MasterLab Deep Learning

Convolutional Nets in TensorFlow

Davide Abati

March 23th, 2018

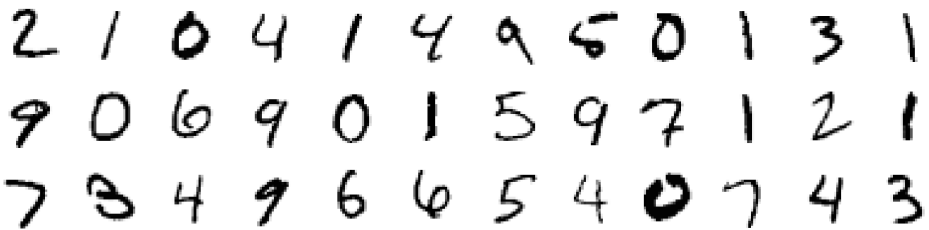
University of Modena and Reggio Emilia

Agenda

MNIST

MNIST Dataset

MNIST[1] is a database of handwritten digits consisting of 60K training images and 10K testing images. All digits have been centered in 28x28 grayscale images.



MNIST Dataset

Due to its simplicity (in 2017!) the MNIST dataset is often considered to be the "Hello World!" in the Machine Learning framework.

In `lab_utils.py` I already implemented a function `get_mnist_data` that handles the download and loading of the dataset.

```
# Load MNIST data
mnist = get_mnist_data('/tmp/mnist', verbose=True)
```

MNIST Classification

Goal

The **goal** of this practice is to **implement a convolutional neural network to perform 10-class classification** on the MNIST dataset.

Suggested way to proceed:

- In the previous practice we implemented a fully-connected net to solve the very same task. Start from your previous working example and change the model to a convolutional net.

Useful Functions

To this purpose, you may find useful the following functions:

- `tf.reshape`
- `tf.layers.conv2d`
- `tf.layers.max_pooling2d`
- `tf.nn.dropout`
- `tf.layers.dense`

Please refer to the docs to know the exact API.

Tiles Segmentation

Tiles Dataset

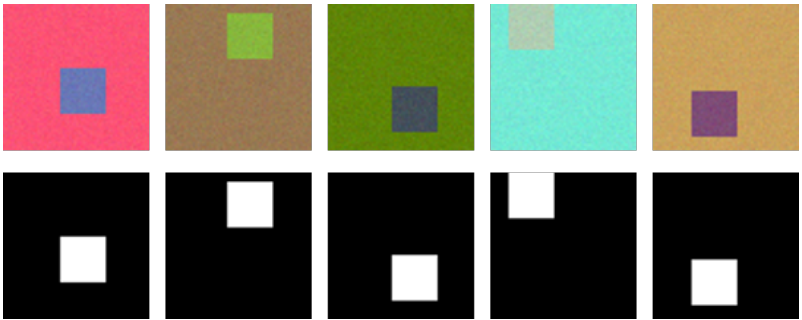
In order to get your hands dirty with a fully convolutional architecture, I prepared a toy dataset that you can use to tackle the task of segmentation.

The **Tiles Dataset** consists of 64x64 images in which a 20x20 square has been drawn in a random location. Both the background and the foreground color have been chosen randomly for each image. Eventually, gaussian noise has been added to both.

The dataset is composed of 10000 training examples, 1000 validation examples and 1000 test examples.

The goal is to segment the square in each image. This is a **toy example of binary segmentation** (e.g. *foreground* vs *background*).

For this task, the output of the network must be itself an image.



Useful Functions

Besides all functions previously introduced, these functions might also be useful:

- `tf.image.resize_bilinear`
- `tf.transpose`
- `tf.gather`
- `tf.expand_dims`
- `tf.nn.softmax_cross_entropy_with_logits`

Please refer to the docs to know the exact API.

Loading the Data

Minimal working example for loading the data:

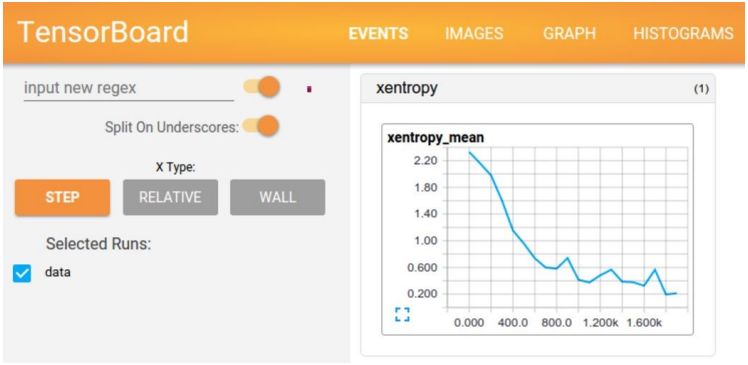
```
from lab_utils import load_tiles_dataset_from_cache
from lab_utils import TilesDataset

if __name__ == '__main__':
    cache_path = 'data/tiles_dataset.pickle'
    tiles_data = load_tiles_dataset_from_cache(cache_path)
```

Outline

- Load the tiles data and take your time to explore the dataset structure
- Keep in mind that to use categorical *cross-entropy* you have to convert the targets into one-hot encoding. If you instead feel quick-and-dirty, you might try to use *MSE* loss and see what happens.
- Visualizing the output of the model in TensorBoard is a good way to get a feeling of what the model is doing. To this purpose `tf.summary.image` could be useful.

TensorBoard is a suite of visualization tools integrated with TensorFlow. You can use TensorBoard to visualize your TensorFlow graph, plot quantitative metrics about the execution of your graph, and show additional data like images that pass through it.

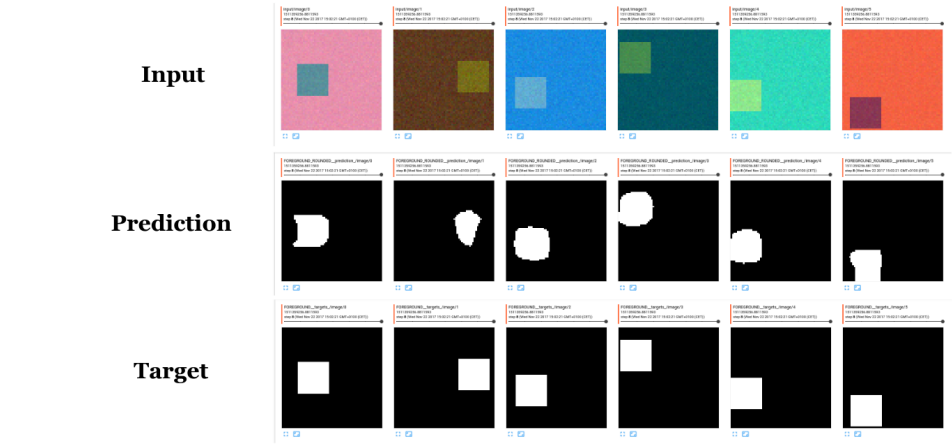


From a very high-level perspective, the lifecycle is the following:

- **tf.summary** operations can be attached to the nodes you'd like to collect summary data from.
- **tf.summary.FileWriter** object is in charge of writing generated logs to a certain directory `logdir`
- **TensorBoard** can be launched from command line as `tensorboard --logdir=<logdir>`. TensorBoard is now available through your web browser at `localhost:6006`.

Official tutorial here:
https://www.tensorflow.org/get_started/summaries_and_tensorboard

This is an example of what you may see while training:



Good Luck!

References

[1] Y. LeCun.
The mnist database of handwritten digits.
http://yann. lecun. com/exdb/mnist/, 1998.