HiDL
High-Performance Deep Learning
http://hidl.cse.ohio-state.edu

MVAPICH
http://mvapich.cse.ohio-state.edu

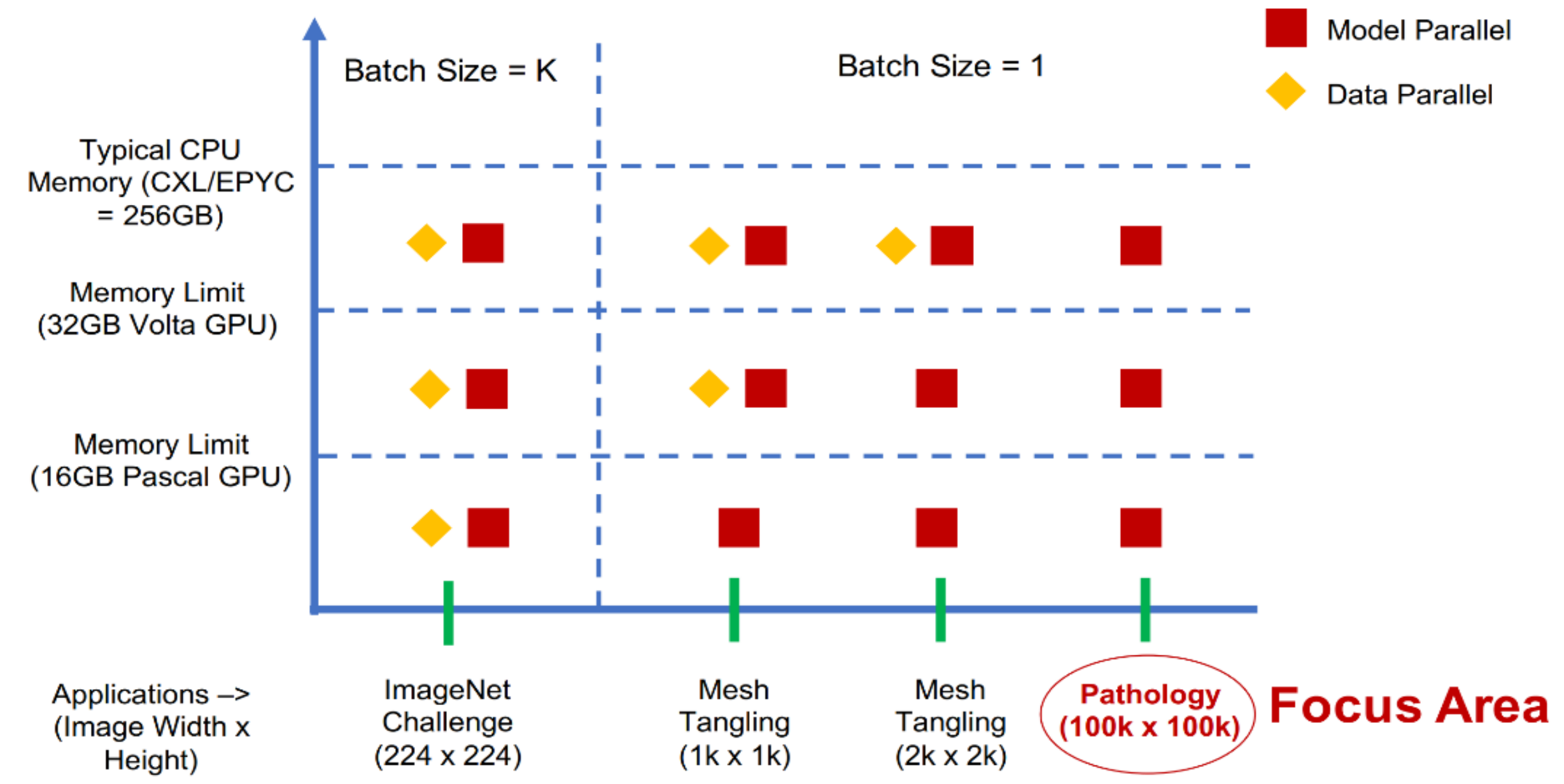
Scaling Distributed DNN Training for Segmentation Models on Large Images

Rayan Hamza, Nawras Alnaasan, Zhengqi Dong, and Arpan Jain
{hamza.23, alnaasan.1, dong.760, jain.575}@osu.edu

THE OHIO STATE UNIVERSITY

MOTIVATION

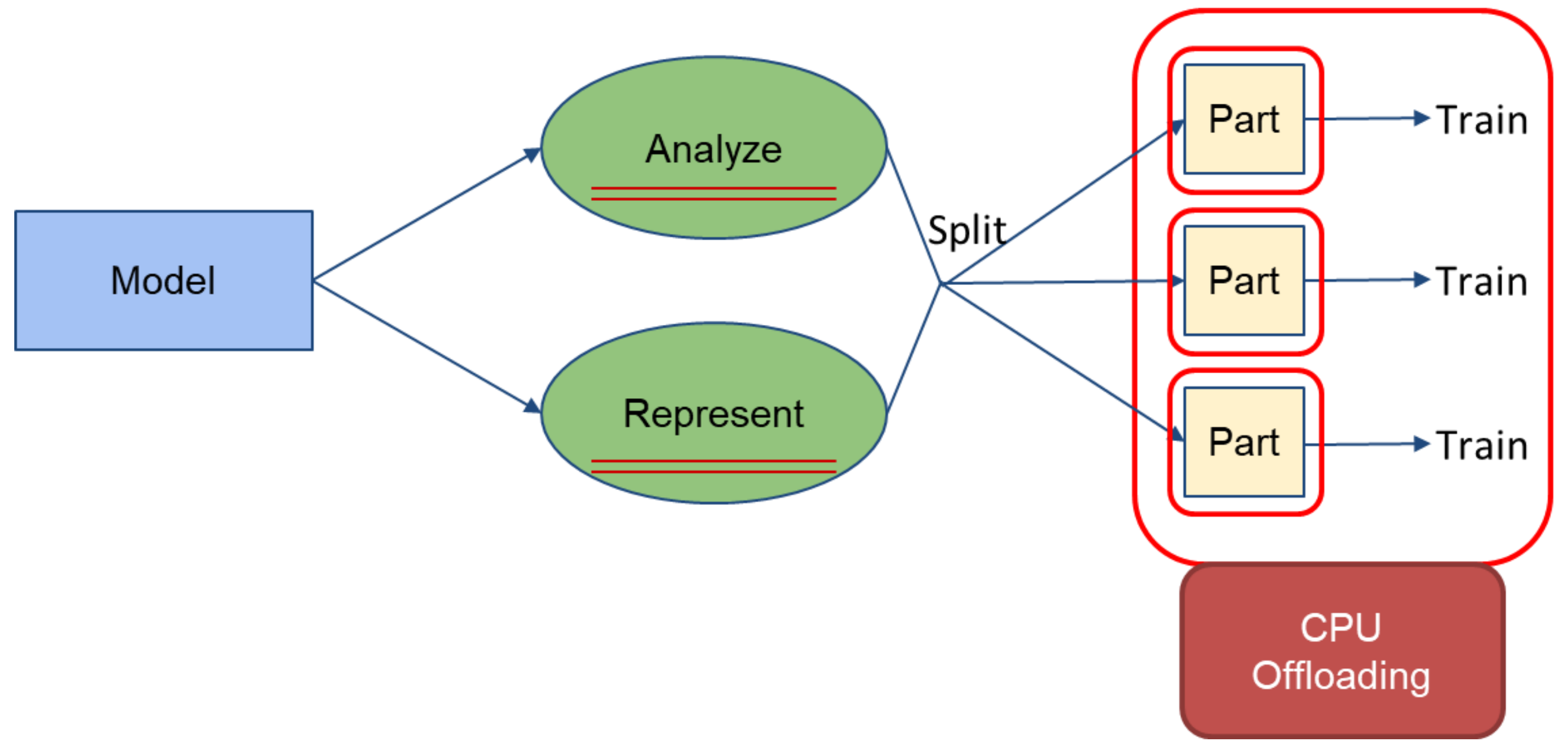
- Resurgence of Deep Learning (DL)
  - Availability of Large Datasets like ImageNet and massively-parallel modern hardware like NVIDIA GPUs
  - Emergence of DL frameworks (Caffe, TensorFlow, PyTorch, etc.)
- Existing DL frameworks cannot train large Deep Neural Networks on very-large images like WSI slides in Digital Pathology
  - GPU Memory is limited so large input images makes DNN model out-of-core (***Single GPU/node is not enough!)***
  - ***Model Parallelism can be used but performance is questionable!***



RESEARCH CHALLENGES

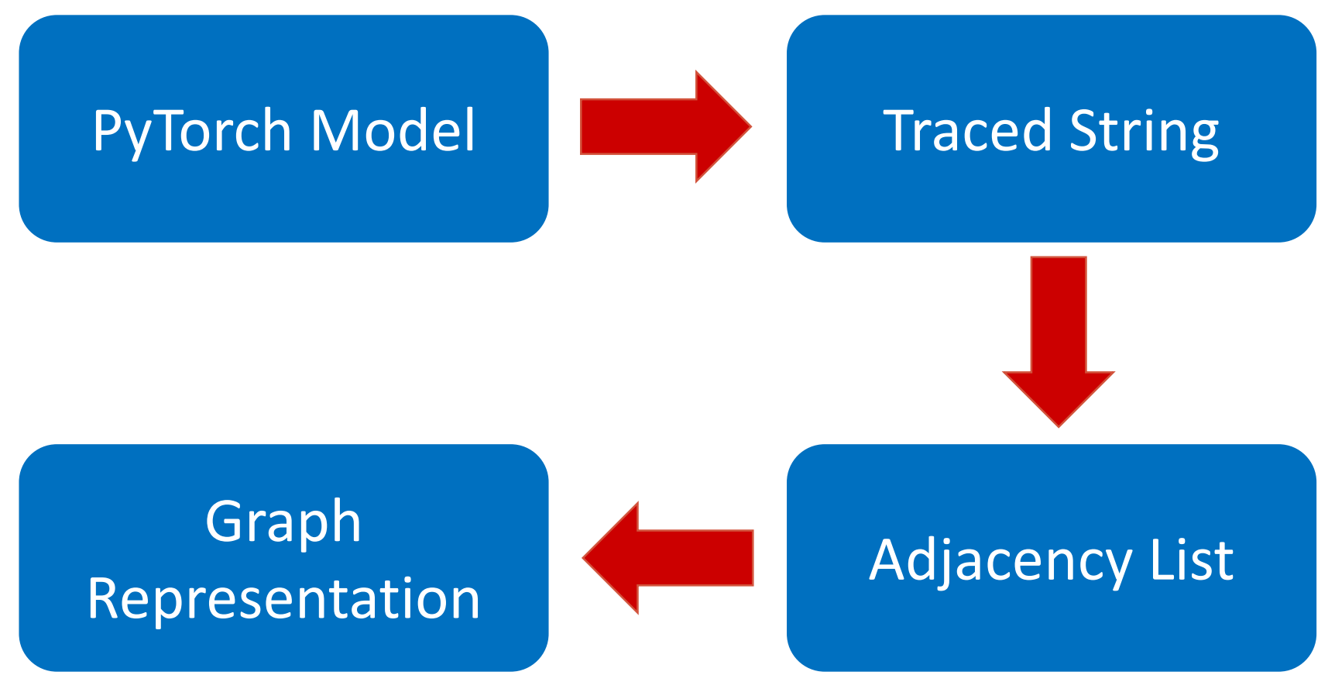
- Use analytical models to estimate execution time for a model split to efficiently split the DNNs across multiple GPUs
- Use PyTorch's Model and API to understand data flow in DNNs written in PyTorch to implement user transparent model-splitting
- Use CPU offloading mechanism to optimize GEMS-MASTER design

PROPOSED FRAMEWORK

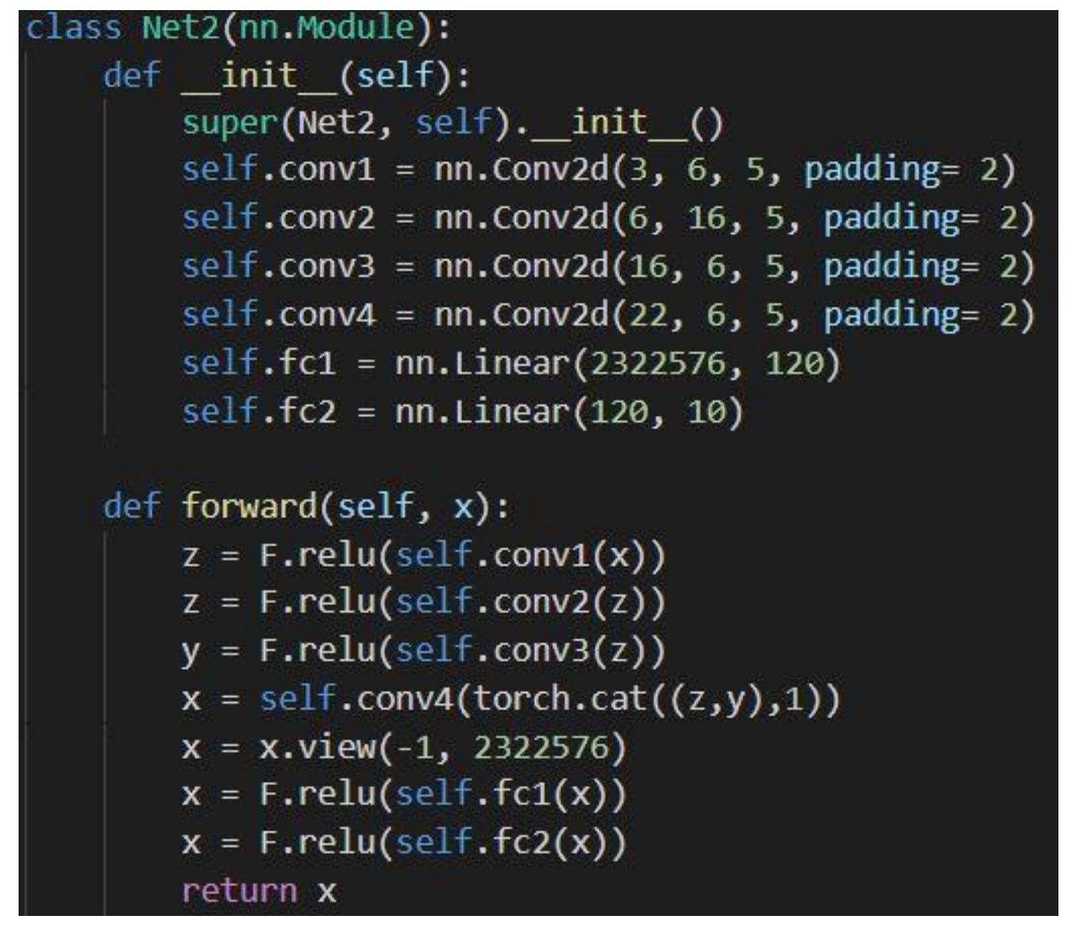

Model
Analyze
Represent
Split
Part
Train
Part
Train
Part
Train
CPU Offloading

NETWORK REPRESENTATION

REPRESENTATION MODULE

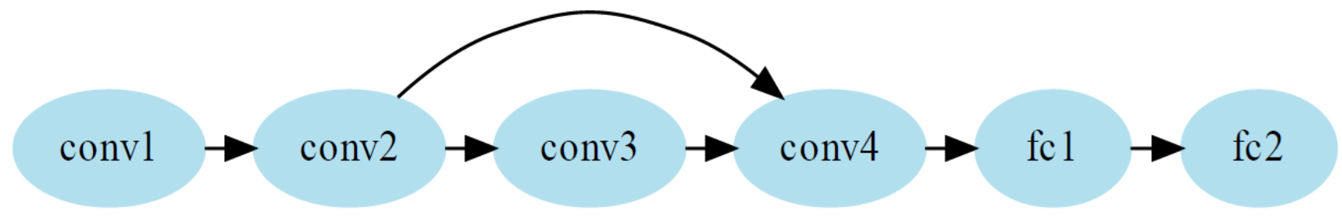

PyTorch Model
Traced String
Adjacency List
Graph Representation

- *No direct method to find layer/block connections in model in PyTorch*
- *Using torch.jit.trace can get a string representation of the entire forward function*
- *Parse string representation and create an adjacency list.*
- *Visualize graph from adjacency list*



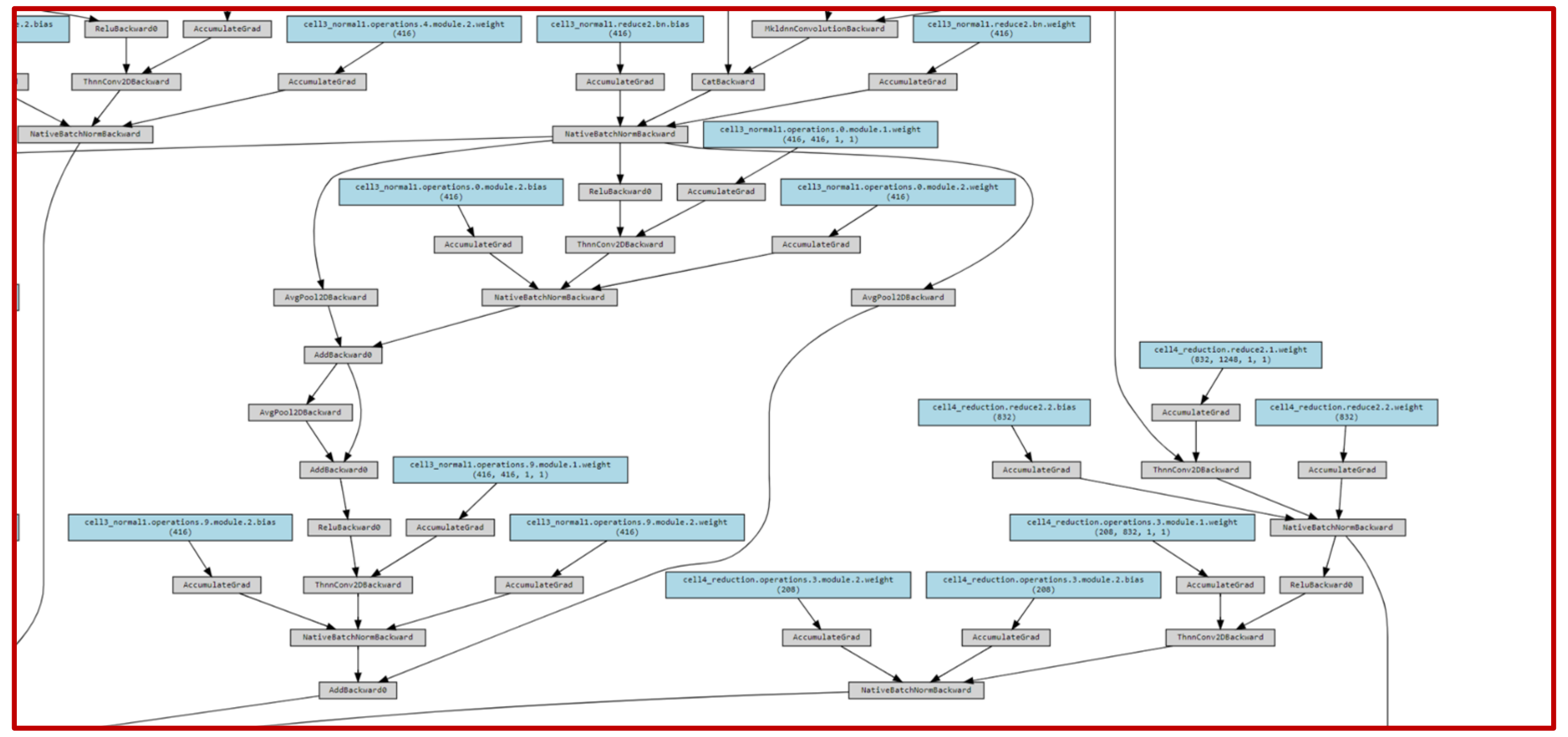
```
class Net2(nn.Module):
    def __init__(self):
        super(Net2, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5, padding= 2)
        self.conv2 = nn.Conv2d(6, 16, 5, padding= 2)
        self.conv3 = nn.Conv2d(16, 6, 5, padding= 2)
        self.conv4 = nn.Conv2d(22, 6, 5, padding= 2)
        self.fc1 = nn.Linear(2322576, 120)
        self.fc2 = nn.Linear(120, 10)

    def forward(self, x):
        z = F.relu(self.conv1(x))
        z = F.relu(self.conv2(z))
        y = F.relu(self.conv3(z))
        x = self.conv4(torch.cat((z,y),1))
        x = x.view(-1, 2322576)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        return x
```

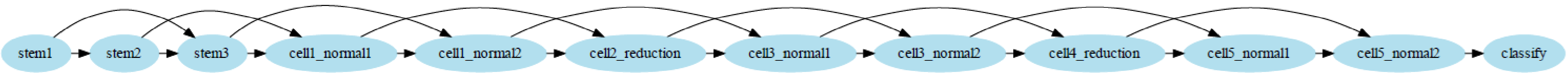
Interconnected CNN Model



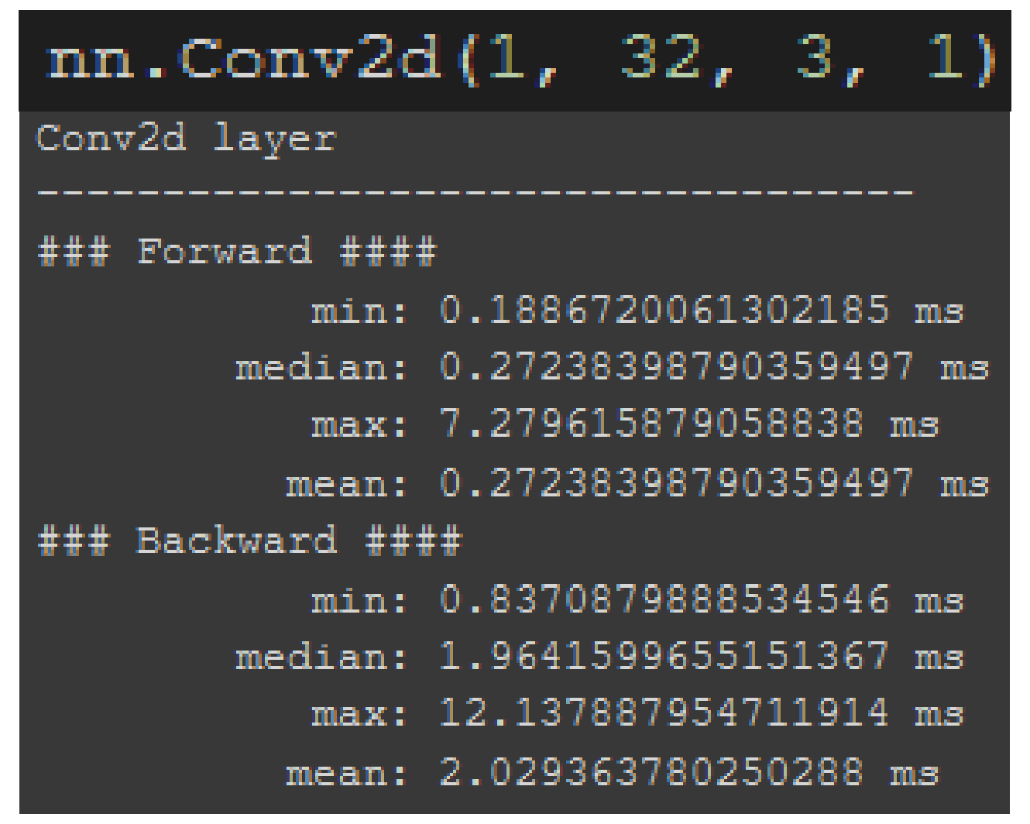
SOLUTION COMPARISON

- *Graphing packages (TorchViz):*
  - *No one-to-one mapping with layers*
  - *Introduces nodes for weights and biases*
  - *Shows graph for backward propagation*
  - *No block-level abstraction*
- *Solution: simpler block-level representation with recursion capabilities on each block.*

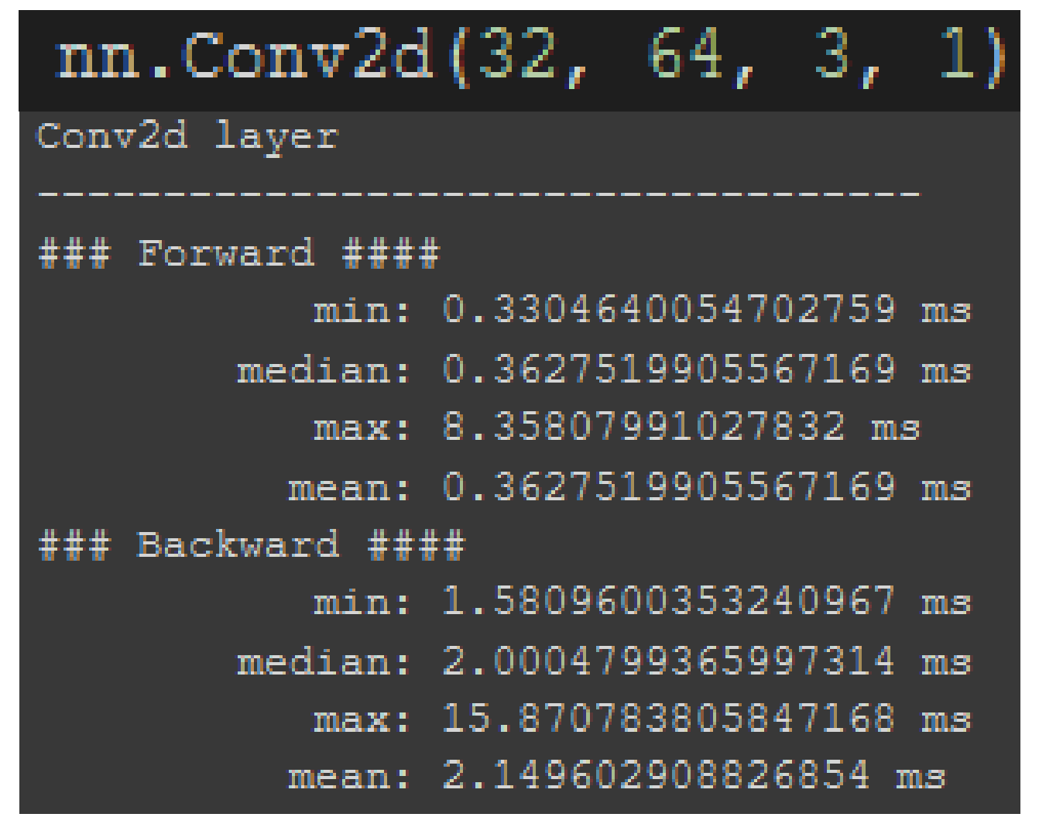




ANALYTICAL MODEL

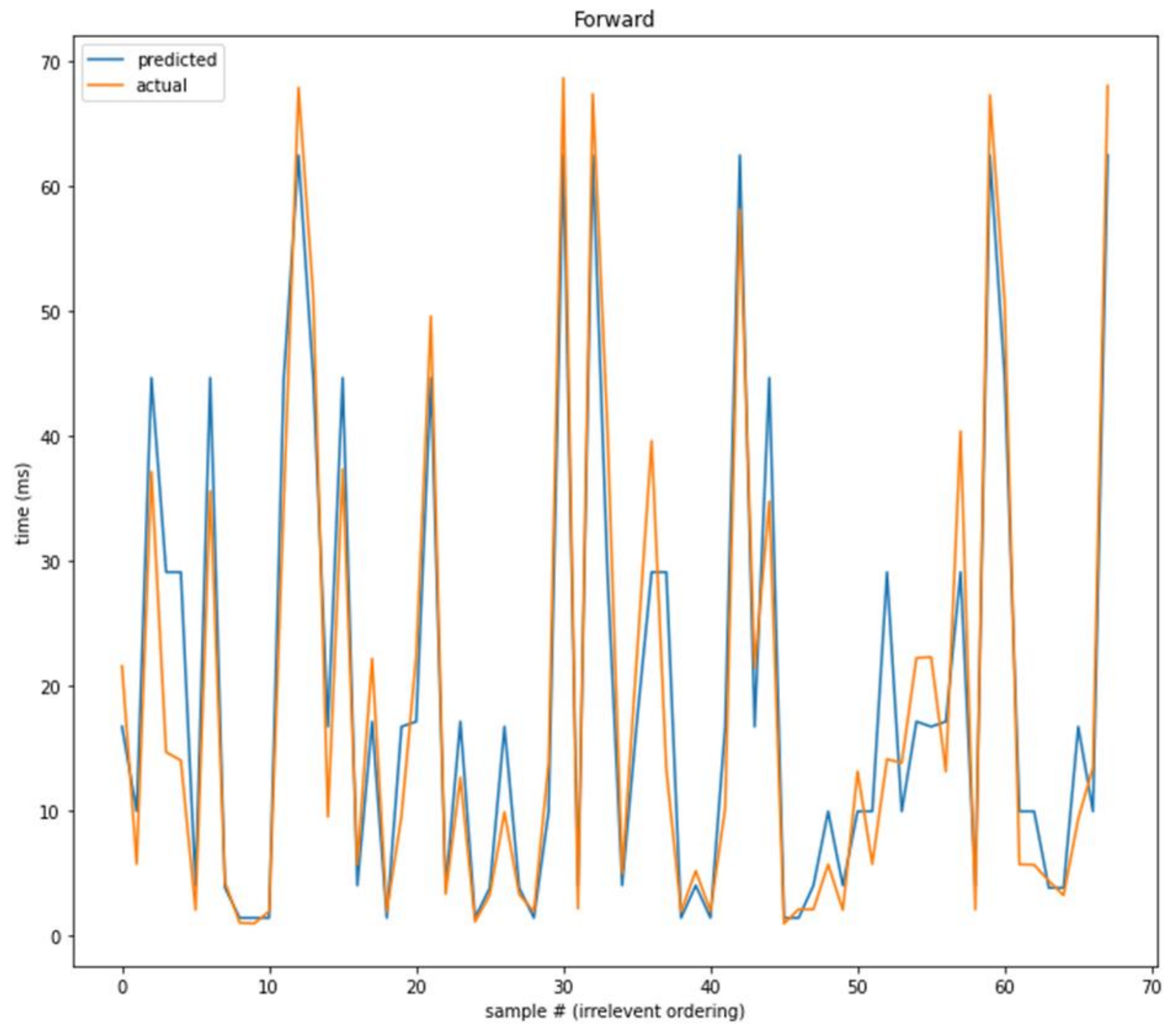


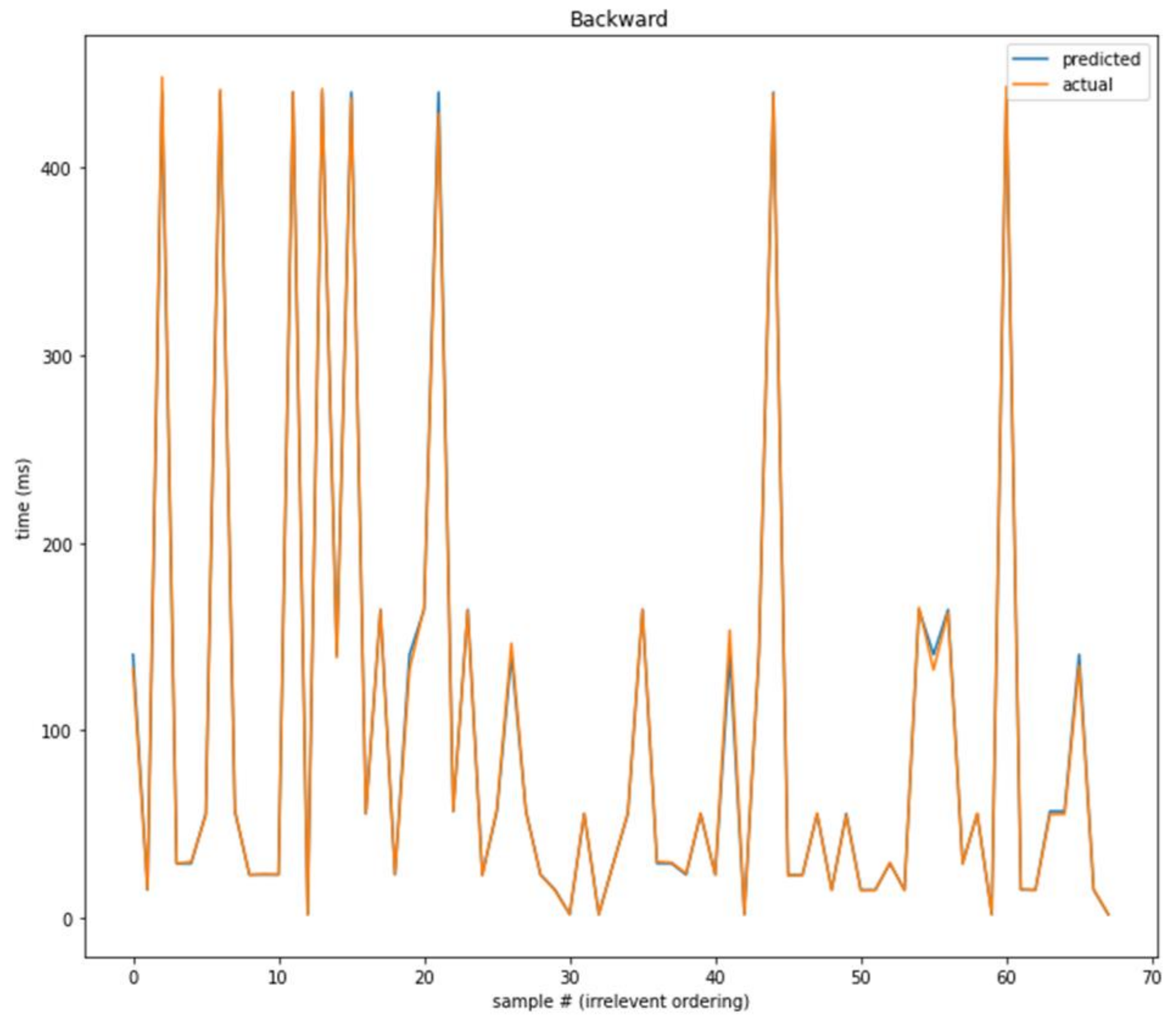
```
nn.Conv2d(1, 32, 3, 1)
Conv2d layer
-----------------------------------
### Forward ####
           min: 0.1886720061302185 ms
        median: 0.27238398790359497 ms
           max: 7.279615879058838 ms
          mean: 0.27238398790359497 ms
### Backward ####
           min: 0.8370879888534546 ms
        median: 1.9641599655151367 ms
           max: 12.137887954711914 ms
          mean: 2.029363780250288 ms
```



```
nn.Conv2d(32, 64, 3, 1)
Conv2d layer
-----------------------------------
### Forward ####
           min: 0.3304640054702759 ms
        median: 0.3627519905567169 ms
           max: 8.35807991027832 ms
          mean: 0.3627519905567169 ms
### Backward ####
           min: 1.5809600353240967 ms
        median: 2.0004799365997314 ms
           max: 15.870783805847168 ms
          mean: 2.149602908826854 ms
```

- *Time estimation of the convolutional layer (nn.Conv2D) defined by in/out channels, kernel size, and batch size.*
- *Calculate basic stats (min, median, max, mean) by running test models.*
- *Predict the time of forward/backward propagation using multivariate polynomial curve fitting.*





CPU OFFLOADING OF OUT-OF-CORE MODELS

Approaches	VGG19 (sky-k80)	VGG19 (bdw-v100)	AlexNet (bdw-v100)	ResNet50 (bdw-v100)	InceptionV3 (bdw-v100)
Baseline on GPU	94.4539	30.0673	24.1323	26.1702	32.2195
Naïve CPU-offloading	179.9194	150.5116	42.2092	113.5793	124.2807
with pin-memory	194.5803	166.7242	47.4363	110.0957	120.2433
non-blocking	179.8007	153.9791	41.5072	113.6041	123.1403
pin-memory and non-blocking	190.4785	160.9983	47.5516	109.3674	119.2649

Time in sec for 20 epochs on Hymenoptera dataset

Model	Number of Parameter
VGG19	139,578,434
AlexNet	57,012,034
ResNet50	23,512,130
InceptionV3	24,348,900

- *Moving some memory from GPU to CPU during training*
- *I/O communication overhead*
- *Affected by hardware architecture, number of parameter.*

SUMMARY OF CONTRIBUTIONS

- Proposed framework for model splitting based on the collaboration between an analytical mode and a network representation technique.
- Designed an analytical model for convolutional layers to estimate execution time for a model split based on in/out channels, kernel size, and batch size.
- Designed a recursive module to represent DNN models as adjacency lists and graphs of blocks/layers and connections between them.
- Analyzed and presented the affects of CPU offloading based on model and hardware used.