Fabrik Documentation

Release 0.2

CloudCV Team

Dec 12, 2018

Contents

Fabrik is an online collaborative platform to build, visualize and train deep learning models via a simple drag-and-drop interface. It allows researchers to collaboratively develop and debug models using a web GUI that supports importing, editing and exporting networks written in widely popular frameworks like Caffe, Keras, and TensorFlow.

Contents:

CHAPTER 1

How to setup

1. First set up a virtualenv

```
sudo apt-get install python-pip python-dev python-virtualenv
virtualenv --system-site-packages ~/Fabrik
source ~/Fabrik/bin/activate
```

2. Clone the repository

```
git clone --recursive https://github.com/Cloud-CV/Fabrik.git
```

3. If you have Caffe, Keras and Tensorflow already installed on your computer, skip this step
   - For Linux users

     ```
     cd Fabrik/requirements
     yes Y | sh caffe_tensorflow_keras_install.sh
     ```

     Open your ~/.bashrc file and append this line at the end

     ```
     export PYTHONPATH=~/caffe/caffe/python:$PYTHONPATH
     ```

     Save, exit and then run

     ```
     source ~/.bash_profile
     cd ..
     ```

   - For Mac users
     - Install Caffe
     - Install Tensorflow
     - Install Keras
4. Install dependencies

- For developers:

```
pip install -r requirements/dev.txt
```

- Others:

```
pip install -r requirements/common.txt
```

1. Install postgres >= 9.5

- Setup postgres database
  - Start postgresql by typing `sudo service postgresql start`
  - Now login as user postgres by running `sudo -u postgres psql` and type the commands below

```
CREATE DATABASE fabrik;
CREATE USER admin WITH PASSWORD 'fabrik';
ALTER ROLE admin SET client_encoding TO 'utf8';
ALTER ROLE admin SET default_transaction_isolation TO 'read committed';
ALTER ROLE admin SET timezone TO 'UTC';
ALTER USER admin CREATEDB;
```

  - Exit psql by typing in \q and hitting enter.

- Migrate

```
python manage.py makemigrations caffe_app
python manage.py migrate
```

1. Install node modules

```
npm install
webpack --progress --watch --colors
```

1.1 Usage

```
KERAS_BACKEND=theano python manage.py runserver
```

1.2 Example

- Use `example/tensorflow/GoogleNet.pbtxt` for tensorflow import
- Use `example/caffe/GoogleNet.prototxt` for caffe import
- Use `example/keras/vgg16.json` for keras import

CHAPTER 2

Basics of Using the Exported Keras Model

We want to export our model for Keras from Fabrik.

1. First, select the 2nd button from the left in the Actions section of the sidebar.
2. A drop-down list should appear. Select Keras.
   - This should download a JSON file to your computer.
3. Rename the file to `model.json`.
4. Load the model from the JSON file using the following code:

```
from keras.models import model_from_json

# Read and load the JSON file
json_file = open('<path_to_file>/model.json', 'r')
loaded_model_json = json_file.read()
json_file.close()

# Use Keras's built in model_from_json function to convert the JSON file to a␣
↪model
loaded_model = model_from_json(loaded_model_json)

# Print a summary of the model to verify that the model loaded correctly
print (loaded_model.summary())
```

2.1 Example1

1. Export this example Keras model (name it `model.json`).
2. Download this data set that we will use to train on (name it `pima-indians-diabetes.csv`).
3. Create a python file (name it `kerasJSONLoader.py`) and insert the following code:

```
from keras.models import model_from_json
import numpy
import os

# Fix random seed to allow similar accuracy measures at the end
numpy.random.seed(7)

# Load pima indians dataset
dataset = numpy.loadtxt('<path_to_file>/pima-indians-diabetes.csv', delimiter=',')

# Split the dataset into input (X) and output (Y) variables
X = dataset[:,0:8]
Y = dataset[:,8]

# Load the model from JSON file
json_file = open('<path_to_file>/model.json', 'r')
loaded_model_json = json_file.read()
json_file.close()
loaded_model = model_from_json(loaded_model_json)

# Configure model for training and testing with accuracy evaluation
loaded_model.compile(loss='binary_crossentropy', optimizer='adam', metrics=[
↪'accuracy'])

# Train the model
loaded_model.fit(X, Y, epochs=150, batch_size=10, verbose=0)

# Evaluate the model
scores = loaded_model.evaluate(X, Y, verbose=0)

# Print final accuracy
print("%s: %.2f%%" % (loaded_model.metrics_names[1], scores[1] * 100))
```

4. Then run the code in terminal.

```
python <path_to_file>/kerasJSONLoader.py
```

You should be getting around 76-78% accuracy.

This code trains and evaluates the loaded model on the dataset.

CHAPTER 3

Use exported model in Keras

3.1 To export the model for Keras in Fabrik

- Click on the export button in the Actions section of the sidebar.
- Select Keras option from the dropdown list.
  - A JSON file will be downloaded to your computer. It may take a while though.
- Rename the file to `kerasModel.json`.

3.2 Load the exported model in python and show it's summary.

- Open a terminal and cd into the directory where the `kerasModel.json` is saved.
- Do `touch kerasModelLoader.py`.
- Open the `kerasModelLoader.py` in any text editor.
- Type the following code into the editor.

```
#import keras' json model loader.
from keras.models import model_from_json

#Open the json file.
model = open('kerasModel.json', 'r')

#Read and close the json file.
loadedModel = model.read()
model.close()

#Load model from json file content.
loadedModel = model_from_json(loadedModel)
```

(continues on next page)

(continued from previous page)

```
#Print the summary
print (loadedModel.summary())
```

- Save the file at the same location where the `kerasModel.json` is saved and close the text editor.
- Switch to the terminal we were using earlier.
- Type `python kerasModelLoader.py`.
- Congrats! You should see a summary of the exported model.
- You can further use the model for training/testing purpose. Read about it more here.

CHAPTER 4

List of all models for which import/export has been tested with Fabrik.

4.1 Recognition

- Cifar10 CNN [Source][Visualise]
- MNIST LeNet [Source][Visualise]
- AlexNet [Source][Visualise]
- All CNN [Source][Visualise]
- CaffeNet [Source][Visualise]
- DenseNet [Source][Visualise]
- GoogLeNet [Source][Visualise]
- InceptionV3 [Source][Visualise]
- Network in Network [Source][Visualise]
- ResNet-101 [Source][Visualise]
- SqueezeNet [Source][Visualise]
- VGG-16 [Source][Visualise]
- DeepYeast [Source][Visualise]
- SpeechNet [Source][Visualise]
- SENet [Source] [Visualise]
- ZFNet [Source]

4.2 Detection

- Vanilla CNN [Source][Visualise]

- FCN32 Pascal [Source][Visualise]
- RCNN [Source][Visualise]
- YOLONet [Source][Visualise]
- Holistically-Nested Edge Detection [Source]

4.3 Retrieval

- MNIST Siamese [Source][Visualise]

4.4 Seq2Seq

- Seq2Seq Translation [Source][Visualise]
- Text Generation [Source][Visualise]
- Pix2Pix [Source] [Visualise]
- Denoising Auto-Encoder [Source]

4.5 Captioning

- COCO Caption [Source][Visualise]
- VQA [Source]
- VQA2 [Source][Visualise]
- VQS [Source] [Visualise]

CHAPTER 5

Segmentation

- Image Segmentation CRF-RNN [Source][Visualise]
- UNET [Source][Visualise]

5.1 Miscellaneous

- Ranking CNN [Source][Visualise]

CHAPTER 6

Using an Exported Caffe Model

In order to export a Caffe Model from Fabrik:

1. Select the 2nd button from the left in the Actions section of the sidebar.
2. A drop-down list should appear. Select Caffe.
    - This should download a prototxt file to your computer.
3. Rename the file to `model.prototxt`.
4. Create a file titled 'solver.prototxt' with the following:

```
net: "path/to/model.prototxt"     # path to the network
test_iter: 200                    # how many mini-batches to test in each validation␣
↪phase
test_interval: 500                # how often do we call the test phase
base_lr: 1e-5                     # base learning rate
lr_policy: "step"                 # step means to decrease lr after a number of␣
↪iterations
gamma: 0.1                        # ratio of decrement in each step
stepsize: 5000                    # how often do we step (should be called step_
↪interval)
display: 20                       # how often do we print training loss
max_iter: 450000                  # maximum amount of iterations
momentum: 0.9
weight_decay: 0.0005              # regularization!
snapshot: 2000                    # taking snapshot is like saving your progress in a␣
↪game
snapshot_prefix: "path/to/model" # path to saved model
solver_mode: GPU                  # choose CPU or GPU for processing, GPU is far␣
↪faster, but CPU is more supported.
```

1. Execute the following using caffe. `caffe` is the executable in the caffe folder (./build/tools/caffe). `solver.prototxt` should be the path to the file we just created.

```
caffe train \
  -gpu 0 \
  -solver solver.prototxt
```

CHAPTER 7

Use exported model in Caffe

7.1 To export the model for Caffe in Fabrik

- Click on the export button in the Actions section of the sidebar.
- Select Caffe option from the dropdown list.
  - A JSON file will be downloaded to your computer. It may take a while though.
- Rename the file to `caffeModel.prototxt`.

7.2 Load the exported model in python and show it's parameters and output sizes.

- Open a terminal and cd into the directory where the `caffeModel.prototxt` is saved.
- Do `touch caffeLoader.py`.
- Open the `caffeLoader.py` in any text editor.
- Type the following code into the editor.

```
import caffe
import numpy as np
from numpy import prod, sum
from pprint import pprint

caffe.set_mode_cpu()                # Change the mode cpu/gpu according to your␣
↪caffe installation

def model_details (model):
    net = caffe.Net(model, caffe.TEST)
    print "############################ Caffe Model Loaded ############################"
```

(continues on next page)

(continued from previous page)

```
    print "\nLayer-wise parameters: \n"
    pprint([(k, v[0].data.shape) for k, v in net.params.items()])
    print "\nTotal number of parameters: " + str(sum([prod(v[0].data.shape) for k, v␣
↪in net.params.items()]))

model = "model.prototxt"            # Change name and path of the model as and if␣
↪required

model_details(model)
```

- Save the file at the same location where the `model.prototxt` file is saved and close the text editor.
- Switch to the terminal we were using earlier.
- Type `python caffeLoader.py`.
- Congrats! You should see the model's parameters and output sizes.
- You can further use the model for training/testing purpose. Read about it more here.

CHAPTER 8

Adding New Layers

- For setup instructions, check README.
- Add your new layer(s) to the data.js file.

8.1 Basics for adding a new layer

- Open the `data.js` file in any text other.
- You should see the line `/* ********** Data Layers ********** */`, it is the category of the layer. There are many categories in the file as mentioned below:
  - Data Layers
  - Vision Layers
  - Recurrent Layers
  - Activation/Neuron Layers
  - Normalization Layers
  - Noise Layers
  - Common Layers
  - Loss Layers
  - Utility Layers
  - Python Layers
- You should add the new layer below the category it belongs to.
- Moving to the next line in the image, we create a new json element (layer). The line `// Only Caffe` tells that this layer is only for caffe and not for keras.
- Add the suitable comment for the new layer or leave it if there is no such need.

8.2 Detailed overview of a layer

- Here is a whole layer shown named `ReLU`. It is a `Activation/Neuron Layer`, that's why it is kept below the line `/* ********** Activation/Neron Layers ********** */`.
- Then add the suitable comment for you layer or leave it empty if it is not for any specific framework.
- Keywords' explanation:
  - name: Name of the layer.
  - color: Color of the layer to be shown in frontend.
  - endpoint: Endpoints of the layer.
    - src: Source endpoint of the layer.
    - trg: Target endpoint of the layer.
  - params: Parameters for the layer.
    - inplace: Checkbox input for the layer.
    - negative_slope: Numerical input for the layer.
    - caffe: Availibility of caffe (Checkbox input).
  - props: It defines the properties of the layer.
  - learn: This declares if the layer can be used for learning.
- We can define different parameters for a layer and it is not limited to `inplace` & `negative_slope`.

8.3 Making the layer visible in Fabrik

- Open pane.js in a text editor, and you should see something like this.
- Now, add a new line for the layer you just added in `data.js` in the section of Activation/Neuron Layer, because this layer belongs to this category.
- `<PaneElement handleClick={this.props.handleClick} id="your_layer_id">your_layer_name</PaneElement>` this line will make your layer visible in Fabrik.

8.4 Adding layer handling to the backend

- Open import_prototxt.py file in a text editor.
- Add a function for the new layer below the category of this layer.
- Load the parameters, do the calculations for your layer in pyhton and return the value of `params` (parameters).
- Move down in the file.
- Add your defined layer in the `layer_dict` array, as shown above.
- Now, open jsonToPrototxt.py in a text editor.
- Add an export function for training and testing of the new layer.
- There you need to load parameters, then train & test values and at last return the trained and tested data.
- Move down in this file as well.

- Add the export function in the `layer_map` array.

8.5 Testing and pushing the new layer.

- Run the fabrik application on you local machine by following the instructions in README file.
- Check the new layer inside the category you added it. See if all the parameters are properly displayed and usable as you wanted.
- If everything is working fine commit your changes and push it to your fork then make a Pull Request.
- Congratulations! Happy contributing :-)

CHAPTER 9

Indices and tables

- genindex
- modindex
- search