Getting PyTorch to Use Your GPU

Training deep learning models on a CPU is super slow. Using a GPU (Graphics Processing Unit) makes this way faster. This guide covers the main ways to get PyTorch using a GPU, focusing on your local machine first.

Hardware Support

Before you install anything, you need to know what hardware you have. PyTorch only works with certain hardware:

- **NVIDIA GPUs:** This is the main, most stable, and best-supported option. PyTorch uses NVIDIA's **CUDA** platform to run on these cards.
- **Apple Silicon (M1, M2, M3, etc.):** Works out-of-the-box on newer Macs. PyTorch uses the **MPS** (Metal Performance Shaders) backend for acceleration.
- **Intel/AMD GPUs:** Support is limited and often experimental (like ROCm for AMD). don't get your hopes up.

Installation: PyTorch Hardware Backends

A simple `pip install torch` usually just installs the CPU-only version. You have to install a version of PyTorch that's built for your specific hardware (CUDA or MPS).

Method A: UV with Next Wheel (The Easy but Experimental Way)

`uv` is a new, super-fast Python package manager that's getting quite popular. It has an experimental version that supports "Next Wheel," which tries to auto-detect your hardware and install the right PyTorch version. It does require you to a specific version of `uv` with this. (pytorch.org/blog/pytorch-wheel-variants/)

1. Install the specific `uv` variant:

macOS / Linux: macOS / Linux:

```
curl -LsSf https://astral.sh/uv/install.sh | \
INSTALLER_DOWNLOAD_URL=https://wheelnext.astral.sh sh
```

Windows (PowerShell): Windows (PowerShell):

```
powershell -ExecutionPolicy Bypass -c \
"$env:INSTALLER_DOWNLOAD_URL='https://wheelnext.astral.sh'; irm \
https://astral.sh/uv/install.ps1 | iex"
```

2. Use uv to install:

```
# 1. Create a virtual environment
uv venv

# 2. Activate the environment
source .venv/bin/activate  # (macOS/Linux)
# .venv\Scripts\activate.ps1    # (Windows PowerShell)

# 3. Install torch
uv pip install torch torchvision torchaudio
```

uv will attempt to detect your hardware (CUDA, MPS, or CPU) and install the appropriate wheel.

Method B: Conda (The Robust & Tested Way)

Conda is the old, reliable package manager. It's really good at handling the messy library dependencies (like the CUDA toolkit) that PyTorch needs. This method is more manual but it's super reliable.

1. Install **Miniconda** (a minimal Anaconda installer).
2. Navigate to the official PyTorch website: https://pytorch.org/get-started/locally/
3. Use the command builder to select your environment (e.g., *Stable, OS, Conda, CUDA 12.1*).
4. This will give you the exact command to run, which includes all the libraries you need.

```
conda install pytorch torchvision torchaudio pytorch-cuda=12.1 -c pytorch -c
    nvidia
```

```
conda install pytorch torchvision torchaudio -c pytorch
```

Conda will figure out and install PyTorch, its dependencies, and the GPU libraries (like pytorch-cuda) all at once.

Google Colab

If you don't have a supported GPU, or you get stuck with the local setup, Google Colab is an easy fallback. You don't need any special hardware or driver setup.

Note: The free GPUs you get can vary and might not be much faster than a good laptop CPU.

1. Navigate to https://colab.research.google.com/.
2. Open or create a new notebook.
3. **Enable the GPU Runtime:** By default, Colab notebooks use a CPU.
   - In the menu, select **Runtime** → **Change runtime type**.
   - Under **Hardware accelerator**, select **T4 GPU**.
4. The environment now has a GPU.

Required Code Changes for GPU Usage

Just installing the package isn't enough. You have to tell PyTorch to move your model and your data tensors to the GPU memory in your code.

1. Define the 'device' Variable

At the start of your script, add some code to check for and select the hardware you have.

```python
import torch

# Check for NVIDIA CUDA
if torch.cuda.is_available():
    device = torch.device("cuda")
    print(f"Using NVIDIA GPU: {torch.cuda.get_device_name(0)}")
# Check for Apple Silicon MPS
elif torch.backends.mps.is_available():
    device = torch.device("mps")
    print("Using Apple GPU (MPS)")
# Fallback to CPU
else:
    device = torch.device("cpu")
    print("No supported GPU found, using CPU")
```

2. Move Your Model to the Device

You only need to do this once, right after you create your model. This moves the model's parameters to the GPU.

```python
model = YourNetworkArchitecture()
model.to(device)
```

3. Move Data Tensors to the Device

You have to move your tensors to the `device` **inside your training loop** for every single batch.

```python
# ... inside your training loop ...
for data, labels in train_loader:

    # Move the data and labels for THIS batch to the device
    data = data.to(device)
    labels = labels.to(device)

    # Proceed with training operations
    optimizer.zero_grad()
    outputs = model(data)
    loss = criterion(outputs, labels)
    loss.backward()
    optimizer.step()
```

If you forget to move both the model and the data to the same device, you'll get a `RuntimeError` stating that tensors are on different devices (e.g., `cpu` and `cuda:0`).

Additional Resources

- **DTU MLOps Course (Development Environment):** https://skaftenicki.github.io/dtu_mlops/s1_development_environment/
- **PyTorch Official Installation Guide:** https://pytorch.org/get-started/locally/