Classification of Audio Digit Recognition

About the Dataset:

A Large dataset of Audio MNIST, 30000 audio samples of spoken digits (0-9) of 60 different speakers.

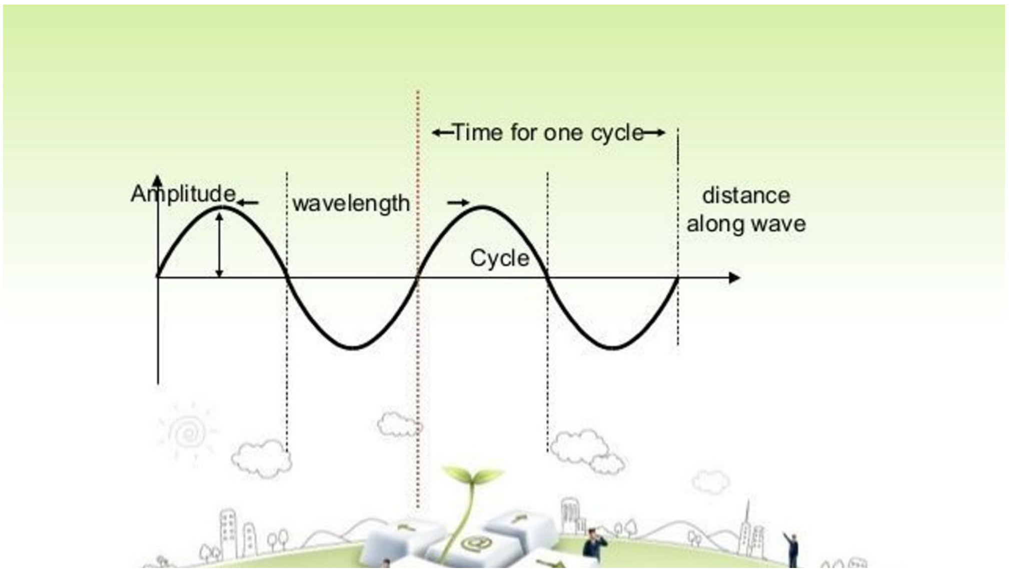
- The dataset consists of 30000 audio samples of spoken digits (0-9) of 60 folders and 500 files each.
- There is one directory per speaker holding the audio recordings.

Link to download the dataset : https://www.kaggle.com/datasets/sripaadsrinivasan/audio-mnist

---

Sound Waves

Sound is a form of energy which makes us hear, it travels in the form of wave. Sound wave can be described by five characteristics.



1. **Wavelength** : The minimum distance in which a sound wave repeats itself is called its wavelength.
2. **Amplitude** : When a wave passes through a medium, the particles of the medium get displaced temporarily from their original undisturbed positions. The maximum displacement of the particles of the medium from their original undisturbed positions, when a wave passes through the medium is called amplitude of the wave.
3. **Time-period** : The time required to produce one complete wave or cycle or cycle is called time-period of the wave.
4. **Frequency** : The number of complete waves or cycles produced in one second is called frequency of the wave.
5. **Velocity of wave** : The distance travelled by a wave in one second is called velocity of the wave or speed of the wave.

A wave is a vibratory disturbance in a medium which carries energy from one point to another without there being a direct contact between the two points.We can say that a wave is produced by the vibrations of the particles of the medium through which it passes.

There are two types of waves: Longitudinal waves and Transverse waves.

- **Longitudinal Waves**: A wave in which the particles of the medium vibrate back and forth in the 'same direction' in which the wave is moving. Medium can be solid, liquid or gases. Therefore, sound waves are longitudinal waves.

- **Transverse Waves**: A wave in which the particles of the medium vibrate up and down 'at right angles' to the direction in which the wave is moving. These waves are produced only in a solids and liquids but not in gases.

Sound is a longitudinal wave which consists of compressions and rarefactions travelling through a medium.

Basics of Digital Audio

- Digital audio is music, speech, and other sounds represented in binary format for use in digital devices.
- Most digital devices have a built-in microphone and audio software, so recording external sounds is easy.
- To digitally record sound, samples of a sound wave are collected at periodic intervals and stored as numeric data in an audio file.
- Sound waves are sampled many times per second by an *analog-to-digital* converter.
- A *digital-to-analog* converter transforms the digital bits into analog sound waves.

What is Waveform?

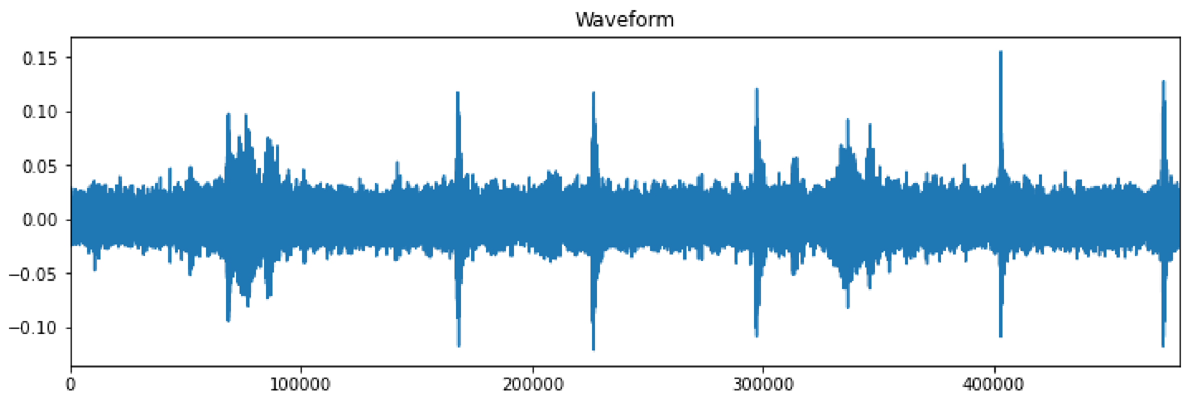
Waveform Audio File Format (*WAVE* or *WAV* due to its filename extension; pronounced "wave") is an audio file format standard, developed by *IBM* and *Microsoft*, for storing an audio bitstream on PCs. It is the main format used on Microsoft Windows systems for uncompressed audio. The usual bitstream encoding is the **linear pulse-code modulation (LPCM)** format.

The WAV file is an instance of a **Resource Interchange File Format (RIFF)** bitstream format method for storing data in chunks, and thus is similar to the 8SVX and the AIFF format used on Amiga and Macintosh computers. The RIFF format acts as a **"wrapper"** for various audio coding formats.

Though a WAV file can contain compressed audio, the most common WAV audio format is uncompressed audio in the linear pulse-code modulation (LPCM) format. LPCM is also the standard audio coding format for audio CDs, which store two-channel LPCM audio sampled at 44,100 Hz with 16 bits per sample. Since LPCM is uncompressed and retains all of the samples of an audio track, professional users or audio experts may use the WAV format with LPCM audio for maximum audio quality.

- Filename extension : `.wav` or `.wave`
- Internet media type : audio/vnd.wave, audio/wav, audio/wave, audio/x-wav
- Type code : WAVE
- Initial release : August 1991
- Latest release : Multiple Channel Audio Data and WAVE Files, 7 March 2007
- Uniform Type Identifier : com.microsoft.waveform-audio
- Extended from : RIFF
- Extended to : BWF, RF64

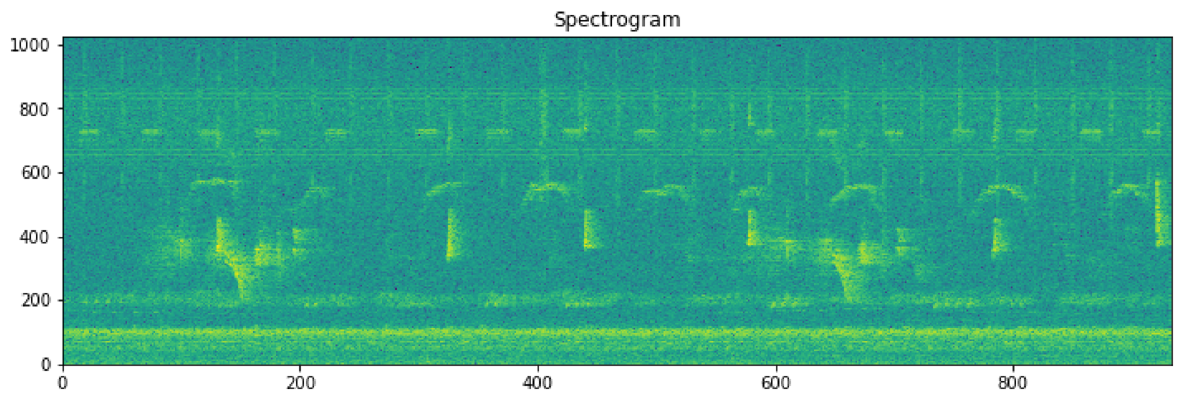
A Waveform is a time series with the signal amplitude at each specific time, if we visualize one of those waveform samples we will get something like this:



Intuitively one might consider modeling this data like a regular time series (e.g. stock price forecasting) using some kind of *RNN* model, in fact, this could be done, but since we are using audio signals, a more appropriate choice is to transform the waveform samples into spectrograms.

Spectrogram:

A spectrogram is an image representation of the waveform signal, it shows its frequency intensity range over time, it can be very useful when we want to evaluate the signal's frequency distribution over time. Below is the spectrogram representation of the waveform image we saw above.



What to do next?

- Convert Audio data into Waveform
- Transform Waveform into Spectrogram
- Classify the digits

Import the necessary dependencies

In [1]:
```python
import os

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import IPython.display as ipd

import tensorflow as tf
import tensorflow_io as tfio
from tensorflow import keras

import logging
logger = tf.get_logger()
logger.setLevel(logging.ERROR)
```

In [7]:
```python
# Main Directory -- Audio data path
data_dir = '../DL/Datasets/AudioMNIST/audioData/'

# Test data directory
test_data_dir = '../DL/Datasets/AudioMNIST/TestAudioData/'
```

In [8]:
```python
def abt(*args):
    for i in args:
        print(f"Total subfolders in the Main directory {i}: {len(os.listdir(i))}")

abt(data_dir)
```

```
Total subfolders in the Main directory ../DL/Datasets/AudioMNIST/audioData/: 10
```

In [9]:
```python
from keras.utils import audio_dataset_from_directory
```

In [10]:

```
train_ds, val_ds = keras.utils.audio_dataset_from_directory(
    directory=data_dir,
    batch_size=6,
    validation_split=0.2,
    labels='inferred',
    label_mode='categorical',
    # sampling_rate = 48000,
    seed=0,
    shuffle=True,
    output_sequence_length=30000,
    subset='both')

label_names = np.array(train_ds.class_names)
print()
print("label names:", label_names)
```

```
Found 7461 files belonging to 10 classes.
Using 5969 files for training.
Using 1492 files for validation.

label names: ['0' '1' '2' '3' '4' '5' '6' '7' '8' '9']
```

Here, we have 7,461 audio files (.wav) belonging to 10 classes and defined the validation split into 0.2, so it takes

- 5,969 files for training the model &
- 1,492 files for validating the model

We specified the **label_mode** into categorical, so that we have 10 classes (i.e., 0-9 digits)

The **output_sequence_length** is set to 30,000 why because these wave audio files is of the duration 1 second approximately and also each every waveform (most of them) are fits into the range of 28,000 - 30,000. If we plot the waveform there will be straight line after some particular **output_sequence_length** and also if we transform that waveform into spectrogram we can able to see some blank space, it shows that in the audio files there are some small silence part in the beginning & end of the audio. It may affect the model's learning ability to learn properly

- If we trim the silence approximately, we can get some detailed audio and plot into waveform & transform into spectrogram and finally feed into the model. Here, I didn't trimmed the audio and transformed the waveforms into spectrograms and directly feed into the model

In [11]:
```
train_ds.element_spec
```

Out[11]:
```
(TensorSpec(shape=(None, 30000, None), dtype=tf.float32, name=None),
 TensorSpec(shape=(None, 10), dtype=tf.float32, name=None))
```

In [12]:
```
val_ds.element_spec
```

Out[12]:
```
(TensorSpec(shape=(None, 30000, None), dtype=tf.float32, name=None),
 TensorSpec(shape=(None, 10), dtype=tf.float32, name=None))
```

The above `train_ds` & `val_ds` is of the datatype **Tensor** (*tf.Tensor*) is of format (batch_size, output_sequence_length, channel) for the audio & for the labels is of format (batch_size, num_classes)

In [13]:
```
train_ds
```

Out[13]:
```
<BatchDataset element_spec=(TensorSpec(shape=(None, 30000, None), dtype=tf.float32,
name=None), TensorSpec(shape=(None, 10), dtype=tf.float32, name=None))>
```

In [14]:
```
val_ds
```

Out[14]:
```
<BatchDataset element_spec=(TensorSpec(shape=(None, 30000, None), dtype=tf.float32,
name=None), TensorSpec(shape=(None, 10), dtype=tf.float32, name=None))>
```

This dataset only contains single channel audio, so we use the `tf.squeeze` function to drop the extra axis:

In [15]:
```
def squeeze(audio, labels):
    audio = tf.squeeze(audio, axis=-1)
    return audio, labels

train_ds = train_ds.map(squeeze, tf.data.AUTOTUNE)
val_ds = val_ds.map(squeeze, tf.data.AUTOTUNE)
```

In [16]:
```
train_ds
```

Out[16]:
```
<ParallelMapDataset element_spec=(TensorSpec(shape=(None, 30000), dtype=tf.float32,
name=None), TensorSpec(shape=(None, 10), dtype=tf.float32, name=None))>
```

In [17]:
```
val_ds
```

Out[17]:
```
<ParallelMapDataset element_spec=(TensorSpec(shape=(None, 30000), dtype=tf.float32,
name=None), TensorSpec(shape=(None, 10), dtype=tf.float32, name=None))>
```

In [18]:
```
train_ds.element_spec
```

Out[18]:
```
(TensorSpec(shape=(None, 30000), dtype=tf.float32, name=None),
 TensorSpec(shape=(None, 10), dtype=tf.float32, name=None))
```

In [19]:
```
val_ds.element_spec
```

Out[19]:
```
(TensorSpec(shape=(None, 30000), dtype=tf.float32, name=None),
 TensorSpec(shape=(None, 10), dtype=tf.float32, name=None))
```

`Dataset.shard` to split the validation set into two halves.

In [20]:
```
test_ds = val_ds.shard(num_shards=2, index=0)
val_ds = val_ds.shard(num_shards=2, index=1)
```

In [21]:
```
test_ds
```

Out[21]:
```
<ShardDataset element_spec=(TensorSpec(shape=(None, 30000), dtype=tf.float32, name=N
one), TensorSpec(shape=(None, 10), dtype=tf.float32, name=None))>
```

In [22]:
```
val_ds
```

Out[22]:
```
<ShardDataset element_spec=(TensorSpec(shape=(None, 30000), dtype=tf.float32, name=N
one), TensorSpec(shape=(None, 10), dtype=tf.float32, name=None))>
```

In [23]:
```
test_ds.element_spec
```

Out[23]:
```
(TensorSpec(shape=(None, 30000), dtype=tf.float32, name=None),
 TensorSpec(shape=(None, 10), dtype=tf.float32, name=None))
```

In [24]:
```
val_ds.element_spec
```

Out[24]:
```
(TensorSpec(shape=(None, 30000), dtype=tf.float32, name=None),
 TensorSpec(shape=(None, 10), dtype=tf.float32, name=None))
```

In [25]:
```
for example_audio, example_labels in train_ds.take(1):
    print(f"Shape of the Audio (batch_size,output_sequence_length) : {example_audio.
    print(f"Shape of the Labels (batch_size,num_labels)            : {example_labels
```

```
Shape of the Audio (batch_size,output_sequence_length) : (6, 30000)
Shape of the Labels (batch_size,num_labels)            : (6, 10)
```

In [26]:
```
np.argmax(example_labels[0])
```

Out[26]:
```
0
```

In [27]:
```
example_labels[0]
```

Out[27]:
```
<tf.Tensor: shape=(10,), dtype=float32, numpy=array([1., 0., 0., 0., 0., 0., 0., 0.,
0., 0.], dtype=float32)>
```

In [28]:
```
example_labels[1]
```

Out[28]:
```
<tf.Tensor: shape=(10,), dtype=float32, numpy=array([0., 0., 0., 0., 0., 0., 0., 0.,
1., 0.], dtype=float32)>
```

In [29]:
```
len(example_labels)
```

Out[29]:
```
6
```

In [30]:
```
len(example_audio)
```

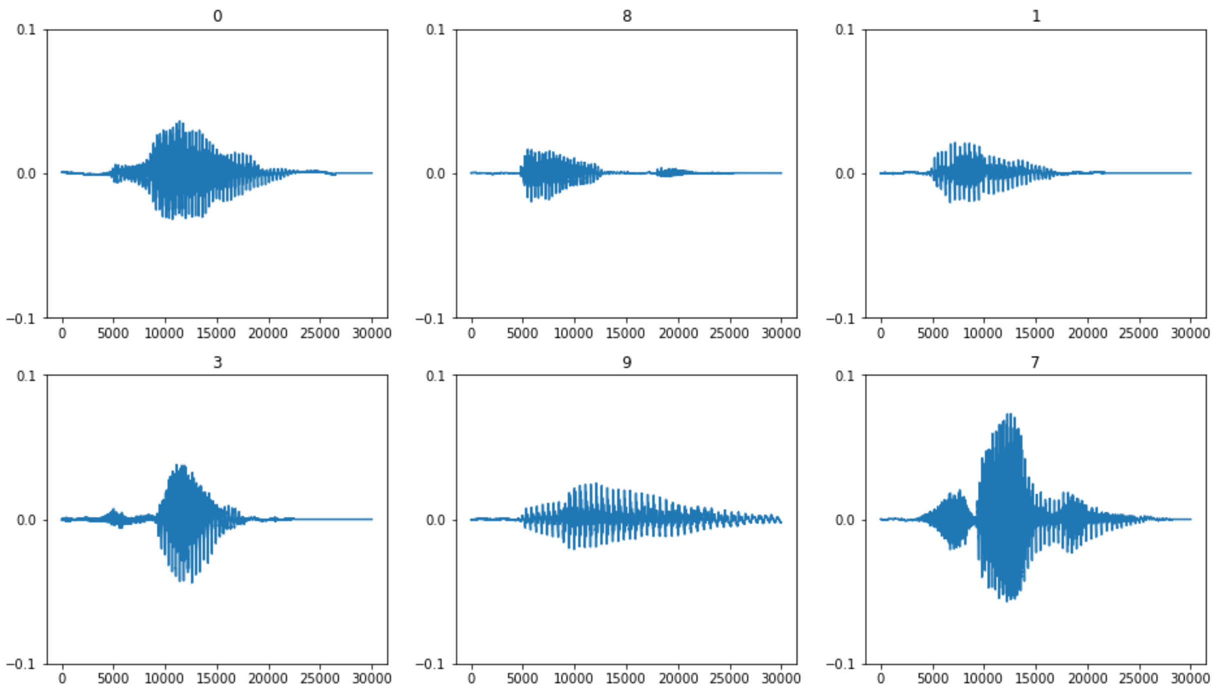
Out[30]:
```
6
```

In [31]:
```
example_audio
```

Out[31]:
```
<tf.Tensor: shape=(6, 30000), dtype=float32, numpy=
array([[ 5.7983398e-04,  5.7983398e-04,  5.7983398e-04, ...,
         0.0000000e+00,  0.0000000e+00,  0.0000000e+00],
       [ 1.8310547e-04,  2.1362305e-04,  2.1362305e-04, ...,
         0.0000000e+00,  0.0000000e+00,  0.0000000e+00],
       [-6.1035156e-05, -6.1035156e-05, -6.1035156e-05, ...,
         0.0000000e+00,  0.0000000e+00,  0.0000000e+00],
       [-3.9672852e-04, -3.3569336e-04, -3.3569336e-04, ...,
         0.0000000e+00,  0.0000000e+00,  0.0000000e+00],
       [-3.9672852e-04, -4.2724609e-04, -3.9672852e-04, ...,
        -1.9836426e-03, -2.0141602e-03, -1.9226074e-03],
       [ 2.4414062e-04,  2.7465820e-04,  2.4414062e-04, ...,
         0.0000000e+00,  0.0000000e+00,  0.0000000e+00]], dtype=float32)>
```

In [32]:
```
rows = 2
cols = 3
n = rows * cols
```

```
fig, axes = plt.subplots(rows, cols, figsize=(16, 9))

for i in range(n):
    if i>=n:
        break
    r = i // cols
    c = i % cols
    ax = axes[r][c]
    ax.plot(example_audio[i].numpy())
    ax.set_yticks(np.arange(-1.2, 1.2, 0.1))
    label = np.argmax(example_labels[i])
    ax.set_title(label)
    ax.set_ylim([-0.1,0.1])

plt.show()
```



Convert waveforms to spectrograms

The waveforms in the dataset are represented in the time domain. Next, we will transform the waveforms from the time-domain signals into the time-frequency-domain signals by computing the short-time Fourier transform (STFT) to convert the waveforms to as spectrograms, which show frequency changes over time and can be represented as 2D images. We will feed the spectrogram images into your neural network to train the model.

A Fourier transform (`tf.signal.fft`) converts a signal to its component frequencies, but loses all time information. In comparison, 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 we can run standard convolutions on.

Create a utility function for converting waveforms to spectrograms:

- The waveforms need to be of the same length, so that when you convert them to spectrograms, the results have similar dimensions. This can be done by simply zero-padding the audio clips that are shorter than one second (using `tf.zeros`).

- When calling `tf.signal.stft`, choose the `frame_length` and `frame_step` parameters such that the generated spectrogram "image" is almost square. For more information on the STFT parameters choice, refer to this Coursera video on audio signal processing and STFT.
- The STFT produces an array of complex numbers representing magnitude and phase. However, in this tutorial you'll only use the magnitude, which you can derive by applying `tf.abs` on the output of `tf.signal.stft`.

In [33]:
```python
def get_spectrogram(waveform):
    # Convert the waveform to a spectrogram via a STFT.
    spectrogram = tf.signal.stft(
        waveform, frame_length=233, frame_step=128)
    # Obtain the magnitude of the STFT.
    spectrogram = tf.abs(spectrogram)
    # Add a `channels` dimension, so that the spectrogram can be used
    # as image-like input data with convolution layers (which expect
    # shape (`batch_size`, `height`, `width`, `channels`).
    spectrogram = spectrogram[..., tf.newaxis]
    return spectrogram
```

In [34]:
```python
for i in range(6):
    label = np.argmax(example_labels[i])
    waveform = example_audio[i]
    spectrogram = get_spectrogram(waveform)

    print('Label:', label)
    print('Waveform shape:', waveform.shape)
    print('Spectrogram shape:', spectrogram.shape)
    print('Audio playback')
    ipd.display(ipd.Audio(waveform, rate=48000)) # Sample Rate of 48,000 gives the b
```

```
Label: 0
Waveform shape: (30000,)
Spectrogram shape: (233, 129, 1)
Audio playback
```

0:00 / 0:00

```
Label: 8
Waveform shape: (30000,)
Spectrogram shape: (233, 129, 1)
Audio playback
```

0:00 / 0:00

```
Label: 1
Waveform shape: (30000,)
Spectrogram shape: (233, 129, 1)
Audio playback
```

0:00 / 0:00

```
Label: 3
Waveform shape: (30000,)
Spectrogram shape: (233, 129, 1)
Audio playback
```

```
Label: 9
Waveform shape: (30000,)
Spectrogram shape: (233, 129, 1)
Audio playback
```

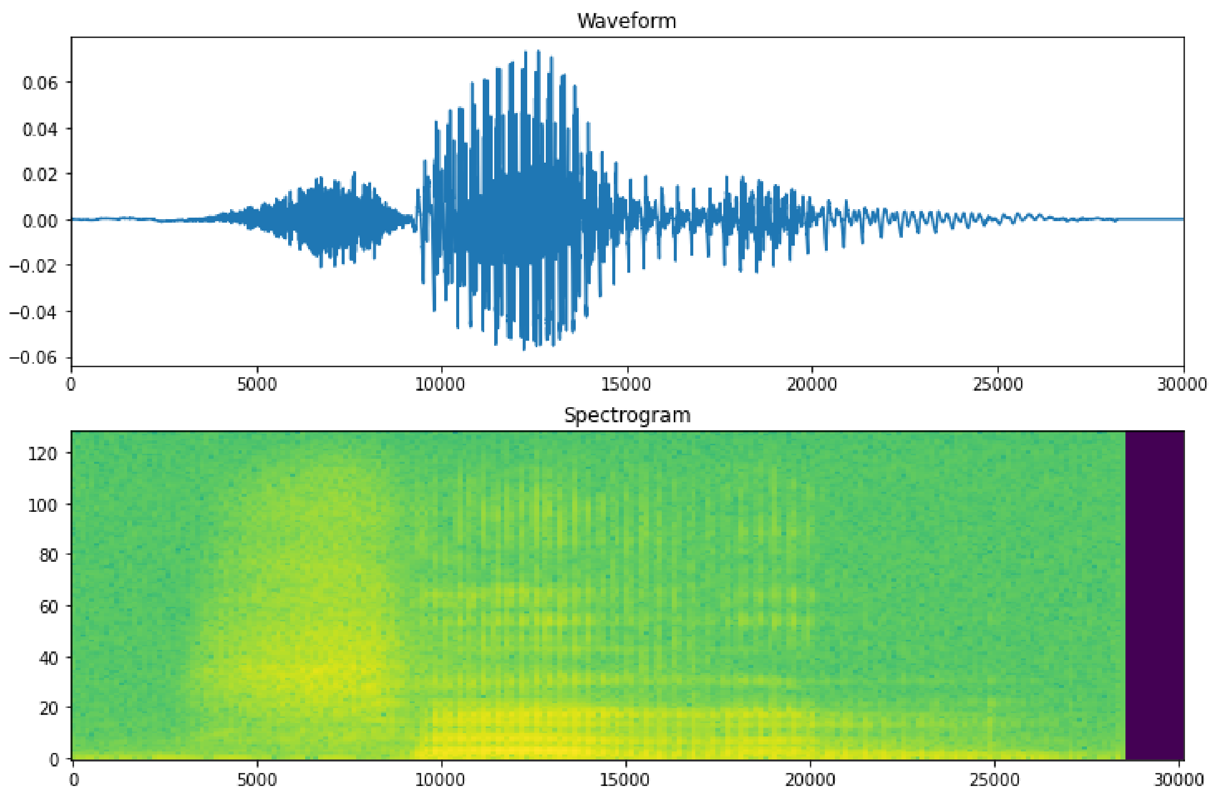
```
Label: 7
Waveform shape: (30000,)
Spectrogram shape: (233, 129, 1)
Audio playback
```

In [35]:

```python
def plot_spectrogram(spectrogram, ax):
    if len(spectrogram.shape) > 2:
        assert len(spectrogram.shape) == 3
        spectrogram = np.squeeze(spectrogram, axis=-1)
        # Convert the frequencies to log scale and transpose, so that the time is
        # represented on the x-axis (columns).
        # Add an epsilon to avoid taking a log of zero.
        log_spec = np.log(spectrogram.T + np.finfo(float).eps)
        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)
```

In [36]:

```python
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, 30000])

plot_spectrogram(spectrogram.numpy(), axes[1])
axes[1].set_title('Spectrogram')
plt.suptitle(str(label))
plt.show()
```



Now, create spectrogramn datasets from the audio datasets:

In [37]:
```
def make_spec_ds(ds):
    return ds.map(
        map_func=lambda audio,label: (get_spectrogram(audio), label),
        num_parallel_calls=tf.data.AUTOTUNE)
```

In [38]:
```
train_spectrogram_ds = make_spec_ds(train_ds)
val_spectrogram_ds = make_spec_ds(val_ds)
test_spectrogram_ds = make_spec_ds(test_ds)
```

In [40]:
```
val_spectrogram_ds
```

Out[40]:
```
<ParallelMapDataset element_spec=(TensorSpec(shape=(None, 233, 129, 1), dtype=tf.flo
at32, name=None), TensorSpec(shape=(None, 10), dtype=tf.float32, name=None))>
```

In [39]:
```
train_spectrogram_ds
```

Out[39]:
```
<ParallelMapDataset element_spec=(TensorSpec(shape=(None, 233, 129, 1), dtype=tf.flo
at32, name=None), TensorSpec(shape=(None, 10), dtype=tf.float32, name=None))>
```

Examine the spectrograms for different examples of the dataset:

In [41]:
```
for example_spectrograms, example_spect_labels in train_spectrogram_ds.take(1):
    break
```

In [42]:
```
example_spectrograms[0]
```

Out[42]:
```
<tf.Tensor: shape=(233, 129, 1), dtype=float32, numpy=
array([[[2.64210869e-02],
        [1.40206721e-02],
        [1.69853083e-04],
        ...,
        [1.90891995e-04],
        [3.11459298e-04],
        [3.75652686e-04]],

       [[2.88308337e-02],
        [1.58098880e-02],
        [1.09514291e-03],
        ...,
        [1.22389101e-04],
        [1.50650041e-04],
        [1.82525255e-04]],

       [[3.77699360e-02],
        [2.13006083e-02],
        [8.10259720e-04],
        ...,
        [3.64726962e-04],
        [1.21405355e-04],
        [2.05786899e-04]],

       ...,

       [[5.56288473e-02],
        [3.26090939e-02],
        [3.39638814e-03],
        ...,
        [2.95879669e-04],
        [8.73188328e-05],
        [7.89575279e-05]],

       [[5.45259491e-02],
        [2.95899436e-02],
        [1.74593274e-03],
        ...,
        [2.09477657e-04],
        [1.58494993e-04],
        [1.64337456e-04]],

       [[6.62344992e-02],
        [3.79272103e-02],
        [3.26105393e-03],
        ...,
        [2.35689833e-04],
        [9.46388536e-05],
        [2.34693289e-05]]], dtype=float32)>
```

In [43]:
```
len(example_spectrograms)
```
Out[43]:
```
6
```

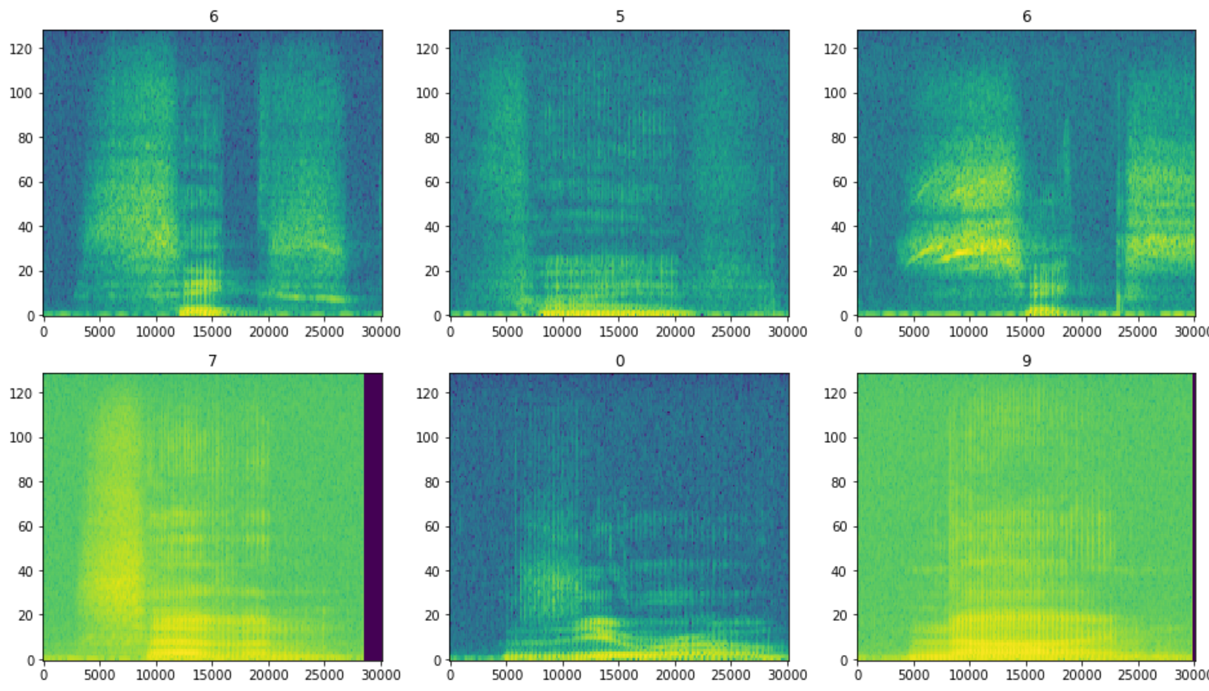
In [44]:
```
len(example_spect_labels)
```
Out[44]:
```
6
```

In [45]:
```
example_spect_labels[0]
```
Out[45]:
```
<tf.Tensor: shape=(10,), dtype=float32, numpy=array([0., 0., 0., 0., 0., 0., 1., 0.,
0., 0.], dtype=float32)>
```

In [46]:
```
np.argmax(example_spect_labels[0])
```

Out[46]:
```
6
```

In [47]:
```
rows = 2
cols = 3
n = rows*cols
fig, axes = plt.subplots(rows, cols, figsize=(16, 9))

for i in range(n):
    r = i // cols
    c = i % cols
    ax = axes[r][c]
    plot_spectrogram(example_spectrograms[i].numpy(), ax)
    ax.set_title(np.argmax(example_spect_labels[i]))

plt.show()
```



Add `Dataset.cache` and `Dataset.prefetch` operations to reduce read latency while training the model:

In [48]:
```
train_spectrogram_ds = train_spectrogram_ds.cache().shuffle(10000).prefetch(tf.data.

val_spectrogram_ds = val_spectrogram_ds.cache().prefetch(tf.data.AUTOTUNE)

test_spectrogram_ds = test_spectrogram_ds.cache().prefetch(tf.data.AUTOTUNE)
```

Deep Neural Network model

In [49]:
```
from tensorflow.keras import layers
from tensorflow.keras import models
```

In [50]:
```
example_spectrograms.shape
```

Out[50]:
```
TensorShape([6, 233, 129, 1])
```

The `example_spectrograms` is of the format _(batch*size,height,width,channel)*

- **datatype : Tensor**
- **size (height,width) : (233,129)**
- **channel : 1 (mono)**

In [51]:
```
example_spectrograms[0]
```

Out[51]:
```
<tf.Tensor: shape=(233, 129, 1), dtype=float32, numpy=
array([[[2.64210869e-02],
        [1.40206721e-02],
        [1.69853083e-04],
        ...,
        [1.90891995e-04],
        [3.11459298e-04],
        [3.75652686e-04]],

       [[2.88308337e-02],
        [1.58098880e-02],
        [1.09514291e-03],
        ...,
        [1.22389101e-04],
        [1.50650041e-04],
        [1.82525255e-04]],

       [[3.77699360e-02],
        [2.13006083e-02],
        [8.10259720e-04],
        ...,
        [3.64726962e-04],
        [1.21405355e-04],
        [2.05786899e-04]],

       ...,

       [[5.56288473e-02],
        [3.26090939e-02],
        [3.39638814e-03],
        ...,
        [2.95879669e-04],
        [8.73188328e-05],
        [7.89575279e-05]],

       [[5.45259491e-02],
        [2.95899436e-02],
        [1.74593274e-03],
        ...,
        [2.09477657e-04],
        [1.58494993e-04],
        [1.64337456e-04]],

       [[6.62344992e-02],
        [3.79272103e-02],
        [3.26105393e-03],
        ...,
        [2.35689833e-04],
        [9.46388536e-05],
        [2.34693289e-05]]], dtype=float32)>
```

In [52]:
```
example_spectrograms.shape[1:]
```

Out[52]:
```
TensorShape([233, 129, 1])
```

In [53]:
```
input_shape = example_spectrograms.shape[1:]
print('Input shape:', input_shape)
num_labels = len(label_names)

# Deep Neural Network Architecture
model = models.Sequential([
    layers.Input(shape=input_shape),
    layers.Resizing(230,120),
    layers.Flatten(),
    layers.Dense(512, activation='relu'),
    layers.Dense(128, activation='relu'),
    layers.Dense(64, activation='relu'),
    layers.Dense(num_labels,activation='softmax'),
])

model.summary()
```

```
Input shape: (233, 129, 1)
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 resizing (Resizing)         (None, 230, 120, 1)       0         
                                                                 
 flatten (Flatten)           (None, 27600)             0         
                                                                 
 dense (Dense)               (None, 512)               14131712  
                                                                 
 dense_1 (Dense)             (None, 128)               65664     
                                                                 
 dense_2 (Dense)             (None, 64)                8256      
                                                                 
 dense_3 (Dense)             (None, 10)                650       
                                                                 
=================================================================
Total params: 14,206,282
Trainable params: 14,206,282
Non-trainable params: 0
_________________________________________________________________
```

In [54]:
```
model.compile(
    optimizer=tf.keras.optimizers.RMSprop(),
    loss=tf.keras.losses.CategoricalCrossentropy(),
    metrics=['accuracy'],
)
```

In [56]:
```
EPOCHS = 10
history = model.fit(
    train_spectrogram_ds,
    validation_data=val_spectrogram_ds,
    epochs=EPOCHS
)
```

```
Epoch 1/10
995/995 [==============================] - 130s 130ms/step - loss: 0.4966 - accurac
y: 0.8612 - val_loss: 0.2779 - val_accuracy: 0.9207
Epoch 2/10
995/995 [==============================] - 120s 121ms/step - loss: 0.1908 - accurac
y: 0.9610 - val_loss: 0.0511 - val_accuracy: 0.9852
Epoch 3/10
995/995 [==============================] - 122s 123ms/step - loss: 0.1237 - accurac
y: 0.9752 - val_loss: 0.0816 - val_accuracy: 0.9772
Epoch 4/10
995/995 [==============================] - 122s 122ms/step - loss: 0.1132 - accurac
y: 0.9834 - val_loss: 0.0802 - val_accuracy: 0.9812
Epoch 5/10
995/995 [==============================] - 122s 123ms/step - loss: 0.1479 - accurac
y: 0.9849 - val_loss: 0.3788 - val_accuracy: 0.9516
Epoch 6/10
995/995 [==============================] - 121s 122ms/step - loss: 0.0985 - accurac
y: 0.9861 - val_loss: 0.1856 - val_accuracy: 0.9866
Epoch 7/10
995/995 [==============================] - 121s 122ms/step - loss: 0.0865 - accurac
y: 0.9898 - val_loss: 0.2683 - val_accuracy: 0.9825
Epoch 8/10
995/995 [==============================] - 121s 122ms/step - loss: 0.0698 - accurac
y: 0.9896 - val_loss: 0.4804 - val_accuracy: 0.9637
Epoch 9/10
995/995 [==============================] - 121s 122ms/step - loss: 0.0865 - accurac
y: 0.9920 - val_loss: 0.4242 - val_accuracy: 0.9651
Epoch 10/10
995/995 [==============================] - 121s 122ms/step - loss: 0.1069 - accurac
y: 0.9893 - val_loss: 0.1657 - val_accuracy: 0.9825
```

In [57]:
```
history.history
```

Out[57]:
```
{'loss': [0.49664798378944397,
  0.19082514941692352,
  0.12368419766426086,
  0.11317168921232224,
  0.147870272397995,
  0.09851767867803574,
  0.08651147782802582,
  0.0697917640209198,
  0.08651598542928696,
  0.10688021779060364],
 'accuracy': [0.8612368106842041,
  0.9609519243240356,
  0.975196897983551,
  0.9834087491035461,
  0.9849170446395874,
  0.9860901832580566,
  0.9897770881652832,
  0.9896095395088196,
  0.9919557571411133,
  0.9892743229866028],
 'val_loss': [0.27786675095558167,
  0.05108669400215149,
  0.08159353584051132,
  0.08018101751804352,
  0.3787652850151062,
  0.1855706423521042,
  0.2682514488697052,
  0.48040804266929626,
  0.4241775572299957,
  0.1656823754310608],
 'val_accuracy': [0.9206989407539368,
  0.9852150678634644,
  0.977150559425354,
  0.9811828136444092,
  0.9516128897666931,
  0.9865591526031494,
  0.9825268983840942,
  0.9637096524238586,
  0.9650537371635437,
  0.9825268983840942]}
```

In [58]:

```python
model.evaluate(test_spectrogram_ds, return_dict=True)
```

```
125/125 [==============================] - 2s 17ms/step - loss: 1.1754 - accuracy:
0.9813
```

Out[58]:

```
{'loss': 1.1753666400909424, 'accuracy': 0.98128342628479}
```

The neural network (model) learned well its

- Accuracy : 98.93%
- Validation Accuracy : 98.25%
- Test Accuracy : 98.13%

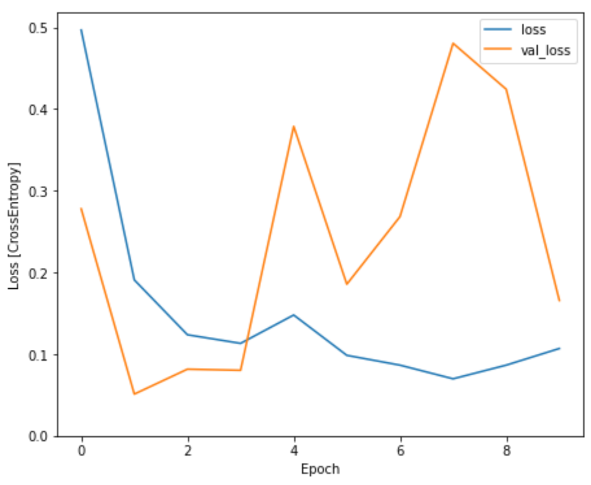
This model is best fitted (not overfitted and underfitted) on train data & test data as well

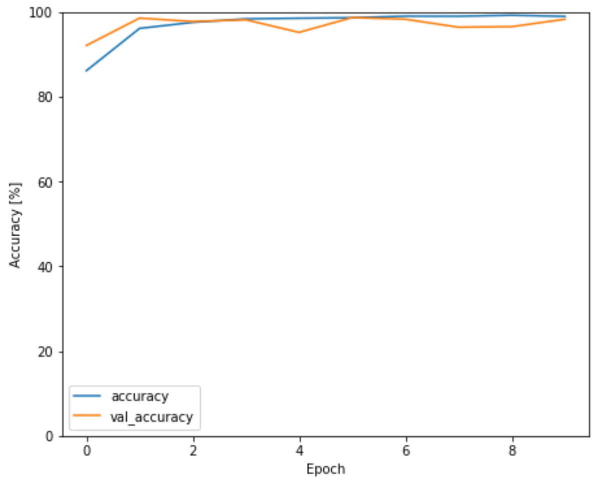
In [60]:

```python
metrics = history.history
plt.figure(figsize=(16,6))
plt.subplot(1,2,1)
plt.plot(history.epoch, metrics['loss'], metrics['val_loss'])
plt.legend(['loss', 'val_loss'])
plt.ylim([0, max(plt.ylim())])
plt.xlabel('Epoch')
plt.ylabel('Loss [CrossEntropy]')

plt.subplot(1,2,2)
plt.plot(history.epoch, 100*np.array(metrics['accuracy']), 100*np.array(metrics['val
plt.legend(['accuracy', 'val_accuracy'])
plt.ylim([0, 100])
plt.xlabel('Epoch')
plt.ylabel('Accuracy [%]')
```

Out[60]:

```
Text(0, 0.5, 'Accuracy [%]')
```





In [59]:

```python
y_pred = model.predict(test_spectrogram_ds)
```

```
125/125 [==============================] - 8s 11ms/step
```

In [61]:
```
y_pred = tf.argmax(y_pred, axis=1)
```

In [62]:
```
y_pred
```

Out[62]:
```
<tf.Tensor: shape=(748,), dtype=int64, numpy=
array([5, 2, 6, 0, 6, 2, 5, 4, 2, 3, 8, 6, 1, 6, 9, 6, 3, 3, 1, 5, 1, 3,
       7, 2, 5, 5, 1, 2, 5, 4, 8, 5, 6, 0, 4, 9, 0, 6, 0, 9, 4, 7, 1, 3,
       5, 4, 7, 7, 3, 7, 5, 5, 1, 0, 0, 6, 1, 4, 5, 8, 7, 9, 2, 7, 5, 5,
       2, 7, 7, 6, 1, 2, 0, 4, 9, 6, 3, 9, 8, 3, 0, 7, 6, 8, 7, 2, 6, 3,
       6, 7, 5, 7, 0, 1, 3, 4, 6, 0, 1, 3, 9, 2, 5, 7, 0, 2, 8, 2, 0, 8,
       3, 8, 8, 3, 0, 3, 6, 6, 0, 2, 4, 9, 9, 7, 6, 3, 0, 6, 5, 8, 9, 1,
       3, 4, 0, 4, 8, 7, 6, 1, 3, 2, 5, 9, 6, 9, 8, 3, 4, 7, 0, 7, 0, 9,
       0, 4, 4, 8, 6, 7, 4, 7, 0, 0, 6, 5, 1, 1, 0, 6, 8, 6, 2, 6, 3, 8,
       7, 6, 7, 6, 4, 0, 4, 5, 6, 1, 8, 1, 2, 3, 0, 1, 1, 7, 8, 5, 8, 8,
       4, 3, 8, 9, 7, 9, 9, 9, 2, 4, 7, 8, 1, 1, 3, 2, 8, 8, 1, 4, 4, 9,
       3, 5, 1, 6, 9, 2, 7, 6, 2, 8, 4, 3, 8, 6, 3, 7, 2, 4, 6, 8, 3, 9,
       2, 1, 9, 7, 2, 2, 1, 0, 3, 9, 7, 6, 4, 2, 7, 7, 9, 6, 8, 5, 1, 0,
       3, 2, 2, 0, 6, 9, 7, 3, 3, 0, 8, 8, 8, 8, 2, 4, 6, 4, 8, 7, 0, 7,
       3, 4, 7, 6, 4, 9, 4, 1, 0, 5, 5, 2, 2, 4, 6, 2, 6, 8, 2, 1, 6, 2,
       9, 3, 3, 8, 7, 9, 8, 6, 7, 3, 5, 3, 3, 9, 5, 2, 6, 9, 4, 1, 2, 8,
       6, 4, 4, 8, 4, 0, 2, 2, 8, 2, 2, 5, 5, 3, 5, 4, 7, 6, 5, 9, 9, 4,
       4, 7, 6, 2, 0, 3, 4, 1, 0, 0, 2, 7, 1, 5, 6, 8, 6, 7, 6, 4, 8, 2,
       7, 7, 8, 9, 0, 1, 8, 8, 0, 4, 9, 1, 6, 9, 4, 1, 1, 3, 2, 4, 2, 5,
       6, 9, 2, 6, 7, 1, 0, 7, 3, 3, 0, 1, 4, 6, 3, 7, 8, 9, 5, 9, 6, 8,
       8, 1, 2, 2, 4, 3, 6, 0, 0, 6, 1, 6, 0, 5, 2, 9, 2, 1, 6, 7, 3, 3,
       4, 6, 7, 0, 4, 2, 8, 1, 9, 4, 3, 2, 1, 5, 2, 9, 2, 5, 3, 0, 5, 7,
       8, 0, 0, 6, 6, 8, 3, 4, 7, 2, 3, 7, 5, 0, 7, 7, 6, 1, 1, 7, 0, 7,
       0, 4, 8, 4, 3, 5, 8, 6, 6, 9, 3, 5, 0, 7, 9, 5, 4, 6, 7, 7, 8, 7,
       1, 2, 2, 5, 4, 5, 7, 2, 8, 2, 1, 9, 1, 2, 2, 5, 0, 3, 4, 6, 7, 0,
       6, 4, 4, 9, 7, 9, 7, 4, 9, 6, 9, 6, 2, 6, 7, 4, 1, 8, 4, 6, 1, 5,
       1, 1, 5, 8, 9, 0, 4, 0, 9, 8, 6, 0, 6, 8, 4, 4, 9, 4, 8, 4, 8, 3,
       6, 8, 5, 7, 9, 0, 5, 2, 5, 3, 3, 8, 2, 0, 0, 7, 8, 6, 8, 9, 8, 6,
       1, 0, 6, 4, 4, 9, 9, 5, 2, 4, 4, 3, 3, 9, 5, 7, 9, 0, 9, 9, 3, 3,
       5, 9, 6, 4, 5, 0, 0, 3, 5, 9, 0, 4, 3, 8, 0, 7, 8, 3, 7, 1, 2, 1,
       1, 0, 1, 2, 3, 4, 3, 6, 2, 6, 1, 5, 6, 4, 7, 3, 7, 8, 3, 3, 1, 7,
       8, 6, 0, 9, 2, 6, 9, 9, 7, 3, 5, 3, 8, 8, 5, 7, 8, 4, 0, 3, 2, 5,
       1, 2, 0, 4, 9, 1, 7, 8, 3, 9, 9, 4, 2, 1, 8, 9, 9, 9, 6, 5, 3, 8,
       5, 9, 0, 9, 7, 5, 6, 9, 7, 4, 4, 8, 9, 0, 9, 2, 3, 0, 3, 0, 8, 0,
       0, 1, 1, 9, 2, 0, 2, 8, 9, 8, 3, 0, 8, 4, 9, 9, 9, 7, 4, 2, 3, 3],
      dtype=int64)>
```

In [63]:
```
y_true = tf.concat(list(test_spectrogram_ds.map(lambda s,lab: lab)), axis=0)
```

In [64]:
```
y_true
```

Out[64]:
```
<tf.Tensor: shape=(748, 10), dtype=float32, numpy=
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 1., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 1., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]], dtype=float32)>
```

In [ ]:
```

```

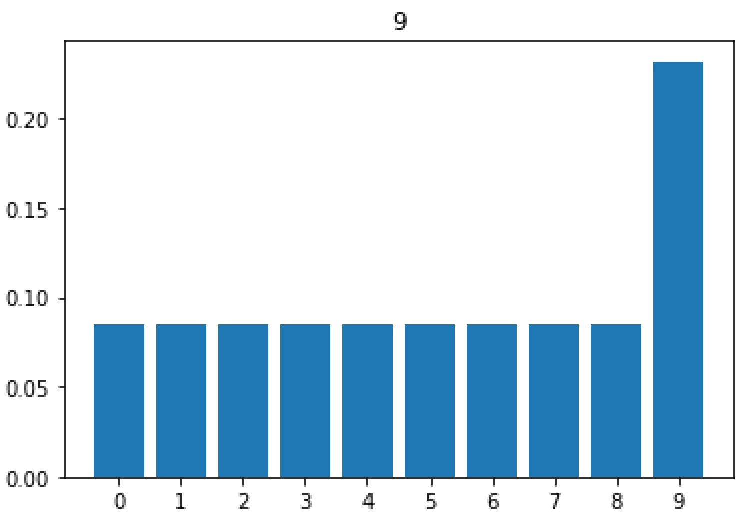
Finally, let's verify the model's prediction output using an input audio file to check the performance.

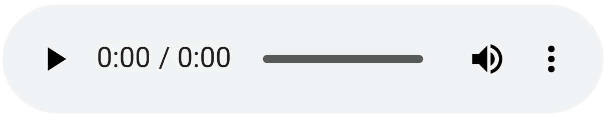
I have already separated each audio file for each digits totally 10 digits into a separate folder called `test_data_dir` it contains 10 wave audio files

In [69]:
```
abt(test_data_dir)
```

```
Total subfolders in the Main directory ../DL/Datasets/AudioMNIST/TestAudioData/: 10
```

In [68]:
```
x = test_data_dir+'9_09_26.wav'
x = tf.io.read_file(str(x))
x, sample_rate = tf.audio.decode_wav(x, desired_channels=1, desired_samples=30000,)
x = tf.squeeze(x, axis=-1)
waveform = x
x = get_spectrogram(x)
x = x[tf.newaxis,...]

prediction = model(x)
plt.bar(label_names, tf.nn.softmax(prediction[0]))
plt.title('9')
plt.show()

ipd.display(ipd.Audio(waveform, rate=48000))
```





Here, our model predicted the audio output as the digit **9** so it learnt well. Let's predict for other digits in the `test_data_dir`

In [71]:
```
from glob import glob
```

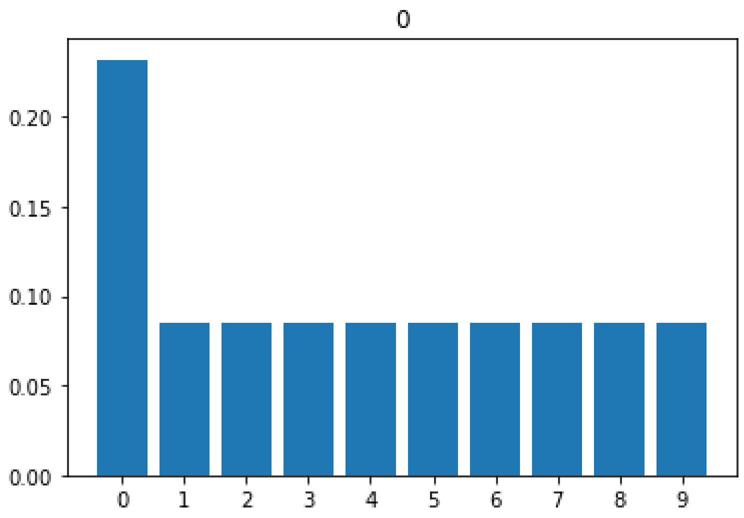
In [72]:
```
audfiles = glob('../DL/Datasets/AudioMNIST/TestAudioData/*.wav')
```

In [73]:
```
audfiles
```

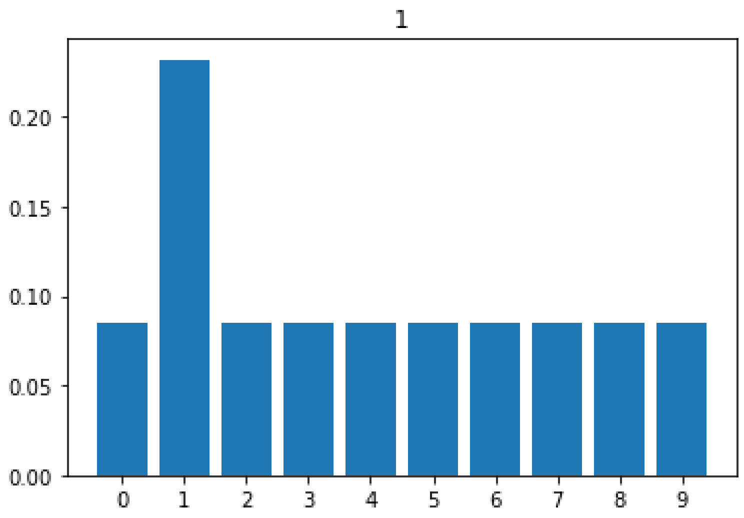
Out[73]:
```
['../DL/Datasets/AudioMNIST/TestAudioData\\0_04_4.wav',
 '../DL/Datasets/AudioMNIST/TestAudioData\\1_08_48.wav',
 '../DL/Datasets/AudioMNIST/TestAudioData\\2_15_40.wav',
 '../DL/Datasets/AudioMNIST/TestAudioData\\3_01_7.wav',
 '../DL/Datasets/AudioMNIST/TestAudioData\\4_05_9.wav',
 '../DL/Datasets/AudioMNIST/TestAudioData\\5_08_13.wav',
 '../DL/Datasets/AudioMNIST/TestAudioData\\6_06_33.wav',
 '../DL/Datasets/AudioMNIST/TestAudioData\\7_11_3.wav',
 '../DL/Datasets/AudioMNIST/TestAudioData\\8_04_48.wav',
 '../DL/Datasets/AudioMNIST/TestAudioData\\9_09_26.wav']
```

In [75]:

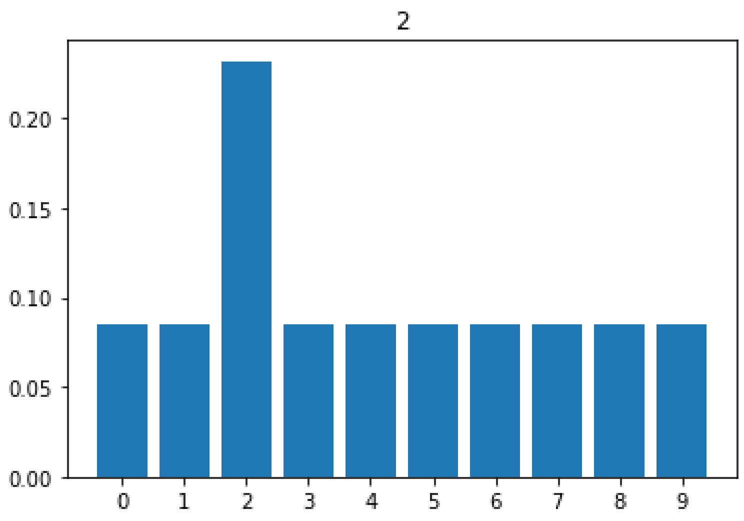
```python
for i in audfiles:
    x = tf.io.read_file(str(i))
    x, sample_rate = tf.audio.decode_wav(x, desired_channels=1, desired_samples=3000
    x = tf.squeeze(x, axis=-1)
    waveform = x
    x = get_spectrogram(x)
    x = x[tf.newaxis,...]

    prediction = model(x)
    plt.bar(label_names, tf.nn.softmax(prediction[0]))
    plt.title((i.split('\\')[1]).split('_')[0])
    plt.show()

    ipd.display(ipd.Audio(waveform, rate=48000))
```



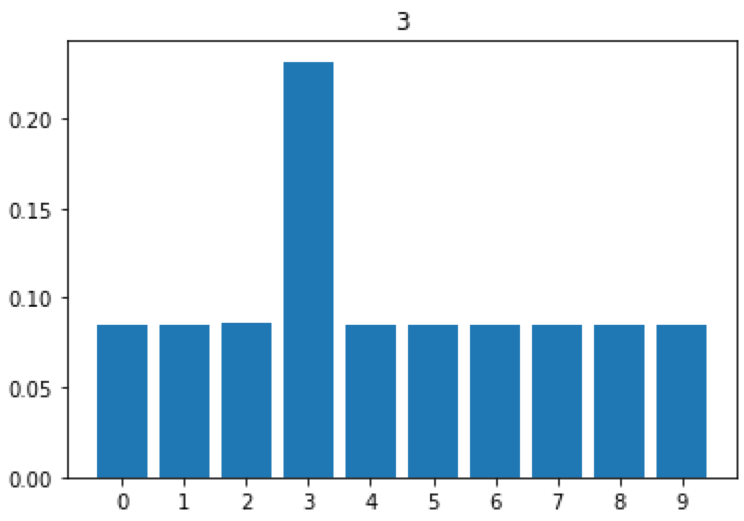
0:00 / 0:00



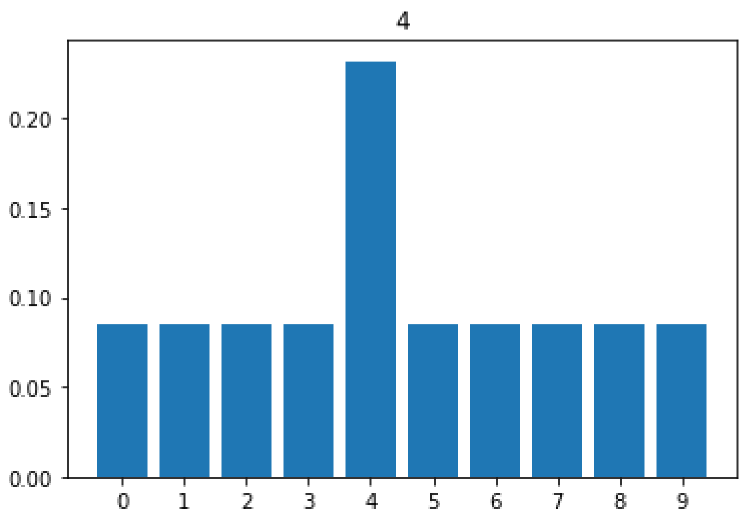
0:00 / 0:00



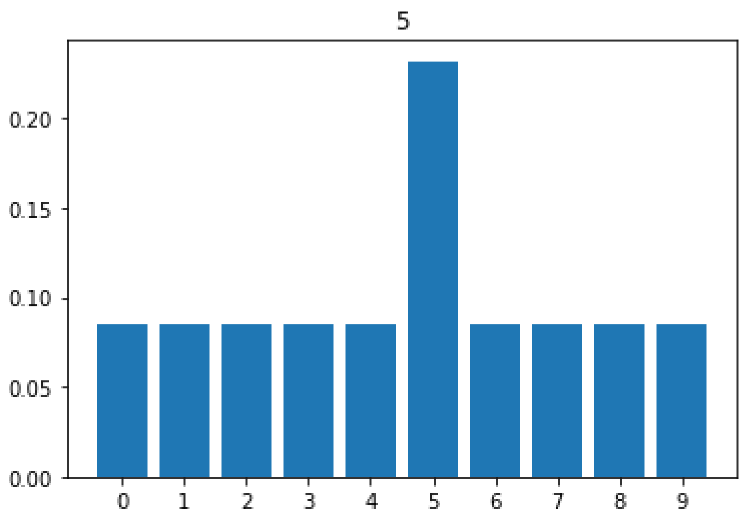
0:00 / 0:00



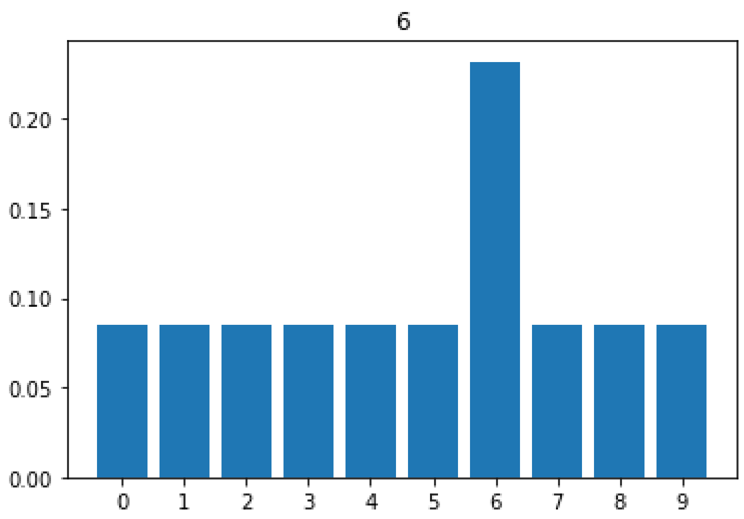
0:00 / 0:00



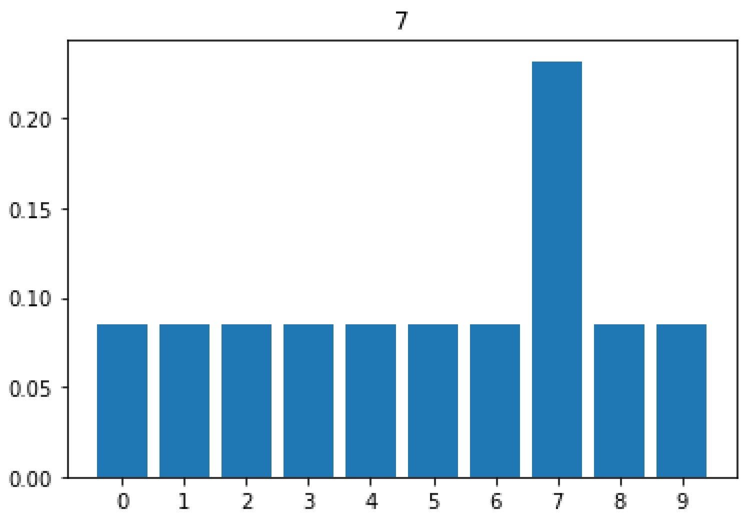
0:00 / 0:00



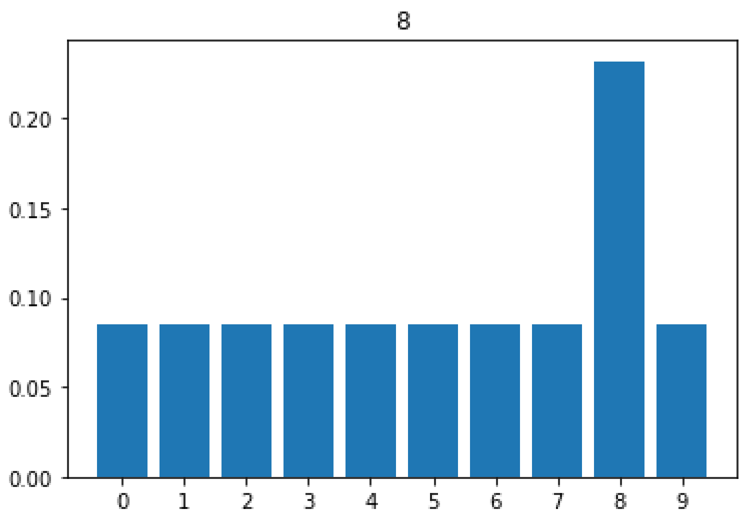
0:00 / 0:00



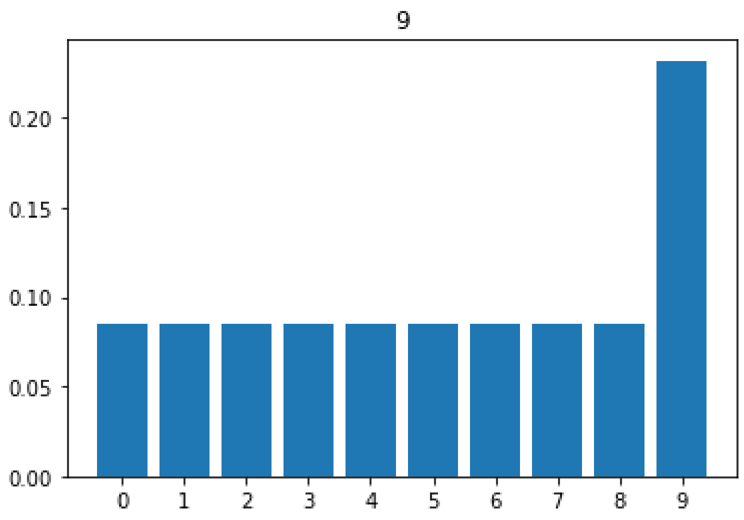
0:00 / 0:00



0:00 / 0:00



0:00 / 0:00



0:00 / 0:00

So, finally our model classified all the test data (i.e., digits in audio files) perfectly.

References:

1. Tensorflow - Simple audio recognition: Recognizing keywords, https://www.tensorflow.org/tutorials/audio/simple_audio
2. Digital Audio basics, https://home.adelphi.edu/~siegfried/cs170/170l2.pdf
3. Waveform audio - Wikipedia, https://en.wikipedia.org/wiki/WAV#:~:text=Waveform%20Audio%20File%20Format%20(WAVE,W