EBook Gratis

APRENDIZAJE caffe

Free unaffiliated eBook created from
Stack Overflow contributors.

#caffe

Tabla de contenido

Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: caffe

It is an unofficial and free caffe ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official caffe.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capítulo 1: Empezando con el cafe

Observaciones

Caffe es una biblioteca escrita en C ++, para facilitar la experimentación y el uso de redes neuronales convolucionales (CNN). Caffe ha sido desarrollado por Berkeley Vision and Learning Center (BVLC).

Caffe es en realidad una abreviatura que se refiere a "Arquitecturas convolucionales para la extracción rápida de características". Este acrónimo encapsula un alcance importante de la biblioteca. Caffe en forma de biblioteca ofrece un marco / arquitectura de programación general que se puede utilizar para realizar una capacitación y pruebas eficientes de las CNN. "Eficiencia" es un sello distintivo de caffe, y se destaca como un objetivo de diseño importante de Caffe.

Caffe es una biblioteca de código abierto lanzada bajo la licencia BSD 2 Clause.

Caffe se mantiene en GitHub

Caffe se puede utilizar para:

- Entrene y pruebe eficientemente múltiples arquitecturas CNN, específicamente cualquier arquitectura que pueda representarse como un gráfico acíclico dirigido (DAG).
- Utilice múltiples GPU (hasta 4) para entrenamiento y pruebas. Se recomienda que todas las GPU sean del mismo tipo. De lo contrario, el rendimiento está limitado por los límites de la GPU más lenta del sistema. Por ejemplo, en el caso de TitanX y GTX 980, el rendimiento estará limitado por este último. No se admite la mezcla de múltiples arquitecturas, por ejemplo, Kepler y Fermi 3 .

Caffe se ha escrito siguiendo los principios de la programación orientada a objetos (OOP) eficiente.

Un buen punto de partida para comenzar una introducción a caffe es obtener una vista de pájaro de cómo funciona caffe a través de sus objetos fundamentales.

Versiones

Versión	Fecha de lanzamiento
1.0	2017-04-19

Examples

Instalación y configuración

Ubuntu

A continuación, encontrará instrucciones detalladas para instalar Caffe, pycaffe y sus dependencias, en Ubuntu 14.04 x64 o 14.10 x64.

Ejecute el siguiente script, por ejemplo, "bash compile_caffe_ubuntu_14.sh" (~ 30 a 60 minutos en un nuevo Ubuntu).

```
# This script installs Caffe and pycaffe.
# CPU only, multi-threaded Caffe.

# Usage:
# 0. Set up here how many cores you want to use during the installation:
# By default Caffe will use all these cores.
NUMBER_OF_CORES=4

sudo apt-get install -y libprotobuf-dev libleveldb-dev libsnappy-dev
sudo apt-get install -y libopencv-dev libhdf5-serial-dev
sudo apt-get install -y --no-install-recommends libboost-all-dev
sudo apt-get install -y libatlas-base-dev
sudo apt-get install -y python-dev
sudo apt-get install -y python-pip git

# For Ubuntu 14.04
sudo apt-get install -y libgflags-dev libgoogle-glog-dev liblmdb-dev protobuf-compiler

# Install LMDB
git clone https://github.com/LMDB/lmdb.git
cd lmdb/libraries/liblmdb
sudo make
sudo make install

# More pre-requisites
sudo apt-get install -y cmake unzip doxygen
sudo apt-get install -y protobuf-compiler
sudo apt-get install -y libffi-dev python-pip python-dev build-essential
sudo pip install lmdb
sudo pip install numpy
sudo apt-get install -y python-numpy
sudo apt-get install -y gfortran # required by scipy
sudo pip install scipy # required by scikit-image
sudo apt-get install -y python-scipy # in case pip failed
sudo apt-get install -y python-nose
sudo pip install scikit-image # to fix https://github.com/BVLC/caffe/issues/50


# Get caffe (http://caffe.berkeleyvision.org/installation.html#compilation)
cd
mkdir caffe
cd caffe
wget https://github.com/BVLC/caffe/archive/master.zip
unzip -o master.zip
cd caffe-master

# Prepare Python binding (pycaffe)
cd python
for req in $(cat requirements.txt); do sudo pip install $req; done
```

```
# to be able to call "import caffe" from Python after reboot:
echo "export PYTHONPATH=$(pwd):$PYTHONPATH " >> ~/.bash_profile
source ~/.bash_profile # Update shell
cd ..

# Compile caffe and pycaffe
cp Makefile.config.example Makefile.config
sed -i '8s/.*/CPU_ONLY := 1/' Makefile.config # Line 8: CPU only
sudo apt-get install -y libopenblas-dev
sed -i '33s/.*/BLAS := open/' Makefile.config # Line 33: to use OpenBLAS
# Note that if one day the Makefile.config changes and these line numbers may change
echo "export OPENBLAS_NUM_THREADS=($NUMBER_OF_CORES)" >> ~/.bash_profile
mkdir build
cd build
cmake ..
cd ..
make all -j$NUMBER_OF_CORES # 4 is the number of parallel threads for compilation: typically
equal to number of physical cores
make pycaffe -j$NUMBER_OF_CORES
make test
make runtest
#make matcaffe
make distribute

# Afew few more dependencies for pycaffe
sudo pip install pydot
sudo apt-get install -y graphviz
sudo pip install scikit-learn
```

Al final, debes ejecutar "source ~ / .bash_profile" manualmente o iniciar un nuevo shell para poder hacer 'python import caffe'.

Habilitar multihilo con Caffe

Caffe puede ejecutarse en múltiples núcleos. Una forma es habilitar el multihilo con Caffe para usar OpenBLAS en lugar del ATLAS predeterminado. Para hacerlo, puedes seguir estos tres pasos:

1. `sudo apt-get install -y libopenblas-dev`
2. Antes de compilar Caffe, edita `Makefile.config` , reemplaza `BLAS := atlas` por `BLAS := open`
3. Después de compilar Caffe, ejecutar la `export OPENBLAS_NUM_THREADS=4` hará que Caffe utilice 4 núcleos.

Pérdida de regularización (caída de peso) en Caffe

En el archivo de resolución , podemos establecer una pérdida de regularización global utilizando las opciones `weight_decay` y `regularization_type` .

En muchos casos queremos diferentes tasas de caída de peso para diferentes capas. Esto se puede hacer configurando la opción `decay_mult` para cada capa en el archivo de definición de red, donde `decay_mult` es el multiplicador en la tasa de disminución de peso global, por lo que la tasa de disminución de peso real aplicada para una capa es `decay_mult*weight_decay` .

Por ejemplo, lo siguiente define una capa convolucional con NO disminución de peso

independientemente de las opciones en el archivo de resolución.

```
layer {
  name: "Convolution1"
  type: "Convolution"
  bottom: "data"
  top: "Convolution1"
  param {
    decay_mult: 0
  }
  convolution_param {
    num_output: 32
    pad: 0
    kernel_size: 3
    stride: 1
    weight_filler {
      type: "xavier"
    }
  }
}
```

Vea este hilo para más información.

Lea Empezando con el cafe en línea: https://riptutorial.com/es/caffe/topic/4382/empezando-con-el-cafe

Capítulo 2: Capas de Python personalizadas

Introducción

Este tutorial lo guiará a través de los pasos para crear una capa personalizada simple para Caffe usando python. Al final, hay algunos ejemplos de capas personalizadas. Por lo general, crearía una capa personalizada para implementar una funcionalidad que no está disponible en Caffe, ajustándola a sus requisitos.

La creación de una capa personalizada de Python agrega algo de sobrecarga a su red y probablemente no sea tan eficiente como una capa personalizada de C ++. Sin embargo, de esta manera, no tendrá que compilar todo el cafe con su nueva capa.

Parámetros

Parámetro	Detalles
parte superior	Una matriz con las mejores manchas de tu capa. Acceda a los datos que se le pasan usando *top [i] .data* , donde **i** es el índice de un blob específico
fondo	Una matriz con las manchas inferiores de tu capa. Acceda a los datos que se le pasan usando la *parte inferior [i] .data* , donde **i** es el índice de un blob específico

Observaciones

- Caffe construir con la capa de Python

Caffe debe compilarse con la opción `WITH_PYTHON_LAYER` :

```
WITH_PYTHON_LAYER=1 make && make pycaffe
```

- ¿Dónde debo guardar el archivo de clase?

Tienes dos opciones (al menos que yo sepa). O bien, puede guardar el archivo de capa personalizado en la misma carpeta en la que va a ejecutar el comando caffe (probablemente donde se encuentren sus archivos de prototxt). Otra forma, también mi favorita, es guardar todas sus capas personalizadas en una carpeta y agregar esta carpeta a su PYTHONPATH.

Referencias

1. El blog de Christopher Bourez.
2. Caffe Github
3. Desbordamiento de pila

Examples

Plantilla de capa

```
import caffe

class My_Custom_Layer(caffe.Layer):
    def setup(self, bottom, top):
        pass

    def forward(self, bottom, top):
        pass

    def reshape(self, bottom, top):
        pass

    def backward(self, bottom, top):
        pass
```

Cosas tan importantes para recordar:

- Tu capa personalizada debe heredar de **caffe.Layer** (así que no olvides *importar caffe*);
- Debe definir los cuatro métodos siguientes: **configuración** , **reenvío** , **remodelación** y **retroceso** ;
- Todos los métodos tienen *los* parámetros *superior* e *inferior* , que son las burbujas que almacenan la entrada y la salida que se pasa a su capa. Puede acceder a él utilizando *top [i] .data* o *bottom [i] .data* , donde *i* es el índice del blob en caso de que tenga más de un blob superior o inferior.

- Método de configuración

El método de configuración se llama una vez durante el tiempo de vida de la ejecución, cuando Caffe crea una instancia de todas las capas. Aquí es donde leerá los parámetros, creará una instancia de los buffers de tamaño fijo.

- remodelar el método

Utilice el método de remodelación para la inicialización / configuración que depende del tamaño del blob inferior (entrada de capa). Se llama una vez cuando se crea una instancia de la red.

- Método hacia adelante

Se llama al método Forward para cada lote de entrada y es donde estará la mayor parte de su lógica.

- Método hacia atrás

El método Backward se llama durante el paso hacia atrás de la red. Por ejemplo, en una capa similar a una convolución, esto sería donde se calcularían los gradientes. Esto es opcional (una capa puede ser solo hacia adelante).

Plantilla Prototxt

Ok, ahora tienes tu capa diseñada! Así es como lo define en su archivo *.prototxt* :

```
layer {
  name: "LayerName"
  type: "Python"
  top: "TopBlobName"
  bottom: "BottomBlobName"
  python_param {
    module: "My_Custom_Layer_File"
    layer: "My_Custom_Layer_Class"
    param_str: '{"param1": 1,"param2":True, "param3":"some string"}'
  }
  include{
        phase: TRAIN
  }
}
```

Observaciones importantes:

- **el tipo** debe ser **Python** ;
- Debe tener un diccionario **python_param** con al menos el **módulo** y los parámetros de **capa** ;
- **módulo se** refiere al archivo donde implementó su capa (sin el *.py*);
- **capa se** refiere al nombre de su clase;
- Puede pasar parámetros a la capa usando **param_str** (más información sobre cómo acceder a ellos más adelante);
- Al igual que cualquier otra capa, puede definir en qué fase desea que esté activa (vea los ejemplos para ver cómo puede verificar la fase actual);

Pasando parámetros a la capa.

Puede definir los parámetros de capa en el prototxt usando *param_str* . Una vez que lo haya hecho, este es un ejemplo de cómo acceder a estos parámetros dentro de la clase de capa:

```
def setup(self, bottom, top):
    params = eval(self.param_str)
    param1 = params["param1"]
    param2 = params.get('param2', False) #I usually use this when fetching a bool
    param3 = params["param3"]

    #Continue with the setup
    # ...
```

Capa de medida

En este ejemplo, diseñaremos una capa de "medida", que genera la matriz de precisión y confusión para un problema binario durante el entrenamiento y la precisión, la tasa de falsos positivos y la tasa de falsos negativos durante la prueba / validación. Aunque Caffe ya tiene una capa de precisión, a veces quieres algo más, como una F-medida.

Este es mi *measureLayer.py* con mi definición de clase:

```
#Remark: This class is designed for a binary problem, where the first class would be the
'negative'
# and the second class would be 'positive'

import caffe
TRAIN = 0
TEST = 1

class Measure_Layer(caffe.Layer):
    #Setup method
    def setup(self, bottom, top):
        #We want two bottom blobs, the labels and the predictions
        if len(bottom) != 2:
            raise Exception("Wrong number of bottom blobs (prediction and label)")

        #And some top blobs, depending on the phase
        if self.phase = TEST and len(top) != 3:
            raise Exception("Wrong number of top blobs (acc, FPR, FNR)")
        if self.phase = TRAIN and len(top) != 5:
            raise Exception("Wrong number of top blobs (acc, tp, tn, fp and fn)")

        #Initialize some attributes
        self.TPs = 0.0
        self.TNs = 0.0
        self.FPs = 0.0
        self.FNs = 0.0
        self.totalImgs = 0

    #Forward method
    def forward(self, bottom, top):
        #The order of these depends on the prototxt definition
        predictions = bottom[0].data
        labels = bottom[1].data

        self.totalImgs += len(labels)

        for i in range(len(labels)): #len(labels) is equal to the batch size
                pred = predictions[i]   #pred is a tuple with the normalized probability
                                        #of a sample i.r.t. two classes
                lab = labels[i]

                if pred[0] > pred[1]:
                        if lab == 1.0:
                                self.FNs += 1.0
                        else:
                                self.TNs += 1.0
                else:
                        if lab == 1.0:
                                self.TPs += 1.0
```

```
                        else:
                                self.FPs += 1.0

        acc = (self.TPs + self.TNs) / self.totalImgs

        try: #just assuring we don't divide by 0
                fpr = self.FPs / (self.FPs + self.TNs)
        except:
                fpr = -1.0

        try: #just assuring we don't divide by 0
                fnr = self.FNs / (self.FNs + self.TPs)
        except:
                fnr = -1.0

       #output data to top blob
       top[0].data = acc
       if self.phase == TRAIN:
           top[1].data = self.TPs
           top[2].data = self.TNs
           top[3].data = self.FPs
           top[4].data = self.FNs
       elif self.phase == TEST:
           top[1].data = fpr
           top[2].data = fnr

    def reshape(self, bottom, top):
        """
        We don't need to reshape or instantiate anything that is input-size sensitive
        """
        pass

    def backward(self, bottom, top):
        """
        This layer does not back propagate
        """
        pass
```

Y este es un ejemplo de un *prototxt* con él:

```
layer {
  name: "metrics"
  type: "Python"
  top: "Acc"
  top: "TPs"
  top: "TNs"
  top: "FPs"
  top: "FNs"

  bottom: "prediction"   #let's supose we have these two bottom blobs
  bottom: "label"

  python_param {
    module: "measureLayer"
    layer: "Measure_Layer"
  }
  include {
    phase: TRAIN
  }
}
```

```
layer {
  name: "metrics"
  type: "Python"
  top: "Acc"
  top: "FPR"
  top: "FNR"

  bottom: "prediction"   #let's supose we have these two bottom blobs
  bottom: "label"

  python_param {
    module: "measureLayer"
    layer: "Measure_Layer"
  }
  include {
    phase: TEST
  }
}
```

Capa de datos

Este ejemplo es una capa de datos personalizada, que recibe un archivo de texto con rutas de imagen, carga un lote de imágenes y las procesa previamente. Solo una sugerencia rápida, Caffe ya tiene una gran variedad de capas de datos y, probablemente, una capa personalizada no es la forma más eficiente si solo desea algo simple.

Mi *dataLayer.py* podría ser algo como:

```
import caffe

class Custom_Data_Layer(caffe.Layer):
    def setup(self, bottom, top):
        # Check top shape
        if len(top) != 2:
                raise Exception("Need to define tops (data and label)")

        #Check bottom shape
        if len(bottom) != 0:
            raise Exception("Do not define a bottom.")

        #Read parameters
        params = eval(self.param_str)
        src_file = params["src_file"]
        self.batch_size = params["batch_size"]
        self.im_shape = params["im_shape"]
        self.crop_size = params.get("crop_size", False)

        #Reshape top
        if self.crop_size:
            top[0].reshape(self.batch_size, 3, self.crop_size, self.crop_size)
        else:
            top[0].reshape(self.batch_size, 3, self.im_shape, self.im_shape)

        top[1].reshape(self.batch_size)

        #Read source file
        #I'm just assuming we have this method that reads the source file
```

```
        #and returns a list of tuples in the form of (img, label)
        self.imgTuples = readSrcFile(src_file)

        self._cur = 0 #use this to check if we need to restart the list of imgs

    def forward(self, bottom, top):
        for itt in range(self.batch_size):
            # Use the batch loader to load the next image.
            im, label = self.load_next_image()

            #Here we could preprocess the image
            # ...

            # Add directly to the top blob
            top[0].data[itt, ...] = im
            top[1].data[itt, ...] = label

    def load_next_img(self):
        #If we have finished forwarding all images, then an epoch has finished
        #and it is time to start a new one
        if self._cur == len(self.imgTuples):
            self._cur = 0
            shuffle(self.imgTuples)

        im, label = self.imgTuples[self._cur]
        self._cur += 1

        return im, label

    def reshape(self, bottom, top):
        """
        There is no need to reshape the data, since the input is of fixed size
        (img shape and batch size)
        """
        pass

    def backward(self, bottom, top):
        """
        This layer does not back propagate
        """
        pass
```

Y el *prototexto* sería como:

```
layer {
  name: "Data"
  type: "Python"
  top: "data"
  top: "label"

  python_param {
    module: "dataLayer"
    layer: "Custom_Data_Layer"
    param_str: '{"batch_size": 126,"im_shape":256, "crop_size":224, "src_file":
"path_to_TRAIN_file.txt"}'
  }
}
```

Lea Capas de Python personalizadas en línea: https://riptutorial.com/es/caffe/topic/10535/capas-

de-python-personalizadas

Capítulo 3: Entrenando un modelo de Caffe con pycaffe.

Examples

Entrenando una red en el conjunto de datos Iris

A continuación se muestra un ejemplo simple para entrenar un modelo de Caffe en el conjunto de datos Iris en Python, usando PyCaffe. También da las salidas previstas dadas algunas entradas definidas por el usuario.

`iris_tuto.py`

```
import subprocess
import platform
import copy

from sklearn.datasets import load_iris
import sklearn.metrics
import numpy as np
from sklearn.cross_validation import StratifiedShuffleSplit
import matplotlib.pyplot as plt
import h5py
import caffe
import caffe.draw


def load_data():
    '''
    Load Iris Data set
    '''
    data = load_iris()
    print(data.data)
    print(data.target)
    targets = np.zeros((len(data.target), 3))
    for count, target in enumerate(data.target):
        targets[count][target]= 1
    print(targets)

    new_data = {}
    #new_data['input'] = data.data
    new_data['input'] = np.reshape(data.data, (150,1,1,4))
    new_data['output'] = targets
    #print(new_data['input'].shape)
    #new_data['input'] = np.random.random((150, 1, 1, 4))
    #print(new_data['input'].shape)
    #new_data['output'] = np.random.random_integers(0, 1, size=(150,3))
    #print(new_data['input'])

    return new_data

def save_data_as_hdf5(hdf5_data_filename, data):
    '''
    HDF5 is one of the data formats Caffe accepts
```

```
    '''
    with h5py.File(hdf5_data_filename, 'w') as f:
        f['data'] = data['input'].astype(np.float32)
        f['label'] = data['output'].astype(np.float32)


def train(solver_prototxt_filename):
    '''
    Train the ANN
    '''
    caffe.set_mode_cpu()
    solver = caffe.get_solver(solver_prototxt_filename)
    solver.solve()


def print_network_parameters(net):
    '''
    Print the parameters of the network
    '''
    print(net)
    print('net.inputs: {0}'.format(net.inputs))
    print('net.outputs: {0}'.format(net.outputs))
    print('net.blobs: {0}'.format(net.blobs))
    print('net.params: {0}'.format(net.params))

def get_predicted_output(deploy_prototxt_filename, caffemodel_filename, input, net = None):
    '''
    Get the predicted output, i.e. perform a forward pass
    '''
    if net is None:
        net = caffe.Net(deploy_prototxt_filename,caffemodel_filename, caffe.TEST)

    #input = np.array([[ 5.1,  3.5,  1.4,  0.2]])
    #input = np.random.random((1, 1, 1))
    #print(input)
    #print(input.shape)
    out = net.forward(data=input)
    #print('out: {0}'.format(out))
    return out[net.outputs[0]]


import google.protobuf
def print_network(prototxt_filename, caffemodel_filename):
    '''
    Draw the ANN architecture
    '''
    _net = caffe.proto.caffe_pb2.NetParameter()
    f = open(prototxt_filename)
    google.protobuf.text_format.Merge(f.read(), _net)
    caffe.draw.draw_net_to_file(_net, prototxt_filename + '.png' )
    print('Draw ANN done!')


def print_network_weights(prototxt_filename, caffemodel_filename):
    '''
    For each ANN layer, print weight heatmap and weight histogram
    '''
    net = caffe.Net(prototxt_filename,caffemodel_filename, caffe.TEST)
    for layer_name in net.params:
        # weights heatmap
        arr = net.params[layer_name][0].data
```

```
        plt.clf()
        fig = plt.figure(figsize=(10,10))
        ax = fig.add_subplot(111)
        cax = ax.matshow(arr, interpolation='none')
        fig.colorbar(cax, orientation="horizontal")
        plt.savefig('{0}_weights_{1}.png'.format(caffemodel_filename, layer_name), dpi=100,
format='png', bbox_inches='tight') # use format='svg' or 'pdf' for vectorial pictures
        plt.close()

        # weights histogram
        plt.clf()
        plt.hist(arr.tolist(), bins=20)
        plt.savefig('{0}_weights_hist_{1}.png'.format(caffemodel_filename, layer_name),
dpi=100, format='png', bbox_inches='tight') # use format='svg' or 'pdf' for vectorial pictures
        plt.close()


def get_predicted_outputs(deploy_prototxt_filename, caffemodel_filename, inputs):
    '''
    Get several predicted outputs
    '''
    outputs = []
    net = caffe.Net(deploy_prototxt_filename,caffemodel_filename, caffe.TEST)
    for input in inputs:
        #print(input)
        outputs.append(copy.deepcopy(get_predicted_output(deploy_prototxt_filename,
caffemodel_filename, input, net)))
    return outputs


def get_accuracy(true_outputs, predicted_outputs):
    '''

    '''
    number_of_samples = true_outputs.shape[0]
    number_of_outputs = true_outputs.shape[1]
    threshold = 0.0 # 0 if SigmoidCrossEntropyLoss ; 0.5 if EuclideanLoss
    for output_number in range(number_of_outputs):
        predicted_output_binary = []
        for sample_number in range(number_of_samples):
            #print(predicted_outputs)
            #print(predicted_outputs[sample_number][output_number])
            if predicted_outputs[sample_number][0][output_number] < threshold:
                predicted_output = 0
            else:
                predicted_output = 1
            predicted_output_binary.append(predicted_output)

        print('accuracy: {0}'.format(sklearn.metrics.accuracy_score(true_outputs[:,
output_number], predicted_output_binary)))
        print(sklearn.metrics.confusion_matrix(true_outputs[:, output_number],
predicted_output_binary))


def main():
    '''
    This is the main function
    '''

    # Set parameters
    solver_prototxt_filename = 'iris_solver.prototxt'
```

```
    train_test_prototxt_filename = 'iris_train_test.prototxt'
    deploy_prototxt_filename  = 'iris_deploy.prototxt'
    deploy_prototxt_filename  = 'iris_deploy.prototxt'
    deploy_prototxt_batch2_filename  = 'iris_deploy_batchsize2.prototxt'
    hdf5_train_data_filename = 'iris_train_data.hdf5'
    hdf5_test_data_filename = 'iris_test_data.hdf5'
    caffemodel_filename = 'iris__iter_5000.caffemodel' # generated by train()

    # Prepare data
    data = load_data()
    print(data)
    train_data = data
    test_data = data
    save_data_as_hdf5(hdf5_train_data_filename, data)
    save_data_as_hdf5(hdf5_test_data_filename, data)

    # Train network
    train(solver_prototxt_filename)

    # Print network
    print_network(deploy_prototxt_filename, caffemodel_filename)
    print_network(train_test_prototxt_filename, caffemodel_filename)
    print_network_weights(train_test_prototxt_filename, caffemodel_filename)

    # Compute performance metrics
    #inputs = input = np.array([[[[ 5.1,  3.5,  1.4,  0.2]]],[[[ 5.9,  3. ,  5.1,  1.8]]]])
    inputs = data['input']
    outputs = get_predicted_outputs(deploy_prototxt_filename, caffemodel_filename, inputs)
    get_accuracy(data['output'], outputs)


if __name__ == "__main__":
    main()
```

Requiere que los dos `iris_train_test.prototxt` e `iris_deploy.prototxt` siguientes estén en la misma carpeta.

`iris_train_test.prototxt` :

```
name: "IrisNet"
layer {
  name: "iris"
  type: "HDF5Data"
  top: "data"
  top: "label"
  include {
    phase: TRAIN
  }
  hdf5_data_param {
    source: "iris_train_data.txt"
    batch_size: 1

  }
}

layer {
  name: "iris"
  type: "HDF5Data"
  top: "data"
```

```
  top: "label"
  include {
    phase: TEST
  }
  hdf5_data_param {
    source: "iris_test_data.txt"
    batch_size: 1

  }
}



layer {
  name: "ip1"
  type: "InnerProduct"
  bottom: "data"
  top: "ip1"
  param {
    lr_mult: 1  # the learning rate multiplier for weights
  }
  param {
    lr_mult: 2  # the learning rate multiplier for biases
  }
  inner_product_param {
    num_output: 50
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
}
layer {
  name: "relu1"
  type: "ReLU"
  bottom: "ip1"
  top: "ip1"
}
layer {
  name: "drop1"
  type: "Dropout"
  bottom: "ip1"
  top: "ip1"
  dropout_param {
    dropout_ratio: 0.5
  }
}


layer {
  name: "ip2"
  type: "InnerProduct"
  bottom: "ip1"
  top: "ip2"
  param {
    lr_mult: 1
  }
  param {
```

```
    lr_mult: 2
  }
  inner_product_param {
    num_output: 50
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
}
layer {
  name: "drop2"
  type: "Dropout"
  bottom: "ip2"
  top: "ip2"
  dropout_param {
    dropout_ratio: 0.4
  }
}



layer {
  name: "ip3"
  type: "InnerProduct"
  bottom: "ip2"
  top: "ip3"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  inner_product_param {
    num_output: 3
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
}

layer {
  name: "drop3"
  type: "Dropout"
  bottom: "ip3"
  top: "ip3"
  dropout_param {
    dropout_ratio: 0.3
  }
}

layer {
  name: "loss"
  type: "SigmoidCrossEntropyLoss"
  # type: "EuclideanLoss"
  # type: "HingeLoss"
```

```
  bottom: "ip3"
  bottom: "label"
  top: "loss"
}
```

`iris_deploy.prototxt` :

```
name: "IrisNet"
input: "data"
input_dim: 1 # batch size
input_dim: 1
input_dim: 1
input_dim: 4


layer {
  name: "ip1"
  type: "InnerProduct"
  bottom: "data"
  top: "ip1"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  inner_product_param {
    num_output: 50
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
}
layer {
  name: "relu1"
  type: "ReLU"
  bottom: "ip1"
  top: "ip1"
}
layer {
  name: "drop1"
  type: "Dropout"
  bottom: "ip1"
  top: "ip1"
  dropout_param {
    dropout_ratio: 0.5
  }
}


layer {
  name: "ip2"
  type: "InnerProduct"
  bottom: "ip1"
  top: "ip2"
  param {
    lr_mult: 1
```

```
  }
  param {
    lr_mult: 2
  }
  inner_product_param {
    num_output: 50
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
}
layer {
  name: "drop2"
  type: "Dropout"
  bottom: "ip2"
  top: "ip2"
  dropout_param {
    dropout_ratio: 0.4
  }
}


layer {
  name: "ip3"
  type: "InnerProduct"
  bottom: "ip2"
  top: "ip3"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  inner_product_param {
    num_output: 3
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
}

layer {
  name: "drop3"
  type: "Dropout"
  bottom: "ip3"
  top: "ip3"
  dropout_param {
    dropout_ratio: 0.3
  }
}
```

iris_solver.prototxt

```
# The train/test net protocol buffer definition
```

```
net: "iris_train_test.prototxt"
# test_iter specifies how many forward passes the test should carry out.
test_iter: 1
# Carry out testing every test_interval training iterations.
test_interval: 1000
# The base learning rate, momentum and the weight decay of the network.
base_lr: 0.0001
momentum: 0.001
weight_decay: 0.0005
# The learning rate policy
lr_policy: "inv"
gamma: 0.0001
power: 0.75
# Display every 100 iterations
display: 1000
# The maximum number of iterations
max_iter: 5000
# snapshot intermediate results
snapshot: 5000
snapshot_prefix: "iris_"
# solver mode: CPU or GPU
solver_mode: CPU # GPU
```

Lea Entrenando un modelo de Caffe con pycaffe. en línea:
https://riptutorial.com/es/caffe/topic/4618/entrenando-un-modelo-de-caffe-con-pycaffe-

Capítulo 4: Normalización de lotes

Introducción

De la documentación :

"Normaliza la entrada para tener una variación de 0 media y / o unidad (1) en el lote.

Esta capa calcula la normalización por lotes como se describe en [1].

[...]

[1] S. Ioffe y C. Szegedy, "Normalización de lotes: Aceleración de la capacitación profunda en la red mediante la reducción del cambio interno de covarianza". arXiv preimpresión arXiv: 1502.03167 (2015). "

Parámetros

Parámetro	Detalles
use_global_stats	De la publicación de rohrbach del 2 de marzo de 2016 , quizás él sepa:
(use_global_stats)	"De manera predeterminada, durante el tiempo de entrenamiento, la red está calculando estadísticas de media / varianza global a través de un promedio de ejecución, que luego se usa en el tiempo de prueba para permitir salidas deterministas para cada entrada. Puede alternar manualmente si la red se está acumulando o usando las estadísticas a través de la opción use_global_stats. IMPORTANTE: para que esta función funcione, DEBE establecer la velocidad de aprendizaje en cero para los tres parámetros de blobs, es decir, param {lr_mult: 0} tres veces en la definición de capa.
(use_global_stats)	Esto significa que, de forma predeterminada (como se establece a continuación en batch_norm_layer.cpp), no tiene que establecer use_global_stats en absoluto en el prototxt. use_global_stats_ = this-> phase_ == TEST; "

Examples

Prototexto para la formación.

La siguiente es una definición de ejemplo para entrenar una capa BatchNorm con escala y sesgo en cuanto al canal. Normalmente, una capa BatchNorm se inserta entre las capas de convolución y rectificación. En este ejemplo, la convolución generaría el blob `layerx` y la rectificación recibiría

el blob `layerx-bn` .

```
layer { bottom: 'layerx' top: 'layerx-bn' name: 'layerx-bn' type: 'BatchNorm'
  batch_norm_param {
    use_global_stats: false  # calculate the mean and variance for each mini-batch
    moving_average_fraction: .999  # doesn't effect training
  }
  param { lr_mult: 0 }
  param { lr_mult: 0 }
  param { lr_mult: 0 }}
# channel-wise scale and bias are separate
layer { bottom: 'layerx-bn' top: 'layerx-bn' name: 'layerx-bn-scale' type: 'Scale',
  scale_param {
    bias_term: true
    axis: 1      # scale separately for each channel
    num_axes: 1  # ... but not spatially (default)
    filler { type: 'constant' value: 1 }           # initialize scaling to 1
    bias_filler { type: 'constant' value: 0.001 }  # initialize bias
}}
```

Más información se puede encontrar en este hilo .

Prototxt para la implementación

El cambio principal necesario es cambiar `use_global_stats` a `true` . Esto cambia al uso de la media móvil.

```
layer { bottom: 'layerx' top: 'layerx-bn' name: 'layerx-bn' type: 'BatchNorm'
  batch_norm_param {
    use_global_stats: true  # use pre-calculated average and variance
  }
  param { lr_mult: 0 }
  param { lr_mult: 0 }
  param { lr_mult: 0 }}
# channel-wise scale and bias are separate
layer { bottom: 'layerx-bn' top: 'layerx-bn' name: 'layerx-bn-scale' type: 'Scale',
  scale_param {
    bias_term: true
    axis: 1      # scale separately for each channel
    num_axes: 1  # ... but not spatially (default)
}}
```

Lea Normalización de lotes en línea: https://riptutorial.com/es/caffe/topic/6575/normalizacion-de-lotes

Capítulo 5: Objetos básicos de Caffe - Solver, Net, Layer y Blob

Observaciones

Un usuario de caffe envía instrucciones para realizar operaciones específicas para caffe objetos. Estos objetos interactúan entre sí en función de sus especificaciones de diseño y realizan la (s) operación (es). Este es un principio básico del paradigma OOP.

Si bien hay muchos tipos de objetos de cafeína (o clases de C ++), para una comprensión básica, nos centramos en 4 objetos importantes de cafeína. Nuestro objetivo en esta etapa es simplemente observar la interacción entre estos objetos en un nivel altamente abstracto donde los detalles específicos de la implementación y el diseño son confusos, y en su lugar se enfoca una vista de la operación a vista de pájaro.

Los 4 objetos básicos de cafe son:

- **Solucionador**
- **Red**
- **Capa**
- **Gota**

Una introducción muy básica y una vista panorámica de su papel en el trabajo de caffe se presenta en puntos concisos en la sección de ejemplos.

Después de leer y obtener una idea básica de cómo interactúan estos objetos de cafeína, se puede leer cada tipo de objeto en detalle en sus temas dedicados.

Examples

Cómo estos objetos interactúan entre sí.

- Un usuario está buscando usar caffe para el entrenamiento y las pruebas de CNN. El usuario decide el diseño de la arquitectura CNN (por ejemplo, No. de capas, No. de filtros y sus detalles, etc.). El usuario también decide la técnica de optimización para los parámetros de entrenamiento y aprendizaje en caso de que se lleve a cabo el entrenamiento. Si la operación es de prueba de vainilla, el usuario especifica un modelo pre-entrenado. Utilizando toda esta información, el usuario crea una instancia de un objeto Solver y proporciona al objeto Solver una instrucción (que decide la (s) operación (es) como entrenar y probar).

- **Solucionador** : este objeto se puede considerar como una entidad que supervisa la capacitación y las pruebas de una CNN. Es el contratista real quien obtiene un CNN en el procesador y en funcionamiento. Está especializada en llevar a cabo las optimizaciones específicas que llevan a una CNN a capacitarse.

- **Net** : Net puede considerarse como un objeto especializado que representa la CNN real sobre la que se realizan las operaciones. Net le indica a Net que asigne realmente memoria a la CNN y la ejecute. Net también es responsable de dar instrucciones que realmente conduzcan a la propagación hacia adelante o hacia atrás a través de la CNN.

- **Capa** : es un objeto que representa una capa particular de una CNN. Así, una CNN está formada por capas. En lo que respecta a caffe, el objeto **Net** crea una instancia de cada tipo de " **Capa** " especificada en la definición de la arquitectura y también conecta diferentes capas entre sí. Una capa específica lleva a cabo un conjunto específico de operaciones (por ejemplo, Max-Pooling, Min-Pooling, 2D Convolution, etc.)

- **Blob** : los datos fluyen a través de una CNN durante el entrenamiento y las pruebas. Estos datos, además de contener datos de usuario, también incluyen varios cálculos intermedios que se realizan a través de CNN. Estos datos están encapsulados en un objeto llamado Blob.

Lea Objetos básicos de Caffe - Solver, Net, Layer y Blob en línea:
https://riptutorial.com/es/caffe/topic/5810/objetos-basicos-de-caffe---solver--net--layer-y-blob

Capítulo 6: Preparar datos para el entrenamiento

Examples

Preparar conjunto de datos de imagen para la tarea de clasificación de imágenes

Caffe tiene una capa de entrada incorporada adaptada para las tareas de clasificación de imágenes (es decir, una etiqueta entera única por imagen de entrada). Esta capa de `"Data"` entrada se basa en una estructura de datos lmdb o leveldb . Para usar la capa `"Data"` , se debe construir la estructura de datos con todos los datos de entrenamiento.

Una guía rápida de `convert_imageset` de Caffe

Construir

Lo primero que debes hacer es crear las herramientas de caffe y caffe (`convert_imageset` es una de estas herramientas).
Después de instalar caffe y `make` de que también ejecutó `make tools` .
Verifique que se haya creado un archivo binario `convert_imageset` en `$CAFFE_ROOT/build/tools` .

Prepara tus datos

Imágenes: ponga todas las imágenes en una carpeta (lo llamaré aquí `/path/to/jpegs/`).
Etiquetas: cree un archivo de texto (por ejemplo, `/path/to/labels/train.txt`) con una línea por imagen de entrada <ruta / a / archivo>. Por ejemplo:

> img_0000.jpeg 1
> img_0001.jpeg 0
> img_0002.jpeg 0

En este ejemplo, la primera imagen está etiquetada como `1` mientras que las otras dos están etiquetadas como `0` .

Convertir el conjunto de datos

Ejecutar el binario en shell

```
~$ GLOG_logtostderr=1 $CAFFE_ROOT/build/tools/convert_imageset \
    --resize_height=200 --resize_width=200 --shuffle  \
    /path/to/jpegs/ \
    /path/to/labels/train.txt \
    /path/to/lmdb/train_lmdb
```

Línea de comando explicada:

- `GLOG_logtostderr` indicador `GLOG_logtostderr` se establece en 1 *antes de* llamar a `convert_imageset` indica el mecanismo de registro para redirigir los mensajes de registro a stderr.
- `--resize_height` y `--resize_width` redimensionan **todas las** imágenes de entrada al mismo tamaño `200x200` .
- `--shuffle` cambia aleatoriamente el orden de las imágenes y no conserva el orden en el archivo `/path/to/labels/train.txt` .
- A continuación se muestra la ruta a la carpeta de imágenes, el archivo de texto de etiquetas y el nombre de salida. Tenga en cuenta que el nombre de salida no debería existir antes de llamar a `convert_imageset` contrario, `convert_imageset` un mensaje de error de miedo.

Otras banderas que podrían ser útiles:

- `--backend` - le permite elegir entre un `lmdb` conjunto de datos o `levelDB` .
- `--gray` - convierte todas las imágenes a escala de grises.
- `--encoded` y `--encoded_type` - mantienen los datos de la imagen en formato comprimido (jpg / png) en la base de datos.
- `--help` - muestra ayuda, vea todos los indicadores relevantes en *Indicadores de tools / convert_imageset.cpp*

Puede consultar `$CAFFE_ROOT/examples/imagenet/convert_imagenet.sh` para ver un ejemplo de cómo usar `convert_imageset` .

ver este hilo para más información.

Preparar datos arbitrarios en formato HDF5.

Además de los conjuntos de datos de clasificación de imágenes , Caffe también tiene `"HDF5Data"` capa `"HDF5Data"` para entradas arbitrarias. Esta capa requiere que todos los datos de capacitación / validación se almacenen en archivos en formato hdf5 .
Este ejemplo muestra cómo usar el módulo `h5py` Python para construir dicho archivo hdf5 y cómo configurar la capa `"HDF5Data"` caffe para leer ese archivo.

Construye el archivo binario hdf5

Suponiendo que tiene un archivo de texto `'train.txt'` con cada línea con un nombre de archivo de imagen y un único número de punto flotante para ser utilizado como objetivo de regresión.

```
import h5py, os
import caffe
import numpy as np

SIZE = 224 # fixed size to all images
with open( 'train.txt', 'r' ) as T :
    lines = T.readlines()
# If you do not have enough memory split data into
# multiple batches and generate multiple separate h5 files
X = np.zeros( (len(lines), 3, SIZE, SIZE), dtype='f4' )
```

```
y = np.zeros( (1,len(lines)), dtype='f4' )
for i,l in enumerate(lines):
    sp = l.split(' ')
    img = caffe.io.load_image( sp[0] )
    img = caffe.io.resize( img, (SIZE, SIZE, 3) ) # resize to fixed size
    # you may apply other input transformations here...
    # Note that the transformation should take img from size-by-size-by-3 and transpose it to
3-by-size-by-size
    X[i] = img
    y[i] = float(sp[1])
with h5py.File('train.h5','w') as H:
    H.create_dataset( 'X', data=X ) # note the name X given to the dataset!
    H.create_dataset( 'y', data=y ) # note the name y given to the dataset!
with open('train_h5_list.txt','w') as L:
    L.write( 'train.h5' ) # list all h5 files you are going to use
```

Configurando la capa `"HDF5Data"`

Una vez que tenga todos los archivos `h5` y los archivos de prueba correspondientes que los enumeran, puede agregar una capa de entrada HDF5 a su `train_val.prototxt` :

```
layer {
  type: "HDF5Data"
  top: "X" # same name as given in create_dataset!
  top: "y"
  hdf5_data_param {
    source: "train_h5_list.txt" # do not give the h5 files directly, but the list.
    batch_size: 32
  }
  include { phase:TRAIN }
}
```

Puedes encontrar más información aquí y aquí .

Como se muestra arriba, pasamos a Caffe una lista de archivos HDF5. Esto se debe a que en la versión actual hay un límite de tamaño de 2 GB para un solo archivo de datos HDF5. Entonces, si los datos de entrenamiento superan los 2 GB, tendremos que dividirlos en archivos separados.

Si un solo archivo de datos HDF5 supera los 2 GB, aparecerá un mensaje de error como

```
Check failed: shape[i] <= 2147483647 / count_ (100 vs. 71) blob size exceeds INT_MAX
```

Si la cantidad total de datos es inferior a 2 GB, ¿dividiremos los datos en archivos separados o no?

De acuerdo con un comentario en el código fuente de Caffe , un solo archivo sería mejor,

> Si shuffle == true, el orden de los archivos HDF5 se baraja, y el orden de los datos dentro de cualquier archivo HDF5 dado se barajan, pero los datos entre archivos diferentes no se intercalan.

Lea Preparar datos para el entrenamiento en línea:
https://riptutorial.com/es/caffe/topic/5344/preparar-datos-para-el-entrenamiento

Creditos

S. No	Capítulos	Contributors
1	Empezando con el cafe	Community, dontloo, Franck Dernoncourt, GoodDeeds, Parag S. Chandakkar, Shai, Tahir Shahzad, Ujjwal Aryan
2	Capas de Python personalizadas	Fernanda Andalo, rafaspadilha
3	Entrenando un modelo de Caffe con pycaffe.	Franck Dernoncourt, Parag S. Chandakkar
4	Normalización de lotes	dasWesen, Jonathan, Shai
5	Objetos básicos de Caffe - Solver, Net, Layer y Blob	Ujjwal Aryan
6	Preparar datos para el entrenamiento	dontloo, malreddysid, Shai, Stephen Leppik