In [58]:
```python
# Imports
import os
import pathlib
import zipfile
import wave


import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import tensorflow as tf

from tensorflow.keras import layers
from tensorflow.keras import models
from IPython import display

seed = 42
tf.random.set_seed(seed)
np.random.seed(seed)
```

In [59]:
```python
# Download Data
DATASET_PATH = 'voice_data/mini_speech_commands'

data_dir = pathlib.Path(DATASET_PATH)
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_commands.zip",
      extract=True,
      cache_dir='.', cache_subdir='voice_data')
```

In [60]:
```python
commands = np.array(tf.io.gfile.listdir(str(data_dir)))
commands = commands[(commands != 'README.md') & (commands != '.DS_Store')]
```

In [61]:
```python
# Load Data
train_ds, val_ds = tf.keras.utils.audio_dataset_from_directory(
    directory=data_dir,
    batch_size=64,
    validation_split=0.2,
    seed=0,
    output_sequence_length=16000,
    subset='both')

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

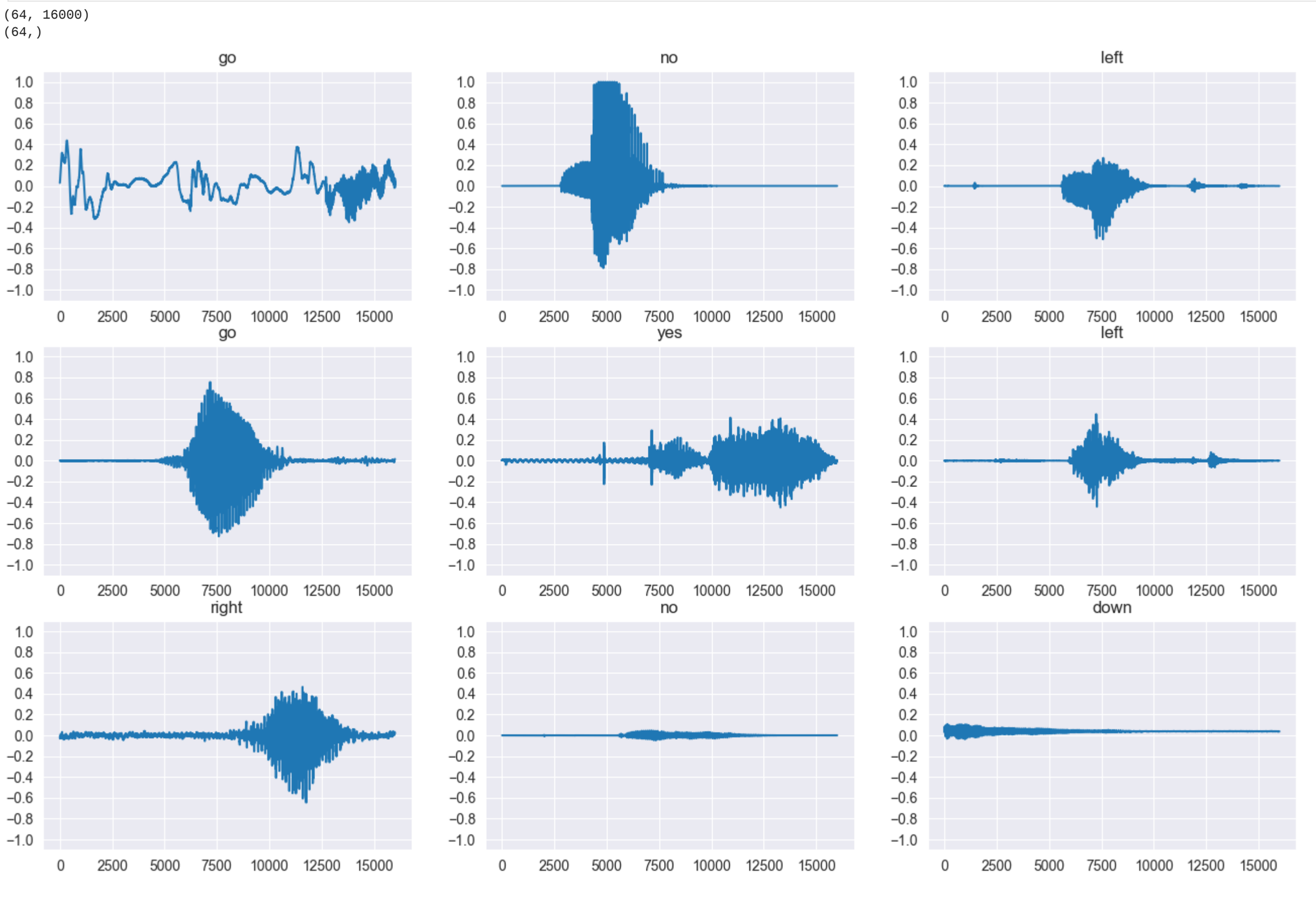
```
Found 8000 files belonging to 8 classes.
Using 6400 files for training.
Using 1600 files for validation.

label names: ['down' 'go' 'left' 'no' 'right' 'stop' 'up' 'yes']
```

In [62]:
```python
# Reduce audio channels to one
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 [63]:
```python
# Split val data set into test and val set
test_ds = val_ds.shard(num_shards=2, index=0)
val_ds = val_ds.shard(num_shards=2, index=1)
```

In [64]:
```python
# Plot a few audio waveforms
for example_audio, example_labels in train_ds.take(1):
  print(example_audio.shape)
  print(example_labels.shape)

label_names[[1,1,3,0]]
plt.figure(figsize=(16, 10))
rows = 3
cols = 3
n = rows * cols
for i in range(n):
  plt.subplot(rows, cols, i+1)
  audio_signal = example_audio[i]
  plt.plot(audio_signal)
  plt.title(label_names[example_labels[i]])
  plt.yticks(np.arange(-1.2, 1.2, 0.2))
  plt.ylim([-1.1, 1.1])
```

```
(64, 16000)
(64,)
```



In [65]:
```python
# Function to Convert Waveform to Spectrogram
def get_spectrogram(waveform):
  # Convert the waveform to a spectrogram via a STFT.
  spectrogram = tf.signal.stft(
      waveform, frame_length=255, 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

for i in range(3):
  label = label_names[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')
  display.display(display.Audio(waveform, rate=16000))
```

```
Label: go
Waveform shape: (16000,)
Spectrogram shape: (124, 129, 1)
Audio playback
```

0:00 / 0:01

```
Label: no
Waveform shape: (16000,)
Spectrogram shape: (124, 129, 1)
Audio playback
```

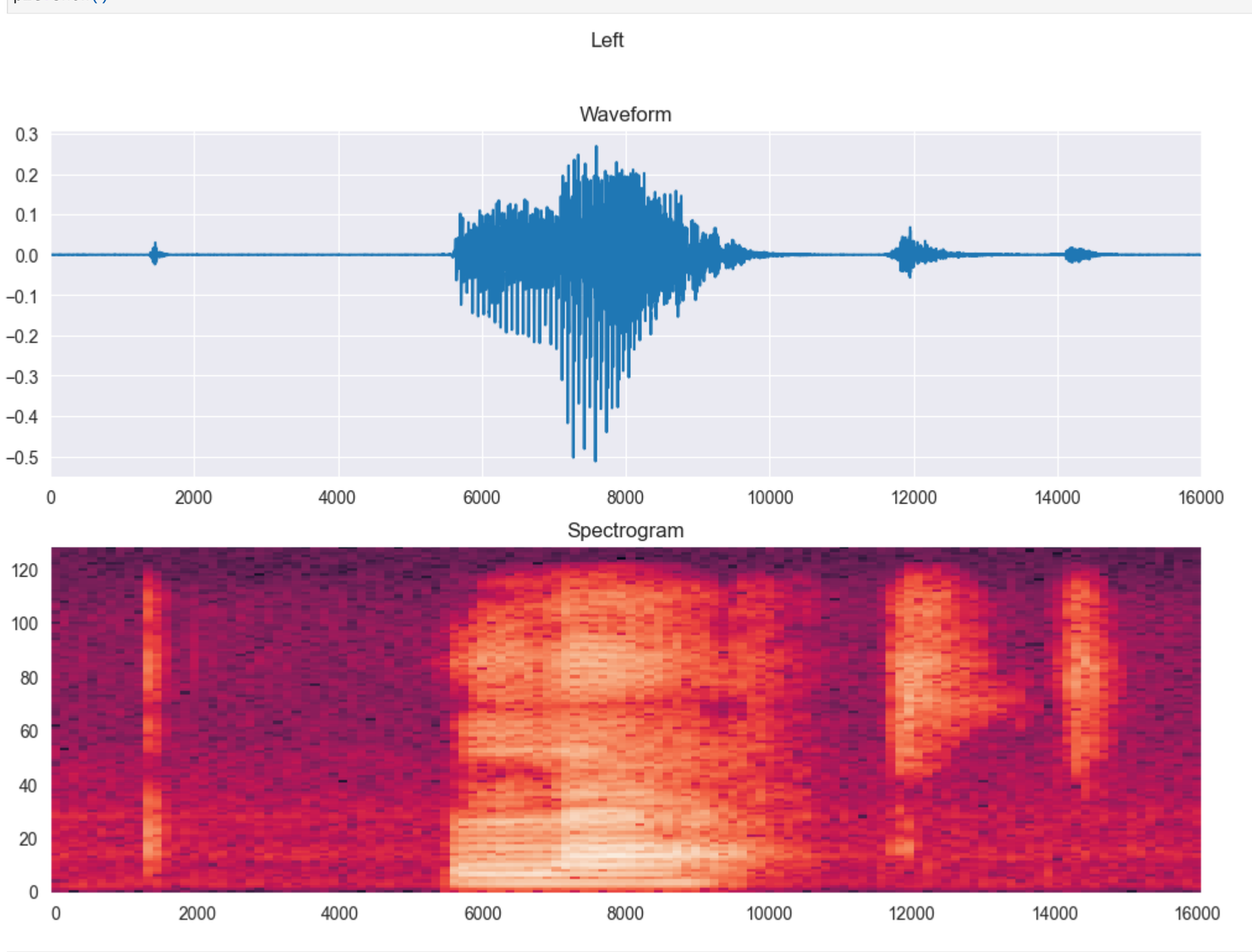
0:00 / 0:01

```
Label: left
Waveform shape: (16000,)
Spectrogram shape: (124, 129, 1)
Audio playback
```

0:00 / 0:01

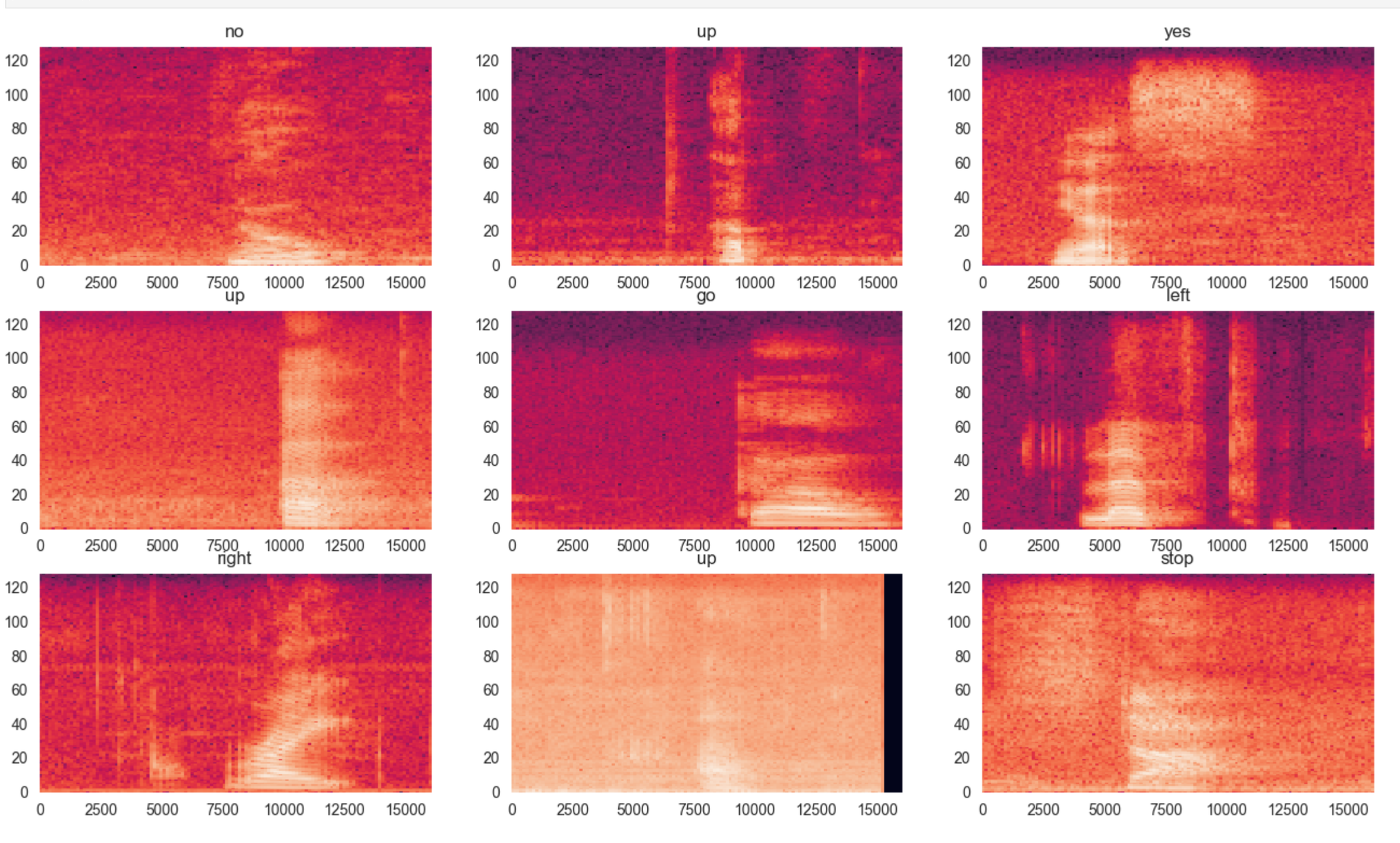
In [66]:
```python
# Function to plot Spectrogram
def plot_spectrogram(spectrogram, ax):
  if len(spectrogram.shape) > 2:
    assert len(spectrogram.shape) == 3
    spectrogram = np.squeeze(spectrogram, axis=-1)
  # Convert the frequencies to log scale and transpose, so that the time is
  # represented 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 [67]:
```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, 16000])

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



In [68]:
```python
# Convert all Waveform dataset to Spectrogram
def make_spec_ds(ds):
  return ds.map(
      map_func=lambda audio,label: (get_spectrogram(audio), label),
      num_parallel_calls=tf.data.AUTOTUNE)

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 [69]:
```python
# What does the spectrogram look like for examples
for example_spectrograms, example_spect_labels in train_spectrogram_ds.take(1):
  break

rows = 3
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(label_names[example_spect_labels[i].numpy()])

plt.show()
```



In [70]:
```python
# Building model
train_spectrogram_ds = train_spectrogram_ds.cache().shuffle(10000).prefetch(tf.data.AUTOTUNE)
val_spectrogram_ds = val_spectrogram_ds.cache().prefetch(tf.data.AUTOTUNE)
test_spectrogram_ds = test_spectrogram_ds.cache().prefetch(tf.data.AUTOTUNE)

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

# Instantiate the `tf.keras.layers.Normalization` layer.
norm_layer = layers.Normalization()
# Fit the state of the layer to the spectrograms
# with `Normalization.adapt`.
norm_layer.adapt(data=train_spectrogram_ds.map(map_func=lambda spec, label: spec))

model = models.Sequential([
    layers.Input(shape=input_shape),
    # Downsample the input.
    layers.Resizing(32, 32),
    # Normalize.
    norm_layer,
    # We have an image so we can use convolution
    layers.Conv2D(32, 3, activation='relu'),
    layers.Conv2D(64, 3, activation='relu'),
    layers.MaxPooling2D(),
    layers.Dropout(0.25),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(num_labels),
])

model.summary()
```

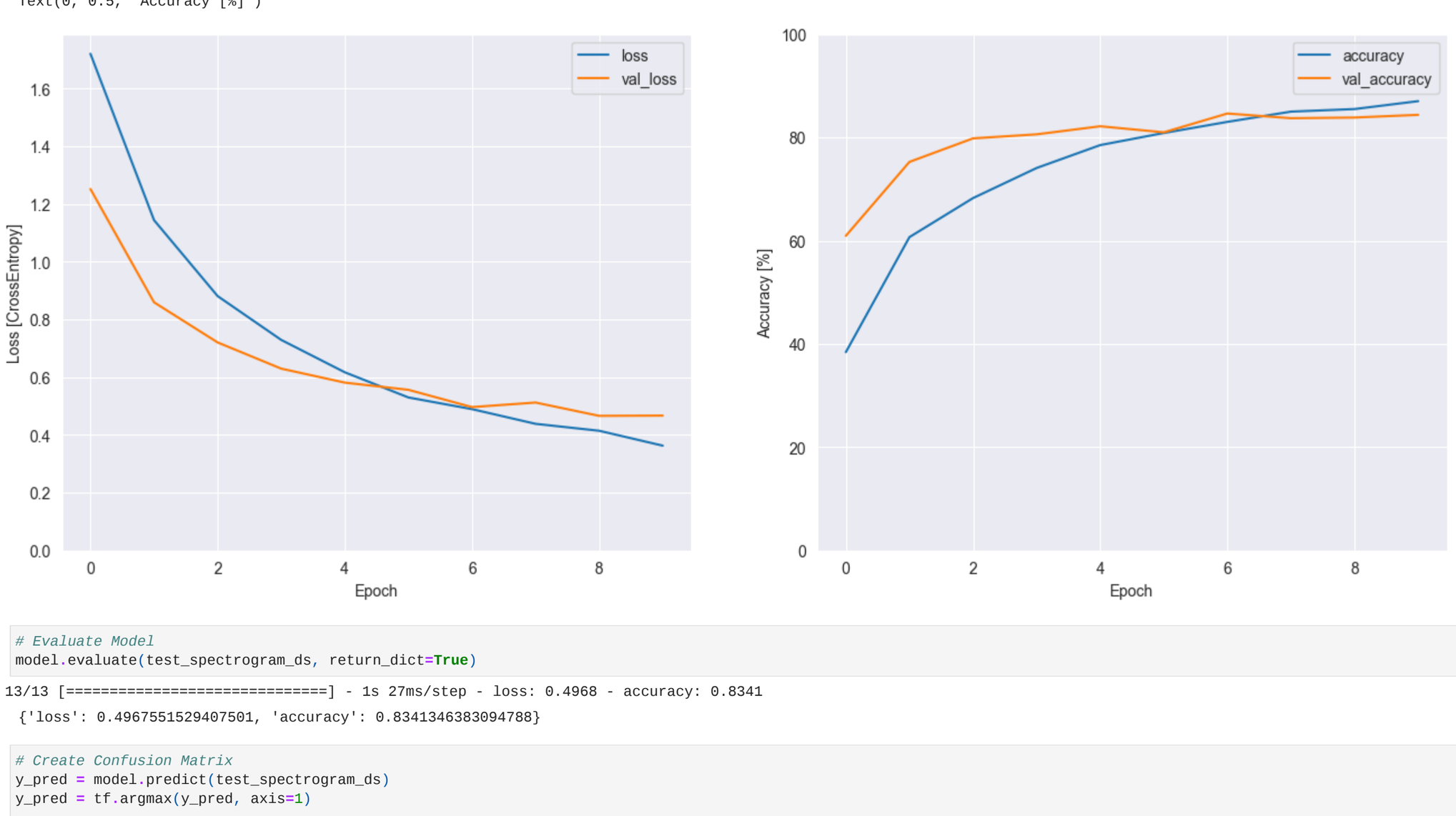
```
Input shape: (124, 129, 1)
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 resizing_2 (Resizing)       (None, 32, 32, 1)         0

 normalization_2 (Normaliza  (None, 32, 32, 1)         3
 tion)

 conv2d_4 (Conv2D)           (None, 30, 30, 32)        320

 conv2d_5 (Conv2D)           (None, 28, 28, 64)        18496

 max_pooling2d_2 (MaxPoolin  (None, 14, 14, 64)        0
 g2D)

 dropout_4 (Dropout)         (None, 14, 14, 64)        0

 flatten_2 (Flatten)         (None, 12544)             0

 dense_4 (Dense)             (None, 128)               1605760

 dropout_5 (Dropout)         (None, 128)               0

 dense_5 (Dense)             (None, 8)                 1032

=================================================================
Total params: 1625611 (6.20 MB)
Trainable params: 1625608 (6.20 MB)
Non-trainable params: 3 (16.00 Byte)
_________________________________________________________________
```

In [71]:
```python
# Compile and train model
model.compile(
    optimizer=tf.keras.optimizers.Adam(),
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    metrics=['accuracy'],
)

EPOCHS = 10
history = model.fit(
    train_spectrogram_ds,
    validation_data=val_spectrogram_ds,
    epochs=EPOCHS,
    callbacks=tf.keras.callbacks.EarlyStopping(verbose=1, patience=2),
)
```

```
Epoch 1/10
100/100 [==============================] - 6s 52ms/step - loss: 1.7203 - accuracy: 0.3841 - val_loss: 1.2518 - val_accuracy: 0.6094
Epoch 2/10
100/100 [==============================] - 5s 49ms/step - loss: 1.1439 - accuracy: 0.6069 - val_loss: 0.8594 - val_accuracy: 0.7526
Epoch 3/10
100/100 [==============================] - 5s 49ms/step - loss: 0.8809 - accuracy: 0.6825 - val_loss: 0.7203 - val_accuracy: 0.7982
Epoch 4/10
100/100 [==============================] - 5s 50ms/step - loss: 0.7291 - accuracy: 0.7408 - val_loss: 0.6295 - val_accuracy: 0.8060
Epoch 5/10
100/100 [==============================] - 5s 52ms/step - loss: 0.6169 - accuracy: 0.7852 - val_loss: 0.5811 - val_accuracy: 0.8216
Epoch 6/10
100/100 [==============================] - 5s 50ms/step - loss: 0.5297 - accuracy: 0.8086 - val_loss: 0.5562 - val_accuracy: 0.8099
Epoch 7/10
100/100 [==============================] - 5s 50ms/step - loss: 0.4892 - accuracy: 0.8302 - val_loss: 0.4962 - val_accuracy: 0.8464
Epoch 8/10
100/100 [==============================] - 5s 50ms/step - loss: 0.4383 - accuracy: 0.8500 - val_loss: 0.5121 - val_accuracy: 0.8372
Epoch 9/10
100/100 [==============================] - 5s 50ms/step - loss: 0.4143 - accuracy: 0.8550 - val_loss: 0.4659 - val_accuracy: 0.8385
Epoch 10/10
100/100 [==============================] - 5s 50ms/step - loss: 0.3627 - accuracy: 0.8703 - val_loss: 0.4668 - val_accuracy: 0.8438
```

In [72]:
```python
# Plot training and validation curves
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_accuracy']))
plt.legend(['accuracy', 'val_accuracy'])
plt.ylim([0, 100])
plt.xlabel('Epoch')
plt.ylabel('Accuracy [%]')
```

Out[72]: `Text(0, 0.5, 'Accuracy [%]')`



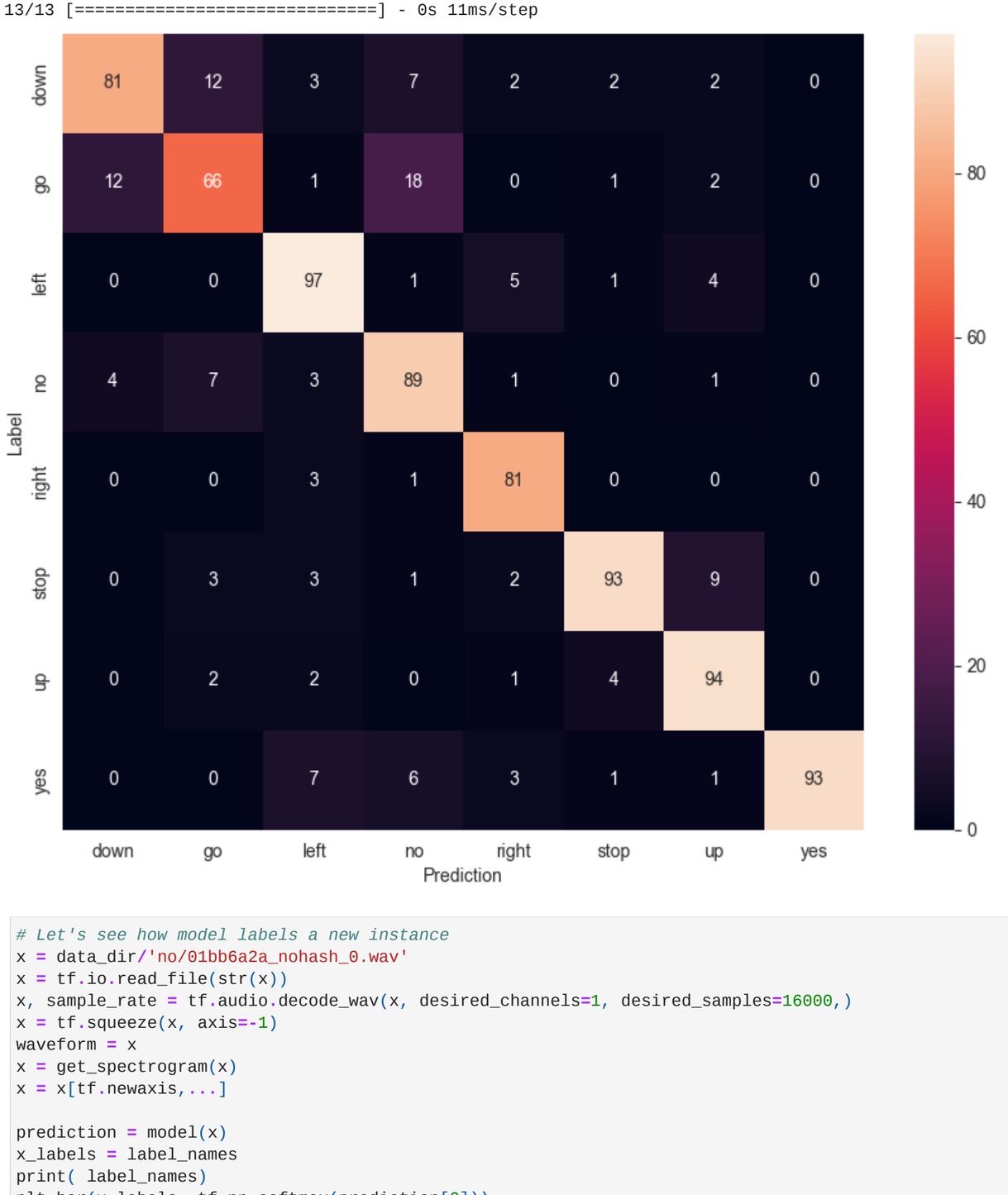
In [73]:
```python
# Evaluate Model
model.evaluate(test_spectrogram_ds, return_dict=True)
```

```
13/13 [==============================] - 1s 27ms/step - loss: 0.4968 - accuracy: 0.8341
```

Out[73]: `{'loss': 0.4967551529407501, 'accuracy': 0.8341346383094788}`

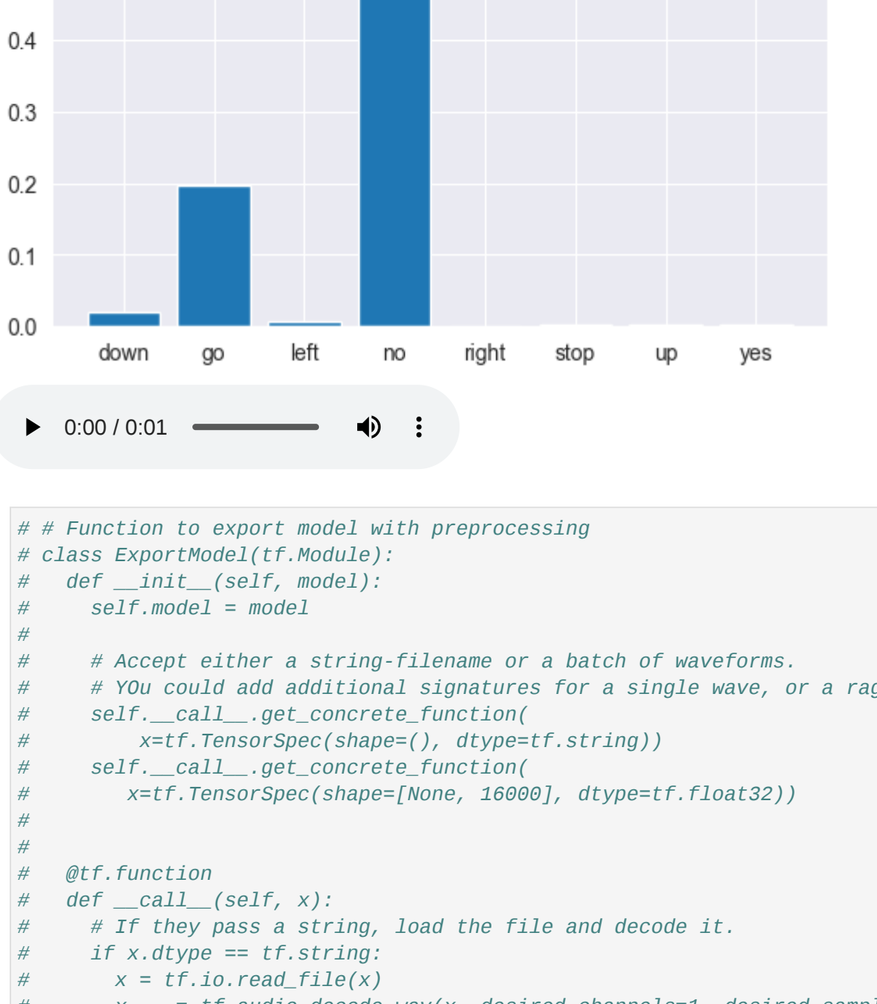
In [74]:
```python
# Create Confusion Matrix
y_pred = model.predict(test_spectrogram_ds)
y_pred = tf.argmax(y_pred, axis=1)

y_true = tf.concat(list(test_spectrogram_ds.map(lambda s,lab: lab)), axis=0)

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

```
13/13 [==============================] - 0s 11ms/step
```



In [86]:
```python
# Let's see how model labels a new instance
x = data_dir/'no/01bb6a2a_nohash_0.wav'
x = tf.io.read_file(str(x))
x, sample_rate = tf.audio.decode_wav(x, desired_channels=1, desired_samples=16000,)
x = tf.squeeze(x, axis=-1)
waveform = x
x = get_spectrogram(x)
x = x[tf.newaxis,...]

prediction = model(x)
x_labels = label_names
print( label_names)
plt.bar(x_labels, tf.nn.softmax(prediction[0]))
plt.title(commands[label_pred[0]])
plt.show()

display.display(display.Audio(waveform, rate=16000))
```

```
['down' 'go' 'left' 'no' 'right' 'stop' 'up' 'yes']
```



0:00 / 0:01

In [76]:
```python
# # Function to export model with preprocessing
# class ExportModel(tf.Module):
#   def __init__(self, model):
#     self.model = model
#
#     # Accept either a string-filename or a batch of waveforms.
#     # YOu could add additional signatures for a single wave, or a ragged-batch.
#     self.__call__.get_concrete_function(
#         x=tf.TensorSpec(shape=(), dtype=tf.string))
#     self.__call__.get_concrete_function(
#        x=tf.TensorSpec(shape=[None, 16000], dtype=tf.float32))
#
#
#   @tf.function
#   def __call__(self, x):
#     # If they pass a string, load the file and decode it.
#     if x.dtype == tf.string:
#       x = tf.io.read_file(x)
#       x, _ = tf.audio.decode_wav(x, desired_channels=1, desired_samples=16000,)
#       x = tf.squeeze(x, axis=-1)
#       x = x[tf.newaxis, :]
#
#     x = get_spectrogram(x)
#     result = self.model(x, training=False)
#
#     class_ids = tf.argmax(result, axis=-1)
#     class_names = tf.gather(label_names, class_ids)
#     return {'predictions':result,
#             'class_ids': class_ids,
#             'class_names': class_names}
```

In [77]:
```python
# export = ExportModel(model)
# export(tf.constant(str(data_dir/'no/01bb6a2a_nohash_0.wav')))
```

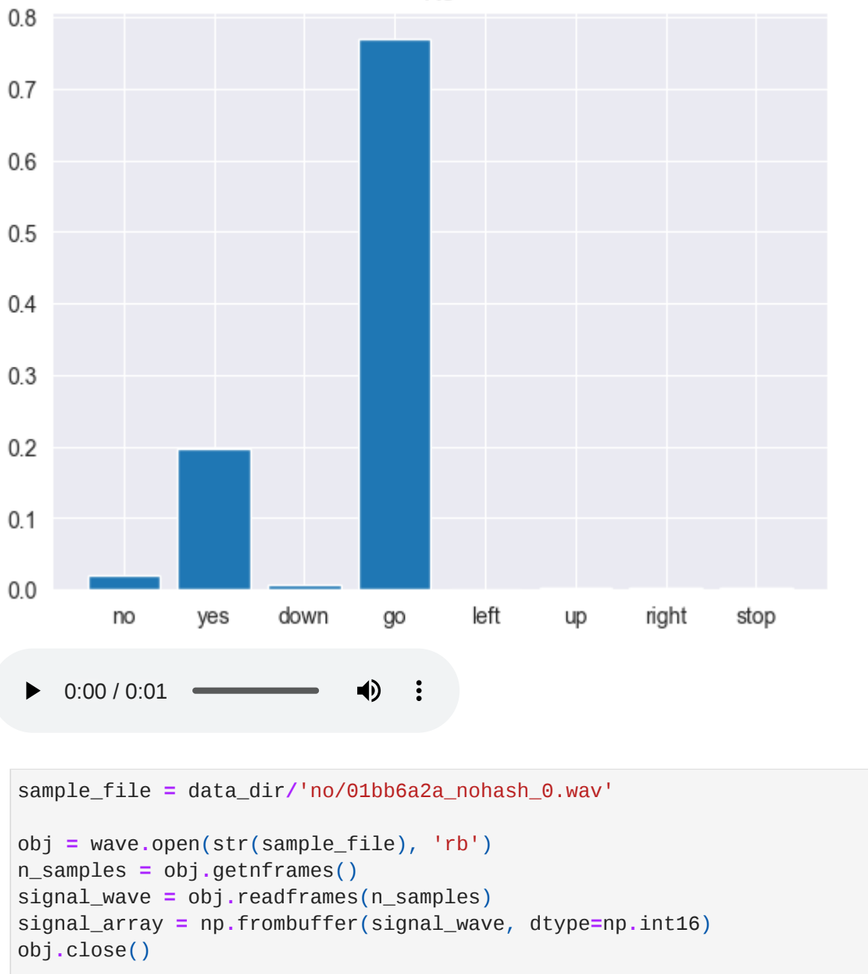
In [78]:
```python
# tf.saved_model.save(export, "saved")
# imported = tf.saved_model.load("saved")
# imported(waveform[tf.newaxis, :])
```

In [79]:
```python
model.save('saved_model')
```

```
INFO:tensorflow:Assets written to: saved_model\assets
INFO:tensorflow:Assets written to: saved_model\assets
```

In [80]:
```python
def zip_directory(directory_path, output_path):
    with zipfile.ZipFile(output_path, 'w', zipfile.ZIP_DEFLATED) as zipf:
        for root, _, files in os.walk(directory_path):
            for file in files:
                file_path = os.path.join(root, file)
                arcname = os.path.relpath(file_path, directory_path)
                zipf.write(file_path, arcname)

directory_to_zip = "./saved_model"
output_zip_file = "./saved_model.zip"

zip_directory(directory_to_zip, output_zip_file)
```

In [81]:
```python
loaded_model = models.load_model("saved_model")

prediction = loaded_model(x)
x_labels = ['no', 'yes', 'down', 'go', 'left', 'up', 'right', 'stop']
plt.bar(x_labels, tf.nn.softmax(prediction[0]))
plt.title('No')
plt.show()

display.display(display.Audio(waveform, rate=16000))
```



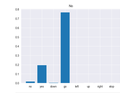
0:00 / 0:01

In [82]:
```python
sample_file = data_dir/'no/01bb6a2a_nohash_0.wav'

obj = wave.open(str(sample_file), 'rb')
n_samples = obj.getnframes()
signal_wave = obj.readframes(n_samples)
signal_array = np.frombuffer(signal_wave, dtype=np.int16)
obj.close()

print(signal_array.shape)
```

```
(16000,)
```

In [83]:
```python
waveform = signal_array / 32768
waveform = tf.convert_to_tensor(waveform, dtype=tf.float32)
spec = get_spectrogram(waveform)
spec = tf.expand_dims(spec, 0)
prediction = loaded_model(spec)
print(prediction)

label_pred = np.argmax(prediction, axis=1)
print(commands[label_pred[0]])
plt.bar(x_labels, tf.nn.softmax(prediction[0]))
plt.title("No")
plt.show()
```

```
tf.Tensor(
[[-0.05990982  2.2682161  -1.2018216   3.627874   -3.2565203  -2.4670265
  -2.0012596  -1.9084055 ]], shape=(1, 8), dtype=float32)
no
```



In [20]: