Sax | Not Sax

Instrument Detection in Digital Audio Using PyTorch Convolutional Neural Networks

by David Luther

Choosing a Dataset

NSynth

- Google Brain's Magenta
- 305,979 single-note samples from 1,006 different instruments

Choosing a Dataset

NSynth

- Google Brain's Magenta
- 305,979 single-note samples from 1,006 different instruments

IRMAS

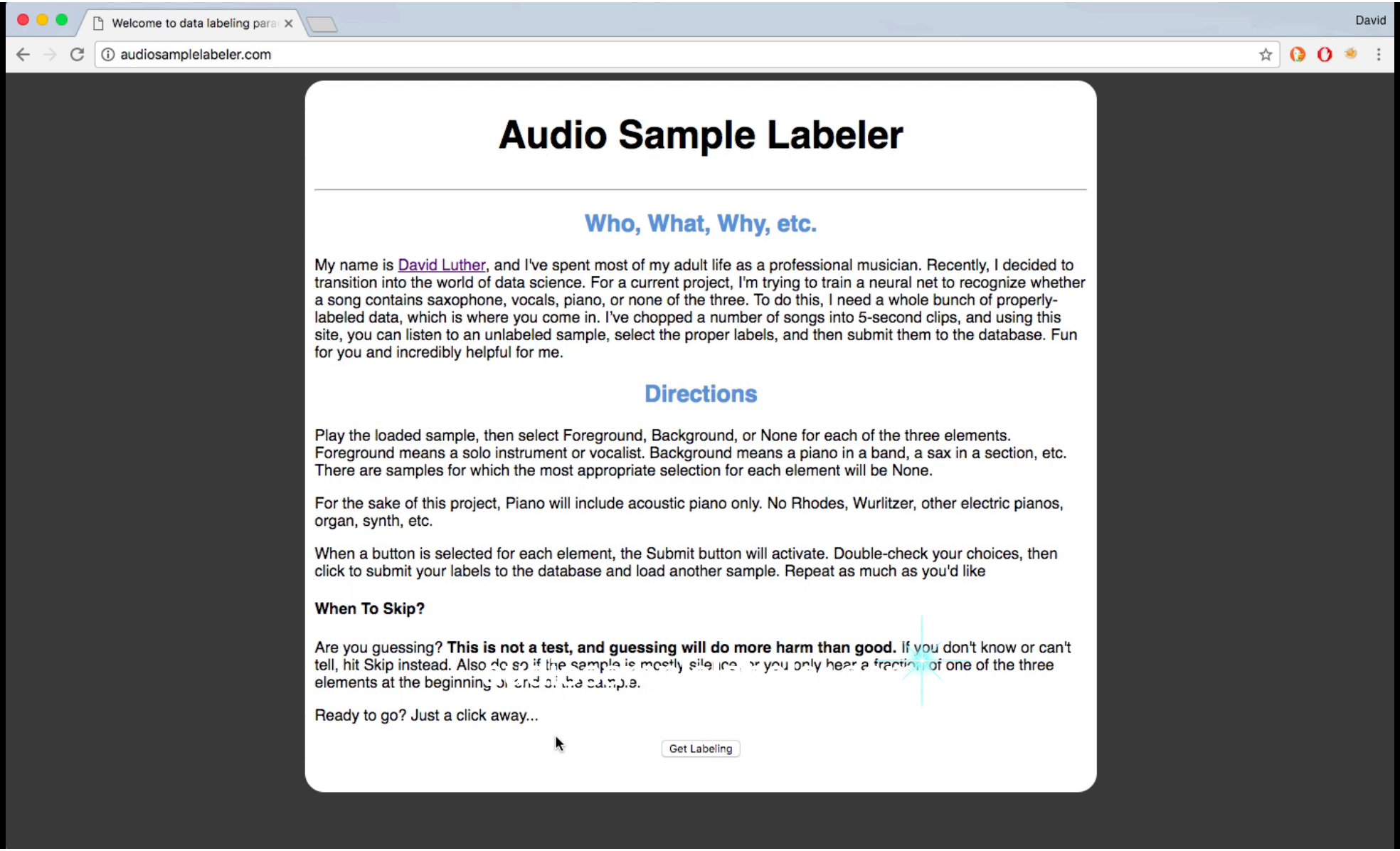
- 6705 excerpts in training set, 2874 in test
- 626 training excerpts with saxophone as primary instrument

DIY!

Creating and Storing Audio Samples

Python
+
Librosa

How to Label 24,000 Audio Samples?


Welcome to data labeling para
David
audiosamplelabeler.com
Audio Sample Labeler
Who, What, Why, etc.
My name is David Luther, and I've spent most of my adult life as a professional musician. Recently, I decided to transition into the world of data science. For a current project, I'm trying to train a neural net to recognize whether a song contains saxophone, vocals, piano, or none of the three. To do this, I need a whole bunch of properly-labeled data, which is where you come in. I've chopped a number of songs into 5-second clips, and using this site, you can listen to an unlabeled sample, select the proper labels, and then submit them to the database. Fun for you and incredibly helpful for me.
Directions
Play the loaded sample, then select Foreground, Background, or None for each of the three elements. Foreground means a solo instrument or vocalist. Background means a piano in a band, a sax in a section, etc. There are samples for which the most appropriate selection for each element will be None.
For the sake of this project, Piano will include acoustic piano only. No Rhodes, Wurlitzer, other electric pianos, organ, synth, etc.
When a button is selected for each element, the Submit button will activate. Double-check your choices, then click to submit your labels to the database and load another sample. Repeat as much as you'd like
When To Skip?
Are you guessing? This is not a test, and guessing will do more harm than good. If you don't know or can't tell, hit Skip instead. Also do so if the sample is mostly silence, or you only hear a fraction of one of the three elements at the beginning or end of the sample.
Ready to go? Just a click away...
Get Labeling

Audio + Neural Net?

Audio + Neural Net?

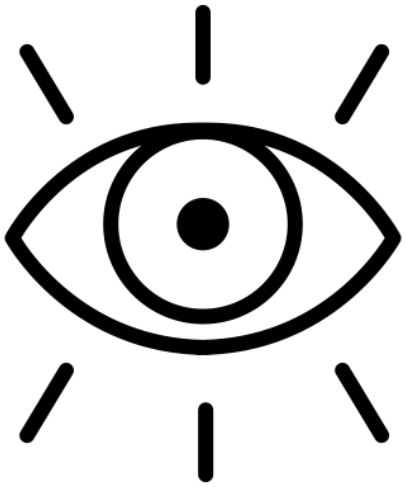
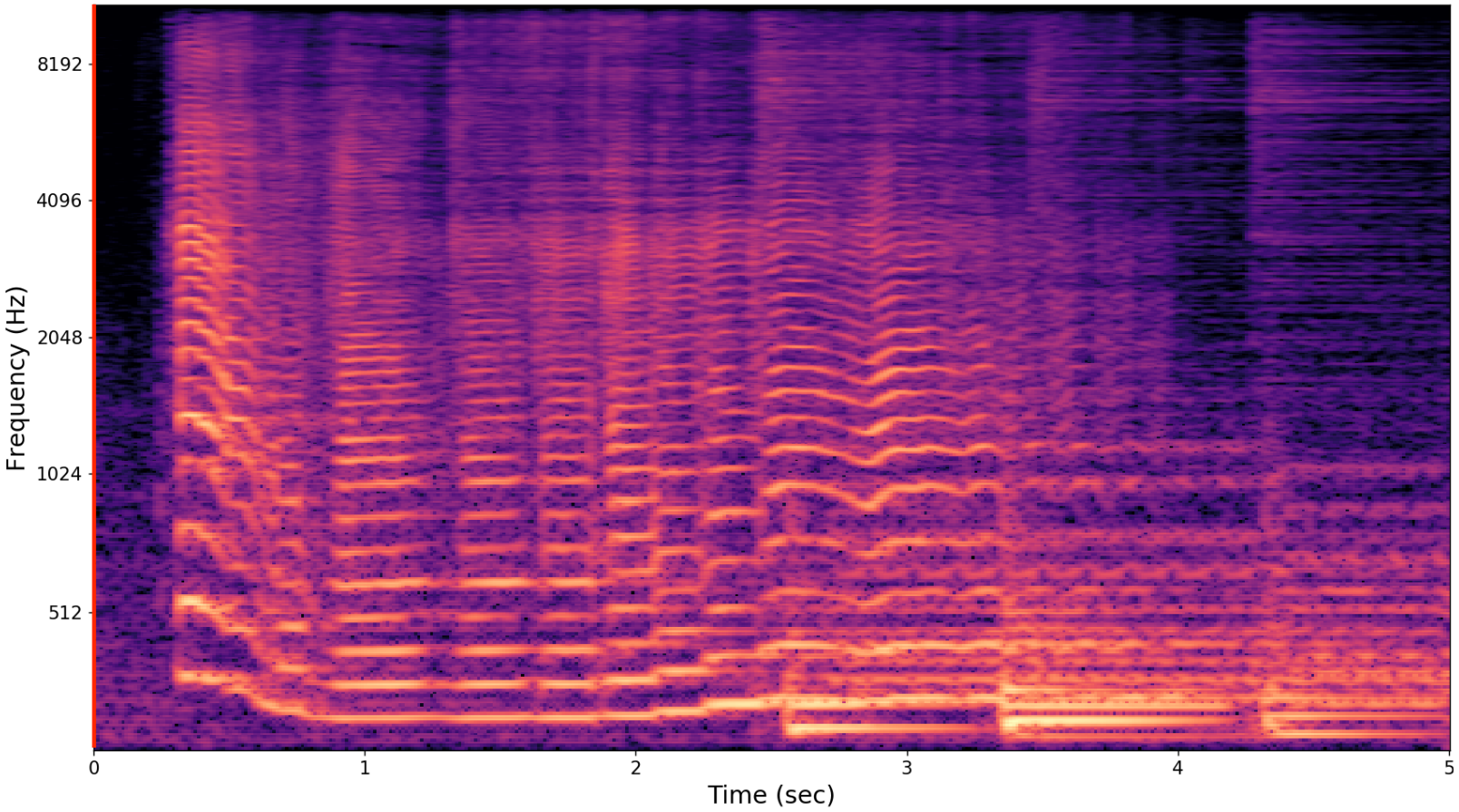


Image Recognition

Spectrograms



Spectrograms

Fast Fourier Transform Parameters

Mel Bands

- Number of frequency “bins”
- Y-axis resolution

Hop Length

- Number of samples between FFT calculations
- Inversely related to X-axis resolution (total samples / hop length)

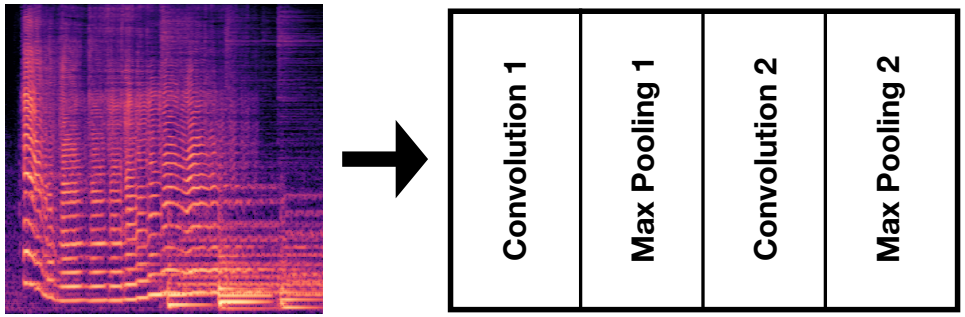
FFT Length

- Number of samples per FFT calculation (4x hop length)

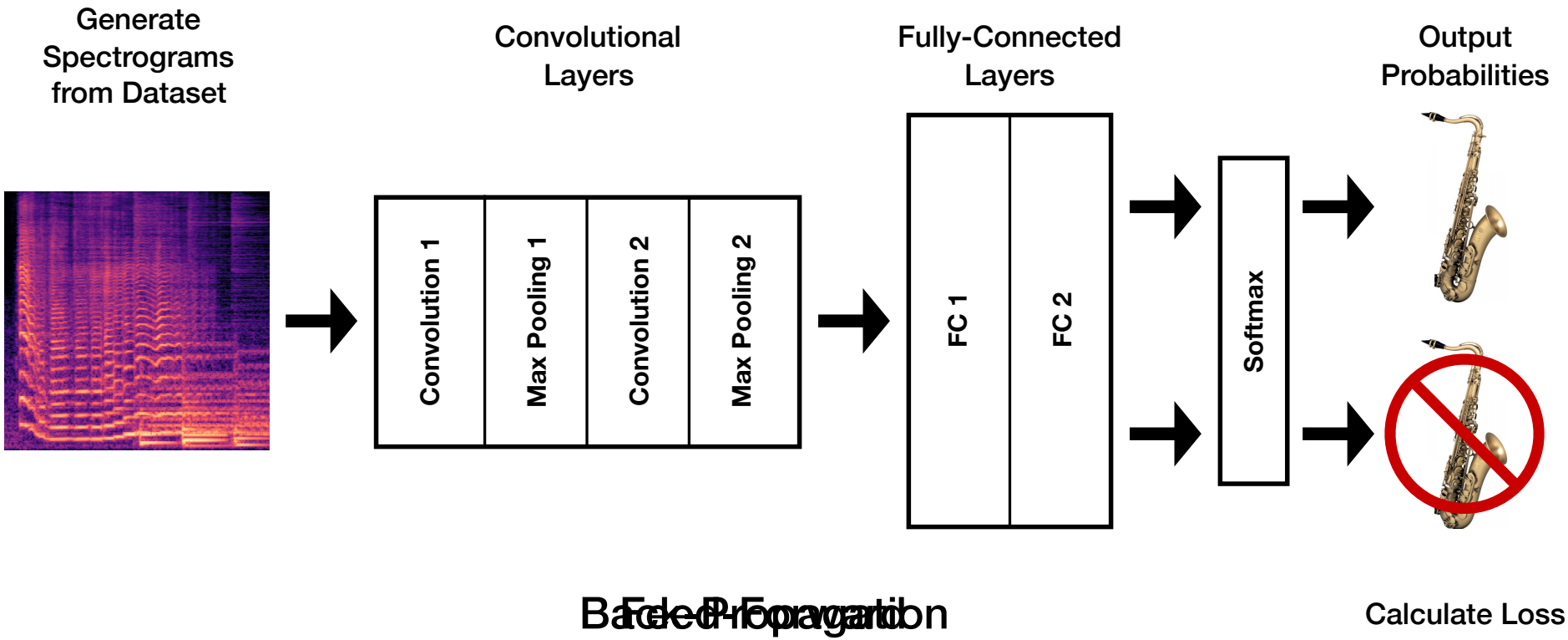
Convolutional Neural Networks

Generate Spectrograms from Dataset

Convolutional Layers



Convolutional Neural Networks



PyTorch: Data Loader & Dataset

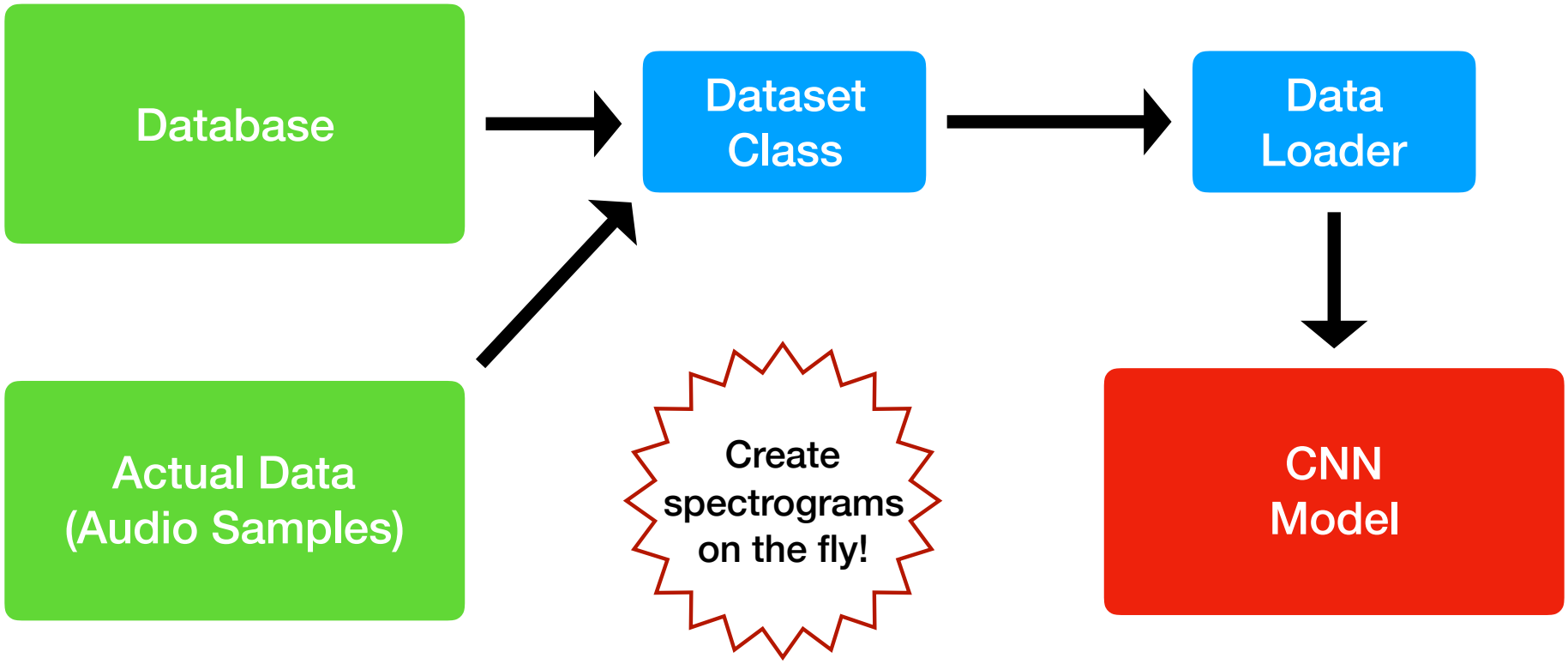
Data Loader

- Built-in class, creates an object used to feed data to model
- Specify batch size, shuffle between epochs, etc.
- Iterate through this object during fit/predict, dispatch data in batches

Dataset

- Self-authored class inheriting from provided parent class
- Passes data to the data loader

PyTorch: Dataset & Data Loader



PyTorch: Dataset Class

Setup

```
import torch.utils.data as data_utils

class SpectroDataset(data_utils.Dataset):

    def __init__(self):
        pass

    def __len__(self):
        pass

    def __getitem__(self, ix):
        pass
```

PyTorch: Dataset Class

`__init__()` method – arguments

```
class SpectroDataset(data_utils.Dataset):

    def __init__(
            self,
            datagroup_df,
            scaling=1,
            dir_in="../audio/wav_chunked",
            transform=None
        ):
```

PyTorch: Dataset Class

`__init__()` method – variables

```
class SpectroDataset(data_utils.Dataset):

    def __init__(
            self,
            datagroup_df,
            scaling=1,
            dir_in="../audio/wav_chunked",
            transform=None
        ):
        self.sample_frame = datagroup_df
        self.scaling = scaling
        self.audio_params = {
            'hl': 256,
            'n_fft': 1024,
            'n_mels': 512
        }
        self.dir_in = dir_in
        self.transform = transform
```

PyTorch: Dataset Class

`__len__()` method

```
class SpectroDataset(data_utils.Dataset):

    # __init__() method hidden

    def __len__(self):
        return self.sample_frame.shape[0]
```

PyTorch: Dataset Class

`__getitem__()` method

```
class SpectroDataset(data_utils.Dataset):

    # __init__(), __len__() methods hidden

    def __getitem__(self, ix):
```

PyTorch: Dataset Class

`__getitem__()` method – pull corresponding ID from data frame

```
class SpectroDataset(data_utils.Dataset):

    # __init__(), __len__() methods hidden

    def __getitem__(self, ix):
        chunk_id = self.sample_frame.loc[ix, 'chunk_id']
```

PyTorch: Dataset Class

`__getitem__()` method – load corresponding audio file

```
class SpectroDataset(data_utils.Dataset):

    # __init__(), __len__() methods hidden

    def __getitem__(self, ix):
        chunk_id = self.sample_frame.loc[ix, 'chunk_id']
        y, sr = audiomod.audio_loader(chunk_id)
```

PyTorch: Dataset Class

`__getitem__()` method – create spectrogram

```
class SpectroDataset(data_utils.Dataset):

    # __init__(), __len__() methods hidden

    def __getitem__(self, ix):
        chunk_id = self.sample_frame.loc[ix, 'chunk_id']
        y, sr = audiomod.audio_loader(chunk_id)
        sample = audiomod.make_spectro(
            y,
            sr,
            hl = int(self.audio_params['hl'] / self.scaling),
            n_fft = int(self.audio_params['n_fft'] / self.scaling),
            n_mels = int(self.audio_params['n_mels'] * self.scaling)
        )
```

PyTorch: Dataset Class

`__getitem__()` method – format for PyTorch

```
class SpectroDataset(data_utils.Dataset):

    # __init__(), __len__() methods hidden

    def __getitem__(self, ix):
        chunk_id = self.sample_frame.loc[ix, 'chunk_id']
        y, sr = audiomod.audio_loader(chunk_id)
        sample = audiomod.make_spectro(
            y,
            sr,
            hl = int(self.audio_params['hl'] / self.scaling),
            n_fft = int(self.audio_params['n_fft'] / self.scaling),
            n_mels = int(self.audio_params['n_mels'] * self.scaling)
        )
        sample = np.expand_dims(sample, 0) # add singleton dimension
        sample = normalize_spec(sample, low=-1) # normalize from -1 to 1
        sample = torch.from_numpy(sample).float() # turn into torch float tensor
```

PyTorch: Dataset Class

`__getitem__()` method – perform any transforms

```
class SpectroDataset(data_utils.Dataset):

    # __init__(), __len__() methods hidden

    def __getitem__(self, ix):
        chunk_id = self.sample_frame.loc[ix, 'chunk_id']
        y, sr = audiomod.audio_loader(chunk_id)
        sample = audiomod.make_spectro(
            y,
            sr,
            hl = int(self.audio_params['hl'] / self.scaling),
            n_fft = int(self.audio_params['n_fft'] / self.scaling),
            n_mels = int(self.audio_params['n_mels'] * self.scaling)
        )
        sample = np.expand_dims(sample, 0)
        sample = normalize_spec(sample, low=-1)
        sample = torch.from_numpy(sample).float()

        if self.transform:
            sample = self.transform(sample)
```

PyTorch: Dataset Class

`__getitem__()` method – return tuple of spectrogram, ground truth, ID

```
class SpectroDataset(data_utils.Dataset):

    # __init__(), __len__() methods hidden

    def __getitem__(self, ix):
        chunk_id = self.sample_frame.loc[ix, 'chunk_id']
        y, sr = audiomod.audio_loader(chunk_id)
        sample = audiomod.make_spectro(
            y,
            sr,
            hl = int(self.audio_params['hl'] / self.scaling),
            n_fft = int(self.audio_params['n_fft'] / self.scaling),
            n_mels = int(self.audio_params['n_mels'] * self.scaling)
        )
        sample = np.expand_dims(sample, 0)
        sample = normalize_spec(sample, low=-1)
        sample = torch.from_numpy(sample).float()

        if self.transform:
            sample = self.transform(sample)

        return sample, self.sample_frame.loc[ix, 'actual'], chunk_id
```

PyTorch: CNN Design

Convolutional Neural Net Class

- Self-authored class inheriting from provided parent class
- Define architecture
- Define feed-forward process

PyTorch: CNN Design

Setup

```
import torch.nn as nn
import torch.nn.functional as F

class CNN(nn.Module):

    def __init__(self):
        pass

    def forward(self, x):
        pass
```

PyTorch: CNN Design

`__init__()` method – setup

```
class CNN(nn.Module):

    def __init__(self, rs, normal=True):
        super(CNN, self).__init__()
        self.rs = rs
```

PyTorch: CNN Design

`__init__()` method – network architecture

```
class CNN(nn.Module):

    def __init__(self, rs, normal=True):
        super(CNN, self).__init__()
        self.rs = rs
        # in channels, out channels, kernel, stride=s
        self.conv1 = nn.Conv2d(1, 10, 5, stride=2)
        # 2x2 kernel, stride=2 -- stride defaults to kernel dims
        self.pool1 = nn.MaxPool2d(2, 2)
```

PyTorch: CNN Design

`__init__()` method – network architecture

```
class CNN(nn.Module):

    def __init__(self, rs, normal=True):
        super(CNN, self).__init__()
        self.rs = rs
        # in channels, out channels, kernel, stride=s
        self.conv1 = nn.Conv2d(1, 10, 5, stride=2)
        # 2x2 kernel, stride=2 -- stride defaults to kernel dims
        self.pool1 = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(10, 20, 5, stride=2)
        self.pool2 = nn.MaxPool2d(2, 2)
```

PyTorch: CNN Design

`__init__()` method – network architecture

```
class CNN(nn.Module):

    def __init__(self, rs, normal=True):
        super(CNN, self).__init__()
        self.rs = rs
        # in channels, out channels, kernel, stride=s
        self.conv1 = nn.Conv2d(1, 10, 5, stride=2)
        # 2x2 kernel, stride=2 -- stride defaults to kernel dims
        self.pool1 = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(10, 20, 5, stride=2)
        self.pool2 = nn.MaxPool2d(2, 2)
        # in channels, out channels
        self.fc1 = nn.Linear(700, 100)
        self.fc2 = nn.Linear(100, 2)
```

PyTorch: CNN Design

`__init__()` method – normal initialization (optional)

```
class CNN(nn.Module):

    def __init__(self, rs, normal=True):
        super(CNN, self).__init__()
        self.rs = rs
        # in channels, out channels, kernel, stride=s
        self.conv1 = nn.Conv2d(1, 10, 5, stride=2)
        # 2x2 kernel, stride=2 -- stride defaults to kernel dims
        self.pool1 = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(10, 20, 5, stride=2)
        self.pool2 = nn.MaxPool2d(2, 2)
        # in channels, out channels
        self.fc1 = nn.Linear(700, 100)
        self.fc2 = nn.Linear(100, 2)
        if normal:
            self.apply(init_norm_auto)
```

PyTorch: CNN Design

`forward()` method

```
class CNN(nn.Module):

    # __init__() method hidden

    def forward(self, x):
```

PyTorch: CNN Design

`forward()` method – convolution

```
class CNN(nn.Module):

    # __init__() method hidden

    def forward(self, x):
        x = self.pool1(F.relu(self.conv1(x)))
        x = self.pool2(F.relu(self.conv2(x)))
```

PyTorch: CNN Design

`forward()` method – reshape

```
class CNN(nn.Module):

    # __init__() method hidden

    def forward(self, x):
        x = self.pool1(F.relu(self.conv1(x)))
        x = self.pool2(F.relu(self.conv2(x)))
        x = x.view(x.size(0), -1) # reshape for fully connected layer
```

PyTorch: CNN Design

`forward()` method – fully-connected layers

```
class CNN(nn.Module):

    # __init__() method hidden

    def forward(self, x):
        x = self.pool1(F.relu(self.conv1(x)))
        x = self.pool2(F.relu(self.conv2(x)))
        x = x.view(x.size(0), -1) # reshape for fully connected layer
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
```

PyTorch: CNN Design

`forward()` method – softmax and return

```
class CNN(nn.Module):

    # __init__() method hidden

    def forward(self, x):
        x = self.pool1(F.relu(self.conv1(x)))
        x = self.pool2(F.relu(self.conv2(x)))
        x = x.view(x.size(0), -1) # reshape for fully connected layer
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = nn.Softmax(x)
        return x
```

PyTorch: Fit Function

- Feed forward / back propagation to train model
- Provide optimizer (SGD, RMSprop, Adam, etc.)
- Provide loss criterion (Cross Entropy Loss, etc.)
- Set number of training epochs (or other stop criterion)

PyTorch: Fit Function

Setup

```
from torch.autograd import Variable

def fit(
    cnn,
    dataset,
    optimizer,
    criterion,
    num_epochs,
    batch_size
    ):
```

PyTorch: Fit Function

Create data loader object

```
def fit(
    cnn,
    dataset,
    optimizer,
    criterion,
    num_epochs,
    batch_size
    ):

    train_loader = data_utils.DataLoader(
        dataset,
        batch_size=batch_size,
        shuffle=True,
        num_workers=2,
        drop_last=True
    )
```

PyTorch: Fit Function

Loss by epoch

```
def fit(
    cnn,
    dataset,
    optimizer,
    criterion,
    num_epochs,
    batch_size
    ):

    train_loader = data_utils.DataLoader(
        dataset,
        batch_size=batch_size,
        shuffle=True,
        num_workers=2,
        drop_last=True
    )

    loss_by_epoch = []
```

PyTorch: Fit Function

Epoch loops

```
def fit(...):
    # data loader hidden

    for epoch in range(num_epochs):
        print("Epoch", epoch+1) # verbosity
        running_loss = 0.0
        loss_per_batch = np.array([])
        then = time.perf_counter()
```

PyTorch: Fit Function

Loop through loader object

```
def fit(...):
    # data loader hidden

    for epoch in range(num_epochs):
        print("Epoch", epoch+1) # verbosity
        running_loss = 0.0
        loss_per_batch = np.array([])
        then = time.perf_counter()
        for i, data in enumerate(train_loader, 1):
            sub_then = time.perf_counter()
```

PyTorch: Fit Function

Unpack minibatch of data

```
def fit(...):
    # data loader hidden

    for epoch in range(num_epochs):
        print("Epoch", epoch+1) # verbosity
        running_loss = 0.0
        loss_per_batch = np.array([])
        then = time.perf_counter()
        for i, data in enumerate(train_loader, 1):
            sub_then = time.perf_counter()
            spectros, labels, _ = data
```

PyTorch: Fit Function

Wrap spectrograms and labels in `Variable()`

```
def fit(...):
    # data loader hidden

    for epoch in range(num_epochs):
        print("Epoch", epoch+1) # verbosity
        running_loss = 0.0
        loss_per_batch = np.array([])
        then = time.perf_counter()
        for i, data in enumerate(train_loader, 1):
            sub_then = time.perf_counter()
            spectros, labels, _ = data
            spectros, labels = Variable(spectros), Variable(labels)
```

PyTorch: Fit Function

Zero out gradients

```
def fit(...):
    # data loader hidden

    for epoch in range(num_epochs):
        print("Epoch", epoch+1) # verbosity
        running_loss = 0.0
        loss_per_batch = np.array([])
        then = time.perf_counter()
        for i, data in enumerate(train_loader, 1):
            sub_then = time.perf_counter()
            spectros, labels, _ = data
            spectros, labels = Variable(spectros), Variable(labels)
            optimizer.zero_grad()
```

PyTorch: Fit Function

Feed-forward

```
def fit(...):
    # data loader hidden

    for epoch in range(num_epochs):
        print("Epoch", epoch+1) # verbosity
        running_loss = 0.0
        loss_per_batch = np.array([])
        then = time.perf_counter()
        for i, data in enumerate(train_loader, 1):
            sub_then = time.perf_counter()
            spectros, labels, _ = data
            spectros, labels = Variable(spectros), Variable(labels)
            optimizer.zero_grad()
            outputs = cnn(spectros)
```

PyTorch: Fit Function

Calculate minibatch loss

```
def fit(...):
    # data loader hidden

    for epoch in range(num_epochs):
        print("Epoch", epoch+1) # verbosity
        running_loss = 0.0
        loss_per_batch = np.array([])
        then = time.perf_counter()
        for i, data in enumerate(train_loader, 1):
            sub_then = time.perf_counter()
            spectros, labels, _ = data
            spectros, labels = Variable(spectros), Variable(labels)
            optimizer.zero_grad()
            outputs = cnn(spectros)
            loss = criterion(outputs, labels)
```

PyTorch: Fit Function

Back-propagation

```
def fit(...):
    # data loader hidden

    for epoch in range(num_epochs):
        print("Epoch", epoch+1) # verbosity
        running_loss = 0.0
        loss_per_batch = np.array([])
        then = time.perf_counter()
        for i, data in enumerate(train_loader, 1):
            sub_then = time.perf_counter()
            spectros, labels, _ = data
            spectros, labels = Variable(spectros), Variable(labels)
            optimizer.zero_grad()
            outputs = cnn(spectros)
            loss = criterion(outputs, labels)
            loss.backward()
```

PyTorch: Fit Function

Update weights

```
def fit(...):
    # data loader hidden

    for epoch in range(num_epochs):
        print("Epoch", epoch+1) # verbosity
        running_loss = 0.0
        loss_per_batch = np.array([])
        then = time.perf_counter()
        for i, data in enumerate(train_loader, 1):
            sub_then = time.perf_counter()
            spectros, labels, _ = data
            spectros, labels = Variable(spectros), Variable(labels)
            optimizer.zero_grad()
            outputs = cnn(spectros)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
```

PyTorch: Fit Function

Verbosty (minibatch loss and time)

```
def fit(...):
    # data loader hidden

    for epoch in range(num_epochs):
        print("Epoch", epoch+1) # verbosity
        running_loss = 0.0
        loss_per_batch = np.array([])
        then = time.perf_counter()
        for i, data in enumerate(train_loader, 1):
            sub_then = time.perf_counter()
            spectros, labels, _ = data
            spectros, labels = Variable(spectros), Variable(labels)
            optimizer.zero_grad()
            outputs = cnn(spectros)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            # verbosity
            sub_now = time.perf_counter()
            print("\r * Batch {} loss: {:.3f}\tTime: {:.3f} ms"
                  .format(i, loss.data[0], (sub_now-sub_then)*1000), end='')
            loss_per_batch = np.append(loss_per_batch, loss.data[0])
            running_loss += loss.data[0]
```

PyTorch: Fit Function

Record/report loss figures

```
def fit(...):
    # data loader hidden

    for epoch in range(num_epochs):
        print("Epoch", epoch+1) # verbosity
        running_loss = 0.0
        loss_per_batch = np.array([])
        then = time.perf_counter()
        for i, data in enumerate(train_loader, 1):
            # minibatch loop hidden

        now = time.perf_counter()
        avg_loss = running_loss / i
        loss_by_epoch.append(loss_per_batch)
        print("\r * Avg loss: {:.3f}\tTime: {:.3f} ms"
              .format(running_loss / i, (now - then) * 1000))
```

PyTorch: Fit Function

Return loss array

```
def fit(...):
    # data loader hidden

    for epoch in range(num_epochs):
        print("Epoch", epoch+1) # verbosity
        running_loss = 0.0
        loss_per_batch = np.array([])
        then = time.perf_counter()
        for i, data in enumerate(train_loader, 1):
            # minibatch loop hidden

        now = time.perf_counter()
        avg_loss = running_loss / i
        loss_by_epoch.append(loss_per_batch)
        print("\r * Avg loss: {:.3f}\tTime: {:.3f} ms"
              .format(running_loss / i, (now - then) * 1000))

    print('\n\aTraining Complete')

    return np.vstack(loss_by_epoch)
```

PyTorch: Fit Function

In [*]:
```
fit(cnn_test,
    train_sub,
    optim.SGD(cnn_test.parameters(), lr=0.01),
    nn.CrossEntropyLoss(),
    10)
```

PyTorch: Predict Function

- Feed forward for train/test data
- Output probabilities of each category
- Translate probabilities to prediction

PyTorch: Predict Function

Setup

```
def predict(cnn, dataset, batch_size=4, res_format='df'):
```

PyTorch: Predict Function

Create data loader object

```
def predict(cnn, dataset, batch_size=4, res_format='df'):

    loader = data_utils.DataLoader(
        dataset,
        batch_size=batch_size,
        shuffle=False, # set to False for predict
        num_workers=2,
        drop_last=False # set to False for predict
    )
```

PyTorch: Predict Function

Loop through loader object, unpack minibatches

```
def predict(cnn, dataset, batch_size=4, res_format='df'):

    loader = data_utils.DataLoader(
        dataset,
        batch_size=batch_size,
        shuffle=False, # set to False for predict
        num_workers=2,
        drop_last=False # set to False for predict
    )

    results = {}

    for data in loader:
        spectros, labels, chunk_ids = data
```

PyTorch: Predict Function

Feed minibatch through model

```
def predict(cnn, dataset, batch_size=4, res_format='df'):

    loader = data_utils.DataLoader(
        dataset,
        batch_size=batch_size,
        shuffle=False, # set to False for predict
        num_workers=2,
        drop_last=False # set to False for predict
    )

    results = {}

    for data in loader:
        spectros, labels, chunk_ids = data
        outputs = cnn(Variable(spectros)) # feed forward
```

PyTorch: Predict Function

Calculate prediction

```
def predict(cnn, dataset, batch_size=4, res_format='df'):

    loader = data_utils.DataLoader(
        dataset,
        batch_size=batch_size,
        shuffle=False, # set to False for predict
        num_workers=2,
        drop_last=False # set to False for predict
    )

    results = {}

    for data in loader:
        spectros, labels, chunk_ids = data
        outputs = cnn(Variable(spectros)) # feed forward
        _, pred = torch.max(outputs.data, 1) # calculate prediction
```

PyTorch: Predict Function

Add results to dictionary with ID as key

```
def predict(cnn, dataset, batch_size=4, res_format='df'):

    loader = data_utils.DataLoader(
        dataset,
        batch_size=batch_size,
        shuffle=False, # set to False for predict
        num_workers=2,
        drop_last=False # set to False for predict
    )

    results = {}

    for data in loader:
        spectros, labels, chunk_ids = data
        outputs = cnn(Variable(spectros)) # feed forward
        _, pred = torch.max(outputs.data, 1) # calculate prediction
        for c_id, y, y_hat, out in zip(chunk_ids, labels, pred, outputs.data):
            results[c_id] = (y, y_hat, out)
```

PyTorch: Predict Function

Convert to pandas dataframe and return

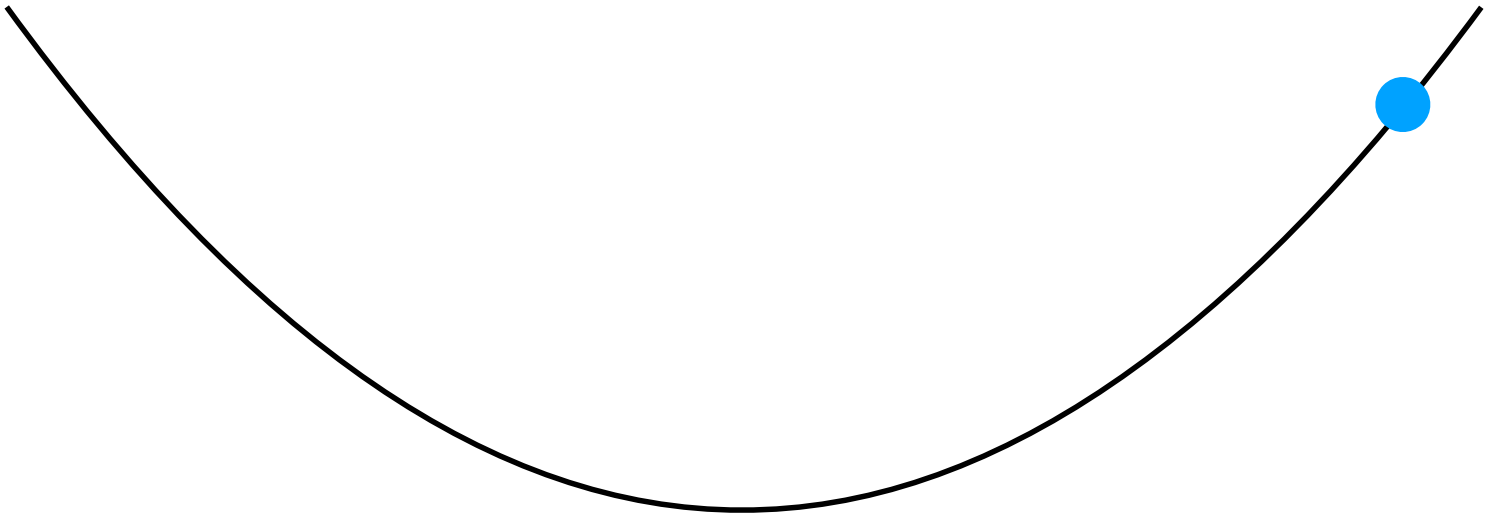
```
def predict(cnn, dataset, batch_size=4, res_format='df'):

    loader = data_utils.DataLoader(
        dataset,
        batch_size=batch_size,
        shuffle=False, # set to False for predict
        num_workers=2,
        drop_last=False # set to False for predict
    )

    results = {}

    for data in loader:
        spectros, labels, chunk_ids = data
        outputs = cnn(Variable(spectros)) # feed forward
        _, pred = torch.max(outputs.data, 1) # calculate prediction
        for c_id, y, y_hat, out in zip(chunk_ids, labels, pred, outputs.data):
            results[c_id] = (y, y_hat, out)

    if res_format == 'df':
        results = results_to_df(results)

    return results
```

Model Design & Training

Early Fit Attempts

In [*]:
```
fit(cnn_test,
    train_sub,
    optim.SGD(cnn_test.parameters(), lr=0.01),
    nn.CrossEntropyLoss(),
    10)
```

Ideal Loss Function



My Loss Function

An approximation…

1. Adjust initialization settings
2. Adjust step size at beginning
3. Increase batch size
4. Optimize number of training epochs

Cross Validation

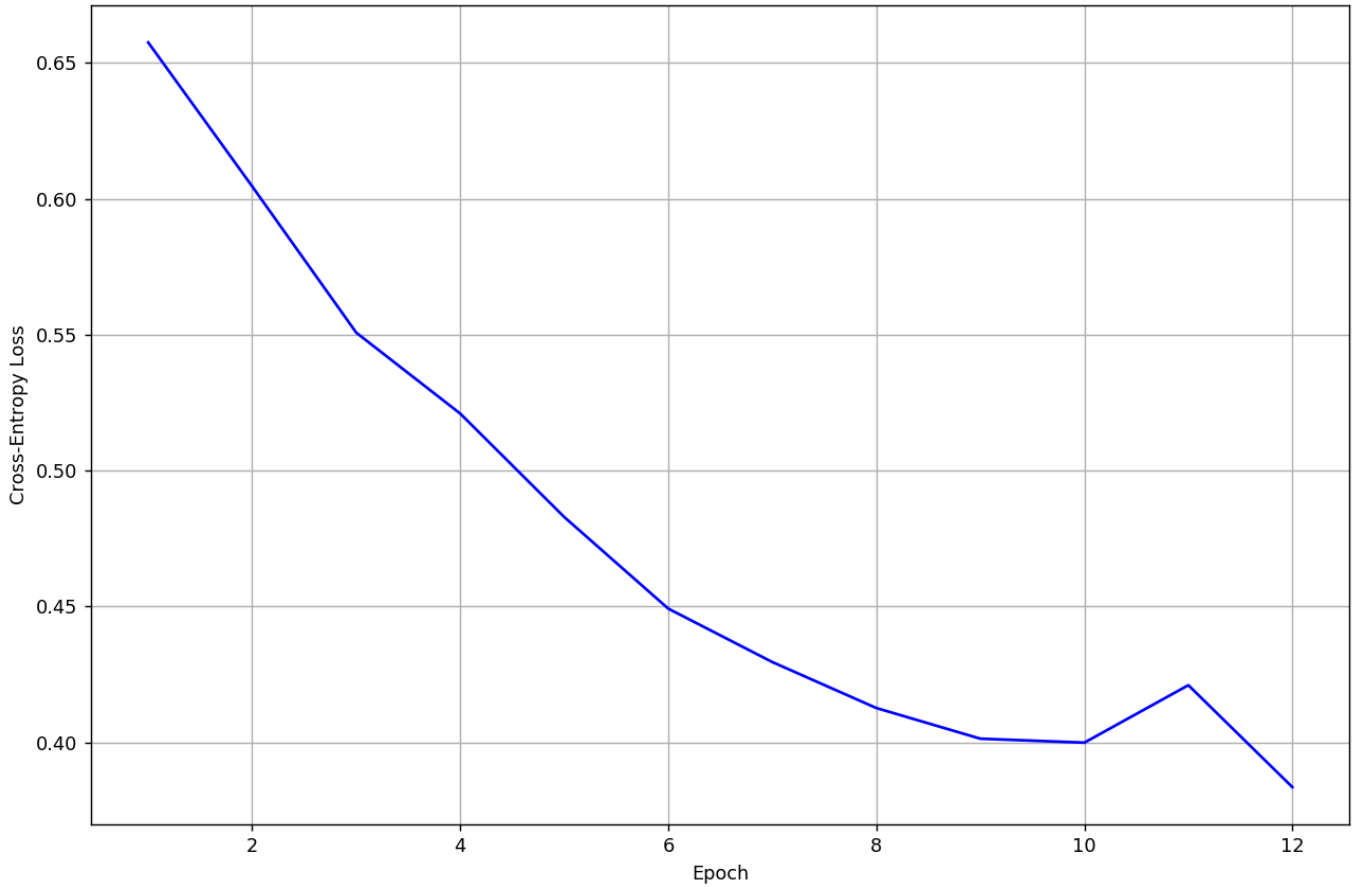
Or “How I Escaped Death Valley And Avoided The Abyss”

No built-in CV utility — roll your own!

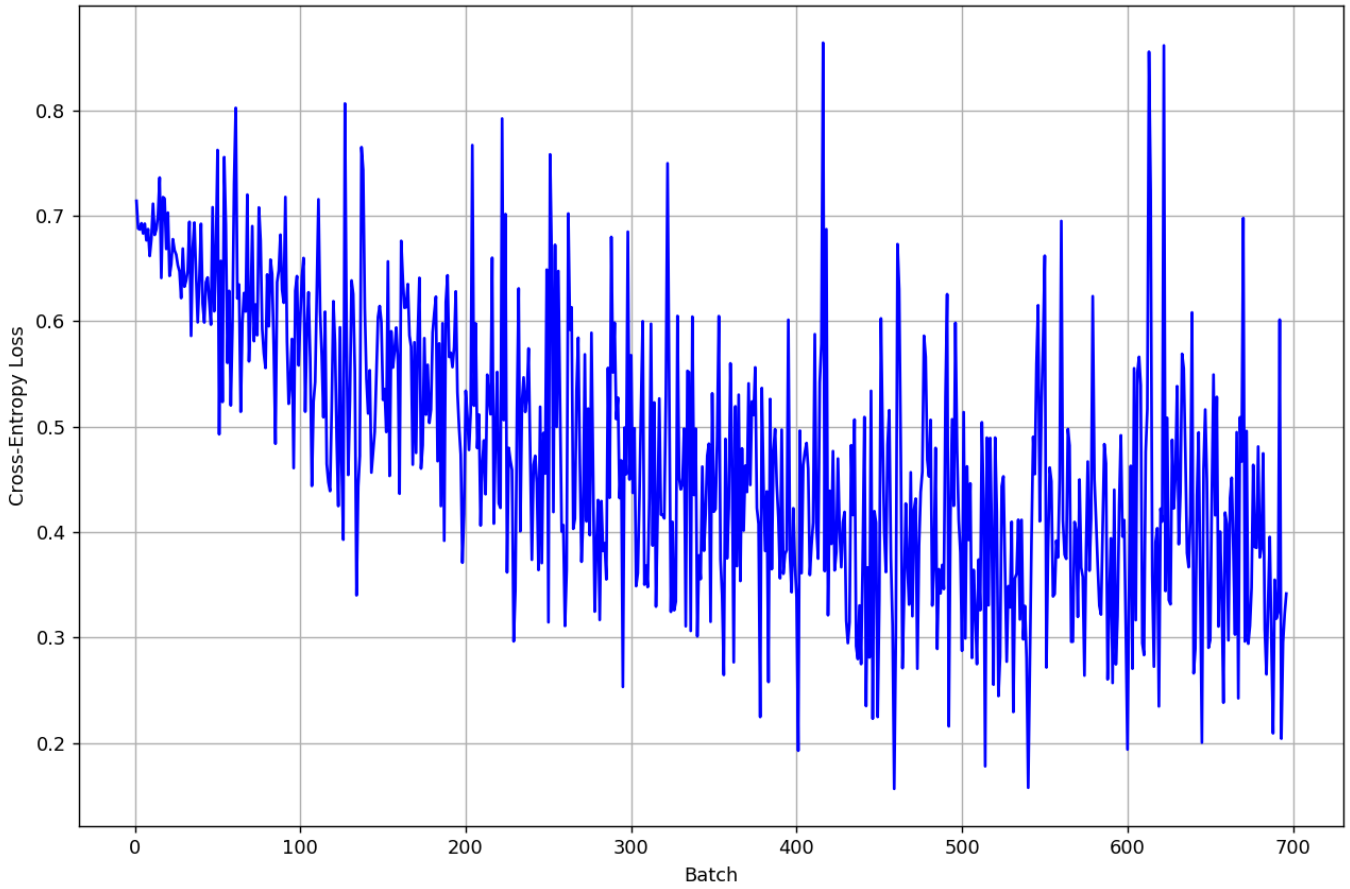
Parameters Tuned

- Initialized weights from normal distribution instead of uniform
- Minibatch size of 16
- Adam optimization method for momentum, dynamic learning rate

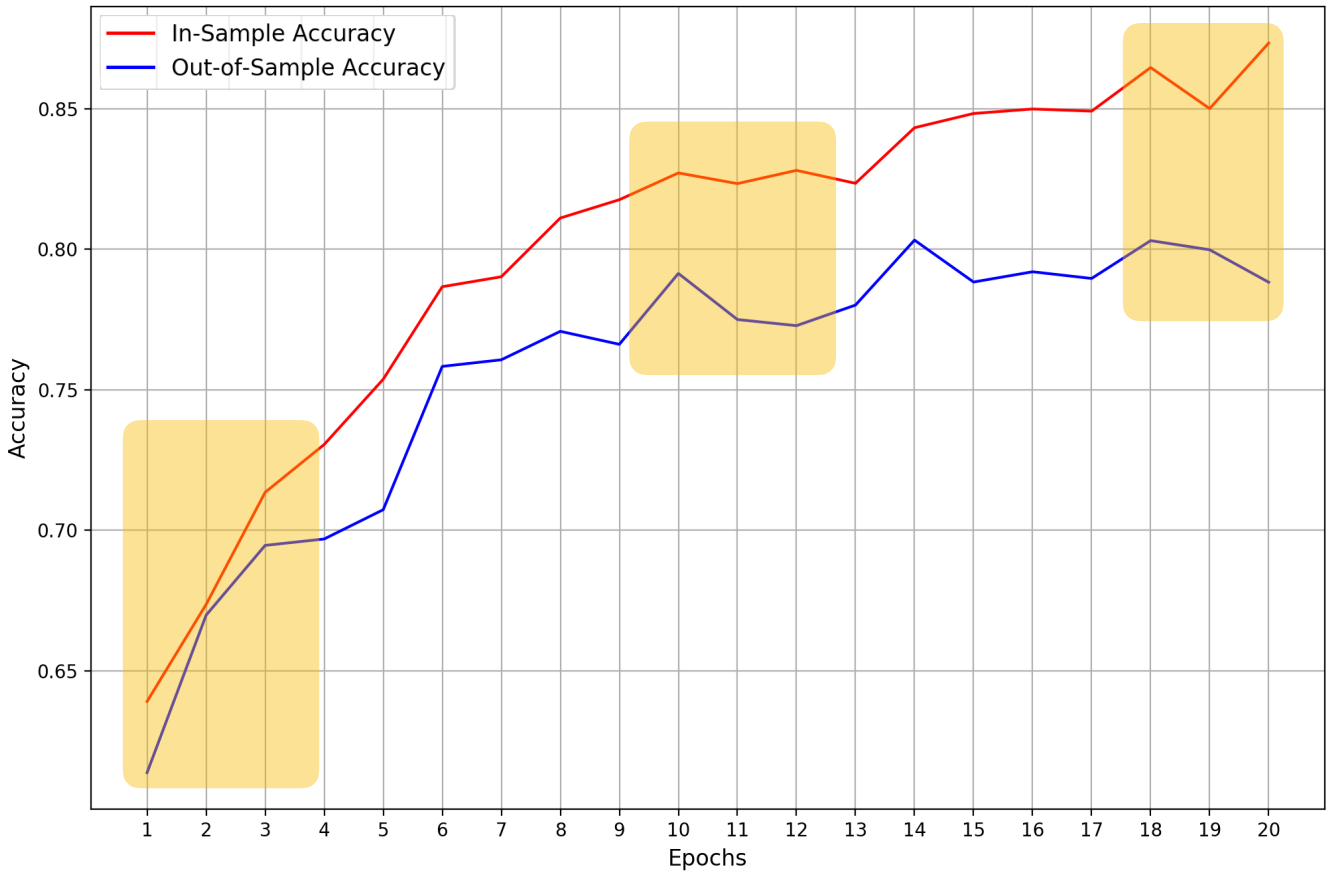
Cross-Entropy Loss (by Epoch)



Cross-Entropy Loss (by Minibatch)



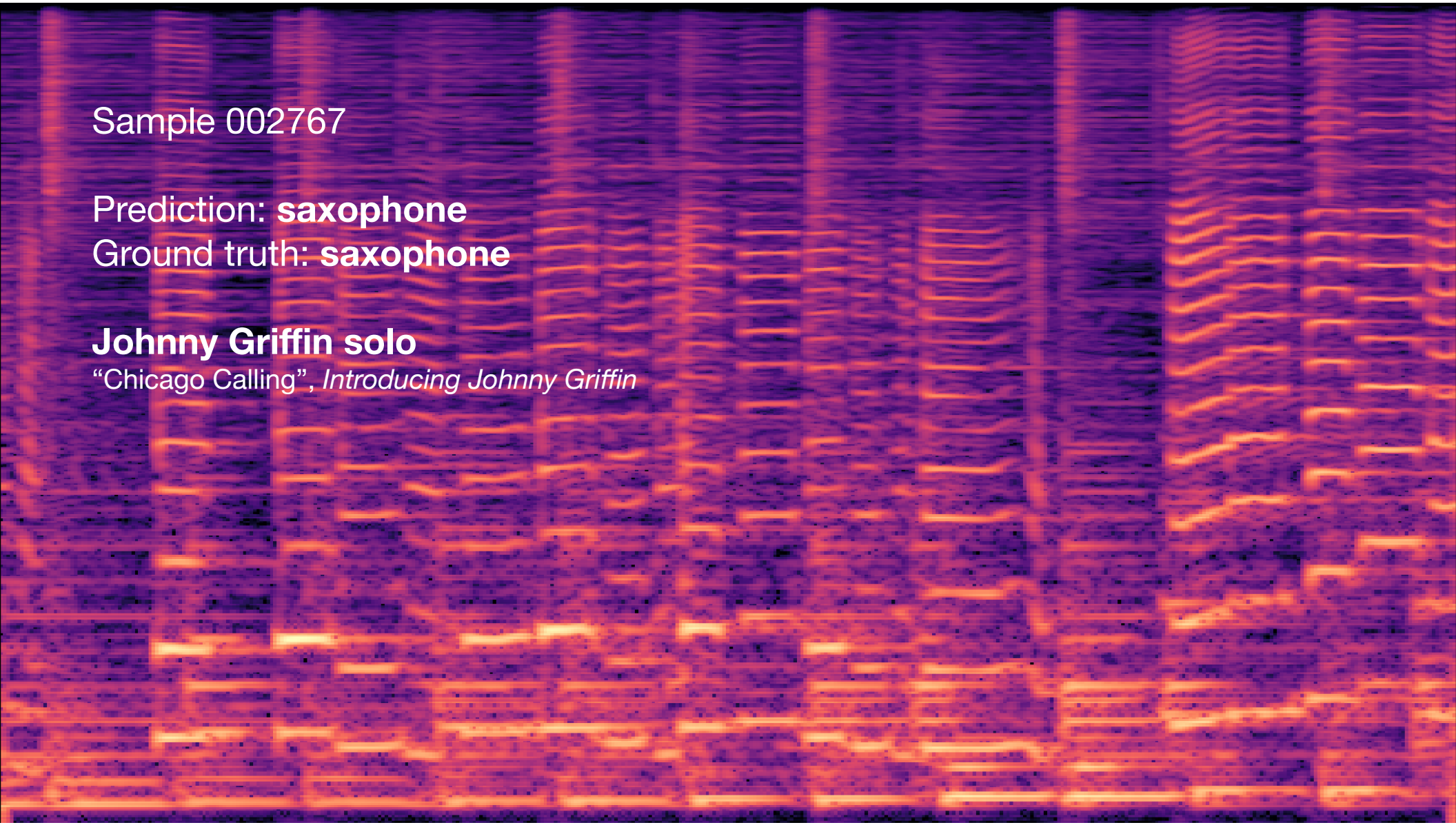
Model Accuracy by Training Epoch



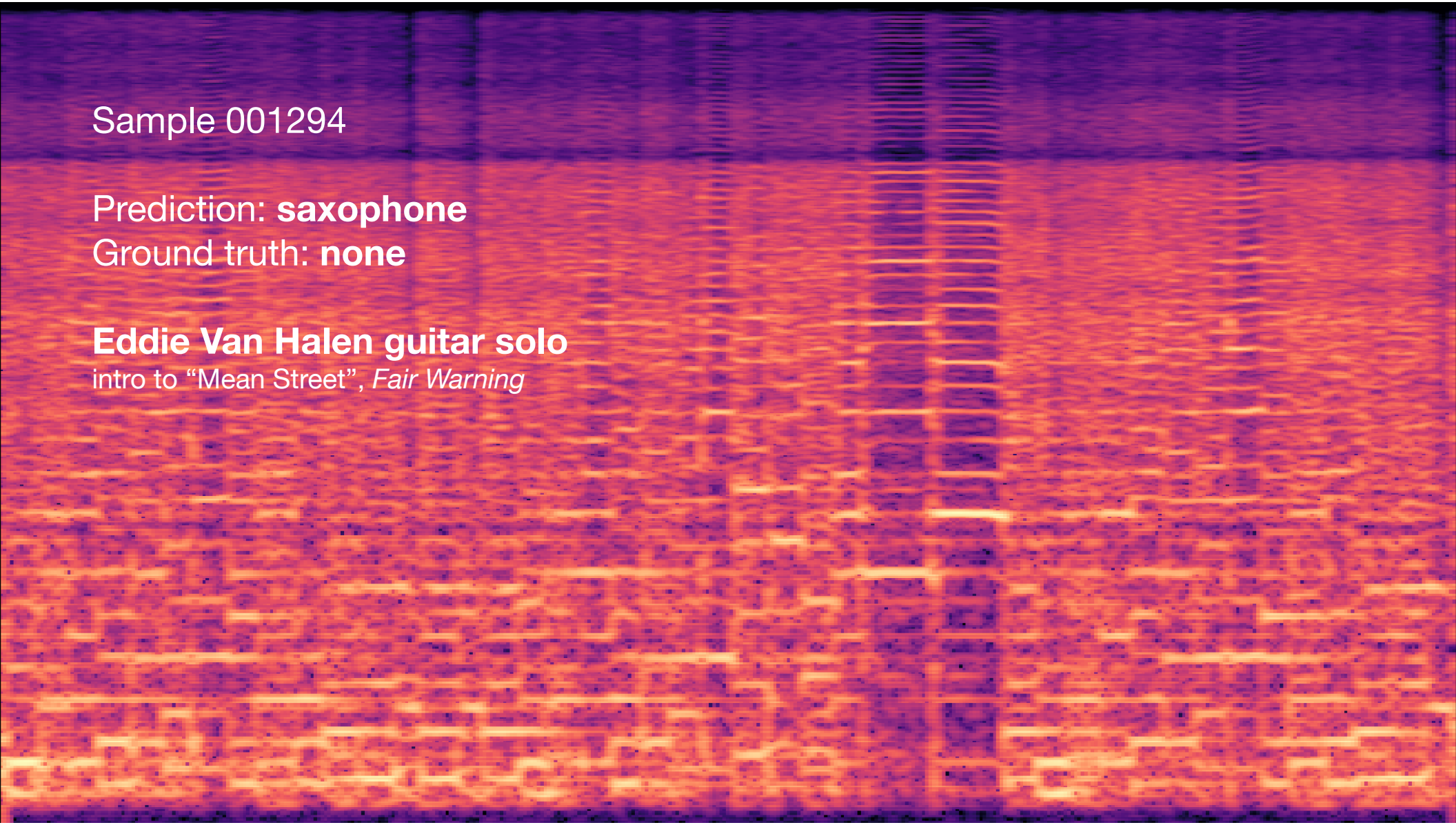
Train/Test Scores

Score	Train	Test

What Did The Model Get Right?


Sample 002767
Prediction: **saxophone**
Ground truth: **saxophone**
Johnny Griffin solo
“Chicago Calling”, *Introducing Johnny Griffin*

And Where Did It Miss?


Sample 001294
Prediction: **saxophone**
Ground truth: **none**
Eddie Van Halen guitar solo
intro to “Mean Street”, *Fair Warning*

Next Steps

1. Tune CNN architecture to increase accuracy, address overfitting
2. PyTorch transfer learning
3. Try with more labeled data
4. Move to GPUs on AWS

David Luther

✉ davidrluther@gmail.com

linkedin.com/in/davidrluther

github.com/davidluther

Thank You!