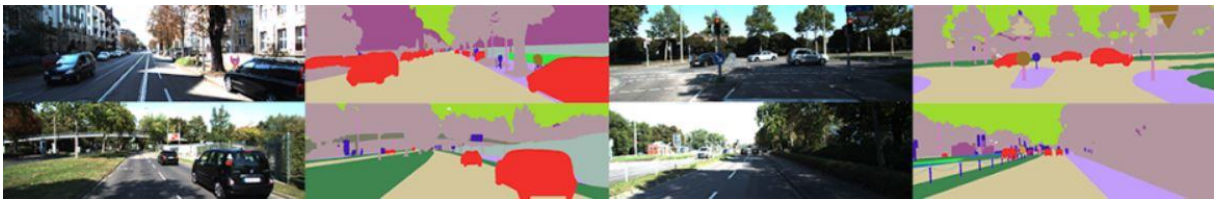
Title: Semantic Segmentation with AWS SageMaker



Domain Backround:

Machine Learning (ML) or in this case Deep Learning (DL) as special case of ML are very magical things, where you're not programming an algorithm to calculate a result, but you're programming an algorithm to take a result and to show you patterns within the result. In this case you teach the system to visualise objects and put the objects into categories or classes (for different cases of object detection on Amazon Web Services (AWS) SageMaker as ML-tool please see the following post: https://medium.com/@kolungade.s/object-detection-image-classification-and-semantic-segmentation-using-aws-sagemaker-e1f768c8f57d). As I am working within the Automotive Industry for several years, I am really interested in putting the ideas of ML and DL into the Automotive Industry. Self-Driving cars on the streets, Self-Working robots within a manufacturing process, or Self Organized Logistics within a production plant are not future thinking anymore, but possible solutions for the Automotive Industry nowadays. Semantic segmentation is the basis for all of these tasks. Before you learn how to fly, you'll need to learn, how to walk. Actually, I start walking now and am already exited, which next steps I'll make.

Problem Statement:

According to tensorflow.org "the task of image segmentation is to train a neural network to output a pixel-wise mask of the image. This helps in understanding the image at a much lower level" (https://www.tensorflow.org/tutorials/images/segmentation).

Basically, semantic segmentation is needed to get an output image, where every pixel of the input image is assigned to a class, e.g. a car, bus, or a pedestrian. It's used to identify objects on a photo, video, etc. Semantic segmentation could let cars know the location of another car or person on the road for Autonomous Driving purposes or let robots know the location of other parts for Augmented Reality Applications in order to avoid collisions in a manufacturing environment.

In order to identify objects in a traffic environment I am using the Kitti Dataset for Lane/Road Detection Evaluation (http://www.cvlibs.net/datasets/kitti/eval_road.php). This allows me to perform a binary segmentation of road-/not-road-objects

This data then will be used to train a Neural Network on photos of streets including objects like cars or pedestrians with the help of Amazon Web Services (AWS) SageMaker. The Neural Network then will identify, if the objects on new photos of a traffic environment belongs to a road or not (it divides the picture into different segments).

Datasets and inputs:

I order to identify objects in a traffic environment I am using the Kitti Dataset downloaded from Kaggle (https://www.kaggle.com/kerneler/starter-kitti-object-detection-162ff5be-6).

The Kitti Dataset was collected by driving around the mid-size city of Karlsruhe (Germany), equipped with two high-resolution cameras, and taking photos in rural areas as well as on highways. “Up to 15 cars and 30 pedestrians are visible per image” (www.cvlibs.net/datasets/kitti/). Basically, it consists out of 7518 photos in the test dataset and 7481 photos in the training dataset. The training dataset is labelled indicating objects like cars or pedestrians per pixel. Hence, it’s a case of supervised learning.

However, for the purpose of this task (binary segmentation), I am using a dataset from the Kitti dataset prepared for this task.

```
# show data statistics

import os
from sklearn.datasets import load_files
from keras.utils import np_utils
import numpy as np
from glob import glob

# load train, test datasets
data_dir = './data'
kitti_dataset_path = os.path.join(data_dir, 'data_road')
train_files = glob(os.path.join(kitti_dataset_path, 'training/image_2/*.png'))
label_files = glob(os.path.join(kitti_dataset_path, 'training/gt_image_2/*_road_*.png'))
test_files = glob(os.path.join(kitti_dataset_path, 'testing/image_2/*.png'))


# print statistics about the dataset

print('There are %d training images.' % len(train_files))

print('There are %d test images.'% len(test_files))
print('There are %d total images.'%(len(train_files)+ len(test_files)))

print()
print('There are %d image labels.'%len(label_files))
```

```
There are 289 training images.
There are 290 test images.
There are 579 total images.

There are 289 image labels.

Using TensorFlow backend.
```

Data Statistics

The following steps will be taken for solving the segmentation task:

Solution statement:

The Kitti Dataset was collected by driving around the mid-size city of Karlsruhe (Germany), equipped with two high-resolution cameras, and taking photos in rural areas as well as on highways. “Up to 15 cars and 30 pedestrians are visible per image” (www.cvlibs.net/datasets/kitti/). It consists out of 7518 photos in the test dataset and 7481 photos in the training dataset. The training dataset is labelled indicating objects like cars or pedestrians per pixel. Hence, it’s a case of supervised learning.

“To identify the contents of an image at the pixel level, use an Amazon SageMaker Ground Truth semantic segmentation labelling task. When assigned a semantic segmentation labelling job, workers classify pixels in the image into a set of predefined labels or classes” (https://docs.aws.amazon.com/sagemaker/latest/dg/sms-semantic-segmentation.html).

In order to solve the given problem, I will create a Neural Network (NN) that performs the binary semantic segmentation.

Benchmark model:

The NN will be a Fully Convolutional Neural Network (FCN) using a technique called Transfer Learning with the VGG16-pretrained-model (more precisely we use encoding-decoding as techniques, see also https://courses.cs.washington.edu/courses/cse576/17sp/notes/Sachin_Talk.pdf).
The FCN is working good for Classification purposes, but let's see how they perform for Segmentation.

Evaluation metrics:

- Validation Accuracy vs. Epochs
- Loss vs. Epochs
  - It will be measured, how the model will be performing for a rising number of epochs

Workflow:

Data Input > Layer 1 (with the weights goings into the layer) > Layer 2 (with the weights goings into the layer) > prediction and true labels > loss function > optimize layer 2 > prediction and true labels