CPEN 455: Deep Learning

Tutorial 6: Introduction to Project (Conditional PixelCNN++)

Qihang Zhang & Sadegh Mahdavi

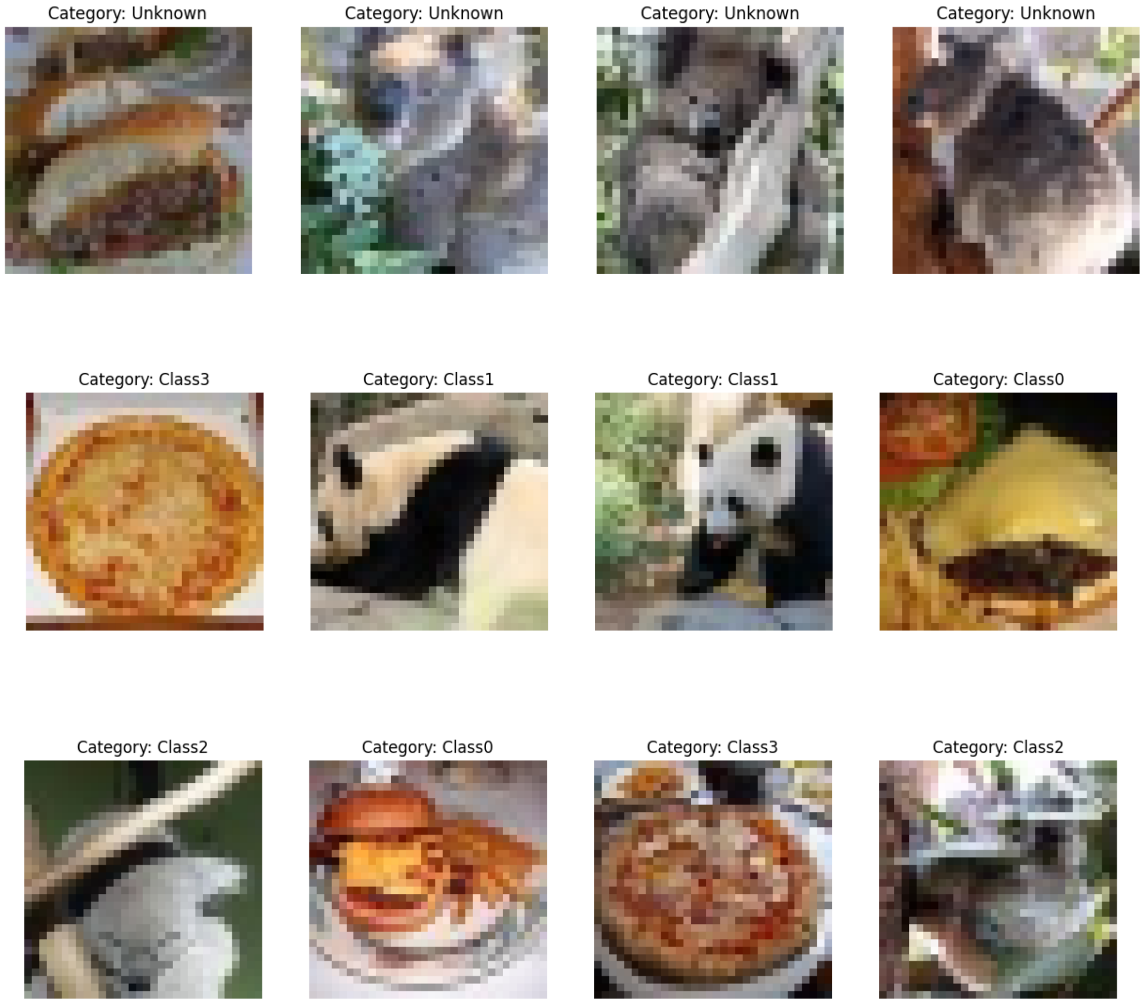
University of British Columbia

Winter, Term 2, 2024

Project Description

- The project is available at https://github.com/DSL-Lab/CPEN455HW-2024W2
- You will modify the original code of PixelCNN++ to create a conditional PixelCNN++ for Image Classification.
- The details of how PixelCNN++ works will be covered in subsequent tutorials, but you can read the README, and the PixelCNN++ paper to get yourself familiar with the code template.

Overview of the Dataset



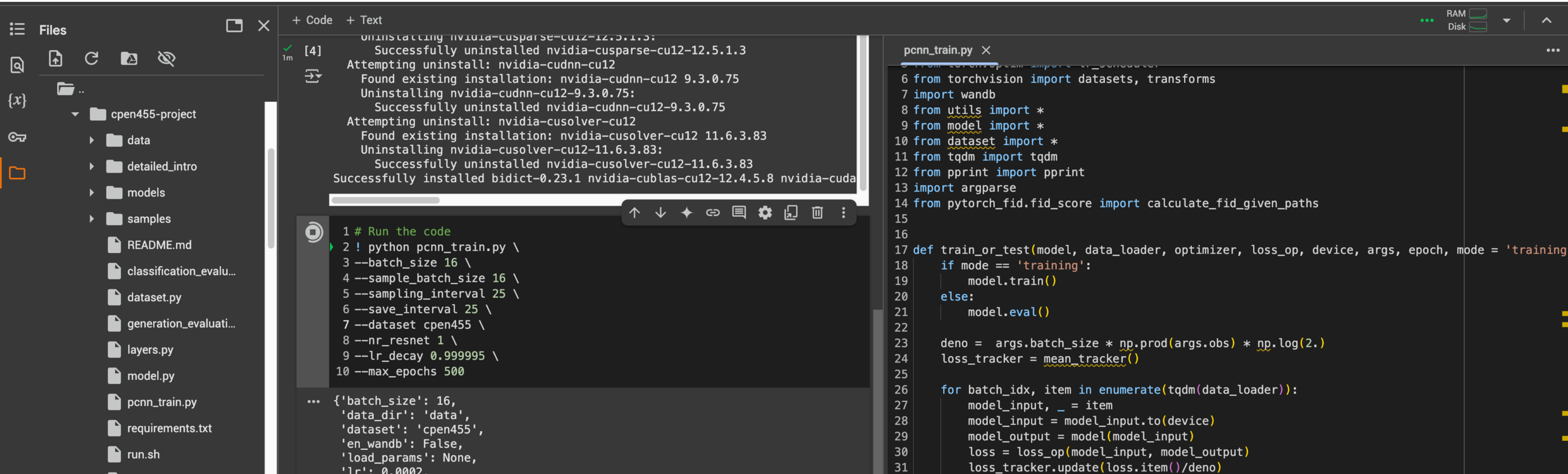
Project Description

- TL;DR: Teach classification to a generative model!
- You will implement:
  - A conditional PixelCNN++
  - Use it to both **generate** and **classify** images.

$$p(c=i|x) = \frac{p_\theta(x|c=i) \times p(c=i)}{\sum_{k=1}^{4} p_\theta(x|c=k)p(c=k)}$$

Running On Colab

- Put the project files into your google drive
- Mount your Google Drive on Colab
- Edit the files
- Run the notebook!



```
  Uninstalling nvidia-cusparse-cu12-12.5.1.3:
    Successfully uninstalled nvidia-cusparse-cu12-12.5.1.3
Attempting uninstall: nvidia-cudnn-cu12
  Found existing installation: nvidia-cudnn-cu12 9.3.0.75
  Uninstalling nvidia-cudnn-cu12-9.3.0.75:
    Successfully uninstalled nvidia-cudnn-cu12-9.3.0.75
Attempting uninstall: nvidia-cusolver-cu12
  Found existing installation: nvidia-cusolver-cu12 11.6.3.83
  Uninstalling nvidia-cusolver-cu12-11.6.3.83:
    Successfully uninstalled nvidia-cusolver-cu12-11.6.3.83
Successfully installed bidict-0.23.1 nvidia-cublas-cu12-12.4.5.8 nvidia-cuda
```

```
# Run the code
! python pcnn_train.py \
--batch_size 16 \
--sample_batch_size 16 \
--sampling_interval 25 \
--save_interval 25 \
--dataset cpen455 \
--nr_resnet 1 \
--lr_decay 0.999995 \
--max_epochs 500
```

```
{'batch_size': 16,
 'data_dir': 'data',
 'dataset': 'cpen455',
 'en_wandb': False,
 'load_params': None,
 'lr': 0.0002,
```

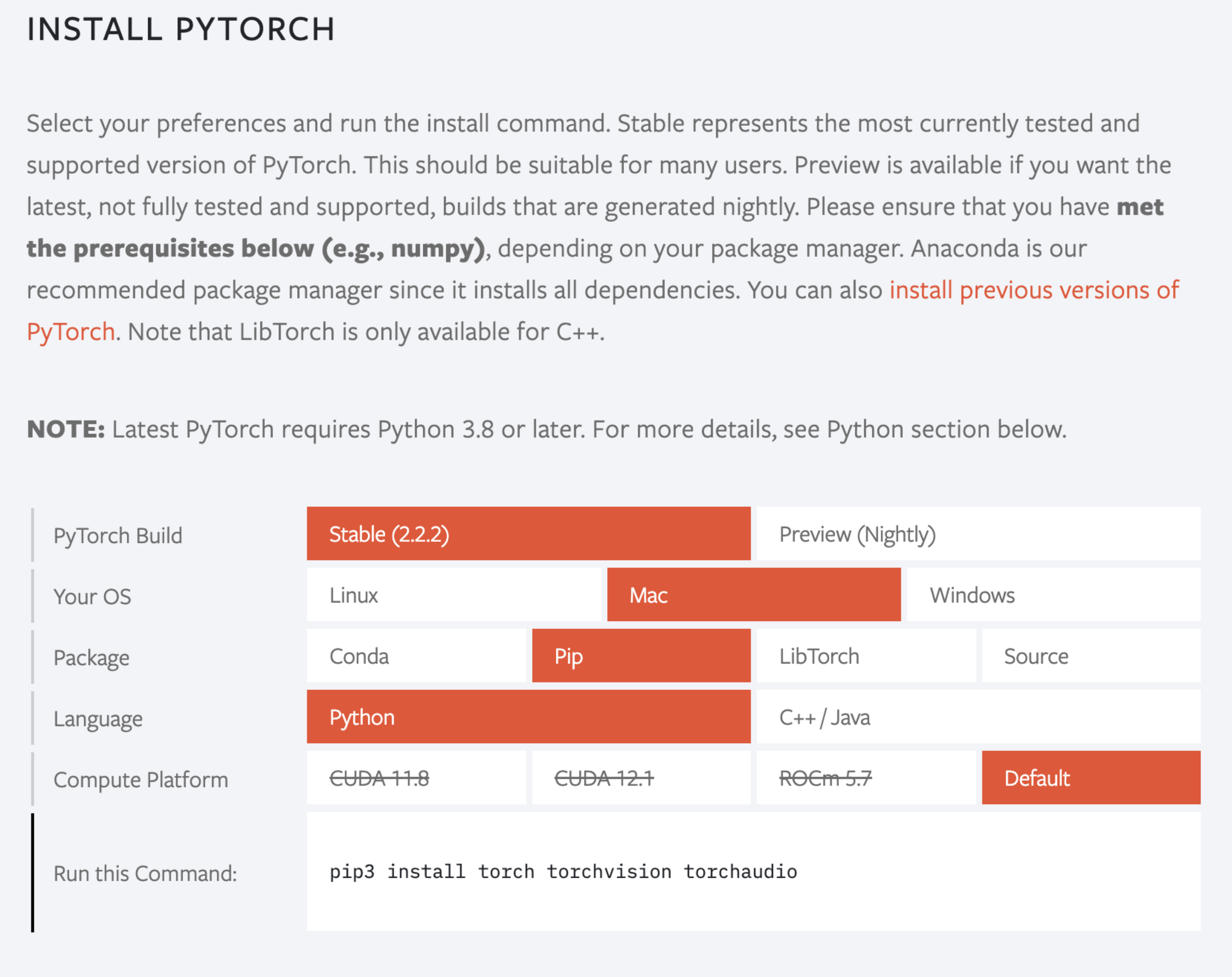
```python
from torch.optim import lr_scheduler
from torchvision import datasets, transforms
import wandb
from utils import *
from model import *
from dataset import *
from tqdm import tqdm
from pprint import pprint
import argparse
from pytorch_fid.fid_score import calculate_fid_given_paths


def train_or_test(model, data_loader, optimizer, loss_op, device, args, epoch, mode = 'training
    if mode == 'training':
        model.train()
    else:
        model.eval()

    deno =  args.batch_size * np.prod(args.obs) * np.log(2.)
    loss_tracker = mean_tracker()

    for batch_idx, item in enumerate(tqdm(data_loader)):
        model_input, _ = item
        model_input = model_input.to(device)
        model_output = model(model_input)
        loss = loss_op(model_input, model_output)
        loss_tracker.update(loss.item()/deno)
```

Running On Colab

- If you work with Colab:
    - Make sure you always use GPU on colab (otherwise code is too slow)
    - Make sure you periodically take a snapshot of your code (no easy Git support on colab)

Conda Env (Running Locally)

- Install pytorch
- pip install –r requirements.txt



INSTALL PYTORCH

Select your preferences and run the install command. Stable represents the most currently tested and supported version of PyTorch. This should be suitable for many users. Preview is available if you want the latest, not fully tested and supported, builds that are generated nightly. Please ensure that you have **met the prerequisites below (e.g., numpy)**, depending on your package manager. Anaconda is our recommended package manager since it installs all dependencies. You can also install previous versions of PyTorch. Note that LibTorch is only available for C++.

NOTE: Latest PyTorch requires Python 3.8 or later. For more details, see Python section below.

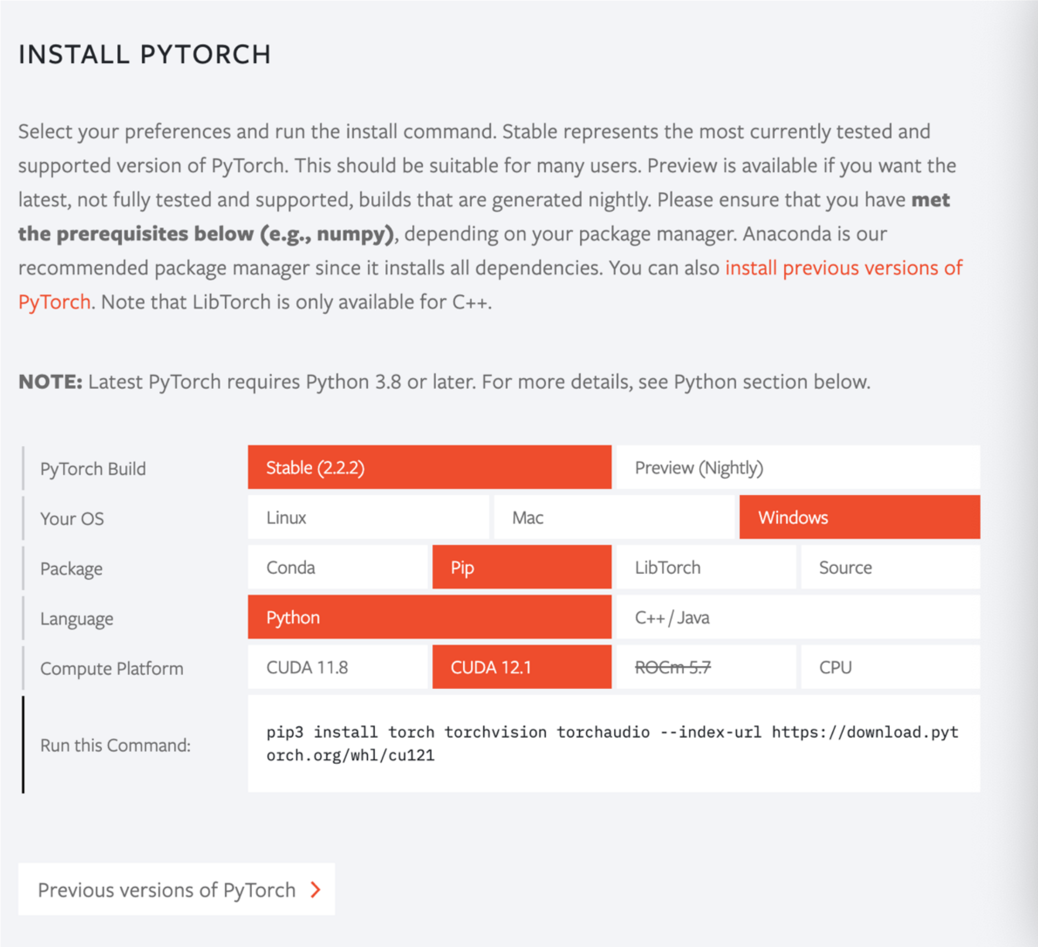
PyTorch Build	[✓ Stable (2.2.2)]	Preview (Nightly)		
Your OS	Linux	[✓ Mac]	Windows	
Package	Conda	[✓ Pip]	LibTorch	Source
Language	[✓ Python]	C++ / Java		
Compute Platform	~~CUDA 11.8~~	~~CUDA 12.1~~	~~ROCm 5.7~~	[✓ Default]

Run this Command:

```
pip3 install torch torchvision torchaudio
```

Conda Env

- Install pytorch
- pip install –r requirements.txt



INSTALL PYTORCH

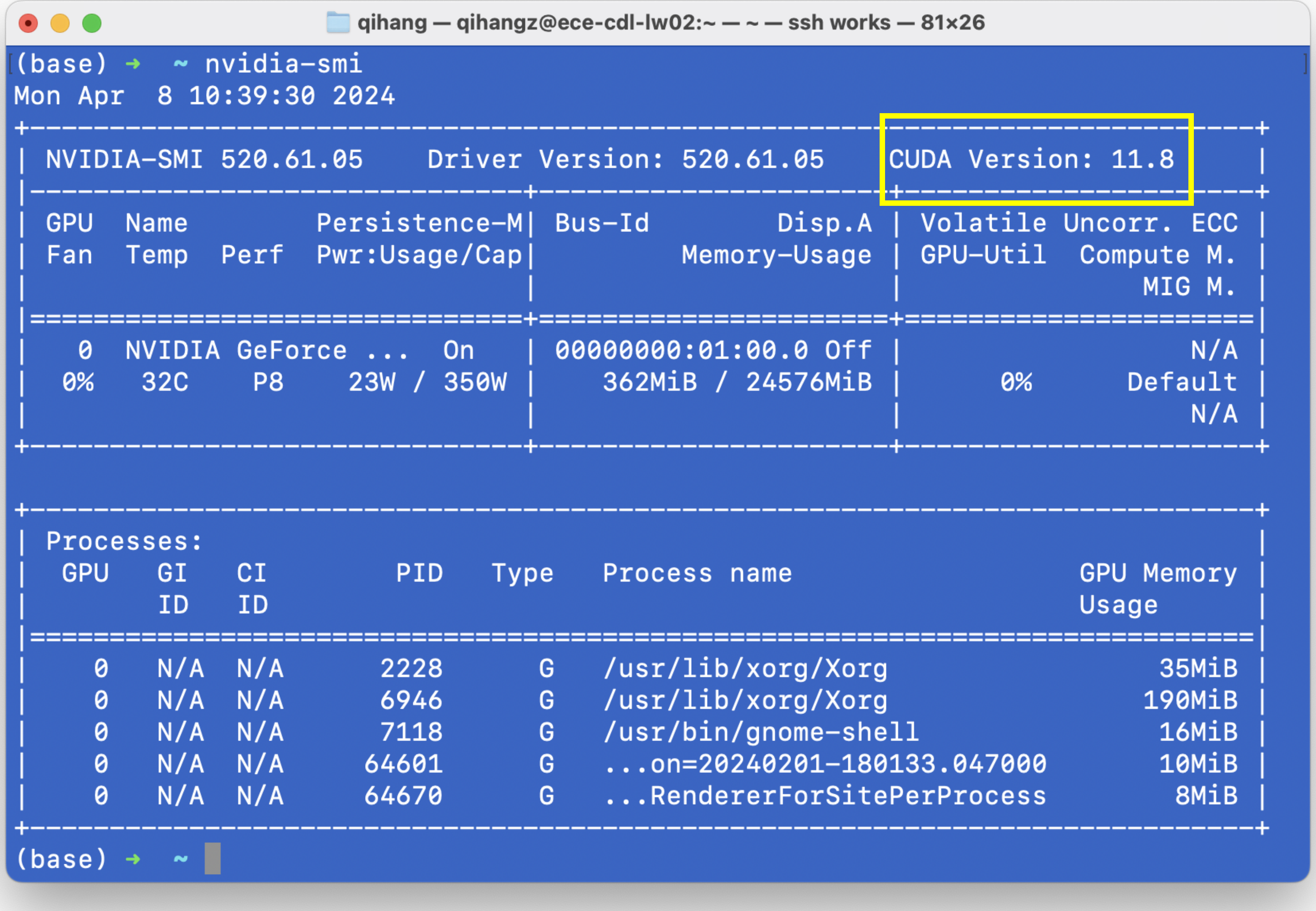
Select your preferences and run the install command. Stable represents the most currently tested and supported version of PyTorch. This should be suitable for many users. Preview is available if you want the latest, not fully tested and supported, builds that are generated nightly. Please ensure that you have **met the prerequisites below (e.g., numpy)**, depending on your package manager. Anaconda is our recommended package manager since it installs all dependencies. You can also install previous versions of PyTorch. Note that LibTorch is only available for C++.

NOTE: Latest PyTorch requires Python 3.8 or later. For more details, see Python section below.

PyTorch Build	[✓ Stable (2.2.2)]	Preview (Nightly)		
Your OS	Linux	Mac	[✓ Windows]	
Package	Conda	[✓ Pip]	LibTorch	Source
Language	[✓ Python]	C++ / Java		
Compute Platform	CUDA 11.8	[✓ CUDA 12.1]	~~ROCm 5.7~~	CPU
Run this Command:	pip3 install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cu121			

Previous versions of PyTorch >

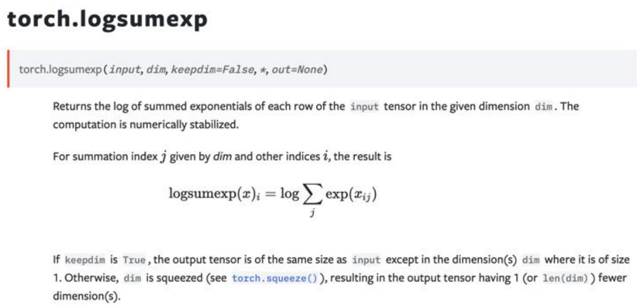
nvidia-smi



```
(base) ➜  ~ nvidia-smi
Mon Apr  8 10:39:30 2024
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 520.61.05    Driver Version: 520.61.05    CUDA Version: 11.8     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  NVIDIA GeForce ...  On   | 00000000:01:00.0 Off |                  N/A |
|  0%   32C    P8    23W / 350W |    362MiB / 24576MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|    0   N/A  N/A      2228      G   /usr/lib/xorg/Xorg                 35MiB |
|    0   N/A  N/A      6946      G   /usr/lib/xorg/Xorg                190MiB |
|    0   N/A  N/A      7118      G   /usr/bin/gnome-shell               16MiB |
|    0   N/A  N/A     64601      G   ...on=20240201-180133.047000       10MiB |
|    0   N/A  N/A     64670      G   ...RendererForSitePerProcess        8MiB |
+-----------------------------------------------------------------------------+
(base) ➜  ~ 
```

Pro Tip: Potential numerical issue you may encounter

- **Never use "log" and "softmax" functions sequentially** (like log(softmax()) function).
- Every deep learning scientist has learned this the hard way (~days of debugging!)
- This may lead to severe numerical issues.
- Instead, the following torch functions are numerically stable (depending on your need):



torch.logsumexp

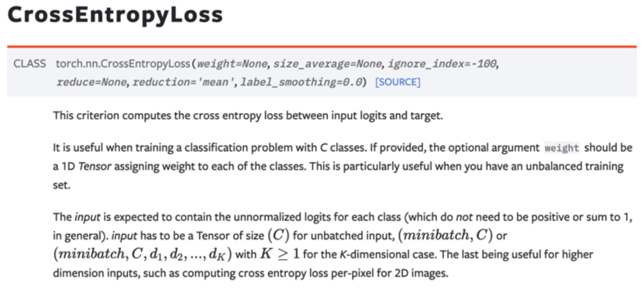
`torch.logsumexp(input, dim, keepdim=False, *, out=None)`

Returns the log of summed exponentials of each row of the `input` tensor in the given dimension `dim`. The computation is numerically stabilized.

For summation index j given by *dim* and other indices i, the result is

$$\text{logsumexp}(x)_i = \log \sum_j \exp(x_{ij})$$

If `keepdim` is `True`, the output tensor is of the same size as `input` except in the dimension(s) `dim` where it is of size 1. Otherwise, `dim` is squeezed (see `torch.squeeze()`), resulting in the output tensor having 1 (or `len(dim)`) fewer dimension(s).



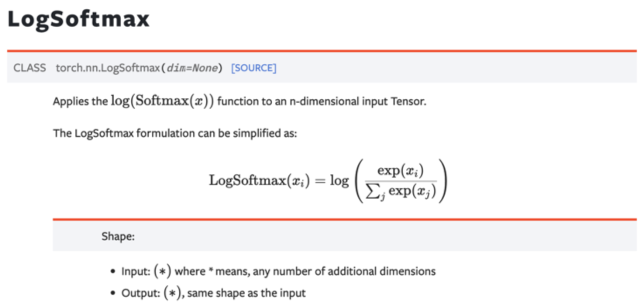
CrossEntropyLoss

CLASS `torch.nn.CrossEntropyLoss(weight=None, size_average=None, ignore_index=-100, reduce=None, reduction='mean', label_smoothing=0.0)` [SOURCE]

This criterion computes the cross entropy loss between input logits and target.

It is useful when training a classification problem with *C* classes. If provided, the optional argument `weight` should be a 1D *Tensor* assigning weight to each of the classes. This is particularly useful when you have an unbalanced training set.

The *input* is expected to contain the unnormalized logits for each class (which do *not* need to be positive or sum to 1, in general). *input* has to be a Tensor of size (C) for unbatched input, $(minibatch, C)$ or $(minibatch, C, d_1, d_2, ..., d_K)$ with $K \geq 1$ for the *K*-dimensional case. The last being useful for higher dimension inputs, such as computing cross entropy loss per-pixel for 2D images.



LogSoftmax

CLASS `torch.nn.LogSoftmax(dim=None)` [SOURCE]

Applies the $\log(\text{Softmax}(x))$ function to an n-dimensional input Tensor.

The LogSoftmax formulation can be simplified as:

$$\text{LogSoftmax}(x_i) = \log\left(\frac{\exp(x_i)}{\sum_j \exp(x_j)}\right)$$

Shape:

- Input: $(*)$ where * means, any number of additional dimensions
- Output: $(*)$, same shape as the input

Leaderboard and Bonus Points [Subject to change]

- Bonus points for:
  - If your model outperforms our baseline (on a Huggingface Leaderboard).
  - Detailed analysis of your model or generated results.

Questions?