Cognitive Computer Vision: Exercise 1

Tensorflow (TF) is an open-source framework for machine learning. In the exercises, we work with the low-level TF API, so called TF Core, because

- experimentation and debugging are both more straight forward when you can use low level TF operations directly, and
- it gives you a mental model of how things work internally when eventually using the higher level APIs.

Useful links:

- TF Core Walkthrough `https://www.tensorflow.org/programmers_guide/low_level_intro`
- Python API documentation `https://www.tensorflow.org/api_docs/python/`
- If you want to work on your own computer: installation instructions `https://www.tensorflow.org/install/`

1 Getting started

Try out the following example by creating a new file called `hello.py` and execute it by running `python hello.py`.

```
1  import tensorflow as tf
2  x = tf.constant('Hello world!')
3  y1 = tf.constant(2, dtype=tf.int32)
4  y2 = tf.constant(1, dtype=tf.int32)
5  z = y1 + y2
6
7  sess = tf.Session()
8  x_eval = sess.run(x)
9  print(x_eval)
10 z_eval = sess.run(z)
11 print(z_eval)
12 y1_eval, y2_eval = sess.run([y1,y2])
13 print(y1_eval, y2_eval)
```

Listing 1: First example

- Almost everything in TF is an operation (op). Ops can be recognised by lowercase names (e.g., `tf.constant`).
- The code first defines a computational graph (Lines 2-5) by declaring the ops to add to it. No computation happens here.
- A session manages the graph and allows to run ops (and thereby evaluate tensors) in the graph. Computation only happens when we run the session (Lines 8, 10, and 12).

The computational graph is shown in Figure 1. The graph shows operations as nodes, and tensors as edges.

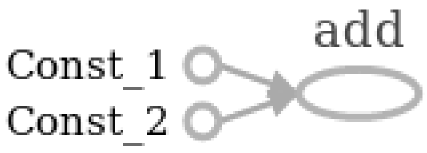

Figure 1: Computational graph for Listing 1.

2 Visualizing the computational graph

Consider the function $f : \mathbb{R}^2 \to \mathbb{R}^2$ defined as

$$f(x) = Ax + b, \tag{1}$$

where $x, b \in \mathbb{R}^2$, and $A \in \mathbb{R}^{2\times 2}$ (a 2-by-2 matrix). Further, let $y \in \mathbb{R}$ and $g : \mathbb{R} \to [-1, 1]$ be

$$g(y) = \sin y \tag{2}$$

- Implement $f(x)$ in Tensorflow for constant$A$, x, and b. Hint: `tf.matmul`
- Find g elementwise: if $f(x) = \begin{bmatrix} y_1 & y_2 \end{bmatrix}^T$, calculate $\begin{bmatrix} g(y_1) & g(y_2) \end{bmatrix}^T$. Hint: `tf.sin`
- Visualize the computational graph you get by adapting the following code:

```
sess = tf.Session()
summary_writer = tf.summary.FileWriter(logdir='./', graph=sess.
   graph)
sess.run(op)
```

Listing 2: Writing the graph

`FileWriter` handles writing the computational graph, and possible other log elements, to disk. We will learn more about it later, but for now we just use it to write the graph. After running the code, check the folder where you ran your code. You will see a file called something like `events.out.tfevents.1236543698.computername`. **Important**: a new event file will be created for each run of your code. Make sure to delete the old event files, or place them in different directories.

- Start up TensorBoard to visualize the graph: in the same directory where your event file is by running in the terminal: `tensorboard --logdir=`. Open the URL indicated in the output in a web browser and explore the GUI.
- Give names to the operations in your graph by using their `name` input argument. Observe how the graph in TensorBoard changes.

3 Feeding external data

We learned before that `session.run()` can be used to fetch data from the Tensorflow backend to Python. The opposite process of feeding external data from Python into a computational graph by using the `feed_dict` argument of `session.run()` together with a `tf.placeholder`: `https://www.tensorflow.org/api_docs/python/tf/placeholder`.

- Define x as a placeholder with the correct shape, and feed in an arbitrary input. Check how the computational graph changes via TensorBoard.

4 2D Convolutions

The code below reads an image, converts it to a floating point array and normalizes it to the range between 0 and 1. Finally, it visualizes the image. Additionally, there is a function that defines a Gaussian smoothing kernel with a given standard deviation[1].

```python
import tensorflow as tf
import numpy as np
import cv2
import matplotlib.pyplot as plt
from PIL import Image

def get_gaussian_kernel(sigma, size=None):
  if not size:
    size = int(round(sigma*3*2+1))|1
  C = cv2.getGaussianKernel(size, sigma)
  C = np.outer(C,C).astype(np.float32)
  C /= np.sum(C)
  return C

img = Image.open('cameraman.png')
img_arr = np.array(img).astype(np.float32)/255.0

imgplot = plt.imshow(img_arr, cmap='gray')
plt.show()
```

Listing 3: 2D convolution starting code

- Use `tf.nn.conv2d` to convolve `cameraman.png` from Moodle with a Gaussian smoothing filter. Visualize the input and output images to see the difference.

Hints:

- Ensure correct shapes of the arrays. Manipulate the shape if required by `tf.expand_dims`, `tf.reshape`, and `tf.squeeze`.
- Use the `shape` property of tensors and TensorBoard visualization to get information about the shapes.

[1]The size of the kernel can be given by the user. If not, the size is inferred from the given standard deviation.