Cognitive Computer Vision

Computer Exercise

Tuesdays 14.15-15.45, Room D-118/119

Dr. Mikko Lauri
Email: lauri@informatik.uni-hamburg.de

SS 2018

Contents

1 Organization of exercises

Organization

- 6 computer exercises
- Course project: Build a visual saliency system (mandatory to pass course)

Figure: Left: Input image Right: Saliency map

Organization

- Exercises support and give you tools and methods to complete the course project.
- The exact instructions for the course project will be published soon. Deadline: June 12th
    1. You will work with a dataset and train a machine learning based visual saliency system.
    2. You will evaluate your results on a test dataset.
    3. You will present your system architecture and results to others (June 19th)
- Best saliency system will be awarded :)

Exercise topics

- Planned outline:
  1. Tensorflow introduction, 2D convolutions
  2. Variables and fully connected layers, training of neural networks
  3. Setting up a basic saliency system
  4. Set-up continued, evaluating a saliency system
  5. Evaluation continued
  6. Performance tuning

Tensorflow: Brief tutorial

- A tensor is a generalization of a matrix. Rank = number of dimensions

```
# rank 0 tensor; a scalar with shape []
3.
# rank 1 tensor; a vector with shape [3]
[1., 2., 3.]
# rank 2 tensor; a matrix with shape [2, 3]
[[1., 2., 3.], [4., 5., 6.]]
# rank 3 tensor with shape [2, 1, 3]
[[[1., 2., 3.]], [[7., 8., 9.]]]
```

Listing 1: Tensors

- Tensorflow is about setting up a graph of operations that take as input tensors, and output tensors.

Tensorflow graph

- A computational graph in TF consists of nodes and edges.
  - Nodes: operations
  - Edges: tensors (inputs and outputs of operations)
- Two phases:
  1. Set up a computational graph that defines what should be computed.
  2. Provide the required input values and run operations in the graph.
- When using TF, make sure you adhere to the phases: First set up everything in your graph. Only then run the required operations!

Tensorflow sessions

- The Python API of TF is merely used to construct the graph and specify which operations to run.
- Operations in the computational graph will actually run using the C++ backend of TF.
- A `tf.Session` allows communication back and forth with the backend, and running any operations.
- If a `tf.Graph` is like a `.py` file, a `tf.Session` is like the python executable.
- Now let's start the exercise to see these concepts in practice!

1. Getting started

- `tf.constant`: op that produces a constant value with given datatype as output.
- `+`: syntactic sugar for `tf.add` – op that takes two inputs and outputs elementwise sum of inputs. See also `*` and `/`
- `out = Session.run(fetches)` runs the op `fetches` (can also be a list or dictionary of ops) and returns the output. Communication from TF backend to Python frontend.

Remember: Set up first, then run

First set up all ops in our computational graph, **only after that** start running ops, and don't add any more!

- Calling any `tf.operation` will add an operation to the computation graph. Operations can **not be removed** from the graph. Be very sure what you are doing if you don't follow this rule.

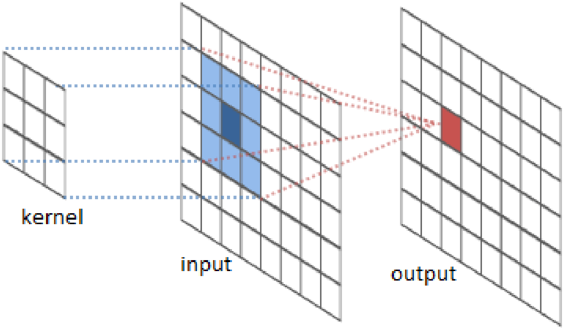
2. Visualizing the computation graph

- By default, unary functions will operate on tensors elementwise. Examples of unary functions: `tf.sin, tf.cos, tf.square, tf.log, ...`
- `tf.matmul` is not `tf.multiply`: matrix vs. elementwise multiplication
- Good practice: give names to your ops

3. Feeding external data

- Recall: `out = Session.run(fetches)` runs the op `fetches` (can also be a list or dictionary of ops) and returns the output. Communication from TF backend to Python frontend.
- `out = Session.run(fetches, feed_dict)` allows user to give a dictionary of elements of type `placeholder: value` to feed to graph. Communication from Python frontend to TF backend.
- Example: `x=tf.placeholder(...),`
  `feed_dict={x: [1.0, 2.0]}`

4. 2D Convolutions



[1]

- Sliding window (kernel) over image, calculate inner product with contents of input.

[1] Image credit: http://jeanvitor.com/convolution-parallel-algorithm-python/

4. 2D Convolutions

```
tf.nn.conv2d(
    input,
    filter,
    strides,
    padding,
    use_cudnn_on_gpu=True,
    data_format='NHWC',
    dilations=[1, 1, 1, 1],
    name=None)
```

- `input` – the input images: tensor of size $\begin{bmatrix} N & H & W & C \end{bmatrix}$: N images, height H, width W, C channels.
- Example: one grayscale image, size of `input` is $\begin{bmatrix} 1 & H & W & 1 \end{bmatrix}$

4. 2D Convolutions

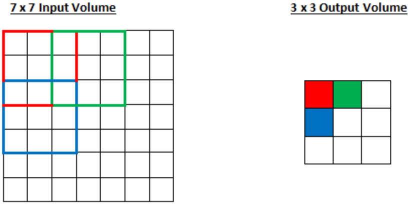
```
tf.nn.conv2d(
    input,
    filter,
    ...)
```

- `filter` – the filtering kernels, tensor of size $\begin{bmatrix} F_h & F_w & N_i & N_o \end{bmatrix}$: filter height F_h, width F_w, N_i input channels, N_o output channels.
- F_h and F_w typically odd, the middle element is the origin of the kernel.
- Example: one filter operating on single-channel input: $\begin{bmatrix} F_h & F_w & 1 & 1 \end{bmatrix}$
- `filter[i,j,k,l]` is the weight of relative location (i, j) for the kth input channel to produce the lth output channel.

4. 2D Convolutions

```
tf.nn.conv2d(...,
    strides, ...)
```

- `stride`: number of units the filter shifts in each dimension, 1-D tensor with 4 elements: S_b, S_h, S_w, S_c



[2]

Figure: Stride 2 for height and width

[2] Image credit: https://adeshpande3.github.io

4. 2D Convolutions

```
tf.nn.conv2d(...,
    padding, ...)
```

- `padding` – How to pad image around the edges?
- `"SAME"`: Pad the input with zeros around the edges to ensure output is of same size as input.
- `"VALID"`: no padding. Output size will be smaller than input depending on the kernel size.

4. 2D Convolutions

- Make sure your input and filter kernel are 4D tensors with appropriate size.
- `input` – the input images: tensor of size $\begin{bmatrix} N & H & W & C \end{bmatrix}$: N images, height H, width W, C channels.
- `filter` – the filtering kernels, tensor of size $\begin{bmatrix} F_h & F_w & N_i & N_o \end{bmatrix}$: filter height F_h, width F_w, N_i input channels, N_o output channels.
- `tf.reshape`, `tf.expand_dims`
- Output will also be 4D tensor