Assignment 09

Please refer to MyCourses/Gradescope for assignment deadlines.

The goal of this assignment is to give you hands-on practice with the following concepts:

- Stereo vision and disparity
- Getting started with running code on GPUs using PyTorch

Part 1 of 3: Computing disparity for depth estimation

You've been provided with some images from the 2005 Middlebury Stereo Vision challenge in the `data/` directory. See https://vision.middlebury.edu/stereo/data/scenes2005/ for more information about the dataset format and how to interpret the files.

Each subfolder contains `view1.png` and `view5.png` – two rectified stereo images of the same scene – and a `disp1.png` and `disp2.png` file which contains ground-truth disparities at each pixel. Recall that disparity is inversely proportional to depth, so the disparity images will be *brighter* where objects are *closer* to the camera. You will notice that the disparity maps contain some black pixels around the edges of some objects. These are pixels where there is no true corresponding pixel in the other image, e.g. if an object is occluded by another object in one view but not another.

The Middlebury vision site linked above contains information about the camera setup such as baseline distance and focal length of the camera, as well as information on how to interpret the values in the disparity maps (hint: since disparities are stored in `png` files, they have been scaled to fit in the 0-255 range, but the 'true' disparities are on a different scale, and each image pair has a `dmin.txt` file that is required for converting disparities to actual depths, although we won't be doing that here. Not all the images have a corresponding ground-truth disparity map.

Part 1.1: implement a simple stereo-matching algorithm

Implement your own (simple) stereo-matching algorithm using numpy and "low-level" OpenCV functions. For grading purposes, you're required to implement a simple baseline algorithm using the sum of absolute differences (SAD) metric and simply set disparity to the best match for each pixel regardless of what its neighbors are doing.

The SAD metric at location `(i, j)`, disparity `d`, and window size `w`, is defined as

$$SAD(i,j,d) = \sum_{k=-w/2}^{w/2} \sum_{l=-w/2}^{w/2} |I_1(i+k, j+l) - I_2(i+k, j+l-d)|$$

Note that, since the images are rectified, we only need to apply the disparity d in the horizontal direction. In practice, we set a minimum and maximum disparity range and set the disparity value at `(i,j)` to argmin$_d SAD(i, j, d)$.

Implement this in the `my_sad_disparity_map()` function. It's reasonable and expected that you may need to write a loop over disparity values, but the rest of the function should be vectorized. (Vectorizing hint: compute a `(h, w, max_disparities)` array of SAD values then use `np.argmin(axis=-1)` for the final result).

Part 1.2: implement the RMS error metric to quantify how good your matcher is

Compute how good/bad the output is compared to the ground-truth disparity map and to OpenCV. In other words, you'll implement one of the metrics in the Middlebury evaluation table, which takes a true disparity map and an estimated disparity map and computes a score. Specifically, you'll implement the root-mean-square (RMS) error metric on non-occluded ("nonocc" in the table) pixels. You've also been provided with a "percent match" metric which you can use for reference. Occluded pixels are those marked with `disparity=0` in `disp1.png` and `disp2.png`. RMS is defined as the square root of the mean of the squared differences between the true and estimated disparities at each pixel. Implement this in the `rms_error()` function.

One subtlety here is that, since an output of `disparity=0` is reserved for "no match", you'll need to account for these in two ways:

- if the true disparity is `0`, then the estimated disparity value should be ignored in the RMS calculation
- if the estimated disparity is `0` but the true disparity is nonzero, then the RMS error should count it simply as a small constant error (configurable as an argument to the `rms_error()` function).

This is not the only way to handle these issues, but it's a simple and reasonable approach, and we'll be testing your implementation against the specification laid out here.

As a mathematical formula, you need to compute

$$RMS = \sqrt{\frac{1}{P}\sum_{ij} \mathbb{I}[d_{true}(i,j) > 0] err(i,j)^2}$$

where

$$err(i,j) = \begin{cases} d_{true}(i,j) - d(i,j) & \text{if } d(i,j) > 0 \\ \texttt{zero\_penalty} & \text{otherwise} \end{cases}$$

and $P = \sum_{ij} \mathbb{I}[d_{true}(i,j) > 0]$ is the number of nonzero "true" disparity values.

The `rms_error()` function should be completely vectorized (no loops). A convenient trick is to use *logical indexing*, like a `[b > 0]` to select all elements of a where `b > 0`.

(Optional) Part 1.3: improve on your algorithm and participate in the class leaderboard

(Optional) Try to improve your algorithm's performance, to be compared to your classmates on a Gradescope leaderboard. Winners will receive extra credit using the same scoring system as in Assignment 3 (1st = 5pts, 2nd = 4pts, 3rd = 3pts, 4th = 2pts, all who improved on the baseline SAD method = 1pt).

Important: once you have `my_sad_disparity_map()` working, don't modify it. Instead, your leaderboard submission should be given in `my_leaderboard_disparity_map()`.

We recommend finishing Parts 2 and 3 of this assignment before returning to optimize your algorithm.

Some ideas to consider for improving your algorithm:

- Output `disparity=0` for pixels that have no match.
- Use a metric besides SAD like SSD or NCC (watch this video for details).
- Adjust the window size and/or use a non-rectangular window.
- Implement a multi-scale approach where you first compute disparities at a low resolution and then refine them at a higher resolution (analogous to the pyramid approach to Lucas-Kanade optical flow)
- Use a sparse matching approach like SIFT-based feature matching for some of the pixels.
- Read up on what algorithms are used inside of OpenCV and try to replicate one of them.
- Implement some form of smoothing so that, when a pixel has a multiple "good" disparity matches, you bias the result towards the disparity value of its neighbors (this involves optimization and is usually done with graph cuts or belief propagation, which we haven't covered yet, so it's a bit of a stretch goal).
- Note that disparities are represented as floating point, so you are allowed to output fractional values, which may help in some cases.
- The code sets a maximum disparity value for each image to 1/8th the image width. Should you adjust this? Should you add a minimum disparity value? Can you find good defaults that work for all images in this dataset?

Whatever approach you take, be sure to inspect the outputs and think carefully about what sort of errors your algorithm is making and how you might address them.

Some skeleton code is provided in `stereo.py`. It can be run using `python stereo.py path/` where `path/` is the path to a folder containing `view1.png`, `view5.png`, and `disp1.png` files.

Example output

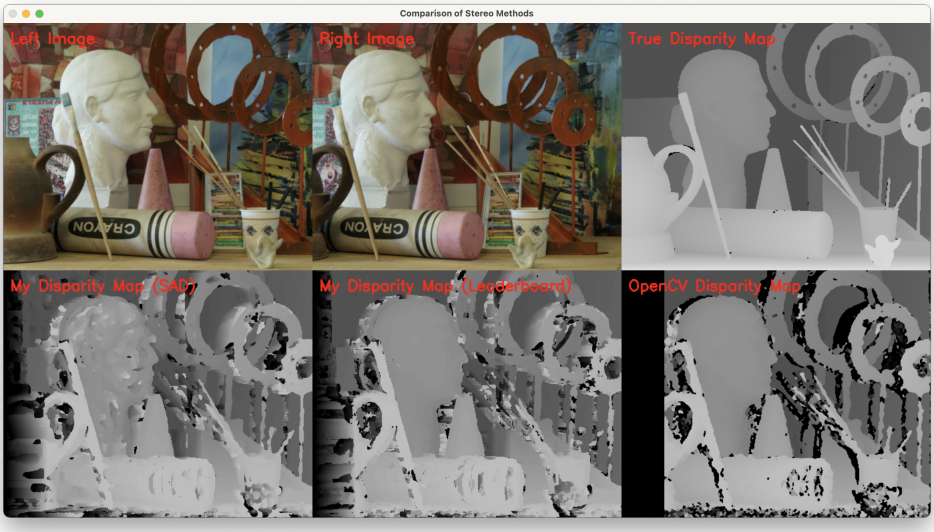

Figure 1: Example screenshot of the main() function running using the `Art/` example from Middlebury

Part 2 of 3: Connect your IDE to the CS department servers

Moving forward, we're going to start using PyTorch and running code on GPUs. This can in principle be done on your personal computer (if it has CUDA support), but regardless of the hardware you have, it's a useful skill to know how to configure your IDE for local development and remote deployment.

You should have access to the following servers in the CS department:

- `granger.cs.rit.edu`
- `weasley.cs.rit.edu`
- `lovegood.cs.rit.edu`

These servers are a shared resources with your classmates and with students in other classes. **It is up to you to use good judgment and not abuse these resources.** For starters, I recommend flipping a 3-sided coin to decide which server to use as a matter of load-balancing.

We'll assume you know the basics of connecting over SSH and running simple linux commands for navigating the filesystem, creating directories, editing plain text files in the shell, etc. If you don't, you should take a moment to familiarize yourself with these concepts.

If using PyCharm, you need to fix a configuration bug before proceeding. Follow these steps:

1. SSH into one of the above servers
2. Open your `~/.bashrc` file in a text editor
3. Paste the following block of code at the end of the file:

```
# PYCHARM-OVER-SSH-FIX.
# The problem: if you run PyCharm on your local machine but configure
↪  it to
# use an "SSH interpreter" (run python here on the server), it
↪  configures
# things by default in /tmp/cache/. However, if one person configures
↪  things
# in /tmp/cache/, then nobody else will have write permissions there,
↪  and
# PyCharm will fail to set up the remote interpreter correctly.
# The solution: set a separate TMPDIR environment variable for each
↪  user
# *and make sure the directory exists using mkdir*.
export TMPDIR=/tmp/${USER}
mkdir -p $TMPDIR
```

If you are using a different IDE (Spyder, VSCode, etc.), you may or may not need to do something similar. Your instructor uses PyCharm, so you're on your own for other IDEs.

Next, you should configure your IDE with a "remote interpreter" and automatic upload of files. The idea is that any changes you make to `.py` files on your local machine will be automatically uploaded to the server[1], and any time you "run" or "debug" a script, the code will be executed on the server. This is super useful when doing things like training machine learning models using server hardware but stepping through line by line in your IDE's debugger.

Here's what this configuration looks like in PyCharm (again, other IDEs will have a similar but different process, for which you're on your own):

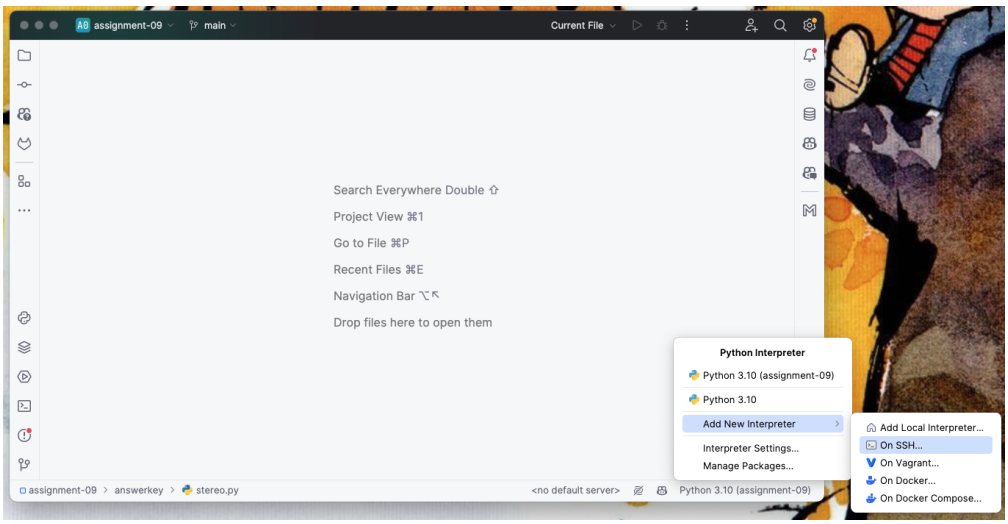

Figure 2: PyCharm remote interpreter configuration Step 1

[1] careful – this is often configured asymmetrically, so changes on the server are not necessarily automatically downloaded back to your machine!

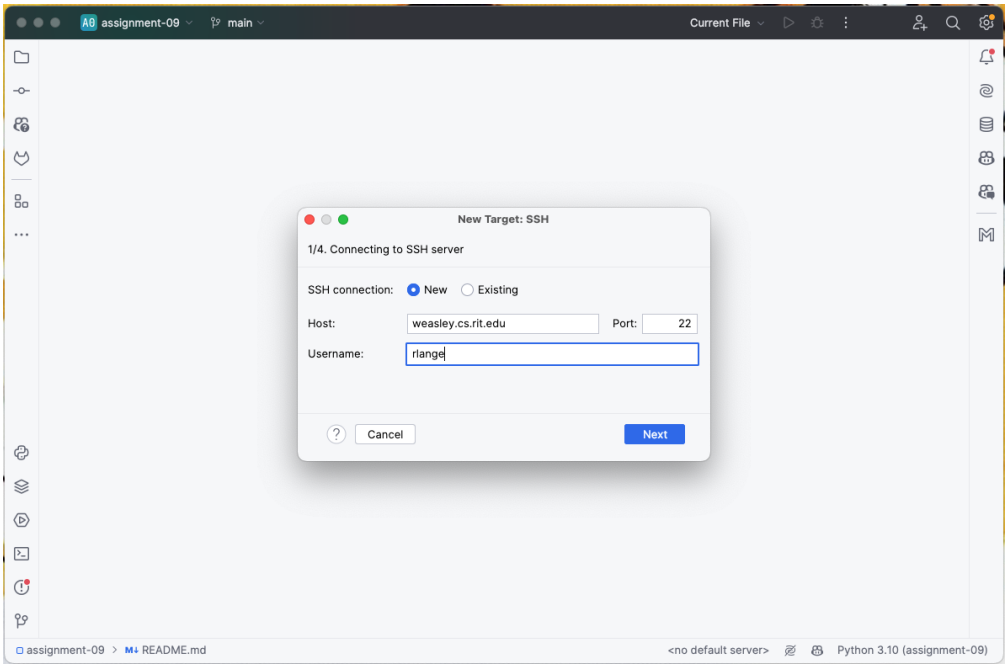

Figure 3: PyCharm remote interpreter configuration Step 2a ("New" SSH configuation if it's your first time connecting to this server)

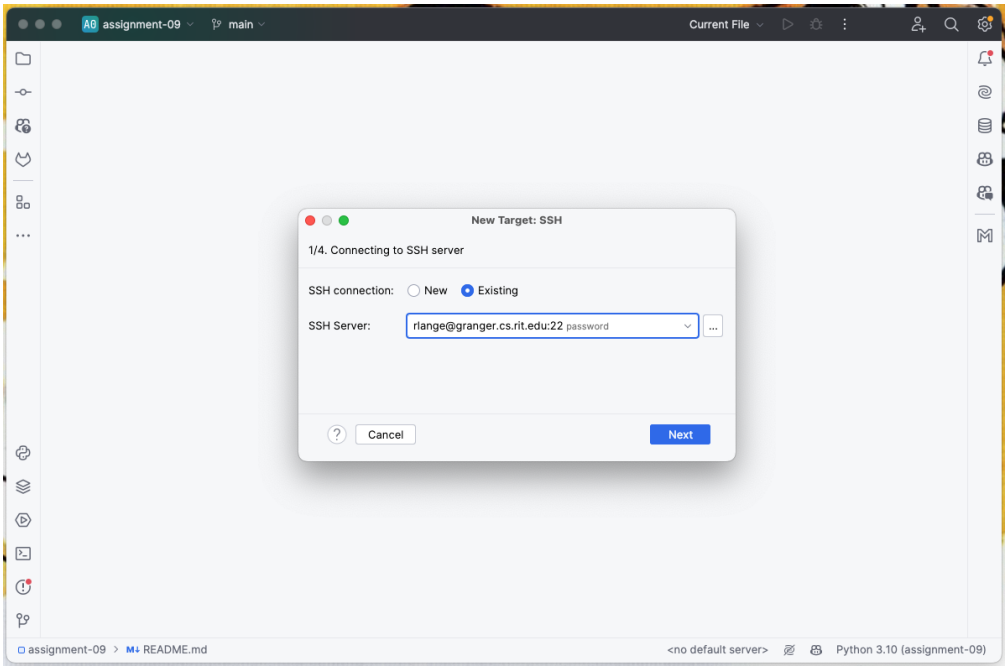

Figure 4: PyCharm remote interpreter configuration Step 2b ("Existing" SSH configuration if you've used this server before (including in other projects))

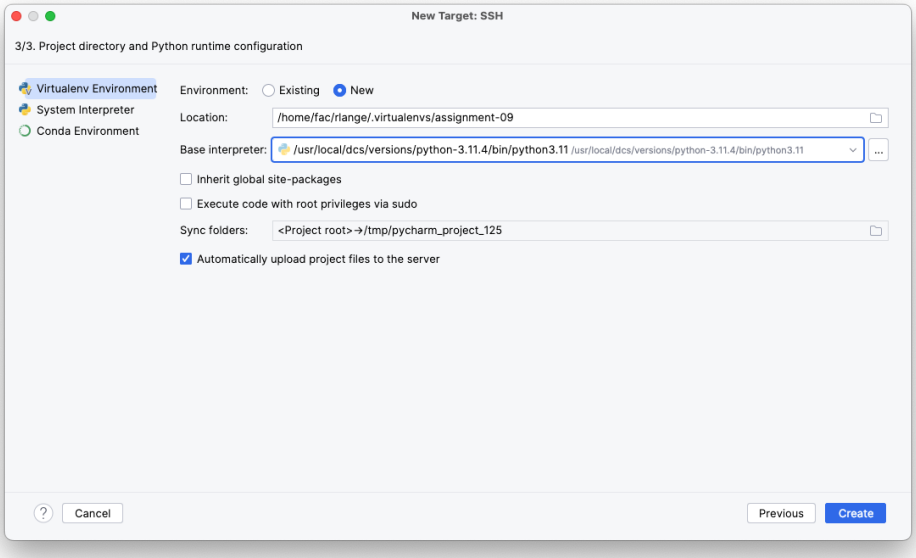

Figure 5: PyCharm remote interpreter configuration Step 3. **Note that the "base interpreter" is not the system default. You should use Python 3.10 or Python 3.11 for this course, both of which are available on the servers in `/usr/local/dcs/versions` as shown here.** Also note that the "Automatic upload" box is checked. This ensures that when you save changes to a file locally, they are automatically sent to the server. Finally, the "sync folders" option defaults to a random `/tmp/` directory on the server, but can be reconfigured to your home directory. This tells PyCharm where to upload files, and where to base its relative paths off of.

Troubleshooting

Server configuration doesn't always go smoothly. Here's what you need to know about PyCharm: the steps above result in three changes to your IDE configuration, each of which can be tweaked after the fact. In PyCharm's settings, you should check issues in three places:

1. Check that you're actually connecting over SSH. To do this, you can open the "Tools > SSH Configurations" OR the "Build, Execution, Deployment > Deployment" tab in the settings and click the "Test Connection" button.

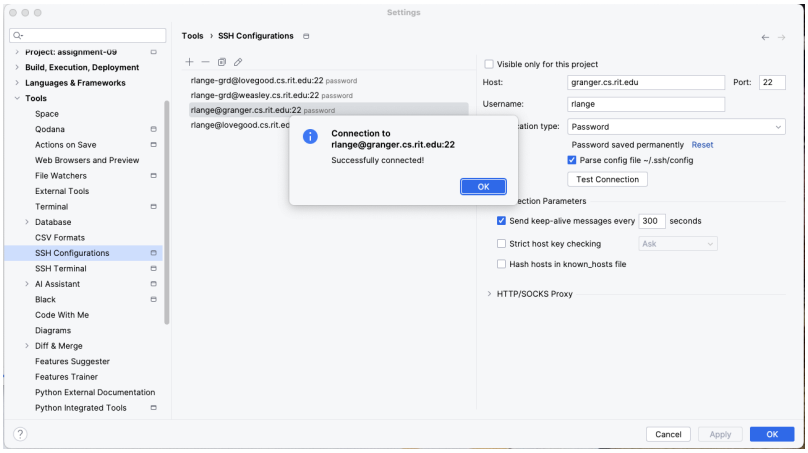

Figure 6: SSH Configuration Test Connection

2. Check that your “Deployment” configuration is correct. This is where you set up where files are uploaded and manage the automatic upload settings. The “Mappings” tab defines which local folders are synced with which remote folders.

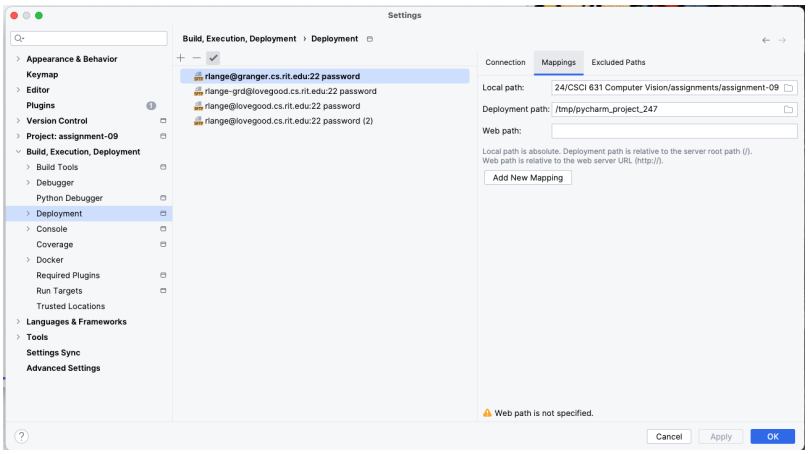

Figure 7: Deployment Configuration

3. Check that your “Python Interpreter” is set up correctly. This is where you set the Python interpreter that will be used to run your code. You can find this in the “Project: > Python Interpreter” settings.

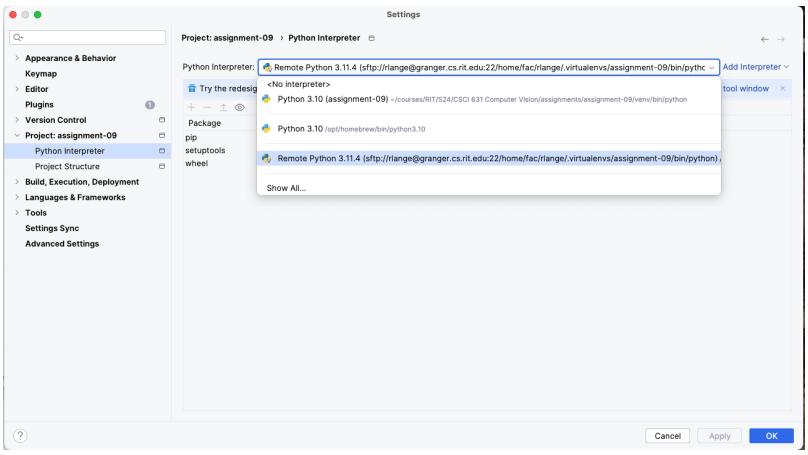

Figure 8: Python Interpreter Configuration

Testing your configuration / proving it works

Once you have your IDE set up to run code on the servers, you should test it. Using your IDE *running locally on your machine, but with the code executing on the server*, run the `gpu_test.py` scipt that we've provided. Then, take a screenshot showing your (local) IDE executing the code remotely and place that screenshot in the `images/` directory.

When you run `make submission.zip`, all `.png` and `.jpg` files in `images/` will automatically be included and uploaded to Gradescope.

Part 3 of 3: Getting started with PyTorch

Now that you have your IDE set up to run code on the servers, you're going to start using PyTorch. PyTorch can run on your machine's CPU, but we need the servers to enable GPU usage. Think of PyTorch as a fancy version of numpy (in fact, they're syntatically almost identical) that can also (1) run on GPUs, and (2) automatically compute gradients for you (more on this later).

Outputs: No python code needs to be submitted for this one, but you'll be writing and running short snippets of code for it. You'll be placing answers in the file `torch_snippets_and_errors.txt`. The first five lines are already filled to get you started.

Instructions: For each of the following bugs/errors/behaviors, write a snippet of code that does it, then write a line in `torch_snippets_and_errors.txt` saying what the error message is (if there is an error message) or what the behavior is (if there is no error message). There are essentially three possible outcomes of each snippet:

- **It runs without error and produces the expected output.** In this case, you should write a line in `torch_snippets_and_errors.txt` saying what the correct and expected behavior was.
- **It runs without error but produces unexpected output.** In this case, you should write a line in `torch_snippets_and_errors.txt` saying what you expected and what the unexpected outcome was.
- **It produces an error message.** In this case, you should write a line in `torch_snippets_and_errors.txt` with the last line of the error message (following the examples in `torch_snippets_and_errors.txt`).

Snippets to write:

1. Add two PyTorch tensors of shape (3, 4) to a PyTorch tensor of shape (4, 3). For example, you could do `torch.randn(3, 4) + torch.randn(4, 3)`
2. Add a PyTorch tensor of shape (3, 4) to a PyTorch tensor of shape (4,). For example, you could do `torch.randn(3, 4) + torch.randn(4)`
3. Add a PyTorch tensor of shape (3, 4) to a PyTorch tensor of shape (4, 1). For example, you could do `torch.randn(3, 4) + torch.randn(4, 1)`
4. Add a pytorch tensor of dtype `torch.float32` to a pytorch tensor of dtype `torch.float64`. For example, you could do `torch.tensor([0, 1, 2], dtype=torch.float32) + torch.tensor([3, 4, 5], dtype=torch.float64)`
5. Create a tensor of all zeros of dtype `torch.uint8`, then try to assign it a value of `-3.14`. For example, you could do `x = torch.zeros(3, dtype=torch.uint8); x[0] = -3.14`

6. Add two pytorch tensors of the same shape and dtype but on different devices (e.g. CPU and GPU). For example, you could do `torch.randn(3, 4) + torch.randn(3, 4).cuda()`
7. Generate some random data to make a scatter plot, then try to plot it with the matplotlib function `plt.scatter`. For example, you could do `plt.scatter(torch.randn(100), torch.randn(100)); plt.show()`.

   No need to include the plot anywhere. Just write down whether it worked or not (do you expect matplotlib, which was designed to plot numpy arrays, to know how to work with pytorch tensors?)
8. Repeat number 7 but where the tensors are on the GPU. For example, as a one-liner you could do `plt.scatter(torch.randn(100).cuda(), torch.randn(100).cuda()); plt.show()`
9. Repeat number 7 but where the tensors have the `requires_grad` attribute set to `True`. For example, you could do

   ```
   x = torch.randn(100, requires_grad=True)
   y = torch.randn(100, requires_grad=True)
   plt.scatter(x, y)
   plt.show()
   ```
10. Repeat number 9 but use `plt.scatter(x.detach(), y.detach())`
11. Create a PyTorch tensor and increment it in-place using the `+=` operator. For example, you could do `x = torch.zeros(3); x += 1`
12. Repeat number 11, but with a tensor that has the `requires_grad` attribute set to `True`. For example, you could do `x = torch.zeros(3, requires_grad=True); x += 1`
13. Calculate the derivative dy/dx where $y = x^2$ at the point $x = 100$. For example, you could do `x = torch.tensor([100.], requires_grad=True); y = x**2; print(torch.autograd.grad(y, x))`. The expected behavior here is that you get a tensor whose value is the derivative of `y` with respect to `x` at the point `x=100`. From calculus, we know $d/dx(x^2) = 2x$, so the expected output is `[200.]`.
14. Repeat number 13 but where `x` is a tensor with more than one element, for example `x = torch.tensor([100., 200., 300.], requires_grad=True)`
15. Repeat number 13 but where `y = torch.log(torch.exp(x))`. From calculus, we know that $d/dx(\log(e^x)) = 1$. Is that what you get?
16. Try to create a roughly 20GB array on CUDA. A single float32 has 4 bytes, so you'll need to create an array with 5 billion elements, such as a tensor of shape (5000, 1000, 1000). For example, you could do `torch.zeros(5000, 1000, 1000, dtype=torch.float32, device='cuda')`.

(Note: since GPUs are shared, this one has the potential to break *other* people's code, and you should test it out only enough to understand what's going on, then stop.)

17. PyTorch shapes images differently than OpenCV. In OpenCV we saw arrays of dtype np.uint8 and shape `(h,w,c)` where `c=1` for grayscale images and `c=3` for BGR or other color spaces. In PyTorch, we'll more often be dealing with tensors of dtype `torch.float32` and values that range in `[0.0, 1.0]` (just like the `uint8_to_float` conversion we've been doing in numpy up until now). In PyTorch, the shape will be `(c,h,w)` instead of `(h,w,c)`. Let's look at an example... I downloaded a famous small color image dataset called CIFAR-10 to the `/local/sandbox/csci631/datasets/cifar10/` directory on all three course servers. You should have read access but not write access to everything in `/local/sandbox/csci631/`. Run the following (all of this **should** work and if it doesn't, we might need to fix some server configuration things)

    ```
    import torchvision
    import matplotlib.pyplot as plt

    transform = torchvision.transforms.ToTensor()
    dataset = torchvision.datasets.CIFAR10(
        "/local/sandbox/csci631/datasets/cifar10",
        train=True,
        transform=transform
    )
    img, label = dataset[0]
    print(img.shape, img.dtype, img.min(), img.max(), label)
    ```

    You don't need to add anything to `torch_snippets_and_errors.txt` for the above, but you should run it and make sure it works. Note that for CIFAR-10, the images are 32x32 pixels. Now, let's try to display the image with matplotlib. Run the following:

    ```
    plt.imshow(img)
    plt.show()
    ```

    This should give an error. Put the error as entry 17 in `torch_snippets_and_errors.txt`.

18. Now, let's try to fix it. Run the following:

    ```
    plt.imshow(img.permute(1, 2, 0))
    plt.show()
    ```

    Write down what you see in `torch_snippets_and_errors.txt` as entry 18 (something like "I see a picture of...").

Collaboration and Generative AI disclosure

Did you collaborate with anyone? Did you use any Generative AI tools? Briefly explain what you did in the `collaboration-disclosure.txt` file.

Reflection on learning objectives

This is optional to disclose, but it helps us improve the course if you can give feedback.

- what did you take away from this assignment?
- what did you spend the most time on?
- about how much time did you spend total?
- what was easier or harder than expected?
- how could class time have better prepared you for this assignment?

Give your reflections in the `reflection.txt` file if you choose to give us feedback.

Submitting

Use `make submission.zip` to generate a zip file that you can upload to Gradescope. You can upload as many times as you like before the deadline. To account for late penalties, contact the instructor or grader directly if you plan on uploading any further revisions after the deadline. (This is to prevent us from having to grade your work twice. Let us know so that we don't start grading files that will be replaced later).