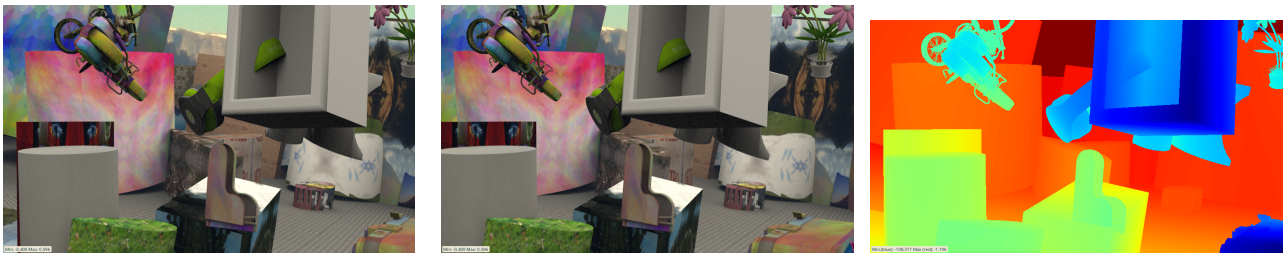
This exercise is on *disparity estimation*, which is the task of finding dense correspondence between two images from a stereo camera pair. Due to the given camera geometry, epipolar lines are known to be horizontal, i.e. correspondences are known to lie on the same horizontal line as the reference points. Disparity estimation can hence be seen as predicting the offset in x-direction between corresponding points (negative when matching from left to right, positive from right to left). Two images from a stereo image pair and the ground truth disparity are shown in Figure 1.



(a) (b) (c)

Figure 1: **Disparity estimation between a stereo camera image pair**: (a) Left camera image. (b) Right camera image. (c) Disparity from left to right, i.e. negative scalar values that indicate the shift in x-direction between corresponding points from the left to the right image. Blue denotes large shifts and red denotes small shifts. Note that objects close to the camera have larger displacements than far away objects.

FlowNet [1] and DispNet [2] are established architectures for the tasks of optical flow and disparity estimation. Note that disparity estimation can be seen as a constrained form of optical flow estimation and approaches for optical flow estimation can hence be also applied for disparity estimation. Both the FlowNet and the DispNet exist in a "S" (Simple) and a "C" (Correlation) variant. The "Simple" variants simply concatenate both input images and rely on the training for learning to search corresponding points. The "Correlation" variants explicitly use a correlation layer within the network, that correlates reference points with points in the other image at fixed offsets. For details, see [1].

In this exercise, we first set up a DispNetC by changing the correlation layer of the FlowNetC from exercise 5. Then we will evaluate both the DispNet and the FlowNet on a disparity estimation task. Finally, we apply the DispNet to two images that were not captured by a stereo camera. For this, we need to rectify the images before feeding them to the network.

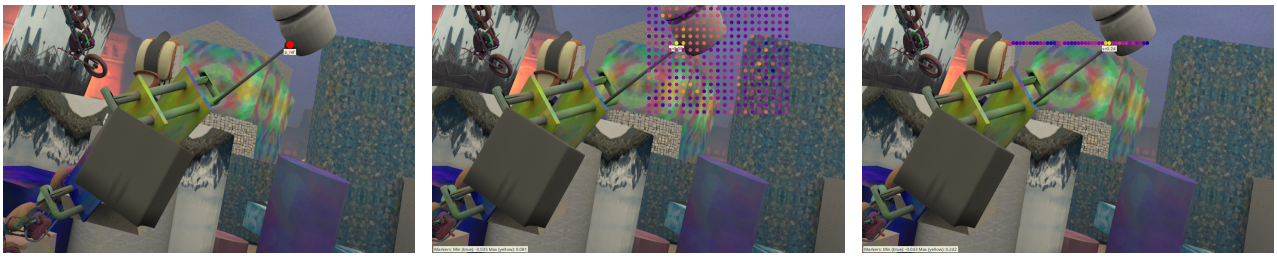
1. Installation

In your miniconda environment, install PyTorch and TorchVision from here, the exact command to setup with GPU support depends on your system. On the pool computers use this command:
`conda install -y pytorch torchvision pytorch-cuda=11.8 -c pytorch -c nvidia`
Afterwards check if your GPU is detected (the command should print True):
`python -c 'import torch; print(torch.cuda.is_available())'`
In case you get the error `chardet not found`, run `pip install chardet`
See learn the basics and other tutorials to get started with PyTorch.

2. Adapt FlowNetC correlation layer to set up a DispNetC

DispNet is a deep network architecture for disparity estimation. A DispNetS differs from a FlowNetS only in some details (e.g. a single output channel and ReLUs at the output). In addition to this, a DispNetC differs from a FlowNetC by using a different correlation layer that exploits the restricted correspondence search space. For this, points in the reference image are correlated with points from the other image not in a quadratic grid as in a FlowNetC, but in the same row, as shown in Figure 2.

We provide weights for a trained DispNetC and DispNetS. To use the DispNetC, your task is to adapt the



(a) (b) (c)

Figure 2: **FlowNetC and DispNetC correlation layers**: (a) Reference image (left stereo camera image) with a selected reference point (red). Note that Correlation is done for all points in the reference feature map. Furthermore, the correlation is done on features not raw pixel values. (b) Correlation points in the other image (right stereo camera image) in a FlowNetC correlation layer. Colors indicate the correlation score. (c) Correlation points in the other image in a DispNetC correlation layer.

correlation layer by implementing the class `DispSamplingPoints` in the file `lib/corr.py`.

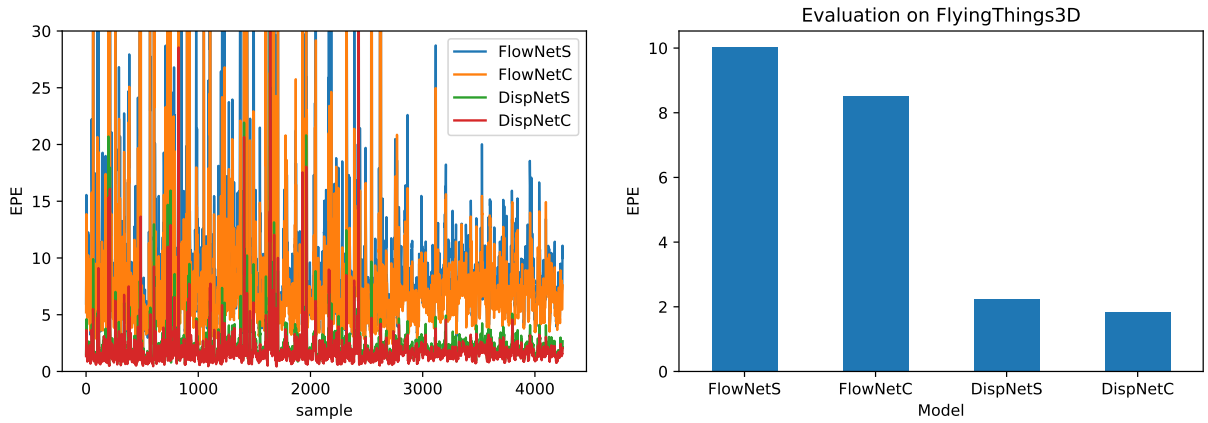
Todo: Complete the class `DispSamplingPoints` in the file `lib/corr.py`. Test your implementation with by running the command `python -m pytest`

In the next task, we will run DispNetC.

3. Evaluate FlowNet and DispNet for disparity evaluation

In principle, disparity between a stereo camera image pair can be estimated with a FlowNet or a DispNet. In case of a FlowNet, one simply ignores the y-component of the optical flow and only considers the x-component. However, a DispNet should give better results, as it is more specialized for the task at hand (regarding the correlation layer and training data).

Your task is to evaluate a FlowNetS, FlowNetC, DispNetS, and DispNetC for disparity estimation on the FlyingThings3D test set. Plot the results in a similar form as shown in Figure 3.



(a) (b)

Figure 3: **FlowNet and DispNet results on FlyingThings3D**: (a) Plot of the average EPEs per sample. (b) Plot of the average EPEs on the whole dataset.

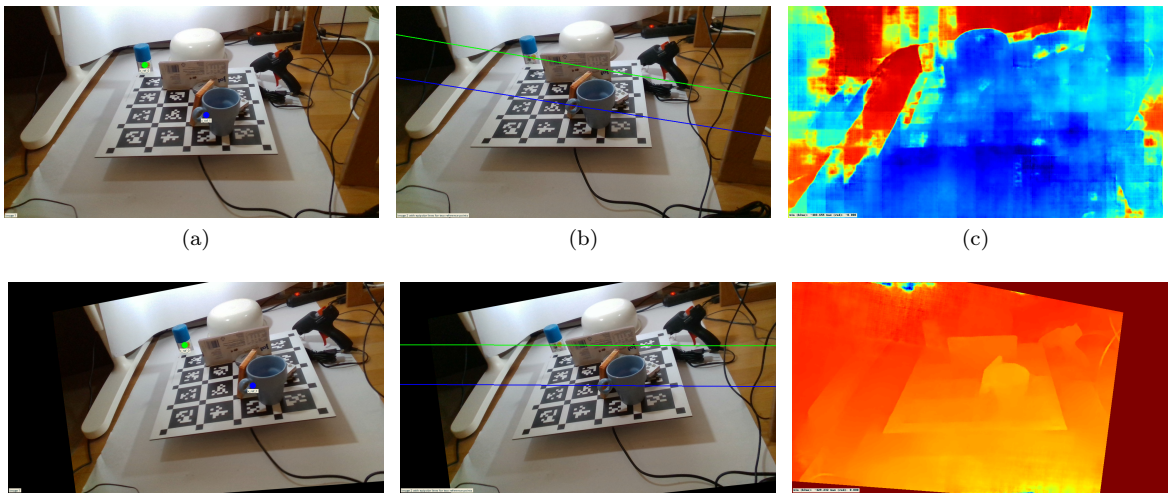
Todo: Evaluate FlowNet and DispNet by following the instructions in the jupyter notebook . You can **either a)** connect your notebook with a GPU machine **or b)** run the evaluations in the command line, then load the results with the notebook.

For option **a)**, we recommend running jupyter notebook using vscode: To do that, you need to connect vscode via remote ssh directly to one of the pool machines, not the login node, and then select an appropriate kernel (python of a conda environment). Please check the FAQ under 'Remote SSH in vscode directly to a pool machine' https://github.com/lmb-freiburg/cv-exercises/blob/master/FAQ.md.

For option **b)**, the commands are given in the notebook.

4. Apply DispNet to images from a monocular camera

Next, we want to apply DispNet to images that were not captured simultaneously with a stereo camera but sequentially with a monocular camera. The input images are shown in Figure 4. For this, the input images need to be rectified before being fed to the network. This means warping the images based on the known camera intrinsics and extrinsics such that the images are coplanar and epipolar lines are horizontal.



(a) (b) (c)

(d) (e) (f)

Figure 4: (a), (b) Input images from a monocular camera with epipolar lines for two reference points. In this general case, epipolar lines are not horizontal which conflicts with the DispNet assumptions. (c) DispNet prediction when feeding the non-rectified images. (d), (e) Image pair after rectification. Note the horizontal epipolar lines. (f) Predicted disparity when applying DispNet to the rectified images and warping the prediction back to the original viewpoint.

Your task is to implement a function that rectifies an image pair with known camera calibration. As intermediate steps, you have to implement functions for projecting 3d points to an image plane, computing epipoles, and computing the essential and fundamental matrices. The notebook `exercise_disparity_estimation.ipynb` will guide you step-by-step through the implementations.

Todo: Implement the functions `project_to_image`, `get_epipole`, `compute_essential_matrix`, `compute_fundamental_matrix`, `rectify_images` in the file `lib/utils.py`. Use the notebook `exercise_disparity_estimation.ipynb` to check your implementations.

You can then run DispNet on the rectified images:

```
python inference.py --model DispNetC --path data/realthings --rectify --auto_restore
mw/dispnetc
```

5. Bonus: Train a DispNetS

The provided code includes training code for a DispNetS and DispNetC. As a bonus, you can train a DispNetS yourself with the following command:

```
python train.py --model DispNetS --iterations 100000 --output output/dispnets_things_100k
```

We suggest training only for 100k iterations to reduce the training time (should take around 7 hours on the given hardware). Of course you can train longer, if you have enough time.

You can evaluate the resulting model with the following command (you should get an EPE of around 4.2):

```
python eval.py --model DispNetS --output output/dispnets_things_100k --restore
output/dispnets_things_100k/checkpoints/checkpoint-model-iter-000100000.pt
```

The tensorboard logs are stored in the output folder and contain important training curves (loss, learning rate, etc.), as well as qualitative outputs from the training and the evaluation.

References

[1] Alexey Dosovitskiy et al. "Flownet: Learning optical flow with convolutional networks". In: *ICCV*. 2015.
[2] Nikolaus Mayer et al. "A Large Dataset to Train Convolutional Networks for Disparity, Optical Flow, and Scene Flow Estimation". In: *CVPR*. June 2016.