batch_exploration

October 18, 2020

1 Tensorflow Implementation

Here, we load the features from our `TFRecord` dataset in our `tensorflow` implementation. Here, we'll use our helper class, the `DatasetLoader` to interact with the `TFRecord` files our feature engineering step has stored. This will allow us to get `iter` objects for both the training and testing datasets. Here, we will get back the example and their labels from our feature engineering steps which generated Spectrograms.

[1]:
```
import os
import json

import tensorflow as tf
import librosa
import matplotlib.pyplot as plt
import numpy as np
```

[2]:
```
ROOT_DIR = '/home/thomas/Dir/ccny/ccny-masters-thesis'
os.chdir(f'{ROOT_DIR}/tensorflow')

from loader import BatchLoader
from dataset import GE2EDatasetLoader
```

The `example_dim` parameter tells us the shape of our features. Since spectrograms can be thought of as 2-dimensional images, here we pass `2`. One could confirm this by checking the metadata object.

[3]:
```
loader = GE2EDatasetLoader(
    root_dir = f'{ROOT_DIR}/feature-data'
)
```

[4]:
```
loader.get_metadata()['feature_shape']
```

[4]:
```
[40, 121]
```

In order to load our Spectrograms in a stream (putting them all in RAM would kill your computer quickly), we can load the training and testing datasets with `get_dataset()` and then get the `__iter__` on the `TFRecordDatset` objects which are returned. This will allow us to load one batch at a time.

[5]:
```
train_dataset = loader.get_train_dataset()
train_it = iter(train_dataset)
```

Now we can load one batch and display the features that end up in our model.

[6]:
```
examples, labels = next(train_it)
```

[7]:
```
examples.shape
```

[7]:
```
TensorShape([32, 40, 121])
```

In the Generalized End-to-End loss world, a batch contains M utterances each from N unique speakers, thus our `batch_size` should be NM. By doing this, the loss at the end of the network can optimize **separating** those N speakers in the embedding space. Here, we return the labels even though we don't have much use for them outside of verification and debugging.

[9]:
```
labels
```

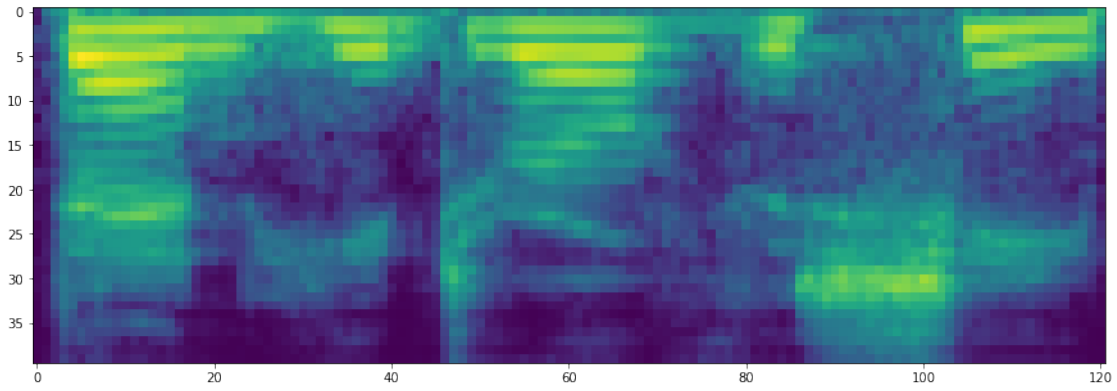
[9]:
```
<tf.Tensor: shape=(32,), dtype=int64, numpy=
array([279, 279, 279, 279, 294, 294, 294, 294, 278, 278, 278, 278, 166,
       166, 166, 166, 238, 238, 238, 238, 323, 323, 323, 323, 353, 353,
       353, 353, 313, 313, 313, 313])>
```

[38]:
```
# this should always be N, M
len(np.unique(labels)), len(labels) // len(np.unique(labels))
```

[38]:
```
(8, 4)
```

[42]:
```
idx = 16 # just an example

plt.figure(figsize=(15,8))
_ = plt.imshow(examples[idx].numpy())
```



[28]:
```
# confirm N from dataset
loader.get_metadata()['speakers_per_batch']
```

[28]: 8

[39]:
```
# confirm M from dataset
loader.get_metadata()['utterances_per_speaker']
```

[39]: 4