Hands on

Object Recognition (on a Robot) with Deep Learning

VVV18 Winter School on Humanoid Robot Programming

Santa Margherita Ligure

Feb. 13 2018

How would you
make a robot recognize objects?

[e.g. a `mug` but also `my mug`]

1. Collect Data

1. Collect Data

Fine-tuning a CNN (pre-trained on ImageNet) on the task

→ we'll do this in Caffe

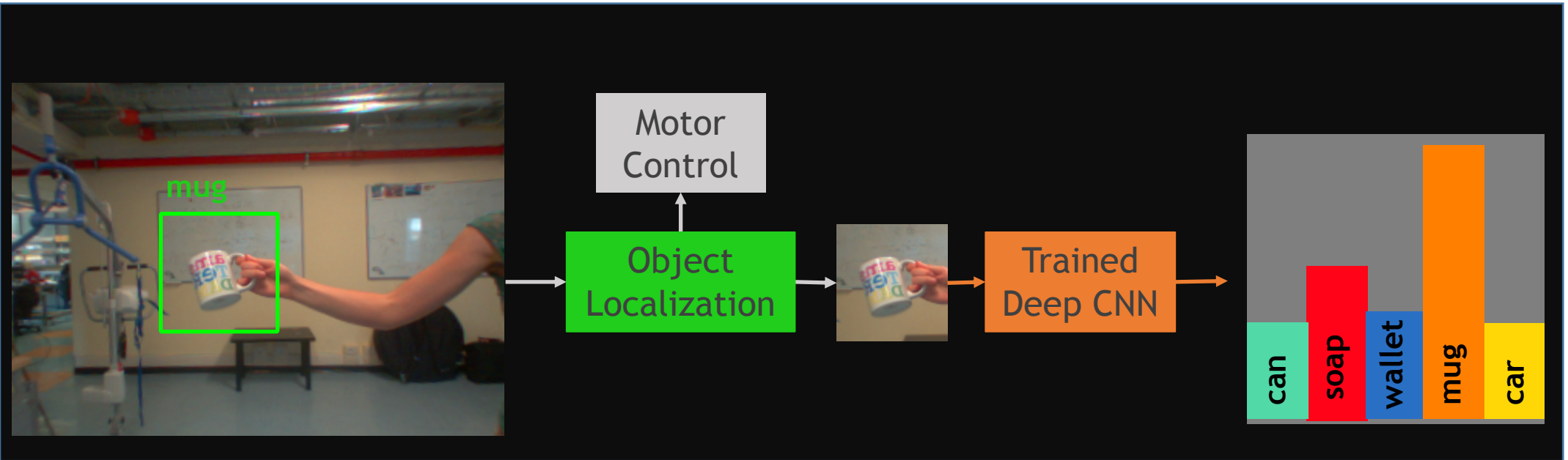
2. Train

1. Collect Data

Fine-tuning a CNN (pre-trained on ImageNet) on the task

→ we'll do this in Caffe

2. Train



3. Deploy

This afternoon

1. The data (we collected for you)

2. Fine-tuning a Deep CNN with Caffe:
    a. Tutorial
    b. Assignment

3. Model Deployment (in YARP)
    a. Tutorial (you can play with)

This afternoon

1. The data (we collected for you)

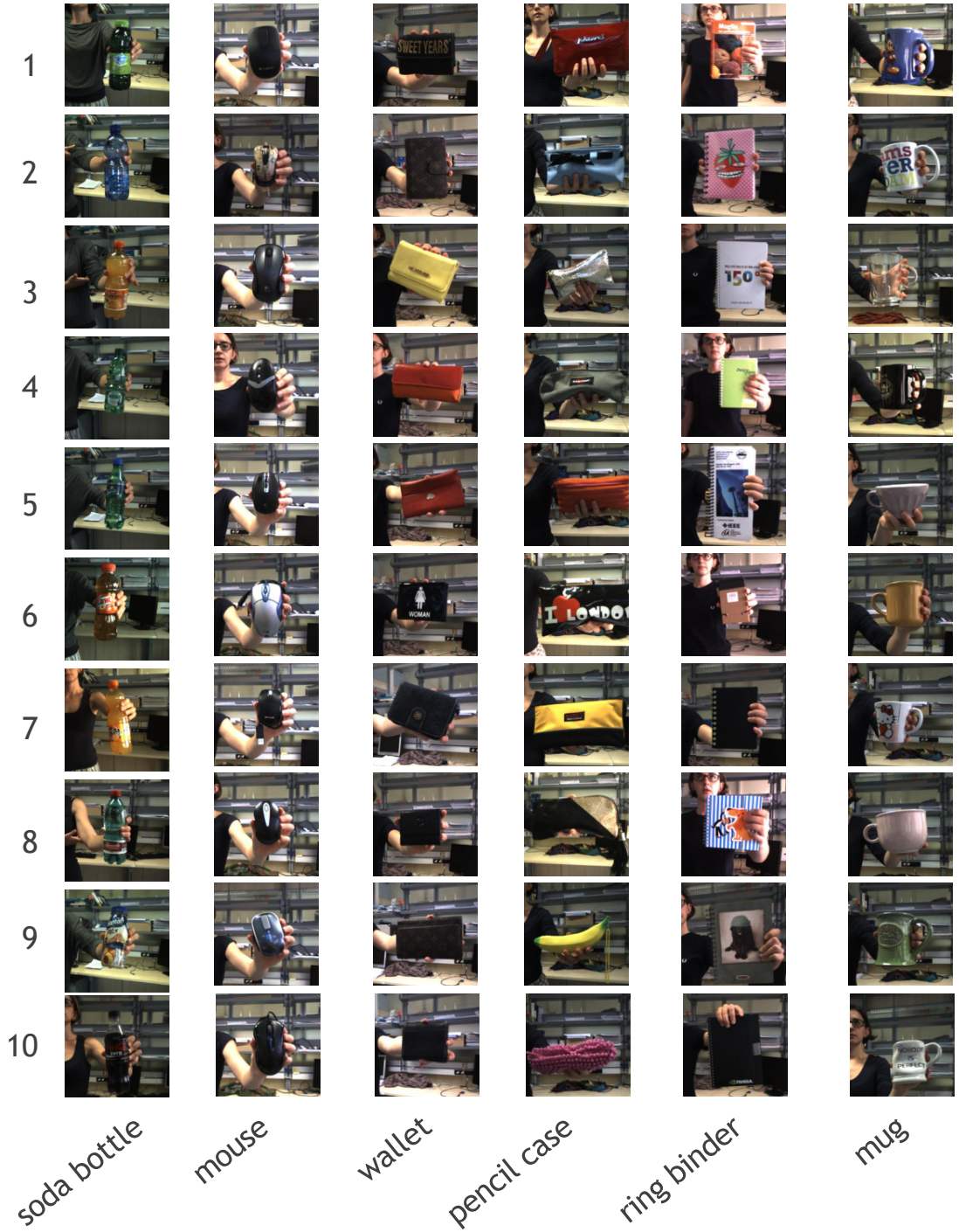
2. Fine-tuning a Deep CNN with Caffe:
    a. Tutorial
    b. Assignment

3. Model Deployment (in YARP)
    a. Tutorial (you can play with)

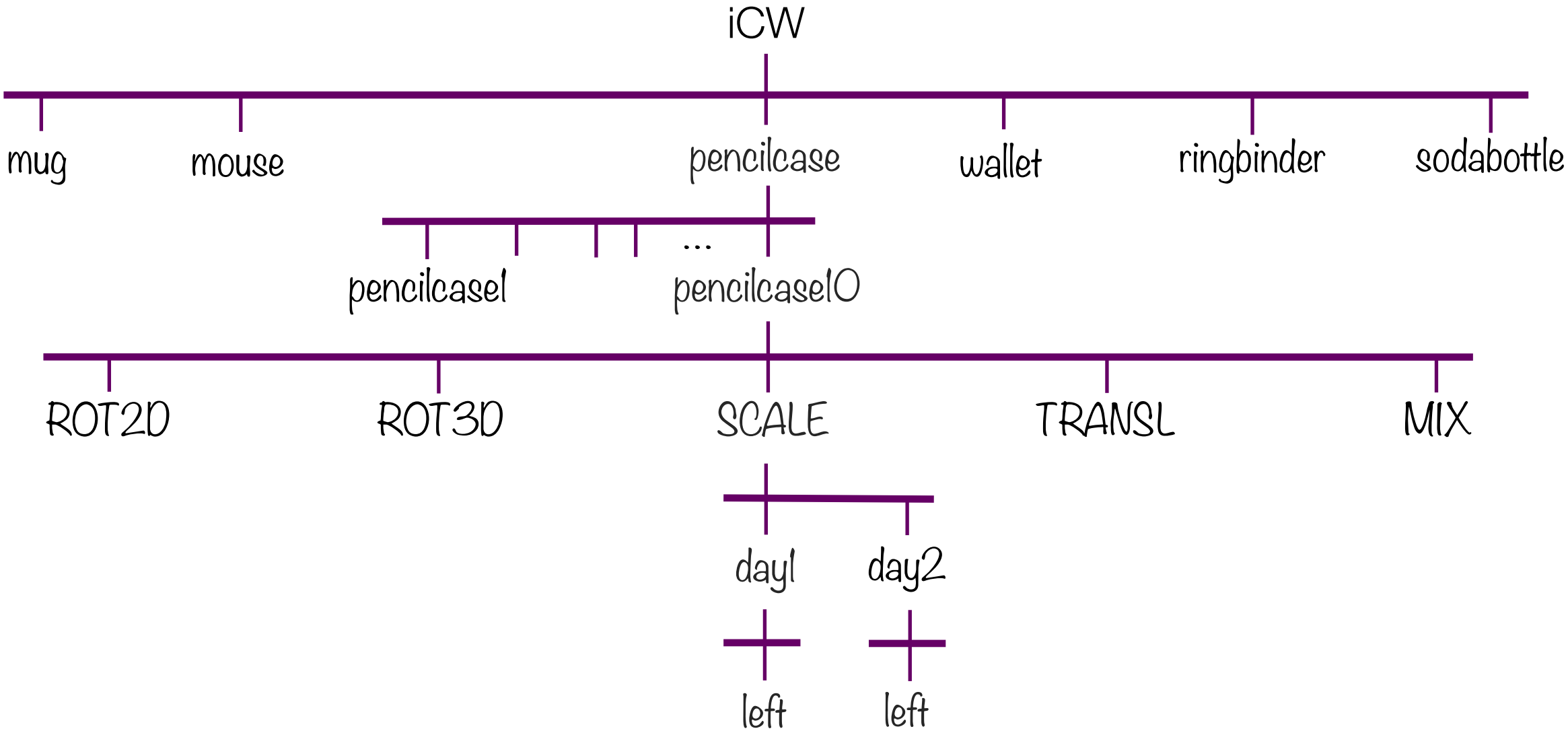
What is the visual world of iCub?

- limited number of objects!
- lots of images for each object:
  - ✓ different viewpoint
    - 2D rotations
    - 3D rotations
    - Scaling
    - Background changes
    - Composition of the above
  - ✓ different illumination
  - ✓ …

A (small) dataset to play with



```
$ cd $ROBOT_CODE/datasets
$ tar –xzvf iCW-vvv18.tar.gz
```



6 x 10 x 5 x 2 x (~100 frames) → **~ 60k images**

This afternoon

1. The data (we collected for you)

2. Fine-tuning a Deep CNN with Caffe:
    a. Tutorial
    b. Assignment

3. Model Deployment (in YARP)
    a. Tutorial (you can play with)

Fine-tuning: Tutorial

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

- ✓ Get ready for the afternoon
- ✓ Get ready for the tutorial

→ Understand what's happening and

1. Run the script create_imagesets.py
2. Run the tester train_and_test_net_tester.sh
3. Run the example train_and_test_net.sh

```
$ cd $ROBOT_CODE/caffe
$ ls
```

Caffe:

Convolutional Architecture for Fast Feature Embedding

Web: http://caffe.berkeleyvision.org/
GitHub: https://github.com/BVLC/caffe
Community: Caffe Users - Google Groups

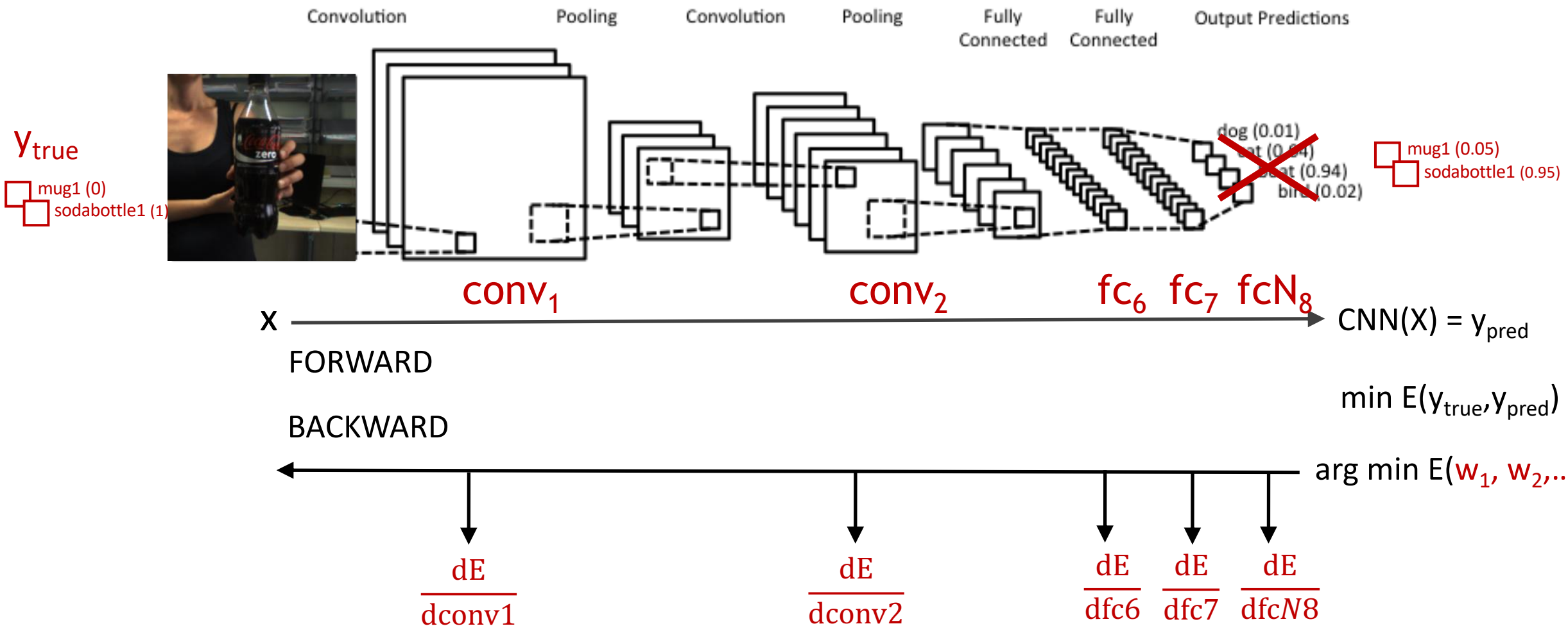
C++ APIs

command-line
Python
Matlab

} interfaces

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

Fine-tuning a CNN on a 2-class identification task



Fine-tuning a CNN on a 2-class identification task

Look at the content of the example in the folder `id_2objects`:

all-3/train_val.prototxt
all-3/deploy.prototxt

NETWORK MODEL

all-3/solver.prototxt

OPTIMIZATION PARAMETERS

imageset_config.yml

DATA (configuration file)

train_and_test_net_tester.sh
train_and_test_net.sh

Fine-tuning a CNN on a 2-class identification task

Look at the content of the example in the folder `id_2objects`:

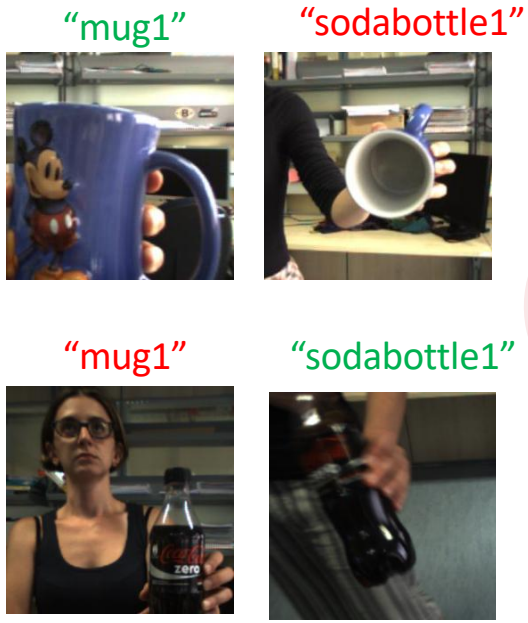
imageset_config.yml

DATA (configuration file)

DATA: Object Identification Task

train.txt, val.txt
(2D ROT, 3D ROT, SCALE, BKG)

test.txt
(MIX)



- ! Viewpoint
- ! Background
- ! Occlusions
- ! Light

```
$ cd $ROBOT_CODE/dl-lab/tutorial_dl-tuning
$ ./create_imagesets.py
```

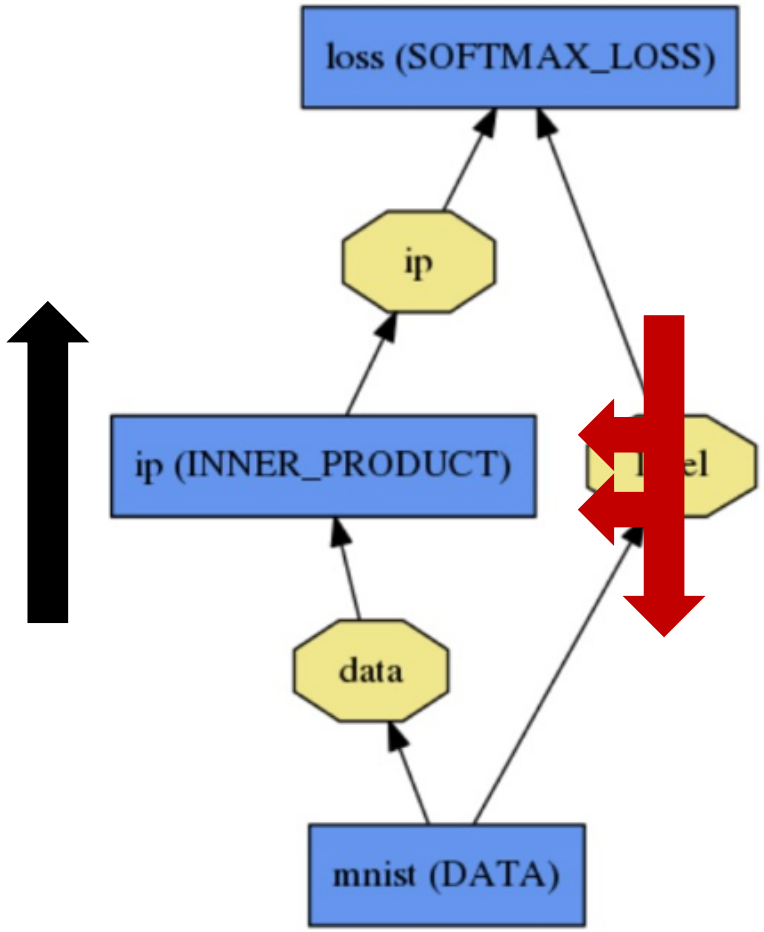
Fine-tuning a CNN on a 2-class identification task

Look at the content of the example in the folder `id_2objects`:

- all-3/train_val.prototxt
- all-3/deploy.prototxt

} **NETWORK MODEL**

NETWORK MODEL: Layers and Blobs



train_val.prototxt, deploy.prototxt

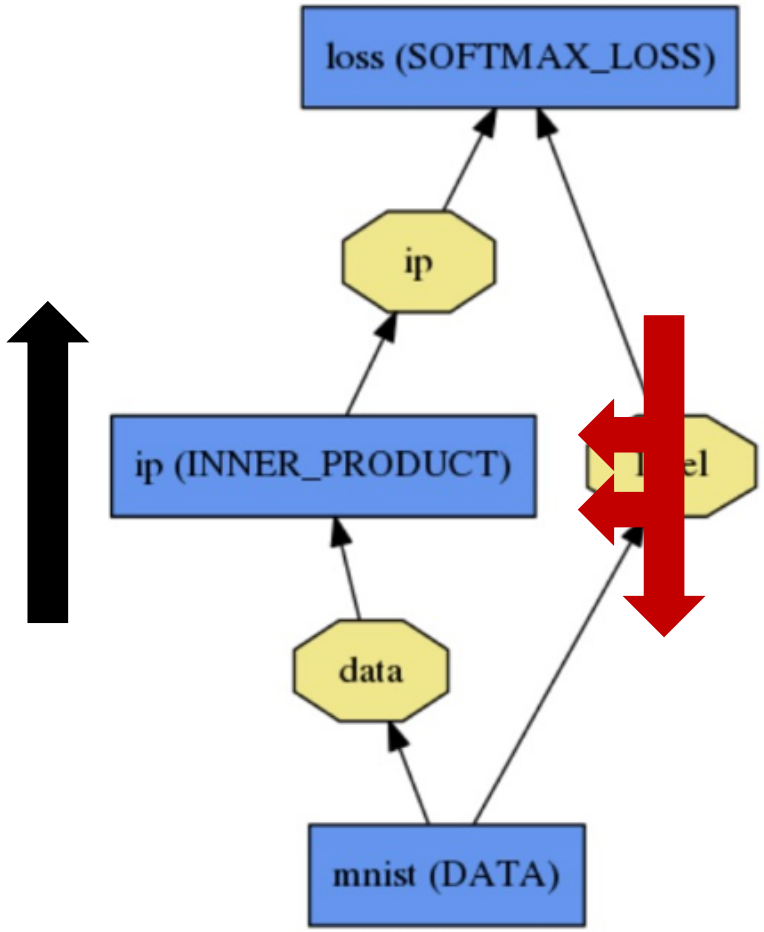
- a **Network** is a set of connected **Layers**:

  each **Layer** is defined by setup, **forward**, **backward** functions

  the **Network**'s passes are composed of layers' steps

- data flow through **Blobs** (N-dim array)

NETWORK MODEL: Layers



- ✓ “Convolution”
- ✓ “Pooling”
- ✓ “InnerProduct”
- ✓ “ReLU”
- ✓ …

1. Subtract mean image (or pixel) of training set
2. Fixed-size crop

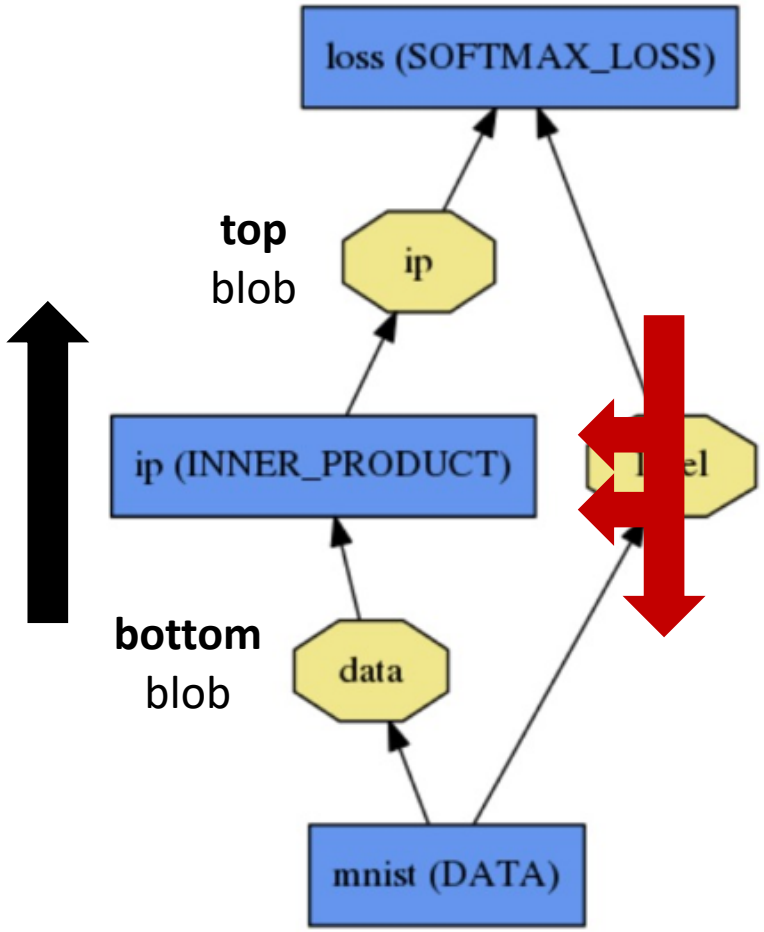
DATA layers (include image pre-processing steps):

- ✓ “Data” (formatted database)
- ✓ “MemoryData” (already in-memory)
- ✓ “ImageData” (image folder)

LOSS layers

- ✓ “SoftmaxWithLoss”
- ✓ “Accuracy”

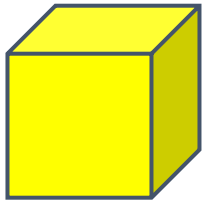
NETWORK MODEL: Blobs



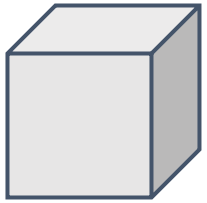
4-dimensional array stored in a C-contiguous fashion:

number N x channel C x height H x width W

- ✓ Hides the overhead of CPU/GPU computation by synchronizing and allocating memory 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

train_val.prototxt: **Data** layers and **TRAIN/TEST phases**

```
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: rename **fc8** and set **num_output**

```
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
    }
  }
}
```

```
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) like in** train_val.prototx **(num_output=2)**

Fine-tuning a CNN on a 2-class identification task

Look at the content of the example in the folder `id_2objects`:

- all-3/solver.prototxt } **OPTIMIZATION PARAMETERS**

OPTIMIZATION PARAMETERS

weights initialization (train_val.prototxt)

solver.prototxt :

at each iteration, process a batch of images (train_val.prototxt):

1. forward pass → output and loss
2. backward pass → gradients
3. update parameters (layer-specific learning rate defined in train_val.prototxt)

periodically:

test on the validation set
save snapshots

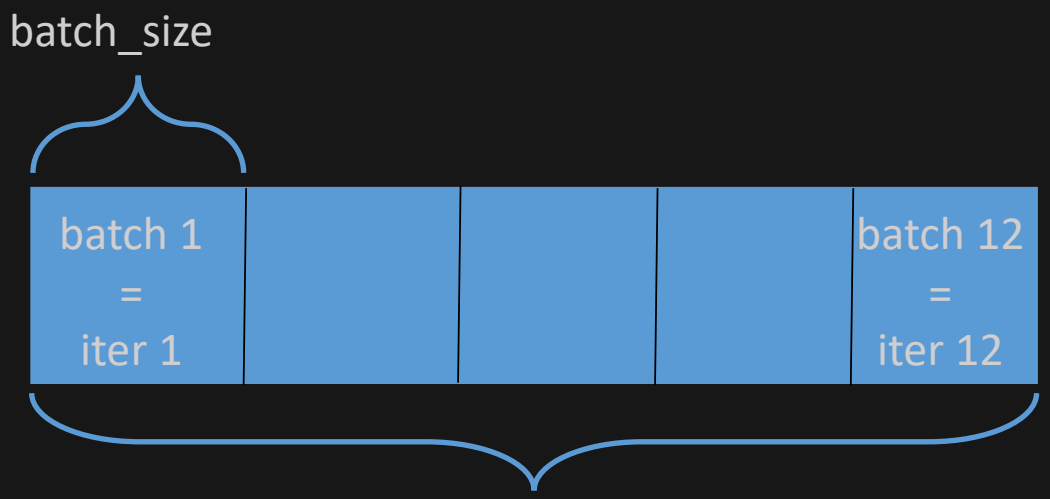
solver.prototxt

```
# number of training iterations
max_iter: 48

## number of iterations to be performed on the validation set
test_iter: 3

## carry out validation every <test_interval> training iterations
test_interval: 12

## display performance every <display> iterations
display: 12
```



training set:

- n = 384
- batch size = 32
- iters/epoch = 384/32= 12
- number of epochs = 4

validation set:

- n = 96
- batch size =32
- iters/epoch =96/32=3
- number of epochs = 1

Fine-tuning a CNN on a 2-class identification task

Look at the content of the example in the folder `id_2objects`:

train_and_test_net.sh

train_and_test_net.sh

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

1. create DATABASES:

```
${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 and plot train/validation curves:

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

→ caffeINFOtrain.txt, caffeINFOval.txt

```
gnuplot
        -e "iodir='${TUTORIAL_DIR}/${EX}/${PROTOCOL}'" ${PLOT_LOG_SH}
```

→ caffeINFO_acc.png, caffeINFo_loss.png, caffeINFO_lr.png

train_and_test_net.sh

4. choose best model (last epoch) and remove other snapshots:

```
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
```

→ final.caffemodel

5. test:

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

→ test_acc.txt

Fine-tuning: Assignment

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

Apply the protocol we have used in the tutorial on a different task:

1. look at imageset_config.yml
2. look at create_imagesets.py and run

} **DATA** (list of images in iCW)

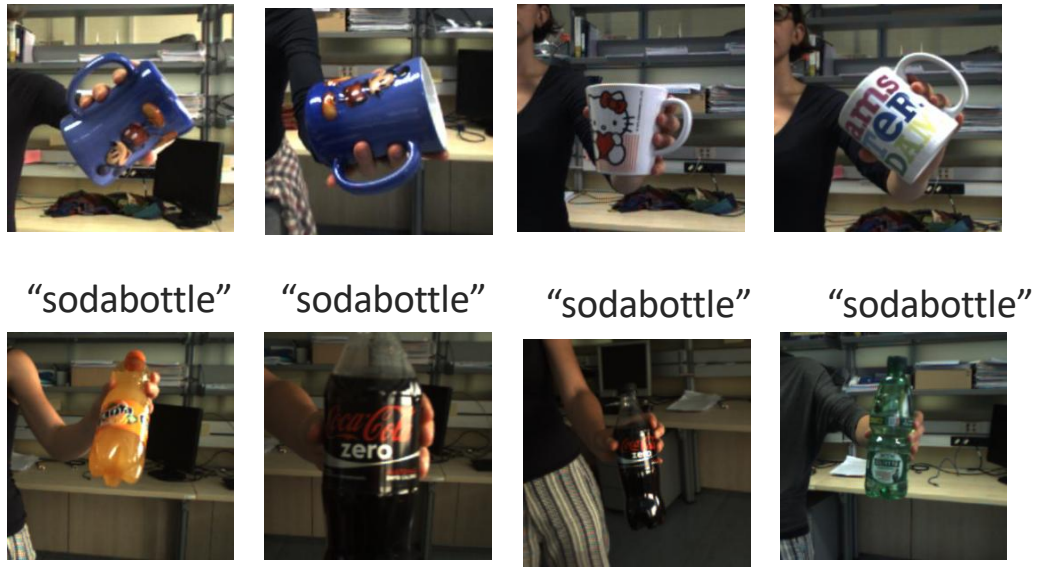
3. configure train_val.prototxt and deploy.prototxt
4. configure solver.prototxt

} **NETWORK MODEL**
} **OPTIMIZATION PARAMETERS**

5. look at train_and_test_net.sh and run

DATA: Object Categorization Task

train.txt, val.txt
(8+1 objects/category)



test.txt
(1 object/category)

- ! Viewpoint
- ! Background
- ! Occlusions
- ! Light
- ! Semantic Variability

```
$ cd $ROBOT_CODE/dl-lab/assignment_dl-tuning
$ ./create_imagesets.py
```

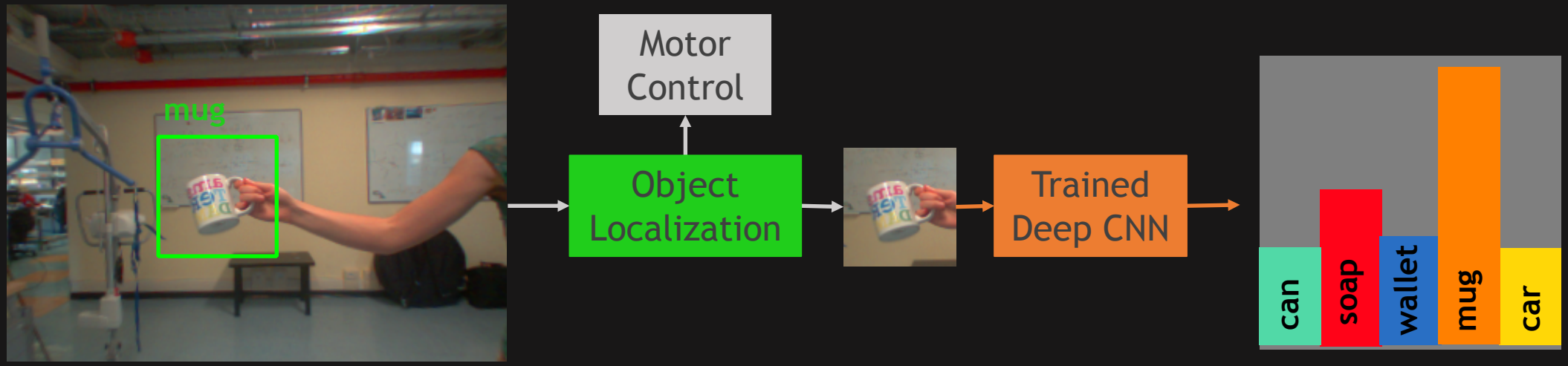
This afternoon

1. The data (we collected for you)

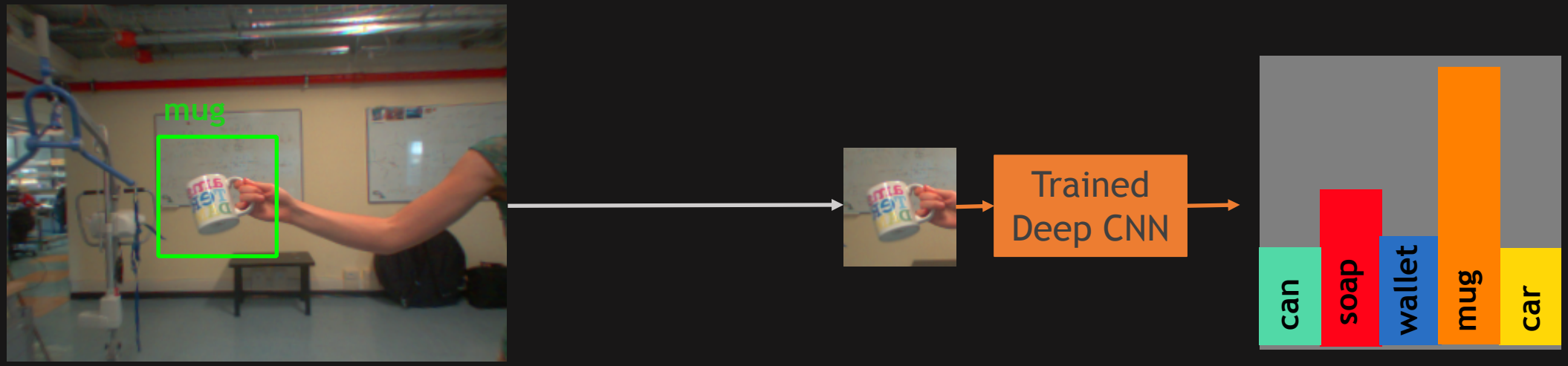
2. Fine-tuning a Deep CNN with Caffe:
    a. Tutorial
    b. Assignment

3. Model Deployment (in YARP)
    a. Tutorial (you can play with)

Model Deployment (in YARP)



Model Deployment (in YARP)



objectRecognizer: a YARP module wrapping Caffe

cv::Mat

yarp::sig::ImageOf<PixelRgb>

data

Convolution

Pooling

Convolution

Pooling

Fully Connected

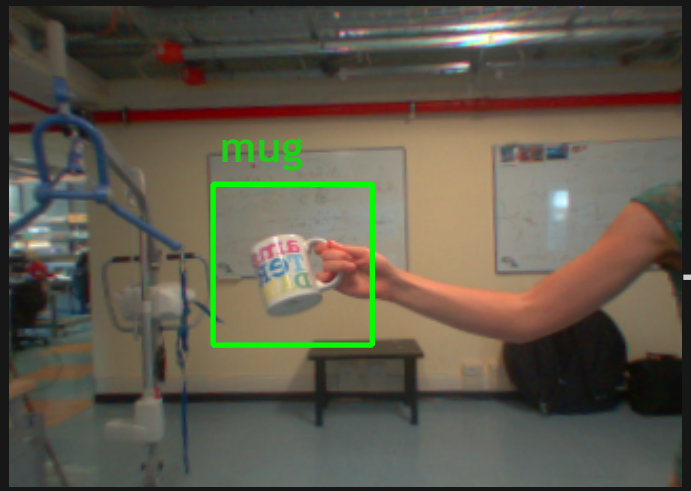
Fully Connected

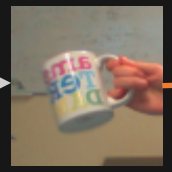
Output Predictions

dog (0.01)
cat (0.04)
boat (0.94)
bird (0.02)

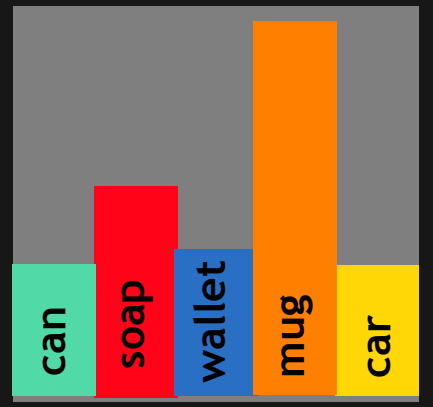
prob

yarp::sig::Vector





Trained Deep CNN



Deployment: Tutorial

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

(Compile and) configure the YARP module to use the model that you trained:

1. configure a deploy_MemoryData_caffenet.prototxt
2. configure a objectRecognizer.ini to use (i) the trained weights and (ii) the specified configuration file

3. run the application from yarpmanager

4. find mug, wallet, bottle and mouse instances and see if your model recognizes them!

deploy_MemoryData_caffenet.prototxt

- ✓ same structure as deploy.protoxt, but “MemoryData” layer (to read cv::Mat)
- ✓ pay attention to:
  - → set the path to mean image of the training set in the “MemoryData” layer
  - → check the name of the layers which you learned from scratch (fc8 → fc8N)
  - → set num_output (fc8N) accordingly to the task (num_output=4 for the assignment)

  you can copy from the deploy.prototxt ☺