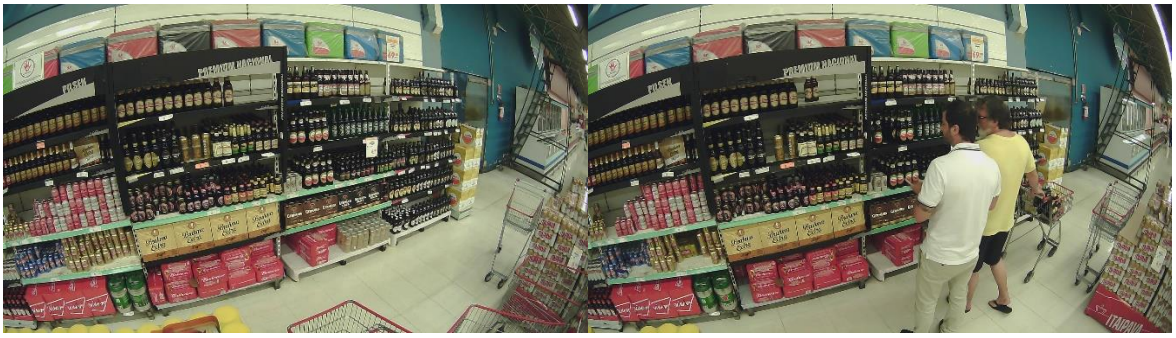
Image differences

Goal

Detect differences between two images using neural networks. The first usage scenario is to detect the product changes on a super markets beer shelf. Here are some example images.



In a further Step the regions where the product changed should be detected.

Detecting differences

To detect differences a neural network with the same input and output resolution should be used. Those NNs could be FCNs (Fully convolutional neural networks) or U-Nets. These NNs are usually used for image segmentation and output one layer for each class. These layers are then used to determine the difference between the images:

Simple difference:

```
for(i in 0..layercount){
       sum += abs(img1[i]-img2[i]);
}
```

To avoid the detection of people as changes a Yolo v3 object detection is used to mask out all people.

First attempt using a pretrained VGG and using the output of one layer (the dark box is the yolo mask).

Here I was using U-Net for the change detection with a quite high resolution. This provided the best results, but the same input resulted in slightly different output because just the encoder was pretrained and not the whole model.

I used a custom seed to be able to achieve constant results.

Detecting product changes

The next step is to detect product changes in the shelfs. Our approach was to upscale the “change” areas and try to extract the dominant colors for every single change and its surroundings and then compare those between the two images to determine if the product changed. Comparing the colors is not as trivial as it sounds because of different lighting, brightness, position of the products etc.

Steps to detect product changes

- To ease up the task I defined sections for each shelf and defined a mask to remove all not necessary parts of the image. This could in the future be achieved by markers to be more independent of camera angles and positions.
- To reduce the calculation time of the U-net I reduced the resolution to (384, 384)
- All Changes detect under a specific threshold are removed
- After the detection the results are downscaled again (*0.2)
- Those changes are used to detect the color changes using KMeans to cluster the colors and get the dominant colors in the full resolution images. These colors (7,9) are compared and the biggest difference between to colors is used as a new metric.
- Again, a threshold is used to get only the big changes.
- The coverage of the color changes inside the sections is used to determine if the section changed or not.

Images:

Differences/Product Changed Areas:

Time needed (1080x1920, 20 regions):

- ~ 8 minutes per pair on RaspberryPi
- ~ 15s per image pair on (Intel(R) Core (TM) i7-6700 CPU) (without GPU acceleration)