EC ENGR 199: Final Report Implementing a 3PXNet Compiler

FEIQIAN ZHU

SUPERVISION: WOJCIECH ROMASZKAN, TIANMU LI

NANOCAD LABORATORY, WINTER 2020

Contents

Intro: what is 3PXNet

Neural Network

- Pruned
- Packed
- Permuted

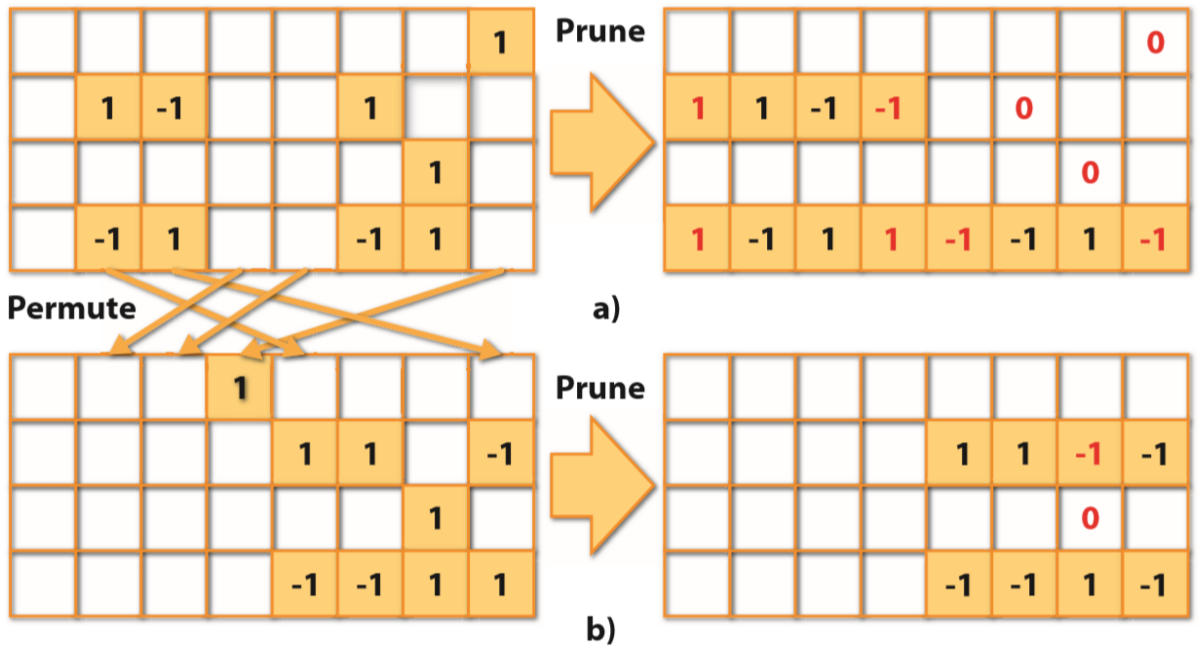


Fig. 2. Pruning with packing constraint of 4 bits a) without permutation, b) with permutation

Current 3PXNet Library

Training engine by Tianmu

- Use Pytorch library
- Pruned neural network
- Apply permutation and packing

Inference engine by Wojciech

- Dense or Sparse
- FC layer
- Conv layer
- Together with padding, pooling, batch normalization

What's left: the compiler

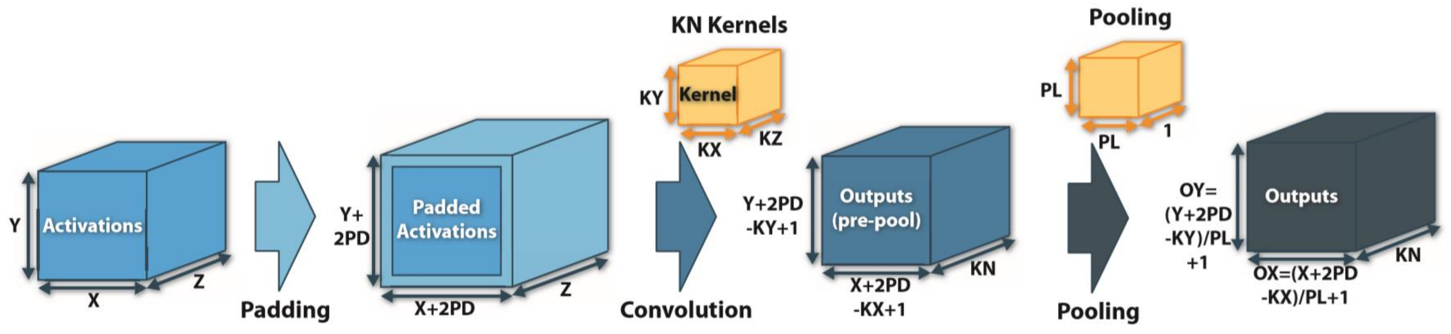
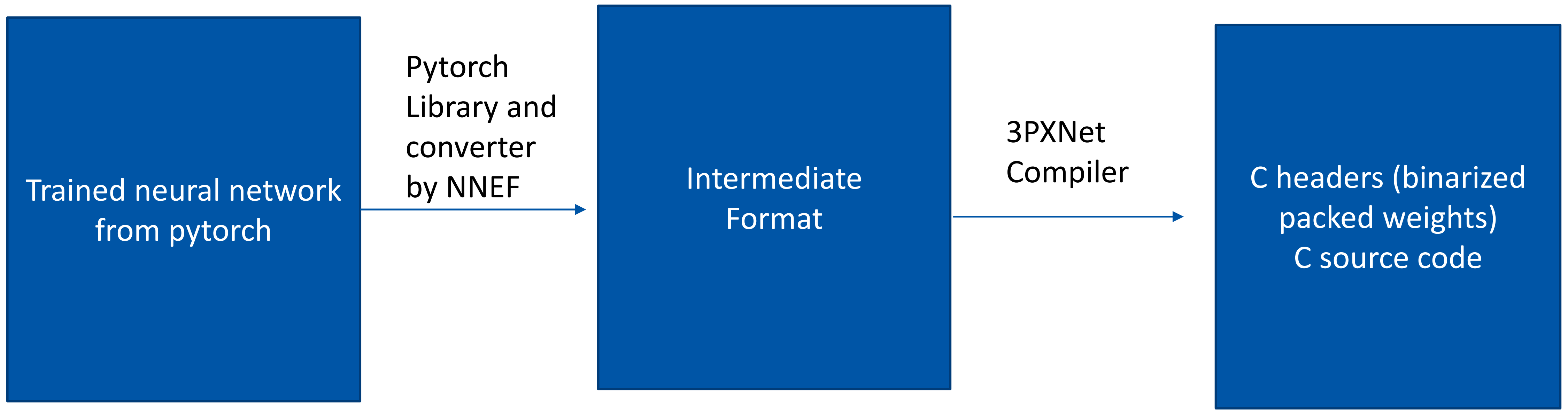


Fig. 5. A schematic view of a padded convolutional layer followed by pooling operation.

3PXNet Compiler - Idea

Automatic translation between pytorch and C

A bridge between training results and software implementation


Trained neural network from pytorch
Pytorch Library and converter by NNEF
Intermediate Format
3PXNet Compiler
C headers (binarized packed weights)
C source code

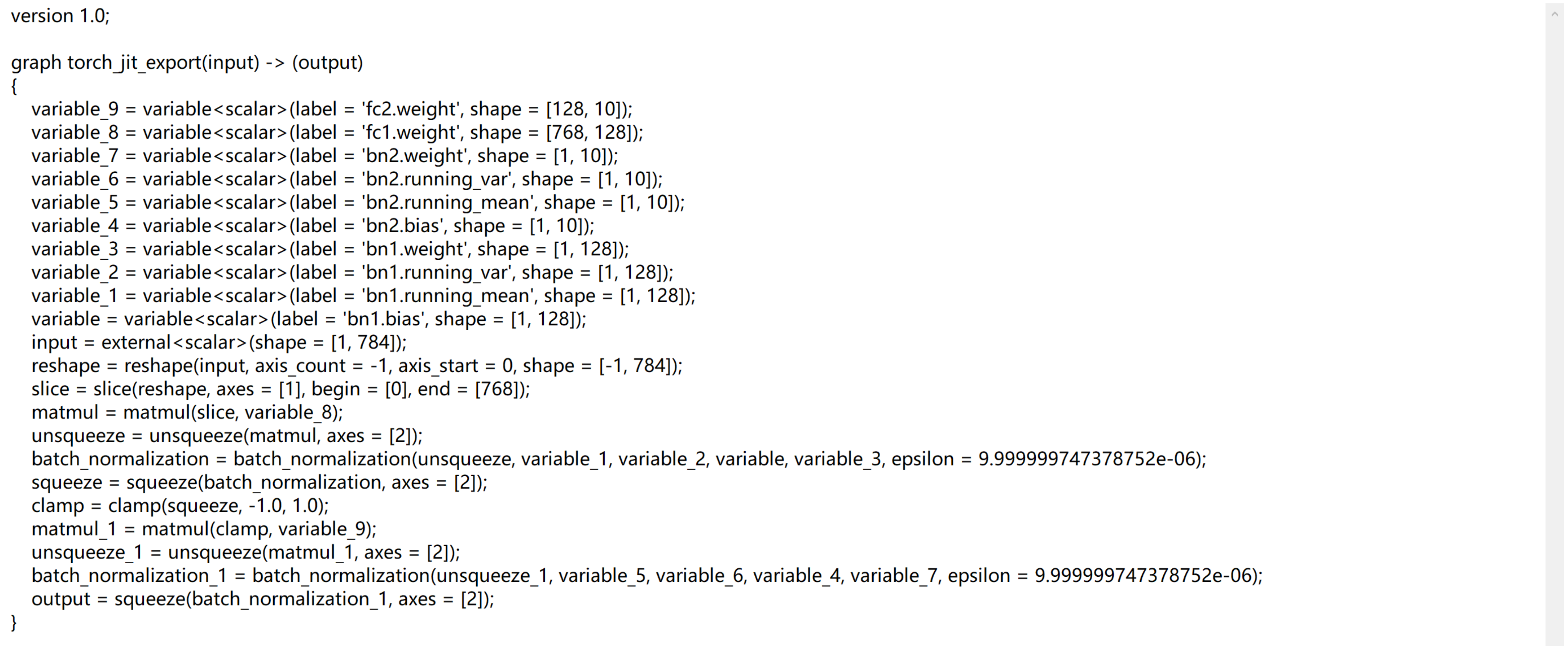
What to convert

ONNX or NNEF

NNEF:

- NNEF uses high level language to represent operations, which can be decomposed to several basic operations
- NNEF does not use concrete data types for activation tensors, while ONNX has various data types such as int and float.
  - “Instead, it describes quantization algorithms on a conceptual level (via real arithmetic) and lets the inference engine choose the appropriate representation for optimal execution.“
- NNEF separates the structure and data into different files.

NNEF: an example



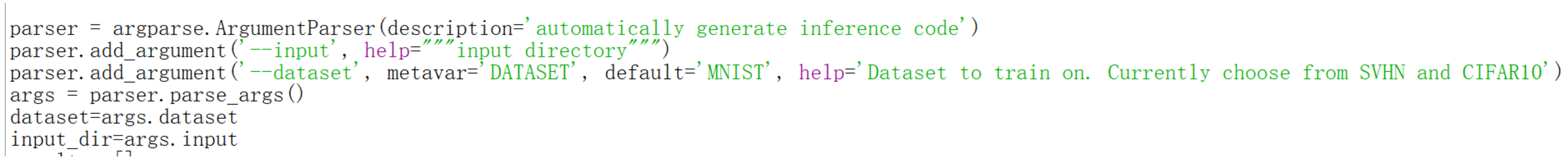
```
version 1.0;

graph torch_jit_export(input) -> (output)
{
    variable_9 = variable<scalar>(label = 'fc2.weight', shape = [128, 10]);
    variable_8 = variable<scalar>(label = 'fc1.weight', shape = [768, 128]);
    variable_7 = variable<scalar>(label = 'bn2.weight', shape = [1, 10]);
    variable_6 = variable<scalar>(label = 'bn2.running_var', shape = [1, 10]);
    variable_5 = variable<scalar>(label = 'bn2.running_mean', shape = [1, 10]);
    variable_4 = variable<scalar>(label = 'bn2.bias', shape = [1, 10]);
    variable_3 = variable<scalar>(label = 'bn1.weight', shape = [1, 128]);
    variable_2 = variable<scalar>(label = 'bn1.running_var', shape = [1, 128]);
    variable_1 = variable<scalar>(label = 'bn1.running_mean', shape = [1, 128]);
    variable = variable<scalar>(label = 'bn1.bias', shape = [1, 128]);
    input = external<scalar>(shape = [1, 784]);
    reshape = reshape(input, axis_count = -1, axis_start = 0, shape = [-1, 784]);
    slice = slice(reshape, axes = [1], begin = [0], end = [768]);
    matmul = matmul(slice, variable_8);
    unsqueeze = unsqueeze(matmul, axes = [2]);
    batch_normalization = batch_normalization(unsqueeze, variable_1, variable_2, variable, variable_3, epsilon = 9.999999747378752e-06);
    squeeze = squeeze(batch_normalization, axes = [2]);
    clamp = clamp(squeeze, -1.0, 1.0);
    matmul_1 = matmul(clamp, variable_9);
    unsqueeze_1 = unsqueeze(matmul_1, axes = [2]);
    batch_normalization_1 = batch_normalization(unsqueeze_1, variable_5, variable_6, variable_4, variable_7, epsilon = 9.999999747378752e-06);
    output = squeeze(batch_normalization_1, axes = [2]);
}
```

Compiler interface

Input: the directory of NNEF format neural network, which data set this neural network is for

Output: C header and source code



```
parser = argparse.ArgumentParser(description='automatically generate inference code')
parser.add_argument('--input', help="""input directory""")
parser.add_argument('--dataset', metavar='DATASET', default='MNIST', help='Dataset to train on. Currently choose from SVHN and CIFAR10')
args = parser.parse_args()
dataset=args.dataset
input_dir=args.input
```

Example: python converter.py --input=FC_Small.nnef --dataset=MNIST

Implementation Details

Read in weight matrix and write them into C headers

- Permutation takes place here

Compute batch normalization threshold and sign and also write them into C headers

Write source code

- For each operation described in the graph:
    - Include and define its weight
    - Determine whether it’s a FC or Conv layer and use corresponding library function

Load data set, using Tianmu’s load_dataset function

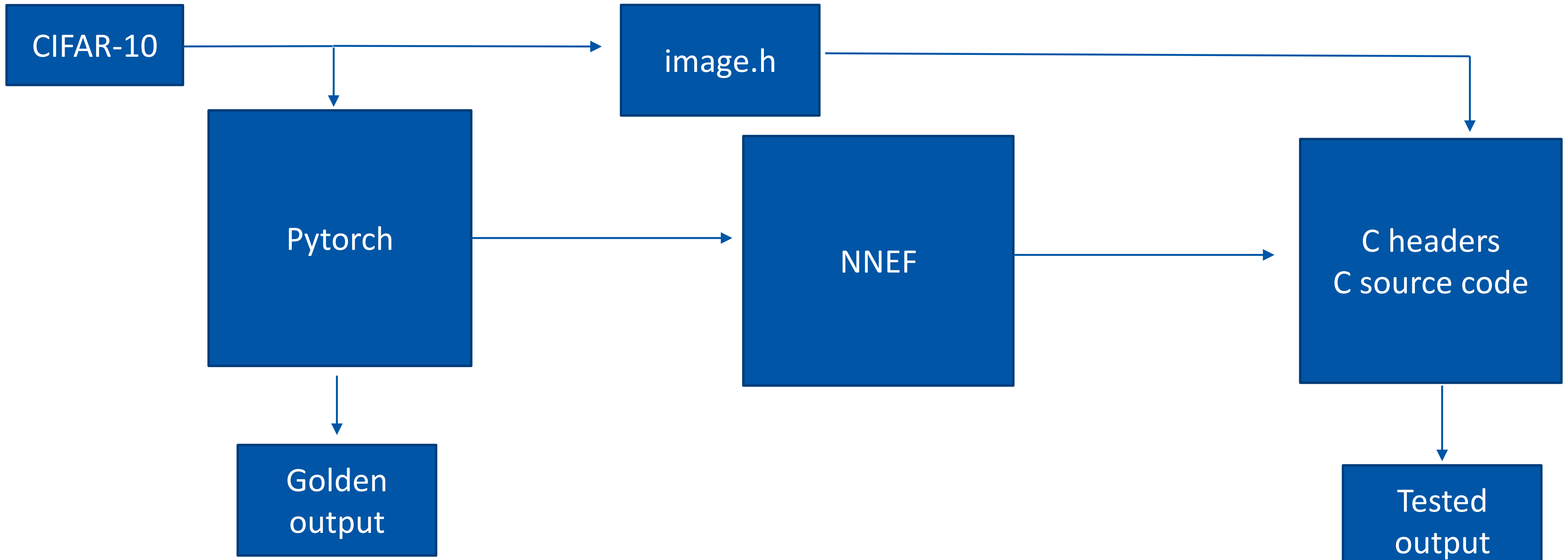
- Write images

Threshold Batch Norm

$$\mu-\beta/\gamma\ \surd(\sigma\hat{}2+\varepsilon))*\mathrm{sign}(\gamma)$$

Current work: Testing Infrastructure

Method:



Currently, there are bugs in reading the binary weight file.