Tutoriel caffe

Stefan Duffner

Laboratoire d'InfoRmatique en Image et Systèmes d'information
LIRIS, UMR 5205 CNRS
INSA de Lyon, France
http://liris.cnrs.fr

27/06/2016

Outline

Outline

Objectifs

Le but de ce tutoriel

1. connaître les fonctionnement de caffe
2. apprendre à utiliser la bibliothèque

Objectifs

Le but de ce tutoriel

1. connaître les fonctionnement de caffe
2. apprendre à utiliser la bibliothèque

Objectifs

Le but de ce tutoriel

1. connaître les fonctionnement de caffe
2. apprendre à utiliser la bibliothèque

caffe

- bibliothèque open-source, développé par BVLC
- très répandu dans la communauté "Computer Vision" (ImageNet)
- http://caffe.berkeleyvision.org
- http://github.com/BVLC/caffe
- C++ (avec interface Python ou Matlab)
- implémentation CPU/GPU rapide
- approche (classique) par "couches"
- points négatifs : peu de documentation, difficile à étendre, très orienté ImageNet
- autres bibliothèques : Torch (Lua), Theano (Python), Cuda-Convnet2 (C++), TensorFlow (Python), MatConvNet (Matlab), mxnet (C++, Python, R, Scala, Julia, Matlab)

caffe

- bibliothèque open-source, développé par BVLC
- très répandu dans la communauté "Computer Vision" (ImageNet)
- http://caffe.berkeleyvision.org
- http://github.com/BVLC/caffe
- C++ (avec interface Python ou Matlab)
- implémentation CPU/GPU rapide
- approche (classique) par "couches"
- points négatifs : peu de documentation, difficile à étendre, très orienté ImageNet
- autres bibliothèques : Torch (Lua), Theano (Python), Cuda-Convnet2 (C++), TensorFlow (Python), MatConvNet (Matlab), mxnet (C++, Python, R, Scala, Julia, Matlab)

caffe

- bibliothèque open-source, développé par BVLC
- très répandu dans la communauté "Computer Vision" (ImageNet)
- http://caffe.berkeleyvision.org
- http://github.com/BVLC/caffe
- C++ (avec interface Python ou Matlab)
- implémentation CPU/GPU rapide
- approche (classique) par "couches"
- points négatifs : peu de documentation, difficile à étendre, très orienté ImageNet
- autres bibliothèques : Torch (Lua), Theano (Python), Cuda-Convnet2 (C++), TensorFlow (Python), MatConvNet (Matlab), mxnet (C++, Python, R, Scala, Julia, Matlab)

caffe

- bibliothèque open-source, développé par BVLC
- très répandu dans la communauté "Computer Vision" (ImageNet)
- http://caffe.berkeleyvision.org
- http://github.com/BVLC/caffe
- C++ (avec interface Python ou Matlab)
- implémentation CPU/GPU rapide
- approche (classique) par "couches"
- points négatifs : peu de documentation, difficile à étendre, très orienté ImageNet
- autres bibliothèques : Torch (Lua), Theano (Python), Cuda-Convnet2 (C++), TensorFlow (Python), MatConvNet (Matlab), mxnet (C++, Python, R, Scala, Julia, Matlab)

caffe

- bibliothèque open-source, développé par BVLC
- très répandu dans la communauté "Computer Vision" (ImageNet)
- http://caffe.berkeleyvision.org
- http://github.com/BVLC/caffe
- C++ (avec interface Python ou Matlab)
- implémentation CPU/GPU rapide
- approche (classique) par "couches"
- points négatifs : peu de documentation, difficile à étendre, très orienté ImageNet
- autres bibliothèques : Torch (Lua), Theano (Python), Cuda-Convnet2 (C++), TensorFlow (Python), MatConvNet (Matlab), mxnet (C++, Python, R, Scala, Julia, Matlab)

caffe

- bibliothèque open-source, développé par BVLC
- très répandu dans la communauté "Computer Vision" (ImageNet)
- http://caffe.berkeleyvision.org
- http://github.com/BVLC/caffe
- C++ (avec interface Python ou Matlab)
- implémentation CPU/GPU rapide
- approche (classique) par "couches"
- points négatifs : peu de documentation, difficile à étendre, très orienté ImageNet
- autres bibliothèques : Torch (Lua), Theano (Python), Cuda-Convnet2 (C++), TensorFlow (Python), MatConvNet (Matlab), mxnet (C++, Python, R, Scala, Julia, Matlab)

caffe

- bibliothèque open-source, développé par BVLC
- très répandu dans la communauté "Computer Vision" (ImageNet)
- http://caffe.berkeleyvision.org
- http://github.com/BVLC/caffe
- C++ (avec interface Python ou Matlab)
- implémentation CPU/GPU rapide
- approche (classique) par "couches"
- points négatifs : peu de documentation, difficile à étendre, très orienté ImageNet
- autres bibliothèques : Torch (Lua), Theano (Python), Cuda-Convnet2 (C++), TensorFlow (Python), MatConvNet (Matlab), mxnet (C++, Python, R, Scala, Julia, Matlab)

caffe

- bibliothèque open-source, développé par BVLC
- très répandu dans la communauté "Computer Vision" (ImageNet)
- http://caffe.berkeleyvision.org
- http://github.com/BVLC/caffe
- C++ (avec interface Python ou Matlab)
- implémentation CPU/GPU rapide
- approche (classique) par "couches"
- points négatifs : peu de documentation, difficile à étendre, très orienté ImageNet
- autres bibliothèques : Torch (Lua), Theano (Python), Cuda-Convnet2 (C++), TensorFlow (Python), MatConvNet (Matlab), mxnet (C++, Python, R, Scala, Julia, Matlab)

caffe

- bibliothèque open-source, développé par BVLC
- très répandu dans la communauté "Computer Vision" (ImageNet)
- http://caffe.berkeleyvision.org
- http://github.com/BVLC/caffe
- C++ (avec interface Python ou Matlab)
- implémentation CPU/GPU rapide
- approche (classique) par "couches"
- points négatifs : peu de documentation, difficile à étendre, très orienté ImageNet
- autres bibliothèques : Torch (Lua), Theano (Python), Cuda-Convnet2 (C++), TensorFlow (Python), MatConvNet (Matlab), mxnet (C++, Python, R, Scala, Julia, Matlab)

Outline

Documentation

- http://caffe.berkeleyvision.org/tutorial
- http://tutorial.caffe.berkeleyvision.org (CVPR 2015 tutorial)
- Diapositives de Princeton Vision Group
- Diapositives de Stanford Computer Vision Lab
- Intel / TAU Summer Workshop on Deep Learning and Caffe (de Boris Ginzburg)
- NVIDIA: Deep Learning for Computer Vision with Caffe and cuDNN

Documentation

- http://caffe.berkeleyvision.org/tutorial
- http://tutorial.caffe.berkeleyvision.org (CVPR 2015 tutorial)
- Diapositives de Princeton Vision Group
- Diapositives de Stanford Computer Vision Lab
- Intel / TAU Summer Workshop on Deep Learning and Caffe (de Boris Ginzburg)
- NVIDIA: Deep Learning for Computer Vision with Caffe and cuDNN

Documentation

- http://caffe.berkeleyvision.org/tutorial
- http://tutorial.caffe.berkeleyvision.org (CVPR 2015 tutorial)
- Diapositives de Princeton Vision Group
- Diapositives de Stanford Computer Vision Lab
- Intel / TAU Summer Workshop on Deep Learning and Caffe (de Boris Ginzburg)
- NVIDIA: Deep Learning for Computer Vision with Caffe and cuDNN

Documentation

- http://caffe.berkeleyvision.org/tutorial
- http://tutorial.caffe.berkeleyvision.org (CVPR 2015 tutorial)
- Diapositives de Princeton Vision Group
- Diapositives de Stanford Computer Vision Lab
- Intel / TAU Summer Workshop on Deep Learning and Caffe (de Boris Ginzburg)
- NVIDIA: Deep Learning for Computer Vision with Caffe and cuDNN

Documentation

- http://caffe.berkeleyvision.org/tutorial
- http://tutorial.caffe.berkeleyvision.org (CVPR 2015 tutorial)
- Diapositives de Princeton Vision Group
- Diapositives de Stanford Computer Vision Lab
- Intel / TAU Summer Workshop on Deep Learning and Caffe (de Boris Ginzburg)
- NVIDIA: Deep Learning for Computer Vision with Caffe and cuDNN

Documentation

- http://caffe.berkeleyvision.org/tutorial
- http://tutorial.caffe.berkeleyvision.org (CVPR 2015 tutorial)
- Diapositives de Princeton Vision Group
- Diapositives de Stanford Computer Vision Lab
- Intel / TAU Summer Workshop on Deep Learning and Caffe (de Boris Ginzburg)
- NVIDIA: Deep Learning for Computer Vision with Caffe and cuDNN

Outline

Installation

L'installation sous Linux est la plus simple.

- http://caffe.berkeleyvision.org/installation.html
- Guide d'installation pour Ubuntu 16.04

Dépendances :

- CUDA (optionnel)
- OpenCV
- BLAS (Basic Linear Algebra Subprograms) (opérations de matrices)
- Boost (pour fonctions de math et shared_ptr)
- glog, gflags (gestion de traces et de drapeaux)
- leveldb, lmdb, hdf5 (bases de données)
- protobuf (gestion de fichiers de configuration)
- python, scipy, numpy (interface "utilisateur")

Nous allons utiliser une machine virtuelle pré-installée.

Installation

L'installation sous Linux est la plus simple.

- http://caffe.berkeleyvision.org/installation.html
  - Guide d'installation pour Ubuntu 16.04

Dépendances :

- CUDA (optionnel)
- OpenCV
- BLAS (Basic Linear Algebra Subprograms) (opérations de matrices)
- Boost (pour fonctions de math et shared_ptr)
- glog, gflags (gestion de traces et de drapeaux)
- leveldb, lmdb, hdf5 (bases de données)
- protobuf (gestion de fichiers de configuration)
- python, scipy, numpy (interface "utilisateur")

Nous allons utiliser une machine virtuelle pré-installée.

Installation

L'installation sous Linux est la plus simple.

- http://caffe.berkeleyvision.org/installation.html
- Guide d'installation pour Ubuntu 16.04

Dépendances :

- CUDA (optionnel)
- OpenCV
- BLAS (Basic Linear Algebra Subprograms) (opérations de matrices)
- Boost (pour fonctions de math et shared_ptr)
- glog, gflags (gestion de traces et de drapeaux)
- leveldb, lmdb, hdf5 (bases de données)
- protobuf (gestion de fichiers de configuration)
- python, scipy, numpy (interface "utilisateur")

Nous allons utiliser une machine virtuelle pré-installée.

Installation

L'installation sous Linux est la plus simple.

- http://caffe.berkeleyvision.org/installation.html
- Guide d'installation pour Ubuntu 16.04

Dépendances :

- CUDA (optionnel)
- OpenCV
- BLAS (Basic Linear Algebra Subprograms) (opérations de matrices)
- Boost (pour fonctions de math et shared_ptr)
- glog, gflags (gestion de traces et de drapeaux)
- leveldb, lmdb, hdf5 (bases de données)
- protobuf (gestion de fichiers de configuration)
- python, scipy, numpy (interface "utilisateur")

Nous allons utiliser une machine virtuelle pré-installée.

Installation

L'installation sous Linux est la plus simple.

- http://caffe.berkeleyvision.org/installation.html
- Guide d'installation pour Ubuntu 16.04

Dépendances :

- CUDA (optionnel)
- OpenCV
- BLAS (Basic Linear Algebra Subprograms) (opérations de matrices)
- Boost (pour fonctions de math et shared_ptr)
- glog, gflags (gestion de traces et de drapeaux)
- leveldb, lmdb, hdf5 (bases de données)
- protobuf (gestion de fichiers de configuration)
- python, scipy, numpy (interface "utilisateur")

Nous allons utiliser une machine virtuelle pré-installée.

Installation

L'installation sous Linux est la plus simple.

- http://caffe.berkeleyvision.org/installation.html
- Guide d'installation pour Ubuntu 16.04

Dépendances :

- CUDA (optionnel)
- OpenCV
- BLAS (Basic Linear Algebra Subprograms) (opérations de matrices)
- Boost (pour fonctions de math et shared_ptr)
- glog, gflags (gestion de traces et de drapeaux)
- leveldb, lmdb, hdf5 (bases de données)
- protobuf (gestion de fichiers de configuration)
- python, scipy, numpy (interface "utilisateur")

Nous allons utiliser une machine virtuelle pré-installée.

Outline

Utilisation

- caffe s'utilise au travers d'une interface.
- Il y a trois options :
  1. la ligne de commande,
  2. python (pycaffe) (conseillé),
  3. Matlab (matcaffe)
- http://caffe.berkeleyvision.org/tutorial/interfaces.html.

Utilisation

- caffe s'utilise au travers d'une interface.
- Il y a trois options :
  1. la ligne de commande,
  2. python (pycaffe) (conseillé),
  3. Matlab (matcaffe).
- http://caffe.berkeleyvision.org/tutorial/interfaces.html.

Utilisation

- caffe s'utilise au travers d'une interface.
- Il y a trois options :
  1. la ligne de commande,
  2. python (pycaffe) (conseillé),
  3. Matlab (matcaffe).
- http://caffe.berkeleyvision.org/tutorial/interfaces.html.

Utilisation

- caffe s'utilise au travers d'une interface.
- Il y a trois options :
  1. la ligne de commande,
  2. python (pycaffe) (conseillé),
  3. Matlab (matcaffe).
- http://caffe.berkeleyvision.org/tutorial/interfaces.html.

Utilisation

- caffe s'utilise au travers d'une interface.
- Il y a trois options :
  1. la ligne de commande,
  2. python (pycaffe) (conseillé),
  3. Matlab (matcaffe).
- http://caffe.berkeleyvision.org/tutorial/interfaces.html.

Utilisation

- caffe s'utilise au travers d'une interface.
- Il y a trois options :
  1. la ligne de commande,
  2. python (pycaffe) (conseillé),
  3. Matlab (matcaffe).
- http://caffe.berkeleyvision.org/tutorial/interfaces.html.

Outline

Préparation de données

- Conversion en un format qui peut être traité par caffe. Deux options :
  1. LMDB (Symas Lightning Memory-Mapped Database) : http://symas.com/mdb/ (choix par défaut de caffe),
  2. LevelDB (de Google) : http://leveldb.org/.
- Bases de données de forme : clé – valeur, très performantes, conçu pour des grandes masses de données.
- Autre possibilité : HDF5 (plutôt pour des étiquettes, petits volumes)

Préparation de données

- Conversion en un format qui peut être traité par caffe. Deux options :
  1. LMDB (Symas Lightning Memory-Mapped Database) : http://symas.com/mdb/ (choix par défaut de caffe),
  2. LevelDB (de Google) : http://leveldb.org/.
- Bases de données de forme : clé – valeur, très performantes, conçu pour des grandes masses de données.
- Autre possibilité : HDF5 (plutôt pour des étiquettes, petits volumes)

Préparation de données

- Conversion en un format qui peut être traité par caffe. Deux options :
  1. LMDB (Symas Lightning Memory-Mapped Database) : http://symas.com/mdb/ (choix par défaut de caffe),
  2. LevelDB (de Google) : http://leveldb.org/.
- Bases de données de forme : clé – valeur, très performantes, conçu pour des grandes masses de données.
- Autre possibilité : HDF5 (plutôt pour des étiquettes, petits volumes)

Préparation de données

- Conversion en un format qui peut être traité par caffe. Deux options :
  1. LMDB (Symas Lightning Memory-Mapped Database) : http://symas.com/mdb/ (choix par défaut de caffe),
  2. LevelDB (de Google) : http://leveldb.org/.
- Bases de données de forme : clé – valeur, très performantes, conçu pour des grandes masses de données.
- Autre possibilité : HDF5 (plutôt pour des étiquettes, petits volumes)

Préparation de données

- Conversion en un format qui peut être traité par caffe. Deux options :
  1. LMDB (Symas Lightning Memory-Mapped Database) : http://symas.com/mdb/ (choix par défaut de caffe),
  2. LevelDB (de Google) : http://leveldb.org/.
- Bases de données de forme : clé – valeur, très performantes, conçu pour des grandes masses de données.
- Autre possibilité : HDF5 (plutôt pour des étiquettes, petits volumes)

Préparation de données (images)

- Pour convertir des images : `convert_imageset` (dans le répertoire "build/tools" de caffe).
- En ligne de commande :

```
convert_imageset --shuffle images/ images.txt train_lmdb
```

  - *images :* répertoire avec les images
  - *images.txt :*

```
chat1.png 0
chat2.png 0
chien1.png 1
chien2.png 1
...
```

  - *train_lmdb :* répertoire de sortie
- créer deux ensembles : apprentissage et test

Préparation de données (images)

- Pour convertir des images : `convert_imageset` (dans le répertoire "build/tools" de caffe).
- En ligne de commande :

```
convert_imageset --shuffle images/ images.txt train_lmdb
```

  - *images :* répertoire avec les images
  - *images.txt :*

```
chat1.png 0
chat2.png 0
chien1.png 1
chien2.png 1
...
```

  - *train_lmdb :* répertoire de sortie

- créer deux ensembles : apprentissage et test

Préparation de données (images)

- Pour convertir des images : `convert_imageset` (dans le répertoire "build/tools" de caffe).
- En ligne de commande :

```
convert_imageset --shuffle images/ images.txt train_lmdb
```

  - *images :* répertoire avec les images
  - *images.txt :*

```
chat1.png 0
chat2.png 0
chien1.png 1
chien2.png 1
...
```

  - *train_lmdb :* répertoire de sortie

- créer deux ensembles : apprentissage et test

Préparation de données (autres)

Pour d'autres types de données :

- écrire un programme de conversion (modifier convert_imageset.cpp)
- HDF5

Préparation de données (autres)

Pour d'autres types de données :

- écrire un programme de conversion (modifier `convert_imageset.cpp`),
- HDF5
  - seulement float32 et float64
  - pas de pré-chargement parallèle de données

Préparation de données (autres)

Pour d'autres types de données :

- écrire un programme de conversion (modifier `convert_imageset.cpp`),
- HDF5
  - seulement float32 et float64
  - pas de pré-chargement parallèle de données

Préparation de données (autres)

Pour d'autres types de données :

- écrire un programme de conversion (modifier `convert_imageset.cpp`),
- HDF5
  - seulement float32 et float64
  - pas de pré-chargement parallèle de données

Préparation de données (autres)

Pour d'autres types de données :

- écrire un programme de conversion (modifier `convert_imageset.cpp`),
- HDF5
  - seulement float32 et float64
  - pas de pré-chargement parallèle de données

Outline

Protocol buffers

- format de données structurées (comme XML, JSON)
- utilisé pour stocker des définitions d'architecture de réseaux de neurones, le "solver" et les modèles appris
- développé par Google
- on définit des **type de messages** dans un fichier `.proto`
- on définit des **messages** dans un fichier `.prototxt` ou `.binaryproto`
- tous les type de messages de caffe sont définit dans le fichier caffe.proto

Protocol buffers

- format de données structurées (comme XML, JSON)
- utilisé pour stocker des définitions d'architecture de réseaux de neurones, le "solver" et les modèles appris
- développé par Google
- on définit des **type de messages** dans un fichier `.proto`
- on définit des **messages** dans un fichier `.prototxt` ou `.binaryproto`
- tous les type de messages de caffe sont définit dans le fichier caffe.proto

Protocol buffers

- format de données structurées (comme XML, JSON)
- utilisé pour stocker des définitions d'architecture de réseaux de neurones, le "solver" et les modèles appris
- développé par Google
- on définit des **type de messages** dans un fichier `.proto`
- on définit des **messages** dans un fichier `.prototxt` ou `.binaryproto`
- tous les type de messages de caffe sont définit dans le fichier caffe.proto

Protocol buffers

- format de données structurées (comme XML, JSON)
- utilisé pour stocker des définitions d'architecture de réseaux de neurones, le "solver" et les modèles appris
- développé par Google
- on définit des **type de messages** dans un fichier `.proto`
- on définit des **messages** dans un fichier `.prototxt` ou `.binaryproto`
- tous les type de messages de caffe sont définit dans le fichier caffe.proto

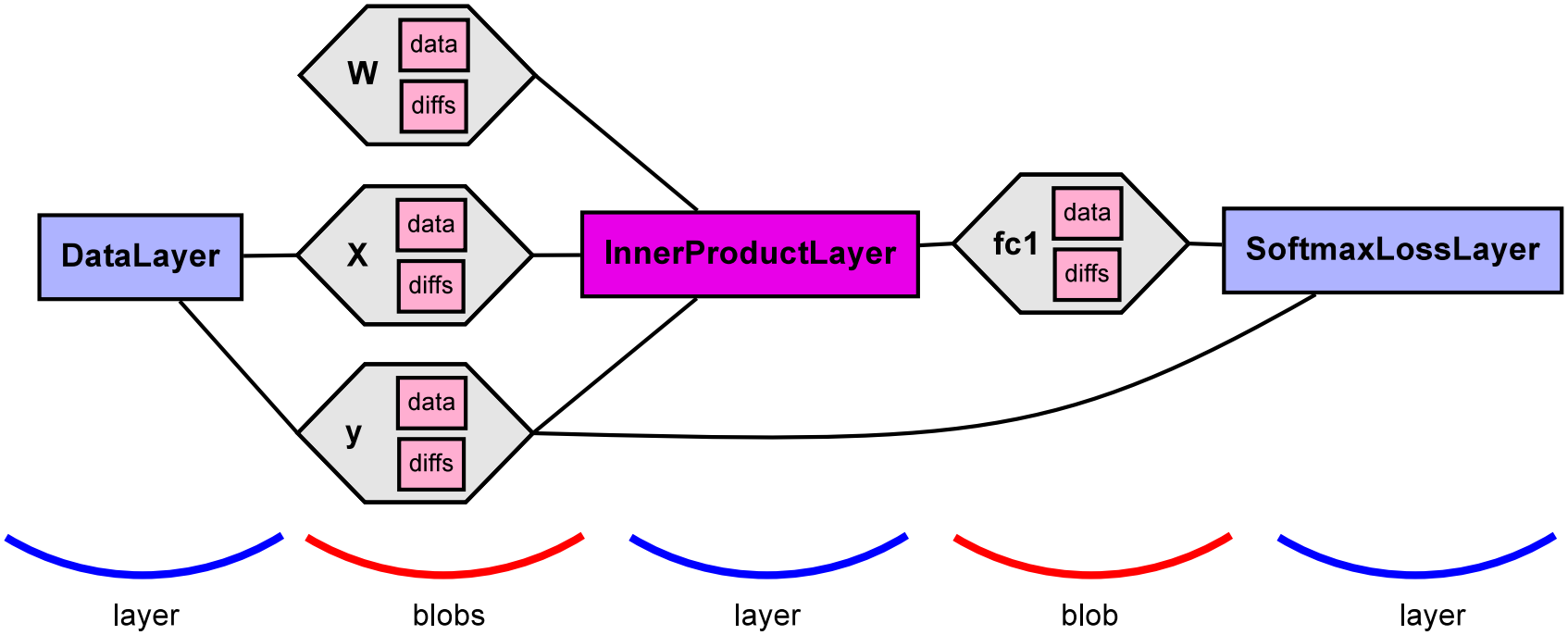
Protocol buffers

- format de données structurées (comme XML, JSON)
- utilisé pour stocker des définitions d'architecture de réseaux de neurones, le "solver" et les modèles appris
- développé par Google
- on définit des **type de messages** dans un fichier `.proto`
- on définit des **messages** dans un fichier `.prototxt` ou `.binaryproto`
- tous les type de messages de caffe sont définit dans le fichier caffe.proto

Protocol buffers

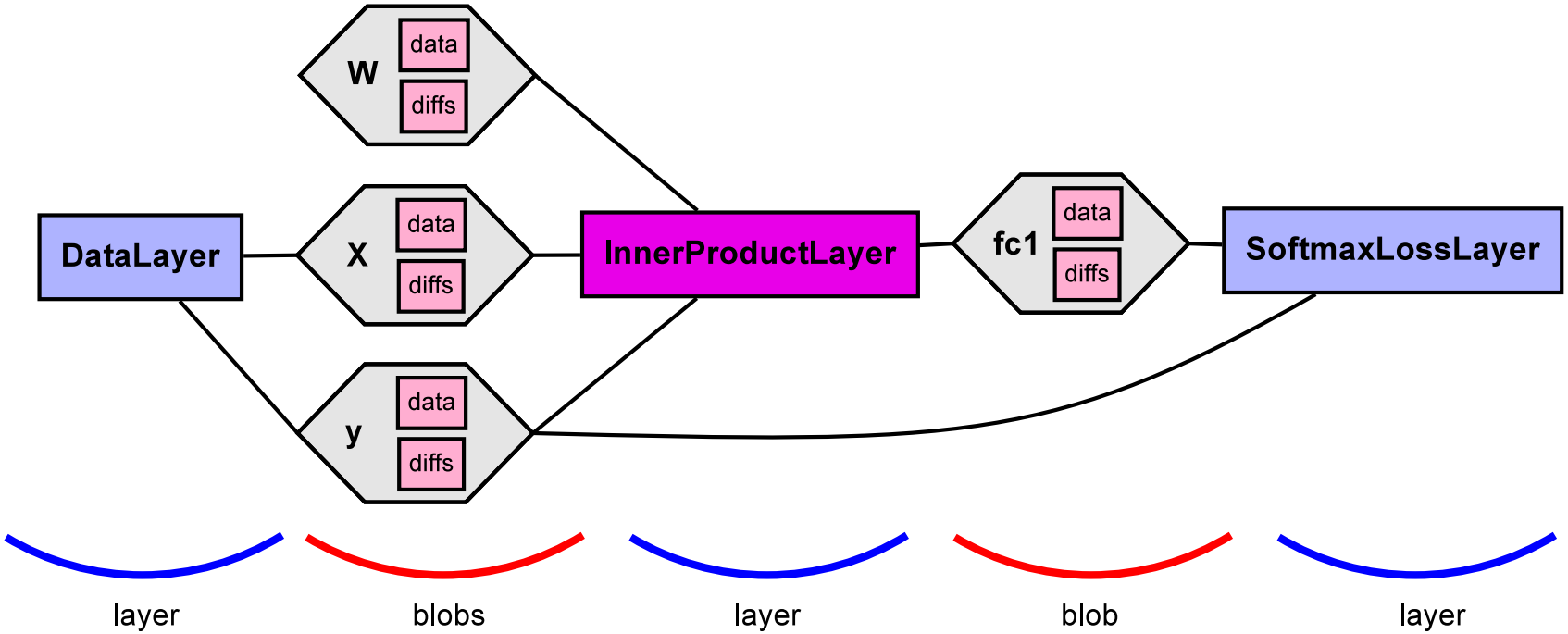
- format de données structurées (comme XML, JSON)
- utilisé pour stocker des définitions d'architecture de réseaux de neurones, le "solver" et les modèles appris
- développé par Google
- on définit des **type de messages** dans un fichier `.proto`
- on définit des **messages** dans un fichier `.prototxt` ou `.binaryproto`
- tous les type de messages de caffe sont définit dans le fichier caffe.proto

Architecture et fonctionnement de caffe



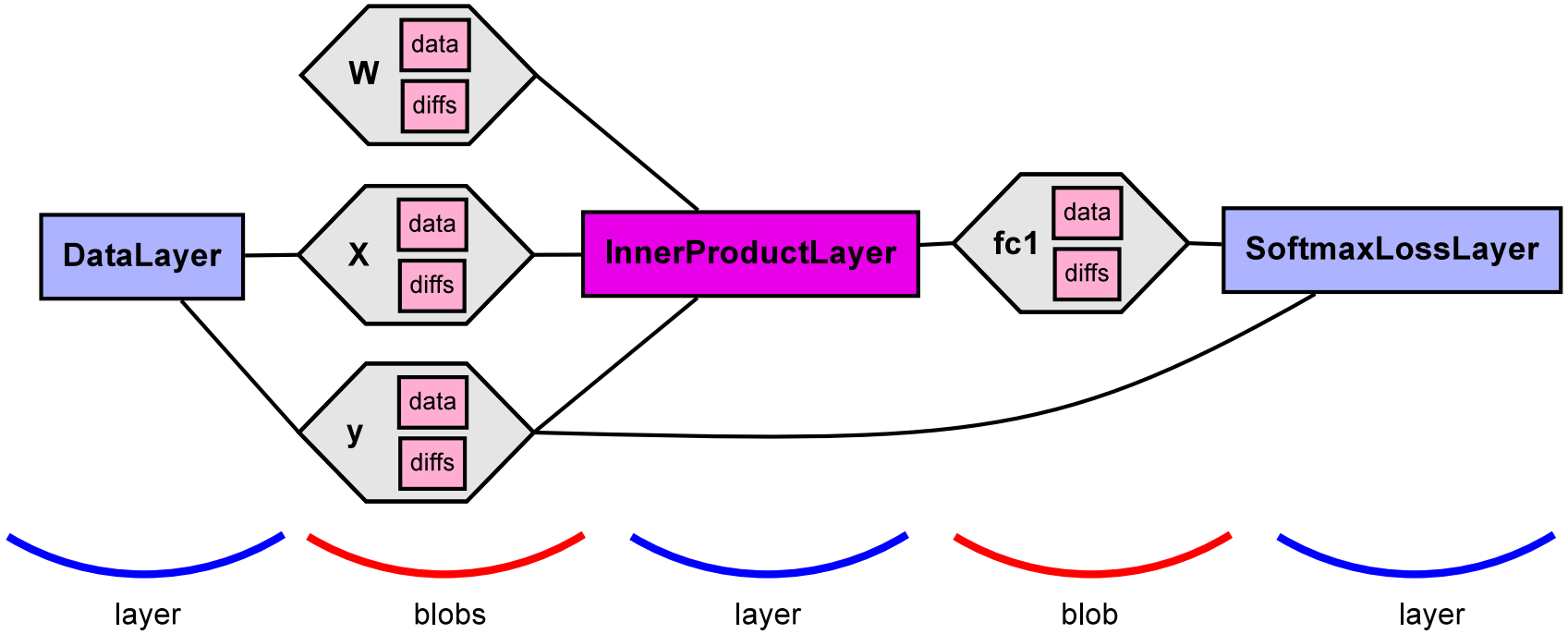
- **Layer** : couche avec un traitement/calcul (paramètres à définir dans le fichier .prototxt)
- **Blob** : structure interne à caffe; contient des données (entrées et dérivées)
- dimensions des données toujours : num_images x channels x height x width

Architecture et fonctionnement de caffe



- **Layer** : couche avec un traitement/calcul (paramètres à définir dans le fichier .prototxt)
- **Blob** : structure interne à caffe; contient des données (entrées et dérivées)
- dimensions des données toujours : num_images x channels x height x width

Architecture et fonctionnement de caffe



- **Layer** : couche avec un traitement/calcul (paramètres à définir dans le fichier .prototxt)
- **Blob** : structure interne à caffe; contient des données (entrées et dérivées)
- dimensions des données toujours : num_images x channels x height x width

Les couches courantes de caffe

- **Data** : charge des données (images) et étiquettes d'une base de données
- **InnerProduct** : couche de neurones (calcul le produit scalaire avec $\mathbf{W}$)
- **Convolution** : couche de convolution
- **Pooling** : couche de pooling (max, average)
- **SoftmaxWithLoss** : couche de sortie qui implémente une fonction de coût softmax
- **Accuracy** : couche de sortie qui calcule la précision de classification

Les couches courantes de caffe

- **Data** : charge des données (images) et étiquettes d'une base de données
- **InnerProduct** : couche de neurones (calcul le produit scalaire avec $\mathbf{W}$)
- **Convolution** : couche de convolution
- **Pooling** : couche de pooling (max, average)
- **SoftmaxWithLoss** : couche de sortie qui implémente une fonction de coût softmax
- **Accuracy** : couche de sortie qui calcule la précision de classification

Les couches courantes de caffe

- **Data** : charge des données (images) et étiquettes d'une base de données
- **InnerProduct** : couche de neurones (calcul le produit scalaire avec $\mathbf{W}$)
- **Convolution** : couche de convolution
- **Pooling** : couche de pooling (max, average)
- **SoftmaxWithLoss** : couche de sortie qui implémente une fonction de coût softmax
- **Accuracy** : couche de sortie qui calcule la précision de classification

Les couches courantes de caffe

- **Data** : charge des données (images) et étiquettes d'une base de données
- **InnerProduct** : couche de neurones (calcul le produit scalaire avec $\mathbf{W}$)
- **Convolution** : couche de convolution
- **Pooling** : couche de pooling (max, average)
- **SoftmaxWithLoss** : couche de sortie qui implémente une fonction de coût softmax
- **Accuracy** : couche de sortie qui calcule la précision de classification

Les couches courantes de caffe

- **Data** : charge des données (images) et étiquettes d'une base de données
- **InnerProduct** : couche de neurones (calcul le produit scalaire avec $\mathbf{W}$)
- **Convolution** : couche de convolution
- **Pooling** : couche de pooling (max, average)
- **SoftmaxWithLoss** : couche de sortie qui implémente une fonction de coût softmax
- **Accuracy** : couche de sortie qui calcule la précision de classification

Les couches courantes de caffe

- **Data** : charge des données (images) et étiquettes d'une base de données
- **InnerProduct** : couche de neurones (calcul le produit scalaire avec $\mathbf{W}$)
- **Convolution** : couche de convolution
- **Pooling** : couche de pooling (max, average)
- **SoftmaxWithLoss** : couche de sortie qui implémente une fonction de coût softmax
- **Accuracy** : couche de sortie qui calcule la précision de classification

Définition d'une couche Data

```
name: "FaceNet"
layer {
  name: "facedata"
  type: "Data"
  top: "data"
  top: "label"
  include {
    phase: TRAIN
  }
  data_param {
    source: "facedb_train"
    batch_size: 16
    backend: LMDB
  }
}
```

Définition d'une couche Convolution

```
layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  convolution_param {
    num_output: 5
    kernel_size: 5
    stride: 1
    weight_filler {
      type: "gaussian"
      std: 0.01
    }
    bias_filler {
      type: "constant"
      value: 0.01
    }
  }
}
```

Définition d'une couche Pooling

```
layer {
  name: "pool1"
  type: "Pooling"
  bottom: "conv1"
  top: "pool1"
  pooling_param {
    pool: AVE
    kernel_size: 2
    stride: 2
  }
}
```

Définition d'une couche InnerProduct

```
layer {
  name: "ip1"
  type: "InnerProduct"
  bottom: "pool1"
  top: "ip1"
  inner_product_param {
    num_output: 20
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
      value: 0.01
    }
  }
}
```

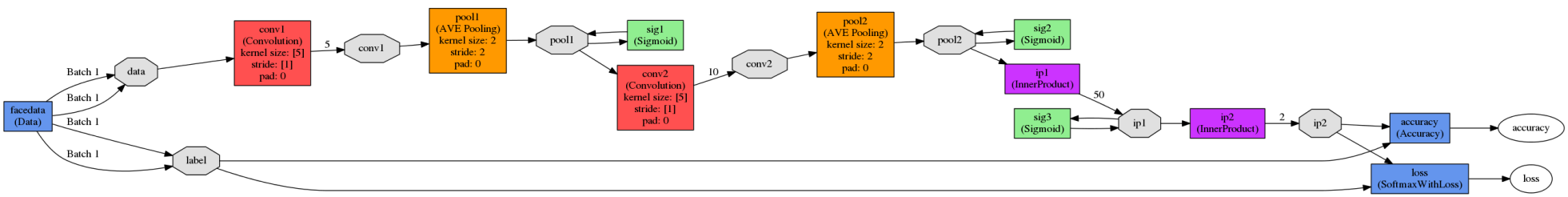
Définition d'une couche Softmax

```
layer {
  name: "loss"
  type: "SoftmaxWithLoss"
  bottom: "ip1"
  bottom: "label"
  top: "loss"
}
```

Visualisation de l'architecture

- on peut visualiser le fichier `.prototxt` avec le script **draw_net.py** dans le dossier "python" de caffe :

```
python draw_net.py model.prototxt output.png
```



Outline

Définition du "solver"

```
net: "model.prototxt"
test_iter: 1
test_interval: 500
base_lr: 0.01
momentum: 0.9
weight_decay: 0.0001
lr_policy: "fixed"
#gamma: 0.0001
#power: 0.75
display: 100
max_iter: 200000
snapshot: 5000
snapshot_prefix: "facenet"
solver_mode: CPU
```

Outline

Apprentissage

- en ligne de commande :

```
caffe train -solver solver.prototxt
```

- programme `caffe` dans le répertoire "build/tools/"
- produit deux fichiers :
  1. fichier_apprentissage.caffemodel : le modèle appris qui peut être évalué et déployé dans une application
  2. fichier_apprentissage.solverstate : fichier d'état qui sert à continuer l'apprentissage plus tard ou avec d'autres paramètres (par exemple pour "fine-tuning")

Apprentissage

- en ligne de commande :

```
caffe train -solver solver.prototxt
```

- programme `caffe` dans le répertoire "build/tools/"
- produit deux fichiers :
  1. fichier `_iter_N.caffemodel` : le modèle appris qui peut être évalué et déployé dans une application
  2. fichier `_iter_N.solverstate` : fichier d'état qui sert à continuer l'apprentissage plus tard ou avec d'autres paramètres (par exemple pour "fine-tuning")

Apprentissage

- en ligne de commande :

```
caffe train -solver solver.prototxt
```

- programme `caffe` dans le répertoire "build/tools/"
- produit deux fichiers :
  1. fichier `.caffemodel` : le modèle appris qui peut être évalué et déployé dans une application
  2. fichier `.solverstate` : fichier d'état qui sert à continuer l'apprentissage plus tard ou avec d'autres paramètres (par exemple pour "fine-tuning").

Apprentissage

- en ligne de commande :

```
caffe train -solver solver.prototxt
```

- programme `caffe` dans le répertoire "build/tools/"
- produit deux fichiers :
  1. fichier `.caffemodel` : le modèle appris qui peut être évalué et déployé dans une application
  2. fichier `.solverstate` : fichier d'état qui sert à continuer l'apprentissage plus tard ou avec d'autres paramètres (par exemple pour "fine-tuning").

Apprentissage

- en ligne de commande :

```
caffe train -solver solver.prototxt
```

- programme `caffe` dans le répertoire "build/tools/"
- produit deux fichiers :
  1. fichier `.caffemodel` : le modèle appris qui peut être évalué et déployé dans une application
  2. fichier `.solverstate` : fichier d'état qui sert à continuer l'apprentissage plus tard ou avec d'autres paramètres (par exemple pour "fine-tuning").

Outline

Test

```
caffe test -model model_test.prototxt -weights facenet.
    caffemodel -iterations 1
```

- model_test.prototxt : version modifiée de model.prototxt
- à la place de la couche Data :

```
input: "data"
input_shape {
  dim: 1
  dim: 1
  dim: 36
  dim: 36
}
```

- à la place de la couche SoftmaxWithLoss :

```
layer {
  name: "prob"
  type: "Softmax"
  bottom: "ip2"
  top: "prob"
}
```

Test

```
caffe test -model model_test.prototxt -weights facenet.
    caffemodel -iterations 1
```

- model_test.prototxt : version modifiée de model.prototxt
- à la place de la couche Data :

```
input: "data"
input_shape {
  dim: 1
  dim: 1
  dim: 36
  dim: 36
}
```

- à la place de la couche SoftmaxWithLoss :

```
layer {
  name: "prob"
  type: "Softmax"
  bottom: "ip2"
  top: "prob"
}
```

Test

```
caffe test -model model_test.prototxt -weights facenet.
    caffemodel -iterations 1
```

- model_test.prototxt : version modifiée de model.prototxt
- à la place de la couche Data :

```
input: "data"
input_shape {
  dim: 1
  dim: 1
  dim: 36
  dim: 36
}
```

- à la place de la couche SoftmaxWithLoss :

```
layer {
  name: "prob"
  type: "Softmax"
  bottom: "ip2"
  top: "prob"
}
```

Outline

Conclusion

- pour comprendre caffe :
  - le site web officiel,
  - les forums + des exemples,
  - le code source de caffe.
- sujets non traités :
  - intégration dans un code C++ ou Python,
  - fine-tuning,
  - extraction de caractéristiques (features),
  - d'autres types de couches et paramètres spécifiques,
  - architectures siamoises,
  - extensions de caffe.

Conclusion

- pour comprendre caffe :
  - le site web officiel,
  - les forums + des exemples,
  - le code source de caffe.
- sujets non traités :
  - intégration dans un code C++ ou Python,
  - fine-tuning,
  - extraction de caractéristiques (features),
  - d'autres types de couches et paramètres (pooling, ...),
  - architectures siamoises,
  - extensions de caffe.

Conclusion

- pour comprendre caffe :
  - le site web officiel,
  - les forums + des exemples,
  - le code source de caffe.
- sujets non traités :
  - intégration dans un code C++ ou Python,
  - fine-tuning,
  - extraction de caractéristiques (features),
  - d'autres types de couches et paramètres (pooling, ...),
  - architectures siamoises,
  - extensions de caffe.

Conclusion

- pour comprendre caffe :
  - le site web officiel,
  - les forums + des exemples,
  - le code source de caffe.
- sujets non traités :
  - intégration dans un code C++ ou Python,
  - fine-tuning,
  - extraction de caractéristiques (features),
  - d'autres types de couches et paramètres (pooling, ...),
  - architectures siamoises,
  - extensions de caffe.

Conclusion

- pour comprendre caffe :
  - le site web officiel,
  - les forums + des exemples,
  - le code source de caffe.
- sujets non traités :
  - intégration dans un code C++ ou Python
  - fine-tuning,
  - extraction de caractéristiques (features)
  - d'autres types de couches et paramètres spécifiques,
  - architectures siamoises,
  - extensions de caffe.

Conclusion

- pour comprendre caffe :
  - le site web officiel,
  - les forums + des exemples,
  - le code source de caffe.
- sujets non traités :
  - intégration dans un code C++ ou Python
  - fine-tuning,
  - extraction de caractéristiques (features)
  - d'autres types de couches et paramètres spécifiques,
  - architectures siamoises,
  - extensions de caffe.

Conclusion

- pour comprendre caffe :
  - le site web officiel,
  - les forums + des exemples,
  - le code source de caffe.
- sujets non traités :
  - intégration dans un code C++ ou Python
  - fine-tuning,
  - extraction de caractéristiques (features)
  - d'autres types de couches et paramètres spécifiques,
  - architectures siamoises,
  - extensions de caffe.

Conclusion

- pour comprendre caffe :
  - le site web officiel,
  - les forums + des exemples,
  - le code source de caffe.
- sujets non traités :
  - intégration dans un code C++ ou Python
  - fine-tuning,
  - extraction de caractéristiques (features)
  - d'autres types de couches et paramètres spécifiques,
  - architectures siamoises,
  - extensions de caffe.

Conclusion

- pour comprendre caffe :
  - le site web officiel,
  - les forums + des exemples,
  - le code source de caffe.
- sujets non traités :
  - intégration dans un code C++ ou Python
  - fine-tuning,
  - extraction de caractéristiques (features)
  - d'autres types de couches et paramètres spécifiques,
  - architectures siamoises,
  - extensions de caffe.

Conclusion

- pour comprendre caffe :
  - le site web officiel,
  - les forums + des exemples,
  - le code source de caffe.
- sujets non traités :
  - intégration dans un code C++ ou Python
  - fine-tuning,
  - extraction de caractéristiques (features)
  - d'autres types de couches et paramètres spécifiques,
  - architectures siamoises,
  - extensions de caffe.

Conclusion

- pour comprendre caffe :
  - le site web officiel,
  - les forums + des exemples,
  - le code source de caffe.
- sujets non traités :
  - intégration dans un code C++ ou Python
  - fine-tuning,
  - extraction de caractéristiques (features)
  - d'autres types de couches et paramètres spécifiques,
  - architectures siamoises,
  - extensions de caffe.