In [1]:
```
# TensorFlow and tf.keras
import tensorflow as tf
from tensorflow.keras import Input, layers
from tensorflow.keras import models
from tensorflow.keras.layers.experimental import preprocessing
from tensorflow.lite.experimental.microfrontend.python.ops import audio_mi
print(tf.__version__)

# Helper libraries
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from tqdm.notebook import tqdm
# from tqdm import tqdm # replace with this if moving out of notebook

import os
import pathlib

from datetime import datetime as dt

from IPython import display
```

```
2.11.0
```

In [2]:
```
# Set seed for experiment reproducibility
seed = 42
tf.random.set_seed(seed)
np.random.seed(seed)
```

In [3]:
```
i16min = -2**15
i16max = 2**15-1
fsamp = 16000
wave_length_ms = 1000
wave_length_samps = int(wave_length_ms*fsamp/1000)
window_size_ms=60
window_step_ms=40
num_filters = 32
use_microfrontend = True
dataset = 'mini-speech'
# dataset = 'full-speech-ds' # use the full speech commands as a pre-built
# dataset = 'full-speech-files' # use the full speech commands stored as f

silence_str = "_silence"
unknown_str = "_unknown"
EPOCHS = 25 #change it if you want
```

##To import commands from google database

In [4]:
```
if dataset == 'mini-speech':
  data_dir = pathlib.Path('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/d
#           extract=True, cache_dir='.', cache_subdir='data')
#   # commands = np.array(tf.io.gfile.listdir(str(data_dir))) # if you want
#   # commands = commands[commands != 'README.md']
# elif dataset == 'full-speech-files':
#   # data_dir = '/dfs/org/Holleman-Coursework/data/speech_dataset'
#   data_dir = pathlib.Path(os.path.join(os.getenv("HOME"), 'data/speech_co

# elif dataset == 'full-speech-ds':
#     raise RuntimeError("full-speech-ds is not really supported yet")
```

In [5]:
```
print(os.path.abspath(data_dir))
```

```
C:\Users\SerDude\Desktop\master\ML_materials\VScode\Project_2\mini_speech
_commands
```

In [6]:
```
commands = ['Active', 'stop']  #change to the new commands
label_list = commands.copy()
label_list.insert(0, silence_str)
label_list.insert(1, unknown_str)
print('label_list:', label_list)
```

```
label_list: ['_silence', '_unknown', 'Active', 'stop']
```

In [7]:
```
if dataset == 'mini-speech' or dataset == 'full-speech-files':
    filenames = tf.io.gfile.glob('data/'+str(data_dir) + '/*/*.wav')
    # with the next commented-out line, you can choose only files for words
    # filenames = tf.concat([tf.io.gfile.glob(str(data_dir) + '/' + cmd +
    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: 9782
Example file tensor: tf.Tensor(b'data\\mini_speech_commands\\Active\\Acti
ve_Sample25_Noise0_Multiplier2.wav', shape=(), dtype=string)
```

In [8]:

```
for i in range(100):
    print(filenames[i].numpy().decode('utf8'))
```

```
data\mini_speech_commands\Active\Active_Sample25_Noise0_Multiplier2.wav
data\mini_speech_commands\down\833a0279_nohash_0.wav
data\mini_speech_commands\yes\e7ea8b76_nohash_3.wav
data\mini_speech_commands\Active\Active_Sample60_RShift2_0.wav
data\mini_speech_commands\stop\51055bda_nohash_0.wav
data\mini_speech_commands\Active\Active_Sample10_RShift2_1.wav
data\mini_speech_commands\stop\15c563d7_nohash_2.wav
data\mini_speech_commands\go\9a69672b_nohash_1.wav
data\mini_speech_commands\no\ad1429cf_nohash_0.wav
data\mini_speech_commands\right\51995cea_nohash_0.wav
data\mini_speech_commands\up\dbb40d24_nohash_4.wav
data\mini_speech_commands\right\a243fcc2_nohash_0.wav
data\mini_speech_commands\up\d0faf7e4_nohash_3.wav
data\mini_speech_commands\right\cb2929ce_nohash_3.wav
data\mini_speech_commands\Active\Active_Pitch_Shift_1_55.wav
data\mini_speech_commands\left\a8ee11c7_nohash_0.wav
data\mini_speech_commands\Active\Active_Sample46_LShift2_1.wav
data\mini_speech_commands\left\cc554de3_nohash_0.wav
data\mini_speech_commands\stop\4a1e736b_nohash_1.wav
```

In [9]:
```
if dataset == 'mini-speech':
  print('Using mini-speech')
  num_train_files = int(0.8*num_samples)
  num_val_files = int(0.1*num_samples)
  num_test_files = num_samples - num_train_files - num_val_files
  train_files = filenames[:num_train_files]
  val_files = filenames[num_train_files: num_train_files + num_val_files]
  test_files = filenames[-num_test_files:]
elif dataset == 'full-speech-files':
  # the full speech-commands set lists which files are to be used
  # as test and validation data; train with everything else
  fname_val_files = os.path.join(data_dir, 'validation_list.txt')
  with open(fname_val_files) as fpi_val:
    val_files = fpi_val.read().splitlines()
  # validation_list.txt only lists partial paths
  val_files = [os.path.join(data_dir, fn) for fn in val_files]
  fname_test_files = os.path.join(data_dir, 'testing_list.txt')

  with open(fname_test_files) as fpi_tst:
    test_files = fpi_tst.read().splitlines()
  # testing_list.txt only lists partial paths
  test_files = [os.path.join(data_dir, fn).rstrip() for fn in test_files]

  # convert the TF tensor filenames into an array of strings so we can use
  train_files = [f.decode('utf8') for f in filenames.numpy()]
  # don't train with the _background_noise_ files; exclude when directory n
  train_files = [f for f in train_files if f.split('/')[-2][0] != '_']
  # validation and test files are listed explicitly in *_list.txt; train wi
  train_files = list(set(train_files) - set(test_files) - set(val_files))
  # now convert back into a TF tensor so we can use the tf.dataset pipeline
  train_files = tf.constant(train_files)
  print("full-speech-files is in progress.  Good luck!")
elif dataset == 'full-speech-ds':
    print("Using full-speech-ds. This is in progress.  Good luck!")
else:
  raise ValueError("dataset must be either full-speech-files, full-speech-d
print('Training set size', len(train_files))
print('Validation set size', len(val_files))
print('Test set size', len(test_files))
```

```
Using mini-speech
Training set size 7825
Validation set size 978
Test set size 979
```

In [10]:
```
def decode_audio(audio_binary):
  audio, _ = tf.audio.decode_wav(audio_binary)
  return tf.squeeze(audio, axis=-1)
```

In [11]:
```
# @tf.function
def get_label(file_path):
  parts = tf.strings.split(file_path, os.path.sep)
  in_set = tf.reduce_any(parts[-2] == label_list)
  label = tf.cond(in_set, lambda: parts[-2], lambda: tf.constant(unknown_s
  # print(f"parts[-2] = {parts[-2]}, in_set = {in_set}, label = {label}")
  # Note: You'll use indexing here instead of tuple unpacking to enable th
  # to work in a TensorFlow graph.
  return  label # parts[-2]
```

In [12]:
```
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
```

In [13]:
```
def get_spectrogram(waveform):
  # Concatenate audio with padding so that all audio clips will be of the
  # same length (16000 samples)
  zero_padding = tf.zeros([wave_length_samps] - tf.shape(waveform), dtype=
  waveform = tf.cast(0.5*waveform*(i16max-i16min), tf.int16)  # scale floa
  equal_length = tf.concat([waveform, zero_padding], 0)
  ## Make sure these labels correspond to those used in micro_features_mic
  spectrogram = frontend_op.audio_microfrontend(equal_length, sample_rate=
                                    window_size=window_size_ms, window_ste
  return spectrogram
```

In [14]:
```
def create_silence_dataset(num_waves, samples_per_wave, rms_noise_range=[0
    # create num_waves waveforms of white gaussian noise, with rms level di
    # to act as the "silence" dataset
    rng = np.random.default_rng()
    rms_noise_levels = rng.uniform(low=rms_noise_range[0], high=rms_noise_
    rand_waves = np.zeros((num_waves, samples_per_wave), dtype=np.float32)
    for i in range(num_waves):
        rand_waves[i,:] = rms_noise_levels[i]*rng.standard_normal(samples_
    labels = [silent_label]*num_waves
    return tf.data.Dataset.from_tensor_slices((rand_waves, labels))
```

In [15]:

```
def wavds2specds(waveform_ds, verbose=True):
  wav, label = next(waveform_ds.as_numpy_iterator())
  one_spec = get_spectrogram(wav)
  one_spec = tf.expand_dims(one_spec, axis=0)  # add a 'batch' dimension a
  one_spec = tf.expand_dims(one_spec, axis=-1) # add a singleton 'channel'

  num_waves = 0 # count the waveforms so we can allocate the memory
  for wav, label in waveform_ds:
    num_waves += 1
  print(f"About to create spectrograms from {num_waves} waves")
  spec_shape = (num_waves,) + one_spec.shape[1:]
  spec_grams = np.nan * np.zeros(spec_shape)  # allocate memory
  labels = np.nan * np.zeros(num_waves)
  idx = 0
  for wav, label in waveform_ds:
    if verbose and idx % 250 == 0:
      print(f"\r {idx} wavs processed", end='')
    spectrogram = get_spectrogram(wav)
    # TF conv layer expect inputs structured as 4D (batch_size, height, wi
    # the microfrontend returns 2D tensors (freq, time), so we need to
    spectrogram = tf.expand_dims(spectrogram, axis=0)  # add a 'batch' dim
    spectrogram = tf.expand_dims(spectrogram, axis=-1) # add a singleton '
    spec_grams[idx, ...] = spectrogram
    new_label = label.numpy().decode('utf8')
    new_label_id = np.argmax(new_label == np.array(label_list))
    labels[idx] = new_label_id # for numeric labels
    # labels.append(new_label) # for string labels
    idx += 1
  labels = np.array(labels, dtype=int)
  output_ds = tf.data.Dataset.from_tensor_slices((spec_grams, labels))
  return output_ds
```

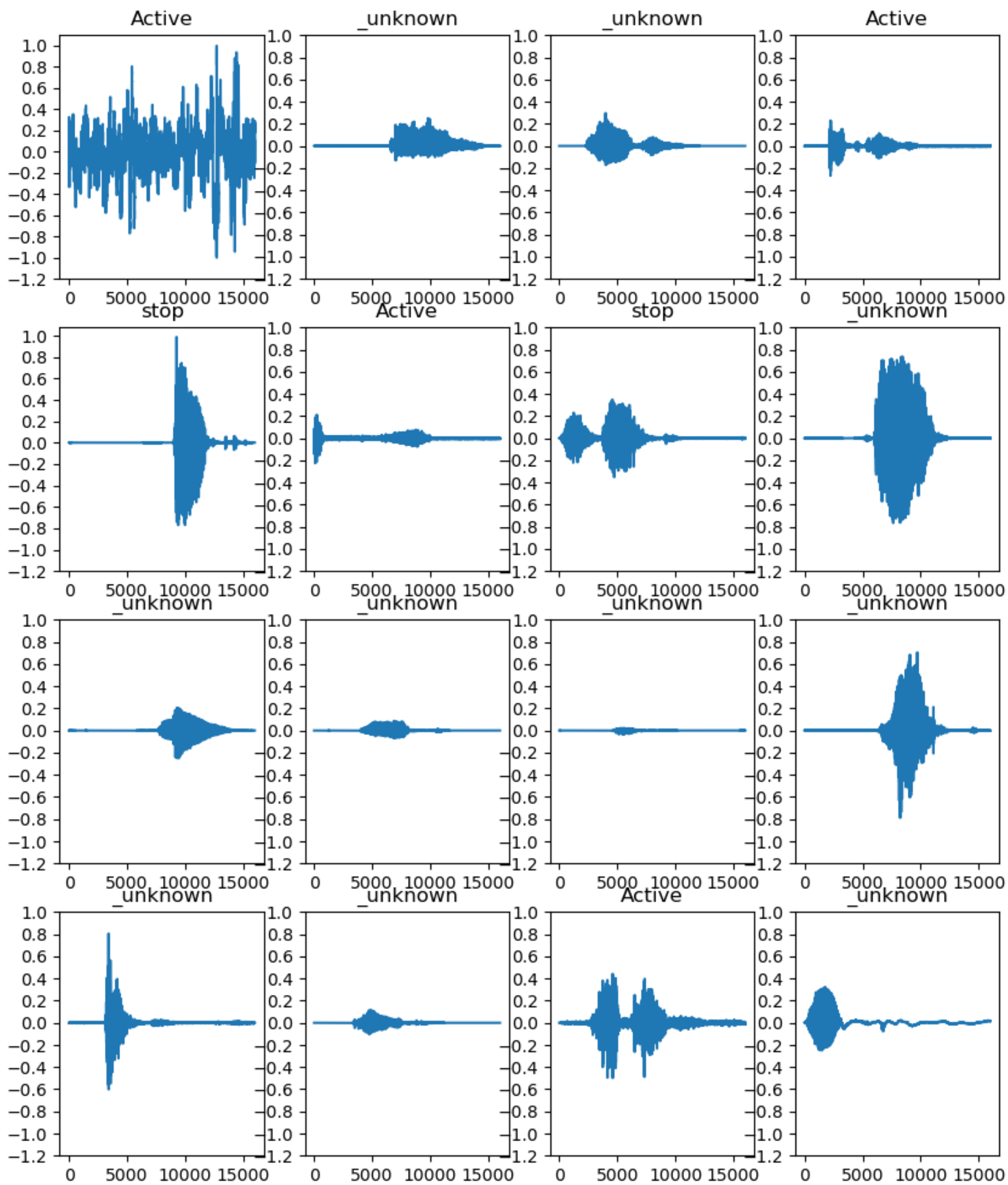
In [16]:

```
AUTOTUNE = tf.data.experimental.AUTOTUNE
num_train_files = len(train_files)
files_ds = tf.data.Dataset.from_tensor_slices(train_files)
waveform_ds = files_ds.map(get_waveform_and_label, num_parallel_calls=AUTO
train_ds = wavds2specds(waveform_ds)
```

```
WARNING:tensorflow:From C:\Users\SerDude\anaconda3\lib\site-packages\tens
orflow\python\autograph\pyct\static_analysis\liveness.py:83: Analyzer.lam
ba_check (from tensorflow.python.autograph.pyct.static_analysis.liveness)
is deprecated and will be removed after 2023-09-23.
Instructions for updating:
Lambda fuctions will be no more assumed to be used in the statement where
they are used, or at least in the same block. https://github.com/tensorfl
ow/tensorflow/issues/56089 (https://github.com/tensorflow/tensorflow/issu
es/56089)
About to create spectrograms from 7825 waves
 7750 wavs processed
```

In [17]:

```
rows = 4
cols = 4
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()
```



In [18]:

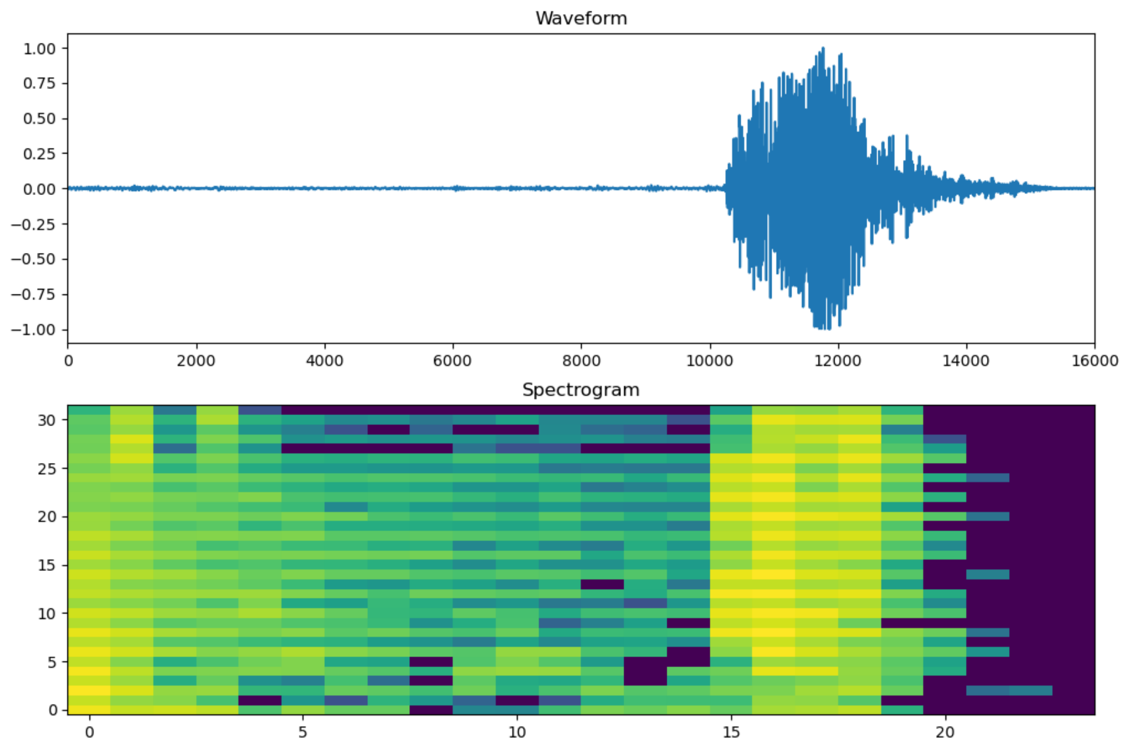
```
for waveform, label in waveform_ds.take(300):
  label = label.numpy().decode('utf-8')
  spectrogram = get_spectrogram(waveform)

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: (24, 32)
Audio playback
```

0:00 / 0:01

In [19]:

```
def plot_spectrogram(spectrogram, ax):
  # transpose so that the time is
  # represented in the x-axis (columns).
  freq_bins = spectrogram.shape[1]
  time_dur = spectrogram.shape[0]
  X = np.arange(time_dur)
  Y = range(freq_bins)
  ax.pcolormesh(X, Y, spectrogram.T)

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()
spectrogram.numpy().shape
```

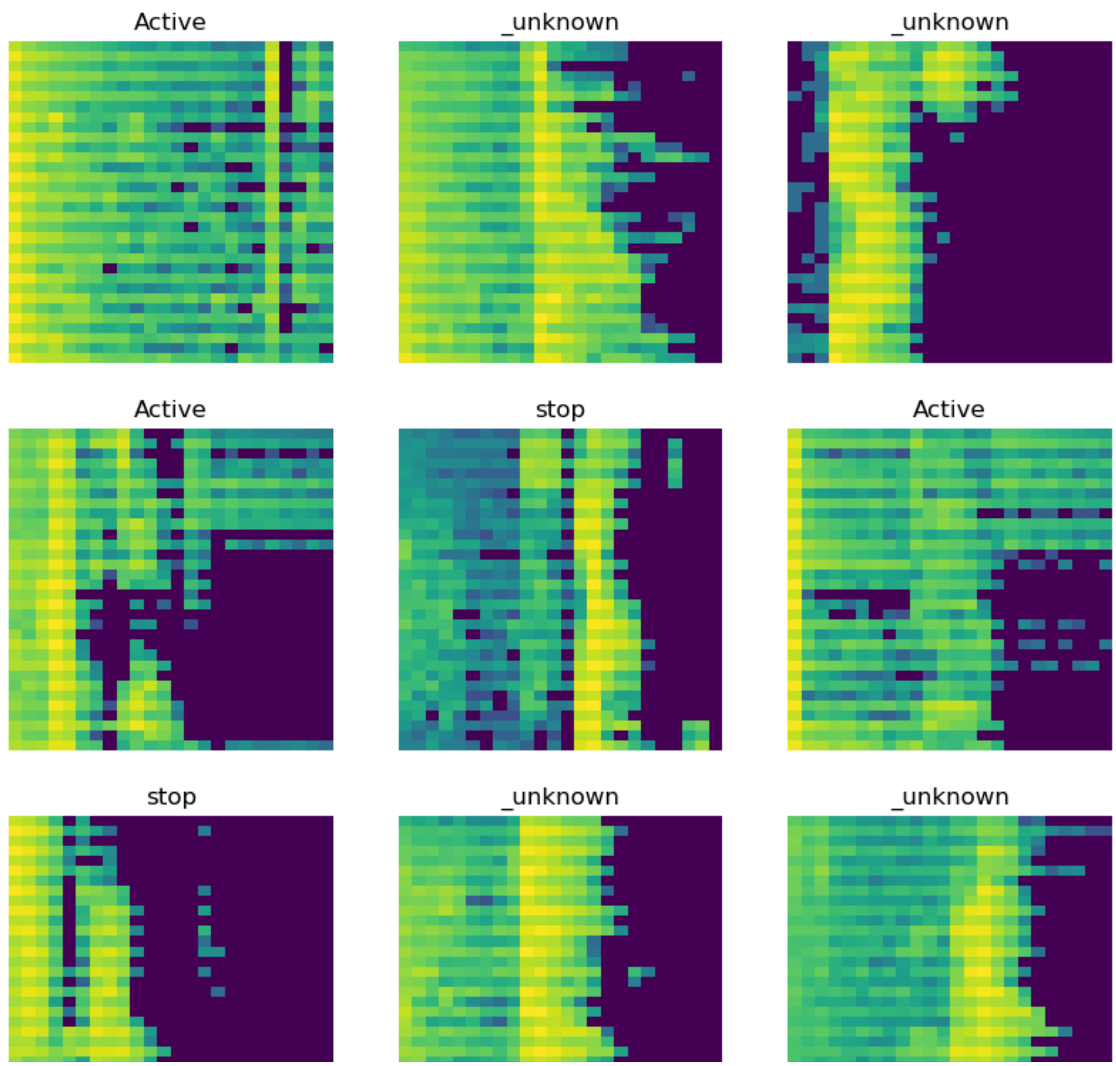


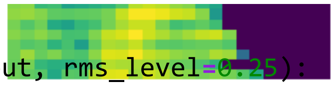
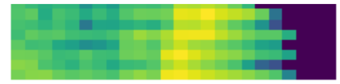
Out[19]: (24, 32)

In [20]:

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

plt.show()
```

```
C:\Users\SerDude\AppData\Local\Temp\ipykernel_34688\1859809011.py:10: DeprecationWarning: `np.int` is a deprecated alias for the builtin `int`. To silence this warning, use `int` by itself. Doing this will not modify any behavior and is safe. When replacing `np.int`, you may wish to use e.g. `np.int64` or `np.int32` to specify the precision. If you wish to review your current use, check the release note link for additional information.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations (https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations)
  ax.set_title(label_list[np.int(label_id)])
```



In [21]:
```
def copy_with_noise(ds_input, rms_level=0.25):
  rng = tf.random.Generator.from_seed(1234)
  wave_shape = tf.constant((wave_length_samps,))
  def add_noise(waveform, label):
    noise = rms_level*rng.normal(shape=wave_shape)
    zero_padding = tf.zeros([wave_length_samps] - tf.shape(waveform), dtyp
    waveform = tf.concat([waveform, zero_padding], 0)
    noisy_wave = waveform + noise
    return noisy_wave, label

  return ds_input.map(add_noise)
```

In [22]:
```
count = 0
for w,l in waveform_ds:
  if w.shape != (16000,):
    print(f"element {count} has shape {w.shape}")
    break
  count += 1
print(count)
```

```
element 20 has shape (13654,)
20
```

In [23]:
```
def pad_16000(waveform, label):
    zero_padding = tf.zeros([wave_length_samps] - tf.shape(waveform), dtyp
    waveform = tf.concat([waveform, zero_padding], 0)
    return waveform, label
```

In [24]:
```
def count_labels(dataset):
    counts = {}
    for _, lbl in dataset:
        if lbl.dtype == tf.string:
            label = lbl.numpy().decode('utf-8')
        else:
            label = lbl.numpy()
        if label in counts:
            counts[label] += 1
        else:
            counts[label] = 1
    return counts
```

In [25]:

```
# Collect what we did to generate the training dataset into a
# function, so we can repeat with the validation and test sets.
def preprocess_dataset(files, num_silent=None, noisy_reps_of_known=None):
  # if noisy_reps_of_known is not None, it should be a list of rms noise le
  # For every target word in the data set, 1 copy will be created with each
  # of noise added to it.  So [0.1, 0.2] will add 2x noisy copies of the ta
  if num_silent is None:
    num_silent = int(0.2*len(files))+1
  print(f"Processing {len(files)} files")
  files_ds = tf.data.Dataset.from_tensor_slices(files)
  waveform_ds = files_ds.map(get_waveform_and_label)
  if noisy_reps_of_known is not None:
    # create a few copies of only the target words to balance the distribut
    # create a tmp dataset with only the target words
    ds_only_cmds = waveform_ds.filter(lambda w,l: tf.reduce_any(l == comman
    for noise_level in noisy_reps_of_known:
       waveform_ds = waveform_ds.concatenate(copy_with_noise(ds_only_cmds,
  if num_silent > 0:
    silent_wave_ds = create_silence_dataset(num_silent, wave_length_samps,
                                            rms_noise_range=[0.01,0.2],
                                            silent_label=silence_str)

    waveform_ds = waveform_ds.concatenate(silent_wave_ds)
  print(f"Added {num_silent} silent wavs and ?? noisy wavs")
  num_waves = 0
  output_ds = wavds2specds(waveform_ds)
  return output_ds
```

In [26]:

```
print(f"We have {len(train_files)}/{len(val_files)}/{len(test_files)} trai
```

```
We have 7825/978/979 training/validation/test files
```

In [27]:

```
train_ds = preprocess_dataset(train_files,num_silent=int(len(train_files)*(
val_ds = preprocess_dataset(val_files)
test_ds = preprocess_dataset(test_files)
```

```
Processing 7825 files
Added 3912 silent wavs and ?? noisy wavs
About to create spectrograms from 22802 waves
 22750 wavs processedProcessing 978 files
Added 196 silent wavs and ?? noisy wavs
About to create spectrograms from 1174 waves
 1000 wavs processedProcessing 979 files
Added 196 silent wavs and ?? noisy wavs
About to create spectrograms from 1175 waves
 1000 wavs processed
```

In [28]:
```
print("training data set")
print(count_labels(train_ds))
print("val_ds data set")
print(count_labels(val_ds))
print("test_ds data set")
print(count_labels(test_ds))
```

```
training data set
{2: 8472, 1: 5612, 3: 4806, 0: 3912}
val_ds data set
{1: 705, 2: 176, 3: 97, 0: 196}
test_ds data set
{1: 683, 2: 194, 3: 102, 0: 196}
```

In [29]:
```
train_ds = train_ds.shuffle(int(len(train_files)*1.2))
val_ds = val_ds.shuffle(int(len(val_files)*1.2))
test_ds = test_ds.shuffle(int(len(test_files)*1.2))
```

In [30]:
```
batch_size = 64
train_ds = train_ds.batch(batch_size)
val_ds = val_ds.batch(batch_size)
```

In [31]:
```
train_ds = train_ds.cache().prefetch(AUTOTUNE)
val_ds = val_ds.cache().prefetch(AUTOTUNE)
```

In [32]:
```
for spectrogram, _ in train_ds.take(1):
  spec1 = spectrogram
# take(1) takes 1 *batch*, so we have to select the first
# spectrogram from it, hence the [0]
print(f"Spectrogram shape {spec1[0].shape}")
print(f"ranges from {np.min(spec1)} to {np.max(spec1)}")   # min/max acros
```

```
Spectrogram shape (24, 32, 1)
ranges from 0.0 to 712.0
```

In [33]:
```
for spectrogram, _ in train_ds.take(1):
  # take(1) takes 1 *batch*, so we have to select the first
  # spectrogram from it, hence the [0]
  input_shape = spectrogram[0].shape
print('Input shape:', input_shape)
num_labels = len(label_list)
```

```
Input shape: (24, 32, 1)
```

In [34]:

```
print('Input shape:', input_shape)

model = models.Sequential([
    layers.Input(shape=input_shape),
    layers.Conv2D(20, 3, activation='relu'),
    layers.MaxPooling2D(name='pool2'),
    layers.DepthwiseConv2D(kernel_size=(3,3), padding='same'),
    layers.Conv2D(32, 1, activation='relu'),
    layers.MaxPooling2D(pool_size=(4,4)),
    layers.Dropout(0.2),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dropout(0.2),
    layers.Dense(128, activation='relu'),
    layers.Dropout(0.2),
    layers.Dense(num_labels),
], name="simple_cnn")

model.summary()
```

```
Input shape: (24, 32, 1)
Model: "simple_cnn"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 22, 30, 20)        200

 pool2 (MaxPooling2D)        (None, 11, 15, 20)        0

 depthwise_conv2d (Depthwise  (None, 11, 15, 20)       200
 Conv2D)

 conv2d_1 (Conv2D)           (None, 11, 15, 32)        672

 max_pooling2d (MaxPooling2D  (None, 2, 3, 32)         0
 )

 dropout (Dropout)           (None, 2, 3, 32)          0

 flatten (Flatten)           (None, 192)               0

 dense (Dense)               (None, 128)               24704

 dropout_1 (Dropout)         (None, 128)               0

 dense_1 (Dense)             (None, 128)               16512

 dropout_2 (Dropout)         (None, 128)               0

 dense_2 (Dense)             (None, 4)                 516

=================================================================
Total params: 42,804
Trainable params: 42,804
Non-trainable params: 0
```

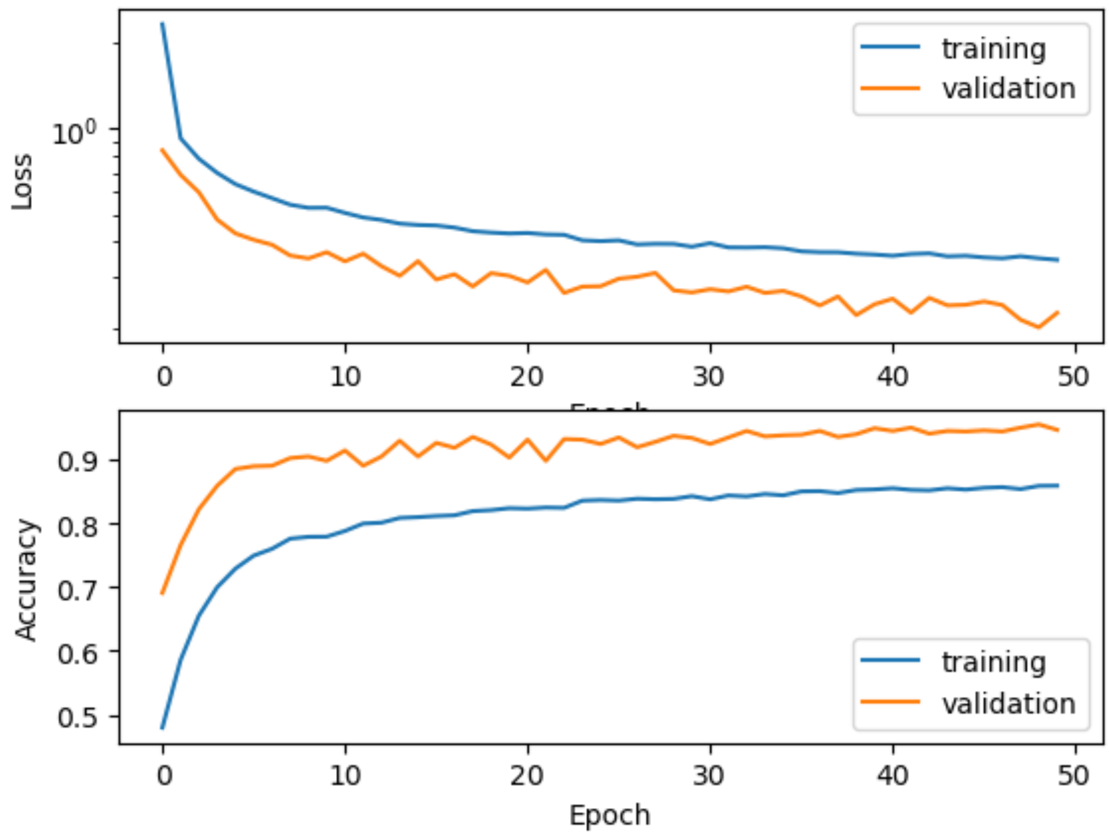
In [35]:
```
model.input.shape[0]
```

In [36]:
```
model.compile(
    optimizer=tf.keras.optimizers.Adam(),
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    metrics=['accuracy'],
)
callback=tf.keras.callbacks.EarlyStopping(
    monitor= 'accuracy',
    verbose=1,
    patience=10,
    restore_best_weights=True
)

history = model.fit(
    train_ds,
    validation_data=val_ds,
    #epochs=EPOCHS
    epochs=50,
    callbacks=[callback]
)
```

```
Epoch 1/50
357/357 [==============================] - 3s 8ms/step - loss: 2.3350 - a
ccuracy: 0.4797 - val_loss: 0.8407 - val_accuracy: 0.6908
Epoch 2/50
357/357 [==============================] - 2s 6ms/step - loss: 0.9265 - a
ccuracy: 0.5859 - val_loss: 0.6891 - val_accuracy: 0.7658
Epoch 3/50
357/357 [==============================] - 2s 6ms/step - loss: 0.7851 - a
ccuracy: 0.6552 - val_loss: 0.5984 - val_accuracy: 0.8220
Epoch 4/50
357/357 [==============================] - 2s 6ms/step - loss: 0.7002 - a
ccuracy: 0.6996 - val_loss: 0.4797 - val_accuracy: 0.8586
Epoch 5/50
357/357 [==============================] - 2s 6ms/step - loss: 0.6394 - a
ccuracy: 0.7288 - val_loss: 0.4290 - val_accuracy: 0.8850
Epoch 6/50
357/357 [==============================] - 2s 6ms/step - loss: 0.6017 - a
ccuracy: 0.7492 - val_loss: 0.4073 - val_accuracy: 0.8893
Epoch 7/50
```

In [37]:

```
metrics = history.history
plt.subplot(2,1,1)
plt.semilogy(history.epoch, metrics['loss'], metrics['val_loss'])
plt.legend(['training', 'validation'])
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.subplot(2,1,2)
plt.plot(history.epoch, metrics['accuracy'], metrics['val_accuracy'])
plt.legend(['training', 'validation'])
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.show()
```



In [38]:

```
date_str = dt.now().strftime("%d%b%Y_%H%M").lower()
model_file_name = f"keyword_model_{date_str}.h5"
print(f"Saving model to {model_file_name}")
model.save(model_file_name, overwrite=False)
```

```
Saving model to keyword_model_03may2023_2000.h5
```

In [39]:
```
# with open(model_file_name.split('.')[0] + '.txt', 'w') as fpo:
#     fpo.write(f"i16min             = {i16min           }\n")
#     fpo.write(f"i16max             = {i16max           }\n")
#     fpo.write(f"fsamp              = {fsamp            }\n")
#     fpo.write(f"wave_length_ms     = {wave_length_ms   }\n")
#     fpo.write(f"wave_length_samps = {wave_length_samps}\n")
#     fpo.write(f"window_size_ms     = {window_size_ms   }\n")
#     fpo.write(f"window_step_ms     = {window_step_ms   }\n")
#     fpo.write(f"num_filters        = {num_filters      }\n")
#     fpo.write(f"use_microfrontend = {use_microfrontend}\n")
#     fpo.write(f"label_list         = {label_list}\n")
#     fpo.write(f"spectrogram_shape = {spectrogram.numpy().shape}\n")
```

In [40]:
```
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)
test_labels
```

Out[40]:
```
array([1, 1, 1, ..., 1, 2, 3])
```

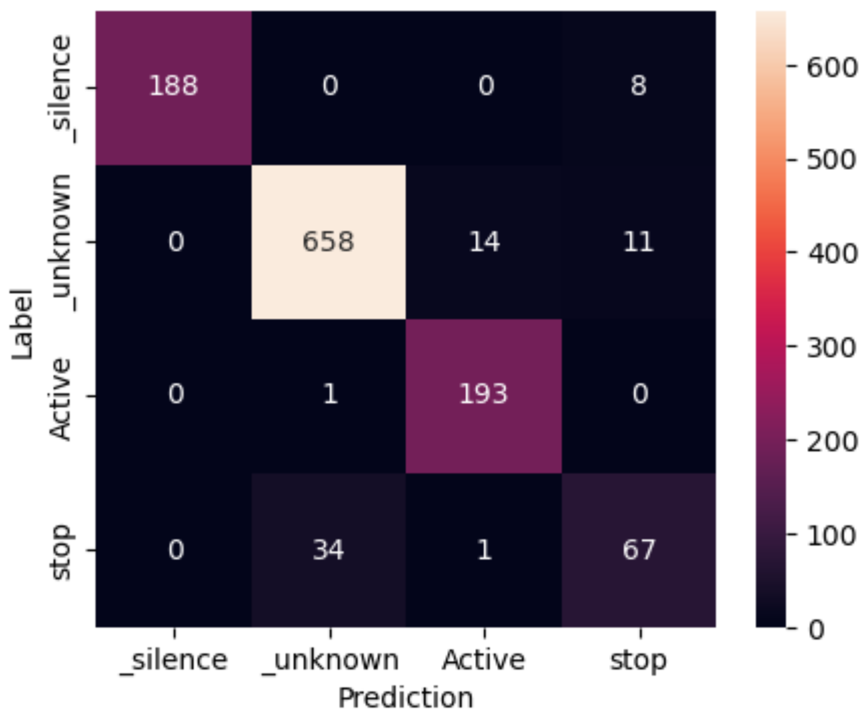
In [41]:
```
y_pred = np.argmax(model.predict(test_audio), axis=1)
y_true = test_labels

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

```
37/37 [==============================] - 0s 1ms/step
Test set accuracy: 94%
```

In [42]:

```
print("On test set:")
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%}')
confusion_mtx = tf.math.confusion_matrix(y_true, y_pred)
plt.figure(figsize=(5, 4))
sns.heatmap(confusion_mtx, xticklabels=label_list, yticklabels=label_list,
            annot=True, fmt='g')
#plt.gca().invert_yaxis() # flip so origin is at bottom left
plt.xlabel('Prediction')
plt.ylabel('Label')
plt.show()
```

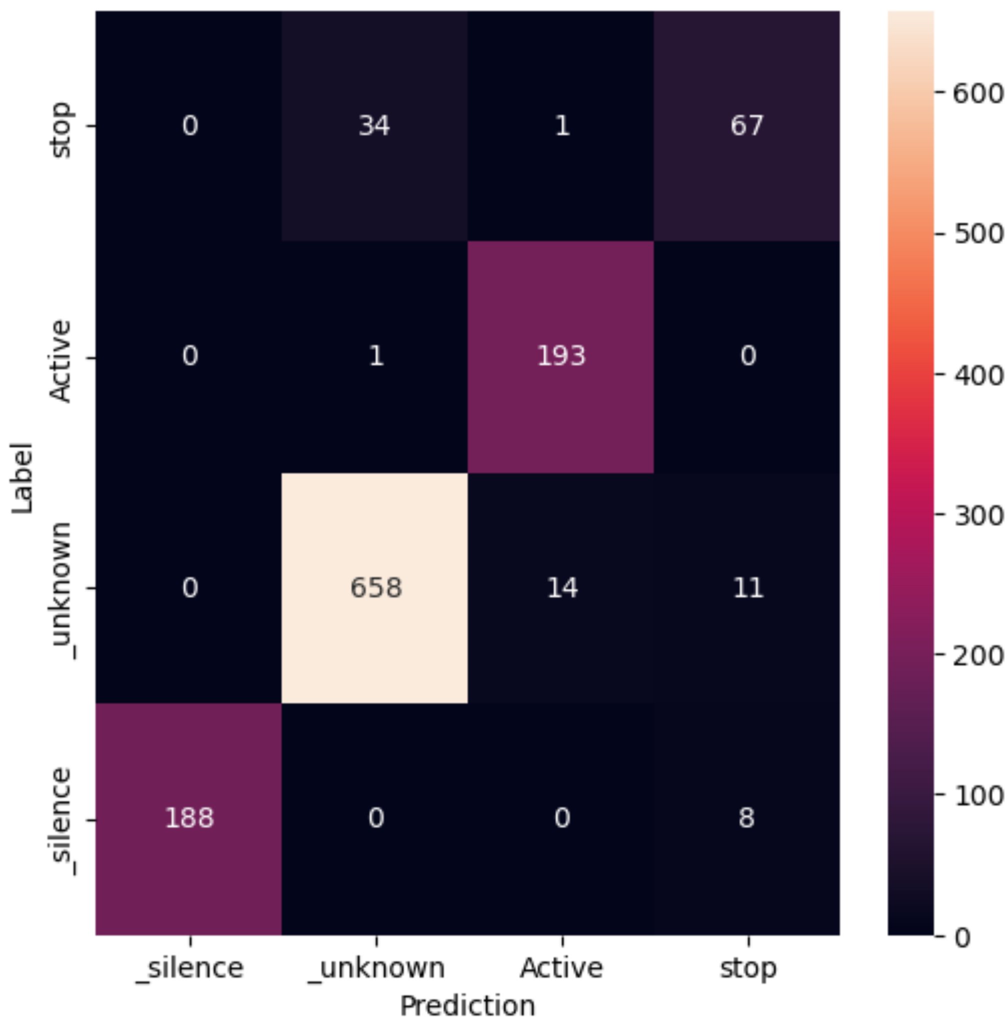
```
On test set:
37/37 [==============================] - 0s 1ms/step
Test set accuracy: 94%
```



In [43]:

```
print("On test set:")
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%}')
confusion_mtx = tf.math.confusion_matrix(y_true, y_pred)
plt.figure(figsize=(6, 6))
sns.heatmap(confusion_mtx, xticklabels=label_list, yticklabels=label_list,
            annot=True, fmt='g')
plt.gca().invert_yaxis() # flip so origin is at bottom left
plt.xlabel('Prediction')
plt.ylabel('Label')
plt.show()
```

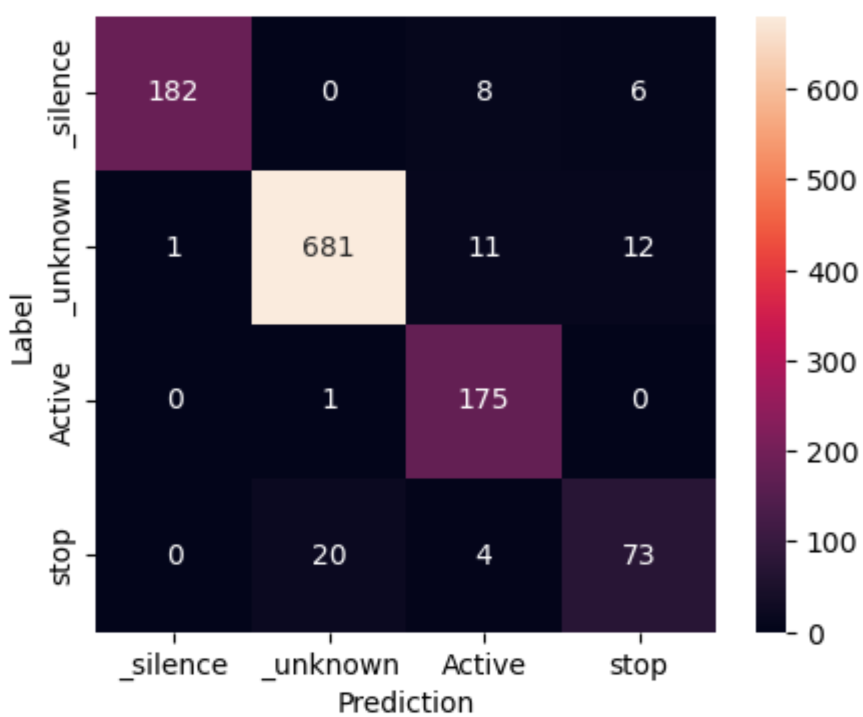
```
On test set:
37/37 [==============================] - 0s 1ms/step
Test set accuracy: 94%
```



In [44]:

```
dset = val_ds.unbatch()
print("On validation set:")

ds_audio = []
ds_labels = []

for audio, label in dset:
  ds_audio.append(audio.numpy())
  ds_labels.append(label.numpy())

ds_labels = np.array(ds_labels)
ds_audio = np.array(ds_audio)

model_out = model.predict(ds_audio)
y_pred = np.argmax(model_out, axis=1)
y_true = ds_labels

ds_acc = sum(y_pred == y_true) / len(y_true)
print(f'Data set accuracy: {ds_acc:.0%}')

confusion_mtx = tf.math.confusion_matrix(y_true, y_pred)
plt.figure(figsize=(5,4))
sns.heatmap(confusion_mtx, xticklabels=label_list, yticklabels=label_list,
            annot=True, fmt='g')
plt.xlabel('Prediction')
plt.ylabel('Label')
plt.show()
```

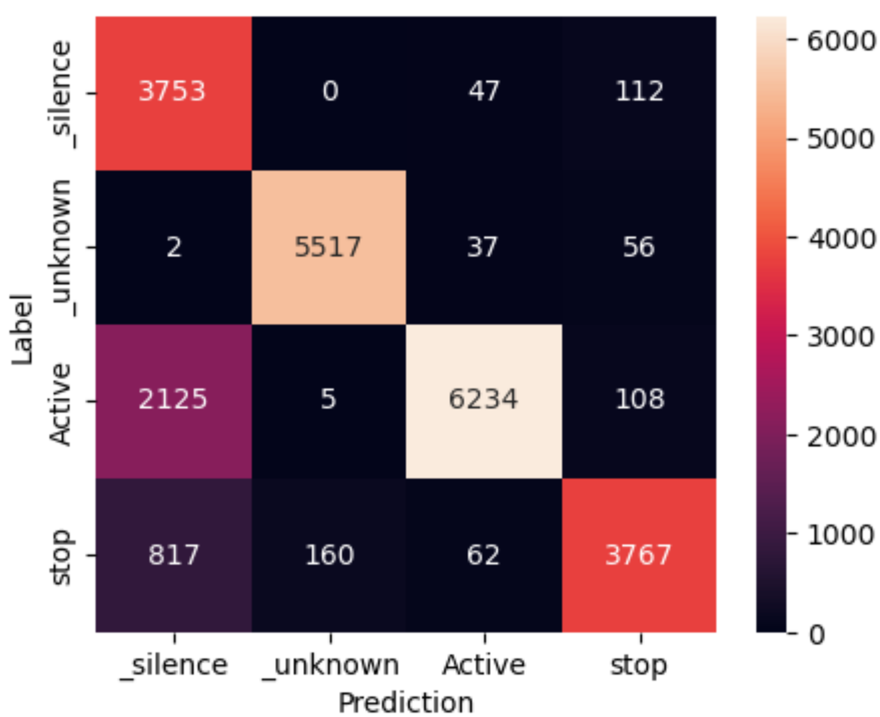
```
On validation set:
37/37 [==============================] - 0s 1ms/step
Data set accuracy: 95%
```



In [45]:

```
dset = train_ds.unbatch()
print("On training set:")

ds_audio = []
ds_labels = []

for audio, label in dset:
  ds_audio.append(audio.numpy())
  ds_labels.append(label.numpy())

ds_labels = np.array(ds_labels)
ds_audio = np.array(ds_audio)

model_out = model.predict(ds_audio)
y_pred = np.argmax(model_out, axis=1)
y_true = ds_labels

ds_acc = sum(y_pred == y_true) / len(y_true)
print(f'Data set accuracy: {ds_acc:.0%}')

confusion_mtx = tf.math.confusion_matrix(y_true, y_pred)
plt.figure(figsize=(5,4))
sns.heatmap(confusion_mtx, xticklabels=label_list, yticklabels=label_list,
            annot=True, fmt='g')
plt.xlabel('Prediction')
plt.ylabel('Label')
plt.show()
```

```
On training set:
713/713 [==============================] - 1s 1ms/step
Data set accuracy: 85%
```

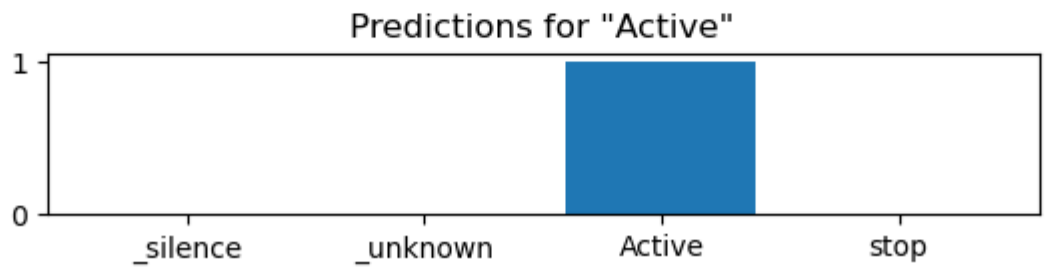


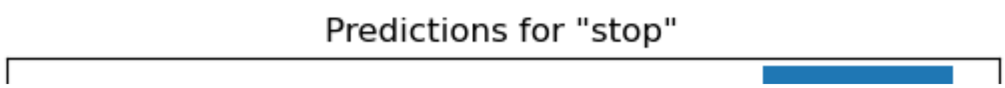
In [ ]:

In [46]:

```
sample_files = [data_dir/'Active/Active_Sample75_Noise0_Multiplier3.wav',
                data_dir/'stop/01bb6a2a_nohash_0.wav',
                data_dir/'yes/8a28231e_nohash_0.wav']
fstr_list = [f"data\\"+str(f) for f in sample_files]
print(fstr_list)
sample_ds = preprocess_dataset(fstr_list, num_silent=1)
count = 1
for spectrogram, label in sample_ds.batch(1):
  prediction = model(spectrogram)
  plt.subplot(len(sample_files)+1, 1, count)
  plt.bar(label_list, tf.nn.softmax(prediction[0]))
  plt.title(f'Predictions for "{label_list[label[0]]}"')
  plt.show()
  count += 1
```

```
['data\\mini_speech_commands\\Active\\Active_Sample75_Noise0_Multiplier3.
wav', 'data\\mini_speech_commands\\stop\\01bb6a2a_nohash_0.wav', 'data\\m
ini_speech_commands\\yes\\8a28231e_nohash_0.wav']
Processing 3 files
Added 1 silent wavs and ?? noisy wavs
About to create spectrograms from 4 waves
 0 wavs processed
```





In [47]:

```
converter = tf.lite.TFLiteConverter.from_keras_model(model)
converter.optimizations = [tf.lite.Optimize.DEFAULT]
```

In [48]:

```
num_calibration_steps = 10
ds_iter = val_ds.unbatch().batch(1).as_numpy_iterator()
def representative_dataset_gen():
  for _ in range(num_calibration_steps):
    next_input = next(ds_iter)[0]
    next_input = next_input.astype(np.float32)  # (DIFF_FROM_LECTURE)
    yield [next_input]
```

In [49]:

```
converter.optimizations = [tf.lite.Optimize.DEFAULT]
converter.representative_dataset = representative_dataset_gen
converter.target_spec.supported_ops = [tf.lite.OpsSet.TFLITE_BUILTINS_INT8
converter.inference_input_type = tf.int8  # or tf.uint8; should match dat_
converter.inference_output_type = tf.int8  # or tf.uint8
```

In [50]:
```
tflite_quant_model = converter.convert()
```

```
WARNING:absl:Found untraced functions such as _jit_compiled_convolution_o
p, _jit_compiled_convolution_op, _jit_compiled_convolution_op, _update_st
ep_xla while saving (showing 4 of 4). These functions will not be directl
y callable after loading.

INFO:tensorflow:Assets written to: C:\Users\SerDude\AppData\Local\Temp\tm
ph58s7zlx\assets

INFO:tensorflow:Assets written to: C:\Users\SerDude\AppData\Local\Temp\tm
ph58s7zlx\assets
C:\Users\SerDude\anaconda3\lib\site-packages\tensorflow\lite\python\conve
rt.py:765: UserWarning: Statistics for quantized inputs were expected, bu
t not specified; continuing anyway.
  warnings.warn("Statistics for quantized inputs were expected, but not "
```

In [51]:
```
fname = 'keyword_model_test.tflite'
with open(fname, "wb") as fpo:
  num_bytes_written = fpo.write(tflite_quant_model)
print(f"Wrote {num_bytes_written} / {len(tflite_quant_model)} bytes to tfl
```

```
Wrote 50408 / 50408 bytes to tflite file
```

In [ ]: