**Report: A Neural Network For Recognition Of Trash**

**Dataset:**

The dataset we used to train out neural network is TACO. TACO is short for Trash Annotations in Context and is a dataset with more than 1500 images taken of litter in diverse environments and around 5000 annotations.

The first step in our project was the utilization of our chosen dataset. The developers of TACO enable the download of the dataset via two files in the git repository they have set up for the project. “deep\_vision\_project/data/annotations.json” holds the information about the images in the dataset, “deep\_vision\_project/download.py” is the script that downloads them from Flickr and saves them in multiple folders in “deep\_vision\_project /data”.

To embed the dataset, we implemented the class “Taco()” in “deep\_vision\_project/dataset.py”. “Taco()” inherits from “torch.utils.data.Dataset” and reads in the images and annotations of the downloaded folders and “deep\_vision\_project/data/annotations.json. Using this interface, we were now able to use “torch.utils.data.Dataloader” to split TACO into a test and training set with variable batch sizes and iterate over these data loaders in our training loop. In the end Taco() would take in an index and return an image with one corresponding target mask, annotating for each pixel what class it had.

To tackle the small size of our dataset we implemented a few simple forms of data augmentation. When enabled, our dataset class would sometimes randomly alter the images it returns by mirroring or flipping the image or changing its brightness.

One problem we ran into, was that the dataset was not consistent. The annotation file referenced images that were apparently no longer in the dataset. We replaced the broken references by intact, random ones.

The annotations of TACO had two different classes: categories (60) and super categories (28). To run an experiment on how different amounts of categories would impact our results we implemented our dataset class, so that it could return masks for normal categories and super categories.

**Network Architecture:**

As architecture for our neural network we have chosen U-Net. It met the criteria we had: a deep convolutional autoencoder with small filter and pooling sizes, small stride, Relu activation function, and skip connections to let the gradient pass. In “deep\_vision\_project/UNet.py” we implemented U-Net. The class “UNet()” inherits from “torch.nn.Module” and constructs the neural network from scratch. When put in an image, the network outputs a fixed number of masks and every mask corresponds with the predictions for the pixels of the image per class of trash.

For performing experiments on the network structure its construction is modular. The number of color channels and outputted masks can be altered, the depth of the network can be reduced and batch normalization can be applied between convolutions.

Since our implementation of U-Net does not apply functions with fixed sizes regarding the image size, the sizes of the images we put in, do no matter and could be variable. Still we decided to resize all images to 572 x 572 pixels in order to reduce computational complexity and improve computation time.

**Training:**

We trained our neural network for the different experiments up to 100 epochs. During this iteration we calculated the mean for each epoch for the following for metrics: Loss on the training set, loss on the test set, intersection over union for all annotations and intersection over union for the binary approach.

As loss function we used cross entropy loss. As optimizer we used ADAM.

As mentioned above we differentiated between multi class and single class segmentation. We modulated this by separating the masks our network would put out. The first mask pointed to the background and therefore already segmented the images binary. The other 28 or 60 masks each pointed to a unique class of trash and multiclass segmented the images.

In the other experiments we halved the depth of the network, applied batch normalization between convolutions, altered the learning rate of Adam and applied data augmentation.

**Analysis:**

For measuring how well our network performed we compared the above-mentioned metrics.

Sadly, almost all experiments showed the same results. After a while both the training and the validation loss would not really decline further. The Intersection over Union for the binary segmentation was always around 90% and the IoU for multiclass segmentation was around 30%. Latter was because an image had approximately 3 annotations, the binary IoU was also calculated in the multiclass IoU and 90% / 3 = 30%.

Although the metrics we analyzed were not to promising we were still able to visualize some medium accurate predictions.

**Conclusion:**

We did not reach the state of the art and we cannot even be sure whether we came close to it. The question we still cannot answer is where we made a mistake. Maybe the concept we applied was flawed and it was determined to output poor results, maybe we made a computational mistake and our network is not able to learn properly, maybe we did just fine, but failed at working out how good the network actually performed.

David Jäck, Cedric Bender