General hints:

- Your code should work with *Python 3.9*.
- We suggest to adhere to the PEP8 style guide and use line lengths of up to 100. One way to achieve this is to format your code with black: `black {source_file_or_dir} -l 100` or add black to your IDE.
- `for` loops can be slow in Python, use vectorized `numpy` operations wherever possible (see assignment 1 for an example).

How to run the exercise and tests:

- See the setup instructions downloadable from our website for installation details.
- We always assume you run commands in the *root folder* of the exercise.
- If you're using miniconda, don't forget to activate your environment with `conda activate cvenv`
- Install the required packages with `pip install -U -r requirements.txt`
- Python files in the *root folder* of the repository contain the scripts to run the code.
- Some exercises use jupyter notebooks. It should be already installed from the requirements. Run `jupyter lab` to start the server.
- Some exercises contain unittests in folder `tests/` to check your solution. Run `python -m pytest .` to run all tests.
- To check your solution for the correct code style, run `pycodestyle -max-line-length 120 .`

---

1. Accessing the data

In this exercise, we will use two common optical flow datasets: FlyingThings and Sintel. They are provided under `/project/cv-ws2324/shared-data`.

2. Optical Flow

Optical flow is the task of estimating the motion between two consecutive frames. It is a regression task since the output is a motion field (u, v) for every pixel of the input image.

(a) (b)

Figure 1: **Optical flow estimation**: (a) Input image pair. (b) Ground truth optical flow between the input image pair.

In this exercise, we will use the FlowNet model, which has a U-Net architecture that consists of an encoder and a decoder. The encoder takes the two input images and learns a feature representation of them, while the decoder learns to construct the optical flow field for all pixels of the image. See the paper FlowNet: Learning Optical Flow with Convolutional Networks for more details. Even more details can be found in the follow-up papers FlowNet 2.0: Evolution of Optical Flow Estimation with Deep Networks and Occlusions, Motion and Depth Boundaries with a Generic Network for Disparity, Optical Flow or Scene Flow Estimation. Note that the models used in this exercise are smaller versions of the FlowNetS and FlowNetC models from the FlowNet 2.0 paper, so the resulting numbers will be slightly worse.

Given the ground truth optical flow, the network can be trained in a supervised manner. Obtaining optical flow ground truth is feasible for synthetic datasets like FlyingChairs and FlyingThings. However, in real dataset, this is very expensive (i.e., we need a motion field for every pixel of the image).

It is a common practice to train first on synthetic dataset to learn the general concepts of finding correspondence between the two frames and later finetune on a real dataset. In this exercise, we already provide a pretrained model that has been trained on the synthetic dataset FlyingThings. The main objective of the exercise is to explore two different approaches during the finetuning phase: supervised and unsupervised training.

Todo: Run the jupyter notebook `flow_visualization.ipynb` to get an intuition of the input data and model output. The visualization of the endpoint error is not implemented yet, you will do this in the next step.

Todo: Implement the function to estimate the evaluation metric functions (`aepe` and `pointwise_epe`) between two optical flows under `lib/metrics.py`.

Todo: Test the pretrained FlowNetC model on both Sintel and FlyingThings, change the dataset parameter appropriately. If your implementation is correct, you should get about 0.72 AEPE on Sintel.

```
python eval.py --output your_output_dir1 --model FlowNetC --dataset FlyingThings3D --auto_restore
pt/flownetc
python eval.py --output your_output_dir1 --model FlowNetC --dataset Sintel --auto_restore
pt/flownetc
```

Note: Start tensorboard with the command `tensorboard --logdir your_output_dir1` to view various logged metrics and data.

Todo: Finetune the pretrained FlowNetS model on the Sintel dataset by running:

```
python train.py --output your_output_dir2 --model FlowNetS --dataset Sintel --auto_restore
pt/flownets --completed_iterations 600000 --iterations 610000
```

Note: To evaluate your own trained models, either add them to the `models.toml` file or use `--restore /path/to/chkpt/checkpoint.pt` instead of the `--auto_restore` flag. Use `--help` to see all available options for the scripts.

Todo: Implement the unsupervised loss function which consists of two terms: data term (so-called photo-metric) and smoothness term (under `lib/loss.py`) as follows:

$$L_{\text{photometric}}(u, v; I(x, y, t), I(x, y, t+1)) = \sum_{i,j} p\Big(I(i, j, t) - I(i + u_{i,j}, j + v_{i,j}, t+1)\Big) \tag{1}$$

$$L_{\text{smoothness}}(u, v) = \sum_{i,j} \Big(p(u_{i,j} - u_{i+1,j}) + p(u_{i,j} - u_{i,j+1}) + p(v_{i,j} - v_{i+1,j}) + p(v_{i,j} - v_{i,j+1})\Big) \tag{2}$$

$$p(x) = (x^2 + \epsilon^2)^{\alpha} \tag{3}$$

For more details about these loss terms, please check the paper Back to Basics: Unsupervised Learning of Optical Flow via Brightness Constancy and Motion Smoothness.

Todo: Finetune the pretrained FlowNetS model on the Sintel dataset using your implementation of the unsupervised loss.

```
python train.py --output your_output_dir3 --model FlowNetS --dataset Sintel --auto_restore
pt/flownets --completed_iterations 600000 --iterations 610000 --photometric --smoothness_loss
```

Todo: Compare the two trained models by testing them on both Sintel and FlyingThings. Explain your observations.