Semantic-Segmentation

Report based on the specifications here.

The goal of this project is to label the pixels of a road in images using a Fully Convolutional Network (FCN).

Files Submitted & Code Quality

The project includes the following files:

- *main.py* containing the main script to train the FCN and calling everything else.
- *helper.py* with original helper functions and new ones for image augmentation.
- *project_tests.py* containing the original function tests (not modified).
- *driving.mp4*: sample video from the Object Detection Lab.
- *output.mp4*: semantic segmentation video result.
- *runs*: sample foder with images result.
- *Project12_LucianoSilveira.pdf*: summarizing the results (this file).

Make sure to download the vgg, dataset and place it on the data folder. The expected tree structure is as follows:

```
data
├───────data_road
│      ├───────testing
│      │      ├───────calib
│      │      └───────image_2
│      └───────training
│            ├───────calib
│            ├───────gt_image_2
│            └───────image_2
└───────vgg
    └───────variables
```

Execute it as follows:

python main.py

Model Architecture and Training Strategy

Model

The suggested VGG16 model is used, taking into account this post.

The VGG model is loaded following the QA session tips.

The model was run on a local machine using a NVidia GTX 960M card. The following modifications were done to the model. Following this post, memory configuration was changed to:

```
config = tf.ConfigProto()
config.gpu_options.allow_growth = True
```

Layers

All the weights are trainable; the following modifications were added to the layers function to solve a ResourceExhaustedError error:

```
vgg_layer7_out = tf.stop_gradient(vgg_layer7_out)
vgg_layer4_out = tf.stop_gradient(vgg_layer4_out)
vgg_layer3_out = tf.stop_gradient(vgg_layer3_out)
```

The 1x1 convolutions used the following parameters, commented are other values tested:

```
kernel_initializer = 1e-2 # 2e-2
kernel_regularizer = 1e-3 # 1e-2, 1e-3, 1e-4, 1e-5
```

For the transpose section the following parameters are used:

```
kernel_size = 4
strides = (2, 2)
padding='same'
```

The final upsampling layer uses a 4x(2,2) value.

Optimize

To build the loss and optimizer operations, it starts with a 2D reshape so the softmax function can be used.

The classification and loss section uses the following parameters with an Adam optimizer:

```
l_rate = 1e-3 # 1e-2, 1e-3, 1e-4
```

The learning rate was not tuned manually; the selected value is 0.001, other values tested are detailed above.

Train

The neural network training uses the following general parameters, other values tested are commeted:

```
epochs = 20 # 1, 2, 10, 20
batch_size = 5 # 1, 2, 4, 5
k_prob = 0.95 # 0.8, 0.9, 1.0
```

The training data consists of the sample data provided plus an augmentation section using:

- Flip the sample and label data.
- Brightness changes.

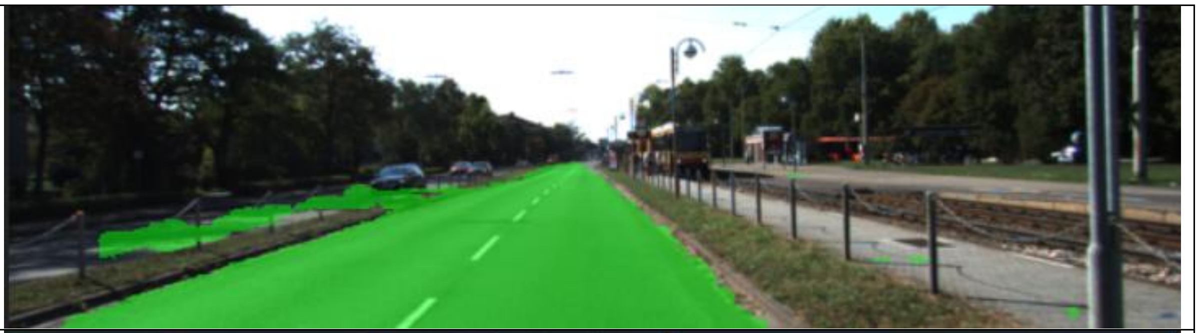
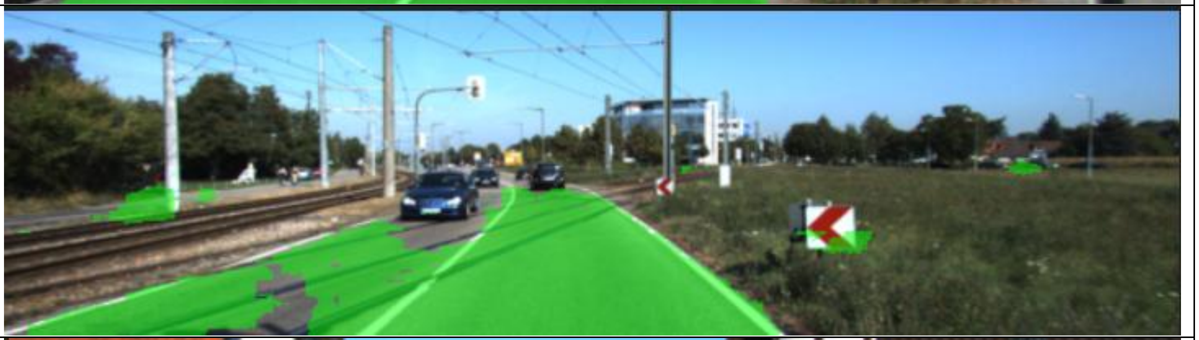
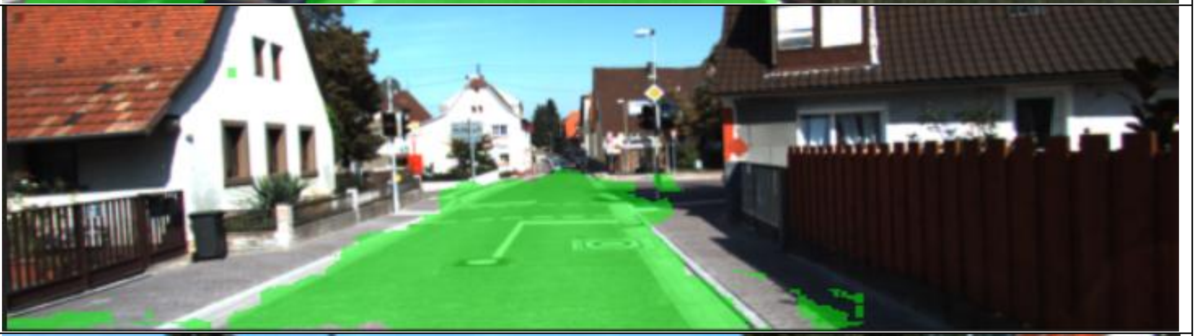
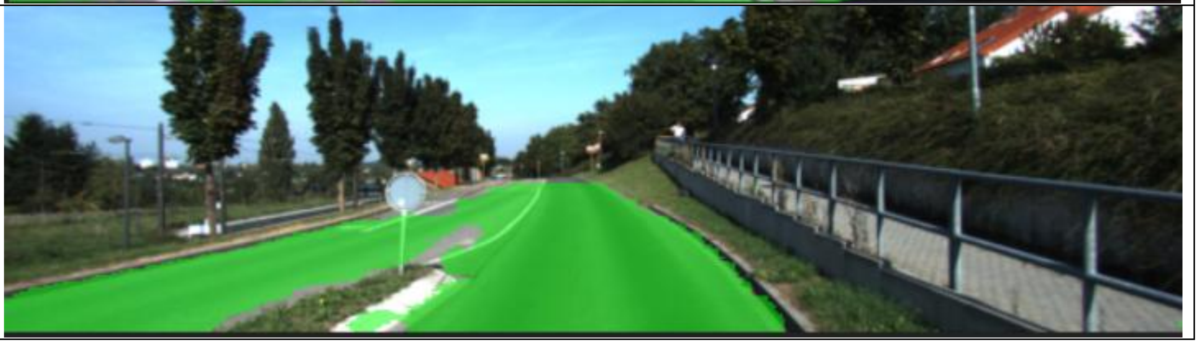
The detail can be checked on the helper.py file, *get_batches_fn* method.

Every 10 iterations the loss is printed to the output. An example of the training process is as follows:

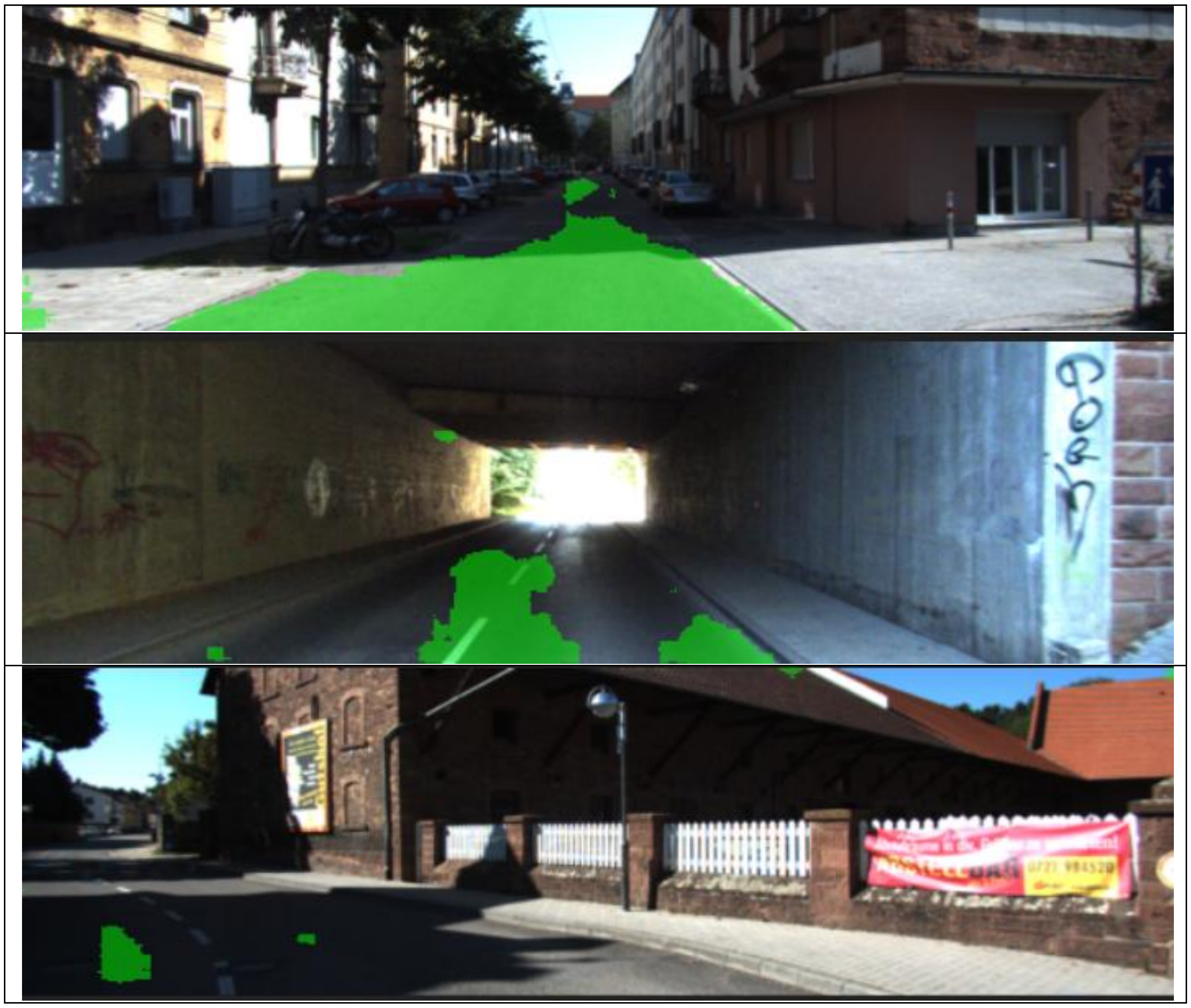
```
Epoch 19 Batch 10 Loss 0.071
Epoch 19 Batch 20 Loss 0.093
Epoch 19 Batch 30 Loss 0.124
Epoch 19 Batch 40 Loss 0.081
Epoch 19 Batch 50 Loss 0.065
Epoch 20 Batch 10 Loss 0.094
Epoch 20 Batch 20 Loss 0.121
Epoch 20 Batch 30 Loss 0.093
Epoch 20 Batch 40 Loss 0.079
Epoch 20 Batch 50 Loss 0.100
Training Finished. Saving test images to: ./runs\1528640940.8673346
Generating video to: ./output.mp4
[MoviePy] >>>> Building video ./output.mp4
[MoviePy] Writing video ./output.mp4
```

Samples

Some good samples from the run folder:

Some bad samples:



Video

For the video generation it was selected the sample from the Object Detection Lab.

The pipeline to process the video is taken from the Term1 section, important functions are:

- *complete_pipeline*: process a frame at a time; it reuses the code from *gen_test_output* helper method to do the semantic segmentation.
- *generate_video*: process the input video and delegates to the previous function for processing.

Parameters:

```
input_file = './driving.mp4'
output_file = './output.mp4'
```

Some samples from the process:

Considerations

To get started I followed the information from the QA.

Other links reviewed are:

- #s-t3-p-semantic-segme Slack channel
- Forum
- Rubric

The whole pipeline takes at least four hours. The training section (up to samples generation) using the previous parameters takes 50 minutes and the rest is consumed by the video generation.

It was run on a Windows 10 box i7 core @2.50GHz with 32GB RAM and a NVidia GTX 960M card with 2.4 teraFlops.