Algolia DevCon - June 28, 29

Build Your Own Voice Recognition Model with Tensorflow by @FlippedC…

★
9,
0
4
5
re
s

Build Your Own Voice

Recognition Model wi

-thTensorflow

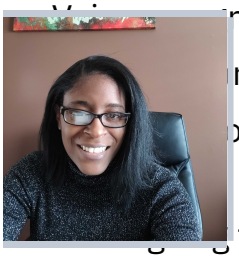
March 7th 2021 | 8min | by @F

lipp
ed
Co
din
g

★
8
0
4
ES
5
re
ad
s

Read by **Dr. One (en-US)**

Audio Presented by



@FlippedCoding

Milecia

Software/Hardware Engineer | Integration Nerd | speaker | Random inventor and slig...

nition is a complex problem across a number of industries. Knowing some of round handling audio data and how to classify sound samples is a good thing our data science toolbox.

to go through an example of classifying some sound clips using Tensorflow. get through this, you'll know enough to be able to build your own voice dels. With additional research, you can take these concepts and apply them to larger, more complex audio files.

You can find the full code in this Github repo.

Getting the Data

Gathering data is one of the hard problems in data science. There's so much data available, but not all of it is easy to use in machine learning problems. You have to make sure that the data is clean, labeled, and complete.

To do our example, we're going to use some audio files released by Google.

ate a new Conducto pipeline. This is where you'll be able to build, train, model and share a link with anybody else interested:

```
###
# Main Pipeline
###
def main() -> co.Serial:
    path = "/conducto/data/pipeline"
    root = co.Serial(image = get_image())

    # Get data from keras for testing and training
    root["Get Data"] = co.Exec(run_whole_thing, f"{path}/raw")

    return root
```

Then we'll start writing the `run_whole_thing` function:

```
def run_whole_thing(out_dir):
    os.makedirs(out_dir, exist_ok=True)
    # Set seed for experiment reproducibility
    seed = 55
    tf.random.set_seed(seed)
    np.random.seed(seed)
    data_dir = pathlib.Path("data/mini_speech_commands")
```

Next, we need to set up the directory to hold the audio files:

```
if not data_dir.exists():
    # Get the files from external source and put them in an accessible directory
    tf.keras.utils.get_file(
      'mini_speech_commands.zip',
       origin="http://storage.googleapis.com/download.tensorflow.org
/data/mini_speech_commands.zip",
       extract=True)
```

Pre-Processing the Data

Now that we have our data in the right directory, we can split it into training, test, and validation datasets.

First, we need to write a few functions to help pre-process the data so that it'll work in our model.

We need the data in a format our algorithm can understand. We'll be using a convolutional neural network, so the data needs to be transformed into images.

This first function will convert the binary audio file into a tensor:

```
# Convert the binary audio file to a tensor
def decode_audio(audio_binary):
```

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

Since we have a tensor we can work with that has the raw data, we need to get the labels to match them. That's what the following function does by getting the label for an audio file from the file path:

```
# Get the label (yes, no, up, down, etc) for an audio file.
def get_label(file_path):
  parts = tf.strings.split(file_path, os.path.sep)
  return parts[-2]
```

Next, we need to associate the audio files with the correct labels. We're doing this and returning a tuple that Tensorflow can work with:

```
# Create a tuple that has the labeled audio files
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
```

We briefly mentioned using the convolutional neural network (CNN) algorithm earlier. This is one of the ways we can handle a voice recognition model like this is. Typically CNNs work really well on image data and help decrease pre-processing time.

We're going to take advantage of that by converting our audio files into spectrograms. A spectrogram is an image of a spectrum of frequencies. If you take a look at an audio file, you'll see it's just frequency data. So we're going to write a function that converts our audio data into images:

```
# Convert audio files to images
```

```
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
```

Now that we have formatted our data as images, we need to apply the correct labels to those images. This is similar to what we did for the original audio files:

```
# Label the images created from the audio files and return a tuple
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
```

The last helper function we need is the one that will handle all of the above operations for any set of audio files we pass it:

```
# Preprocess any audio files
def preprocess_dataset(files, autotune, commands):
  # Creates the dataset
  files_ds = tf.data.Dataset.from_tensor_slices(files)

  # Matches audio files with correct labels
  output_ds = files_ds.map(get_waveform_and_label,
                num_parallel_calls=autotune)
  # Matches audio file images to the correct labels
  output_ds = output_ds.map(
    get_spectrogram_and_label_id,  num_parallel_calls=autotune)
  return output_ds
```

Now that we have all of these helper functions, we get to split the data.

Splitting the Data into Datasets

Converting audio files to images helps make the data easier to process with a CNN and that's why we wrote all of those helper functions. We'll do a couple of things to make splitting the data more simple.

First, we'll get a list of all of the potential commands for the audio files that we'll use in a few other places in the code:

```
# Get all of the commands for the audio files
commands = np.array(tf.io.gfile.listdir(str(data_dir)))
commands = commands[commands != 'README.md']
```

Then we'll get a list of all of the files in the data directory and shuffle them so we can assign random values to each of the datasets we need:

```
# Get a list of all the files in the directory
filenames = tf.io.gfile.glob(str(data_dir) + '/*/*')

# Shuffle the file names so that random bunches can be used as the training, testing, and
validation sets
filenames = tf.random.shuffle(filenames)

# Create the list of files for training data
train_files = filenames[:6400]

# Create the list of files for validation data
validation_files = filenames[6400: 6400 + 800]

# Create the list of files for test data
test_files = filenames[-800:]
```

Now we have our training, validation, and test files clearly separated so we can go ahead and pre-process these files to get them ready to build and test our model. We're using autotune here to tune the value of our parameters dynamically at runtime:

```
autotune = tf.data.AUTOTUNE
```

This first example is just to show how the pre-processing works and it gives us the `spectrogram_ds` value that we'll need in a bit:

```
# Get the converted audio files for training the model
files_ds = tf.data.Dataset.from_tensor_slices(train_files)
waveform_ds = files_ds.map(
  get_waveform_and_label, num_parallel_calls=autotune)
spectrogram_ds = waveform_ds.map(
  get_spectrogram_and_label_id, num_parallel_calls=autotune)
```

Since you've seen what it's like to go through the pre-processing steps, we can go ahead and use the helper function to handle this for all of the datasets:

```
# Preprocess the training, test, and validation datasets
train_ds = preprocess_dataset(train_files, autotune, commands)
validation_ds = preprocess_dataset(
  validation_files, autotune, commands)
test_ds = preprocess_dataset(test_files, autotune, commands)
```

We want to set a number of training examples that run in each iteration of the epochs so we'll set a batch size:

```
# Batch datasets for training and validation
batch_size = 64
train_ds = train_ds.batch(batch_size)
validation_ds = validation_ds.batch(batch_size)
```

Lastly, we can reduce the amount of latency in training our model by taking advantage of caching:

```
# Reduce latency while training
train_ds = train_ds.cache().prefetch(autotune)
validation_ds = validation_ds.cache().prefetch(autotune)
```

Our datasets are finally in a form that we can train the model with.

Building the Model

Since our datasets are clearly defined, we can go ahead and build the model. We'll be using a CNN to create our model so we'll need to get the shape of the data to get the correct shape for our layers. Then we go ahead build the model sequentially:

```
# Build model
for spectrogram, _ in spectrogram_ds.take(1):
  input_shape = spectrogram.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()
```

We do some configuration on the model so that it gives us the best accuracy possible:

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

The model is built so now all that's left is training it.

Training the Model

After all of the work did pre-processing the data and building the model, training is relatively simple. We determine how many epochs we want to run with our training and validation datasets:

```
# Finally train the model and return info about each epoch
EPOCHS = 10
model.fit(
    train_ds,
    validation_data=validation_ds,
    epochs=EPOCHS,
    callbacks=tf.keras.callbacks.EarlyStopping(verbose=1, patience=2),
)
```

That's it! The model has been trained and now we just need to test it.

Testing the Model

Now that we have a model with roughly 83% accuracy, it's time we test how well it performs on new data. So we take our test dataset and split the audio files from the labels:

```
# Test the model
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)
```

Then we take the audio data and use it in our model to see if it predicts the correct label:

```
# See how accurate the model is when making predictions on the test dataset
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%}')
```

Finishing the Pipeline

There's just a tiny bit of code that you'll need to finish your pipeline and make it

shareable with anyone. This defines the image that will be used in this Conducto pipeline and handles the file execution:

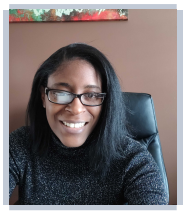
```
###
# Pipeline Helper functions
###
def get_image():
  return co.Image(
    "python:3.8-slim",
    copy_dir=".",
    reqs_py=["conducto", "tensorflow", "keras"],
  )

if __name__ == "__main__":
  co.main(default=main)
```

Now you can run `python pipeline.py --local` in your terminal and it should spin up a link to a new Conducto pipeline. If you don't have an account, you can make one for free here.

Conclusion

This is one of the ways you can solve an audio processing problem, but it can be much more complex depending on what data you're trying to analyze. Building it in a pipeline makes it easy to share with coworkers and get help or feedback if you run into bugs.

Previously published here.



by Milecia @FlippedCoding.

Software/Hardware Engineer | International tech speaker | Random inventor and slightly mad scientist

Check out my site!

RELATED STORIES

The Next 11 Th

ingsYouShouldDoForCI/CDPipelineOptimizatio

n

Published at Nov 09, 2020 by FlippedCoding #devops

A Roadmap to Mastering

Python

Published at Jun 09, 2023 by techtweeter

#programming

Demystifyi-

ngDimensionalModelling:UnveilingtheWhat,Why

y ’ a n d W h o ' s

Pu bl is he d at Ju n 08 , 20 23 by di sa #d at a-s ci en ce

U n l e a s h i n g

gThePowerOfCreativity:DivingIntoTheNewGoogl

-eSearchGenerativeExperience

Published at Jun 07, 2023 by mariaj #g

o
o
g
l
e

What is Data Profiling? Concepts and Example

s

Published at Jun 07, 2023 by spjatin4 #data #profiling

Nvidia's Ascent P

Published at Jun 07, 2023 by shekharyarkhan #trending-te