LeNet-5 using PyTorch

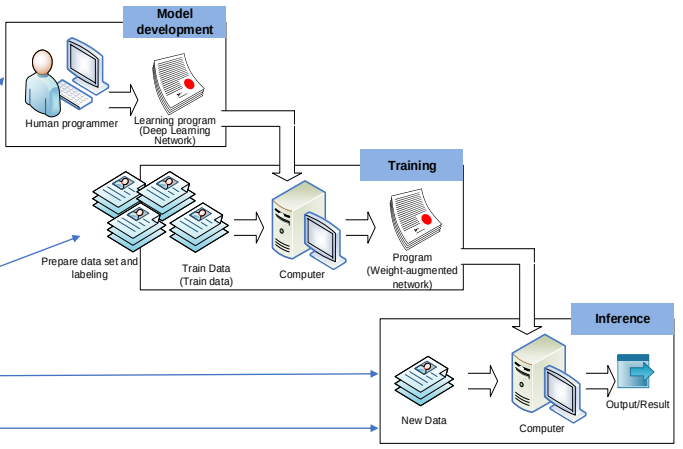
- Building model, training model, running model, and getting weights -

2021

Ando Ki, Ph.D.
adki@future-ds.com

Table of contents

- Preparing PyTorch
- Building LeNet-5 model
- Training LeNet-5
- Running LeNet-5
- Getting weights



What is PyTorch

- **PyTorch** is a machine learning Python library, developed by the Facebook AI research group.
  - Python package for machine learning, backed by Facebook

PYTORCH

Installing PyTorch

- Visit: https://pytorch.org/get-started/locally/
- Select your preferences and run the install command.
- Run the command
  - note 'conda' is required.

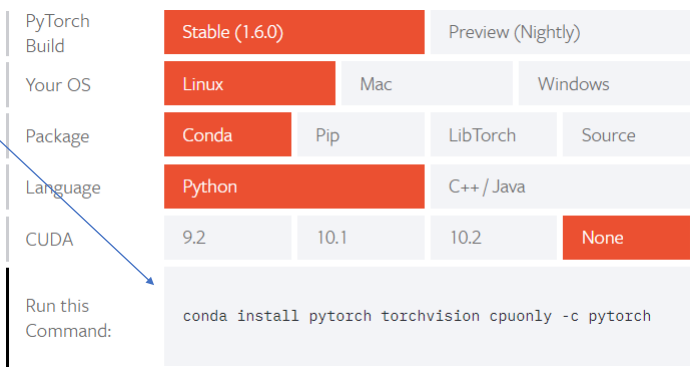
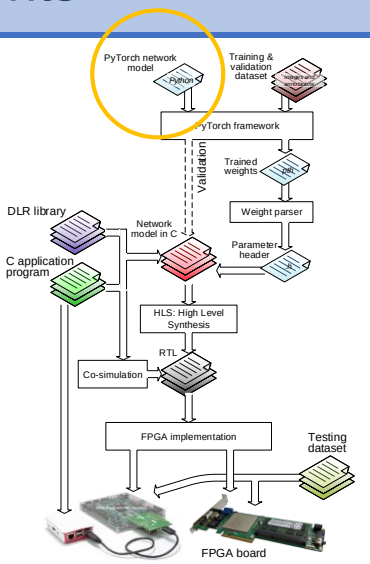


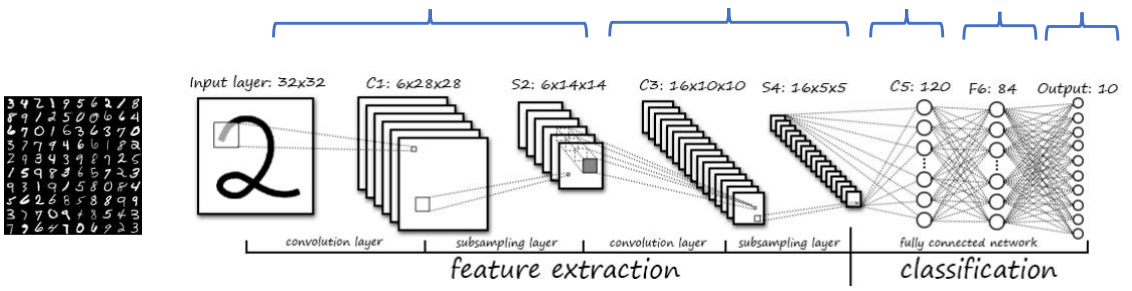
Table of contents

- Preparing PyTorch
- **Building LeNet-5 model**
- Training LeNet-5
- Running LeNet-5
- Getting weights

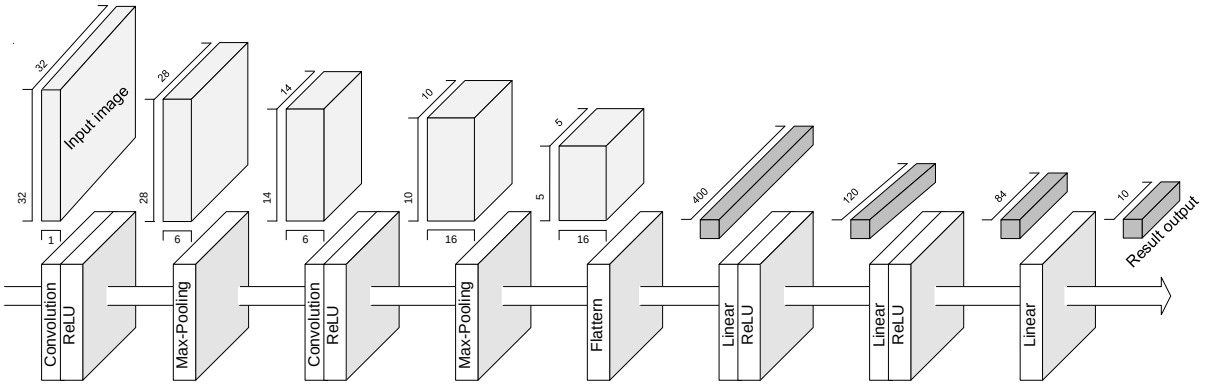


LeNet-5 for MNIST

- LeNet is one of the popular convolutional networks, and works well on digit classification tasks.
  - 1024 (32x32) inputs of black and white ➔ converted to floating number 0.0 ~ 1.0
  - 10 outputs representing digit 0 to 9



LeNet-5 network



Modified LeNet-5 network

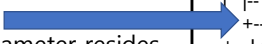
Layer		feature map (channel x W x H)		kernel size	stride	padding	Activation	Parameters (bias+weight)	Feature map
input	image	1	32x32	.	.	.	.	.	1,024
1	CONV	6	28x28	5x5	1	0	ReLU	6+150	4,704
2	MaxPOOL	6	14x14	2x2	2	0	.	0	1,176
3	CONV	16	10x10	5x5	1	0	ReLU	16+2,400	1,600
4	MaxPOOL	16	5x5	2x2	2	0	.	0	400
5	FC	.	120	.			ReLU	120+48,000	120
6	FC	.	84	.	.	.	ReLU	84+10,080	84
7	FC	.	10	.	.	.	softmax	10+840	10
								61,706	9,118

 # bias = # channel
 # weight for convolution= # in_channel x # out_channel x # kernel_size
 # feature map = # channel x # in_feature_map
 # weight for FC = # in_feature x # out_feature

How to get the project

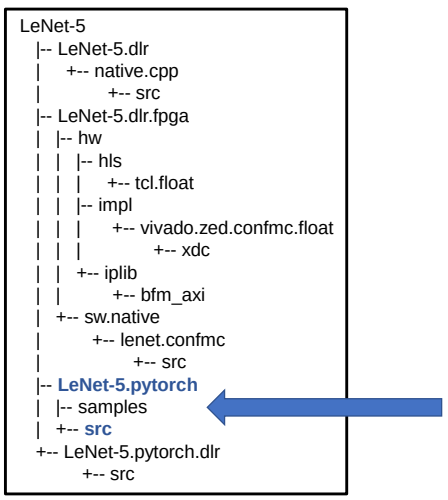
- Get a clone
  - $ sudo apt install git
  - $ git clone https://github.com/adki/DLR_Projects.git
  - $ cd DLR_Projects
- Go to 'LeNet-5/LeNet-5.python' directory

Where the trained parameter resides.



```
LeNet-5
  |-- LeNet-5.dlr
  |     +-- native.cpp
  |           +-- src
  |-- LeNet-5.dlr.fpga
  |   |-- hw
  |   |   |-- hls
  |   |   |     +-- tcl.float
  |   |   |-- impl
  |   |   |     +-- vivado.zed.confmc.float
  |   |   |           +-- xdc
  |   |   +-- iplib
  |   |         +-- bfm_axi
  |   +-- sw.native
  |         +-- lenet.confmc
  |               +-- src
  |-- LeNet-5.pytorch
  |   |-- samples
  |   +-- src
  +-- LeNet-5.pytorch.dlr
        +-- src
```


Directory structure

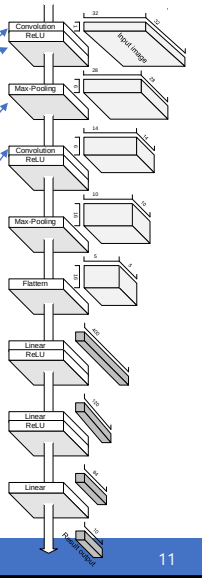


```
LeNet-5
  |-- LeNet-5.dlr
  |     +-- native.cpp
  |           +-- src
  |-- LeNet-5.dlr.fpga
  |   |-- hw
  |   |   |-- hls
  |   |   |     +-- tcl.float
  |   |   |-- impl
  |   |   |     +-- vivado.zed.confmc.float
  |   |   |           +-- xdc
  |   |   +-- iplib
  |   |         +-- bfm_axi
  |   +-- sw.native
  |         +-- lenet.confmc
  |               +-- src
  |-- LeNet-5.pytorch
  |   |-- samples
  |   +-- src
  +-- LeNet-5.pytorch.dlr
        +-- src
```

See 'lenet5_model.py' in 'src' directory.

LeNet-5 model (1/2)

```
import torch
import torch.nn as nn
import torch.nn.functional as F
class Lenet5Model(nn.Module):
    def __init__(self, input_channels=1):
        super(Lenet5Model, self).__init__()
        self.input_channels = input_channels
        self.model = nn.Sequential(
                    nn.Conv2d( in_channels=input_channels
                             , out_channels=6
                             , kernel_size=(5,5)
                             , stride=1
                             , padding=0
                             , bias=True),# nn.Conv2d(1,6,5)
                    nn.ReLU(),
                    nn.MaxPool2d( kernel_size=(2,2)
                                , stride=2),# nn.MaxPool2d(2)
                    nn.Conv2d( in_channels=6
                             , out_channels=16
                             , kernel_size=(5,5)
                             , stride=1
                             , padding=0
                             , bias=True),# nn.Conv2d(6,16,5)
                    nn.ReLU(),
```



LeNet-5 model (2/2)

```
                    nn.MaxPool2d( kernel_size=(2,2)
                                , stride=2),# nn.MaxPool2d(2)
                    nn.Flatten( start_dim=1
                              , end_dim=-1),
                    nn.Linear( in_features=16*5*5
                             , out_features=120 # nn.Linear(400, 120)
                             , bias=True),
                    nn.ReLU(),
                    nn.Linear( in_features=120
                             , out_features=84 # nn.Linear(84, 84)
                             , bias=True),
                    nn.ReLU(),
                    nn.Linear( in_features=84
                             , out_features=10 # nn.Linear(84, 10)
                             , bias=True)
                    )

    def forward(self, x=1): # for training
        y = self.model(x)
        return y

    def infer(self, x=1, softmax=True): # for inference
        y = self.forward(x)
        if softmax: y = F.softmax(y, dim=1)
        return y
```

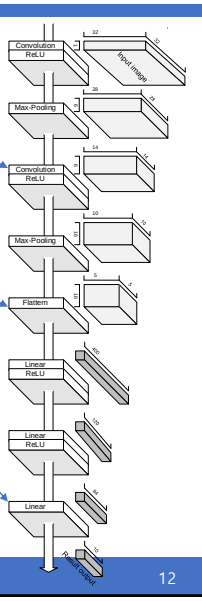
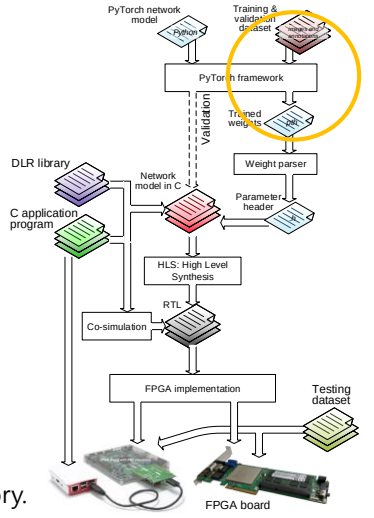


Table of contents

- Preparing PyTorch
- Building LeNet-5 model
- **Training LeNet-5**
- Running LeNet-5
- Getting weights




```
LeNet-5
  |-- LeNet-5.dlr
  |
  |-- LeNet-5.dlr.fpga
  |
  |-- LeNet-5.pytorch
  |   |-- samples
  |   +-- src
  +-- LeNet-5.pytorch.dlr
```

See 'lenet5_train.py' in 'src' directory.

LeNet-5 training (1/5)

```
if __name__ == '__main__':
    args = get_args()

    train_loader, test_loader = get_dataset( args ) # get dataset
    model, optimizer, cross_error = build_model( args ) # prepare network model
    model.train() # set the mode to train

    for epoch in range(args.epochs): # user specified num of epochs
        for idx, (train_x, train_label) in enumerate(train_loader): # over mini-bach
            model.train() # set the mode to train
            train_one_mini_batch(args, model, train_x, train_label, cross_error, optimizer)

        correct = 0; sum = 0
        for idx, (test_x, test_label) in enumerate(test_loader):
            model.eval() # set the mode to evaluate (not to train)
            c, s = evaluate_one_mini_batch(args, model, test_x, test_label)
            correct += c # accumulate the num of mached
            sum += s # accumulate the number of items

        accuracy = correct/sum # ratio of correct from sum
        print(f"epoch: {epoch}, accuracy: {accuracy}")

        if save_checkpoint(args, model, accuracy, epoch): break
```

LeNet-5 training (2/5)

```
def get_dataset( args ):
    train_dataset = mnist.MNIST( root='dataset.train'
                  , train=True
                  , download=True
                  , transform=transforms.Compose([
                               transforms.Resize((32, 32))
                              ,transforms.Grayscale(num_output_channels=args.input_channels)
                              ,transforms.ToTensor()]))
    test_dataset  = mnist.MNIST( root='dataset.test'
                  , train=False
                  , download=True
                  , transform=transforms.Compose([
                               transforms.Resize((32, 32))
                              ,transforms.Grayscale(num_output_channels=args.input_channels)
                                          ,transforms.ToTensor()]))
    train_loader  = DataLoader( train_dataset
                              , batch_size=args.batch_size
                              , num_workers=8)
    test_loader   = DataLoader( test_dataset
                              , batch_size=args.batch_size
                              , num_workers=8)
    return train_loader, test_loader
```


LeNet-5 training (3/5)

```
def build_model( args ):
    if args.pre_trained_type == 'none':
        model = Lenet5Model(args.input_channels)
    else:
        # load checkpoints if specified
    optimizer = SGD(model.parameters(), lr=args.learning_rate)
    cross_error = CrossEntropyLoss() # loss function
    return model, optimizer, cross_error
```


LeNet-5 training (4/5)

```
def train_one_mini_batch( args
                        , model
                        , images # input images
                        , labels # expected label for the input images
                        , cross_error # error function
                        , optimizer # otptimizer
                        ):
    """
    It runs a train on a mini-batch, which consists of a number of images.
    """
    predicts = model(images.float())
    error = cross_error(predicts, labels.long()) # CrossEntropyLoss(calculated, expected)
    optimizer.zero_grad()
    error.backward() # loss
    optimizer.step()
    return error
```


LeNet-5 training (5/5)

```
def evaluate_one_mini_batch( args
                           , model
                           , images # input images
                           , labels # expected label
                           ):
    """
    It runs an evaluation on a mini-batch, which consists of a number of images.
    """
    predicts = model(images.float()).detach()
    predicts_ys = np.argmax(predicts, axis=-1) # get id of max value
    matched = predicts_ys == labels
    correct = np.sum(matched.numpy(), axis=-1) # num of mached
    sum = matched.shape[0] # number of items (images) in the mini-batch
    return correct, sum
```


Running for training

```
adki@ando-pc: ~/work/seminars/202008_ETRI_AI_SOC/master/week3/c...
File  Edit  View  Search  Terminal  Help
[adki@ando-pc] source pytorch-ven/bin/activate
(pytorch-ven) [adki@ando-pc] python3 src/lenet5_train.py
idx: 0, error: 2.3028676509857178
idx: 100, error: 2.302647113800049
idx: 200, error: 2.3013064861297607
idx: 300, error: 2.3020970821380615
idx: 400, error: 2.299935817718506
idx: 500, error: 2.299577474594116
epoch: 0, accuracy: 0.1873
idx: 0, error: 2.2927770614624023

idx: 500, error: 1.4611560106277466
epoch: 64, accuracy: 0.9897
idx: 0, error: 1.4708176851272583
idx: 100, error: 1.4624897241592407
idx: 200, error: 1.4714397192001343
idx: 300, error: 1.465348482131958
idx: 400, error: 1.4611520767211914
idx: 500, error: 1.4611542224884033
epoch: 65, accuracy: 0.9902
(pytorch-ven) [adki@ando-pc]
```

$ make run.train

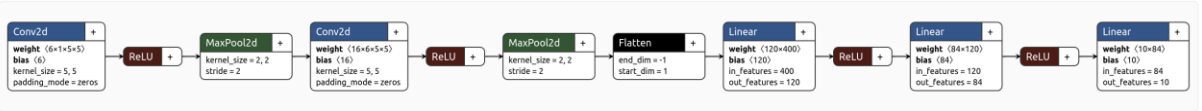
Start virtual environment

Running PyTorch

Stop when accuracy is larger than 0.99

Network from ONNX using Netron

- Load ‘mnist_model_final.pth’.



ONNX: Open Neural Network Exchange: https://onnx.ai/

https://lutzroeder.github.io/netron/

Monitoring progress using Tensorboard

- Make sure there is no 'tensorboard' server process still running.
- --> $ pgrep -u ${USER} -f tensorboard | xargs kill -9

- $ tensorboard --logdir=tensorboard serve
- $ google-chrome http://localhost:6006

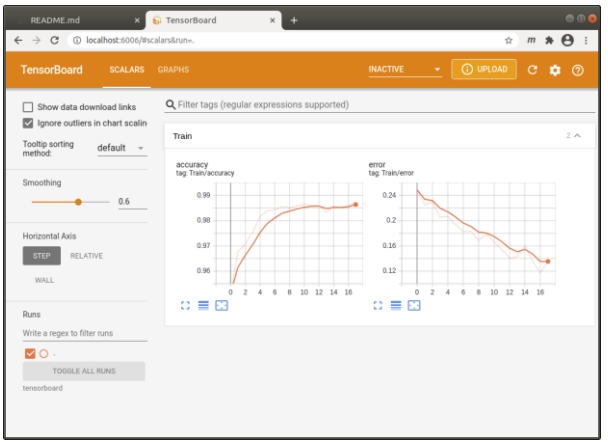
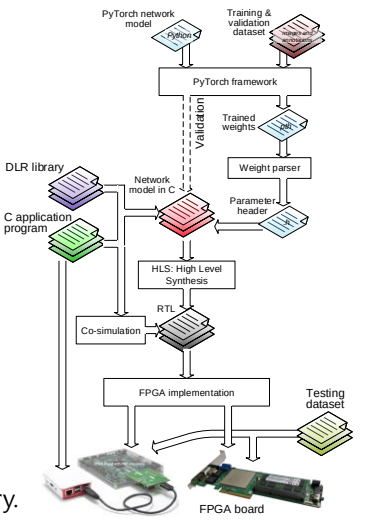


Table of contents

- Preparing PyTorch
- Building LeNet-5 model
- Training LeNet-5
- **Running LeNet-5**
- Getting weights

```
LeNet-5
  |-- LeNet-5.dlr
  |
  |-- LeNet-5.dlr.fpga
  |
  |-- LeNet-5.pytorch
  |   |-- samples
  |   +-- src
  +-- LeNet-5.pytorch.dlr
```

See 'lenet5_infer.py' in 'src' directory.

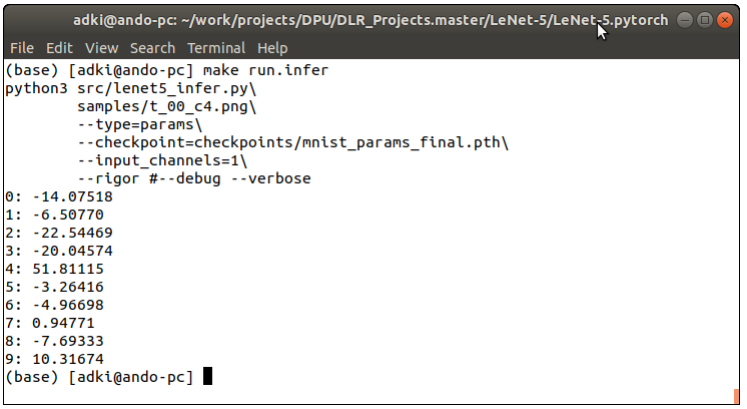


LeNet-5 inferencing

```
if __name__ == '__main__':
    args = get_args()
    extension = os.path.splitext(args.checkpoint)[1]
    if extension == '.pth':
        if args.type == 'model':
            model = torch.load(args.checkpoint)
        elif args.type == 'params':
            model = Lenet5Model(args.input_channels)
            model.load_state_dict(torch.load(args.checkpoint))
    model.eval() # not for train

    img = Image.open(args.image)
    img = img.resize((32,32), Image.ANTIALIAS)
    img.show() #img.save('x.png')
    if img.mode != 'L': # Not Luminance; convert to grayscale
        img = img.convert('L') # get luminance using Pillow convert()
        img = ImageOps.invert(img)
    data = tv.transforms.ToTensor()(img)
    data = data.view(-1,args.input_channels,32,32)
    result = model.infer(data, args.softmax).view(10)
    for idx in range(10):
        print(f"{idx}: {result[idx]:.5f}")
```


Running



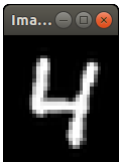
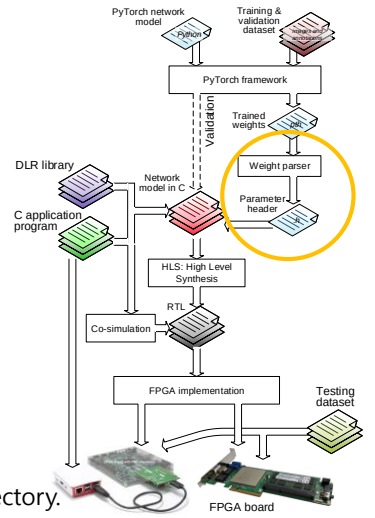


Table of contents

- Preparing PyTorch
- Building LeNet-5 model
- Training LeNet-5
- Running LeNet-5
- **Getting weights**




```
LeNet-5
  |-- LeNet-5.dlr
  |
  |-- LeNet-5.dlr.fpga
  |
  |-- LeNet-5.pytorch
  |   |-- samples
  |   +-- src
  +-- LeNet-5.pytorch.dlr
```

See 'lenet5_params.py' in 'src' directory.

Getting parameters

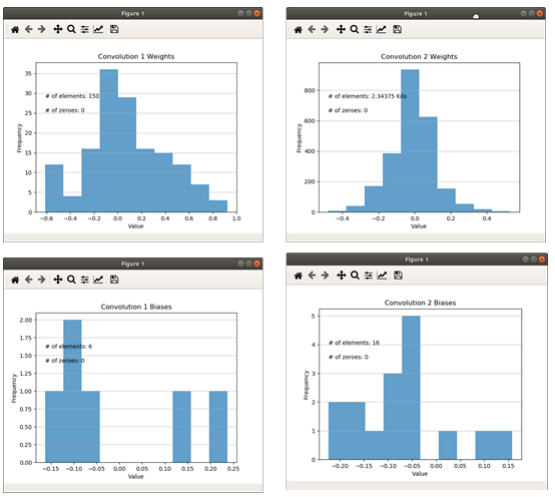
```
adki@ando-pc: ~/work/projects/DPU/DLR_Projects.master/LeNet-5/LeNet-5.pytorch
File  Edit  View  Search  Terminal  Help
(base) [adki@ando-pc] make run.params
if [ -f checkpoints/mnist_params_final.pth ]; then\
        python3 src/lenet5_params.py\
                --input_channels=1\
                --checkpoint checkpoints/mnist_params_final.pth\
                --txt x.txt\
                --bin y.bin\
                --header lenet5_params.h\
                --darknet lenet5.weights\
                --verbose;\
else\
        echo \"checkpoints/mnist_params_final.pth\" not found;\
fi
binary file "y.bin" size OK.
(base) [adki@ando-pc]
```

C header file: lenet5_params.h

```
const float conv1_bias[6]= { //  torch.Size([6])
-0.12715423, 0.31861544, 0.28352895, 0.0013063019, 0.00051533437, -0.024330929
};
const float conv1_weight[150]= { //  torch.Size([6, 1, 5, 5])
-0.2506813, 0.02748762, 0.22563115, 0.3041184, 0.2677712, -0.4858233,
...};
const float conv2_bias[16]= { //  torch.Size([16])
-0.20479624, -0.10206448, -0.088119015, 0.092432074, -0.0009077392, -0.13282152,
...};
const float conv2_weight[2400]= { //  torch.Size([16, 6, 5, 5])
...};
const float fc1_bias[120]= { //  torch.Size([120])
-0.0105780745, -0.00825386, 0.02470576, -0.036895536, 0.041986756, 0.0034808407,
...};
const float fc1_weight[48000]= { //  torch.Size([120, 400])
...};
const float fc2_bias[84]= { //  torch.Size([84])
0.016163042, -0.08301891, 0.0631961, 0.05435405, -0.08308794, 0.06956346,
...};
const float fc2_weight[10080]= { //  torch.Size([84, 120])
...};
const float fc3_bias[10]= { //  torch.Size([10])
-0.11194815, 0.088522755, -0.04713014, -0.11380346, 0.00060801016, -0.0901531,
...};
const float fc3_weight[840]= { //  torch.Size([10, 84])
...};
```


Analysis weights and biases

- $ make run.histogram



Analysis weights and biases