Technical Report

IRI-TR-20-01

Corner detection of deformable fabric using Deep Learning

Albert Mitjans Coma
Marc Maceira
Guillem Alenyà

January, 2020

Institut de Robòtica i Informàtica Industrial

Abstract

Robotic manipulation of deformable objects such as a towel is one of the most challenging tasks in the field of service robotics. Their unpredictable shape and pose makes it difficult to identify the most relevant parts than can be used for grasping. In this report we design a deep neural network that finds all the visible corners of a wrinkled towel. We also create a dataset to train the network. The results obtained show that it is possible to detect the corners of a cloth, but with some constraints due to the small number of images inside the dataset.

Institut de Robòtica i Informàtica Industrial (IRI)
Consejo Superior de Investigaciones Científicas (CSIC)
Universitat Politècnica de Catalunya (UPC)
Llorens i Artigas 4-6, 08028, Barcelona, Spain
Tel (fax): +34 93 401 5750 (5751)
http://www.iri.upc.edu

Corresponding author:
Albert Mitjans Coma
tel: +34 93 401 0775
amitjans@iri.upc.edu
http://www.iri.upc.edu/staff/amitjans

1 Introduction

With the advent of technology, the use of service robots has been increasingly regularly for various applications ranging from communication, personal transport to household tasks. Although there is huge research going on in the field of Assistive robotics, the scope of handling garments by assistive robots is limited. Handling and manipulation of garments is difficult for the robot since it lacks advanced sensing capabilities like humans such as vision, touch, dexterity and ability to understand and process the information. When clothes are manipulated under unconstrained conditions, one of the key aspects to be solved is the determination of grasping points. Such grasping points can be used by domestic robots during cloth manipulation tasks.

In order to address this task, in this report we design a network that is capable of finding the visible corners of a wrinkled towel. In addition, we build a dataset of RGB and depth images to train the network so that, from a depth image of a wrinkled cloth, it will output a heatmap with gaussians on the possible corners of the towel.

This technical report is structured as follows. Section 2 explains the methodologies used to build the dataset for training the network. Afterwards, in section 3, the structure of the neural network is introduced. The training of the network is explained in section 4. Finally we present our results and conclusions in sections 5 and 6. In addition, the project's README file is annexed at the end of the report.

2 Dataset

For training the deep convolutional network, we use depth images of clothes. The advantage of using depth images is that the framework does not depend on color of the clothes, and can generalize easily to other similar types of clothes.

The dataset is acquired by capturing the depth and RGB images of a towel with its corners marked with colored tape. The images were taken always from the same viewpoint and in every image the towel has different shape and position. The complexity of the shape of the towel increases with the image number, with folded towels in the initial images and completely wrinkled towels in the last ones.

Color is used to annotate the images, and thus speed the process of creating a dataset, which is very time-consuming. We measured the corner's coordinates using color segmentation with OpenCV. Thanks to the marks placed on each corner, we can obtain their coordinates by simply segmenting each color in the HSV (Hue, Saturation, Value) representation and then keeping the coordinates of the center value of each color. To avoid any external object from interfering with the color segmentation, the background of the image is covered with paperboard, as can be seen in Figure 1. We tried to improve the rgb illumination with spotlights but we lost the depth from the sensor.

Two different cameras were tested for capturing the dataset: an Intel Realsense and an Asus XtionPRO Live. Finally the Asus Xtion camera was used, since its depth sensor was the only one able to detect the little changes in depth produced by the towel.

We acquired a total number of 400 images. Each image is defined by 3 files: the depth and RGB image, and a CSV containing the coordinates of all its corners (corners which are not visible are fixed with a value of -1). All the data is separated into two groups: one for training (80%) and another for validation (20%).

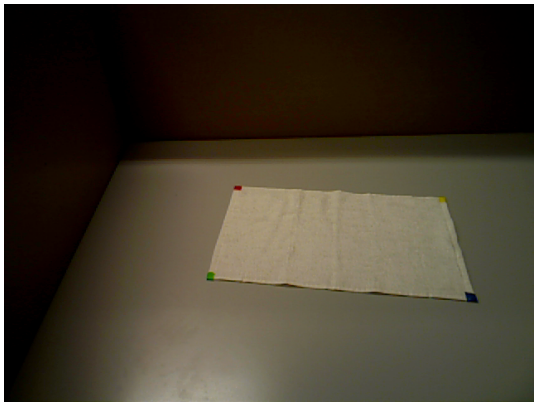

Figure 1: Raw RGB image taken by an Asus XtionPRO LIVE camera.

Since the RGB image is used to obtain the coordinates of each corner, but this coordinates are to be used on the depth image, it is very important that both RGB and depth images are synchronized. That is why we subscribe to the following topics from the Xtion camera:

- */camera/depth_registered/hw_registered/image_rect_raw*
- */camera/rgb/image_rect_color*

The depth registered processing applies a transform to the depth image to register its depth frame to the RGB frame. This can be done directly by the device (*hw_registered*) or by a software registration pipeline (*sw_registered*). Even though with both methods you get the same results, they represent them differently. The *sw_registered* gives 0-value to the pixels that can't be measured by the depth sensor and outputs a uint16 image while the *hw_registered* assigns a NaN value to these pixels and outputs a float32 image.

For the creation of the dataset, the software registered processing was used. For the real time version, the images were captured with a Raspberry Pi. In this case, we noticed that the software registered version had a big delay that prevented us from using the camera in real time. Therefore, the actual code uses the hardware registered processing when capturing images. It is important to mention that, since the two methods give outputs with different data types, they must be saved with different file-formats (TIFF for the uint16 images and PNG for the float32 images). Consequently, if new images are captured for extending the dataset, the code must be modified to detect this new file-format.

ROS uses its own Image encoding (*sensor_msgs/Image.msg*). In order to work with these messages, they need to be converted to OpenCV images. To do so, the ROS package *cv_bridge* is used.

Finally, the image is cropped from 480 $\times$ 640 pixels to a size of 304 $\times$ 496 pixels to eliminate all the parts that don't contribute to our problem (like the paperboard or the zero-values generated by the depth registered processing).

The code used to capture images can be found in this github repository.

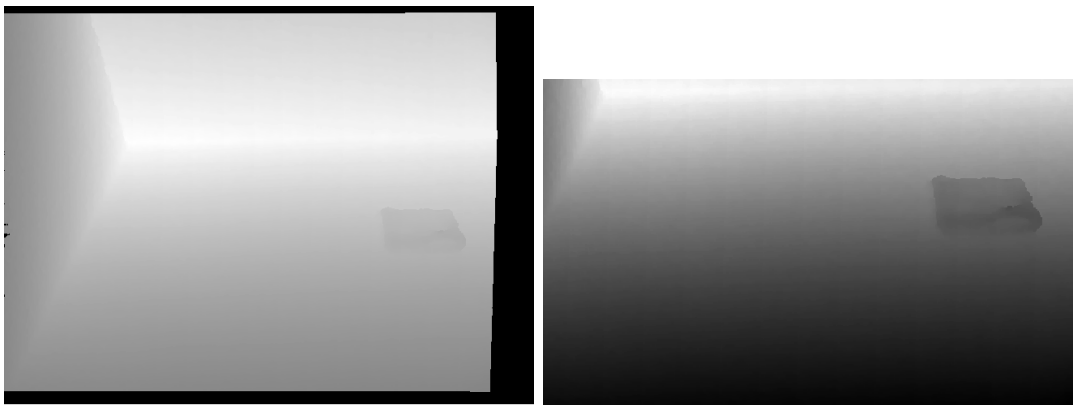

Figure 2: Original depth image (480 × 640 pixels) taken by the camera and its cropped version (304 × 495 pixels).

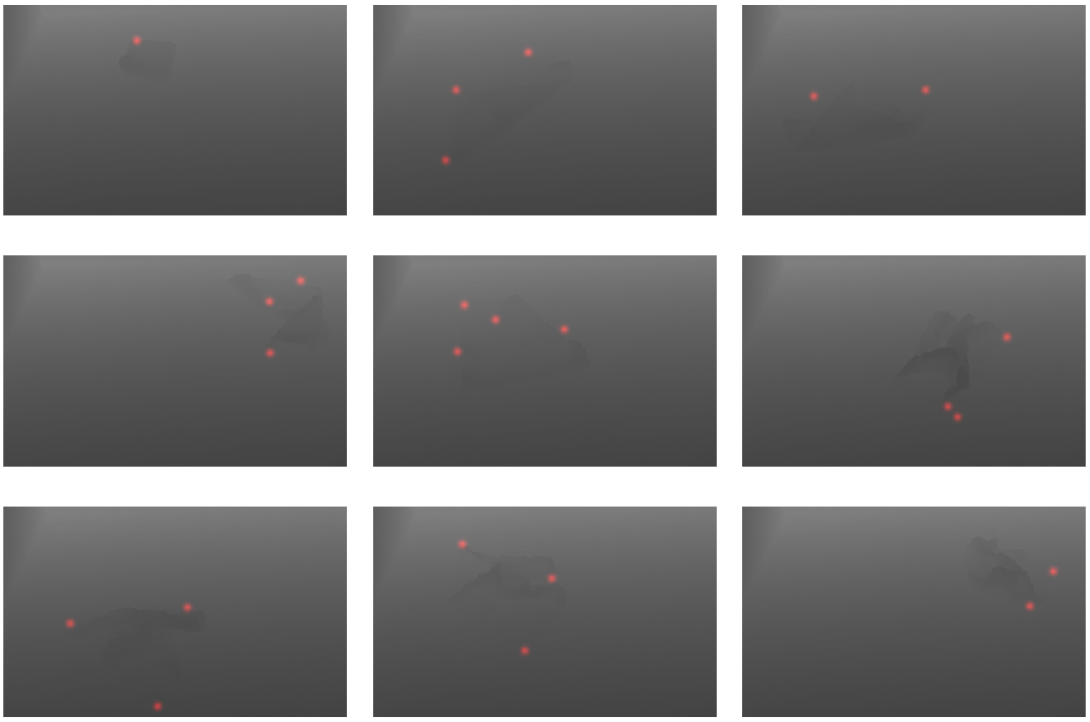

Figure 3: Ground truth images (with annotations) from the dataset.

3 Hourglass network

The problem raised in this report must be solved by what is called object detection. Object detection consists on determining where objects are located in a given image (object localization) and which category each object belongs to (object classification). In this specific work we don't need to implement object classification since all the objects we want to detect are from the same class (corners). Object localization is divided into two stages: informative region selection and feature extraction.

Informative region selection. As different objects may appear in any position of the image and have different aspect ratios or sizes, it is a natural choice to scan the whole image with a multi-scale sliding window. Although this exhaustive strategy can find out all possible positions of the objects, its short-comings are also obvious. Due to a large number of candidate windows, it is computationally expensive and produces too many redundant windows. However, if only a fixed number of sliding window templates are applied, unsatisfactory regions may be produced.

Feature extraction. To recognize objects, we need to extract visual features which can provide a semantic and robust representation. However, due to the diversity of appearances, illumination conditions and backgrounds, it's difficult to manually design a robust feature descriptor to perfectly describe all kinds of objects.

Thanks to the emergency of Deep Neural Networks (DNNs), a significant gain is obtained with the introduction of Regions with CNN features (R-CNN)[3]. DNNs, or the most representative CNNs, act in a quite different way from traditional approaches. They have deeper architectures with the capacity to learn more complex features than the shallow ones. Also the expressivity and robust training algorithms allow to learn informative object representations without the need to design features manually.

Since the proposal of R-CNN, a great deal of improved models have been suggested, including YOLO[2], which accomplishes object detection via a fixed-grid regression. However, we found another model which adapted better to our dataset since its size was way smaller than YOLO: the Hourglass network.

Hourglass networks have been successfully used to detect human body parts (Stacked Hourglass Networks for Human Pose Estimation[1]). Given an input image, the hourglass network pools it down to a very low resolution, and then uses upsampling to bring the image back to its original size. With this structure the network is capable of capturing information at every scale.

In this case, the output obtained is a heatmap where the network predicts the probability of a corner's presence at each and every pixel of the image.

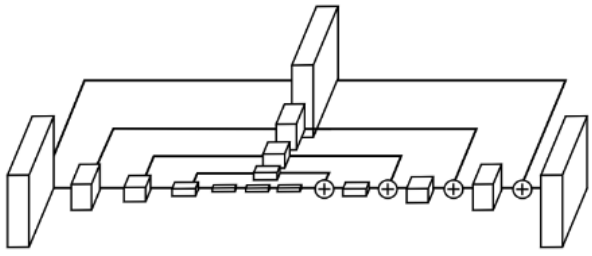

Figure 4: An illustration of a single "hourglass" module. Each box in the figure corresponds to a residual module.

The network's architecture is taken further by stacking 2 hourglasses end-to-end, feeding the output of the first one as input into the next. This provides the network with a mechanism for repeated bottom-up, top-down inference allowing for reevaluation of initial estimates and features across the whole image. The key to this approach is the prediction of an intermediate heatmap upon which we can apply a loss. Predictions are generated after passing through each hour-glass where the network has had an opportunity to process features at both local and global contexts. Subsequent stages of bottom-up, top-down processing allow for a deeper reconsideration of these features. Figure 5 shows how this implementation eliminates the false positives and emphasizes the true positives.

This network is used on RGB images and thus needs a 3-channel image as input. Since the depth image consists of only 1 channel, we tried several options:

1. Use 3 copies of the depth channel.
2. Compute the edges and the contours of the image (with Pillow's ImageFilter Module) and use them as 2^{nd} and 3^{rd} channels.
3. Add an initial 1x1 convolutional layer that uses 3 different kernels and thus generates a 3-channel image from the depth channel.

The edges of the image are obtained by filtering the image using the following convolution matrix:

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & -8 & -1 \\ -1 & -1 & -1 \end{bmatrix} .$$

We will discuss which one is better in section 5.

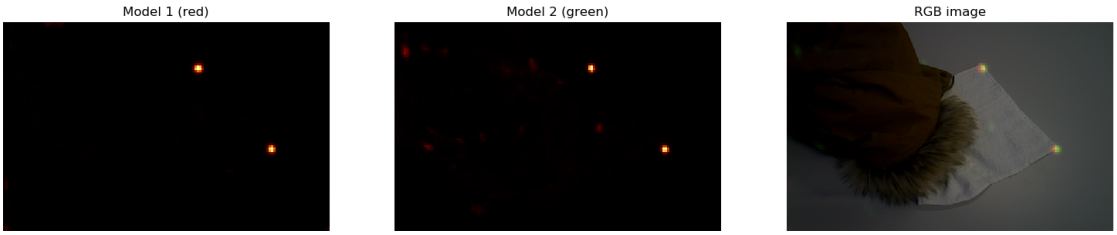

Figure 5: Output of a model that uses the heatmaps generated by both hour-glasses to compute the loss (Model 1) and another one that uses only the last heatmap (Model 2).

4 Training

The network is trained on the dataset mentioned before, composed of 400 images (320 training, 80 testing). For data augmentation, we randomly apply transforms on images that include horizontal flips, random crops and scaling. All input images are then padded into square images of 495×495 pixels (the hourglass accepts images of many sizes thus images with lower resolution can be used if needed).

Even with data augmentation, the dataset is still small to train a network of this size. For this reason, training is done on a model already pre-trained on the MPII Human Pose dataset. This helps training to achieve better model performance quickly.

Given that the initial model was pre-trained with rmsprop as optimizer, we are required to use the same optimizer. We used other optimizers such as Adam (without the help of the pre-trained model) but we obtained poor results. We used several learning rates but we got the best results when using a learning rate of $5e^{-4}$ and dropping it by a factor of 10 every 30 epochs. Batch normalization is also used to improve training. Training with 200 epochs takes about 170 minutes and a single forward pass of the network takes approximately 80ms.

For supervision, a Mean-Squared Error (MSE) loss is applied comparing the predicted heatmap to a ground-truth heatmap consisting of 2D gaussians (with standard deviation of 7 pixels) centered on the corners locations. This loss, as mentioned in section 3, is applied to the heatmap of every stacked hourglass of the model (in our case the network is formed by 2-stacked hourglasses). The final loss is just the sum of this individual losses.

We trained the network with smaller gaussians, but we got worse results. Given the size of the pieces of colored taped in each corner, it is hard for the coordinates to be in the actual point of the corner. If the gaussian created is small enough, sometimes the gaussian won't cover the actual corner. This leads to bad-labeled data and thus a worse performance of the network.

5 Results

5.1 Evaluation

Evaluation is done by comparing the maximums of the output's gaussians with the coordinates of all visible corners in the image. If a maximum falls within a certain distance from any of the corners that particular gaussian is considered as a true positive. The distance threshold has been set to 8 pixels, which is compatible with the accuracy of actual robotic grippers. By computing the number of true positives, false positives and false negatives the recall and precision of the network can be calculated with

$$Recall = \frac{true\ positives}{true\ positives + false\ negatives} \tag{1}$$

$$Precision = \frac{true\ positives}{true\ positives + false\ positives}\,. \tag{2}$$

Basically Recall will show the percentage of corners that the network found with respect to the total number of corners in the image. Precision is a calculation of the percentage of good answers of the network with respect to the total number of answers. In our problem, given that the network will be used to find grasping points, it is preferable to prioritize a high precision. It is more useful to get fewer grasping points than get all the grasping points with plenty of false positive.

As explained in section 2, not all images have all corners visible, also for some applications it is enough to detect a reduced number of corners. Thus, precision results are provided for detecting 1,2,3 and 4 corners in an image (from now on, this values will be called n-precision score, n corresponding to the number of corners). In the first case, for example, precision will be 1 if the max value of the network's output corresponds to a real corner and 0 otherwise. When computing the 2-precision score, we will look at the 2 detected gaussians with a higher value in its center, and so on. The precision value will only be computed for the image if the network's output has a number of gaussians equal or higher than the number of corners of the precision we are computing (if the network's output has 3 positives, the value of the 4-precision score won't be updated).

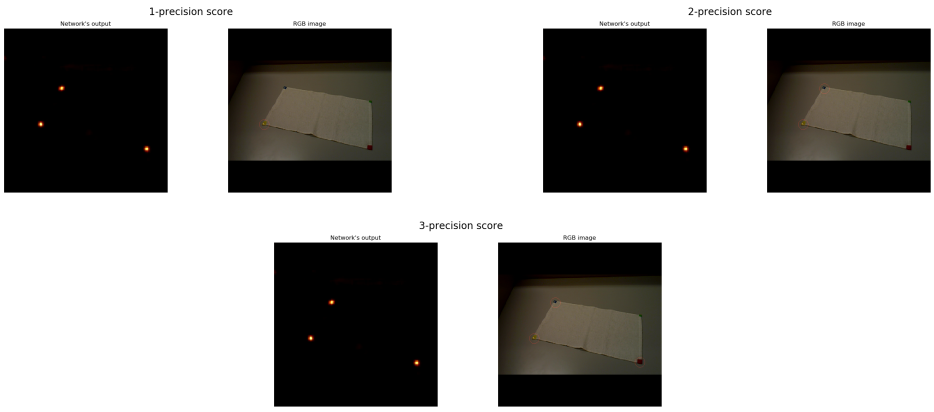

Figure 6: Graphic explanation of the 1-precision, 2-precision and 3-precision scores.

As it is said in section 3, the edges and contours of the images are used to give additional information to the network and thus improving its performance. To get a better look at the advantages of this filters, they are both implemented separately. Figure 7 shows the results obtained and compares them to the results of the original implementation (depth/depth/depth). It can be seen how the contours filter significantly improves the network's 3-precision score while the edges filter doesn't carry much additional information for the network, since the precision is similar (or even a bit worse) than when using only depth for the 3 channels of the input image. However, we obtain the best precision when using a combination of the 3 different channels.

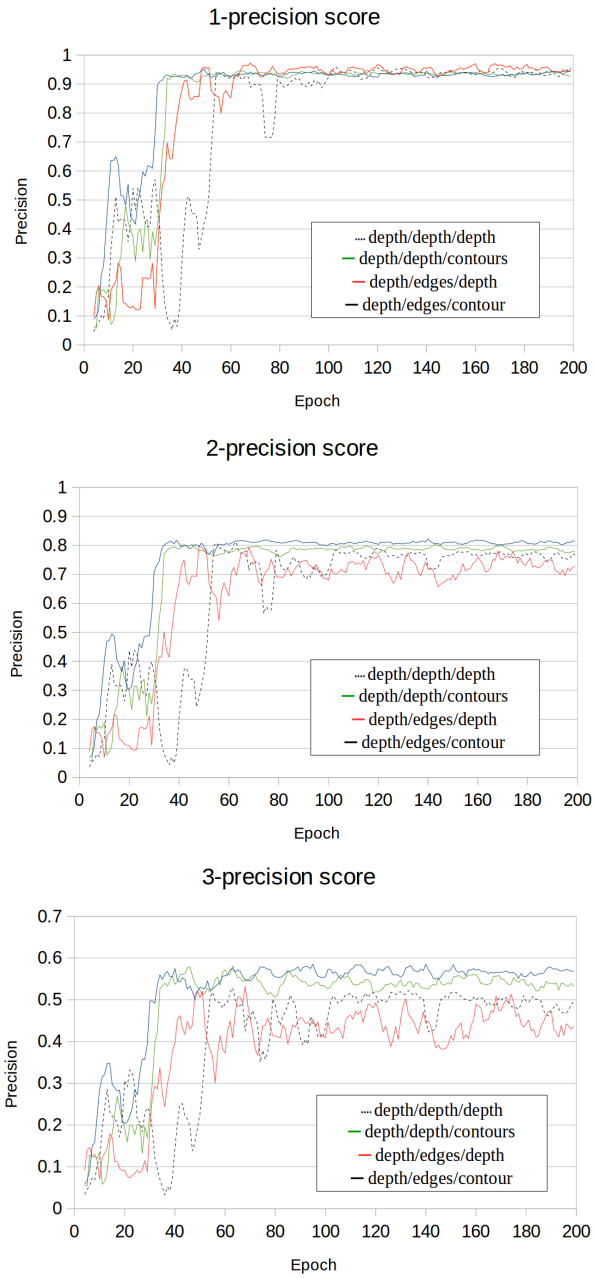

Figure 7: Comparison of validation precision as training progresses.

In Figure 8 we compare the results of the 3 approaches used to fill the missing channels of the input. Although all the different approaches detect at least 1 corner with the same precision, when using 3 copies of the depth channel the precision gets worst for multiple detections. A precision similar to the aforementioned depth/edges/contours (from now on called DEC) implementation is obtained when inserting the network a 2-D image and using a convolutional layer to transform it into a 3-D image with 3 channels.

In the figures mentioned above the 4-precision score is not shown because of its irrelevance (due to the small number of validation images where all the corners are visible).

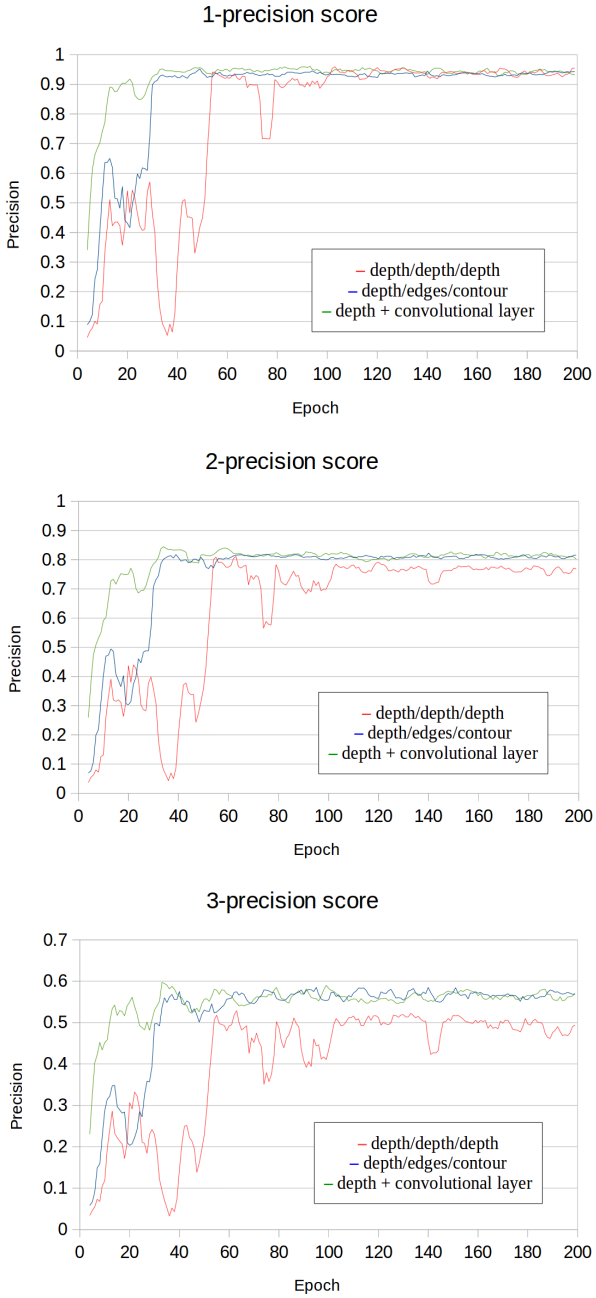

Figure 8: Comparison of validation precision as training progresses.

5.2 Real-time display

By following the steps in Appendix A, the output of the network can be displayed in real-time. This can be used to test the trained models with images from outside the dataset.

In Figure 9 the two best models obtained in section 5.1 are compared. Model 1 and Model 2 correspond to the DEC model and the model with an extra convolutional layer respectively. Even though both models have almost the exact same precision, it looks like when testing both models in real-time, the second model tends to generate more false positives than the first one.

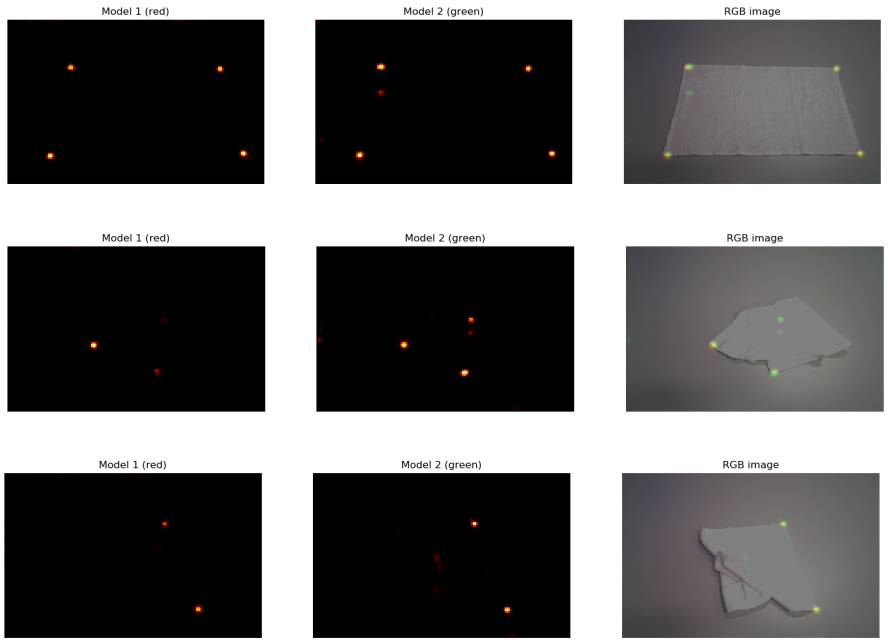

Figure 9: Comparison of the output of 2 different models.

Figure 10 shows how the network fails at detecting the corners that aren't located right on top of the table. This can be attributable to the great similarity (in the depth image) between this particular corners and any wrinkle in the towel. Given that, during training, the net is fed with many wrinkles in the towel that are not annotated but few annotated corners of this characteristics, the net is not able to detect them.

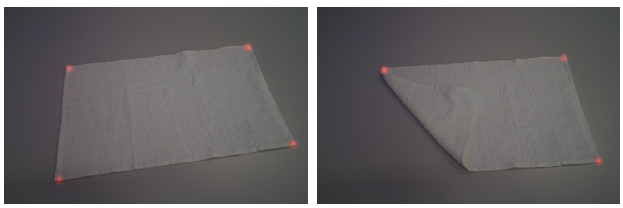

Figure 10: Superposition of the output of the network on the RGB image.

6 Conclusions

In this work, we have provided a methodology to detect corners of a towel in a table. The work consists in the acquisition of the dataset and the training of the network. The network is able to detect visible corners of a wrinkled towel in a controlled environment. However, it is observed how the precision of the detection is highly dependent on the angle of the camera and the distance between the camera and the towel. This is due to the significantly small size of our dataset, composed of images taken with the same vision angle.

As a future work, rotations to the images on the dataset should be applied in order to get more images with different viewpoints. Also, more images should be taken to increase the dataset and thus avoid the constraints seen in section 5.

A README (github)

Go to github.

Hourglass for corner detection of deformable fabric

This repo contains the code structure used for the detection of the corners of a wrinkled towel.

Installation

Create conda environment

```
conda create -n ENVIRONMENT_NAME python=3
conda activate ENVIRONMENT_NAME
```

Clone and install requirements

```
git clone https://github.com/AlbertMitjans/pytorch-corner-detection.git
cd pytorch-corner-detection/
conda install --file requirements.txt
```

Download pretrained weights

```
cd checkpoints/
bash get_weights.sh
```

Download dataset

```
cd data/
bash get_dataset.sh
```

Run test

Evaluates the model on the dataset.

```
python3 main.py --train False --ckpt checkpoints/best_ckpt/model.pth
```

Testing log

```
* Recall(\%): 54.830     * Precision(\%):  (97.727, 79.804, 41.202, 14.815)
```

The precision is computed for the (1, 2, 3, 4) corners detected with highest confidence (the gaussians with a highest value on its center).

Run train

Trains the network from scratch or from a given ckpt.

```
python3 main.py
```

Training log

```
Epoch: [5][300/312] Loss.avg: 0.3615    Recall(\%): 21.622
Precision num. corners (\%): (22.591, 18.563, 15.833, 16.809)
```

References

[1] Alejandro Newell, Kaiyu Yang, and Jia Deng. Stacked hourglass networks for human pose estimation. *CoRR*, abs/1603.06937, 2016.

[2] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *CoRR*, abs/1804.02767, 2018.

[3] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015.

IRI reports

This report is in the series of IRI technical reports.
All IRI technical reports are available for download at the IRI website
http://www.iri.upc.edu.