Lab 4: PyTorch Essentials

Duration: 3-5 minutes **Presenter Guide:** This lab introduces PyTorch for deep learning on audio spectrograms

---

Slide 1: Lab Overview (30 seconds)

Title: "Lab 4: PyTorch Essentials"

Subtitle: "From NumPy Arrays to GPU-Accelerated Tensors"

What you'll learn: - PyTorch tensors basics - NumPy → PyTorch conversion - GPU acceleration - Simple neural network operations

Team Member Connections: - **Ryan:** Completed "Learn the Basics" - tensors, DataLoader, autograd - **Yovannoa:** Completed "60 Minute Blitz" - full neural network training - **cervanj2's Architecture:** PyTorch powers the Inference Layer (U-Net)

Slide 2: Why PyTorch for Audio ML? (45 seconds)

Title: "PyTorch: The Deep Learning Framework"

Why PyTorch? 1. **GPU Acceleration:** 10-100x faster than CPU for neural networks 2. **Automatic Differentiation:** Calculates gradients automatically for training 3. **Research-Friendly:** Easy to experiment and modify models 4. **U-Net Implementation:** Our forked Pytorch-UNet repo uses PyTorch

The Learning Path: - **Ryan:** Learned PyTorch basics with Fashion-MNIST (images) - **Yovannoa:** Trained classifier on CIFAR-10 (images) - **Our Project:** Apply same PyTorch concepts to audio spectrograms

Key Insight: "Spectrograms are just 2D images! Image classification concepts (from Ryan and Yovannoa) transfer directly to audio separation."

Slide 3: PyTorch Tensors - Hello World (1 minute 30 seconds)

Title: "Your First PyTorch Tensors"

Code Example:

```python
import torch
import numpy as np

print("=== PyTorch Setup ===")
print(f"PyTorch version: {torch.__version__}")
print(f"CUDA available: {torch.cuda.is_available()}")

if torch.cuda.is_available():
    print(f"CUDA version: {torch.version.cuda}")
    print(f"GPU: {torch.cuda.get_device_name(0)}")

# Choose device (GPU if available, CPU otherwise)
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(f"\nUsing device: {device}")

# Create a tensor (simulating a spectrogram)
# Shape: 1025 freq bins × 646 time frames (from Lab 3)
spectrogram_data = np.random.randn(1025, 646).astype(np.float32)
print(f"\nNumPy array shape: {spectrogram_data.shape}")
print(f"NumPy array dtype: {spectrogram_data.dtype}")

# Convert to PyTorch tensor
tensor = torch.from_numpy(spectrogram_data)
print(f"\nPyTorch tensor shape: {tensor.shape}")
print(f"PyTorch tensor dtype: {tensor.dtype}")
print(f"Tensor device: {tensor.device}")  # Initially on CPU

# Move to GPU (if available)
tensor_gpu = tensor.to(device)
print(f"\nAfter .to(device):")
print(f"Tensor device: {tensor_gpu.device}")

# Basic operations on GPU
tensor_processed = torch.relu(tensor_gpu)  # ReLU activation
tensor_scaled = tensor_processed * 0.5     # Scaling

print(f"\nProcessed tensor shape: {tensor_processed.shape}")
print(f"Processed tensor device: {tensor_processed.device}")

# Move back to CPU for saving/visualization
result = tensor_processed.cpu().numpy()
print(f"\nBack to NumPy: {result.shape}, {result.dtype}")
```

Expected Output (with GPU):

```
=== PyTorch Setup ===
PyTorch version: 2.0.0
CUDA available: True
CUDA version: 12.1
GPU: NVIDIA GeForce RTX 2070 Super Max-Q

Using device: cuda

NumPy array shape: (1025, 646)
NumPy array dtype: float32

PyTorch tensor shape: torch.Size([1025, 646])
PyTorch tensor dtype: torch.float32
Tensor device: cpu

After .to(device):
Tensor device: cuda:0

Processed tensor shape: torch.Size([1025, 646])
Processed tensor device: cuda:0

Back to NumPy: (1025, 646), float32
```

Key Concepts: - Tensor = PyTorch's version of NumPy array - `.to(device)` moves tensor to GPU - GPU operations are much faster for neural networks - Easy conversion: NumPy ↔ PyTorch

---

Slide 4: The Complete Audio-to-Tensor Pipeline (1 minute 15 seconds)

Title: “Bringing It All Together: Labs 1-4”

Complete Code Example:

```python
import librosa
import numpy as np
import torch

# Device setup
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(f"Processing on: {device}")

# Step 1: Load audio (Lab 1)
audio, sr = librosa.load('sample_audio.wav', sr=22050)
print(f"1. Audio loaded: {audio.shape}")

# Step 2: NumPy ready (Lab 2)
print(f"2. NumPy array ready: dtype={audio.dtype}")

# Step 3: Create spectrogram with librosa (Lab 3)
stft = librosa.stft(audio, n_fft=2048, hop_length=512)
magnitude = np.abs(stft)
print(f"3. Spectrogram created: {magnitude.shape}")

# Step 4: Convert to PyTorch tensor (Lab 4)
# Add batch and channel dimensions for U-Net
# U-Net expects shape: (batch_size, channels, freq, time)
tensor = torch.from_numpy(magnitude).float()
tensor = tensor.unsqueeze(0).unsqueeze(0)  # Add batch and channel dims
tensor = tensor.to(device)

print(f"4. PyTorch tensor ready: {tensor.shape}")
print(f"   Shape breakdown:")
print(f"   - Batch size: {tensor.shape[0]}")
print(f"   - Channels: {tensor.shape[1]}")
print(f"   - Frequency bins: {tensor.shape[2]}")
print(f"   - Time frames: {tensor.shape[3]}")
print(f"   - Device: {tensor.device}")

# Ready for U-Net processing!
print(f"\nTensor is now ready for U-Net!")
```

Expected Output:

```
Processing on: cuda

1. Audio loaded: (661500,)
2. NumPy array ready: dtype=float32
3. Spectrogram created: (1025, 646)
4. PyTorch tensor ready: torch.Size([1, 1, 1025, 646])
   Shape breakdown:
   - Batch size: 1
   - Channels: 1
   - Frequency bins: 1025
   - Time frames: 646
   - Device: cuda:0

Tensor is now ready for U-Net!
```

The Full Pipeline:

```
Audio file
   ↓ [Lab 1: librosa.load()]
NumPy array (waveform)
   ↓ [Lab 2: NumPy operations]
NumPy array (processed)
   ↓ [Lab 3: librosa.stft()]
NumPy array (spectrogram)
   ↓ [Lab 4: torch.from_numpy()]
PyTorch tensor (CPU)
   ↓ [Lab 4: .to(device)]
PyTorch tensor (GPU)
   ↓ [Lab 5: U-Net processing]
Processed spectrogram
```

Slide 5: Ryan and Yovannoa's PyTorch Lessons (1 minute)

Title: "Learning from Team Tutorials"

Ryan's Key Learnings (Fashion-MNIST):

```python
# GPU verification (like Ryan did)
if torch.cuda.is_available():
    device = torch.device('cuda')
    print(f"GPU: {torch.cuda.get_device_name(0)}")
else:
    device = torch.device('cpu')

# Tensor operations (Ryan's tutorial)
x = torch.randn(64, 784)  # Batch of 64 images
x = x.to(device)          # Move to GPU

# Simple network (Ryan's architecture)
flatten = torch.nn.Flatten()
linear = torch.nn.Linear(784, 128)
relu = torch.nn.ReLU()

output = relu(linear(flatten(x)))
```

Yovannoa's Key Learnings (60 Minute Blitz):

```python
# Autograd example (Yovannoa's tutorial)
x = torch.randn(3, requires_grad=True)
y = x * 2

# Backward propagation
y.backward(torch.ones_like(y))
print(x.grad)  # Gradients calculated automatically!

# Training loop structure (Yovannoa learned this)
for epoch in range(num_epochs):
    for batch in dataloader:
        # 1. Zero gradients
        optimizer.zero_grad()

        # 2. Forward pass
        outputs = model(inputs)
        loss = criterion(outputs, labels)

        # 3. Backward pass
        loss.backward()

        # 4. Update weights
        optimizer.step()
```

How This Applies to U-Net: - Same GPU acceleration concepts - Same forward/backward propagation - Same training loop structure - Just different data (audio

spectrograms instead of images) - Different architecture (U-Net instead of simple classifiers)

Slide 6: Preview: U-Net Processing (30 seconds)

Title: "What Happens in Lab 5"

U-Net Forward Pass (Simplified):

```
# This is what happens inside U-Net
def unet_forward(spectrogram_tensor):
    # Input: (batch, 1, freq, time)
    # U-Net architecture will:
    # 1. Encoder: Downsample and learn features
    # 2. Bottleneck: Compress information
    # 3. Decoder: Upsample and reconstruct
    # Output: (batch, 1, freq, time) - separated vocal mask

    return separated_mask

# After U-Net
input_spec = torch.Size([1, 1, 1025, 646])
output_mask = unet_model(input_spec)  # Lab 5!
print(f"Output: {output_mask.shape}")  # Same shape!
```

cervanj2's Architecture: - **Preprocessing Layer:** Audio → Spectrogram (Labs 1-3) - **Convert to Tensor:** NumPy → PyTorch (Lab 4) ← We are here - **Inference Layer:** U-Net processing (Lab 5) ← Next! - **Post-processing Layer:** Tensor → Audio (reverse Labs 3-4)

Demo Script (Detailed Timing) - Provided by Claude

0:00 - 0:30: Introduction

"Welcome to Lab 4: PyTorch Essentials! We've learned how to load audio, process it with NumPy, and transform it with librosa. Now we're going to convert our spectrograms into PyTorch tensors so U-Net can process them. This builds directly on Ryan and Yovannoa's PyTorch tutorials."

0:30 - 1:15: Why PyTorch?

"Why do we need PyTorch? [Show Slide 2]. Two main reasons: GPU acceleration and automatic differentiation.

Ryan and Yovannoa both learned that GPUs are 10 to 100 times faster than CPUs for neural network operations. For U-Net processing, this means seconds instead of minutes.

Second, PyTorch's autograd automatically calculates gradients during training. This is what both Ryan and Yovannoa used in their tutorials - and it's what will train our U-Net model.

Here's the cool part: spectrograms are just 2D images! Ryan worked with Fashion-MNIST images, Yovannoa with CIFAR-10 images. We're working with spectrogram images. The PyTorch concepts are exactly the same."

1:15 - 2:45: First Tensors

"Let's create our first PyTorch tensors. [Show Slide 3 and run code]

First, we check if we have a GPU available… [show output]

Ryan has an NVIDIA GeForce RTX 2070 Super Max-Q - that's what we might see here. If no GPU, PyTorch falls back to CPU gracefully.

Now let's convert a spectrogram to a tensor… [show conversion]

See how easy that is? NumPy array becomes a PyTorch tensor with one line. Initially it's on CPU, but watch what happens when we move it to GPU with `.to(device)`…

[Show GPU transfer]

Now it's on the GPU! Operations like ReLU (which we'll use in U-Net) happen instantly on the GPU. This is exactly what Ryan and Yovannoa learned in their tutorials."

2:45 - 4:00: Complete Pipeline

"Let's put all four labs together. [Show Slide 4 and run complete pipeline]

We start with an audio file… load it with librosa (Lab 1)… it's a NumPy array (Lab 2)… create a spectrogram with STFT (Lab 3)… and now convert to PyTorch tensor (Lab 4).

Notice we add two extra dimensions - batch size and channels. U-Net expects a 4D tensor: (batch, channels, frequency, time). This is the same format Ryan and Yovannoa used for image data.

Finally, we move it to GPU, and now it's ready for U-Net processing! This tensor represents a single spectrogram, sitting on the GPU, ready to be separated into vocals and music."

4:00 - 4:45: Team Learnings

"Ryan and Yovannoa's tutorials taught us exactly what we need. [Show Slide 5].

Ryan verified GPU access, learned tensor operations, and built simple networks. Yovannoa learned about autograd and the training loop structure.

All of these concepts apply directly to our U-Net model. Same GPU acceleration, same autograd, same training loop. The only difference is our data - audio spectrograms instead of Fashion-MNIST or CIFAR-10 images."

4:45 - 5:00: Wrap-up

"That's Lab 4! We can now take audio all the way from a file to a GPU-ready PyTorch tensor. Next up - Lab 5, the finale - where we process these tensors with U-Net for actual vocal separation.

Questions?"

Code Files to Provide

`lab4_pytorch_basics.py`

```python
"""
Lab 4: PyTorch Essentials
Tensor operations and GPU acceleration
"""
import torch
import numpy as np
import librosa

def check_pytorch_setup():
    """Check PyTorch and GPU availability"""
    print("=== PyTorch Setup ===")
    print(f"PyTorch version: {torch.__version__}")
    print(f"CUDA available: {torch.cuda.is_available()}")
```

```
    if torch.cuda.is_available():
        print(f"CUDA version: {torch.version.cuda}")
        print(f"GPU count: {torch.cuda.device_count()}")
        print(f"Current GPU: {torch.cuda.current_device()}")
        print(f"GPU name: {torch.cuda.get_device_name(0)}")

    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
    print(f"\nSelected device: {device}")
    return device

def numpy_to_pytorch_demo(device):
    """Demonstrate NumPy to PyTorch conversion"""
    print("\n=== NumPy → PyTorch Conversion ===")

    # Create NumPy array (simulating spectrogram)
    numpy_array = np.random.randn(1025, 646).astype(np.float32)
    print(f"NumPy array: shape={numpy_array.shape}, dtype={numpy_array.dty

    # Convert to PyTorch tensor
    tensor_cpu = torch.from_numpy(numpy_array)
    print(f"PyTorch tensor (CPU): shape={tensor_cpu.shape}, dtype={tensor_

    # Move to GPU
    tensor_gpu = tensor_cpu.to(device)
    print(f"PyTorch tensor (GPU): shape={tensor_gpu.shape}, device={tensor

    # Operations on GPU
    tensor_processed = torch.relu(tensor_gpu)
    tensor_scaled = tensor_processed * 0.5

    # Back to NumPy
    result = tensor_scaled.cpu().numpy()
    print(f"Back to NumPy: shape={result.shape}, dtype={result.dtype}")

    return result

def audio_to_tensor_pipeline(audio_path, device):
    """Complete pipeline: Audio file → PyTorch tensor on GPU"""
    print("\n=== Complete Audio → Tensor Pipeline ===")

    # Lab 1: Load audio
    audio, sr = librosa.load(audio_path, sr=22050)
    print(f"1. [Lab 1] Audio loaded: {audio.shape}")

    # Lab 2: NumPy operations
    print(f"2. [Lab 2] NumPy ready: dtype={audio.dtype}")

    # Lab 3: Create spectrogram
    stft = librosa.stft(audio, n_fft=2048, hop_length=512)
    magnitude = np.abs(stft)
    print(f"3. [Lab 3] Spectrogram: {magnitude.shape}")

    # Lab 4: Convert to PyTorch tensor
    tensor = torch.from_numpy(magnitude).float()

    # Add batch and channel dimensions for U-Net
    # U-Net expects: (batch, channels, freq, time)
    tensor = tensor.unsqueeze(0).unsqueeze(0)

    # Move to GPU
    tensor = tensor.to(device)

    print(f"4. [Lab 4] PyTorch tensor: {tensor.shape}")
    print(f"   - Batch size: {tensor.shape[0]}")
    print(f"   - Channels: {tensor.shape[1]}")
    print(f"   - Frequency: {tensor.shape[2]}")
    print(f"   - Time: {tensor.shape[3]}")
    print(f"   - Device: {tensor.device}")
```

```
    return tensor

def simple_neural_operation(tensor):
    """Demonstrate simple neural network operation"""
    print("\n=== Simple Neural Network Operation ===")

    # Common operations in neural networks
    normalized = torch.nn.functional.normalize(tensor, dim=2)
    activated = torch.relu(normalized)
    pooled = torch.nn.functional.avg_pool2d(activated, kernel_size=2)

    print(f"Input shape: {tensor.shape}")
    print(f"After normalize: {normalized.shape}")
    print(f"After ReLU: {activated.shape}")
    print(f"After pooling: {pooled.shape}")

    return pooled

if __name__ == "__main__":
    # Check setup
    device = check_pytorch_setup()

    # NumPy to PyTorch demo
    result = numpy_to_pytorch_demo(device)

    # Complete pipeline (requires audio file)
    # tensor = audio_to_tensor_pipeline('sample_audio.wav', device)

    # Simple neural operations
    # output = simple_neural_operation(tensor)
```

Visual Aids Needed

1. **NumPy ↔ PyTorch Flow Diagram**
2. **CPU vs GPU Speed Comparison Chart**
3. **4D Tensor Shape Breakdown:** (batch, channels, freq, time)
4. **Complete Pipeline Visualization:** Labs 1-4 flow

Backup Slides

Extra: DataLoader Preview

```
from torch.utils.data import Dataset, DataLoader

class AudioDataset(Dataset):
    def __init__(self, audio_files):
        self.files = audio_files

    def __len__(self):
        return len(self.files)

    def __getitem__(self, idx):
        # Load and process audio
        return spectrogram_tensor

dataloader = DataLoader(dataset, batch_size=8, shuffle=True)
```

Extra: Common PyTorch Operations

- `torch.cat()`: Concatenate tensors
- `torch.reshape()`: Change tensor shape

- `torch.nn.functional.*`: Neural network operations

Q&A Prep - Provided by Claude

Q: "What if I don't have a GPU?" A: "PyTorch works fine on CPU - it's just slower. For learning and small models, CPU is okay. For training U-Net, GPU is highly recommended."

Q: "How much faster is GPU?" A: "10-100x faster depending on the operation. For U-Net training, this could be the difference between hours and minutes."

Q: "Do tensors always need to be on GPU?" A: "No. Keep them on CPU until you need neural network operations. Move to GPU → process → move back to CPU for saving."

Q: "What's the difference between torch.Tensor and torch.tensor?" A: "`torch.tensor()` (lowercase) is the preferred way to create tensors. `torch.Tensor` is a class."

Success Criteria

Students should be able to: - [ ] Check for GPU availability - [ ] Convert NumPy arrays to PyTorch tensors - [ ] Move tensors between CPU and GPU - [ ] Understand 4D tensor shape for U-Net - [ ] Connect Ryan and Yovannoa's tutorials to audio processing

---

Connection Summary

Ryan's Tutorial: GPU access, tensor basics, simple networks **Yovannoa's Tutorial:** Autograd, training loops, complete workflow **cervanj2's Architecture:** PyTorch powers the Inference Layer **Next Lab:** U-Net processes these tensors for separation