-02, -4.48333509e-02,  5.37069403e-02,
       -3.97735499e-02, -5.36061302e-02, -4.31677401e-02,  5.16716903e-03,
        6.93309233e-02,  2.17184033e-02,  2.80777644e-02, -6.68777898e-03,
        4.29893956e-02, -6.58701034e-03,  2.30927672e-02, -2.21706461e-02,
       -3.88433821e-02, -4.11779322e-02, -2.08206475e-02,  1.99485980e-02,
       -2.79723760e-02, -7.11276233e-02, -6.48017451e-02,  4.48469780e-02,
        4.72299494e-02, -6.91493228e-02,  5.33572212e-02,  1.97497383e-02,
       -8.35282058e-02,  8.88175070e-02, -6.57924414e-02,  1.03326418e-01,
       -4.15926687e-02,  1.17592532e-02, -1.10870793e-01,  4.69601229e-02],
      dtype=float32)
```

### *Step 11.4:- Weights for Layer [3]*

```
model.get_weights()[3]
```

```
     array([[ 0.06568483,  0.23173128, -0.23441082],
            [-0.15249006, -0.05274262,  0.12696849],
            [-0.09750769, -0.03921815,  0.10007721],
            ...,
            [-0.17221509,  0.06782542, -0.03295461],
            [-0.16315362, -0.00453885,  0.15091746],
            [-0.11915002, -0.11885497,  0.16523157]], dtype=float32)
```

# *Step 12:- Apply Affine quantization scheme on Trained model weights layer by layer*

## *Step 12.1:- Quantization apply on Weights for Layer [0]*

```
T0 = np.array (model.get_weights()[0])
print(T0)
```

```
     [[-0.00563288  0.02722069  0.02178445 ...  0.11146132 -0.04921982
        0.03150596]
      [-0.01186763  0.04691156 -0.13949604 ...  0.05764583 -0.04431278
       -0.03428394]
      [-0.14132024  0.00147425 -0.11869487 ...  0.18961735  0.02061029
```

```
  -0.2226579 ]
 ...
 [ 0.09847562 -0.04395953  0.05241163 ... -0.09560476  0.05311788
  -0.03395343]
 [ 0.04786448  0.05726088  0.03543131 ... -0.1665768   0.1144514
  -0.06121617]
 [ 0.05060462  0.04284142  0.08905675 ... -0.10912168 -0.01330861
  -0.12079769]]
```

```
# Min-Max value of float_32 tensor (x) find out for scale (s) and zero point (z)

b0 = np.amax(T0)
print(b0)
```

```
0.5366043
```

```
a0 = np.amin(T0)
print(a0)
```

```
-0.47293857
```

```
# scale value

scale0 = (b0-a0)/15

print(scale0)
```

```
0.06730285485585531
```

```
# zero point

zero_point0 = np.round(-a0*15/(b0-a0))

print(zero_point0)
```

```
7.0
```

```
f0 = np.round(T0/scale0 + zero_point0)
print(f0)
T0q = np.clip(f0, a_min=0, a_max=15) # Here min & max value we can change as per Tbit.
# But, I have checked for 4 bit
print (T0q)
```

```
[[ 7.  7.  7. ...  9.  6.  7.]
 [ 7.  8.  5. ...  8.  6.  6.]
 [ 5.  7.  5. ... 10.  7.  4.]
 ...
 [ 8.  6.  8. ...  6.  8.  6.]
 [ 8.  8.  8. ...  5.  9.  6.]
 [ 8.  8.  8. ...  5.  7.  5.]]
[[ 7.  7.  7. ...  9.  6.  7.]
 [ 7.  8.  5. ...  8.  6.  6.]
 [ 5.  7.  5. ... 10.  7.  4.]
 ...
 [ 8.  6.  8. ...  6.  8.  6.]
 [ 8.  8.  8. ...  5.  9.  6.]
 [ 8.  8.  8. ...  5.  7.  5.]]
```

```
T0dq = scale0*(T0q - zero_point0)
print(T0dq)
```

```
[[ 0.          0.          0.         ...  0.1346057  -0.06730285
   0.        ]
 [ 0.          0.06730285 -0.1346057  ...  0.06730285 -0.06730285
  -0.06730285]
 [-0.1346057   0.         -0.1346057  ...  0.20190856  0.
  -0.20190856]
 ...
 [ 0.06730285 -0.06730285  0.06730285 ... -0.06730285  0.06730285
  -0.06730285]
 [ 0.06730285  0.06730285  0.06730285 ... -0.1346057   0.1346057
  -0.06730285]
 [ 0.06730285  0.06730285  0.06730285 ... -0.1346057   0.
  -0.1346057 ]]
```

```
# print input and target tensors
print("Input Tensor:\n", T0)
print("Target Tensor:\n", T0dq)
mse = tf.keras.losses.MeanAbsoluteError()
mse(T0dq, T0).numpy()
```

```
Input Tensor:
 [[-0.00563288  0.02722069  0.02178445 ...  0.11146132 -0.04921982
   0.03150596]
 [-0.01186763  0.04691156 -0.13949604 ...  0.05764583 -0.04431278
  -0.03428394]
 [-0.14132024  0.00147425 -0.11869487 ...  0.18961735  0.02061029
  -0.2226579 ]
 ...
 [ 0.09847562 -0.04395953  0.05241163 ... -0.09560476  0.05311788
  -0.03395343]
 [ 0.04786448  0.05726088  0.03543131 ... -0.1665768   0.1144514
  -0.06121617]
 [ 0.05060462  0.04284142  0.08905675 ... -0.10912168 -0.01330861
  -0.12079769]]
Target Tensor:
 [[ 0.          0.          0.         ...  0.1346057  -0.06730285
   0.        ]
 [ 0.          0.06730285 -0.1346057  ...  0.06730285 -0.06730285
  -0.06730285]
 [-0.1346057   0.         -0.1346057  ...  0.20190856  0.
  -0.20190856]
 ...
 [ 0.06730285 -0.06730285  0.06730285 ... -0.06730285  0.06730285
  -0.06730285]
 [ 0.06730285  0.06730285  0.06730285 ... -0.1346057   0.1346057
  -0.06730285]
 [ 0.06730285  0.06730285  0.06730285 ... -0.1346057   0.
  -0.1346057 ]]
0.016865954
```

```
# print input and target tensors
print("Input Tensor:\n", T0)
print("Target Tensor:\n", T0dq)
mse = tf.keras.losses.MeanSquaredError()
mse(T0dq, T0).numpy()
```

```
     Input Tensor:
      [[-0.00563288  0.02722069  0.02178445 ...  0.11146132 -0.04921982
        0.03150596]
      [-0.01186763  0.04691156 -0.13949604 ...  0.05764583 -0.04431278
       -0.03428394]
      [-0.14132024  0.00147425 -0.11869487 ...  0.18961735  0.02061029
       -0.2226579 ]
      ...
      [ 0.09847562 -0.04395953  0.05241163 ... -0.09560476  0.05311788
       -0.03395343]
      [ 0.04786448  0.05726088  0.03543131 ... -0.1665768   0.1144514
       -0.06121617]
      [ 0.05060462  0.04284142  0.08905675 ... -0.10912168 -0.01330861
       -0.12079769]]
     Target Tensor:
      [[ 0.          0.          0.         ...  0.1346057  -0.06730285
        0.        ]
      [ 0.          0.06730285 -0.1346057  ...  0.06730285 -0.06730285
       -0.06730285]
      [-0.1346057   0.         -0.1346057  ...  0.20190856  0.
       -0.20190856]
      ...
      [ 0.06730285 -0.06730285  0.06730285 ... -0.06730285  0.06730285
       -0.06730285]
      [ 0.06730285  0.06730285  0.06730285 ... -0.1346057   0.1346057
       -0.06730285]
      [ 0.06730285  0.06730285  0.06730285 ... -0.1346057   0.
       -0.1346057 ]]
     0.00037872815
```

## *Step 12.2:- Quantization apply on Weights for Layer [1]*

```
T1 = np.array (model.get_weights()[1])
print(T1)
```

```
     [[ 0.06777401  0.06759165 -0.11989291 ... -0.01104744 -0.04181176
       -0.12480928]
      [-0.11874168  0.01187701  0.00720231 ...  0.08293021 -0.0663819
       -0.03104337]
      [ 0.15734038 -0.1954211  -0.03614312 ... -0.04541576  0.01764638
       -0.07291853]
      ...
      [-0.09940577  0.12286972 -0.30298692 ...  0.08476383  0.01410181
       -0.05769118]
      [-0.03236272  0.04511095 -0.00150096 ...  0.03450888 -0.00148458
        0.07267053]
      [ 0.09258701  0.06765915  0.10189385 ...  0.07742485 -0.19675745
        0.0231267 ]]
```

```
# Min-Max value of float_32 tensor (x) find out for scale (s) and zero point (z)

b1 = np.amax(T1)
print(b1)
```

```
     0.5347112
```

```
a1 = np.amin(T1)
print(a1)
```

```
     -0.56059057
```

```
# scale value

scale1 = (b1-a1)/15

print(scale1)
```

```
     0.07302011648813883
```

```
# zero point

zero_point1 = np.round(-a1*15/(b1-a1))

print(zero_point1)
```

```
     8.0
```

```
f1 = np.round(T1/scale1 + zero_point1)
print(f1)
T1q = np.clip(f1, a_min=0, a_max=15) # Here min & max value we can change as per Tbit.
# But, I have checked for 4 bit
print (T1q)
```

```
     [[ 9.  9.  6. ...  8.  7.  6.]
      [ 6.  8.  8. ...  9.  7.  8.]
      [10.  5.  8. ...  7.  8.  7.]
      ...
      [ 7. 10.  4. ...  9.  8.  7.]
      [ 8.  9.  8. ...  8.  8.  9.]
      [ 9.  9.  9. ...  9.  5.  8.]]
     [[ 9.  9.  6. ...  8.  7.  6.]
      [ 6.  8.  8. ...  9.  7.  8.]
      [10.  5.  8. ...  7.  8.  7.]
      ...
      [ 7. 10.  4. ...  9.  8.  7.]
      [ 8.  9.  8. ...  8.  8.  9.]
      [ 9.  9.  9. ...  9.  5.  8.]]
```

```
T1dq = scale1*(T1q - zero_point1)
```

```
print(T1dq)
```

```
[[ 0.07302012  0.07302012 -0.14604023 ...  0.         -0.07302012
  -0.14604023]
 [-0.14604023  0.          0.         ...  0.07302012 -0.07302012
   0.        ]
 [ 0.14604023 -0.21906035  0.         ... -0.07302012  0.
  -0.07302012]
 ...
 [-0.07302012  0.14604023 -0.29208046 ...  0.07302012  0.
  -0.07302012]
 [ 0.          0.07302012  0.         ...  0.          0.
   0.07302012]
 [ 0.07302012  0.07302012  0.07302012 ...  0.07302012 -0.21906035
   0.        ]]
```

```
# print input and target tensors
print("Input Tensor:\n", T1)
print("Target Tensor:\n", T1dq)
mse = tf.keras.losses.MeanAbsoluteError()
mse(T1dq, T1).numpy()
```

```
Input Tensor:
 [[ 0.06777401  0.06759165 -0.11989291 ... -0.01104744 -0.04181176
  -0.12480928]
 [-0.11874168  0.01187701  0.00720231 ...  0.08293021 -0.0663819
  -0.03104337]
 [ 0.15734038 -0.1954211  -0.03614312 ... -0.04541576  0.01764638
  -0.07291853]
 ...
 [-0.09940577  0.12286972 -0.30298692 ...  0.08476383  0.01410181
  -0.05769118]
 [-0.03236272  0.04511095 -0.00150096 ...  0.03450888 -0.00148458
   0.07267053]
 [ 0.09258701  0.06765915  0.10189385 ...  0.07742485 -0.19675745
   0.0231267 ]]
Target Tensor:
 [[ 0.07302012  0.07302012 -0.14604023 ...  0.         -0.07302012
  -0.14604023]
 [-0.14604023  0.          0.         ...  0.07302012 -0.07302012
   0.        ]
 [ 0.14604023 -0.21906035  0.         ... -0.07302012  0.
  -0.07302012]
 ...
 [-0.07302012  0.14604023 -0.29208046 ...  0.07302012  0.
  -0.07302012]
 [ 0.          0.07302012  0.         ...  0.          0.
   0.07302012]
 [ 0.07302012  0.07302012  0.07302012 ...  0.07302012 -0.21906035
   0.        ]]
0.01831078
```

```
# print input and target tensors
print("Input Tensor:\n", T1)
print("Target Tensor:\n", T1dq)
mse = tf.keras.losses.MeanSquaredError()
mse(T1dq, T1).numpy()
```

```
Input Tensor:
 [[ 0.06777401  0.06759165 -0.11989291 ... -0.01104744 -0.04181176
  -0.12480928]
 [-0.11874168  0.01187701  0.00720231 ...  0.08293021 -0.0663819
  -0.03104337]
 [ 0.15734038 -0.1954211  -0.03614312 ... -0.04541576  0.01764638
  -0.07291853]
 ...
 [-0.09940577  0.12286972 -0.30298692 ...  0.08476383  0.01410181
  -0.05769118]
 [-0.03236272  0.04511095 -0.00150096 ...  0.03450888 -0.00148458
   0.07267053]
 [ 0.09258701  0.06765915  0.10189385 ...  0.07742485 -0.19675745
   0.0231267 ]]
Target Tensor:
 [[ 0.07302012  0.07302012 -0.14604023 ...  0.         -0.07302012
  -0.14604023]
 [-0.14604023  0.          0.         ...  0.07302012 -0.07302012
   0.        ]
 [ 0.14604023 -0.21906035  0.         ... -0.07302012  0.
  -0.07302012]
 ...
 [-0.07302012  0.14604023 -0.29208046 ...  0.07302012  0.
  -0.07302012]
 [ 0.          0.07302012  0.         ...  0.          0.
   0.07302012]
 [ 0.07302012  0.07302012  0.07302012 ...  0.07302012 -0.21906035
   0.        ]]
0.00044777602
```

## Step 12.3:- Quantization apply on Weights for Layer [2]

```
T2 = np.array (model.get_weights()[2])
print(T2)
```

```
  1.01613581e+00  9.09827352e-01  8.63439560e-01  1.01857758e+00
  9.57526326e-01  9.97431695e-01  8.66218805e-01  1.02473211e+00
  1.10267115e+00  1.00714552e+00  9.51363444e-01  9.29126859e-01
  9.79502380e-01  1.10982239e+00  1.15015268e+00  1.06933308e+00
  9.71925318e-01  1.01608205e+00  9.85301197e-01  1.11632621e+00
  1.08904934e+00  9.98482883e-01  1.00662923e+00  8.89398873e-01
  9.66598511e-01  1.00164521e+00  9.69219387e-01  9.81615841e-01
  9.78496075e-01  9.30068433e-01  1.08858550e+00  1.04268372e+00
  1.03943658e+00  9.98060167e-01  1.03112936e+00  9.04340446e-01
  1.01481199e+00  1.01915789e+00  9.85788584e-01  1.07223654e+00
  9.64191139e-01  9.75218356e-01  9.54456747e-01  9.18242931e-01
  9.21438098e-01  1.02076674e+00  1.02725565e+00  9.94487286e-01
  9.68944848e-01  1.01321733e+00  1.07815278e+00  9.77605164e-01
  9.06899571e-01  1.11797512e+00  1.01943362e+00  1.00336015e+00
  1.04066145e+00  9.65861440e-01  9.83944893e-01  1.14388287e+00
  1.03661418e+00  9.12949681e-01  1.04882288e+00  9.84841287e-01
  1.03915215e+00  1.12184322e+00  9.94718552e-01  1.02396071e+00
  9.55791354e-01  1.07630527e+00  8.85838568e-01  1.01847661e+00
 -6.87905448e-03  2.26764567e-02  1.61300693e-02 -4.78645694e-03
 -3.10650449e-02 -3.40941325e-02  1.86896343e-02  8.92082788e-03
 -1.12632727e-02 -4.27742349e-03 -1.09403403e-02 -1.16341393e-02
 -1.27771422e-02  5.80965132e-02  3.48129161e-02 -5.40827110e-04
  9.86322854e-03  1.41829094e-02  1.41390378e-03 -1.77349355e-02
 -4.02648561e-02 -2.20553763e-02 -1.13608371e-02  1.75506119e-02
 -2.14500669e-02 -3.69772837e-02  2.29502581e-02 -1.24770729e-02
 -5.55435419e-02  1.58325657e-02  1.15060564e-02 -3.48312296e-02
 -1.16121890e-02 -5.87414484e-03 -4.99279574e-02  1.39855286e-02
  5.39994389e-02 -1.30007407e-02 -3.35767567e-02 -4.65596244e-02
  5.79028251e-03  4.01817597e-02 -1.38439750e-02 -2.52826065e-02
  2.35471502e-03 -1.32176848e-02  1.31066060e-02 -1.73702072e-02
  2.99341208e-03  3.02875526e-02  6.13649329e-03 -3.80107835e-02
  8.92498344e-02  3.36106718e-02  1.37243960e-02  2.57514641e-02
 -1.37995910e-02  2.07069535e-02 -1.94300748e-02  7.80594209e-03
  1.69660319e-02 -2.07808986e-02  1.87570509e-02  4.19962918e-03
  1.31348753e-02  2.73659378e-02 -6.34227600e-03  2.08946858e-02
  2.76989844e-02  1.24754542e-02  1.24331750e-03  3.50610279e-02
  8.38968065e-03  1.73458811e-02  3.79903913e-02  3.86711061e-02
 -3.02877147e-02 -6.84603080e-02  6.03204546e-03  5.75914122e-02
  7.32691810e-02 -1.00181192e-01 -8.78253428e-04 -3.94931762e-03
  1.24641173e-02  3.15759927e-02 -1.63893700e-02  5.58563136e-02
  8.36112946e-02 -9.35319439e-02 -8.45441967e-02 -4.42930311e-02
 -7.39627182e-02 -3.11709289e-03  3.02279051e-02 -1.27332797e-03
  5.26266769e-02 -8.00020546e-02 -1.78254209e-02 -8.73361826e-02
 -3.86155471e-02  2.00129122e-01  4.14125435e-02 -3.53166834e-02
  9.49775502e-02 -4.39387601e-04 -3.78712825e-02  5.03545478e-02
  7.97494203e-02 -1.86536964e-02  7.75164180e-03 -3.42939422e-02
 -8.76416042e-02 -1.15766712e-02 -3.89277488e-02  1.36305951e-02
 -3.55004035e-02 -2.46527586e-02  5.07961847e-02  4.12495323e-02
  4.08685356e-02 -2.98484452e-02  7.19511136e-02  6.64878637e-03
  3.72174121e-02  1.29167000e-02 -4.48333509e-02  5.37069403e-02
 -3.97735499e-02 -5.36061302e-02 -4.31677401e-02  5.16716903e-03
  6.93309233e-02  2.17184033e-02  2.80777644e-02 -6.68777898e-03
  4.29893956e-02 -6.58701034e-03  2.30927672e-02 -2.21706461e-02
 -3.88433821e-02 -4.11779322e-02 -2.08206475e-02  1.99485980e-02
 -2.79723760e-02 -7.11276233e-02 -6.48017451e-02  4.48469780e-02
  4.72299494e-02 -6.91493228e-02  5.33572212e-02  1.97497383e-02
 -8.35282058e-02  8.88175070e-02 -6.57924414e-02  1.03326418e-01
 -4.15926687e-02  1.17592532e-02 -1.10870793e-01  4.69601229e-02]
```

```
# Min-Max value of float_32 tensor (x) find out for scale (s) and zero point (z)

b2 = np.amax(T2)
print(b2)
```

```
1.1501527
```

```
a2 = np.amin(T2)
print(a2)
```

```
-0.22343871
```

```
# scale value

scale2 = (b2-a2)/15

print(scale2)
```

```
0.09157276153564453
```

```
# zero point

zero_point2 = np.round(-a2*15/(b2-a2))

print(zero_point2)
```

```
2.0
```

```
from pandas.compat.numpy import np_array_datetime64_compat
f2 = np.round(T2/scale2 + zero_point2)
print(f2)
T2q = np.clip(f2, a_min=0, a_max=15) # Here min & max value we can change as per Tbit.
# But, I have checked for 4 bit
print (T2q)
```

```
[ 2.  0.  1.  1.  2.  2.  2.  1.  2.  1. -0.  1.  1. -0.  2.  2.  2.  1.
  1.  1.  1.  4.  3.  2.  2.  2.  1.  2.  2.  2.  2.  1.  1.  2.  1.  3.
  0.  2.  3.  2.  2.  2.  3.  1.  3.  3.  1.  3.  1.  1.  2.  1.  2.  2.
  0.  2.  3.  1.  1.  1.  2.  3.  0.  3.  2.  1.  2.  3.  2.  1.  3.  2.
  2.  4.  2.  3.  2.  1.  2.  2. 12. 12. 13. 13. 14. 14. 13. 12. 13. 12.
 11. 13. 12. 13. 11. 13. 14. 13. 12. 12. 13. 14. 15. 14. 13. 13. 13. 14.
 14. 13. 13. 12. 13. 13. 13. 13. 13. 12. 14. 13. 13. 13. 13. 12. 13. 13.
 13. 14. 13. 13. 12. 12. 12. 13. 13. 13. 13. 13. 14. 13. 12. 14. 13. 13.
 13. 13. 13. 14. 13. 12. 13. 13. 13. 14. 13. 13. 12. 14. 12. 13.  2.  2.
  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  3.  2.  2.  2.  2.  2.  2.
  2.  2.  2.  2.  2.  2.  2.  2.  1.  2.  2.  2.  2.  2.  1.  2.  3.  2.
  2.  1.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  3.  2.  2.  2.
  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.
  2.  2.  2.  1.  2.  3.  3.  1.  2.  2.  2.  2.  2.  3.  3.  1.  1.  2.
  1.  2.  2.  2.  3.  1.  2.  1.  2.  4.  2.  2.  3.  2.  2.  3.  3.  2.
  2.  2.  1.  2.  2.  2.  2.  2.  3.  2.  2.  2.  3.  2.  2.  2.  2.  3.
  2.  1.  2.  2.  3.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  1.
  1.  2.  3.  1.  3.  2.  1.  3.  1.  3.  2.  2.  1.  3.]
[ 2.  0.  1.  1.  2.  2.  2.  1.  2.  1.  0.  1.  1.  0.  2.  2.  2.  1.
  1.  1.  1.  4.  3.  2.  2.  2.  1.  2.  2.  2.  2.  1.  1.  2.  1.  3.
  0.  2.  3.  2.  2.  2.  3.  1.  3.  3.  1.  3.  1.  1.  2.  1.  2.  2.
  0.  2.  3.  1.  1.  1.  2.  3.  0.  3.  2.  1.  2.  3.  2.  1.  3.  2.
  2.  4.  2.  3.  2.  1.  2.  2. 12. 12. 13. 13. 14. 14. 13. 12. 13. 12.
 11. 13. 12. 13. 11. 13. 14. 13. 12. 12. 13. 14. 15. 14. 13. 13. 13. 14.
 14. 13. 13. 12. 13. 13. 13. 13. 13. 12. 14. 13. 13. 13. 13. 12. 13. 13.
 13. 14. 13. 13. 12. 12. 12. 13. 13. 13. 13. 13. 14. 13. 12. 14. 13. 13.
 13. 13. 13. 14. 13. 12. 13. 13. 13. 14. 13. 13. 12. 14. 12. 13.  2.  2.
  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  3.  2.  2.  2.  2.  2.  2.
  2.  2.  2.  2.  2.  2.  2.  2.  1.  2.  2.  2.  2.  2.  1.  2.  3.  2.
  2.  1.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  3.  2.  2.  2.
  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.
  2.  2.  2.  1.  2.  3.  3.  1.  2.  2.  2.  2.  2.  3.  3.  1.  1.  2.
  1.  2.  2.  2.  3.  1.  2.  1.  2.  4.  2.  2.  3.  2.  2.  3.  3.  2.
  2.  2.  1.  2.  2.  2.  2.  2.  3.  2.  2.  2.  3.  2.  2.  2.  2.  3.
  2.  1.  2.  2.  3.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  1.
  1.  2.  3.  1.  3.  2.  1.  3.  1.  3.  2.  2.  1.  3.]
```

```
T2dq = scale2*(T2q - zero_point2)
print(T2dq)
```

```
[ 0.         -0.18314552 -0.09157276 -0.09157276  0.          0.
  0.         -0.09157276  0.         -0.09157276 -0.18314552 -0.09157276
 -0.09157276 -0.18314552  0.          0.          0.         -0.09157276
 -0.09157276 -0.09157276 -0.09157276  0.18314552  0.09157276  0.
  0.          0.         -0.09157276  0.          0.          0.
  0.         -0.09157276 -0.09157276  0.         -0.09157276  0.09157276
 -0.18314552  0.          0.09157276  0.          0.          0.
  0.09157276 -0.09157276  0.09157276  0.09157276 -0.09157276  0.09157276
 -0.09157276 -0.09157276  0.         -0.09157276  0.          0.
 -0.18314552  0.          0.09157276 -0.09157276 -0.09157276 -0.09157276
  0.          0.09157276 -0.18314552  0.09157276  0.         -0.09157276
  0.          0.09157276  0.         -0.09157276  0.09157276  0.
  0.          0.18314552  0.          0.09157276  0.         -0.09157276
  0.          0.          0.9157276   0.9157276   1.0073004   1.0073004
  1.0988731   1.0988731   1.0073004   0.9157276   1.0073004   0.9157276
  0.82415485  1.0073004   0.9157276   1.0073004   0.82415485  1.0073004
  1.0988731   1.0073004   0.9157276   0.9157276   1.0073004   1.0988731
  1.1904459   1.0988731   1.0073004   1.0073004   1.0073004   1.0988731
  1.0988731   1.0073004   1.0073004   0.9157276   1.0073004   1.0073004
  1.0073004   1.0073004   1.0073004   0.9157276   1.0988731   1.0073004
  1.0073004   1.0073004   1.0073004   0.9157276   1.0073004   1.0073004
  1.0073004   1.0988731   1.0073004   1.0073004   0.9157276   0.9157276
  0.9157276   1.0073004   1.0073004   1.0073004   1.0073004   1.0073004
  1.0988731   1.0073004   0.9157276   1.0988731   1.0073004   1.0073004
  1.0073004   1.0073004   1.0073004   1.0988731   1.0073004   0.9157276
  1.0073004   1.0073004   1.0073004   1.0988731   1.0073004   1.0073004
  0.9157276   1.0988731   0.9157276   1.0073004   0.          0.
  0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.          0.          0.09157276
  0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.          0.          0.
  0.          0.         -0.09157276  0.          0.          0.
  0.          0.         -0.09157276  0.          0.09157276  0.
  0.         -0.09157276  0.          0.          0.          0.
  0.          0.          0.          0.          0.          0.
  0.          0.          0.09157276  0.          0.          0.
  0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.          0.          0.
  0.          0.          0.         -0.09157276  0.          0.09157276
  0.09157276 -0.09157276  0.          0.          0.          0.
  0.          0.09157276  0.09157276 -0.09157276 -0.09157276  0.
 -0.09157276  0.          0.          0.          0.09157276 -0.09157276
  0.         -0.09157276  0.          0.18314552  0.          0.
  0.09157276  0.          0.          0.09157276  0.09157276  0.
  0.          0.         -0.09157276  0.          0.          0.
  0.          0.          0.09157276  0.          0.          0.
  0.09157276  0.          0.          0.          0.          0.09157276
  0.         -0.09157276  0.          0.          0.09157276  0.
  0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.          0.         -0.09157276
 -0.09157276  0.          0.09157276 -0.09157276  0.09157276  0.
 -0.09157276  0.09157276 -0.09157276  0.09157276  0.          0.
 -0.09157276  0.09157276]
```

```
# print input and target tensors
print("Input Tensor:\n", T2)
print("Target Tensor:\n", T2dq)
```

```
mse = tf.keras.losses.MeanAbsoluteError()
mse(T2dq, T2).numpy()
```

```
      -8.35282058e-02  8.88175070e-02 -6.57924414e-02  1.03326418e-01
      -4.15926687e-02  1.17592532e-02 -1.10870793e-01  4.69601229e-02]
     Target Tensor:
      [ 0.         -0.18314552 -0.09157276 -0.09157276  0.          0.
       0.         -0.09157276  0.         -0.09157276 -0.18314552 -0.09157276
      -0.09157276 -0.18314552  0.          0.          0.         -0.09157276
      -0.09157276 -0.09157276 -0.09157276  0.18314552  0.09157276  0.
       0.          0.         -0.09157276  0.          0.          0.
       0.         -0.09157276 -0.09157276  0.         -0.09157276  0.09157276
      -0.18314552  0.          0.09157276  0.          0.          0.
       0.09157276 -0.09157276  0.09157276  0.09157276 -0.09157276  0.09157276
      -0.09157276 -0.09157276  0.         -0.09157276  0.          0.
      -0.18314552  0.          0.09157276 -0.09157276 -0.09157276 -0.09157276
       0.          0.09157276 -0.18314552  0.09157276  0.         -0.09157276
       0.          0.09157276  0.         -0.09157276  0.09157276  0.
       0.          0.18314552  0.          0.09157276  0.         -0.09157276
       0.          0.          0.9157276   0.9157276   1.0073004   1.0073004
       1.0988731   1.0988731   1.0073004   0.9157276   1.0073004   0.9157276
       0.82415485  1.0073004   0.9157276   1.0073004   0.82415485  1.0073004
       1.0988731   1.0073004   0.9157276   0.9157276   1.0073004   1.0988731
       1.1904459   1.0988731   1.0073004   1.0073004   1.0073004   1.0988731
       1.0988731   1.0073004   1.0073004   0.9157276   1.0073004   1.0073004
       1.0073004   1.0073004   1.0073004   0.9157276   1.0988731   1.0073004
       1.0073004   1.0073004   1.0073004   0.9157276   1.0073004   1.0073004
       1.0073004   1.0988731   1.0073004   1.0073004   0.9157276   0.9157276
       0.9157276   1.0073004   1.0073004   1.0073004   1.0073004   1.0073004
       1.0988731   1.0073004   0.9157276   1.0988731   1.0073004   1.0073004
       1.0073004   1.0073004   1.0073004   1.0988731   1.0073004   0.9157276
       1.0073004   1.0073004   1.0073004   1.0988731   1.0073004   1.0073004
       0.9157276   1.0988731   0.9157276   1.0073004   0.          0.
       0.          0.          0.          0.          0.          0.
       0.          0.          0.          0.          0.          0.09157276
       0.          0.          0.          0.          0.          0.
       0.          0.          0.          0.          0.          0.
       0.          0.         -0.09157276  0.          0.          0.
       0.          0.         -0.09157276  0.          0.09157276  0.
       0.         -0.09157276  0.          0.          0.          0.
       0.          0.          0.          0.          0.          0.
       0.          0.          0.09157276  0.          0.          0.
       0.          0.          0.          0.          0.          0.
       0.          0.          0.          0.          0.          0.
       0.          0.          0.          0.          0.          0.
       0.          0.          0.         -0.09157276  0.          0.09157276
       0.09157276 -0.09157276  0.          0.          0.          0.
       0.          0.09157276  0.09157276 -0.09157276 -0.09157276  0.
      -0.09157276  0.          0.          0.          0.09157276 -0.09157276
       0.         -0.09157276  0.          0.18314552  0.          0.
       0.09157276  0.          0.          0.09157276  0.09157276  0.
       0.          0.         -0.09157276  0.          0.          0.
       0.          0.          0.09157276  0.          0.          0.
       0.09157276  0.          0.          0.          0.          0.09157276
       0.         -0.09157276  0.          0.          0.09157276  0.
       0.          0.          0.          0.          0.          0.
       0.          0.          0.          0.          0.         -0.09157276
      -0.09157276  0.          0.09157276 -0.09157276  0.09157276  0.
      -0.09157276  0.09157276 -0.09157276  0.09157276  0.          0.
      -0.09157276  0.09157276]
     0.031512724
```

```
0.021512734
```

```
# print input and target tensors
print("Input Tensor:\n", T2)
print("Target Tensor:\n", T2dq)
mse = tf.keras.losses.MeanSquaredError()
mse(T2dq, T2).numpy()
```

```
 -8.35282058e-02  8.88175070e-02 -6.57924414e-02  1.03326418e-01
 -4.15926687e-02  1.17592532e-02 -1.10870793e-01  4.69601229e-02]
Target Tensor:
 [ 0.         -0.18314552 -0.09157276 -0.09157276  0.          0.
  0.         -0.09157276  0.         -0.09157276 -0.18314552 -0.09157276
 -0.09157276 -0.18314552  0.          0.          0.         -0.09157276
 -0.09157276 -0.09157276 -0.09157276  0.18314552  0.09157276  0.
  0.          0.         -0.09157276  0.          0.          0.
  0.         -0.09157276 -0.09157276  0.         -0.09157276  0.09157276
 -0.18314552  0.          0.09157276  0.          0.          0.
  0.09157276 -0.09157276  0.09157276  0.09157276 -0.09157276  0.09157276
 -0.09157276 -0.09157276  0.         -0.09157276  0.          0.
 -0.18314552  0.          0.09157276 -0.09157276 -0.09157276 -0.09157276
  0.          0.09157276 -0.18314552  0.09157276  0.         -0.09157276
  0.          0.09157276  0.         -0.09157276  0.09157276  0.
  0.          0.18314552  0.          0.09157276  0.         -0.09157276
  0.          0.          0.9157276   0.9157276   1.0073004   1.0073004
  1.0988731   1.0988731   1.0073004   0.9157276   1.0073004   0.9157276
  0.82415485  1.0073004   0.9157276   1.0073004   0.82415485  1.0073004
  1.0988731   1.0073004   0.9157276   0.9157276   1.0073004   1.0988731
  1.1904459   1.0988731   1.0073004   1.0073004   1.0073004   1.0988731
  1.0988731   1.0073004   1.0073004   0.9157276   1.0073004   1.0073004
  1.0073004   1.0073004   1.0073004   0.9157276   1.0988731   1.0073004
  1.0073004   1.0073004   1.0073004   0.9157276   1.0073004   1.0073004
  1.0073004   1.0988731   1.0073004   1.0073004   0.9157276   0.9157276
  0.9157276   1.0073004   1.0073004   1.0073004   1.0073004   1.0073004
  1.0988731   1.0073004   0.9157276   1.0988731   1.0073004   1.0073004
  1.0073004   1.0073004   1.0073004   1.0988731   1.0073004   0.9157276
  1.0073004   1.0073004   1.0073004   1.0988731   1.0073004   1.0073004
  0.9157276   1.0988731   0.9157276   1.0073004   0.          0.
  0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.          0.          0.09157276
  0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.          0.          0.
  0.          0.         -0.09157276  0.          0.          0.
  0.          0.         -0.09157276  0.          0.09157276  0.
  0.         -0.09157276  0.          0.          0.          0.
  0.          0.          0.          0.          0.          0.
  0.          0.          0.09157276  0.          0.          0.
  0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.          0.          0.
  0.          0.          0.         -0.09157276  0.          0.09157276
  0.09157276 -0.09157276  0.          0.          0.          0.
  0.          0.09157276  0.09157276 -0.09157276 -0.09157276  0.
 -0.09157276  0.          0.          0.          0.09157276 -0.09157276
  0.         -0.09157276  0.          0.18314552  0.          0.
  0.09157276  0.          0.          0.09157276  0.09157276  0.
  0.          0.         -0.09157276  0.          0.          0.
  0.          0.          0.09157276  0.          0.          0.
  0.09157276  0.          0.          0.          0.          0.09157276
  0.         -0.09157276  0.          0.          0.09157276  0.
  0.          0.09157276  0.          0.          0.09157276  0.
  0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.          0.         -0.09157276
 -0.09157276  0.          0.09157276 -0.09157276  0.09157276  0.
 -0.09157276  0.09157276 -0.09157276  0.09157276  0.          0.
 -0.09157276  0.09157276]
0.00063613185
```

### Step 12.4:- Quantization apply on Weights for Layer [3]

```
T3 = np.array (model.get_weights()[3])
print(T3)
```

```
[[ 0.06568483  0.23173128 -0.23441082]
 [-0.15249006 -0.05274262  0.12696849]
 [-0.09750769 -0.03921815  0.10007721]
 ...
 [-0.17221509  0.06782542 -0.03295461]
 [-0.16315362 -0.00453885  0.15091746]
 [-0.11915002 -0.11885497  0.16523157]]
```

```
# Min-Max value of float_32 tensor (x) find out for scale (s) and zero point (z)

b3 = np.amax(T3)
print(b3)
```

```
0.33345005
```

```
a3 = np.amin(T3)
print(a3)
```

```
-0.3088779
```

```
# scale value

scale3 = (b3-a3)/15

print(scale3)
```

```
0.042821860313415526
```

```
# zero point

zero_point3= np.round(-a3*15/(b3-a3))

print(zero_point3)
```

```
7.0
```

```
f3 = np.round(T3/scale3 + zero_point3)
```

```
print(f3)
T3q = np.clip(f3, a_min=0, a_max=15) # Here min & max value we can change as per Tbit.
# But, I have checked for 4 bit
print (T3q)

     [[ 9. 12.  2.]
      [ 3.  6. 10.]
      [ 5.  6.  9.]
      ...
      [ 3.  9.  6.]
      [ 3.  7. 11.]
      [ 4.  4. 11.]]
     [[ 9. 12.  2.]
      [ 3.  6. 10.]
      [ 5.  6.  9.]
      ...
      [ 3.  9.  6.]
      [ 3.  7. 11.]
      [ 4.  4. 11.]]

T3dq = scale3*(T3q - zero_point3)
print(T3dq)

     [[ 0.08564372  0.2141093  -0.2141093 ]
      [-0.17128745 -0.04282186  0.1284656 ]
      [-0.08564372 -0.04282186  0.08564372]
      ...
      [-0.17128745  0.08564372 -0.04282186]
      [-0.17128745  0.          0.17128745]
      [-0.1284656  -0.1284656   0.17128745]]


# print input and target tensors
print("Input Tensor:\n", T3)
print("Target Tensor:\n", T3dq)
mse = tf.keras.losses.MeanAbsoluteError()
mse(T3dq, T3).numpy()

     Input Tensor:
      [[ 0.06568483  0.23173128 -0.23441082]
      [-0.15249006 -0.05274262  0.12696849]
      [-0.09750769 -0.03921815  0.10007721]
      ...
      [-0.17221509  0.06782542 -0.03295461]
      [-0.16315362 -0.00453885  0.15091746]
      [-0.11915002 -0.11885497  0.16523157]]
     Target Tensor:
      [[ 0.08564372  0.2141093  -0.2141093 ]
      [-0.17128745 -0.04282186  0.1284656 ]
      [-0.08564372 -0.04282186  0.08564372]
      ...
      [-0.17128745  0.08564372 -0.04282186]
      [-0.17128745  0.          0.17128745]
      [-0.1284656  -0.1284656   0.17128745]]
     0.010624968


# print input and target tensors
```

```
print("Input Tensor:\n", T3)
print("Target Tensor:\n", T3dq)
mse = tf.keras.losses.MeanSquaredError()
mse(T3dq, T3).numpy()
```

```
Input Tensor:
 [[ 0.06568483  0.23173128 -0.23441082]
 [-0.15249006 -0.05274262  0.12696849]
 [-0.09750769 -0.03921815  0.10007721]
 ...
 [-0.17221509  0.06782542 -0.03295461]
 [-0.16315362 -0.00453885  0.15091746]
 [-0.11915002 -0.11885497  0.16523157]]
Target Tensor:
 [[ 0.08564372  0.2141093  -0.2141093 ]
 [-0.17128745 -0.04282186  0.1284656 ]
 [-0.08564372 -0.04282186  0.08564372]
 ...
 [-0.17128745  0.08564372 -0.04282186]
 [-0.17128745  0.          0.17128745]
 [-0.1284656  -0.1284656   0.17128745]]
0.00015097122
```

✓ 0s completed at 7:08 PM ● ×