SPEECH RECOGNATION SEDERHANA MENGGUNAKAN TENSORFLOW

Tutorial ini akan menunjukkan cara membangun jaringan pengenalan suara dasar yang mengenali sepuluh kata berbeda. Penting untuk diketahui bahwa sistem pengenalan suara dan suara yang sebenarnya jauh lebih kompleks, tetapi seperti MNIST untuk gambar, ini akan memberi Anda pemahaman dasar tentang teknik yang terlibat. Setelah Anda menyelesaikan tutorial ini, Anda akan memiliki model yang mencoba mengklasifikasikan klip audio satu detik sebagai "turun", "pergi", "kiri", "tidak", "kanan", "berhenti", "atas " dan ya".

A. Persiapan

Impor modul dan dependensi yang diperlukan.

```
import os
import pathlib

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

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


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

B. Impor kumpulan data Perintah Ucapan

Anda akan menulis skrip untuk mengunduh sebagian dari **kumpulan data Perintah Ucapan** (https://www.tensorflow.org/datasets/catalog/speech_commands) . Dataset asli terdiri dari lebih dari 105.000 file audio WAV dari orang-orang yang mengucapkan tiga puluh kata yang berbeda. Data ini dikumpulkan oleh Google dan dirilis di bawah lisensi CC BY.

Anda akan menggunakan sebagian dari kumpulan data untuk menghemat waktu dengan pemuatan data. Ekstrak mini_speech_commands.zip dan muat menggunakan **tf.data** API (https://www.tensorflow.org/api_docs/python/tf/data).

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

Periksa statistik dasar tentang kumpulan data.

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

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

Ekstrak file audio ke dalam daftar dan kocok.

```
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/right/a8e25ebb_nohash_0.wav', shape=(), dtype=string)
```

Pisahkan file menjadi set pelatihan, validasi, dan pengujian masing-masing menggunakan rasio 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
```

C. Membaca file audio dan labelnya

File audio awalnya akan dibaca sebagai file biner, yang ingin Anda ubah menjadi tensor numerik.

Untuk memuat file audio, Anda akan menggunakan **tf.audio.decode_wav (https://www.tensorflow.org/api_docs/python/tf/audio/decode_wav)**, yang mengembalikan audio yang disandikan WAV sebagai Tensor dan laju sampel.

File WAV berisi data deret waktu dengan sejumlah sampel per detik. Setiap sampel mewakili amplitudo sinyal audio pada waktu tertentu. Dalam sistem 16-bit, seperti file dalam mini_speech_commands, nilainya berkisar antara -32768 hingga 32767. Kecepatan

sampel untuk kumpulan data ini adalah 16kHz. Perhatikan bahwa **tf.audio.decode_wav** akan menormalkan nilai ke kisaran [-1.0, 1.0].

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

Label untuk setiap file WAV adalah direktori induknya.

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

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

Mari kita definisikan metode yang akan mengambil nama file dari file WAV dan mengeluarkan tupel yang berisi audio dan label untuk pelatihan yang diawasi.

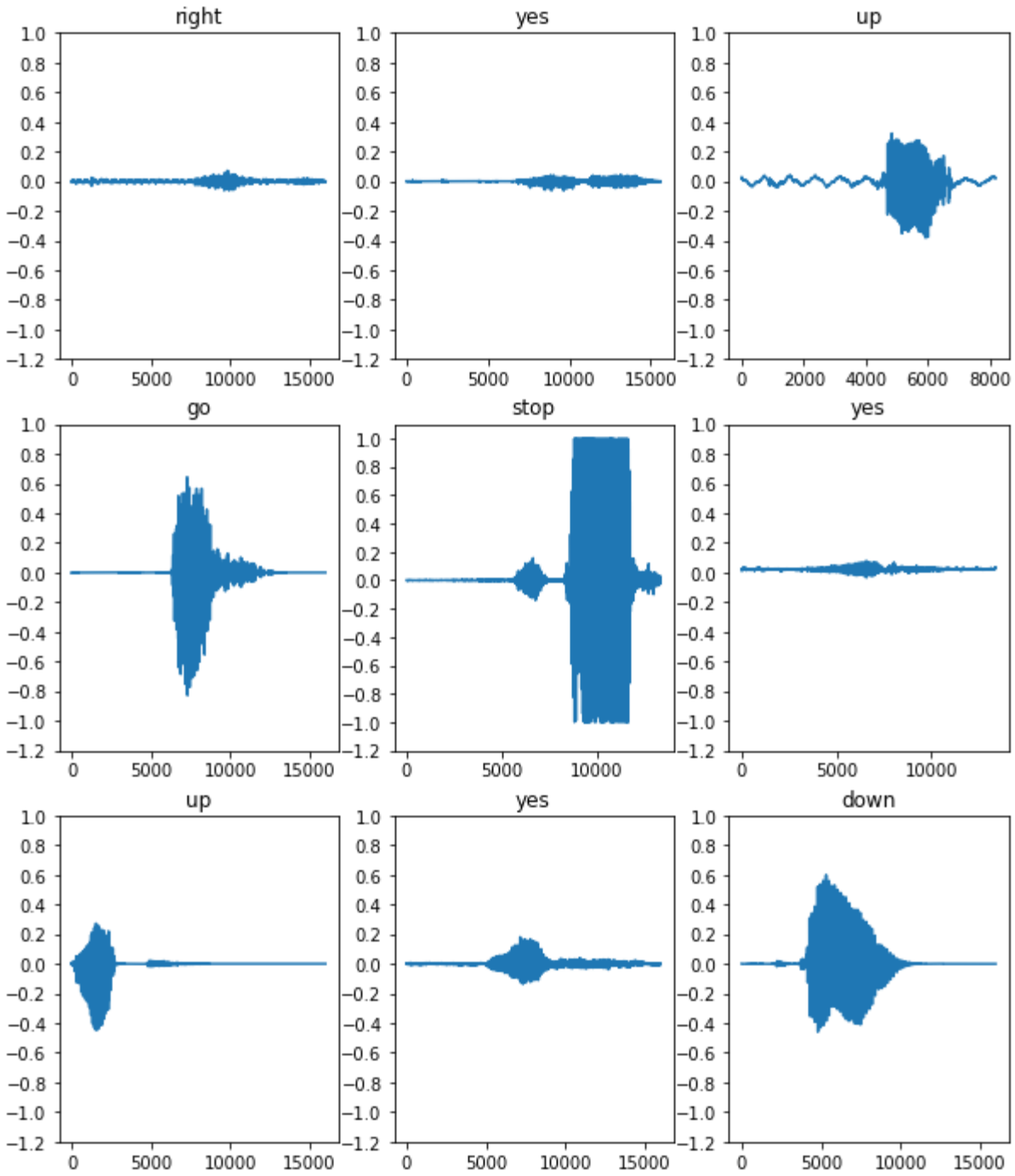
```
def get_waveform_and_label(file_path):
  label = get_label(file_path)
  audio_binary = tf.io.read_file(file_path)
  waveform = decode_audio(audio_binary)
  return waveform, label
```

Anda sekarang akan menerapkan process_path untuk membangun set pelatihan Anda untuk mengekstrak pasangan label audio dan memeriksa hasilnya. Anda akan membuat set validasi dan pengujian menggunakan prosedur serupa nanti.

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

Mari kita periksa beberapa bentuk gelombang audio dengan label yang sesuai.

```
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()
```



D. Spektogram

Anda akan mengubah bentuk gelombang menjadi spektogram, yang menunjukkan perubahan frekuensi dari waktu ke waktu dan dapat direpresentasikan sebagai gambar 2D. Ini dapat dilakukan dengan menerapkan transformasi Fourier waktu singkat (STFT) untuk mengubah audio menjadi domain frekuensi waktu.

Transformasi Fourier **(tf.signal.fft)** (https://www.tensorflow.org/api_docs/python/tf/signal/fft) mengubah sinyal menjadi frekuensi komponennya, tetapi kehilangan semua informasi waktu. STFT **(tf.signal.stft)** membagi sinyal menjadi beberapa jendela waktu dan menjalankan transformasi Fourier pada setiap jendela, mempertahankan beberapa informasi waktu, dan mengembalikan tensor 2D yang dapat Anda gunakan untuk menjalankan konvolusi standar.

STFT menghasilkan array bilangan kompleks yang mewakili besaran dan fase. Namun, Anda hanya memerlukan besaran untuk tutorial ini, yang dapat diturunkan dengan menerapkan **tf.abs** (https://www.tensorflow.org/api_docs/python/tf/signal/fft) pada output **tf.signal.stft.**

Pilih **frame_length** dan **frame_step** parameter sedemikian rupa sehingga "gambar" spektogram yang dihasilkan hampir persegi.

Anda juga ingin agar bentuk gelombang memiliki panjang yang sama, sehingga ketika Anda mengubahnya menjadi gambar spektogram, hasilnya akan memiliki dimensi yang serupa. Ini dapat dilakukan hanya dengan nol padding klip audio yang lebih pendek dari satu detik.

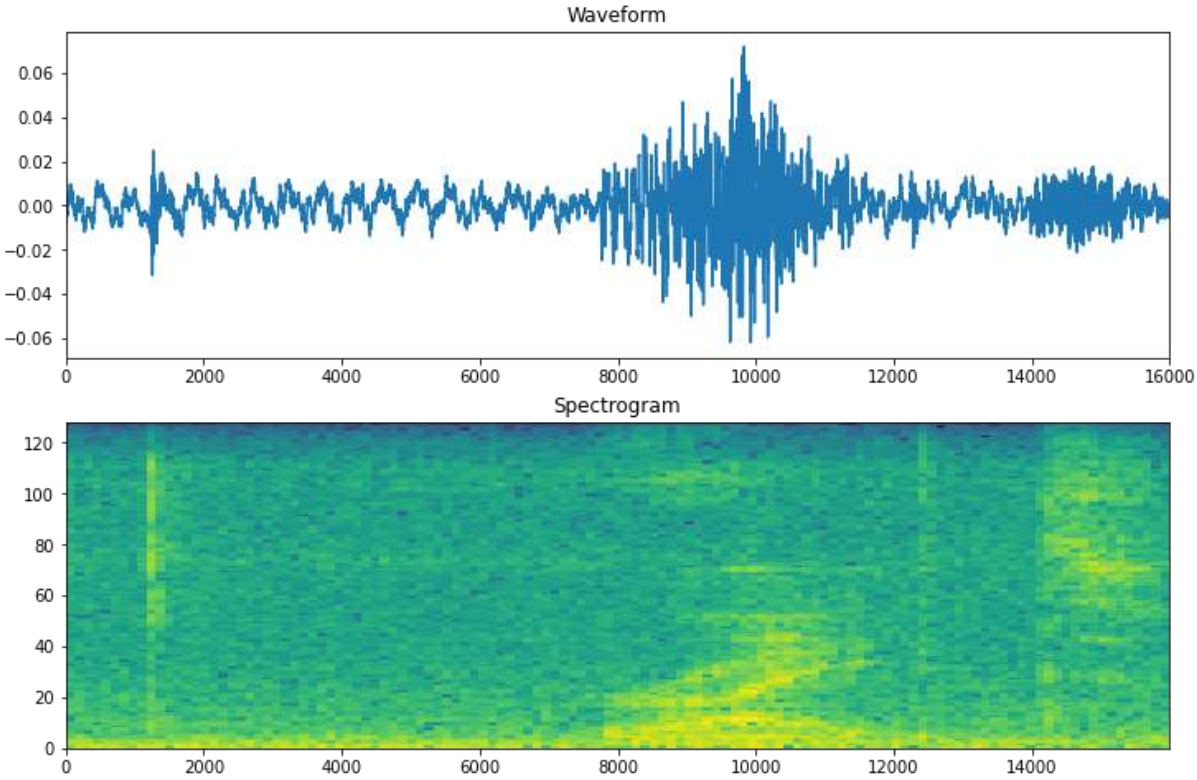
```
def get_spectrogram(waveform):
  # Padding for files with less than 16000 samples
  zero_padding = tf.zeros([16000] - tf.shape(waveform), dtype=tf.float32)

  # Concatenate audio with padding so that all audio clips will be of the
  # same length
  waveform = tf.cast(waveform, tf.float32)
  equal_length = tf.concat([waveform, zero_padding], 0)
  spectrogram = tf.signal.stft(
      equal_length, frame_length=255, frame_step=128)

  spectrogram = tf.abs(spectrogram)

  return spectrogram
```

Selanjutnya, Anda akan menjelajahi data. Bandingkan bentuk gelombang, spektogram, dan audio sebenarnya dari satu contoh dari kumpulan data.

```
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))
```

```
def plot_spectrogram(spectrogram, ax):
  # Convert to frequencies to log scale and transpose so that the time is
  # represented in the x-axis (columns). An epsilon is added to avoid 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)



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()
```

```
/home/kbuilder/.local/lib/python3.7/site-packages/ipykernel_launcher.py:9: MatplotlibDeprecationWarning:
shading='flat' when X and Y have the same dimensions as C is deprecated since 3.3.  Either specify the corners
of the quadrilaterals with X and Y, or pass shading='auto', 'nearest' or 'gouraud', or set
rcParams['pcolor.shading'].  This will become an error two minor releases later.
  if __name__ == '__main__':
```



Sekarang ubah dataset bentuk gelombang untuk memiliki gambar spektogram dan label yang sesuai sebagai ID bilangan bulat.

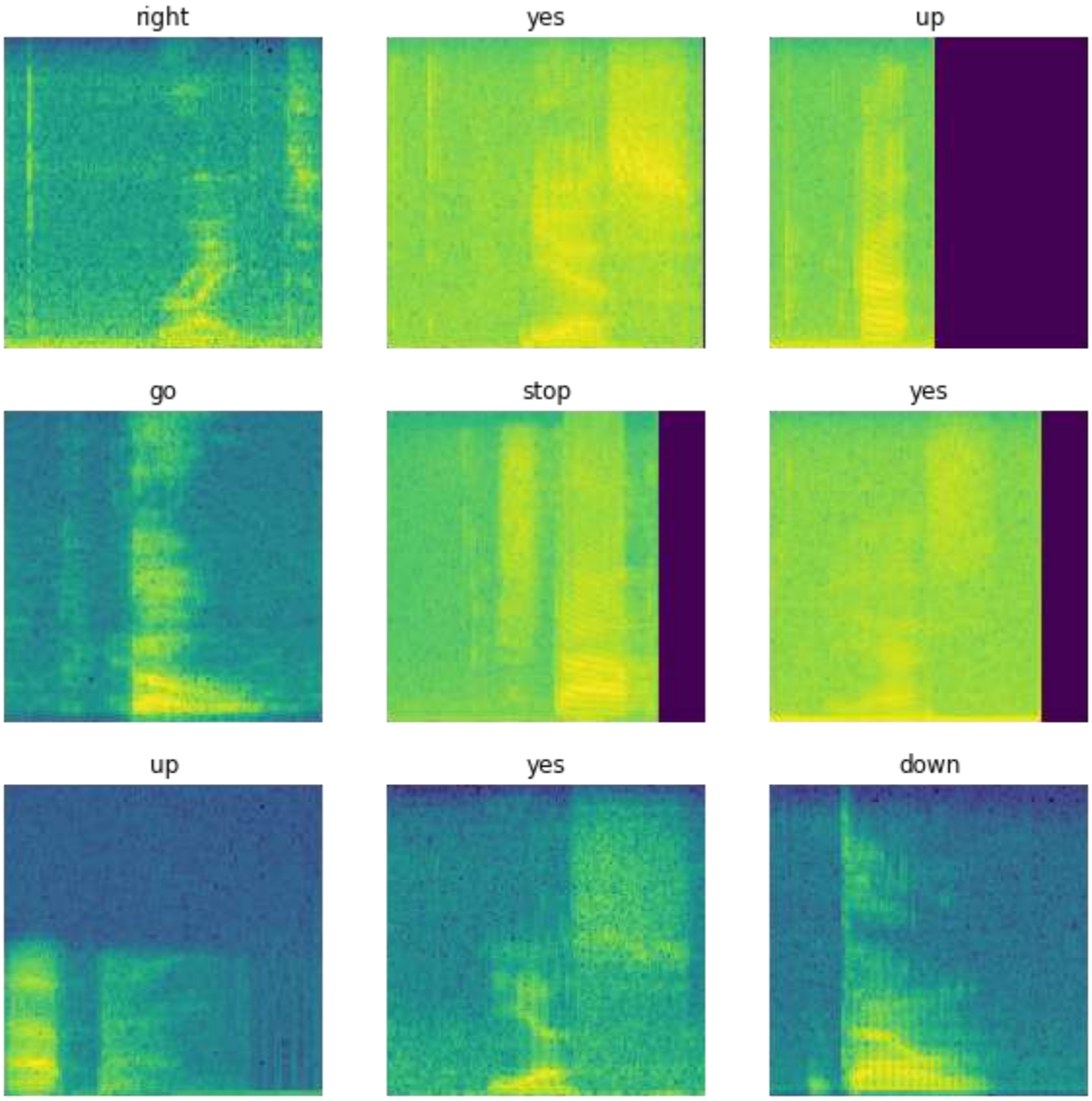
```
def get_spectrogram_and_label_id(audio, label):
  spectrogram = get_spectrogram(audio)
  spectrogram = tf.expand_dims(spectrogram, -1)
  label_id = tf.argmax(label == commands)
  return spectrogram, label_id

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

Periksa "gambar" spektogram untuk sampel yang berbeda dari kumpulan data.

```
rows = 3
cols = 3
n = rows*cols
fig, axes = plt.subplots(rows, cols, figsize=(10, 10))
for i, (spectrogram, label_id) in enumerate(spectrogram_ds.take(n)):
  r = i // cols
  c = i % cols
```

```
  ax = axes[r][c]
  plot_spectrogram(np.squeeze(spectrogram.numpy()), ax)
  ax.set_title(commands[label_id.numpy()])
  ax.axis('off')

plt.show()
```

/home/kbuilder/.local/lib/python3.7/site-packages/ipykernel_launcher.py:9: MatplotlibDeprecationWarning: shading='flat' when X and Y have the same dimensions as C is deprecated since 3.3. Either specify the corners of the quadrilaterals with X and Y, or pass shading='auto', 'nearest' or 'gouraud', or set rcParams['pcolor.shading']. This will become an error two minor releases later.
  if __name__ == '__main__':



E. Bangun dan latih model

Sekarang Anda dapat membangun dan melatih model Anda. Namun sebelum melakukannya, Anda harus mengulangi prapemrosesan set pelatihan pada set validasi dan pengujian.

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

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

Batch set pelatihan dan validasi untuk pelatihan model.

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

Tambahkan set data cache() dan prefetch() operasi untuk mengurangi latensi baca saat melatih model.

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

Untuk modelnya, Anda akan menggunakan jaringan saraf convolutional sederhana (CNN), karena Anda telah mengubah file audio menjadi gambar spektogram. Model ini juga memiliki lapisan prapemrosesan tambahan berikut:

- Sebuah Resizing layer untuk downsample input untuk memungkinkan model untuk kereta cepat.

- Sebuah Normalization lapisan untuk menormalkan setiap pixel dalam gambar berdasarkan deviasi mean dan standar.

  Untuk Normalizationlapisan, adaptmetodenya pertama-tama perlu dipanggil pada data pelatihan untuk menghitung statistik agregat (yaitu mean dan standar deviasi).

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

norm_layer = preprocessing.Normalization()
norm_layer.adapt(spectrogram_ds.map(lambda x, _: x))

model = models.Sequential([
    layers.Input(shape=input_shape),
    preprocessing.Resizing(32, 32),
    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.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
```

```
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 14, 14, 64)       0
_________________________________________________________________
dropout (Dropout)            (None, 14, 14, 64)       0
_________________________________________________________________
flatten (Flatten)            (None, 12544)            0
_________________________________________________________________
dense (Dense)                (None, 128)              1605760
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)              0
_________________________________________________________________
dense_1 (Dense)              (None, 8)                1032
=================================================================
Total params: 1,625,611
Trainable params: 1,625,608
Non-trainable params: 3
_________________________________________________________________
```

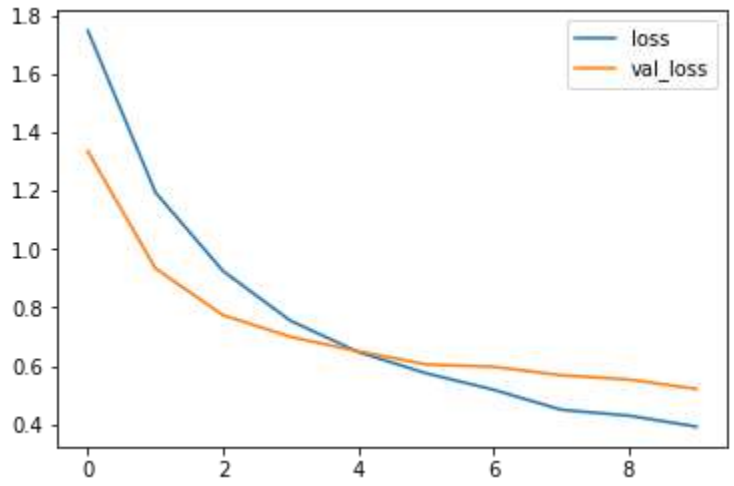
```
model.compile(
    optimizer=tf.keras.optimizers.Adam(),
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    metrics=['accuracy'],
)

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 [==============================] - 13s 40ms/step - loss: 1.7503 - accuracy: 0.3659 - val_loss: 1.3361 - val_accuracy: 0.5600
Epoch 2/10
100/100 [==============================] - 0s 4ms/step - loss: 1.1948 - accuracy: 0.5809 - val_loss: 0.9340 - val_accuracy: 0.7225
Epoch 3/10
100/100 [==============================] - 0s 4ms/step - loss: 0.9253 - accuracy: 0.6773 - val_loss: 0.7738 - val_accuracy: 0.7588
Epoch 4/10
100/100 [==============================] - 0s 4ms/step - loss: 0.7543 - accuracy: 0.7341 - val_loss: 0.6993 - val_accuracy: 0.7650
```

```
Epoch 5/10
100/100 [==============================] - 0s 4ms/step - loss: 0.6473 - accuracy: 0.7719 - val_loss: 0.6500 - val_accuracy: 0.7800
Epoch 6/10
100/100 [==============================] - 0s 4ms/step - loss: 0.5746 - accuracy: 0.7977 - val_loss: 0.6049 - val_accuracy: 0.7975
Epoch 7/10
100/100 [==============================] - 0s 4ms/step - loss: 0.5181 - accuracy: 0.8155 - val_loss: 0.5967 - val_accuracy: 0.8200
Epoch 8/10
100/100 [==============================] - 0s 4ms/step - loss: 0.4493 - accuracy: 0.8402 - val_loss: 0.5678 - val_accuracy: 0.8100
Epoch 9/10
100/100 [==============================] - 0s 4ms/step - loss: 0.4292 - accuracy: 0.8491 - val_loss: 0.5531 - val_accuracy: 0.8175
Epoch 10/10
100/100 [==============================] - 0s 4ms/step - loss: 0.3914 - accuracy: 0.8598 - val_loss: 0.5204 - val_accuracy: 0.8275
```

Mari kita periksa validasi loss curves dan pelatihan untuk melihat peningkatan model Anda selama pelatihan.

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



F. Evaluasi kinerja set tes

Mari kita jalankan model pada set pengujian dan periksa kinerjanya.

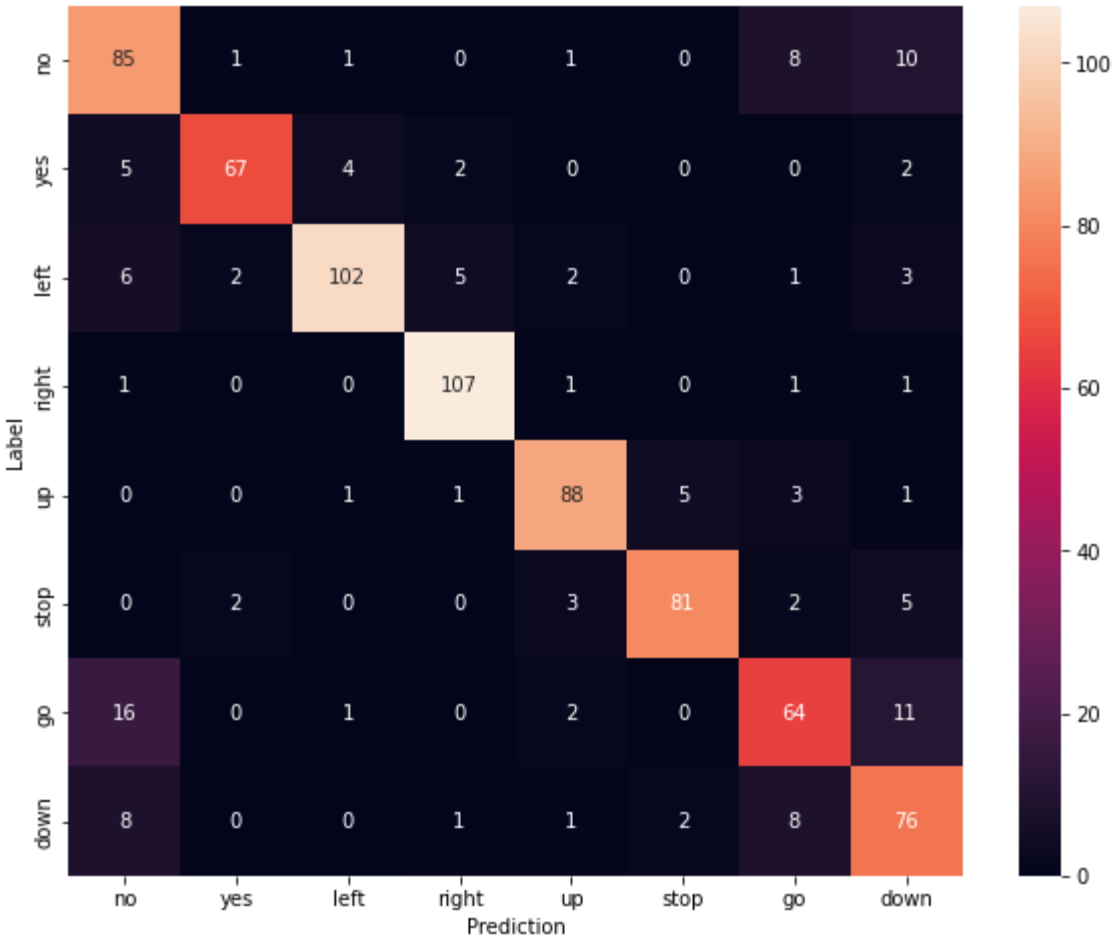
```
test_audio = []
test_labels = []

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: 84%

G. Tampilkan confusion matrix

Matriks konfusi sangat membantu untuk melihat seberapa baik model bekerja pada setiap perintah dalam set pengujian.

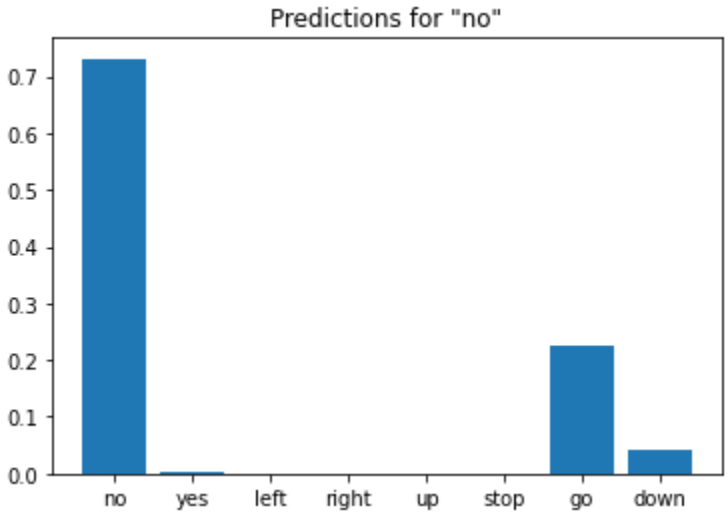
```
confusion_mtx = tf.math.confusion_matrix(y_true, y_pred)
plt.figure(figsize=(10, 8))
sns.heatmap(confusion_mtx, xticklabels=commands, yticklabels=commands,
            annot=True, fmt='g')
plt.xlabel('Prediction')
plt.ylabel('Label')
plt.show()
```



H. Jalankan inferensi pada file audio

Terakhir, verifikasi output prediksi model menggunakan file audio input dari seseorang yang mengatakan "tidak". Seberapa baik performa model Anda?

```
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()
```



Anda dapat melihat bahwa model Anda dengan sangat jelas mengenali perintah audio sebagai "no".

Tutorial ini menunjukkan bagaimana Anda dapat melakukan klasifikasi audio sederhana menggunakan jaringan saraf convolutional dengan TensorFlow dan Python.