Hands on

Deep Learning

VVV17 Winter School on Humanoid Robot Programming

Santa Margherita Ligure

Feb. 7 2017

Outline

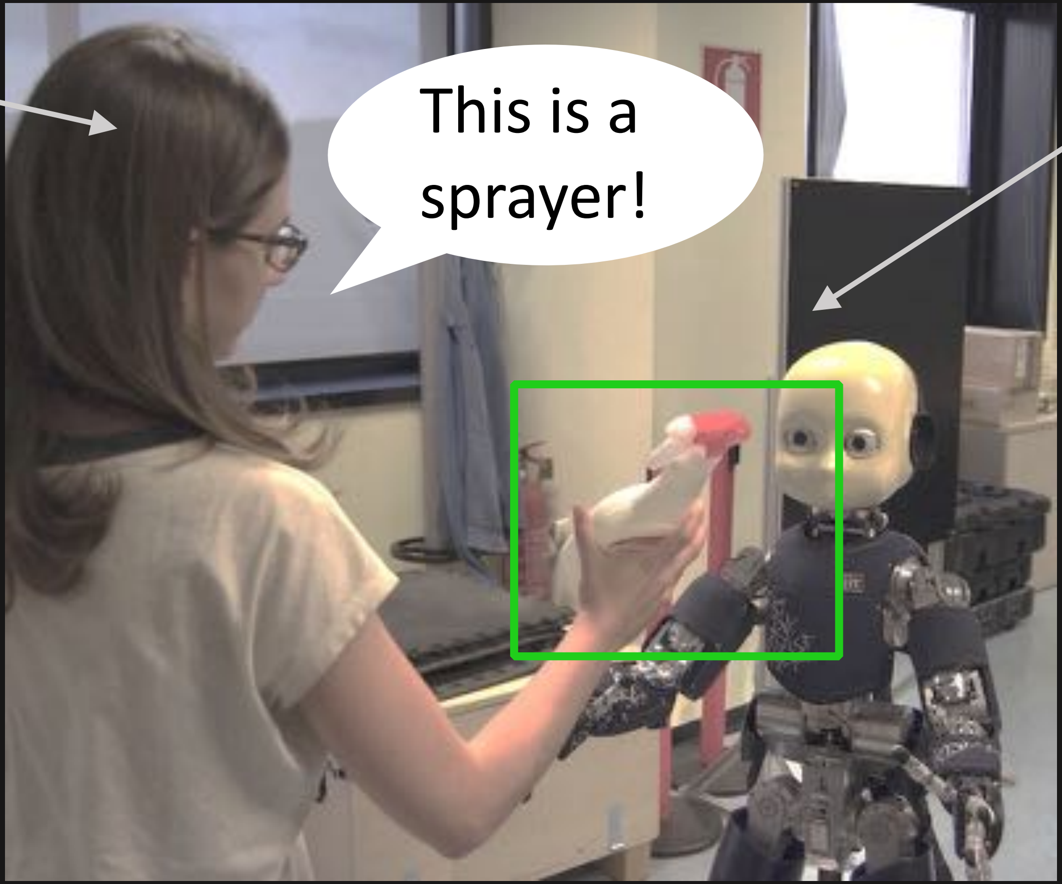
1. The iCubWorld Project

2. Deep Learning using Caffe

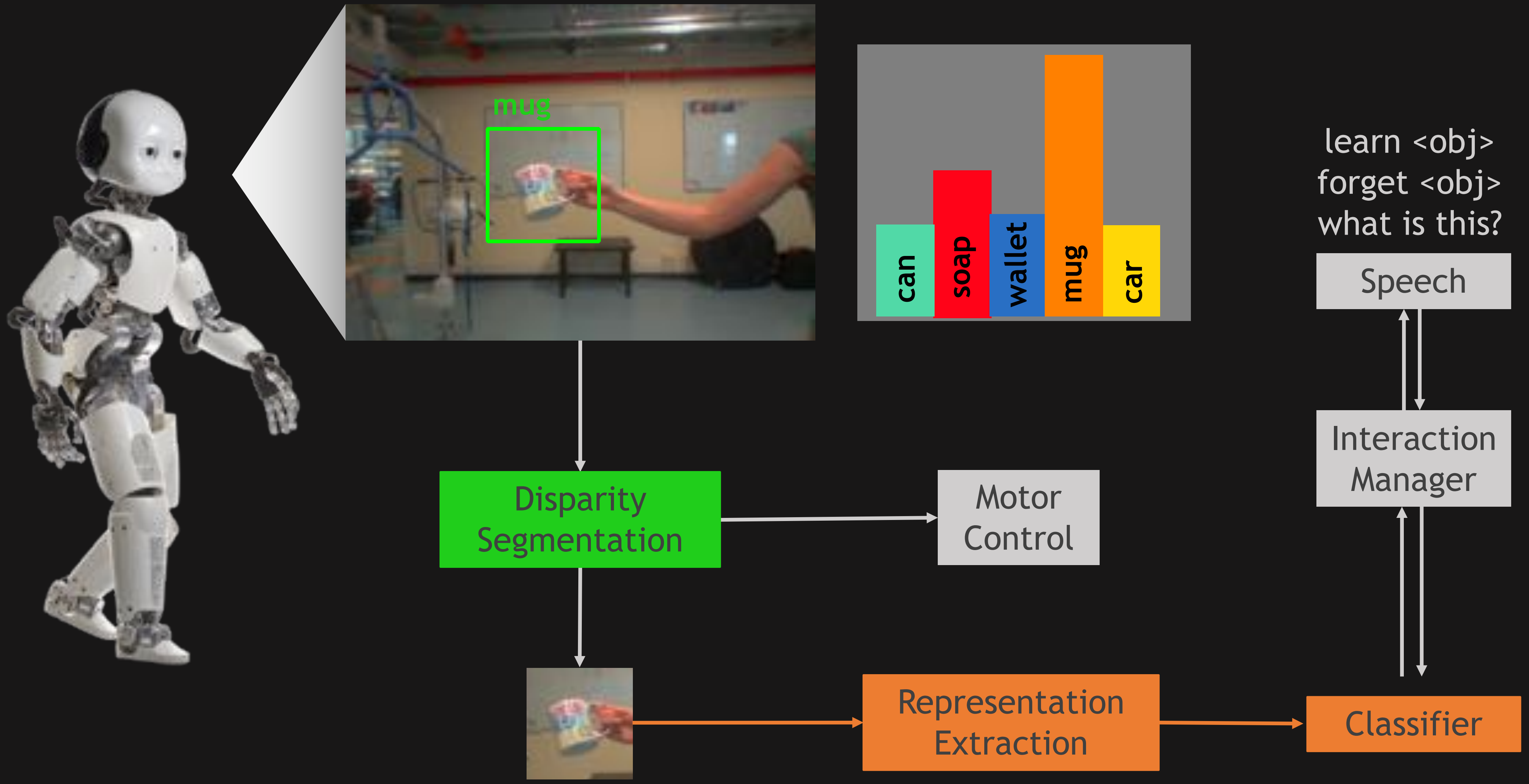
Teaching iCub to See: HRI Framework

Verbal instructions of a “teacher”

iCub learning the object



On the Fly Object Recognition



The Need for an iCubWorld Dataset

What is the Visual World of a Robot?

- not so many object/instances

http://image-net.org/

The Need for an iCubWorld Dataset

What is the Visual World of a Robot?

- not so many object/instances
- lots of viewpoint changes

The Need for an iCubWorld Dataset

What is the Visual World of a Robot?

- not so many object/instances
- lots of viewpoint changes
- …
- uninformative background
- self supervised
- …

→ Different from usual vision tasks!!

iCubWorld Datasets

https://robotology.github.io/iCubWorld/

2012-13
1st and 2nd iCW Releases

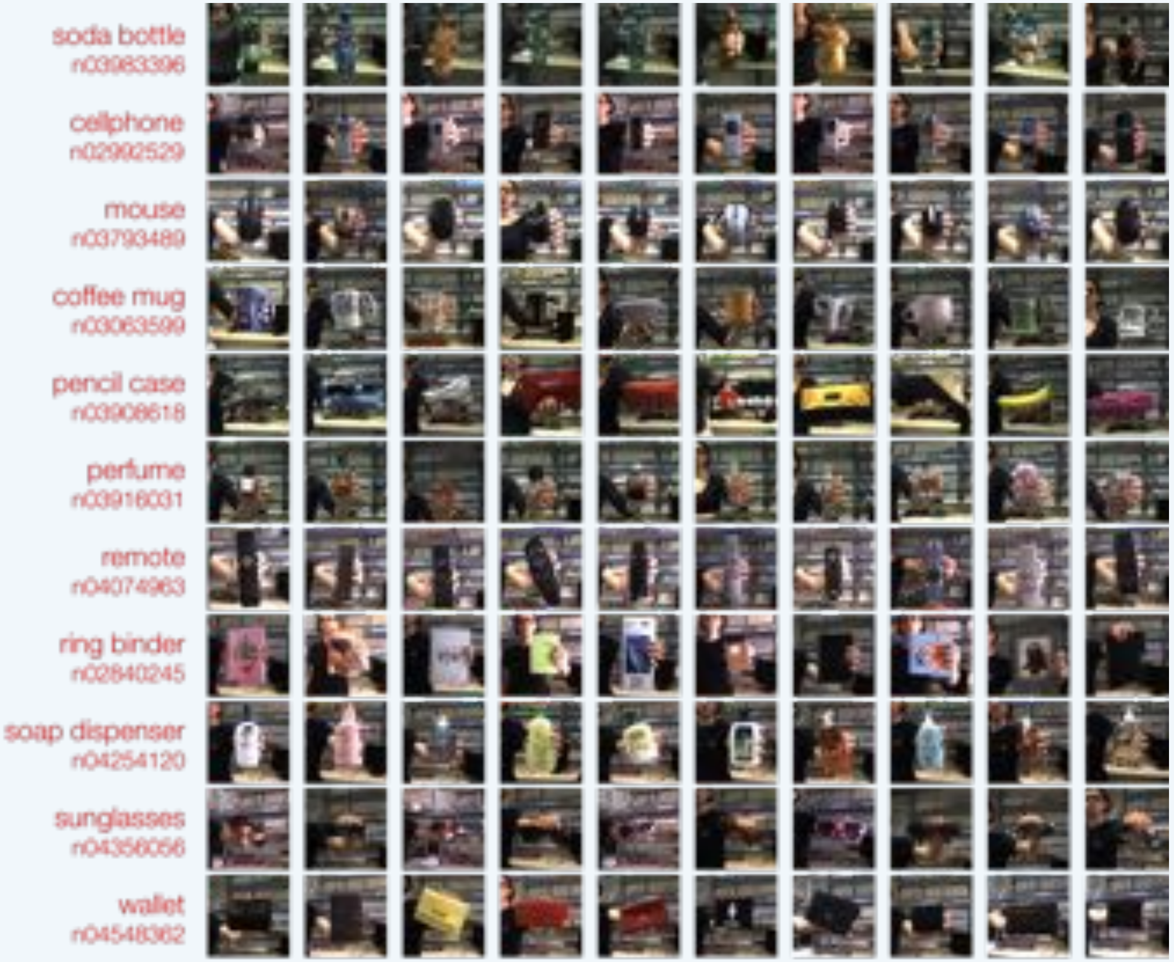
2014-15
iCubWorld28

2016
iCubWorld – Transformations

Latest Release: iCubWorld – Transformations

20 categories

10 object instances per category



Latest Release: iCubWorld – Transformations

20 categories

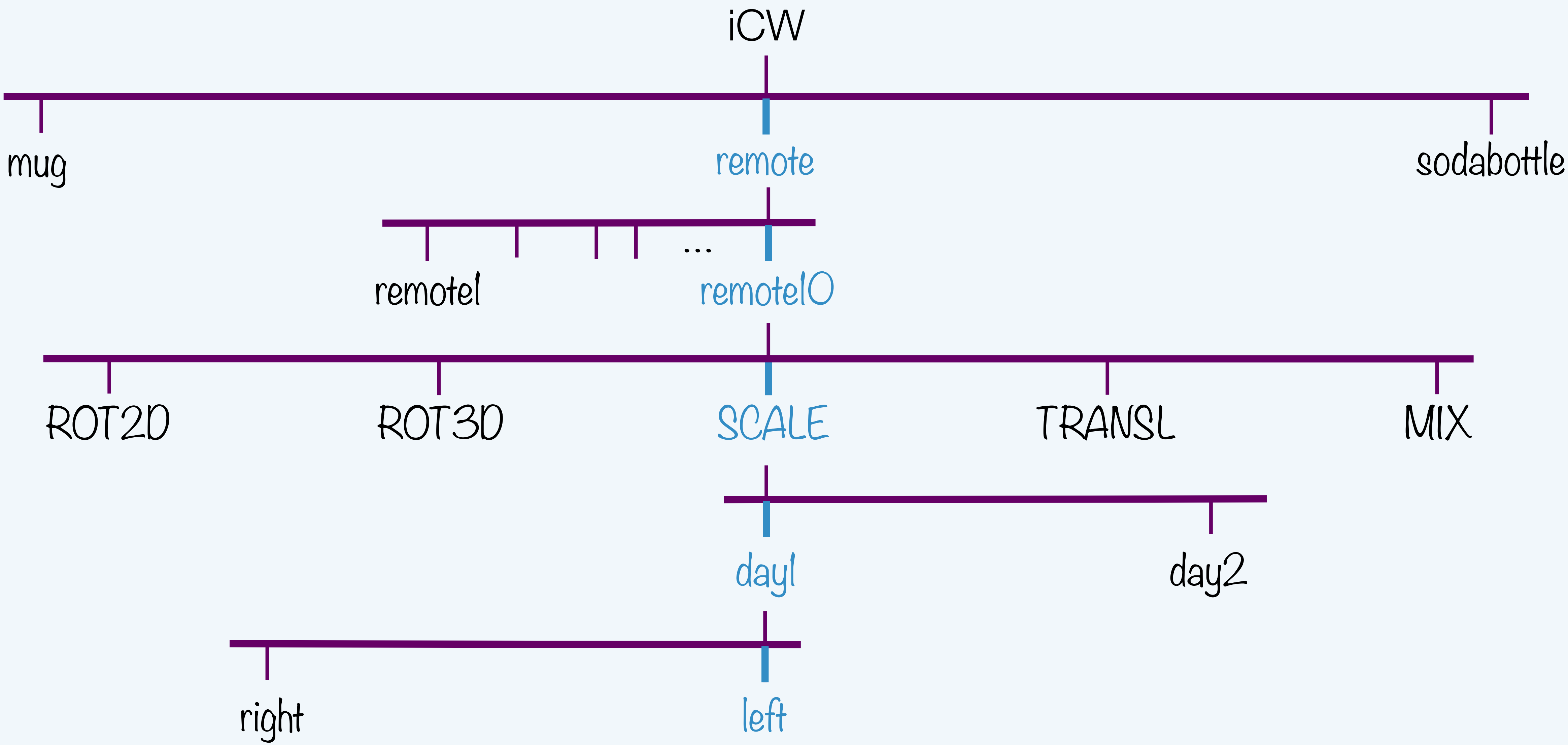
10 object instances per category

10 sequences per object:

5 transformations
2 days

stereo sequences

iCubWorld (iCW): subset for the labs



3 x 10 x 5 x 2 x 1 x (~ 150 frames) → **~ 50k images**

Outline

1. The iCubWorld Project

2. Deep Learning using Caffe

Caffe
(Convolutional Architecture for Fast Feature Embedding)

C++ framework
command-line, **Python** and **Matlab** interfaces

to define and train **(Convolutional) Neural Networks**
(on CPUs/CUDA GPUs)

Web: http://caffe.berkeleyvision.org/

GitHub: https://github.com/BVLC/caffe

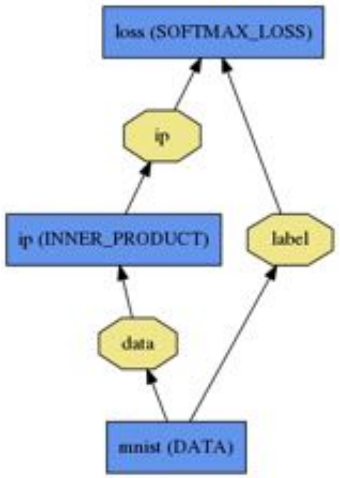
Community: Caffe Users - Google Groups

Caffe
(Convolutional Architecture for Fast Feature Embedding)

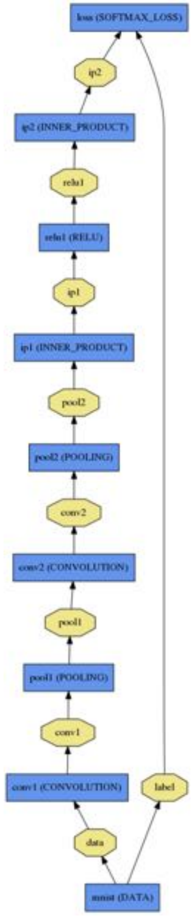
Nets

Layers

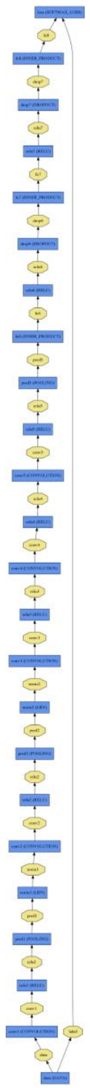
Blobs



LogReg ↑

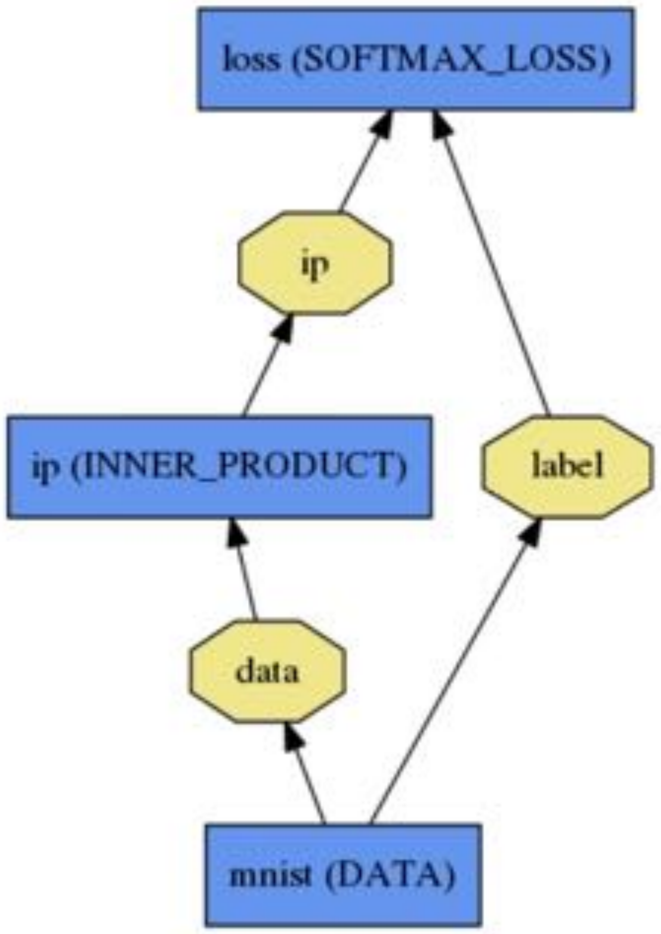


LeNet →



ImageNet, Krizhevsky 2012 →

Net

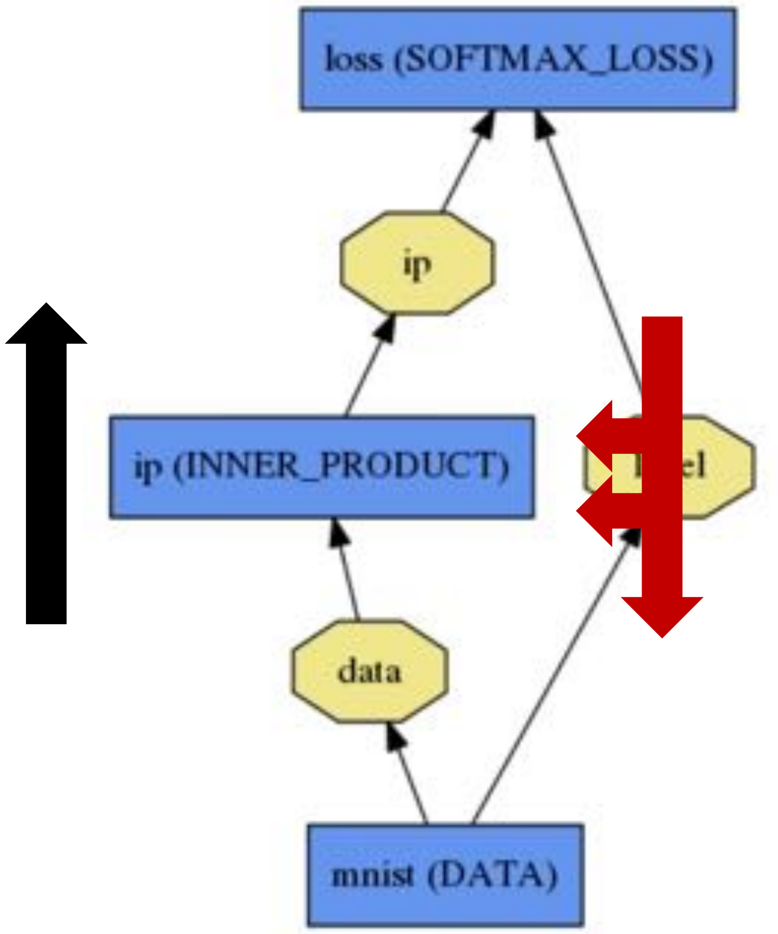


A network is a set of Layers and their connections.

Data and derivatives flow through the net as Blobs (an array interface).

Forward/Backward are the essential **Net** computations.

Layers



Defined by **setup**, **forward** and **backward** computations:

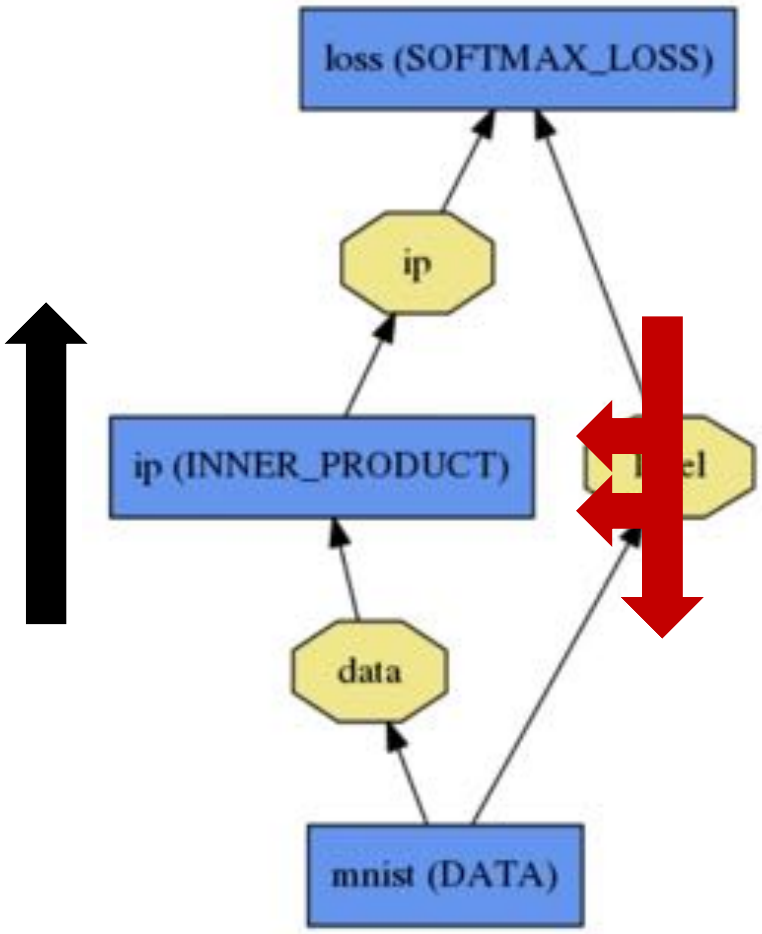
setup: run once for initialization

forward: make output given input

backward: make gradient of output
- w.r.t. bottom
- w.r.t. parameters

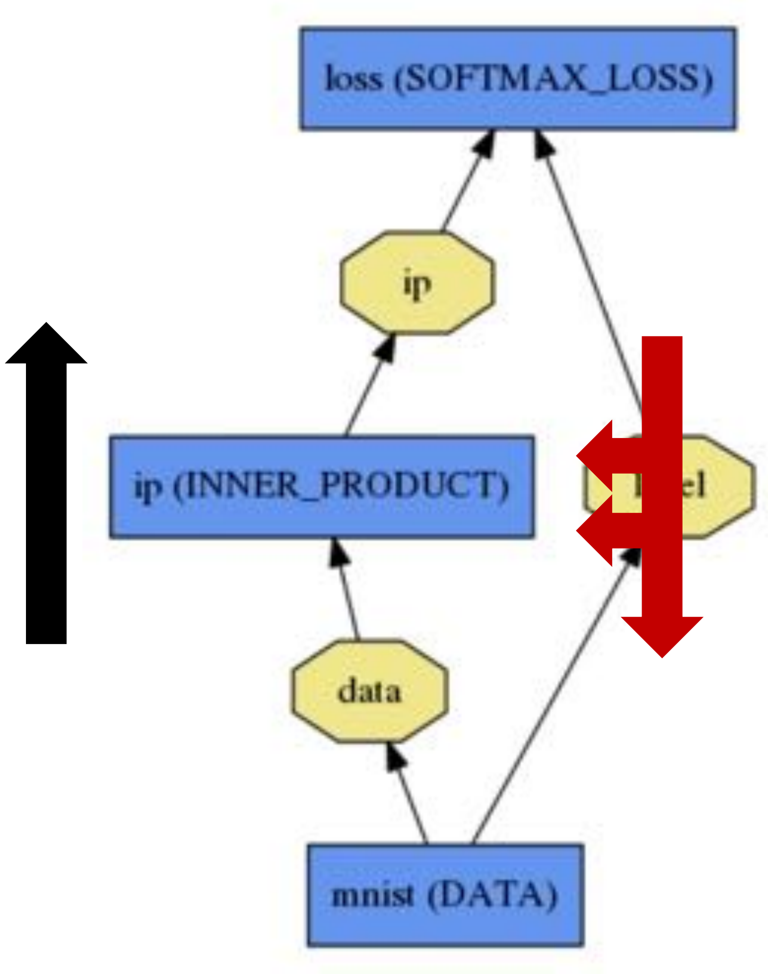
The Net's **Forward** and **Backward** passes are composed of the layers' steps (*Compositional Modeling*)

[LAYER CATALOGUE](on Caffe website)



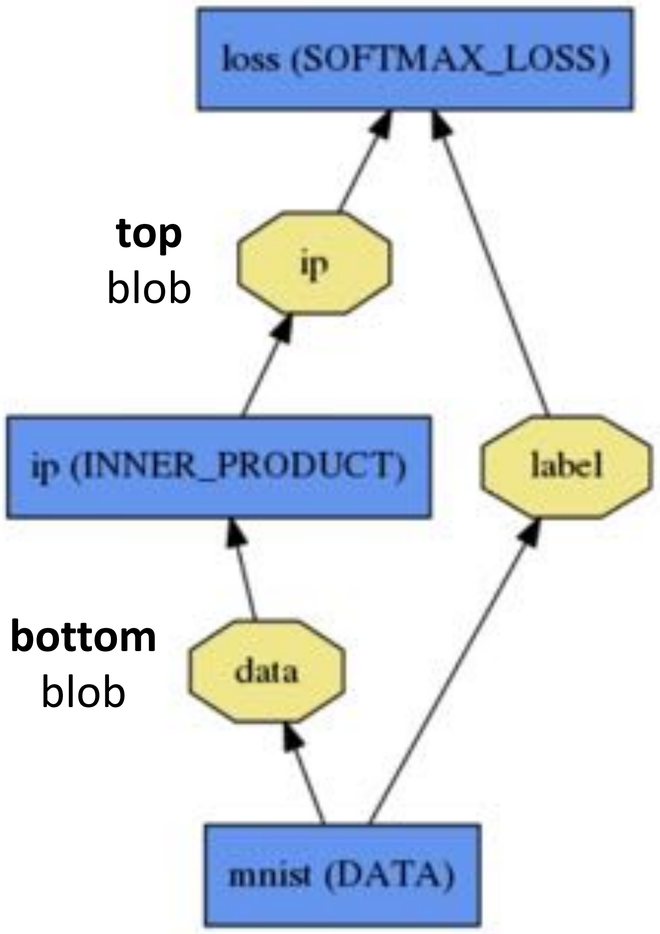
- ✓ "Convolution"
- ✓ "Pooling"
- ✓ "LRN" (normalization)
- ✓ "InnerProduct"
- ✓ "ReLU" (non linearity)
- ✓ …
- ✓ **DATA layers** (include image pre-processing steps):
  - ✓ database ("Data")
  - ✓ in-memory ("MemoryData")
  - ✓ image list ("ImageData")
- ✓ **LOSS layers**
  - ✓ softmax ("SoftmaxWithLoss")
  - ✓ accuracy ("Accuracy")

(Common) Image Preprocessing Steps



- ✓ "Convolution"
- ✓ "Pooling"
- ✓ "LRN" (normalization)
- ✓ "InnerProduct"
- ✓ "ReLU" (non linearity)
- ✓ …
- ✓ **DATA layers** (include image pre-processing steps):
  1. Subtract mean image of the training set (or mean pixel)
  2. Crop of expected size (e.g. 227x227x3)
- ✓ **LOSS layers**
  - ✓ softmax ("SoftmaxWithLoss")
  - ✓ accuracy ("Accuracy")

Blobs

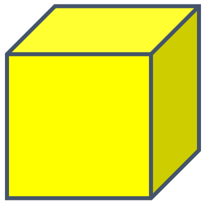


4-dimensional array stored in a C-contiguous fashion:

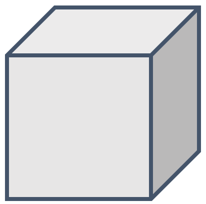
number N x channel C x height H x width W

Hides the overhead of CPU/GPU computation by synchronizing as needed and allocating memory on host/device lazily.

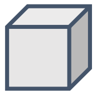
Provides a unified memory interface:



Data
*N*umber x Channel x *H*eight x *W*idth
256 x 3 x 227 x 227 for CaffeNet train input



Parameter: Convolution Weight
N Output x *K* Input x *H*eight x *W*idth
96 x 3 x 11 x 11 for CaffeNet conv1



Parameter: Convolution Bias
96 x 1 x 1 x 1 for CaffeNet conv1

Solver (Optimizer)

Initializes the model and the training.

At each iteration:

forward → output and loss
backward → gradients
parameter updates
solver state update

Periodic validation.
Periodic snapshots of the model and solver state.

Many different kinds implemented in Caffe, e.g.:

- SGD
- Nesterov Accelerated Gradient
- Adam
- …

Hands On

1. Warm Up

- running out-of-the-box
- opening the box

2. Assignment

- change configuration files and run different fine-tuning protocols
- [optional] suggestions for deeper exploration

Hands On

1. Warm Up

- running out-of-the-box
- opening the box

2. Assignment

- change configuration files and run different fine-tuning protocols
- [optional] suggestions for deeper exploration

Running out-of-the-box

Follow instructions here: https://github.com/vvv-school/tutorial_dl-tuning

1. run the tester train_and_test_net_tester.sh
   (you should already have done this)

2. run the example train_and_test_net.sh

Opening the box

Look at the content of the example:

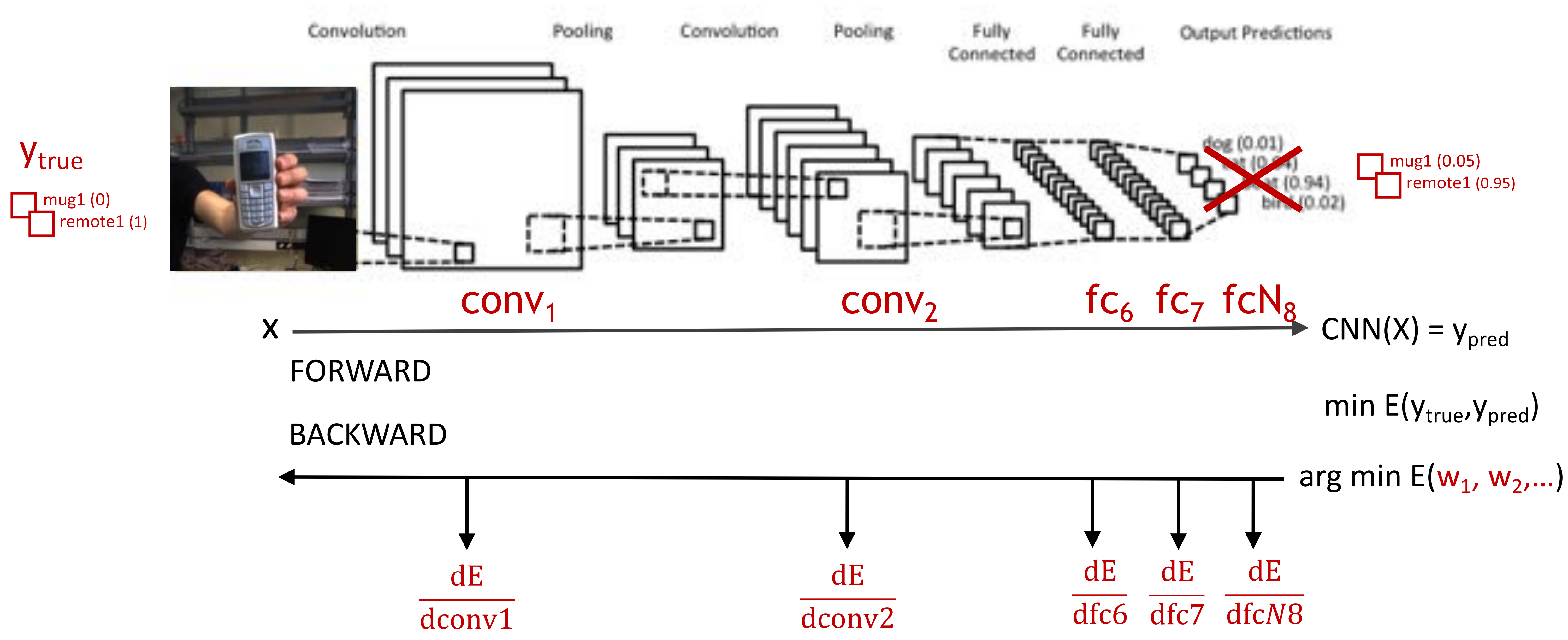
$ cd $LAB_DIR/tutorial_dl-tuning/id_2objects_caffenet

- all-0/train_val.prototxt
- all-0/deploy.prototxt
- all-0/solver.prototxt

- images_lists/train.txt
- images_lists/val.txt
- images_lists/test.txt
- images_lists/labels.txt

train_and_test_net.sh

What’s happening?
Fine-tuning CaffeNet on a 2-class identification task



train_val.prototxt

batch_size (train)

…

lr_mult (conv1-5)

lr_mult (fc6)

lr_mult (fc7)

lr_mult (fc8N)

…

[num_output]

solver.prototxt

max_iter

[test_interval,
display]

base_lr

train_val.prototxt

batch_size (train)

…

lr_mult (conv1-5)

lr_mult (fc6)

lr_mult (fc7)

lr_mult (fc8N)

…

[num_output]

solver.prototxt

max_iter

[test_interval,
display]

base_lr

1. which layer fine-tune/learn from scratch
2. with which learning rate(s)
3. varying batch size

train_val.prototxt

batch_size (train)
batch_size (validation)

…
lr_mult (conv1-5)
lr_mult (fc6)
lr_mult (fc7)
lr_mult (fc8N)
…
num_output

deploy.prototxt

fc6
fc7
fc8N

num_output

train_and_test_net.sh

First defines paths (to caffe executables, to images, etc.), then:

1. create databases (train & val LMDBs and mean.binaryproto):

```
${Caffe_ROOT}/build/tools/convert_imageset \
        --resize_width=256 --resize_height=256 --shuffle \
        ${IMAGES_DIR} ${FILELIST_TRAIN} ${LMDB_TRAIN}
```

→ lmdb_train (lmdb_val)

```
${Caffe_ROOT}/build/tools/compute_image_mean \
        ${LMDB_TRAIN} ${BINARYPROTO_MEAN}
```

→ mean.binaryproto

train_and_test_net.sh

2. train:

```
${Caffe_ROOT}/build/tools/caffe \
        train -solver ${SOLVER_FILE} -weights ${WEIGHTS_FILE} \
        --log_dir=${TUTORIAL_DIR}/${EX}/${PROTOCOL}
```

→ icw_iter_*.caffemodel, icw_iter_*.solverstate, caffe.INFO

3. parse caffe.INFO:

```
${TUTORIAL_DIR}/scripts/parse_caffe_log.sh \
        ${TUTORIAL_DIR}/${EX}/${PROTOCOL}/caffe.INFO
```

→ caffeINFOtrain.txt, caffeINFOval.txt (tables)

train_and_test_net.sh

4. plot caffeINFOtrain.txt, caffeINFOval.txt (e.g. with MATLAB):

```
matlab -nodisplay -nodesktop -r \
    "addpath('${TUTORIAL_DIR}/scripts');
     try
        plot_log('${TUTORIAL_DIR}/${EX}/${PROTOCOL}');
     catch;
     end;
     quit"
```

→ matlab_caffeINFO_acc.png, matlab_caffeINFo_loss.png, matlab_caffeINFO_lr.png

train_and_test_net.sh

5. eventually choose best (or last) epoch:

```
snap_list=(`ls -t icw_iter*.caffemodel`)
FINAL_SNAP=${TUTORIAL_DIR}/${EX}/${PROTOCOL}/${snap_list[0]}
FINAL_MODEL=${TUTORIAL_DIR}/${EX}/${PROTOCOL}/final.caffemodel
mv ${FINAL_SNAP} ${FINAL_MODEL}
rm ${TUTORIAL_DIR}/${EX}/${PROTOCOL}/icw_iter_*.solverstate
rm ${TUTORIAL_DIR}/${EX}/${PROTOCOL}/icw_iter_*.caffemodel
```

3. test final.caffemodel:

```
${TUTORIAL_DIR}/scripts/src/build/classify_image_list_vvv17 \
        ${DEPLOY_FILE} ${FINAL_MODEL} ${BINARYPROTO_MEAN} \
        ${LABELS_FILE} ${IMAGES_DIR} ${FILELIST_TEST} \
        ${TUTORIAL_DIR}/${EX}/${PROTOCOL}/test_acc_day1.txt
        ${PRINT_PREDICTIONS} ${IMG_DELAY}
```

→ test_acc_day1.txt

Hands On

1. Warm Up

- running out-of-the-box
- opening the box

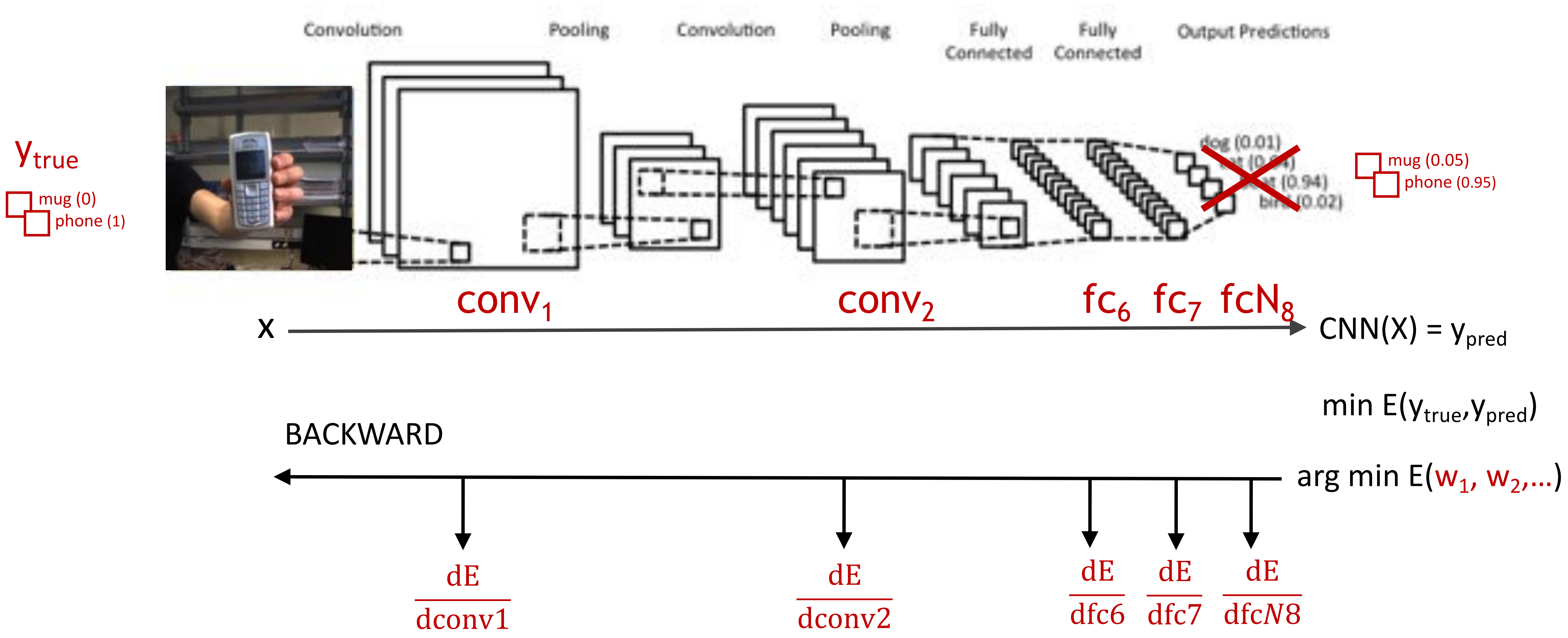
2. Assignment

- change configuration files and run different fine-tuning protocols
- [optional] suggestions for deeper exploration

Assignment

1. Apply the protocol we have used in the tutorial on a different task
2. Try fine-tuning also fc7, fc6
3. Try fine-tuning all layers
4. Try learning fc7, fc6 from scratch
5. Vary the batch size

name	batch size (train)	conv layers	fc6	fc7	fc8	# epochs	acc day1	acc day2
all-0	32	lr: 0	lr: 0	lr: 0	lr: 1e-2 from scratch	6	?	?



name	batch size (train)	conv layers	fc6	fc7	fc8	# epochs	acc day1	acc day2
all-0	32	lr: 0	lr: 0	lr: 0	lr: 1e-2 from scratch	6	?	?
conv-0_fc6-2_fc7-2	32	lr: 0	lr: 1e-2	lr: 1e-2	lr: 1e-2 from scratch	6	?	?
conv-0_fc6-3_fc7-3	32	lr: 0	lr: 1e-3	lr: 1e-3	lr: 1e-2 from scratch	6	?	?
all-3	32	lr: 1e-3	lr: 1e-3	lr: 1e-3	lr: 1e-2 from scratch	6	?	?
conv-0_fc6N-2_fc7N-2	32	lr: 0	lr: 1e-2 from scratch	lr: 1e-2 from scratch	lr: 1e-2 from scratch	18	?	?
conv-0_fc6N-4_fc7N-4	32	lr: 0	lr: 1e-4 from scratch	lr: 1e-4 from scratch	lr: 1e-2 from scratch	36	?	?
all-3_batch-8	8	lr: 1e-3	lr: 1e-3	lr: 1e-3	lr: 1e-2 from scratch	6	?	?
all-0_batch-1	1	lr: 0	lr: 0	lr: 0	lr: 1e-2 from scratch	6	?	?

name	batch size (train)	conv layers	fc6	fc7	fc8	# epochs	acc day1	acc day2
all-0	32	lr: 0	lr: 0	lr: 0	lr: 1e-2 from scratch	6	?	?
conv-0_fc6-2_fc7-2	32	lr: 0	lr: 1e-2	lr: 1e-2	lr: 1e-2	6	?	?
conv-0_fc6-3							?	?
all-3							?	?
conv-0_fc6N-2							?	?
conv-0_fc6N-4			from scratch	from scratch	from scratch		?	?
all-3_batch-8	8	lr: 1e-3	lr: 1e-3	lr: 1e-3	lr: 1e-2 from scratch	6	?	?
all-0_batch-1	1	lr: 0	lr: 0	lr: 0	lr: 1e-2 from scratch	6	?	?

Send your results to:

https://goo.gl/forms/3kn1jV1jcQizlBa92

Things you may want to try…

Try changing the task reducing the number of example frames per object in the training set:

1. create a new exercise folder (call it e.g. id_10objects_less_frames) by copying id_10objects_caffenet
2. open id_10objects.m in the new folder and change setlist.max_frames.train (try e.g. 20, or 10, or 5)
3. open example_task_generation.m and modify the paths to
   (i) use the configuration file at point 2)
   (ii) create training, validation and test sets in the new folder
   → run the script to generate the new training set
4. implement the protocols all-0 and all-3 on this new task and run them

→ how much does the recognition accuracy decrease? is there a better protocol?

Things you may want to try…

Try to perform an object categorization task:

1. create a new exercise folder (call it e.g. cat_3categories) by copying id_10objects_caffenet
2. open id_10objects.m in the new folder (you may rename it to match the new task) and change it to generate a task of discrimination between the 3 categories in iCW (e.g. by training on 7 objects/cat, validating on 2, and testing one the remaining 1)
3. open example_task_generation.m and modify the paths to
   (i) use the configuration file at point 2)
   (ii) create training, validation and test sets in the new folder
   → run the script to generate the new training set
4. implement the protocols all-0 and all-3 on this new task and run them

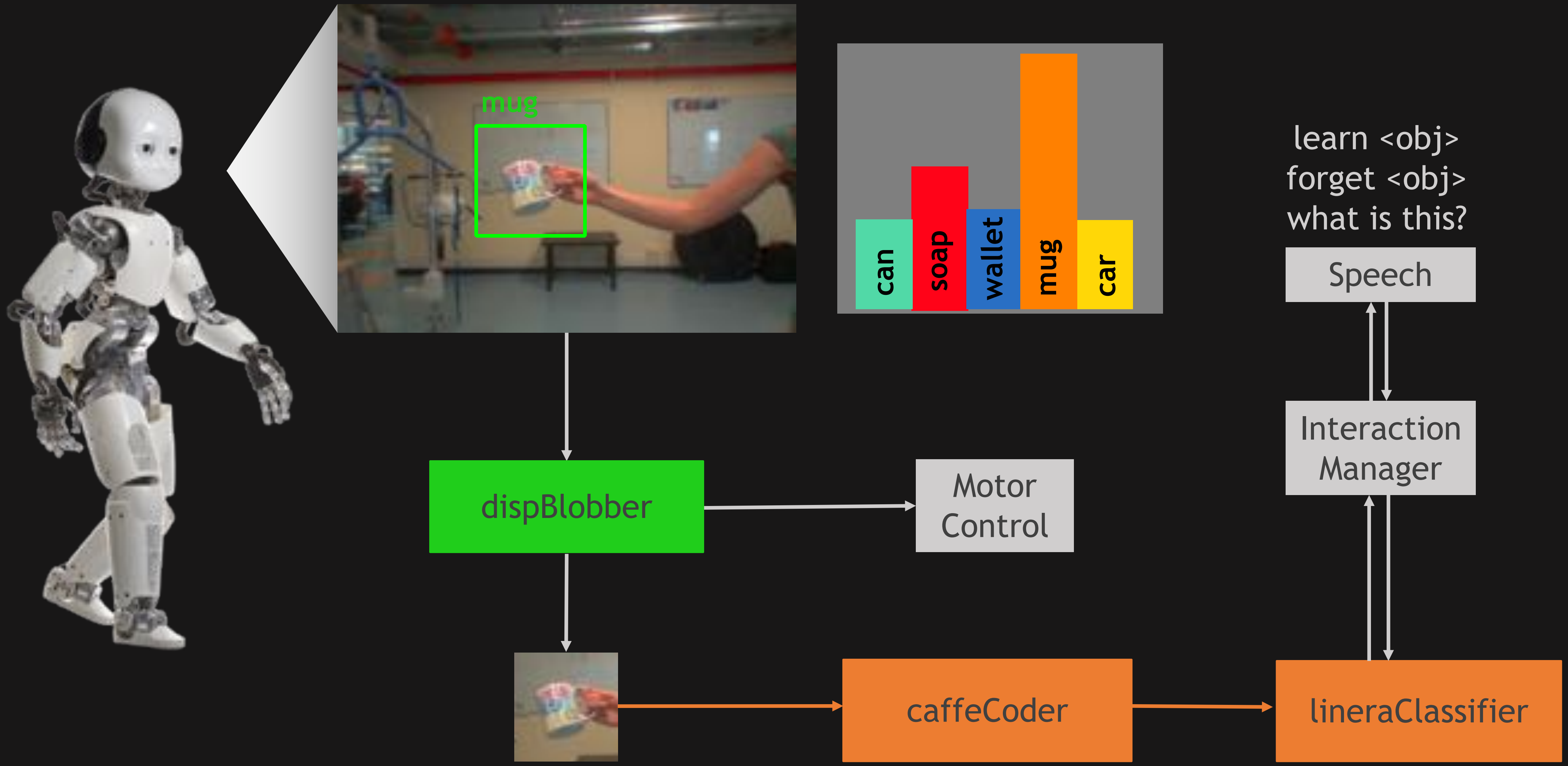
→ how much does the recognition accuracy change? is there a better protocol?
→ is this task more or less difficult? why?

Things you may want to try…

Deploy the fine-tuned model in the recognition pipeline on the iCub!

- create a folder with your surname
- put inside the final.caffemodel and mean.binaryproto
- together with a test.prototxt (see next for how to create this file…)

On the Fly Object Recognition Demo on iCub



Pointers to GitHub Repositories

DEMO: [robotology/onthefly-recognition]()

→ manager
→ speech
→ tracking

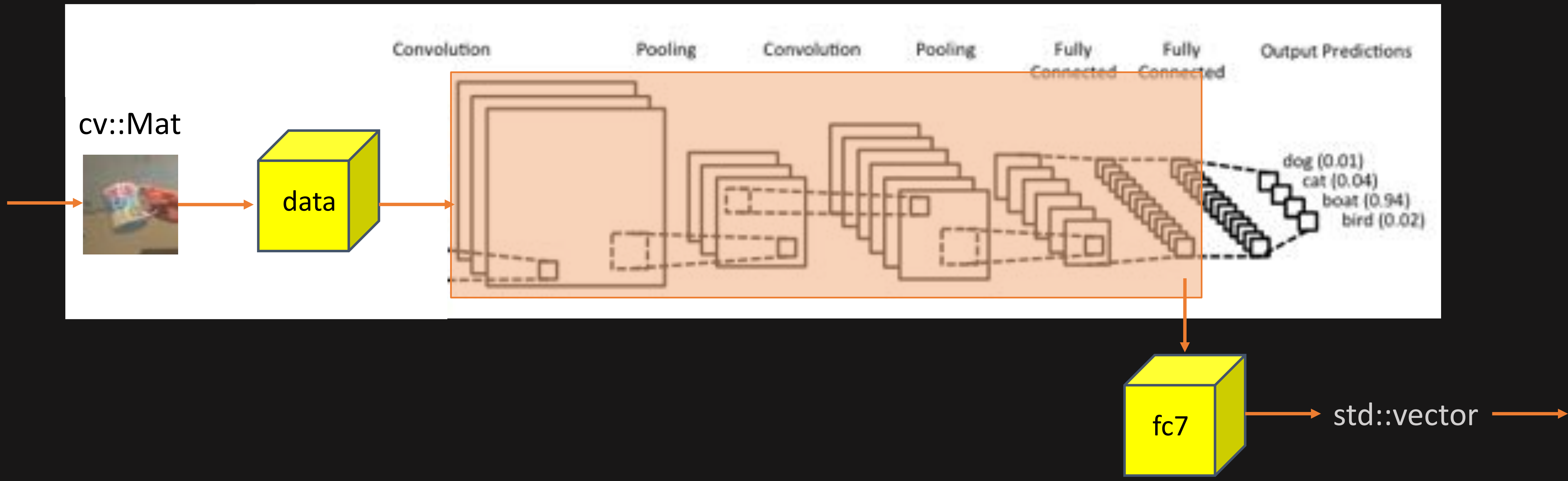
SEGMENTATION: [robotology/segmentation]()

→ dispBlobber

RECOGNITION PIPELINE: [robotology/himrep]()

→ caffeCoder
→ linearClassifier

caffeCoder: (YARP) wrapper for caffe



caffeCoder: call with your own files

provided that you have:

/home/icub/.local/share/yarp/contexts/himrep/pasquale/final.caffemodel
/home/icub/.local/share/yarp/contexts/himrep/pasquale/mean.binaryproto
/home/icub/.local/share/yarp/contexts/himrep/pasquale/test.prototxt

call (you can try also on the VM):

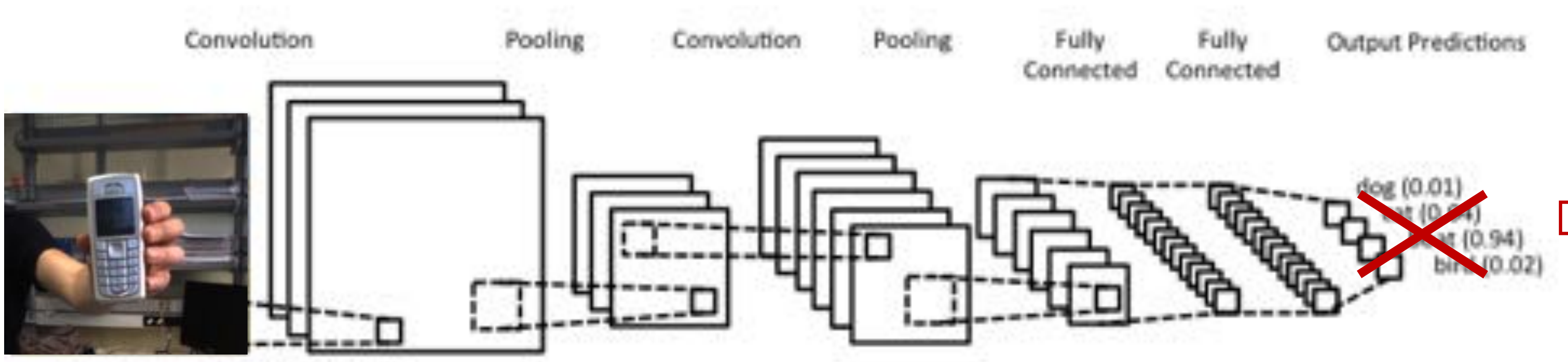
```
caffeCoder --pretrained_binary_proto_file
/home/icub/.local/share/yarp/contexts/himrep/pasquale/final.caffemodel
--feature_extraction_proto_file pasquale/test.prototxt --extract_features_blob_names fc6N
```

More details on assignment…

1. Fine-tuning CaffeNet on a ~~2~~ 10-class identification task

y_{true}

mug (0)
phone (1)



mug (0.05)
phone (0.95)

$conv_1$ $conv_2$ fc_6 fc_7 fcN_8

$x \longrightarrow CNN(X) = y_{pred}$

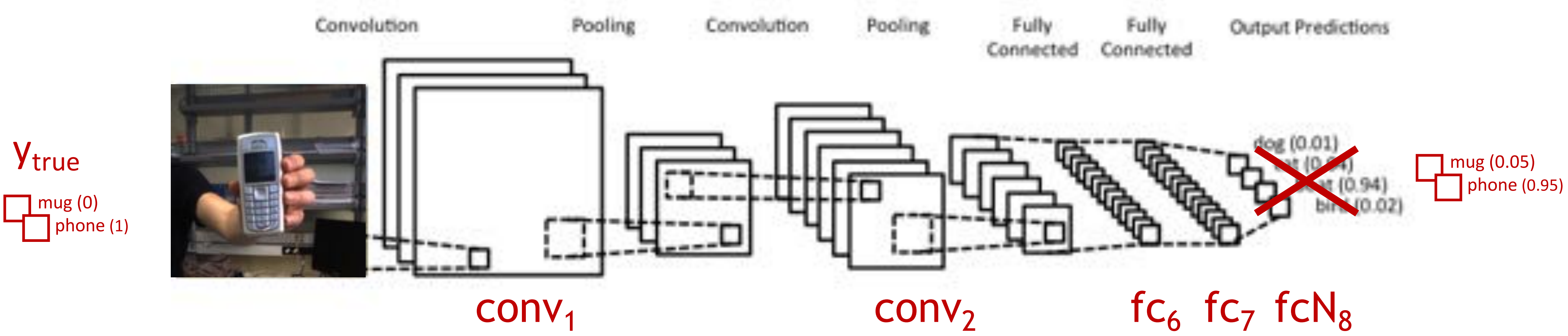
$\min E(y_{true}, y_{pred})$

BACKWARD

$\arg\min E(w_1, w_2, \ldots)$

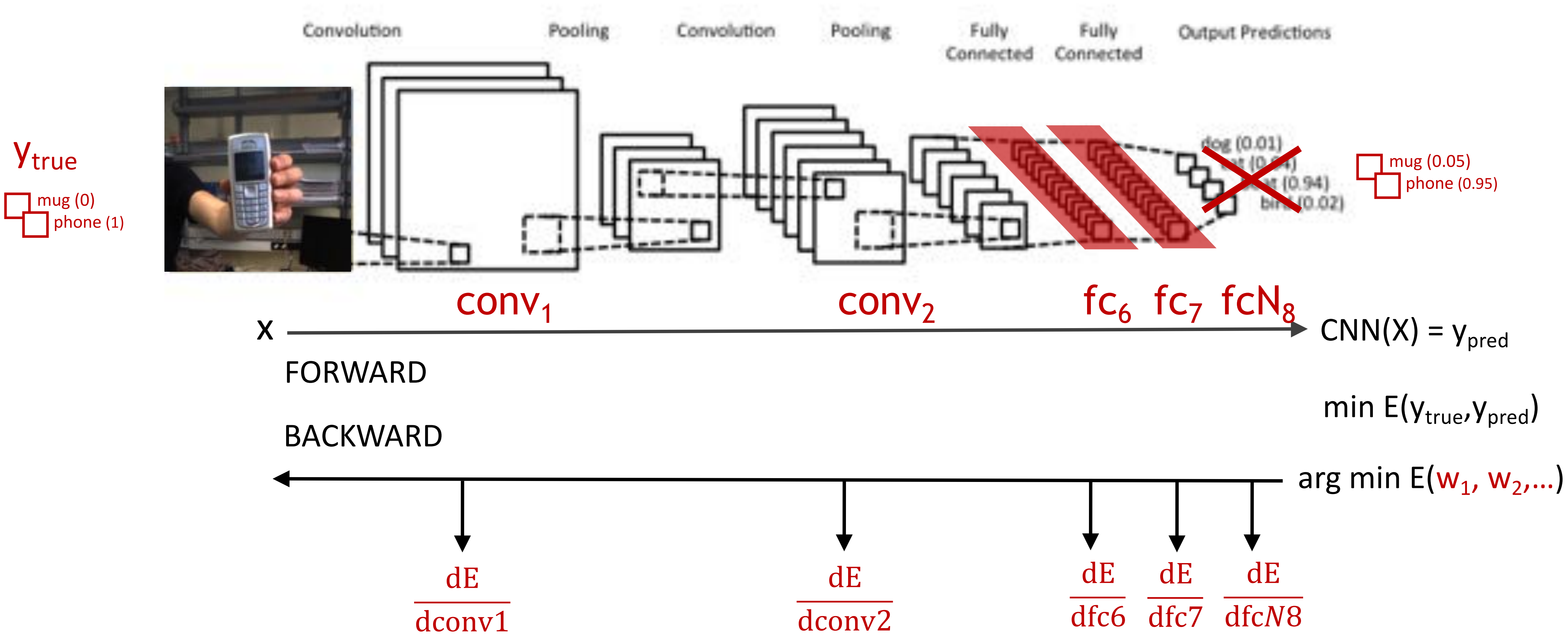
$\frac{dE}{dconv1}$ $\frac{dE}{dconv2}$ $\frac{dE}{dfc6}$ $\frac{dE}{dfc7}$ $\frac{dE}{dfcN8}$

1. Fine-tuning CaffeNet on a ~~2~~-class identification task
10

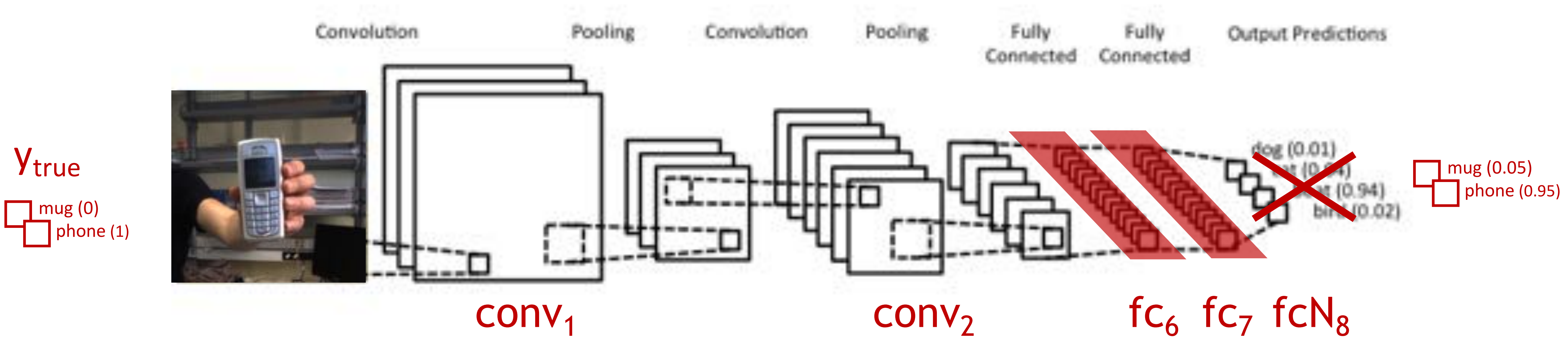


name	batch size (train)	conv layers	fc6	fc7	fc8	# epochs	acc day1	acc day2
all-0	32	lr: 0	lr: 0	lr: 0	lr: 1e-2 from scratch	6	?	?

2. Try fine-tuning fc7,fc6



2. Try fine-tuning fc7,fc6



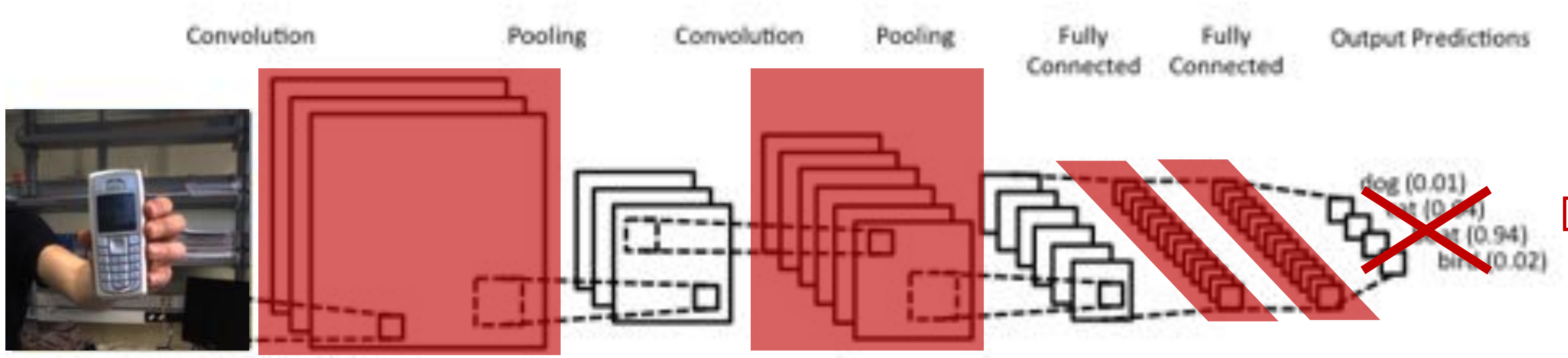
name	batch size (train)	conv layers	fc6	fc7	fc8	# epochs	acc day1	acc day2
conv-0_fc6-2_fc7-2	32	lr: 0	lr: 1e-2	lr: 1e-2	lr: 1e-2 from scratch	6	?	?
conv-0_fc6-3_fc7-3	32	lr: 0	lr: 1e-3	lr: 1e-3	lr: 1e-2 from scratch	6	?	?

3. Try fine-tuning all layers

y_{true}

mug (0)

phone (1)



mug (0.05)

phone (0.95)

$conv_1$ $conv_2$ fc_6 fc_7 fcN_8

x $\longrightarrow$ $CNN(X) = y_{pred}$

FORWARD

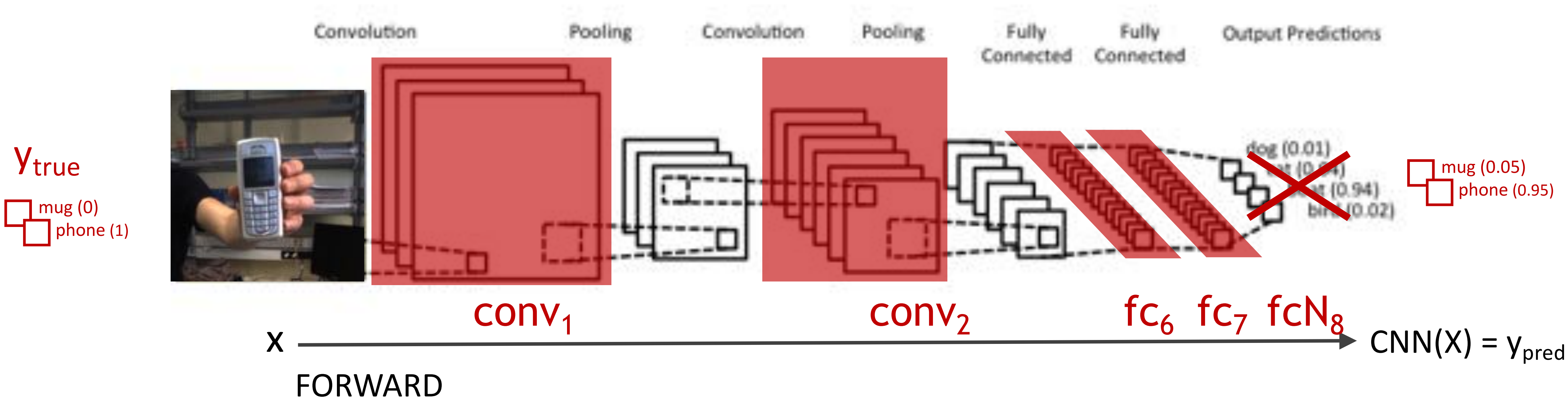
$\min E(y_{true}, y_{pred})$

BACKWARD

$\longleftarrow$ $\arg\min E(w_1, w_2, \dots)$

$\frac{dE}{dconv1}$ $\frac{dE}{dconv2}$ $\frac{dE}{dfc6}$ $\frac{dE}{dfc7}$ $\frac{dE}{dfcN8}$

3. Try fine-tuning all layers

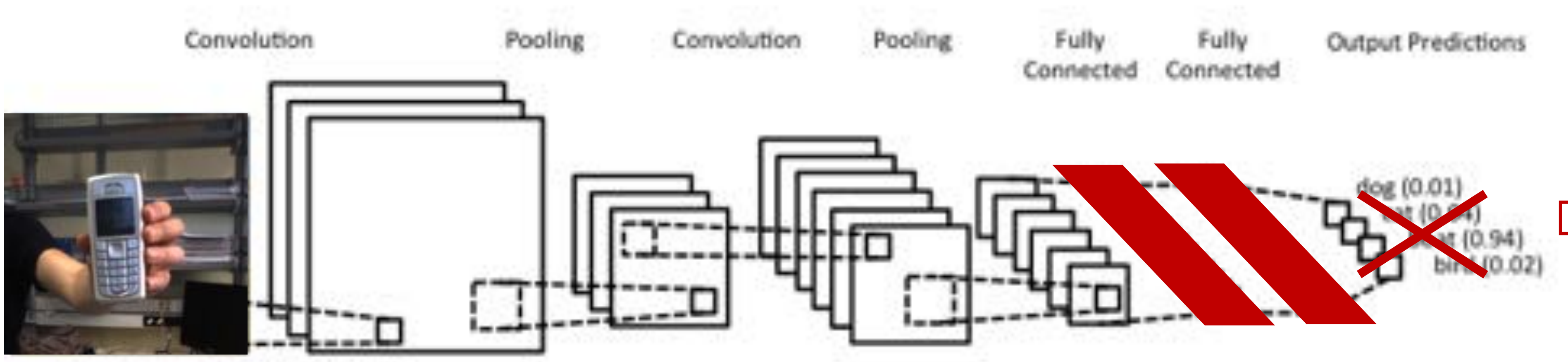


name	batch size (train)	conv layers	fc6	fc7	fc8	# epochs	acc day1	acc day2
all-3	32	lr: 1e-3	lr: 1e-3	lr: 1e-3	lr: 1e-2 from scratch	6 (12)	?	?

4. Is it possible to learn from scratch fc7,fc6?

y_{true}

mug (0)
phone (1)



mug (0.05)
phone (0.95)

$conv_1$ $conv_2$ $fcN_6 fcN_7 fcN_8$

x ⟶ $CNN(X) = y_{pred}$

FORWARD

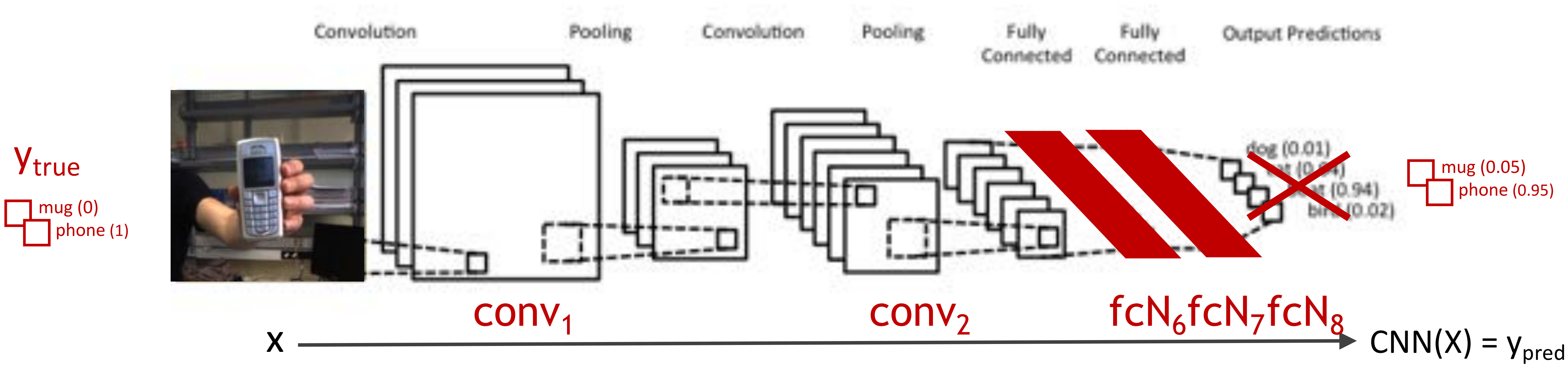
BACKWARD

$\min E(y_{true}, y_{pred})$

$\arg\min E(w_1, w_2, \ldots)$

$\frac{dE}{dconv1}$ $\frac{dE}{dconv2}$ $\frac{dE}{dfcN6}$ $\frac{dE}{dfcN7}$ $\frac{dE}{dfcN8}$

4. Is it possible to learn from scratch fc7,fc6?



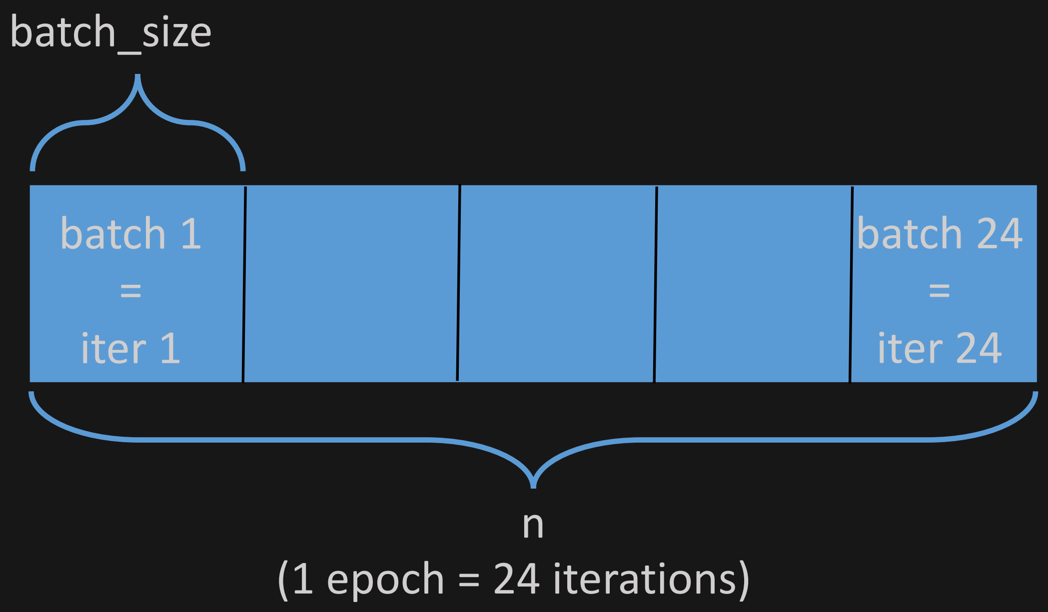
name	batch size (train)	conv layers	fc6	fc7	fc8	# epochs	acc day1	acc day2
conv-0_fc6N-2_fc7N-2	32	lr: 0	lr: 1e-2 from scratch	lr: 1e-2 from scratch	lr: 1e-2 from scratch	18	?	?
conv-0_fc6N-4_fc7N-4	32	lr: 0	lr: 1e-4 from scratch	lr: 1e-4 from scratch	lr: 1e-2 from scratch	18 (36)	?	?

5. Vary the batch size

name	batch size (train)	conv layers	fc6	fc7	fc8	# epochs	acc day1	acc day2
all-3_batch-8	8	lr: 1e-3	lr: 1e-3	lr: 1e-3	lr: 1e-2 from scratch	6	?	?
all-0_batch-1	1	lr: 0	lr: 0	lr: 0	lr: 1e-2 from scratch	6	?	?

solver.prototxt: hot to set batch-related parameters

```
net: "train_val.prototxt”
solver_mode: CPU

# carry out <max_iter> training iterations
max_iter: 144 (6 x 24)

## the validation will carry out <test_iter> iterations
test_iter: 10

## carry out validation every <test_interval> training iterations
test_interval: 24

## display every <display> iterations
display: 24

## save model every <snapshot> iterations
snapshot: 0
snapshot_prefix: "icw”
```



training set:
n = 768
batch_size = 32
iters_per_epoch = 768/32= 24
max_epoch = 6

validation set:
n = 320
batch_size =32
iters_per_epoch =320/32=10
test_epoch=1

train_val.prototxt: set the batch size

```
layer {
   name: "data"
   type: "Data"
   top: "data"
   top: "label"
   include {
      phase: TRAIN
   }
   transform_param {
      mirror: true
      crop_size: 227
      mean_file: "../mean.binaryproto”
   }
   data_param {
      source: "../lmdb_train”
      batch_size: 32
      backend: LMDB
   }
}
```

```
layer {
   name: "data"
   type: "Data"
   top: "data"
   top: "label"
   include {
      phase: TEST
   }
   transform_param {
      mirror: false
      crop_size: 227
      mean_file: "../mean.binaryproto”
   }
   data_param {
      source: "../lmdb_val”
      batch_size: 32
      backend: LMDB
   }
}
```

train_val.prototxt: set lr_mult for conv and fc layers

```
layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  param {
    lr_mult: <x>  [1,10,0.1…]
    decay_mult: 1
  }
  param {
    lr_mult: <2x>  [2,20,0.2…]
    decay_mult: 0
  }
 … …
… …
  convolution_param {
    num_output: 96
    kernel_size: 11
    stride: 4
    weight_filler {
      type: "gaussian"
      std: 0.01
    }
    bias_filler {
      type: "constant"
      value: 0
    }
  }
}
```

train_val.prototxt: set lr_mult for conv and fc layers

```
layer {
   name: "fc6"
   type: "InnerProduct"
   bottom: "pool5"
   top: "fc6"
   param {
      lr_mult: <x>  [e.g.: 1,10,0.1...]
      decay_mult: 1
   }
   param {
      lr_mult: <2x>  [e.g.: 2,20,0.2...]
      decay_mult: 0
   }
... ...
```

```
... ...
   inner_product_param {
      num_output: 4096
      weight_filler {
         type: "gaussian"
         std: 0.005
      }
      bias_filler {
         type: "constant"
         value: 1
      }
   }
}
```

train_val.prototxt: rename layers (and related output blobs) that you learn from scratch

```
layer {
   name: "fc8N"
   type: "InnerProduct"
   bottom: ”fc7"
   top: ”fc8N"
   param {
      lr_mult: 1
     decay_mult: 1
   }
   param {
      lr_mult: 2
      decay_mult: 0
   }
… …
```

```
… …
   inner_product_param {
      num_output: 2
      weight_filler {
         type: "gaussian"
         std: 0.01
      }
      bias_filler {
         type: "constant"
         value: 0
     }
   }
}
```

train_val.prototxt: rename layers (and related output blobs) that you learn from scratch

```
layer {
    name: "accuracy"
    type: "Accuracy"
    bottom: "fc8N"
    bottom: "label"
    top: "accuracy”
}
```

```
layer {
    name: "loss"
    type: "SoftmaxWithLoss"
    bottom: "fc8N"
    bottom: "label"
    top: "loss”
}
```

deploy.prototxt

- ✓ same structure as train_val.protoxt, but

  no DATA layer
  no learning parameters (lr_mult…)

- ✓ pay attention to:

  → rename the layers which you learned from scratch (fc8 → fc8N)

  → set num_output (fc8N) accordingly to the task (num_output=2)

test.prototxt: you have to create it

✓ same structure as train_val.prototxt, but

"MemoryData" DATA layer (see next slide)
no learning parameters

✓ pay attention to:

→ rename the layers which you learned from scratch (fc8 → fc8N)

→ set num_output (fc8N) accordingly to the task (num_output=2)

(you can copy the deploy.prototxt)

test.prototxt: example “MemoryData” layer

```
layer {
  name: "data"
  type: "MemoryData"
  top: "data"
  top: "label"
  transform_param {
    mirror: false
    crop_size: 227
    mean_file: "/home/icub/.local/share/yarp/contexts/himrep/pasquale/mean.binaryproto "
  }
  memory_data_param {
    batch_size: 1
    channels: 3
    height: 227
    width: 227
  }
}
```