Machine Learning Engineer Nanodegree

Capstone Proposal

Christoph Soehnel
December 2nd, 2017

Proposal

Domain Background

In Computer Vision not only the pixel content, i. e. the visual appearance, of an image is important but also the meaning of the pixel values with respect to a specified reference frame. This allows us to measure distances like depths in the actual world what is useful for many applications from computer-aided inspection to robotics and self-driving cars.

While there are others, depth estimation in the pure image domain can be categorized in two main approaches:

1. Stereo Vision
2. Structure from Motion

Since the first approach requires at least two cameras that record time-synchronized images, it is somewhat costly to build what is an issue for industries that want to offer computer vision solutions in a low-cost high-volume market. The second approach only utilizes one (moving) camera but has drawbacks with stationary scenes and moving objects within the scenes what makes it less robust compared to Stereo Vision. While technically more challenging, there has also been research to the problem of monocular depth estimation, especially recently with the raise of Deep Learning, e. g. Godard et al.[1]

Problem Statement

I would like to address the problem of dense depth estimation using only monocular image cues, i. e. a single image.

As depth is a function of disparity, i. e. the translation between corresponding pixels in two images, the actual task is dense disparity estimation without any second frame, neither from a second camera nor from the same moving camera. Since I want to generate an estimate for every single pixel of an input image and not only for a subset of them, the result is called dense, not sparse.

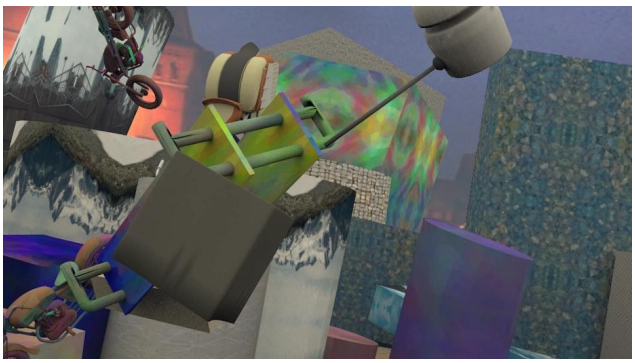
Datasets and Inputs

There are several datasets for disparity estimation available. Most of them lack either dense ground truth data, because the disparity reference is generated through other imperfect algorithms[2], or

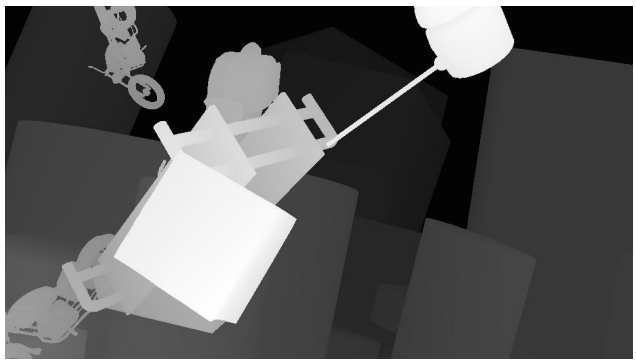
1 http://visual.cs.ucl.ac.uk/pubs/monoDepth/
2 https://www.cityscapes-dataset.com/

contain not enough samples[3] for Deep Learning approaches, Mayer et al.[4] introduced a new dataset to overcome these issues.

Throughout this project the dataset “FlyingThings3D” will be used. It consists of 21818 training images and 4248 testing images. Each image has a spatial resolution of 960 x 540 px and comes with a corresponding dense ground truth disparity image of the same resolution. Both, the training/testing[5] images and the disparity[6] images are available on the Web[7].



Sample image from the test set



Corresponding ground truth disparity image

Setup

The software setup to be used during this project consists of Ubuntu GNOME 16.04.3 LTS (64 Bit), NVIDIA Cuda 9.0.176/cuDNN 7.0.4 and Keras 2.1.1 on top of Tensorflow 1.4.1 (compiled from source). The code will be written in Python 3.5.2. A complete list of required packages will be supplied in the project report.

The hardware setup to be used for training, testing and runtime measurements consists of an AMD Ryzen 1700X, 32 GB of RAM and a single NVIDIA GeForce GTX 1080 Ti with 11 GB of VRAM.

Solution Statement

Inspired by the publication of Isola et al.[8], I would like to apply a Conditional Adversarial Network to transform a single input image into its corresponding disparity image. This will be done by letting a “Generator” network learn to generate a disparity image from an input image and a noise vector. A “Discriminator” network learns to distinguish these generated disparity images from real disparity images. The training converges, when the “Generator” produces disparity images of such high quality that the “Discriminator” is not able to do its job. The following scheme is copied from Isola et al.:

3 http://sintel.is.tue.mpg.de/
4 https://arxiv.org/pdf/1512.02134.pdf
5 https://lmb.informatik.uni-freiburg.de/data/SceneFlowDatasets_CVPR16/Release_april16/data/FlyingThings3D/raw_data/flyingthings3d__frames_cleanpass_webp.tar
6 https://lmb.informatik.uni-freiburg.de/data/SceneFlowDatasets_CVPR16/Release_april16/data/FlyingThings3D/derived_data/flyingthings3d__disparity.tar.bz2
7 https://lmb.informatik.uni-freiburg.de/resources/datasets/SceneFlowDatasets.en.html
8 https://phillipi.github.io/pix2pix/

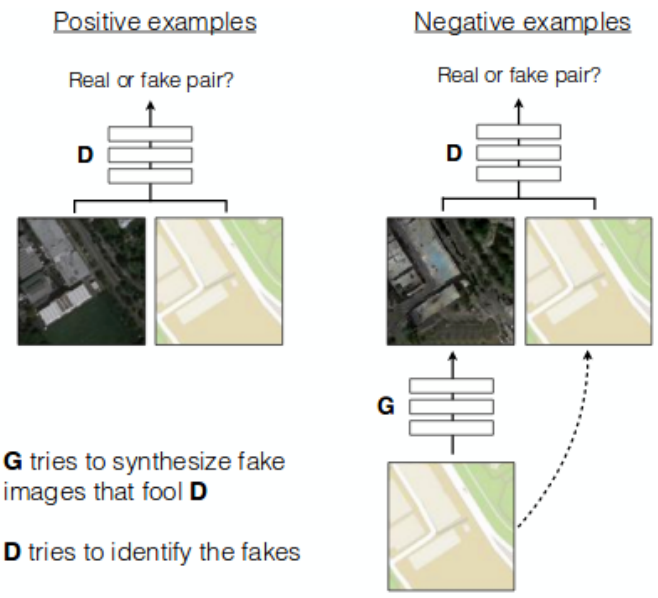


Figure 2: Training a conditional GAN to predict aerial photos from maps. The discriminator, D, learns to classify between real and synthesized pairs. The generator learns to fool the discriminator. Unlike an unconditional GAN, both the generator and discriminator observe an input image.

The beauty of this approach is, that the loss function will be learned implicitly during training of the networks avoiding the need for formulating an appropriate loss function by hand.

As far as I know, Conditional Adversarial Networks have not been used for depth/disparity estimation of single images yet.

Benchmark Model

Mayer et al. supply benchmark results for their approach ("DispNet") on their dataset and also compare it to a classic Stereo Vision approach ("SGM"). Within the capstone project I will compare my model to at least both of the mentioned models by calculating the error to the given ground truth data in the same fashion. With this method I will be able to state how my upcoming model performs in contrast to the baseline.

Evaluation Metrics

To be consistent with Mayer et al., I will use the endpoint error (EPE)[9] to quantify the performance of my solution and to compare it against the benchmark model:

$$E_{EPE_{disp}} = \frac{1}{n} \sum_{i \in I} |d_e^i - d_g^i|$$

This is the mean of the absolute errors over all pixels in the image, comparing the estimated disparity value of a pixel against its ground truth disparity value.

9 https://arxiv.org/pdf/1612.02590.pdf

Project Design

The project will be split into 6 phases:

1. **Preparation**
   After downloading the dataset, I will write the necessary code to inspect the data and to check it for consistency. Basic handling of the WEBP[10] format for the images and the PFM[11] format of the disparity images is also part of this phase. In addition I will ensure that all necessary libraries and dependencies are available on the development machine (→ Setup). As an additional step I will split the training set into training and validation sets to have some data to tune the network on.

2. **Implementation**
   I will implement the Conditional Adversarial Network according to the formulation of Isola et al. with Python in Keras. For the first attempt I will stick to the exact layout and parametrization given by the authors. Also every other components of the program, like evaluation metric and overall framework will be implemented here.

3. **Training**
   After the implementation has been completed, I will train the network using the chosen dataset and cater for basic issues that might come up with the first implementation. During this phase I will produce the first numbers regarding the model performance on the validation set.

4. **Tuning**
   In the “Tuning” phase of the project, I will try to improve the model performance on the validation set by altering the model layout and the parameters of the training process. I will stop this phase when a reasonable good performance is reached or the results cannot be improved.

5. **Evaluation**
   During this phase I will do the evaluation run on the test set to get the final performance of my model.

6. **Report**
   After all results are ready, I will write the final report according to the given template and comparing the results of my final model to the benchmark model in a detailed discussion.

10 https://en.wikipedia.org/wiki/WebP
11 https://lmb.informatik.uni-freiburg.de/resources/datasets/SceneFlow/assets/code/python_pfm.py