




Riconoscimento audio semplice: riconoscimento delle parole chiave


Esegui in Google Colab (https://colab.research.google.com/github/tensorflow/docs/blob/master/site/en/tutorials/audio/simple_audio.ipynb?hl=it)


Visualizza l'origine su GitHub (https://github.com/tensorflow/docs/blob/master/site/en/tutorials/audio/simple_audio.ipynb)


Scarica (https://storage.googleapis.com/tensorflow_docs/docs/site_quaderno/en/tutorials/audio/simple_audio.ipynb)

Questo tutorial mostra come preelaborare file audio nel formato WAV e creare e addestrare un modello di riconoscimento vocale automatico (https://en.wikipedia.org/wiki/Speech_recognition) di base (ASR) per riconoscere dieci parole diverse. Utilizzerai una parte del set di dati dei comandi vocali (https://www.tensorflow.org/datasets/catalog/speech_commands?hl=it) (Warden, 2018 (<https://arxiv.org/abs/1804.03209>)), che contiene brevi clip audio (di un secondo o meno) di comandi, come "giù", "vai", "sinistra", "no", "destra", "stop", "su" e "sì".

I sistemi (<https://ai.googleblog.com/search/label/Speech%20Recognition>) di riconoscimento vocale e audio del mondo reale sono complessi. Ma, come la classificazione delle immagini con il set di dati MNIST (<https://www.tensorflow.org/tutorials/quickstart/beginner?hl=it>) , questo tutorial dovrebbe darti una comprensione di base delle tecniche coinvolte.

Impostare

Importa i moduli e le dipendenze necessari. Nota che utilizzerai seaborn (<https://seaborn.pydata.org/>) per la visualizzazione in questo tutorial.

```
import os
import pathlib

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

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

Importa il mini set di dati dei comandi vocali

Per risparmiare tempo con il caricamento dei dati, lavorerai con una versione più piccola del set di dati dei comandi vocali. Il set di dati originale (https://www.tensorflow.org/datasets/catalog/speech_commands?hl=it) è costituito da oltre 105.000 file audio nel formato di file audio WAV (Waveform) (<https://www.aelius.com/njh/wavemetatools/doc/riffmci.pdf>) di persone che pronunciano 35 parole diverse. Questi dati sono stati raccolti da Google e rilasciati con una licenza CC BY.

Scarica ed estrai il file `mini_speech_commands.zip` contenente i set di dati dei comandi vocali più piccoli con `tf.keras.utils.get_file` (https://www.tensorflow.org/api_docs/python/tf/keras/utils/get_file?hl=it) :

```
DATASET_PATH = '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='data')

```

```

Downloading data from http://storage.googleapis.com/download.tensorflow.org/data/mini_speech_commands.zip
182083584/182082353 [=====] - 1s 0us/step
182091776/182082353 [=====] - 1s 0us/step

```

I clip audio del set di dati sono archiviati in otto cartelle corrispondenti a ciascun comando vocale: `no` , `yes` , `down` , `go` , `left` , `up` , `right` e `stop` :

```

commands = np.array(tf.io.gfile.listdir(str(data_dir)))
commands = commands[commands != 'README.md']
print('Commands:', commands)

```

```

Commands: ['stop' 'left' 'no' 'go' 'yes' 'down' 'right' 'up']

```

Estrai le clip audio in un elenco chiamato nomi di `filenames` e mescolalo in modo casuale:

```

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: 8000
Number of examples per label: 1000
Example file tensor: tf.Tensor(b'data/mini_speech_commands/yes/db72a474_nohash_0.wav', shape=(), dtype=string)

```

Dividi i nomi dei `filenames` in set di addestramento, convalida e test utilizzando rispettivamente un rapporto 80:10:10:

```

train_files = filenames[:6400]
val_files = filenames[6400: 6400 + 800]
test_files = filenames[-800:]

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

```

```

Training set size 6400
Validation set size 800
Test set size 800

```

Leggi i file audio e le loro etichette

In questa sezione si pre-elabora il dataset, creando tensori decodificati per le forme d'onda e le etichette corrispondenti. Notare che:

- Ciascun file WAV contiene dati di serie temporali con un determinato numero di campioni al secondo.

- Ogni campione rappresenta l' ampiezza (<https://en.wikipedia.org/wiki/Amplitude>) del segnale audio in quel momento specifico.
- In un sistema a 16 bit (https://en.wikipedia.org/wiki/Audio_bit_depth) , come i file WAV nel set di dati Mini Speech Commands, i valori di ampiezza vanno da -32.768 a 32.767.
- La frequenza ([https://en.wikipedia.org/wiki/Sampling_\(signal_processing\)#Audio_sampling](https://en.wikipedia.org/wiki/Sampling_(signal_processing)#Audio_sampling)) di campionamento per questo set di dati è 16kHz.

La forma del tensore restituito da `tf.audio.decode_wav` (https://www.tensorflow.org/api_docs/python/tf/audio/decode_wav?hl=it) è `[samples, channels]` , dove i `channels` sono 1 per mono o 2 per stereo. Il mini set di dati dei comandi vocali contiene solo registrazioni mono.

```
test_file = tf.io.read_file(DATASET_PATH+'down/0a9f9af7_nohash_0.wav')
test_audio, _ = tf.audio.decode_wav(contents=test_file)
test_audio.shape
```

```
TensorShape([13654, 1])
```

Ora definiamo una funzione che preelabora i file audio WAV grezzi del set di dati in tensori audio:

```
def decode_audio(audio_binary):
    # Decode WAV-encoded audio files to `float32` tensors, normalized
    # to the [-1.0, 1.0] range. Return `float32` audio and a sample rate.
    audio, _ = tf.audio.decode_wav(contents=audio_binary)
    # Since all the data is single channel (mono), drop the `channels`
    # axis from the array.
    return tf.squeeze(audio, axis=-1)
```

Definisci una funzione che crei etichette utilizzando le directory principali per ogni file:

- Dividi i percorsi dei file in `tf.RaggedTensor` (https://www.tensorflow.org/api_docs/python/tf/RaggedTensor?hl=it) s (tensori con dimensioni irregolari, con sezioni che possono avere lunghezze diverse).

```
def get_label(file_path):
    parts = tf.strings.split(
        input=file_path,
        sep=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]
```

Definisci un'altra funzione di supporto, `get_waveform_and_label` , che metta tutto insieme:

- L'input è il nome del file audio WAV.
- L'output è una tupla contenente i tensori audio ed etichetta pronti per l'apprendimento supervisionato.

```
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
```

Costruisci il training set per estrarre le coppie di etichette audio:

- Crea un `tf.data.Dataset` (https://www.tensorflow.org/api_docs/python/tf/data/Dataset?hl=it) con `Dataset.from_tensor_slices` (https://www.tensorflow.org/api_docs/python/tf/data/Dataset?hl=it#from_tensor_slices) e `Dataset.map` (https://www.tensorflow.org/api_docs/python/tf/data/Dataset?hl=it#map) , utilizzando `get_waveform_and_label` definito in precedenza.

In seguito creerai i set di convalida e test utilizzando una procedura simile.

```
AUTOTUNE = tf.data.AUTOTUNE

files_ds = tf.data.Dataset.from_tensor_slices(train_files)

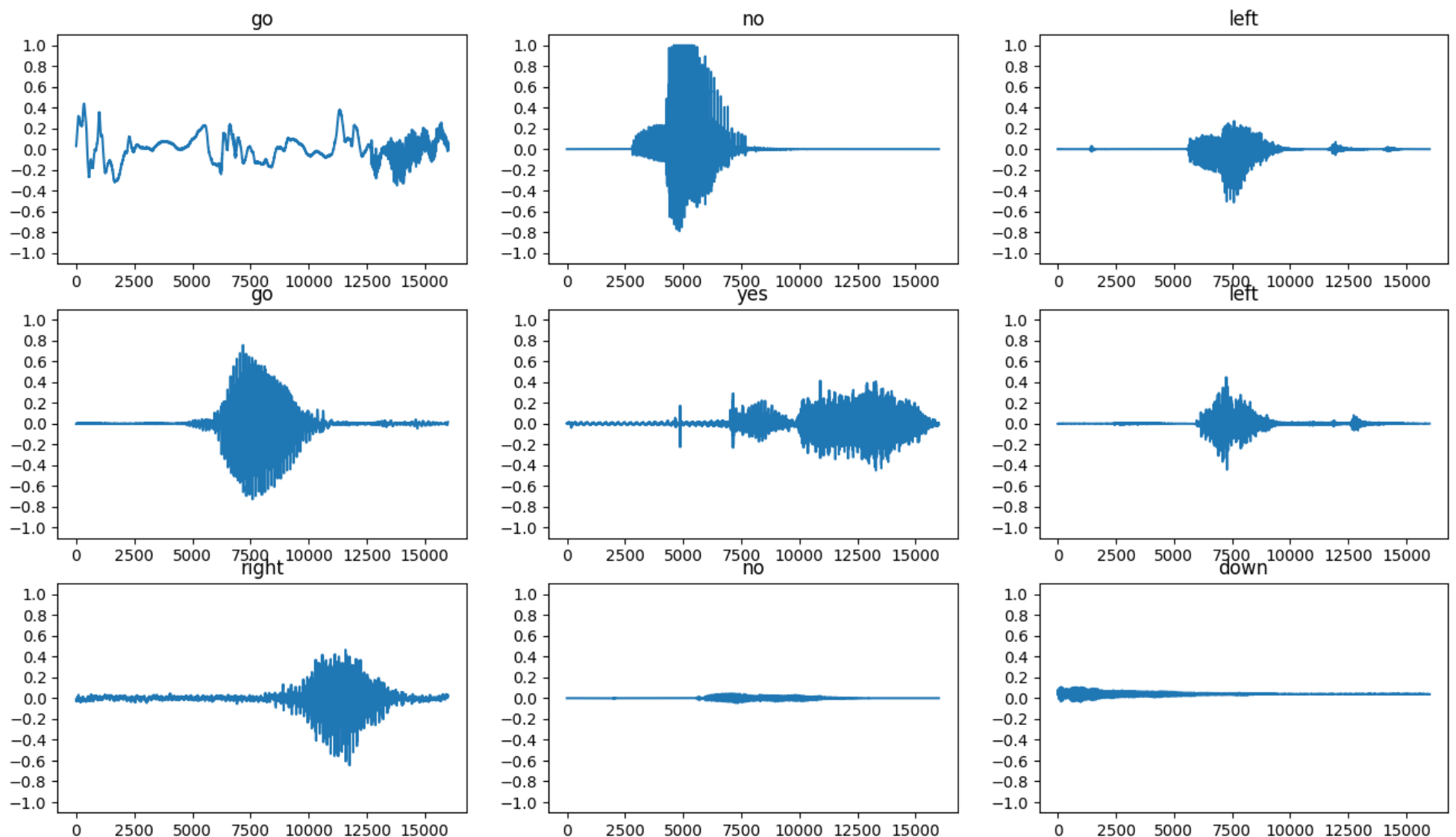
waveform_ds = files_ds.map(
    map_func=get_waveform_and_label,
    num_parallel_calls=AUTOTUNE)
```

Tracciamo alcune forme d'onda audio:

```
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()
```



Converti le forme d'onda in spettrogrammi

Le forme d'onda nel set di dati sono rappresentate nel dominio del tempo. Successivamente, trasformerai le forme d'onda dai segnali nel dominio del tempo nei segnali nel dominio della frequenza del tempo calcolando la trasformata di Fourier a breve termine (STFT) (https://en.wikipedia.org/wiki/Short-time_Fourier_transform) per convertire le forme d'onda in spettrogrammi (<https://en.wikipedia.org/wiki/Spectrogram>), che mostrano le variazioni di frequenza nel tempo e possono essere rappresentato come immagini 2D. Inverai le immagini dello spettrogramma nella tua rete neurale per addestrare il modello.

Una trasformata di Fourier (`tf.signal.fft` (https://www.tensorflow.org/api_docs/python/tf/signal/fft?hl=it)) converte un segnale nelle sue

frequenze componenti, ma perde tutte le informazioni temporali. In confronto, STFT (`tf.signal.stft` (https://www.tensorflow.org/api_docs/python/tf/signal/stft?hl=it)) divide il segnale in finestre temporali ed esegue una trasformata di Fourier su ciascuna finestra, preservando alcune informazioni temporali e restituendo un tensore 2D su cui è possibile eseguire convoluzioni standard.

Creare una funzione di utilità per convertire le forme d'onda in spettrogrammi:

- Le forme d'onda devono essere della stessa lunghezza, in modo che quando le si converte in spettrogrammi, i risultati abbiano dimensioni simili. Questo può essere fatto semplicemente riempiendo di zero le clip audio che sono più brevi di un secondo (usando `tf.zeros` (https://www.tensorflow.org/api_docs/python/tf/zeros?hl=it)).
- Quando si chiama `tf.signal.stft` (https://www.tensorflow.org/api_docs/python/tf/signal/stft?hl=it) , scegliere i parametri `frame_length` e `frame_step` modo tale che l'"immagine" dello spettrogramma generato sia quasi quadrata. Per ulteriori informazioni sulla scelta dei parametri STFT, fare riferimento a [questo video Coursera](https://www.coursera.org/lecture/audio-signal-processing/stft-2-tjEQe) (<https://www.coursera.org/lecture/audio-signal-processing/stft-2-tjEQe>) sull'elaborazione del segnale audio e STFT.
- L'STFT produce una matrice di numeri complessi che rappresentano l'ampiezza e la fase. Tuttavia, in questo tutorial utilizzerai solo la grandezza, che puoi ricavare applicando `tf.abs` (https://www.tensorflow.org/api_docs/python/tf/math/abs?hl=it) sull'output di `tf.signal.stft` (https://www.tensorflow.org/api_docs/python/tf/signal/stft?hl=it) .

```
def get_spectrogram(waveform):
    # Zero-padding for an audio waveform with less than 16,000 samples.
    input_len = 16000
    waveform = waveform[:input_len]
    zero_padding = tf.zeros(
        [16000] - tf.shape(waveform),
        dtype=tf.float32)
    # Cast the waveform tensors' dtype to float32.
    waveform = tf.cast(waveform, dtype=tf.float32)
    # Concatenate the waveform with `zero_padding`, which ensures all audio
    # clips are of the same length.
    equal_length = tf.concat([waveform, zero_padding], 0)
    # Convert the waveform to a spectrogram via a STFT.
    spectrogram = tf.signal.stft(
        equal_length, 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
```

Quindi, inizia a esplorare i dati. Stampa le forme della forma d'onda tensorizzata di un esempio e lo spettrogramma corrispondente e riproduci l'audio originale:

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

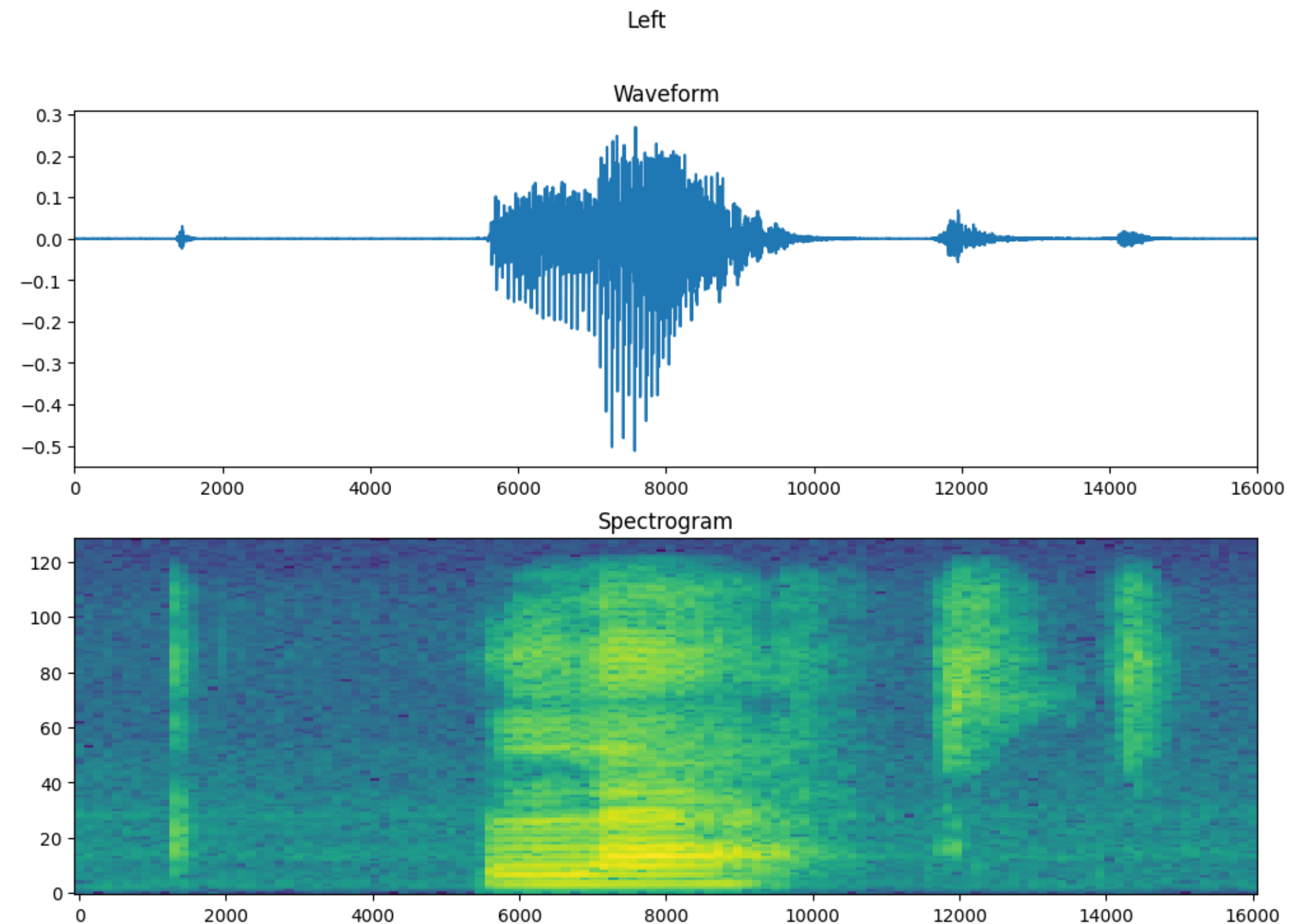
Ora definiamo una funzione per visualizzare uno spettrogramma:

```
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)
```

Traccia la forma d'onda dell'esempio nel tempo e lo spettrogramma corrispondente (frequenze nel tempo):

```
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()
```



Definire ora una funzione che trasformi il dataset della forma d'onda in spettrogrammi e le relative etichette come ID interi:


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

Mappa `get_spectrogram_and_label_id` tra gli elementi del set di dati con `Dataset.map` (https://www.tensorflow.org/api_docs/python/tf/data/Dataset?hl=it#map) :

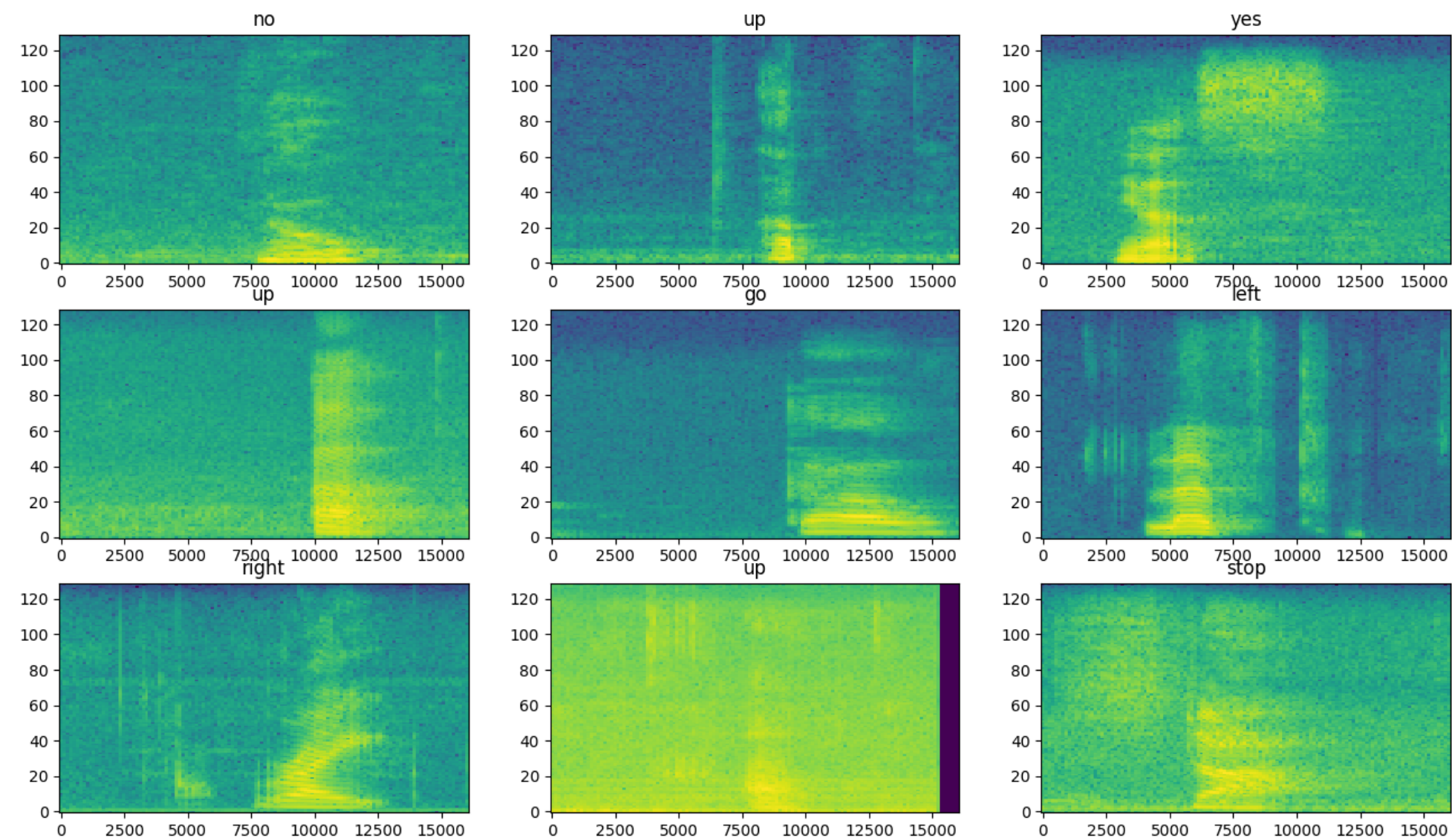
```
spectrogram_ds = waveform_ds.map(
    map_func=get_spectrogram_and_label_id,
    num_parallel_calls=AUTOTUNE)
```

Esaminare gli spettrogrammi per diversi esempi del set di dati:

```
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(spectrogram.numpy(), ax)
    ax.set_title(commands[label_id.numpy()])
    ax.axis('off')

plt.show()
```



Costruisci e addestra il modello

Ripetere la preelaborazione del set di addestramento sui set di convalida e test:

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

Batch i set di addestramento e convalida per l'addestramento del modello:

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

Aggiungi le operazioni `Dataset.cache` (https://www.tensorflow.org/api_docs/python/tf/data/Dataset?hl=it#cache) e `Dataset.prefetch` (https://www.tensorflow.org/api_docs/python/tf/data/Dataset?hl=it#prefetch) per ridurre la latenza di lettura durante il training del modello:

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

Per il modello, utilizzerai una semplice rete neurale convoluzionale (CNN), poiché hai trasformato i file audio in immagini spettrogramma.

Il tuo modello `tf.keras.Sequential` (https://www.tensorflow.org/api_docs/python/tf/keras/Sequential?hl=it) utilizzerà i seguenti livelli di preelaborazione Keras:

- `tf.keras.layers.Resizing` (https://www.tensorflow.org/api_docs/python/tf/keras/layers/Resizing?hl=it) : per eseguire il downsampling dell'input per consentire al modello di allenarsi più velocemente.
- `tf.keras.layers.Normalization` (https://www.tensorflow.org/api_docs/python/tf/keras/layers/Normalization?hl=it) : per normalizzare ogni pixel nell'immagine in base alla sua media e deviazione standard.

Per il livello di `Normalization`, il suo metodo di `adapt` dovrebbe prima essere chiamato sui dati di addestramento per calcolare le statistiche aggregate (ovvero la media e la deviazione standard).

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

# 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=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,
    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.Dense(120, activation='relu'),
layers.Dropout(0.5),
layers.Dense(num_labels),
])

model.summary()
```

Input shape: (124, 129, 1)
Model: "sequential"

Layer (type)	Output Shape	Param #
resizing (Resizing)	(None, 32, 32, 1)	0
normalization (Normalization)	(None, 32, 32, 1)	3
conv2d (Conv2D)	(None, 30, 30, 32)	320
conv2d_1 (Conv2D)	(None, 28, 28, 64)	18496

Configura il modello Keras con l'ottimizzatore Adam e la perdita di entropia incrociata:

```
model.compile(
    optimizer=tf.keras.optimizers.Adam(),
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    metrics=['accuracy'],
)
```

Addestra il modello in 10 epoche a scopo dimostrativo:

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

```
Epoch 1/10
100/100 [=====] - 6s 41ms/step - loss: 1.7503 - accuracy: 0.3630 - val_loss: 1.2850 - val_acc
Epoch 2/10
100/100 [=====] - 0s 5ms/step - loss: 1.2101 - accuracy: 0.5698 - val_loss: 0.9314 - val_acc
Epoch 3/10
100/100 [=====] - 0s 5ms/step - loss: 0.9336 - accuracy: 0.6703 - val_loss: 0.7529 - val_acc
Epoch 4/10
100/100 [=====] - 0s 5ms/step - loss: 0.7503 - accuracy: 0.7397 - val_loss: 0.6721 - val_acc
Epoch 5/10
100/100 [=====] - 0s 5ms/step - loss: 0.6367 - accuracy: 0.7741 - val_loss: 0.6061 - val_acc
Epoch 6/10
100/100 [=====] - 0s 5ms/step - loss: 0.5650 - accuracy: 0.7987 - val_loss: 0.5489 - val_acc
Epoch 7/10
100/100 [=====] - 0s 5ms/step - loss: 0.5099 - accuracy: 0.8183 - val_loss: 0.5344 - val_acc
```

Tracciamo le curve di perdita di training e validazione per verificare come il tuo modello è migliorato durante il training:

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

png

Valuta le prestazioni del modello

Esegui il modello sul set di prova e verifica le prestazioni del modello:

```
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%}')
```

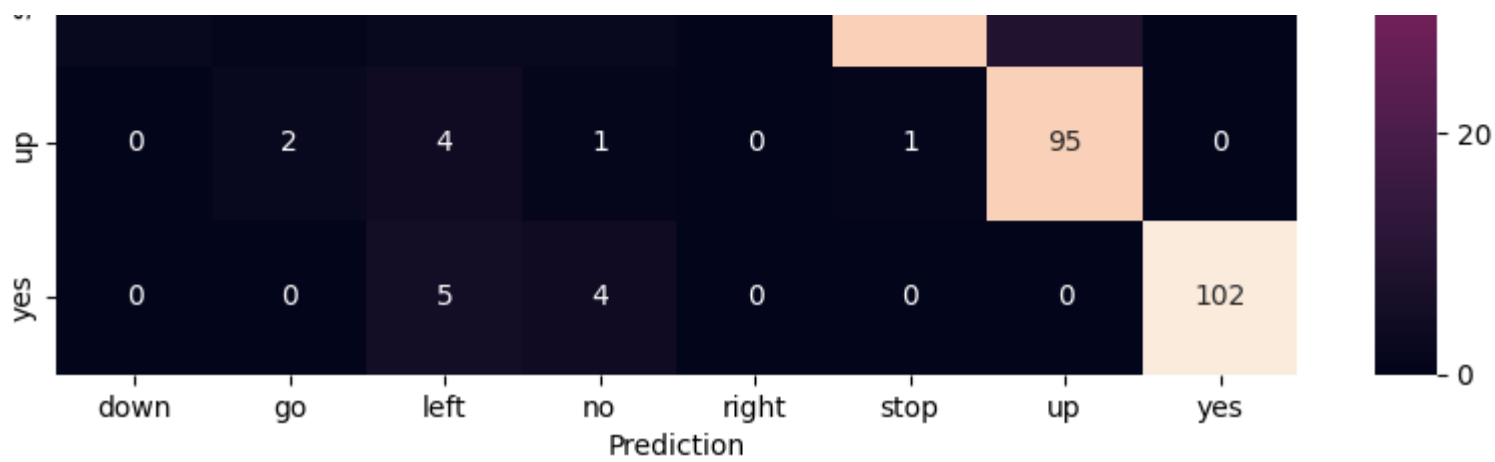
Test set accuracy: 85%

Visualizza una matrice di confusione

Usa una matrice di confusione (<https://developers.google.com/machine-learning/glossary?hl=it#confusion-matrix>) per verificare quanto bene il modello ha classificato ciascuno dei comandi nel set di test:

```
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()
```





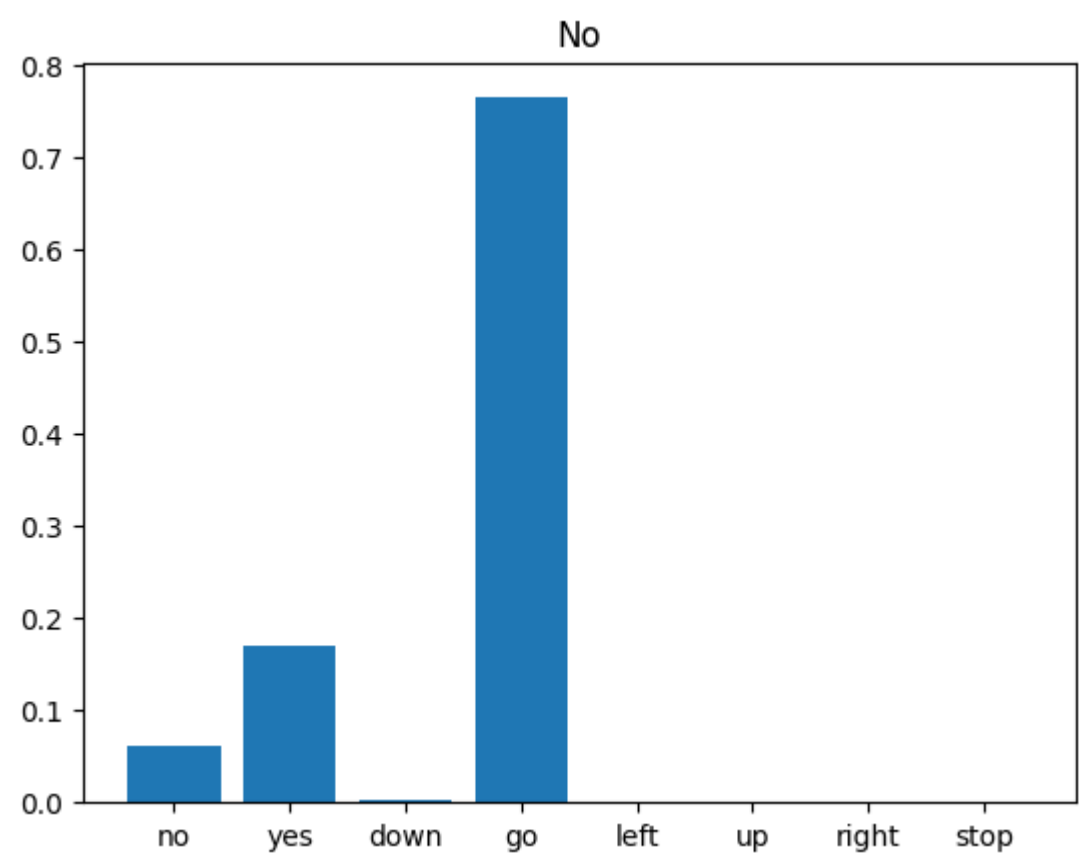
Esegui l'inferenza su un file audio

Infine, verifica l'output della previsione del modello utilizzando un file audio di input di qualcuno che dice "no". Quanto bene si comporta il tuo modello?

```
sample_file = data_dir/'no/01bb6a2a_nohash_0.wav'

sample_ds = preprocess_dataset([str(sample_file)])

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()
```



Come suggerisce l'output, il tuo modello dovrebbe aver riconosciuto il comando audio come "no".

Prossimi passi

Questo tutorial ha dimostrato come eseguire una semplice classificazione audio/riconoscimento vocale automatico utilizzando una rete neurale convoluzionale con TensorFlow e Python. Per saperne di più, considera le seguenti risorse:

- Il tutorial [sulla classificazione del suono con YAMNet](https://www.tensorflow.org/hub/tutorials/yamnet?hl=it) (https://www.tensorflow.org/hub/tutorials/yamnet?hl=it) mostra come utilizzare l'apprendimento del trasferimento per la classificazione audio.
- I taccuini della [sfida di riconoscimento vocale TensorFlow di Kaggle](https://www.kaggle.com/c/tensorflow-speech-recognition-challenge/overview) (https://www.kaggle.com/c/tensorflow-speech-recognition-challenge/overview) .
- Il [TensorFlow.js - Riconoscimento audio utilizzando il codelab di apprendimento del trasferimento](https://codelabs.developers.google.com/codelabs/tensorflowjs-audio-codelab/index.html?hl=it#0) (https://codelabs.developers.google.com/codelabs/tensorflowjs-audio-codelab/index.html?hl=it#0) insegna come creare la tua app Web

(<https://codelabs.developers.google.com/codelabs/tensorflowjs-audio-codelab/index.html?hl=it#0>) insegna come creare la tua app web interattiva per la classificazione audio.

- Un tutorial sul deep learning per il recupero di informazioni musicali (<https://arxiv.org/abs/1709.04396>) (Choi et al., 2017) su arXiv.
- TensorFlow ha anche un supporto aggiuntivo per la preparazione e l'aumento dei dati audio (<https://www.tensorflow.org/io/tutorials/audio?hl=it>) per aiutare con i tuoi progetti basati sull'audio.
- Prendi in considerazione l'utilizzo della libreria librosa (<https://librosa.org/>) , un pacchetto Python per l'analisi di musica e audio.

Except as otherwise noted, the content of this page is licensed under the [Creative Commons Attribution 4.0 License](https://creativecommons.org/licenses/by/4.0/) (<https://creativecommons.org/licenses/by/4.0/>), and code samples are licensed under the [Apache 2.0 License](https://www.apache.org/licenses/LICENSE-2.0) (<https://www.apache.org/licenses/LICENSE-2.0>). For details, see the [Google Developers Site Policies](https://developers.google.com/site-policies?hl=it) (<https://developers.google.com/site-policies?hl=it>). Java is a registered trademark of Oracle and/or its affiliates.

Last updated 2022-01-26 UTC.