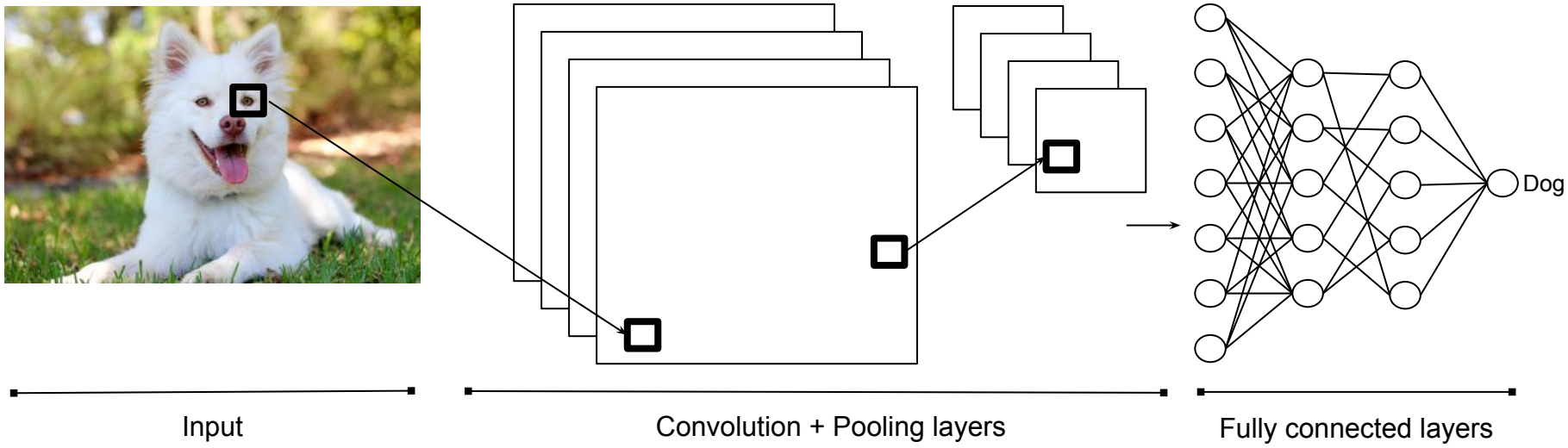
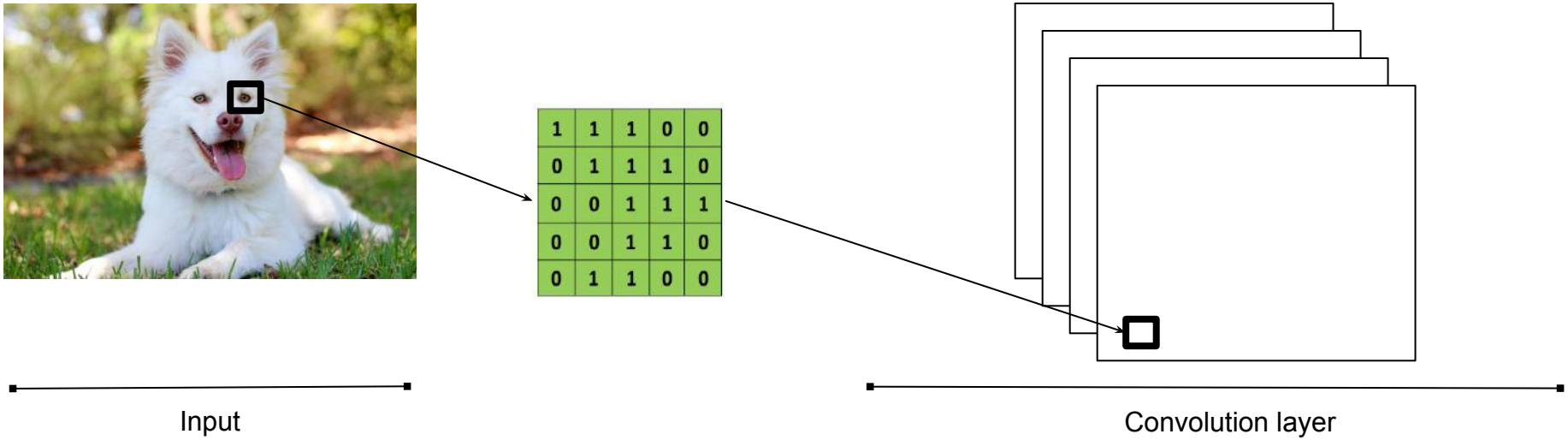
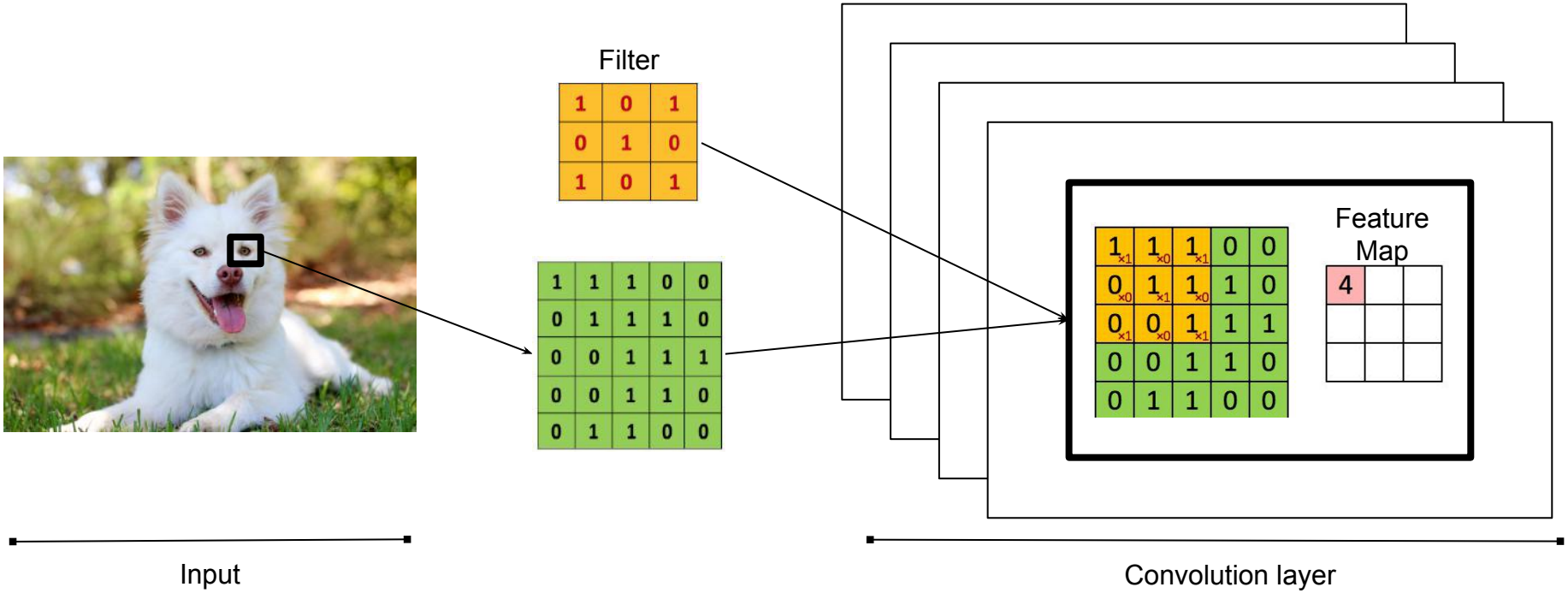
GIST: Efficient Data Encoding for Deep Neural Network Training

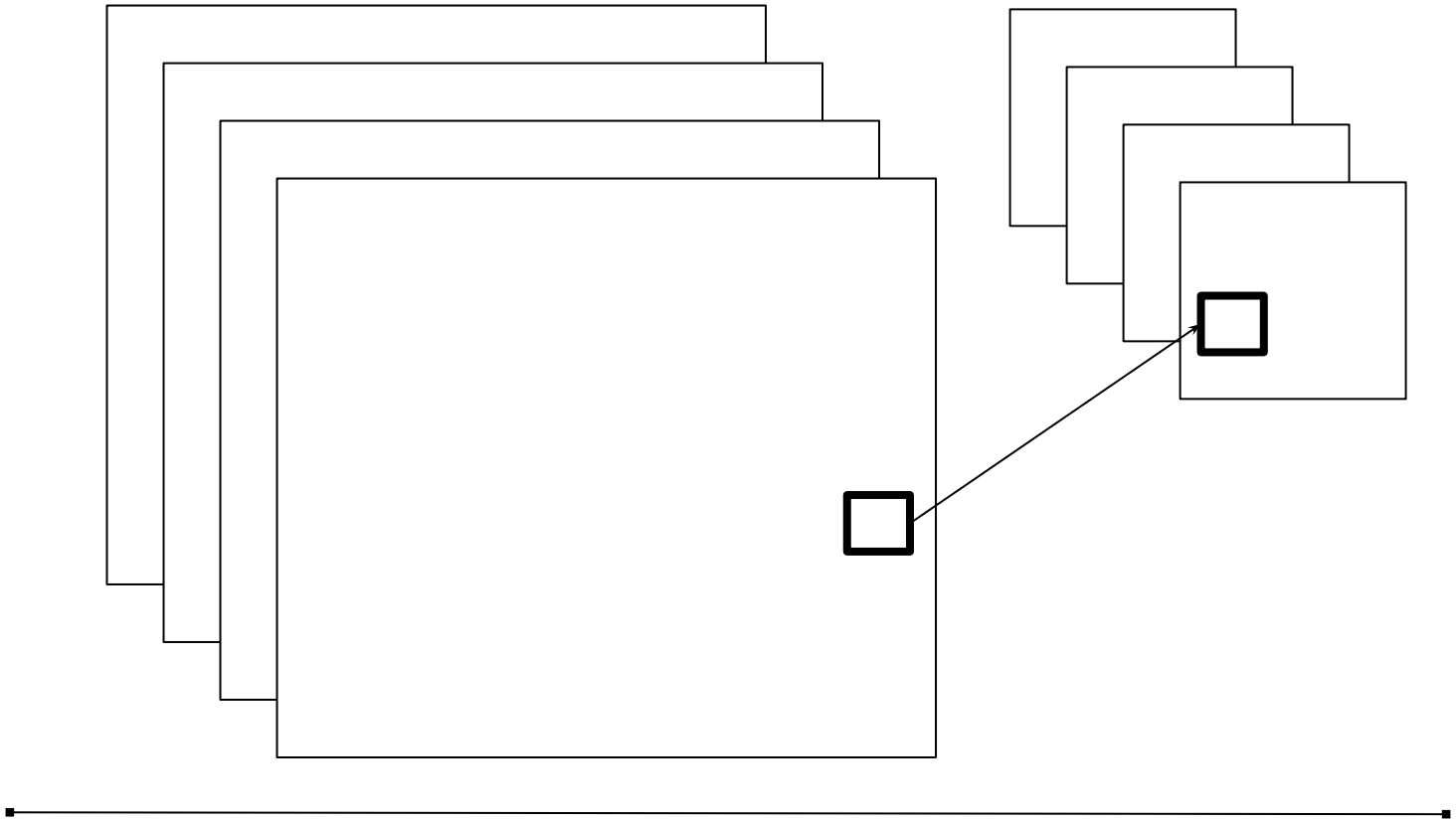
Animesh Jain, Amar Phanishayee, Jason Mars, Lingjia Tang, and Gennady Pekhimenko
Presented by: Peniel Argaw, Elizabeth Bondi, Hamish Nicholson, and Sam Lurye

Background

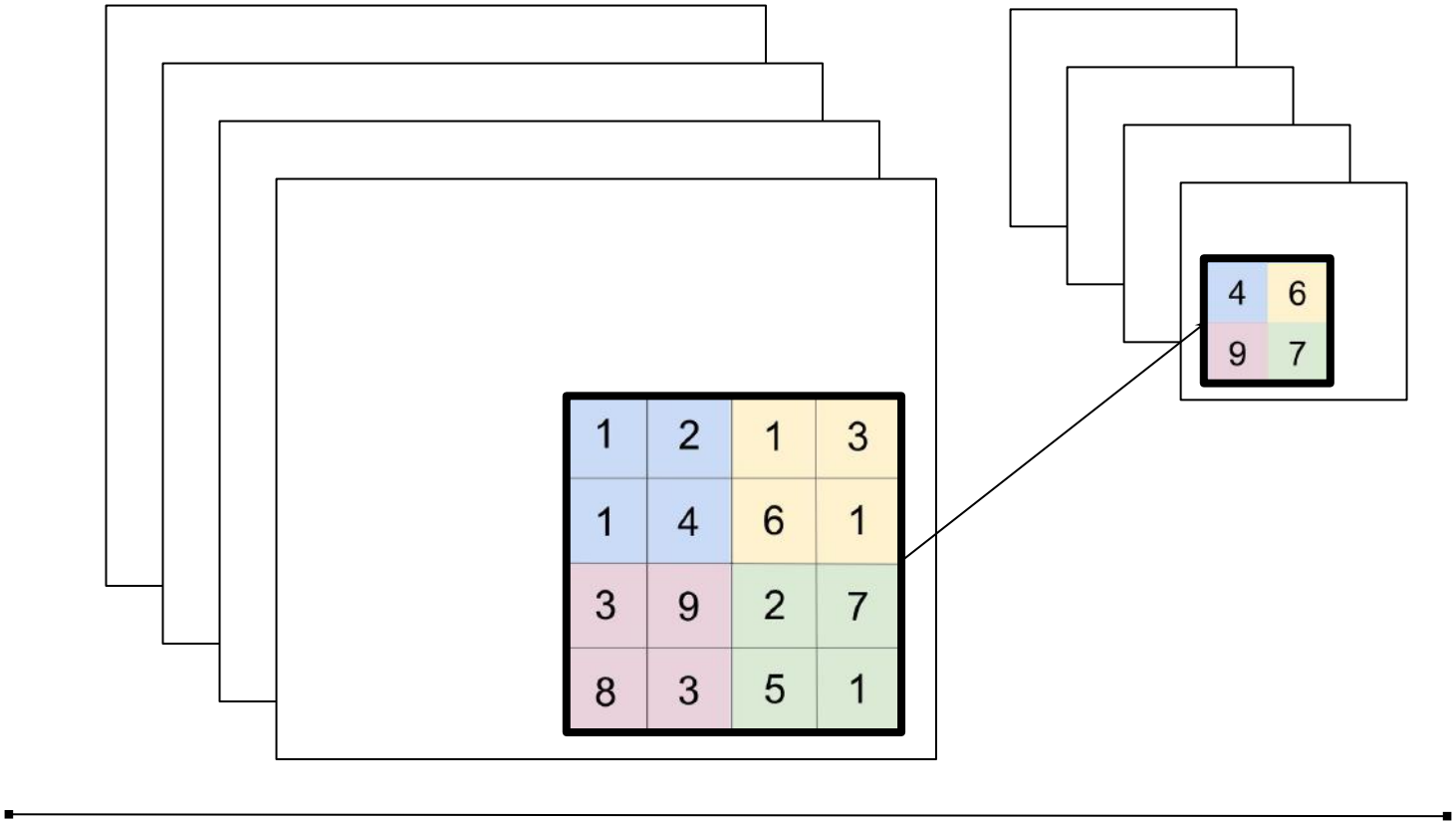

Dog
Input
Convolution + Pooling layers
Fully connected layers


1 1 1 0 0
0 1 1 1 0
0 0 1 1 1
0 0 1 1 0
0 1 1 0 0
Input
Convolution layer

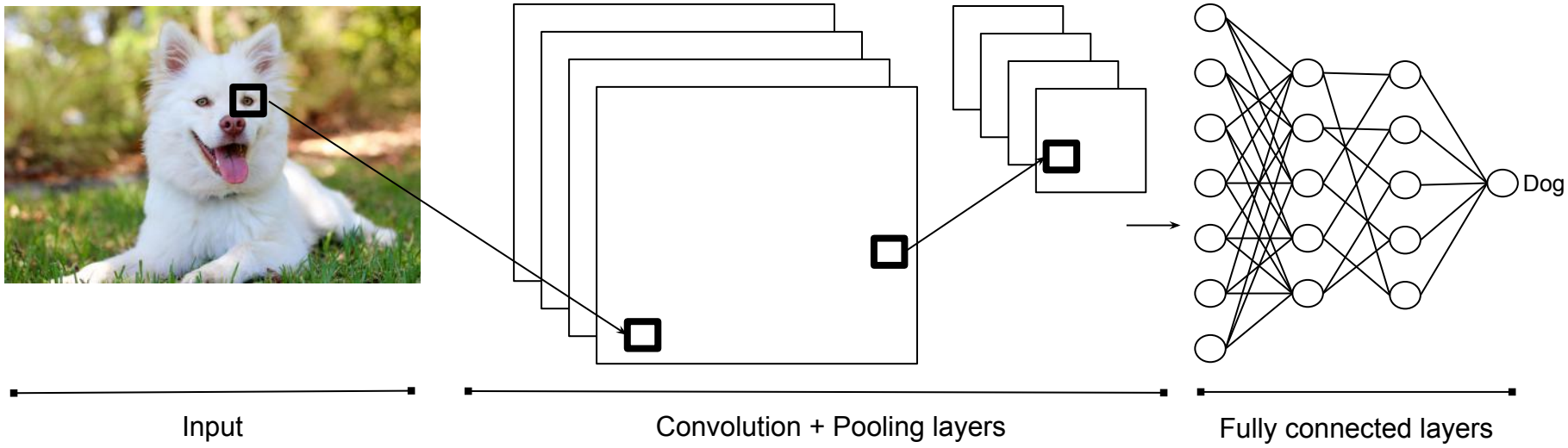

Filter
1 0 1
0 1 0
1 0 1
1 1 1 0 0
0 1 1 1 0
0 0 1 1 1
0 0 1 1 0
0 1 1 0 0
1 1 1 0 0
0 1 1 1 0
0 0 1 1 1
0 0 1 1 0
0 1 1 0 0
Feature Map
4
Input
Convolution layer



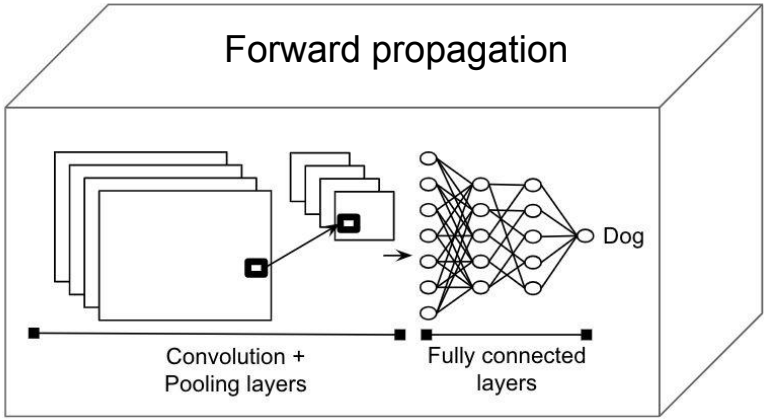
Pooling layer (max pooling)



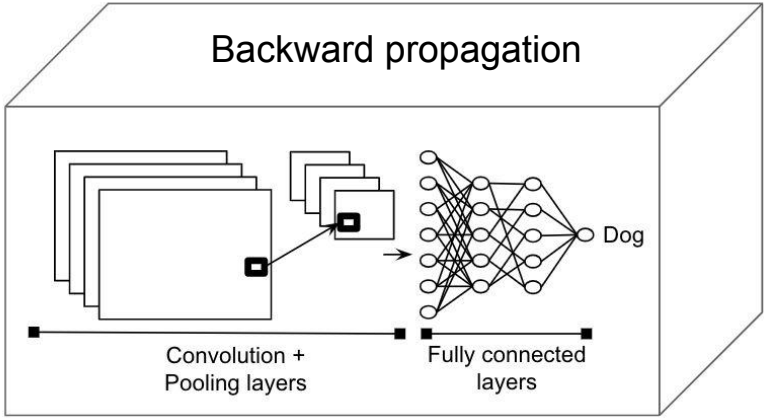
Pooling layer (max pooling)


Dog
Input
Convolution + Pooling layers
Fully connected layers

Input



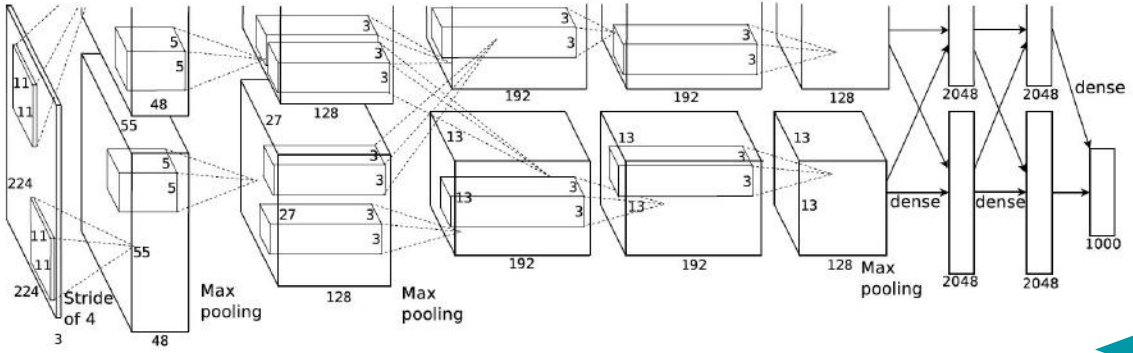
Error
calculation



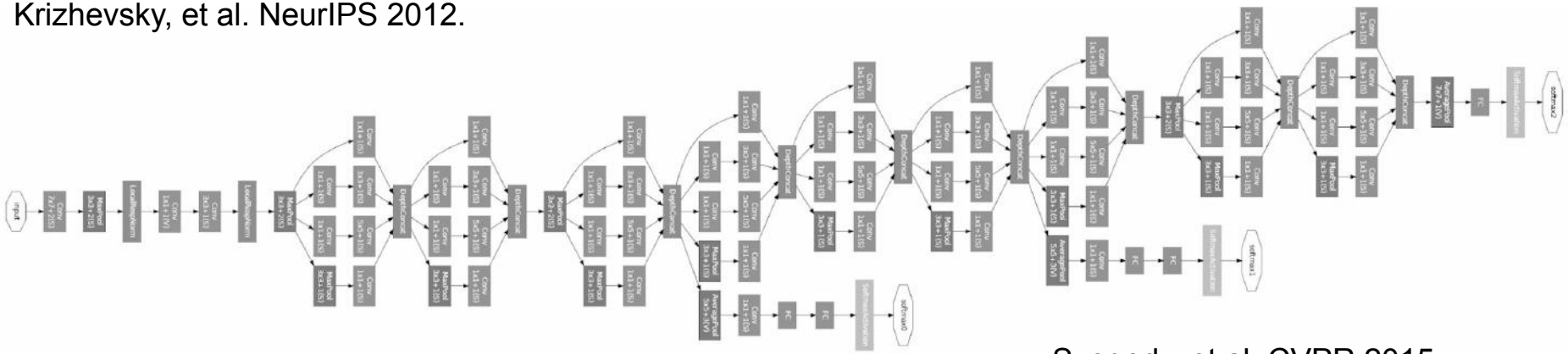
Motivation


traffic light
traffic light
traffic light
traffic light
traffic light
bus
truck
car
car
car
truck
truck
car
person
person
person
person
person
person
handbag

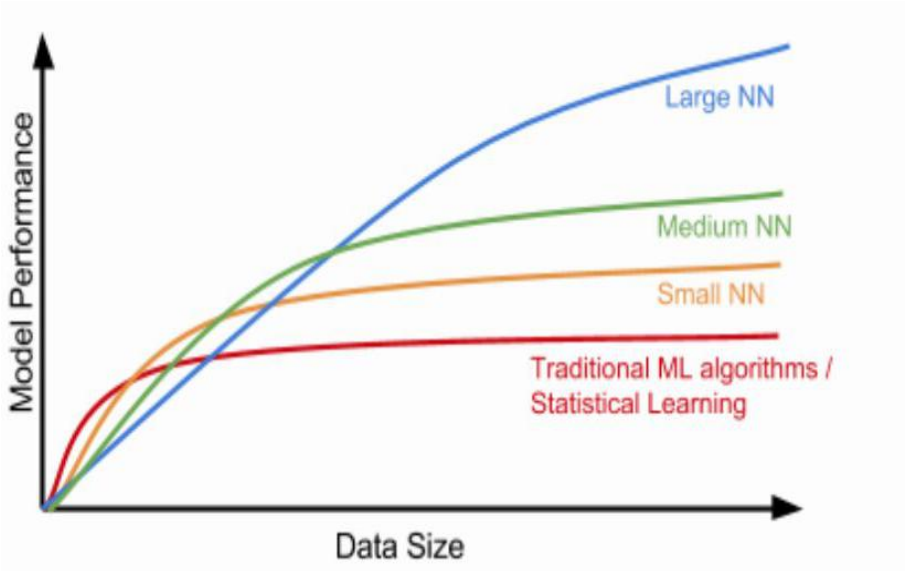

What can I help you with?
Side button

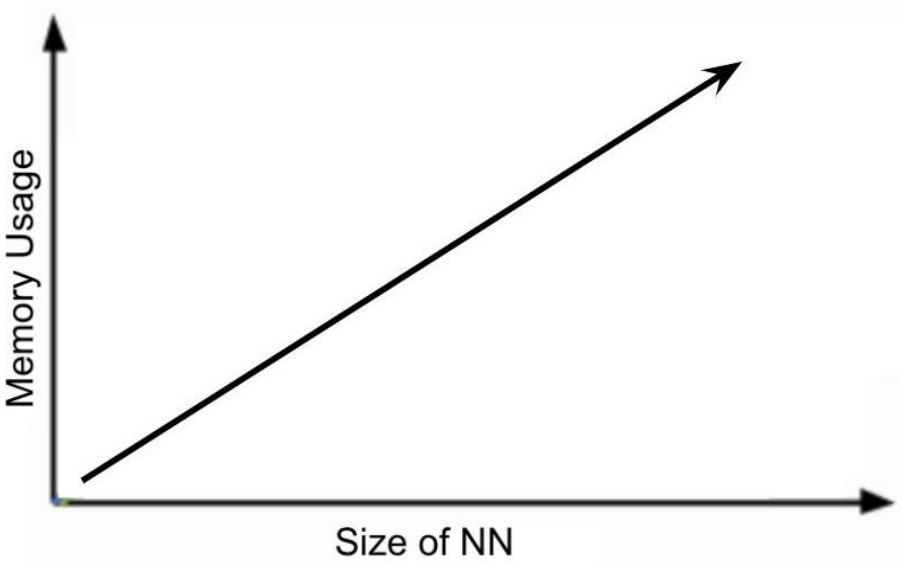


Krizhevsky, et al. NeurIPS 2012.



Szegedy, et al. CVPR 2015.


Model Performance
Large NN
Medium NN
Small NN
Traditional ML algorithms /
Statistical Learning
Data Size


Memory Usage
Size of NN

pytorch / pytorch

Used by 24.6k | Watch 1.4k | Star 37.1k | Fork 9.4k

Code | Issues 3,994 | Pull requests 1,319 | Actions | Projects 5 | Wiki | Security | Insights

RuntimeError: CUDA out of memory. Tried to allocate 12.50 MiB (GPU 0; 10.92 GiB total capacity; 8.57 MiB already allocated; 9.28 GiB free; 4.68 MiB cached) #16417

New issue

Closed [redacted] opened this issue on Jan 27, 2019 · 64 comments

[redacted] commented on Jan 27, 2019 • edited

CUDA Out of Memory error but CUDA memory is almost empty

I am currently training a lightweight model on very large amount of textual data (about 70GiB of text). For that I am using a machine on a cluster ('grele' of the grid5000 cluster network).

I am getting after 3h of training this very strange CUDA Out of Memory error message:
`RuntimeError: CUDA out of memory. Tried to allocate 12.50 MiB (GPU 0; 10.92 GiB total capacity; 8.57 MiB already allocated; 9.28 GiB free; 4.68 MiB cached)` .
According to the message, I have the required space but it does not allocate the memory.

Any idea what might cause this ?

For information, my preprocessing relies on `torch.multiprocessing.Queue` and an iterator over the lines of my source data to preprocess the data on the fly.

Assignees

No one assigned

Labels

needs reproduction

Projects

None yet

Milestone

No milestone

Linked pull requests

Unable to allocate cuda memory, when there is enough of cached memory

1 Dec '18

Can someone please explain this:

```
RuntimeError: CUDA out of memory.
Tried to allocate 350.00 MiB
(GPU 0; 7.93 GiB total capacity; 5.73 GiB already allocated;
324.56 MiB free; 1.34 GiB cached)
```

If there is 1.34 GiB cached, how can it not allocate 350.00 MiB?

There is only one process running. torch-1.0.0/cuda10

And a related question:

Are there any tools to show which python objects consume GPU RAM (besides the pytorch preloaded structures which take some 0.5GB per process) ? i.e. is there some way to query pytorch for a reference to variables that are on CUDA and perhaps from there make some deductions?

Thank you.

4

CUDA out of memory issue when training #549

Closed [redacted] opened this issue on Apr 27, 2019 · 2 comments

[redacted] commented on Apr 27, 2019

The stacktrace is listed below. I've changed the batch_size and sequence_length to a smaller number, and tried both gpu.

```
---------------- training start ----------------------
Epoch 1/5.....
[training] 1/4376 [                                  ] -6.1s/step- loss: 2.6189- acc: 0.0000 - f1: 0.0000Traceback (most recent call last):
  File "train_bert.py", line 142, in <module>
    main()
  File "train_bert.py", line 136, in main
    trainer.train()
  File "/home/XXX/jupyter_projects/BERT-chinese-text-classification-pytorch-master/pybert/train/trainer.py", line 178, in train
    train_log = self._train_epoch()
  File "/home/XXX/jupyter_projects/BERT-chinese-text-classification-pytorch-master/pybert/train/trainer.py", line 141, in _train_epoch
    logits = self.model(input_ids, segment_ids,input_mask)
  File "/home/XXX/.conda/envs/ie/lib/python3.7/site-packages/torch/nn/modules/module.py", line 489, in __call__
    result = self.forward(*input, **kwargs)
  File "/home/XXX/.conda/envs/ie/lib/python3.7/site-packages/torch/nn/parallel/data_parallel.py", line 149, in forward
    outputs = self.parallel_apply(replicas, inputs, kwargs)
  File "/home/XXX/.conda/envs/ie/lib/python3.7/site-packages/torch/nn/parallel/data_parallel.py", line 153, in parallel_apply
    return parallel_apply(replicas, inputs, kwargs, self.device_ids[:len(replicas)])
  File "/home/XXX/.conda/envs/ie/lib/python3.7/site-packages/torch/nn/parallel/parallel_apply.py", line 83, in parallel_apply
    raise output
  File "/home/XXX/.conda/envs/ie/lib/python3.7/site-packages/torch/nn/parallel/parallel_apply.py", line 59, in _worker
    output = module(*input, **kwargs)
  File "/home/XXX/.conda/envs/ie/lib/python3.7/site-packages/torch/nn/modules/module.py", line 489, in __call__
    result = self.forward(*input, **kwargs)
  File "/home/XXX/jupyter_projects/BERT-chinese-text-classification-pytorch-master/pybert/model/nn/bert_fine.py", line 21, in forward
    output_all_encoded_layers=output_all_encoded_layers)
  File "/home/XXX/.conda/envs/ie/lib/python3.7/site-packages/torch/nn/modules/module.py", line 489, in __call__
    result = self.forward(*input, **kwargs)
  File "/home/XXX/.local/lib/python3.7/site-packages/pytorch_pretrained_bert/modeling.py", line 714, in forward
    output_all_encoded_layers=output_all_encoded_layers)
  File "/home/XXX/.conda/envs/ie/lib/python3.7/site-packages/torch/nn/modules/module.py", line 489, in __call__
    result = self.forward(*input, **kwargs)
  File "/home/XXX/.local/lib/python3.7/site-packages/pytorch_pretrained_bert/modeling.py", line 396, in forward
    hidden_states = layer_module(hidden_states, attention_mask)
  File "/home/XXX/.conda/envs/ie/lib/python3.7/site-packages/torch/nn/modules/module.py", line 489, in __call__
    result = self.forward(*input, **kwargs)
  File "/home/XXX/.local/lib/python3.7/site-packages/pytorch_pretrained_bert/modeling.py", line 382, in forward
    intermediate_output = self.intermediate(attention_output)
  File "/home/XXX/.conda/envs/ie/lib/python3.7/site-packages/torch/nn/modules/module.py", line 489, in __call__
    result = self.forward(*input, **kwargs)
  File "/home/XXX/.local/lib/python3.7/site-packages/pytorch_pretrained_bert/modeling.py", line 366, in forward
    hidden_states = self.intermediate_act_fn(hidden_states)
  File "/home/XXX/.local/lib/python3.7/site-packages/pytorch_pretrained_bert/modeling.py", line 119, in gelu
    return x * 0.5 * (1.0 + torch.erf(x / math.sqrt(2.0)))
RuntimeError: CUDA out of memory. Tried to allocate 3.00 MiB (GPU 0; 10.92 GiB total capacity; 1.67 GiB already allocated; 5.50 MiB free; 26.40 MiB cached)
```

[redacted] commented on Apr 28, 2019 Member

Try reducing the batch size?

Any other personal experience?
Solutions?

```
~/anaconda2/envs/torchreid/lib/python3.7/site-packages/torch/nn/modules/module.py in __call__(self, *input, **kwargs)
    530             result = self._slow_forward(*input, **kwargs)
    531         else:
--> 532             result = self.forward(*input, **kwargs)
    533         for hook in self._forward_hooks.values():
    534             hook_result = hook(self, input, result)

~/anaconda2/envs/torchreid/lib/python3.7/site-packages/torch/nn/modules/conv.py in forward(self, input)
    343
    344     def forward(self, input):
--> 345         return self.conv2d_forward(input, self.weight)
    346
    347 class Conv3d(_ConvNd):

~/anaconda2/envs/torchreid/lib/python3.7/site-packages/torch/nn/modules/conv.py in conv2d_forward(self, input, weight)
    340                             _pair(0), self.dilation, self.groups)
    341         return F.conv2d(input, weight, self.bias, self.stride,
--> 342                         self.padding, self.dilation, self.groups)
    343
    344     def forward(self, input):

RuntimeError: CUDA out of memory. Tried to allocate 256.00 MiB (GPU 0; 10.73 GiB total capacity; 7.79 GiB already allocated; 85.62 MiB free; 8.00 GiB reserved in total b
y PyTorch)
```

Any other personal experience? Solutions?

Open **RuntimeError: CUDA out of memory. Tried to allocate 20.00 MiB (GPU 0; 15.90 GiB total capacity; 14.66 GiB already allocated; ....** #685

alipetiwala opened this issue on Apr 25, 2019 · 14 comments

any other pre-trained word embeddings for this. Would like to understand if you guys have a work around for this.

[redacted] commented on May 8, 2019 Collaborator

Hello **@tsu3010** this likely happens because a mini-batch is pushed through the BERT model that requires too much GPU memory, i.e. there are too long texts in the dataset and the mini-batch size too large (see issue 549)

You could try reducing the mini-batch size from 32 to 8. You could filter or truncate long texts from the dataset to make it so a mini-batch fits into memory.

5

GIST

Open RuntimeError: CUDA out of me ied to allocate 20.00 MiB (GPU 0; 15.90 l capacity; 14.66 GiB already allocated; .... #685

alipetiwala opened this issue on 2019 · 14 comments

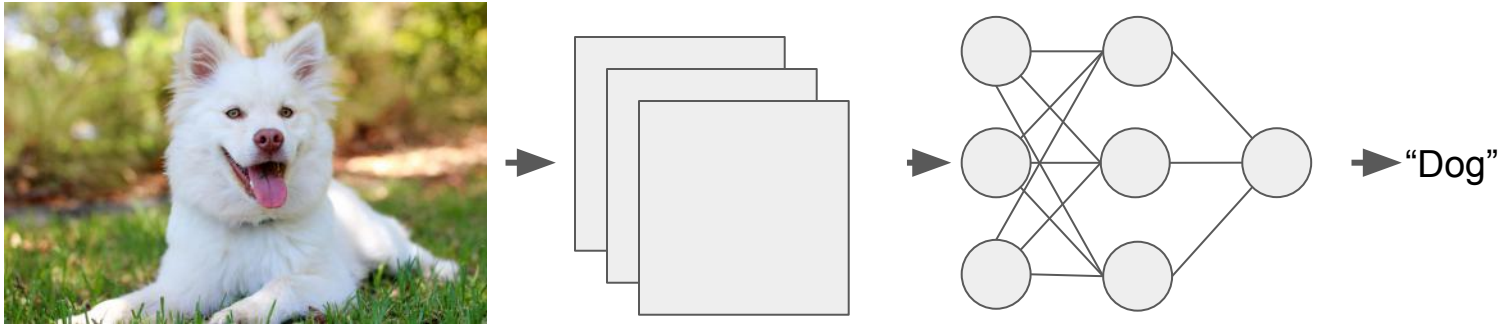
any other pre-trained w mbe s for this. Would like to understand if you guys have k around for this.

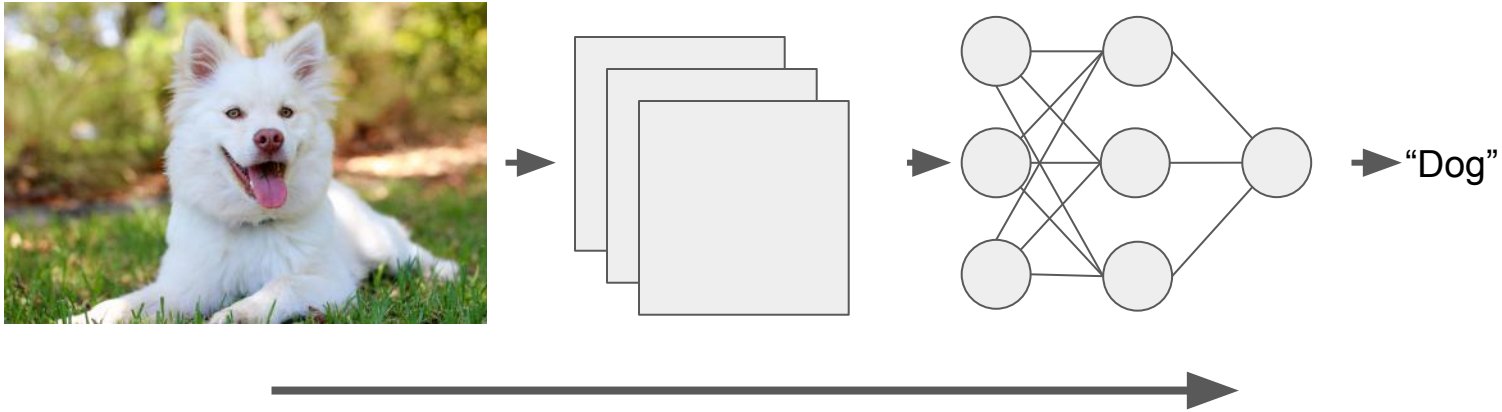
comm d on May 8, 2019 Collaborator

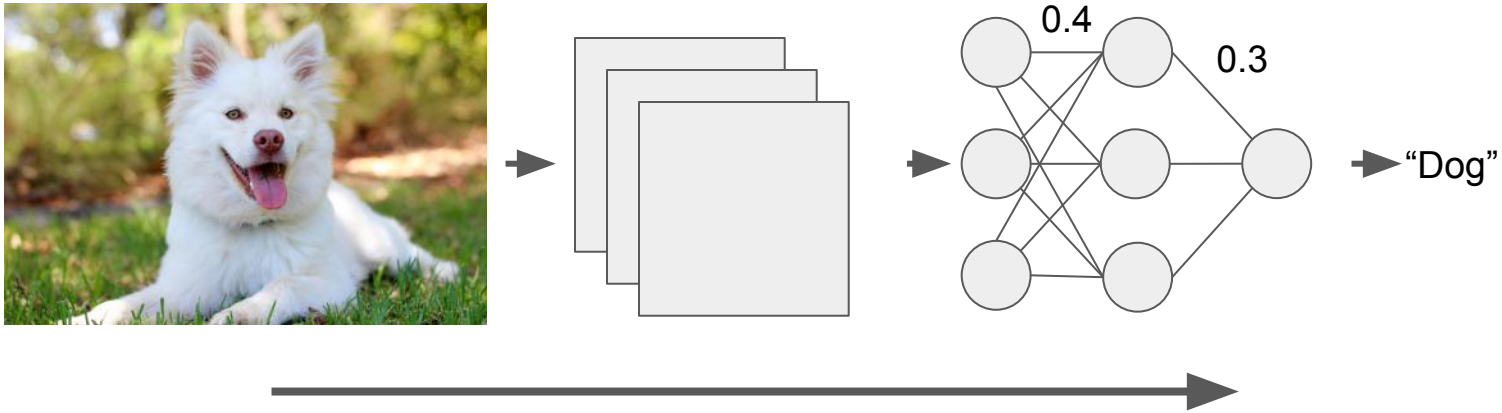
Hello @tsu3010 likely happens because a mini-batch is p l through the BERT model that requires too muc PU memory, i.e. there are too long texts in t aset and the mini-batch size t large (see issue 54

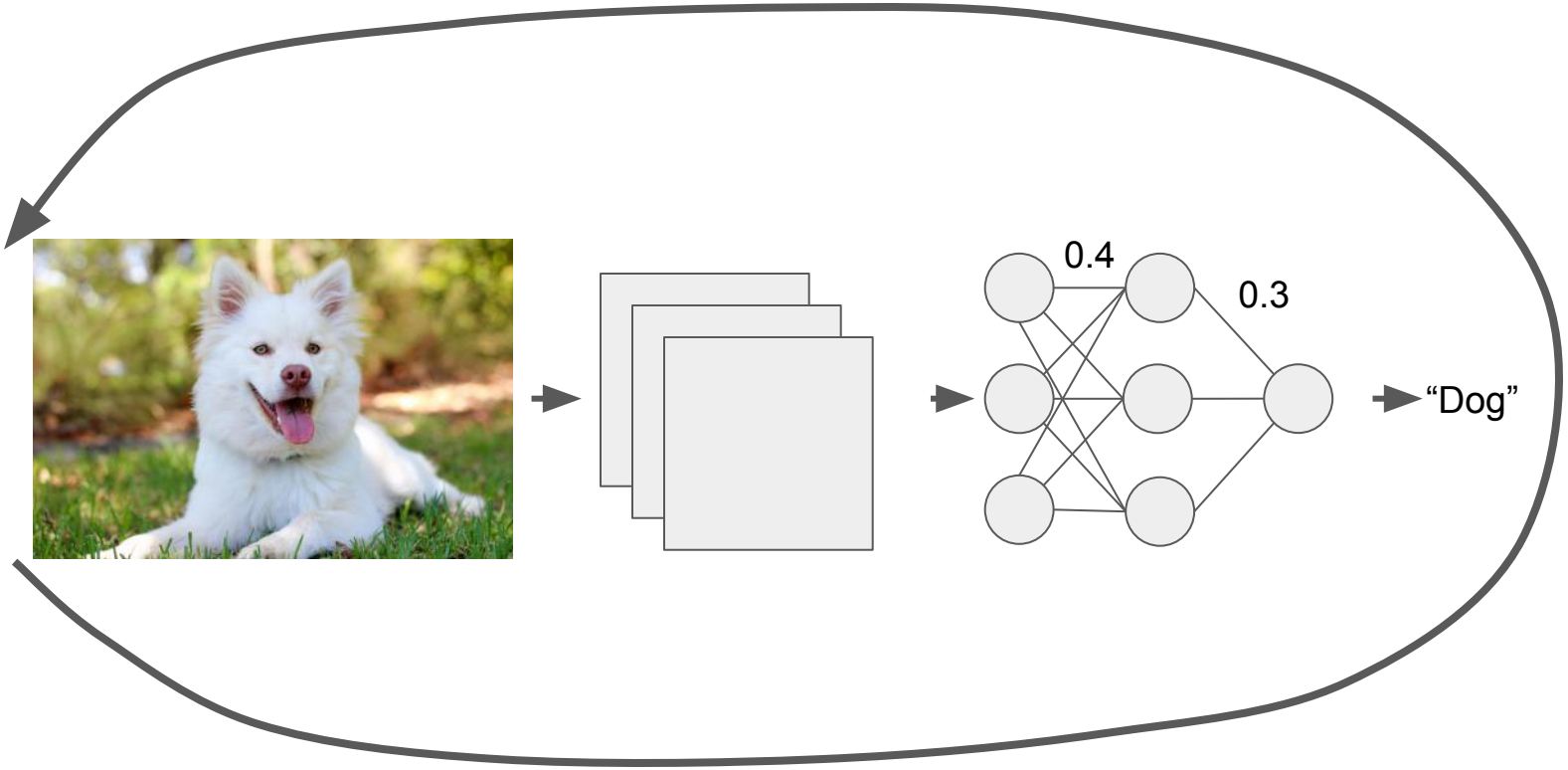
You could try reduci e mini-batch size from 32 to 8. You could filter or t e long texts fr e dataset to make it so a i-batch fits into memory.

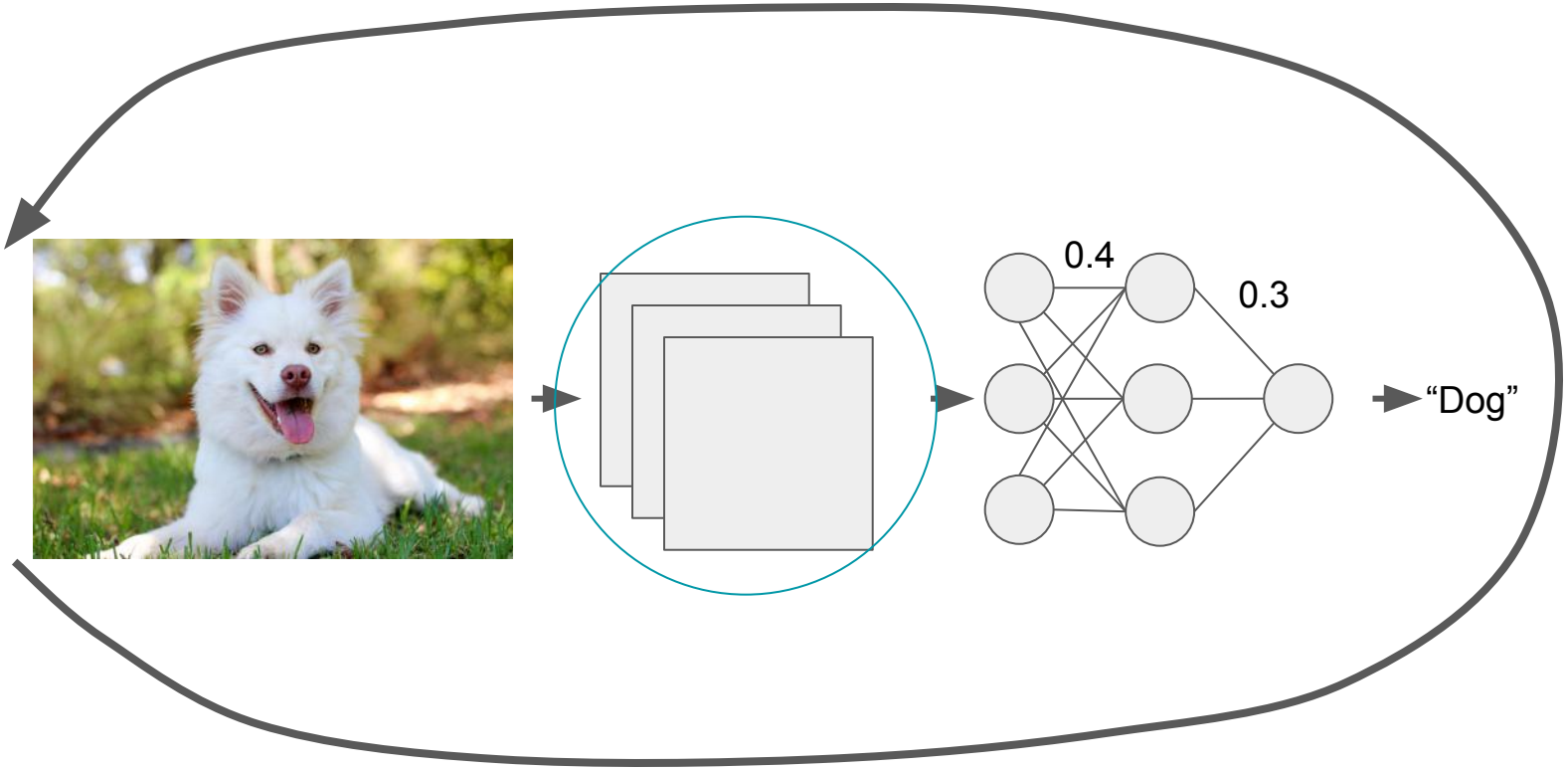
5


"Dog"


"Dog"


0.4
0.3
“Dog”


0.4
0.3
“Dog”


0.4
0.3
“Dog”

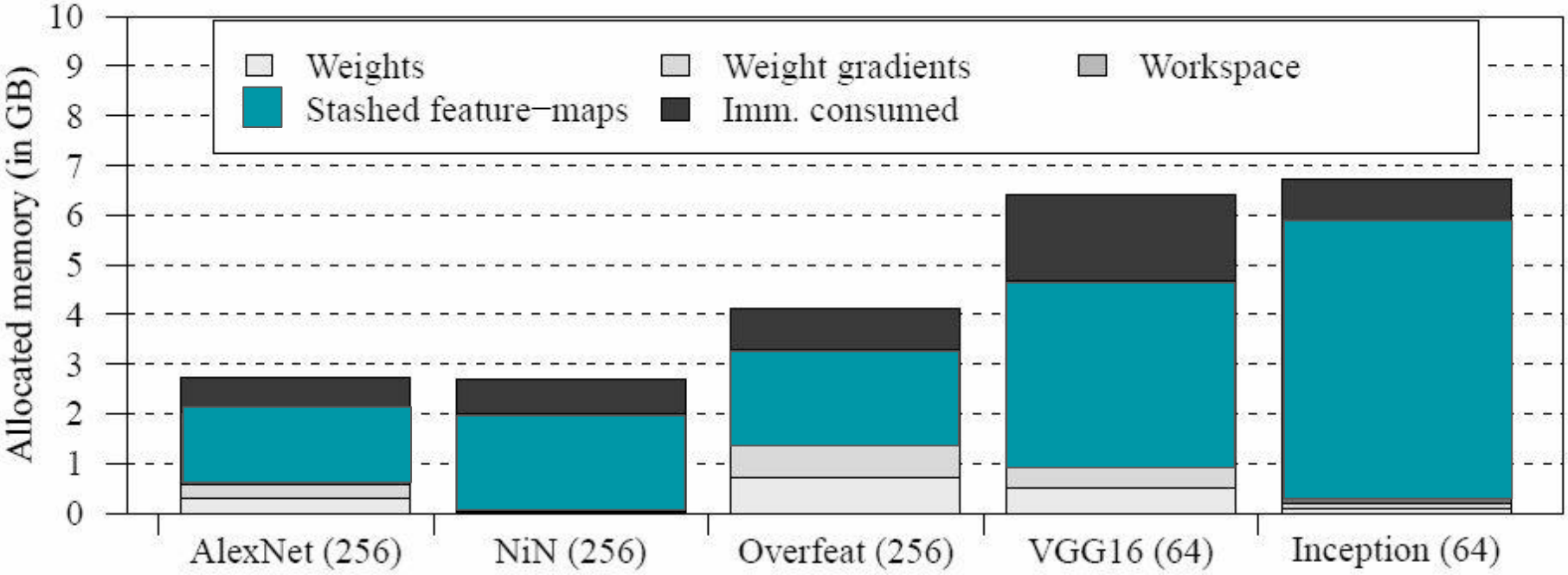
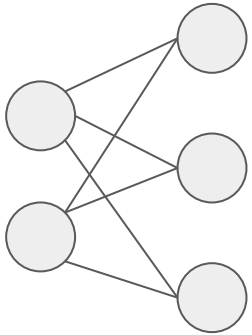

Allocated memory (in GB)
10
9
8
7
6
5
4
3
2
1
0
Weights
Weight gradients
Workspace
Stashed feature−maps
Imm. consumed
AlexNet (256)
NiN (256)
Overfeat (256)
VGG16 (64)
Inception (64)

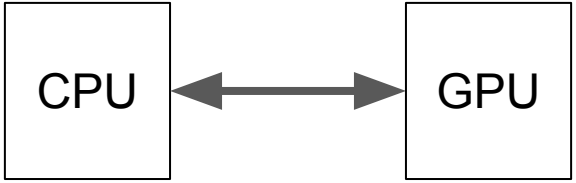
Figure 1



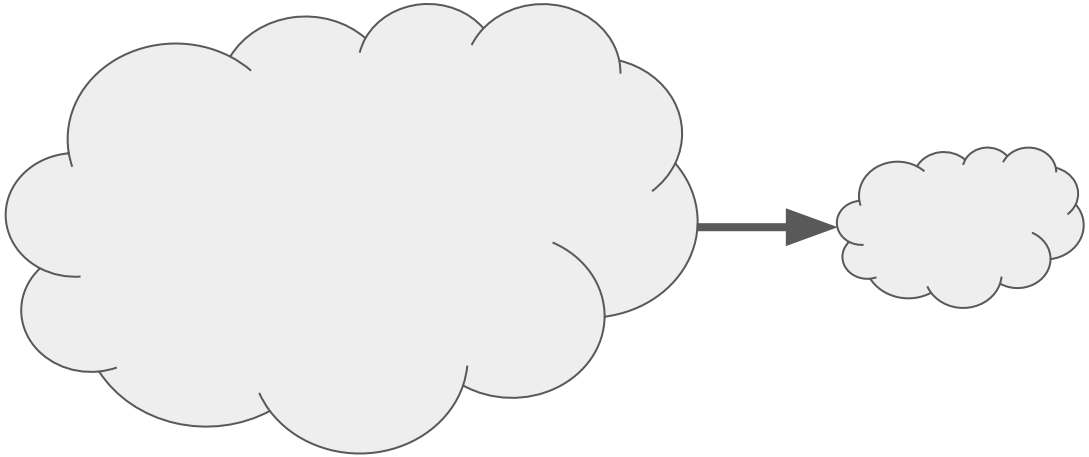
Single → Half Precision

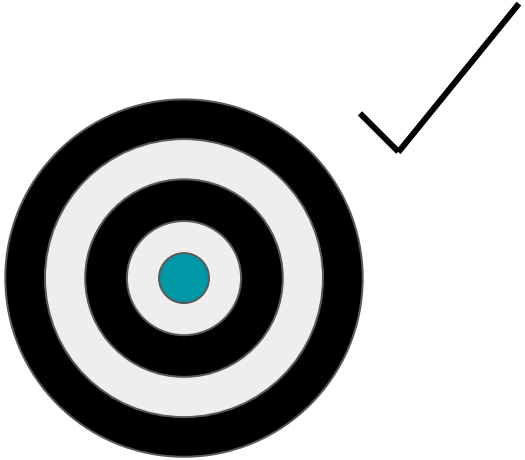
```
>>> x = numpy.float32(68.123)
>>> x
68.123
>>> y = numpy.float16(68.123)
>>> y
68.1
```

vDNN

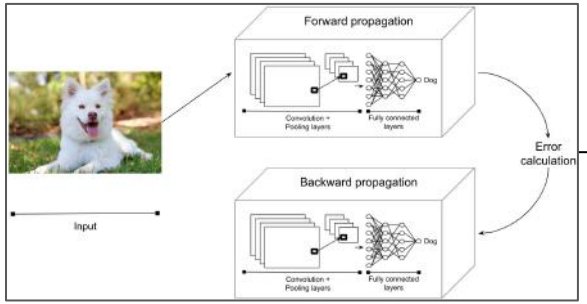


How does this compare to MISTIQUE, KeystoneML?



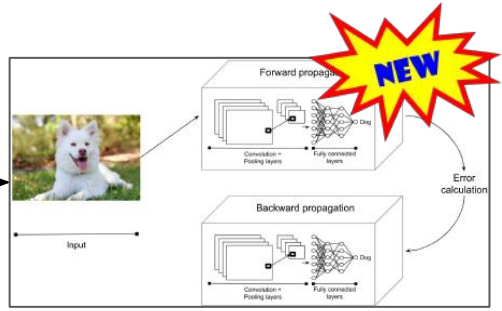


GIST Overview



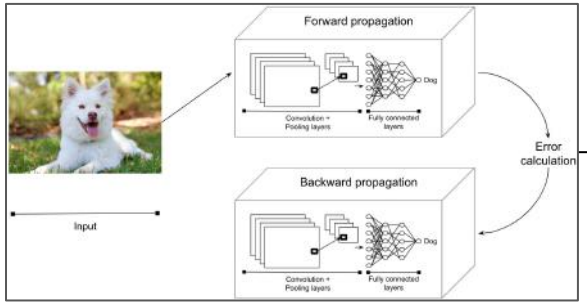
GIST Schedule Builder

Apply encodings



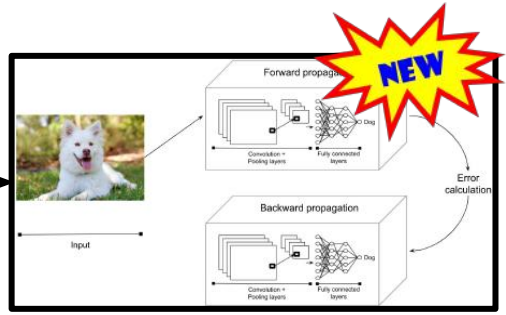
Static liveness analysis

CNTK static memory allocator



GIST Schedule Builder

Apply encodings

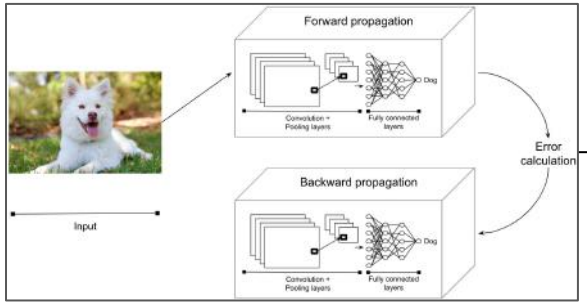


Static liveness analysis

CNTK static memory allocator

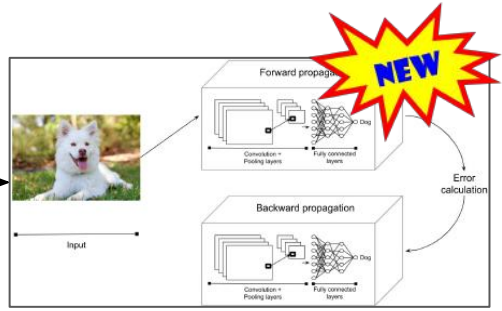
Apply encodings

- Lossless Encoding
- Lossy Encoding



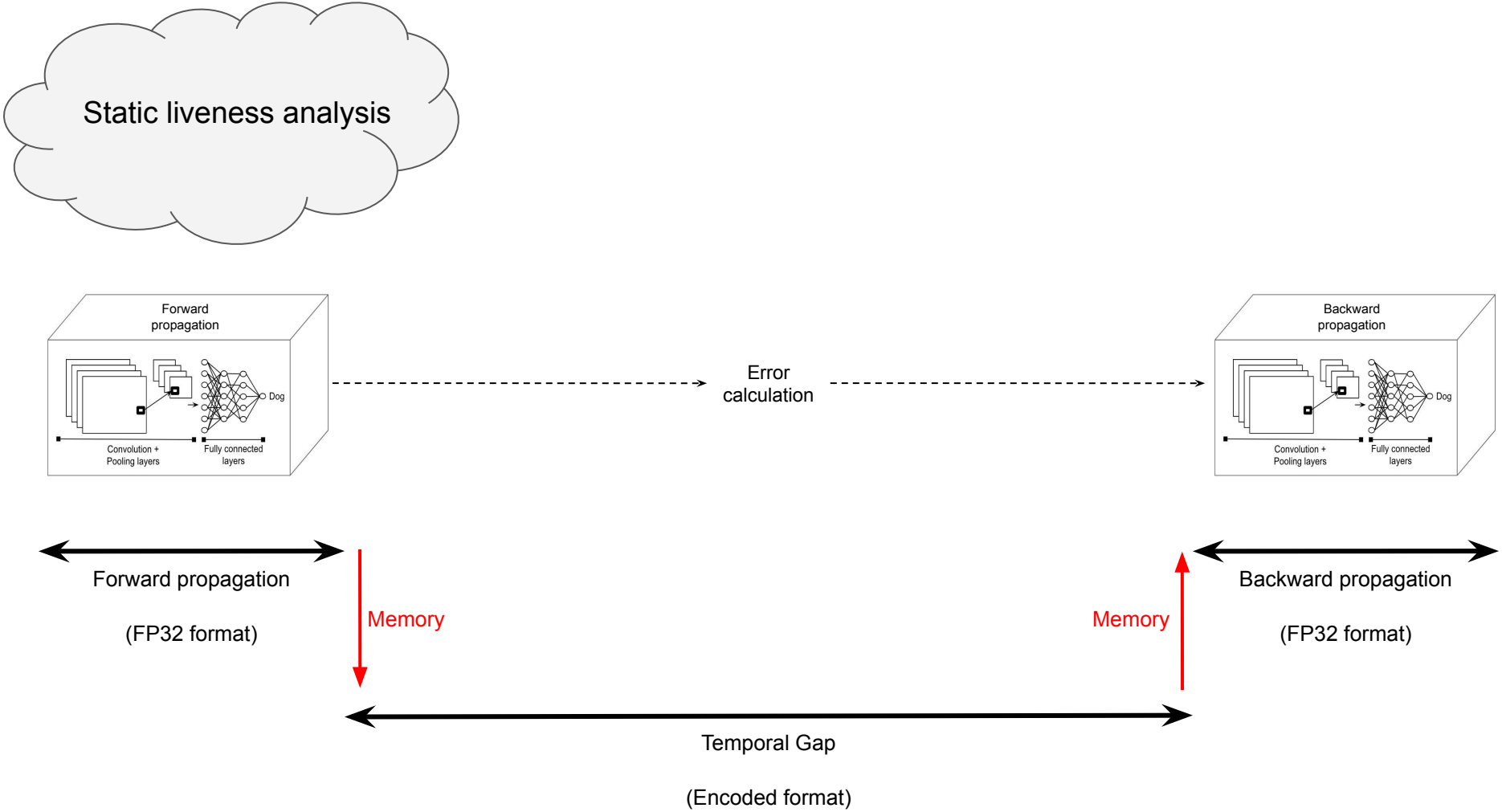
GIST Schedule Builder

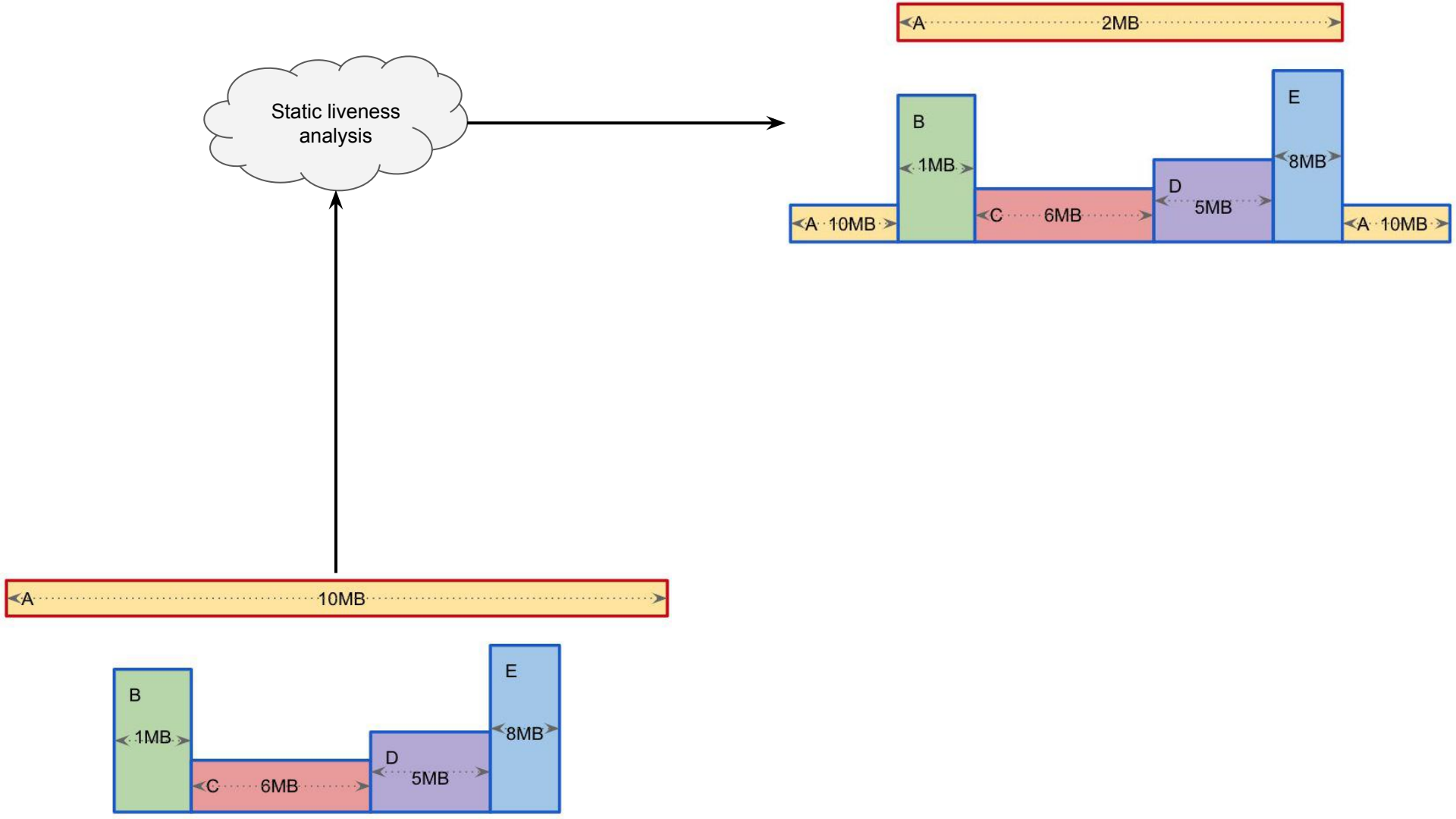
Apply encodings

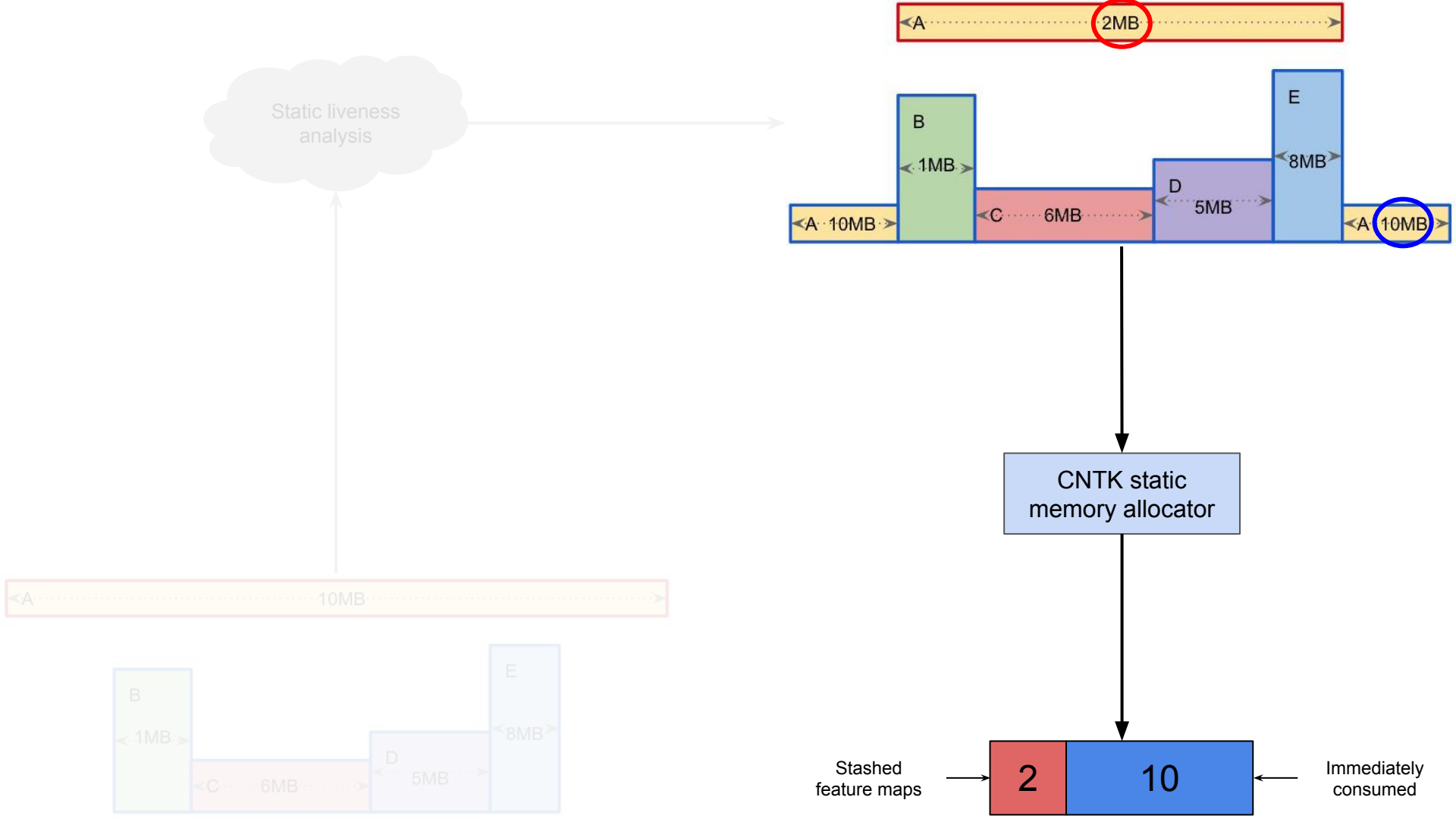


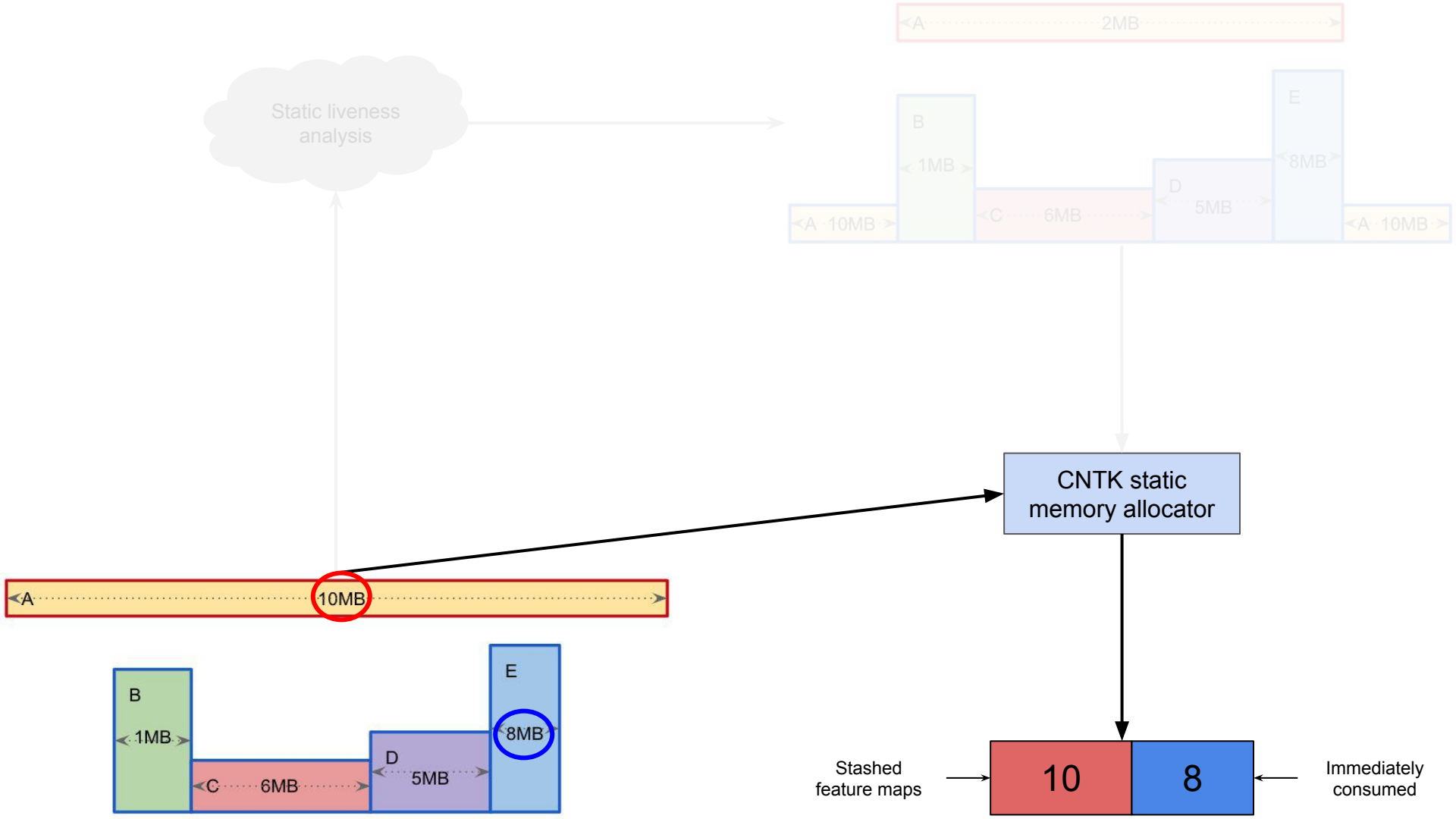
Static liveness analysis

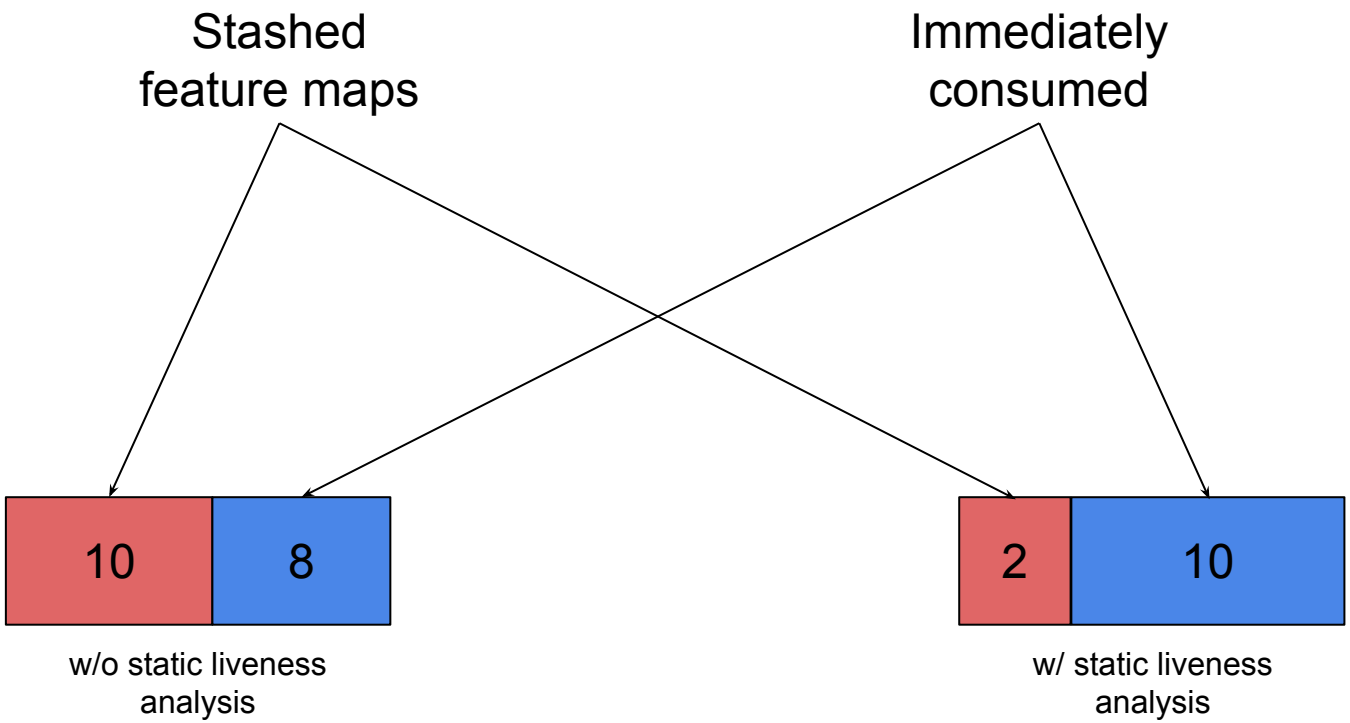
CNTK static memory allocator

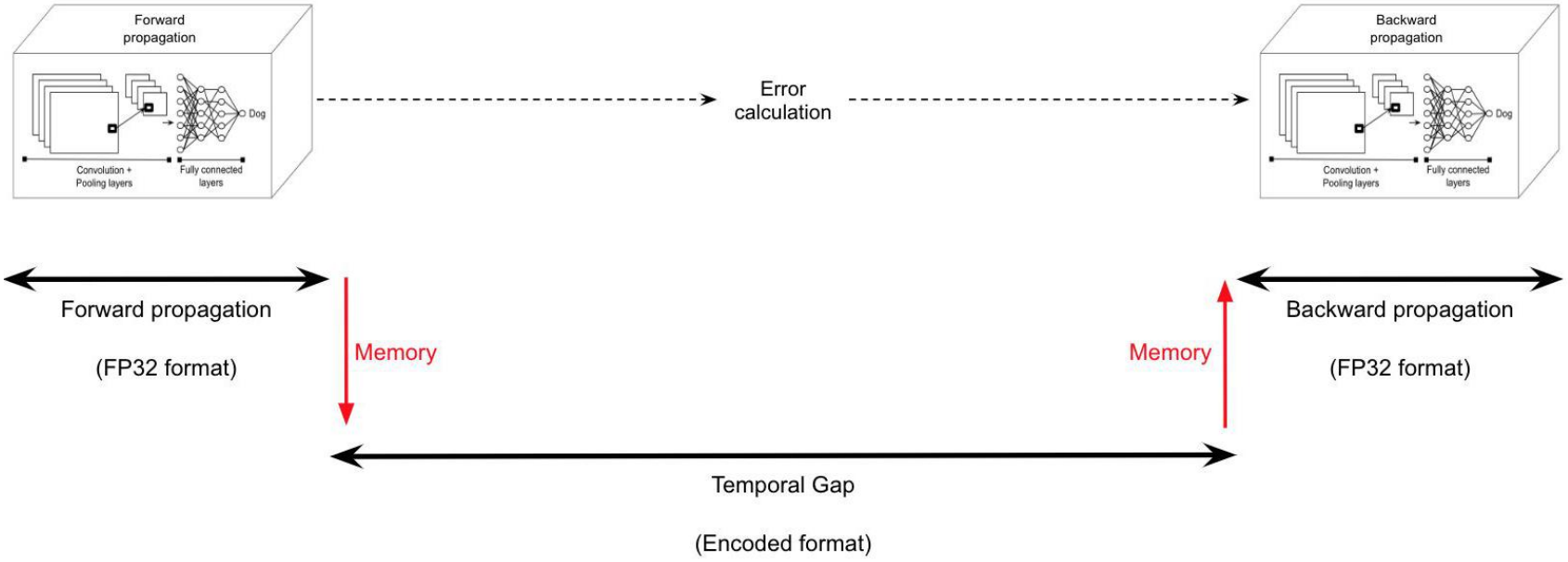

Static liveness analysis
Forward propagation
Dog
Convolution + Pooling layers
Fully connected layers
Error calculation
Backward propagation
Dog
Convolution + Pooling layers
Fully connected layers
Forward propagation
(FP32 format)
Memory
Memory
Backward propagation
(FP32 format)
Temporal Gap
(Encoded format)


Static liveness analysis
A
10MB
B
1MB
C
6MB
D
5MB
E
8MB
A
2MB
B
1MB
C
6MB
D
5MB
E
8MB
A
10MB
A
10MB


Static liveness analysis
A
10MB
B
1MB
C
6MB
D
5MB
E
8MB
A
2MB
B
1MB
C
6MB
D
5MB
E
8MB
A
10MB
CNTK static memory allocator
Stashed feature maps
2
10
Immediately consumed

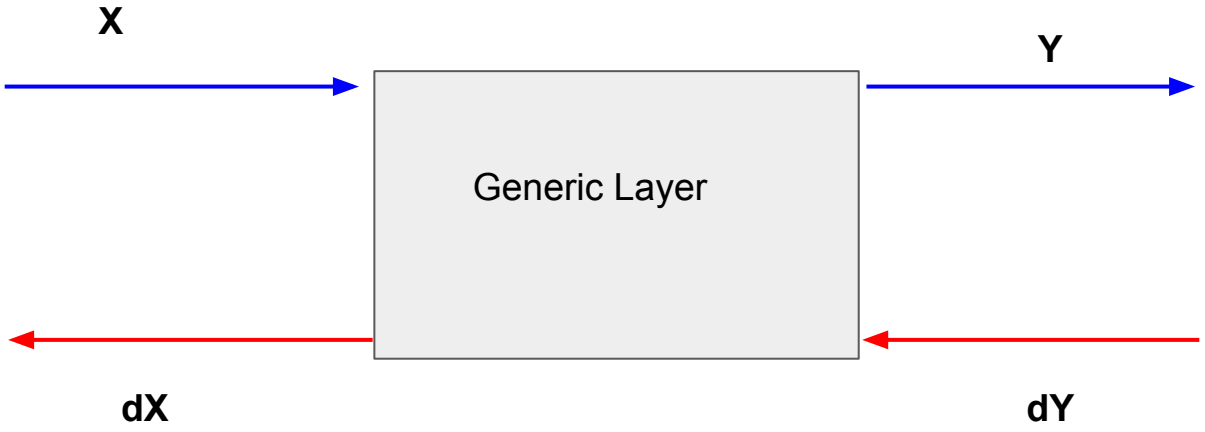

Static liveness analysis
A
2MB
B
1MB
C
6MB
D
5MB
E
8MB
A
10MB
CNTK static memory allocator
A
10MB
B
1MB
C
6MB
D
5MB
E
8MB
Stashed feature maps
10
8
Immediately consumed


Stashed
feature maps
Immediately
consumed
10
8
2
10
w/o static liveness
analysis
w/ static liveness
analysis



Can this be implemented to avoid the extra memory usage?

Details


X
Y
Generic Layer
dX
dY

Max-Pooling Forward Pass

X

5	23	14	17
8	11	14	2
0	0	4	1
3	9	9	5

2x2 Max-Pool →

Y

23	17
9	9

Naive Backpropagation

dX

0	0.2	0	0.8
0	0	0	0
0	0	0	0
0	0.4	0.5	0

2x2 Max-Pool backprop

Y

23	17
9	9

X

5	23	14	17
8	11	14	2
0	0	4	1
3	9	9	5

dY

0.2	0.8
0.4	0.5

Gist Max-Pool Backpropagation

dX

0	0.2	0	0.8
0	0	0	0
0	0	0	0
0	0.4	0.5	0

2x2 Max-Pool backprop

dY

0.2	0.8
0.4	0.5

$YToX_{map}$

1	1
3	2

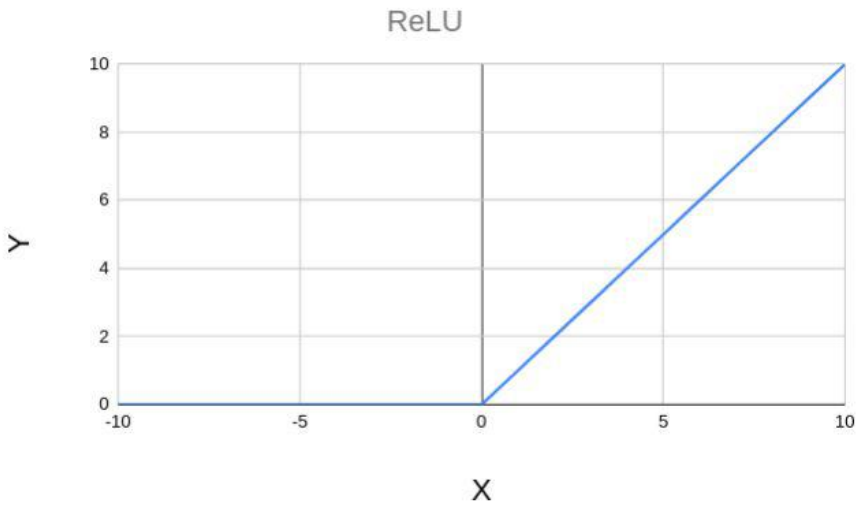
ReLU Standard Implementation

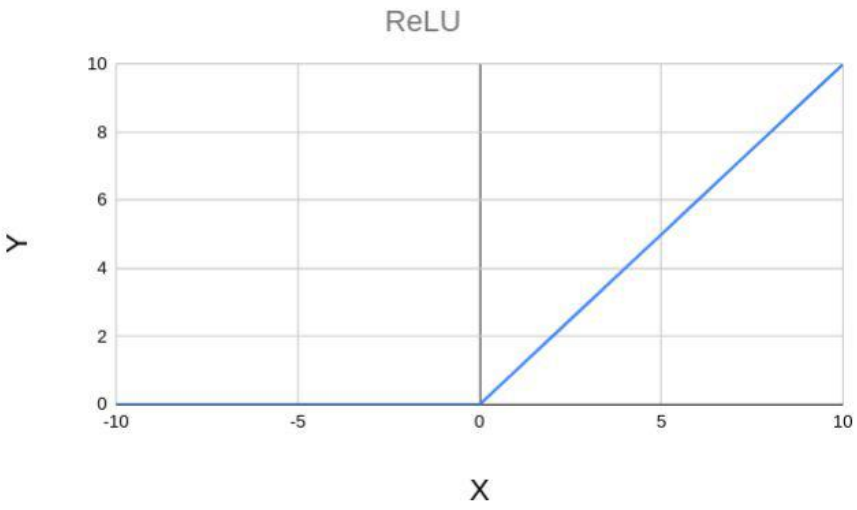
-4	3	2	-3
7	-1	4	8
5	-4	6	-8
7	-1	-2	4

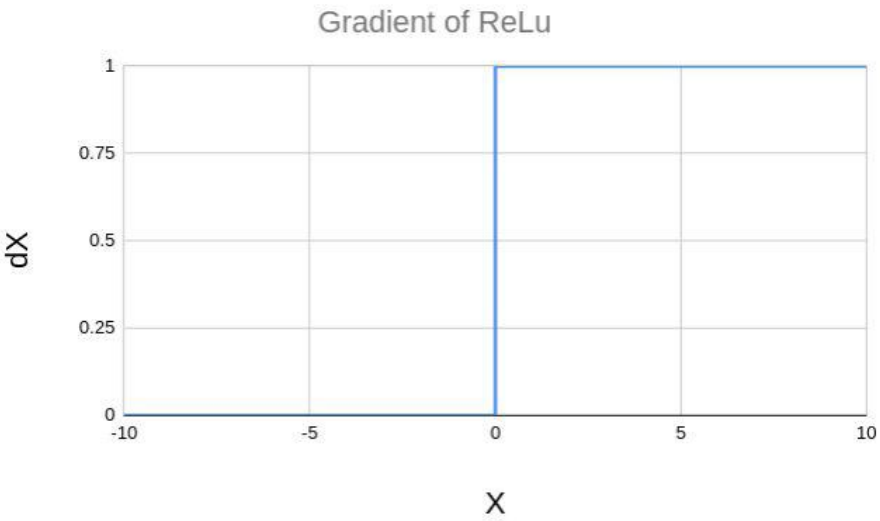
ReLU →

FP32

0	3	2	0
7	0	4	8
5	0	6	0
7	0	0	4


ReLU
10
8
6
4
2
0
Y
-10
-5
0
5
10
X


ReLU
Y
X
10
8
6
4
2
0
-10
-5
0
5
10


Gradient of ReLu
dX
X
1
0.75
0.5
0.25
0
-10
-5
0
5
10

Binarize

-4	3	2	-3
7	-1	4	8
5	-4	6	-8
7	-1	-2	4

ReLU

FP32

0	3	2	0
7	0	4	8
5	0	6	0
7	0	0	4

Binarize

-4	3	2	-3
7	-1	4	8
5	-4	6	-8
7	-1	-2	4

ReLU

FP32

0	3	2	0
7	0	4	8
5	0	6	0
7	0	0	4

1 bit

0	1	1	0
1	0	1	1
1	0	1	0
1	0	0	1

Sparse Storage Dense Compute

Sparse conv-ReLU output

0	0	3	0
3	0	0	1
0	2	4	0
7	0	0	0

FP32

FP32

0	0	3	0
3	0	0	1
0	2	4	0
7	0	0	0

Sparse Storage Dense Compute

Sparse conv-ReLU output

0	0	3	0
3	0	0	1
0	2	4	0
7	0	0	0

FP32

FP32

0	0	3	0
3	0	0	1
0	2	4	0
7	0	0	0

Non-zero values

3
3
1
2
4
7

FP32

Column indices

2
0
3
1
2
0

1 byte

Non-zero values preceding the ith row

0
1
3
5

Delayed Precision Reduction

-4	3	2	-3
7	-1	4	8
5	-4	6	-8
7	-1	-2	4

ReLU

FP32

0	3	2	0
7	0	4	8
5	0	6	0
7	0	0	4

Delayed Precision Reduction

-4	3	2	-3
7	-1	4	8
5	-4	6	-8
7	-1	-2	4

ReLU

FP32

0	3	2	0
7	0	4	8
5	0	6	0
7	0	0	4

FP16/FP10/FP8

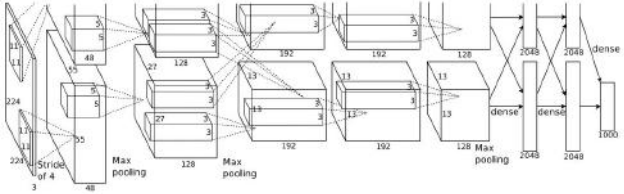
0	3	2	0
7	0	4	8
5	0	6	0
7	0	0	4

What other layers could we encode more efficiently?

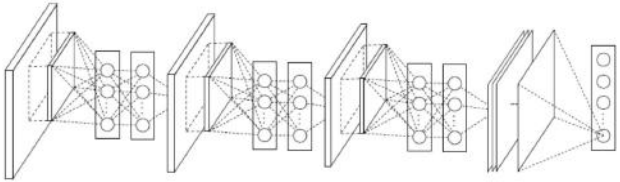
Results

Experimental Setup

AlexNet

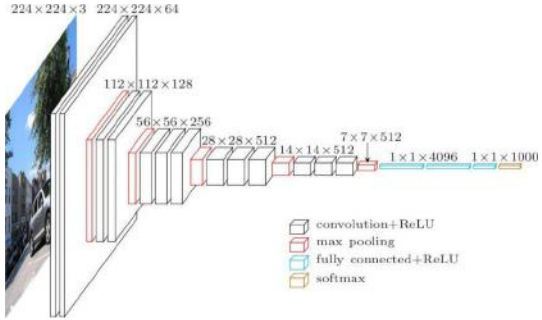


Network-in-Network



Overall Structure of Network In Network (NIN)

VGG16



Memory Footprint Ratio (MFR):

(memory used by baseline) / (memory used with GIST)

CNTK Baseline: does GIST use less memory overall?

Yes!

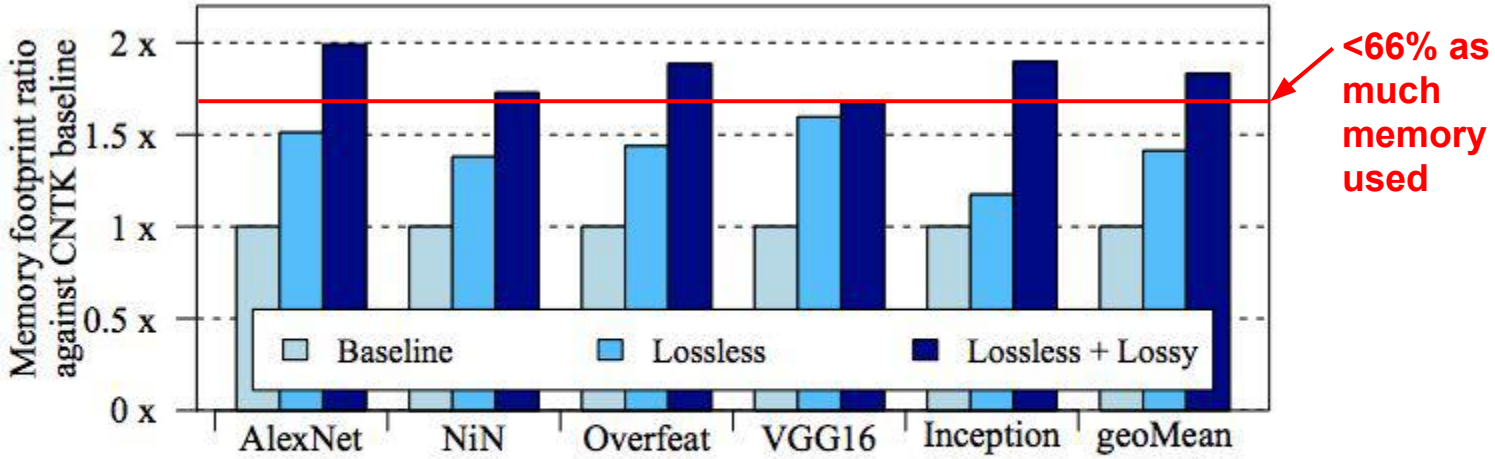


Figure 8: Evaluation of memory footprint reduction - *Gist* cuts down total memory footprint significantly

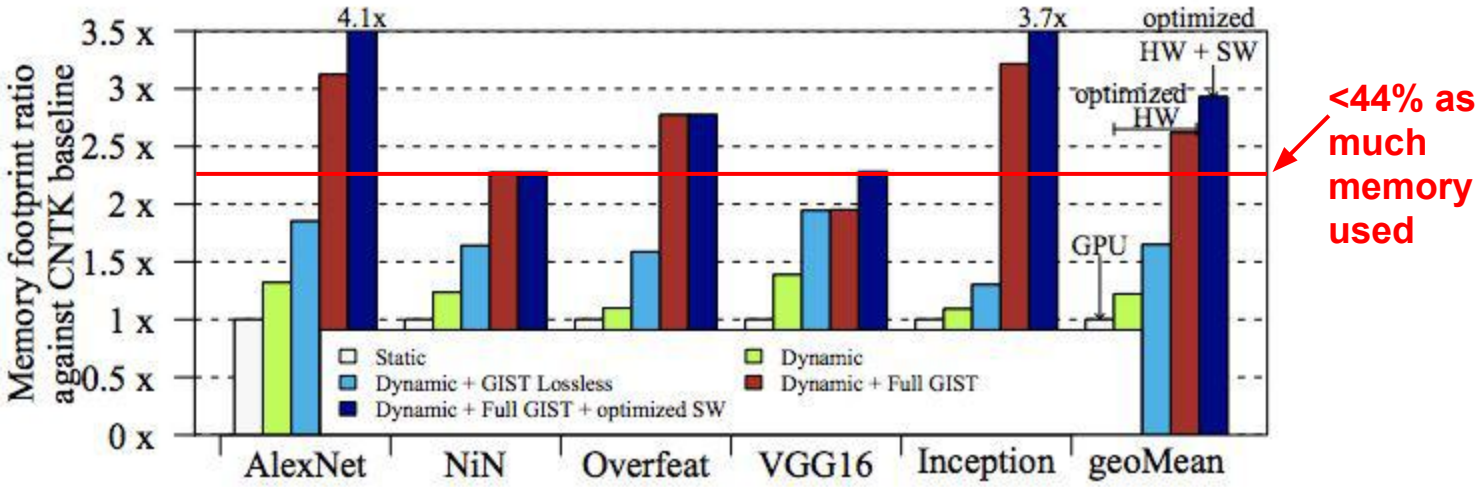


Figure 17: Impact of optimized hardware (dynamic allocation), *Gist* and optimized software (cuDNN)

Investigation Baseline: how much impact does each GIST encoding have on its target layer type?

6% as much memory used in ReLU → Pool

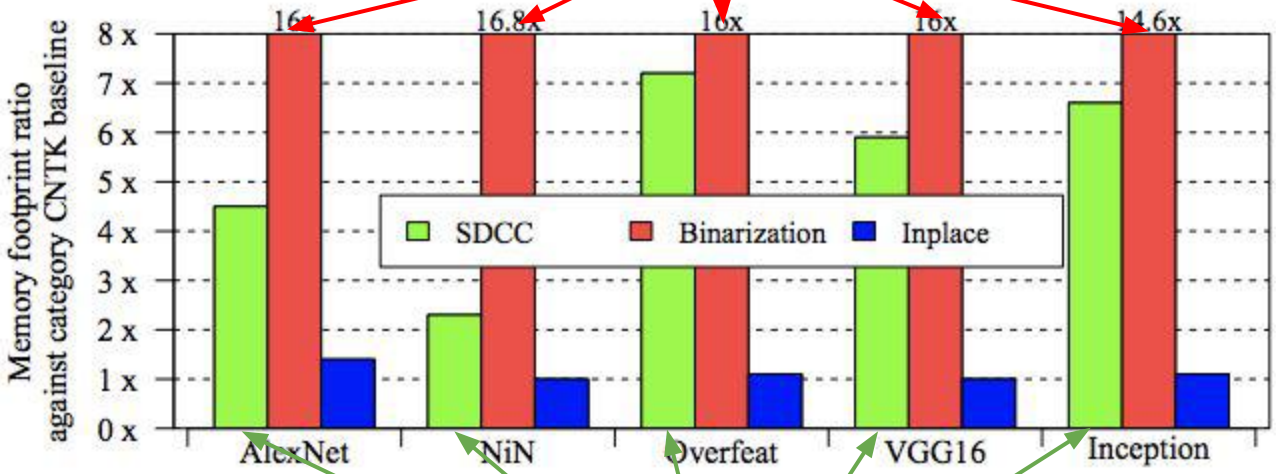
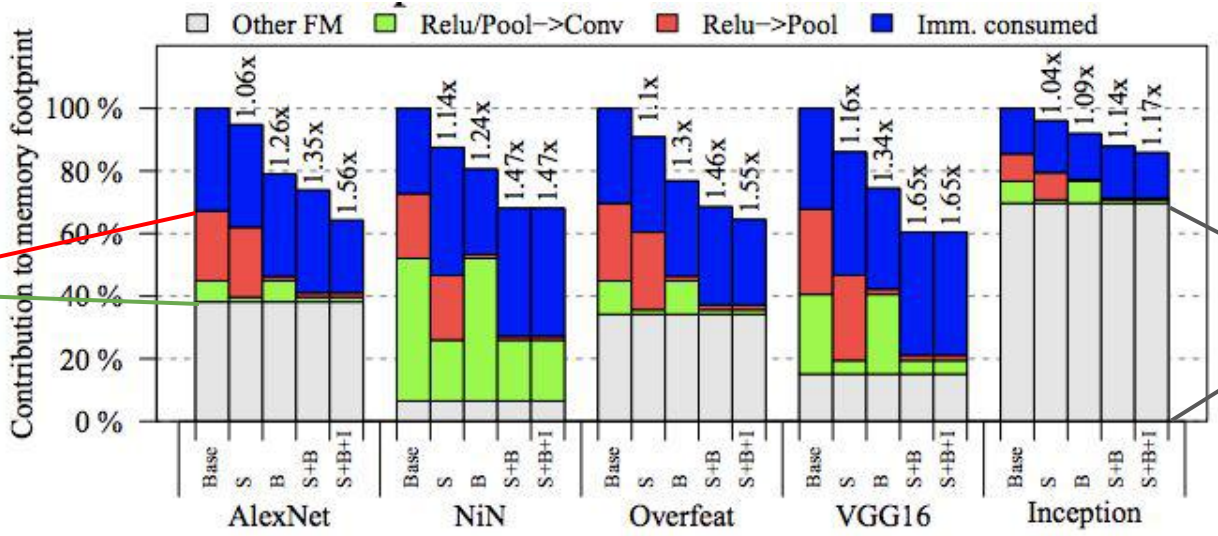


Figure 11: *Gist* lossless encodings MFR on target categories

14-50% as much memory used in ReLU/Pool → Conv

Gist lossless can only optimize the **red** and **green** sections

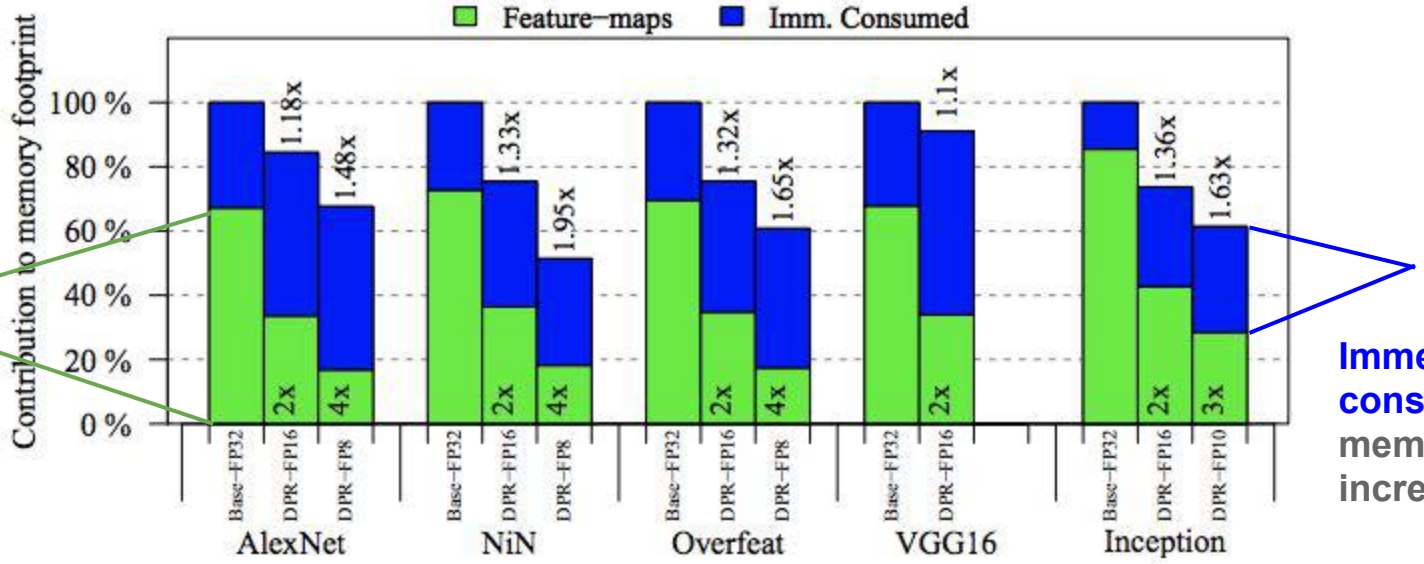


Gist lossless can't optimize any of this

Figure 10: Impact of *Gist* lossless techniques (S - SSDC, B - Binarize, I - Inplace) on memory footprint of different data structures. Total MFR for each configuration is present at the top of each bar

Lossy Encoding: Delayed Precision Reduction (DPR)

This includes the gray and green sections from the previous figure



Immediately consumed memory increases

Figure 13: Impact of DPR encodings. Total MFR is present at the top and MFR achieved on the stashed feature maps is present at the bottom of each bar. Lowest precision for VGG16 with no loss in accuracy is FP16.

Half-precision in the forward pass does not train at all

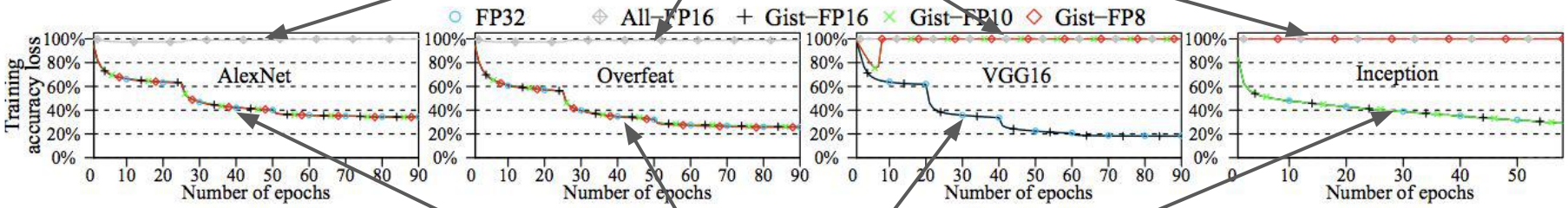


Figure 12: Impact of DPR Encoding on accuracy. The smallest format, having same accuracy compared to FP32 format, for AlexNet and Overfeat is FP8, for Inception is FP10 and for VGG16 is FP16; DPR achieves aggressive bit savings.

Half-precision in the backward pass trains perfectly

Does any of this matter?

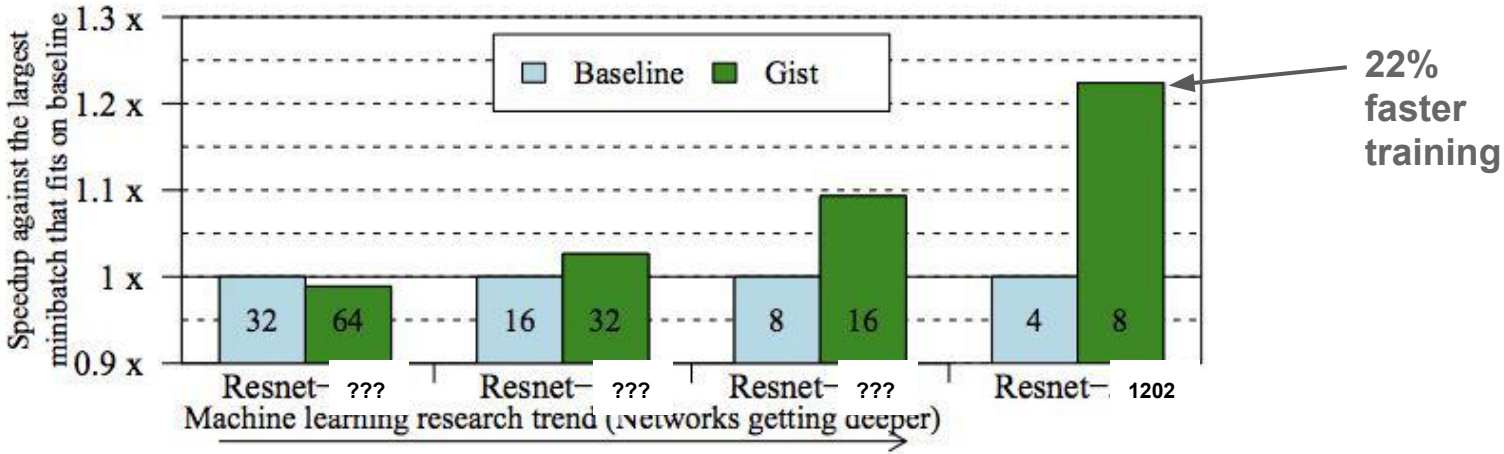


Figure 16: *Gist* enables training deeper networks with larger minibatch, achieving better performance. The largest minibatch that fits in the GPU memory is present at the bottom of the bar

Key Takeaways

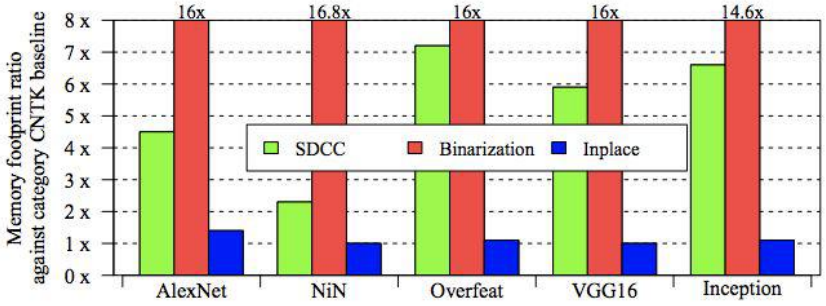


Figure 11: *Gist* lossless encodings MFR on target categories

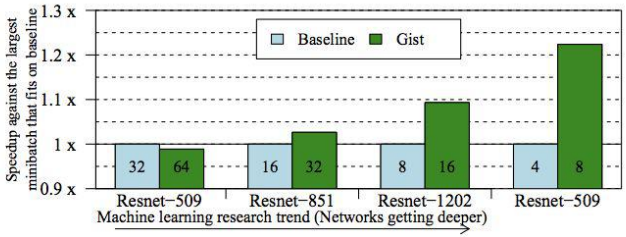


Figure 16: *Gist* enables training deeper networks with larger minibatch, achieving better performance. The largest minibatch that fits in the GPU memory is present at the bottom of the bar

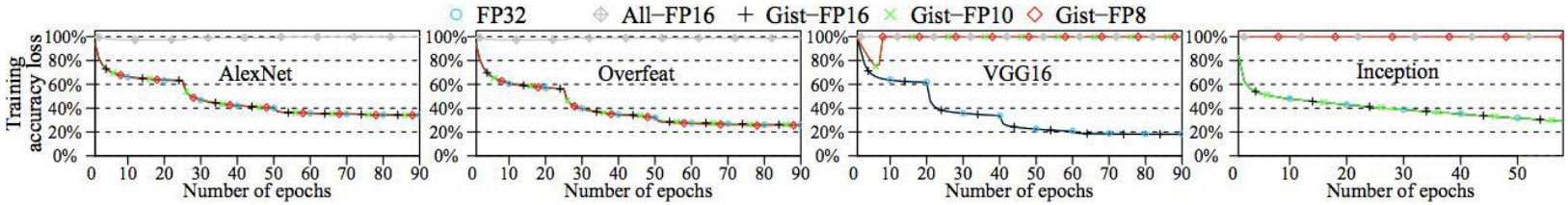
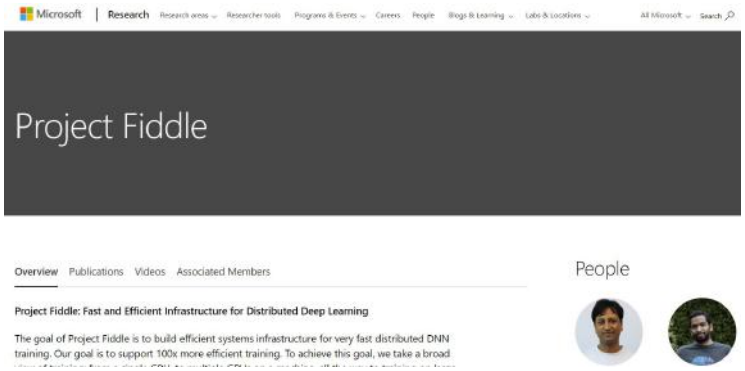


Figure 12: Impact of DPR Encoding on accuracy. The smallest format, having same accuracy compared to FP32 format, for AlexNet and Overfeat is FP8, for Inception is FP10 and for VGG16 is FP16; DPR achieves aggressive bit savings.

Extra Discussion

Follow-up/How To?



PyTorch

https://pytorch.org/docs/stable/notes/cuda.html#memory-management

`.half()`, `.float()`, NVIDIA Apex

TensorFlow

https://www.tensorflow.org/guide/gpu
https://www.tensorflow.org/guide/keras/mixed_precision