Attention-based Extraction of Structured Information from Street View Imagery

A TensorFlow model for real-world image text extraction problems.

This folder contains the code needed to train a new Attention OCR model on the FSNS dataset to transcribe street names in France. You can also train the code on your own data.

More details can be found in our paper:

"Attention-based Extraction of Structured Information from Street View Imagery"

Description

- Paper presents a model based on ConvNets, RNN's and a novel attention mechanism. Achieves **84.2%** on FSNS beating the previous benchmark (**72.46%**). Also studies the speed/accuracy tradeoff that results from using CNN feature extractors of different depths.

Contacts

Authors

- Zbigniew Wojna (zbigniewwojna@gmail.com)
- Alexander Gorban (gorban@google.com)

Maintainer

- Xavier Gibert (@xavigibert)

Table of Contents

- Requirements
- Dataset
- How to use this code
- Using your own image data
- How to use a pre-trained model
- Disclaimer

Requirements

1. Install the TensorFlow library (instructions). For example:

```
python3 -m venv ~/.tensorflow
source ~/.tensorflow/bin/activate
pip install --upgrade pip
pip install --upgrade tensorflow-gpu=1.15
```

2. At least 158GB of free disk space to download the FSNS dataset:

```
cd research/attention_ocr/python/datasets
aria2c -c -j 20 -i ../../../street/python/fsns_urls.txt
cd ..
```

3. 16GB of RAM or more; 32GB is recommended.
4. `train.py` works with both CPU and GPU, though using GPU is preferable. It has been tested with a Titan X and with a GTX980.

Dataset

The French Street Name Signs (FSNS) dataset is split into subsets, each of which is composed of multiple files. Note that these datasets are very large. The approximate sizes are:

- Train: 512 files of 300MB each.
- Validation: 64 files of 40MB each.
- Test: 64 files of 50MB each.
- The datasets download includes a directory `testdata` that contains some small datasets that are big enough to test that models can actually learn something.
- Total: around 158GB

The download paths are in the following list:

```
https://download.tensorflow.org/data/fsns-20160927/charset_size=134.txt
https://download.tensorflow.org/data/fsns-20160927/test/test-00000-of-00064
...
https://download.tensorflow.org/data/fsns-20160927/test/test-00063-of-00064
https://download.tensorflow.org/data/fsns-20160927/testdata/arial-32-00000-of-00001
https://download.tensorflow.org/data/fsns-20160927/testdata/fsns-00000-of-00001
https://download.tensorflow.org/data/fsns-20160927/testdata/mnist-sample-00000-of-
00001
https://download.tensorflow.org/data/fsns-20160927/testdata/numbers-16-00000-of-00001
https://download.tensorflow.org/data/fsns-20160927/train/train-00000-of-00512
...
https://download.tensorflow.org/data/fsns-20160927/train/train-00511-of-00512
https://download.tensorflow.org/data/fsns-20160927/validation/validation-00000-of-
00064
...
https://download.tensorflow.org/data/fsns-20160927/validation/validation-00063-of-
00064
```

All URLs are stored in the research/street repository in the text file `python/fsns_urls.txt`.

How to use this code

To run all unit tests:

```
cd research/attention_ocr/python
find . -name "*_test.py" -printf '%P\n' | xargs python3 -m unittest
```

To train from scratch:

```
python train.py
```

To train a model using pre-trained Inception weights as initialization:

```
wget http://download.tensorflow.org/models/inception_v3_2016_08_28.tar.gz
tar xf inception_v3_2016_08_28.tar.gz
python train.py --checkpoint_inception=./inception_v3.ckpt
```

To fine tune the Attention OCR model using a checkpoint:

```
wget http://download.tensorflow.org/models/attention_ocr_2017_08_09.tar.gz
tar xf attention_ocr_2017_08_09.tar.gz
python train.py --checkpoint=model.ckpt-399731
```

Using your own image data

You need to define a new dataset. There are two options:

1. Store data in the same format as the FSNS dataset and just reuse the python/datasets/fsns.py module. E.g., create a file datasets/newtextdataset.py:

```
import fsns

DEFAULT_DATASET_DIR = 'path/to/the/dataset'

DEFAULT_CONFIG = {
    'name':
        'MYDATASET',
    'splits': {
        'train': {
            'size': 123,
            'pattern': 'tfexample_train*'
        },
        'test': {
            'size': 123,
            'pattern': 'tfexample_test*'
        }
    },
    'charset_filename':
        'charset_size.txt',
    'image_shape': (150, 600, 3),
    'num_of_views':
        4,
    'max_sequence_length':
        37,
    'null_code':
        42,
    'items_to_descriptions': {
        'image':
            'A [150 x 600 x 3] color image.',
        'label':
```

```
            'Characters codes.',
        'text':
            'A unicode string.',
        'length':
            'A length of the encoded text.',
        'num_of_views':
            'A number of different views stored within the image.'
    }
}


def get_split(split_name, dataset_dir=None, config=None):
  if not dataset_dir:
    dataset_dir = DEFAULT_DATASET_DIR
  if not config:
    config = DEFAULT_CONFIG

  return fsns.get_split(split_name, dataset_dir, config)
```

You will also need to include it into the `datasets/__init__.py` and specify the dataset name in the command line.

```
python train.py --dataset_name=newtextdataset
```

Please note that eval.py will also require the same flag.

To learn how to store a data in the FSNS format please refer to the https://stackoverflow.com/a/44461910/743658.

2. Define a new dataset format. The model needs the following data to train:

- images: input images, shape [batch_size x H x W x 3];
- labels: ground truth label ids, shape=[batch_size x seq_length];
- labels_one_hot: labels in one-hot encoding, shape [batch_size x seq_length x num_char_classes];

Refer to python/data_provider.py for more details. You can use python/datasets/fsns.py as the example.

How to use a pre-trained model

The inference part was not released yet, but it is pretty straightforward to implement one in Python or C++.

The recommended way is to use the Serving infrastructure.

To export to SavedModel format:

```
python model_export.py \
  --checkpoint=model.ckpt-399731 \
  --export_dir=/tmp/attention_ocr_export
```

Alternatively you can:

1. define a placeholder for images (or use directly an numpy array)
2. create a graph

```
endpoints = model.create_base(images_placeholder, labels_one_hot=None)
```

3. load a pretrained model
4. run computations through the graph:

```
predictions = sess.run(endpoints.predicted_chars,
                       feed_dict={images_placeholder:images_actual_data})
```

5. Convert character IDs (predictions) to UTF8 using the provided charset file.

Please note that tensor names may change overtime and old stored checkpoints can become unloadable. In many cases such backward incompatible changes can be fixed with a string substitution to update the checkpoint itself or using a custom var_list with assign_from_checkpoint_fn. For anything other than a one time experiment please use the TensorFlow Serving.

Disclaimer

This code is a modified version of the internal model we used for our paper. Currently it reaches 83.79% full sequence accuracy after 400k steps of training. The main difference between this version and the version used in the paper - for the paper we used a distributed training with 50 GPU (K80) workers (asynchronous updates), the provided checkpoint was created using this code after ~6 days of training on a single GPU (Titan X) (it reached 81% after 24 hours of training), the coordinate encoding is disabled by default.