8-bit Inference with TensorRT

Szymon Migacz, NVIDIA

May 8, 2017

Intro

- Goal: Convert FP32 CNNs into INT8 without significant accuracy loss.
- Why: INT8 math has higher throughput, and lower memory requirements.
- Challenge: INT8 has significantly lower precision and dynamic range than FP32.
- Solution: Minimize loss of information when quantizing trained model weights to INT8 and during INT8 computation of activations.
- Result: Method was implemented in TensorRT. It does not require any additional fine tuning or retraining.

Outline

- INT8 compute
- Quantization
- Calibration
- Workflow in TensorRT
- Results

INT8 Inference

Challenge

- INT8 has significantly lower precision and dynamic range compared to FP32.

	Dynamic Range	Min Positive Value
FP32	-3.4×10^{38} ~ $+3.4 \times 10^{38}$	1.4×10^{-45}
FP16	-65504 ~ +65504	5.96×10^{-8}
INT8	-128 ~ +127	1

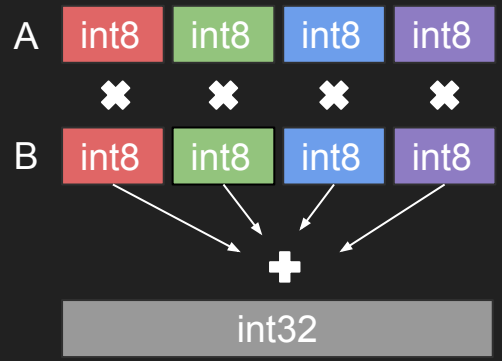
- Requires more than a simple type conversion from FP32 to INT8.

High-throughput INT8 math

DP4A - INT8 dot product

- Requires sm_61+ (Pascal TitanX, GTX 1080, Tesla P4, P40 and others).
- Four-way byte dot product accumulated in 32-bit result.

```
Result += A[0] * B[0] +
          A[1] * B[1] +
          A[2] * B[2] +
          A[3] * B[3]
```



Context

- Performance.
- No accuracy loss.
- Hence solution has to be “simple” and compute efficient.

Linear quantization

Representation:

Tensor Values = FP32 scale factor * int8 array + FP32 bias

Do we really need bias?

Two matrices:

```
A = scale_A * QA + bias_A
B = scale_B * QB + bias_B
```

Let's multiply those 2 matrices:

```
A * B = scale_A * scale_B * QA * QB +
        scale_A * QA * bias_B       +
        scale_B * QB * bias_A       +
        bias_A  * bias_B
```

Do we really need bias?

Two matrices:

```
A = scale_A * QA + bias_A
B = scale_B * QB + bias_B
```

Let's multiply those 2 matrices:

```
A * B = scale_A * scale_B * QA * QB +
        scale_A * QA * bias_B       +
        scale_B * QB * bias_A       +
        bias_A  * bias_B
```

Do we really need bias? No!

Two matrices:

```
A = scale_A * QA
B = scale_B * QB
```

Let’s multiply those 2 matrices:

```
A * B = scale_A * scale_B * QA * QB
```

Symmetric linear quantization

Representation:

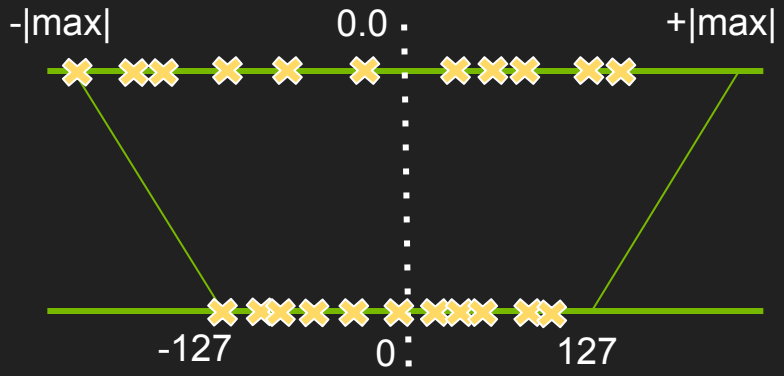
Tensor Values = FP32 scale factor * int8 array

One FP32 scale factor for the entire int8 tensor

Q: How do we set scale factor?

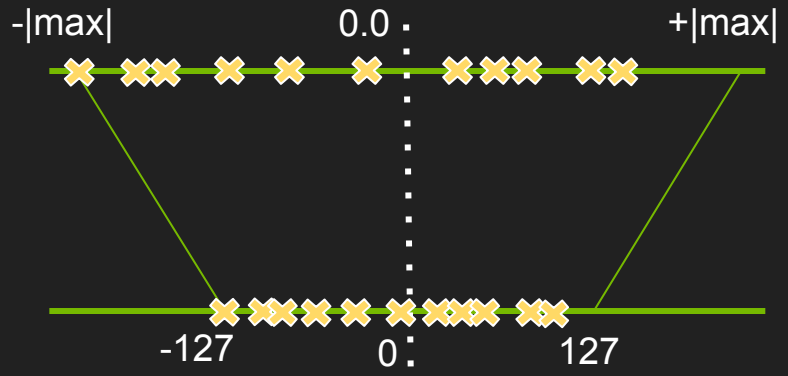
Quantization

- No saturation: map |max| to 127



Quantization

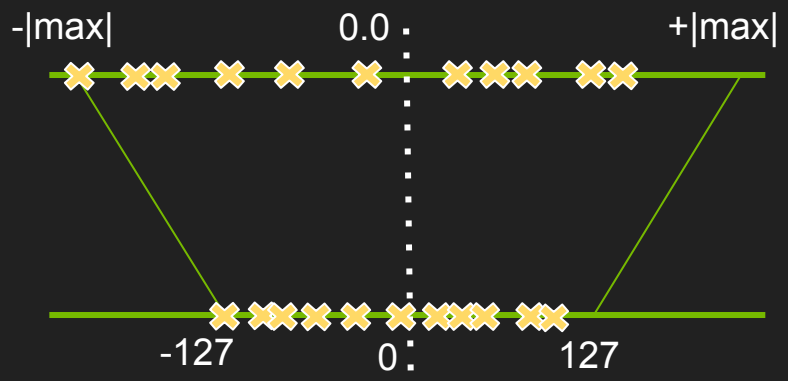
- No saturation: map |max| to 127



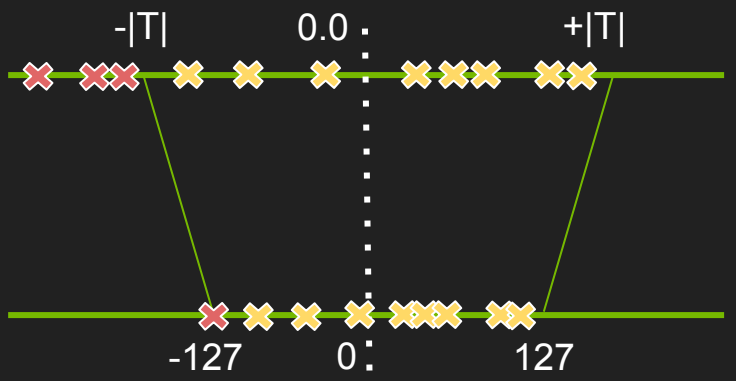
- Significant accuracy loss, in general

Quantization

- No saturation: map |max| to 127



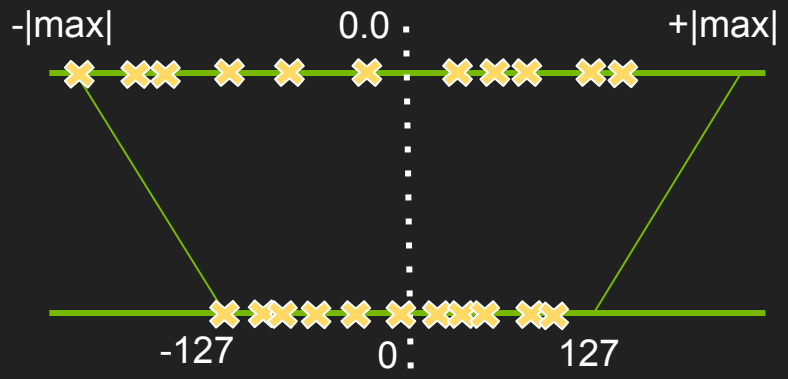
- Saturate above |threshold| to 127



- Significant accuracy loss, in general

Quantization

- No saturation: map |max| to 127



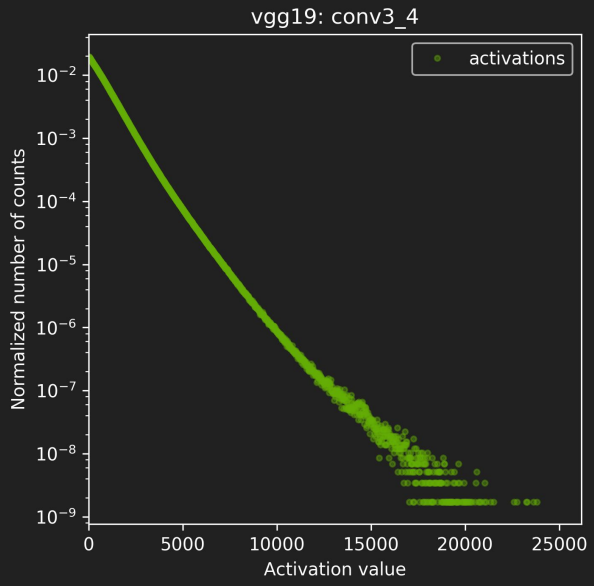
- Saturate above |threshold| to 127

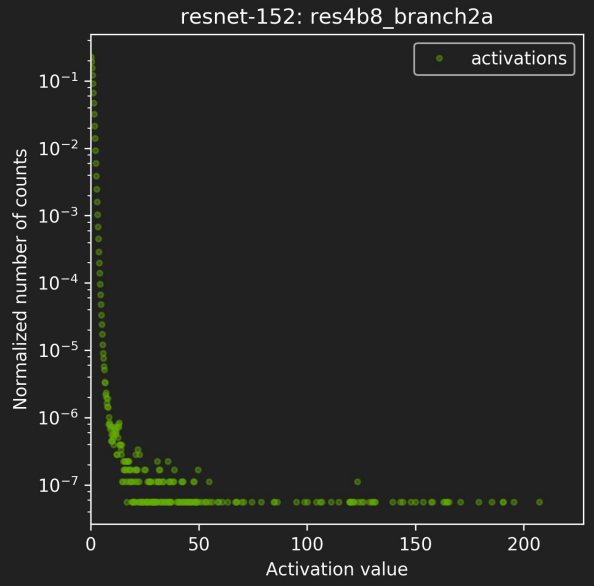
- Significant accuracy loss, in general

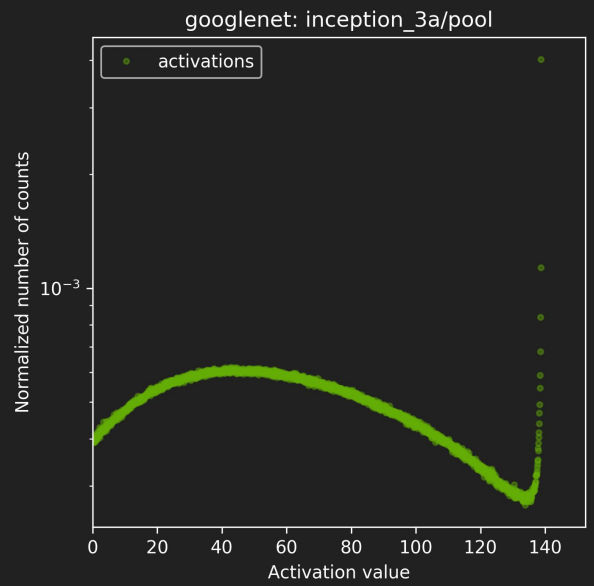
- Weights: no accuracy improvement
- Activations: improved accuracy
- Which |threshold| is optimal?

Q: How to optimize threshold selection?

- It’s always a tradeoff between range and precision of the INT8 representation.





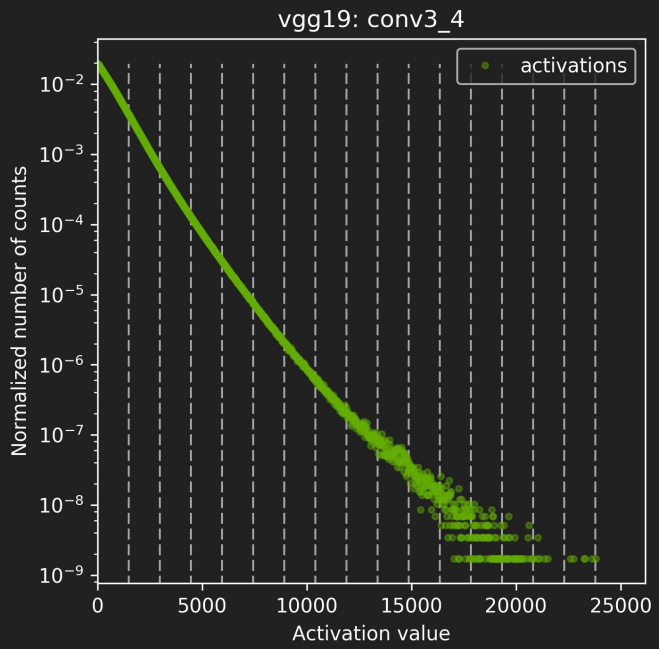


A: Minimize information loss, since FP32 → INT8 is just re-encoding information.

“Relative Entropy” of two encodings

- INT8 model encodes the same information as the original FP32 model.
- We want to minimize loss of information.
- Loss of information is measured by Kullback-Leibler divergence (AKA *relative entropy* or *information divergence*).
  - P, Q - two discrete probability distributions.
  - `KL_divergence(P,Q):= SUM(P[i] * log(P[i] / Q[i] ), i)`
- **Intuition:** KL divergence measures the amount of information lost when approximating a given encoding.

Solution: Calibration



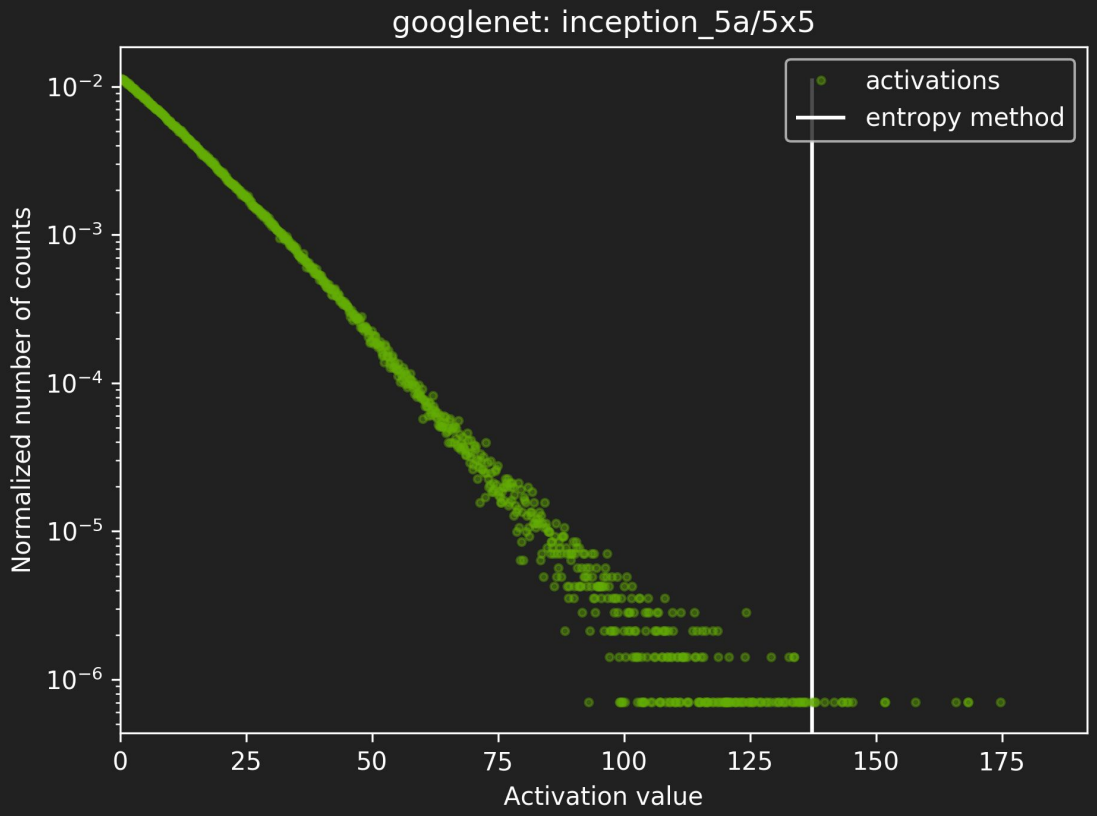
- Run FP32 inference on Calibration Dataset.
- For each Layer:
  - collect histograms of activations.
  - generate many quantized distributions with different saturation thresholds.
  - pick threshold which minimizes KL_divergence(ref_distr, quant_distr).
- Entire process takes a few minutes on a typical desktop workstation.

Calibration Dataset

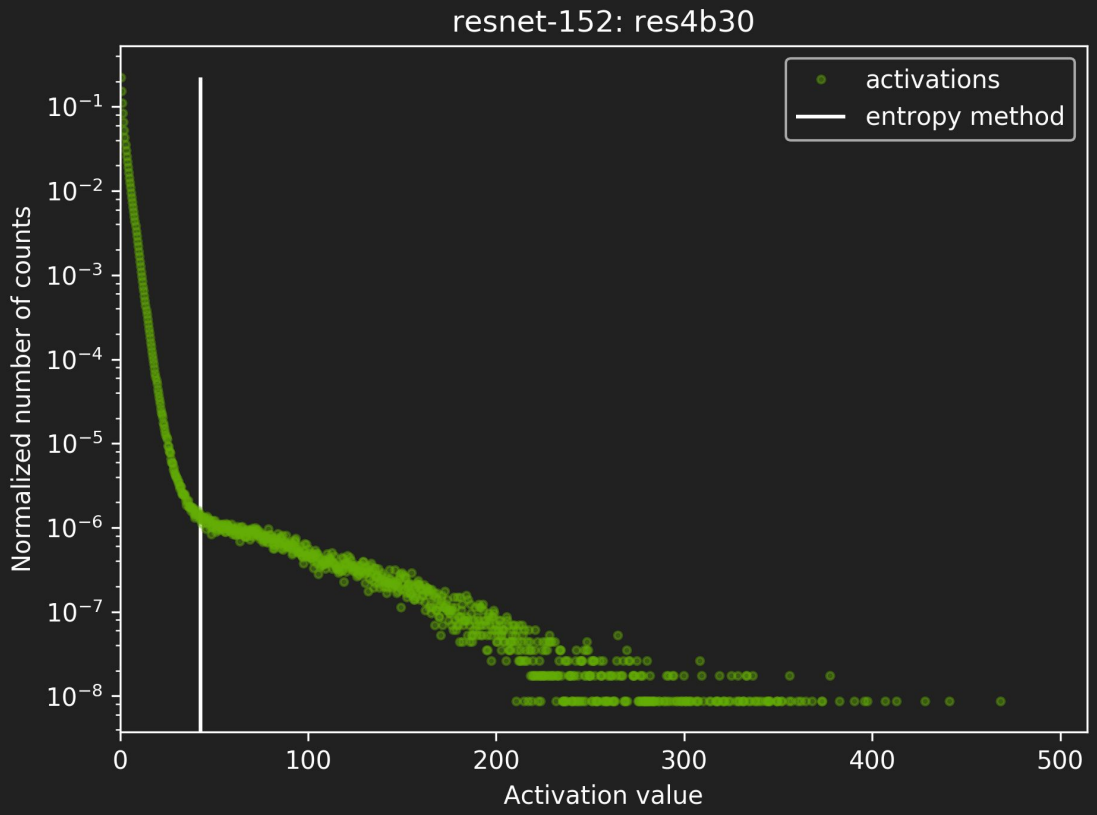
- Representative.
- Diverse.
- Ideally a subset of validation dataset.
- 1000s of samples

Results from Calibration

Results From Calibration #1

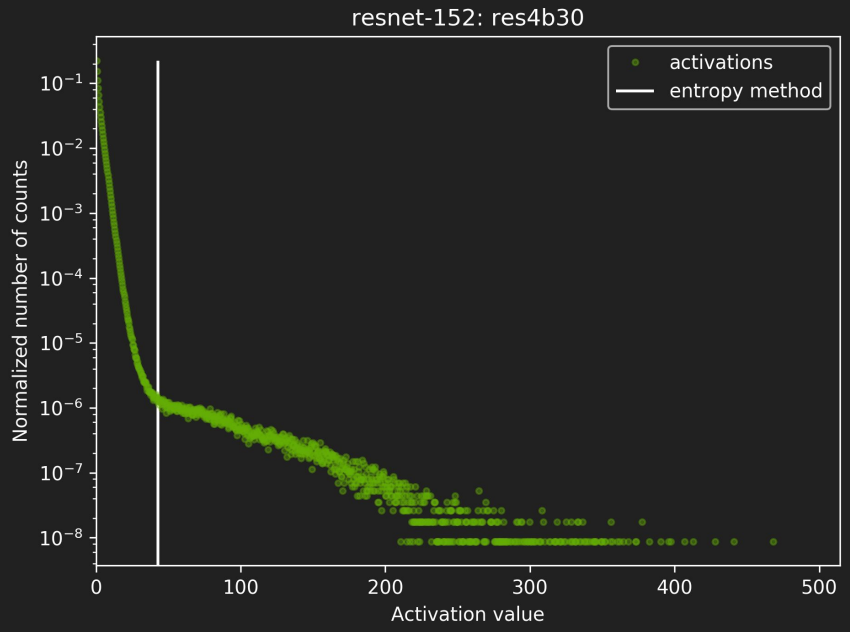


Results From Calibration #2

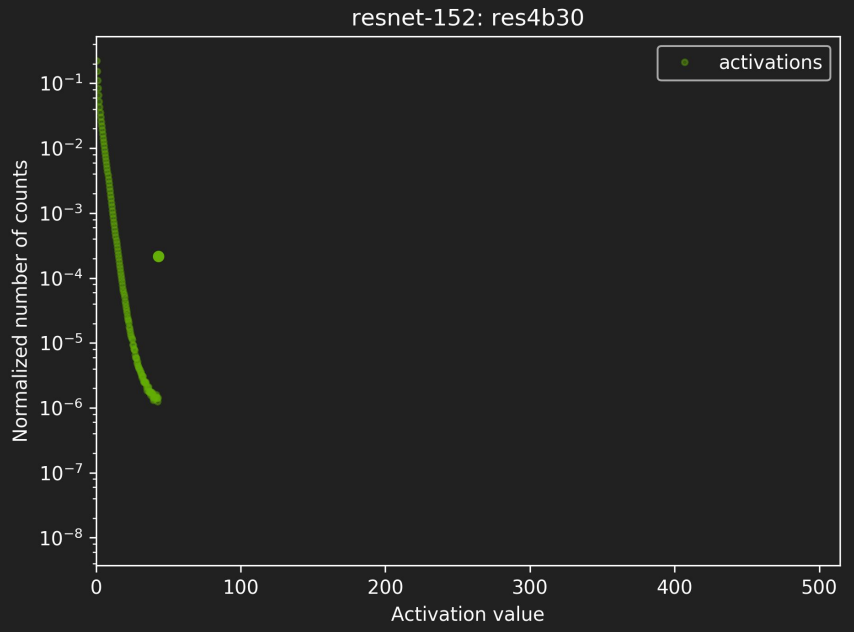


Results From Calibration #2

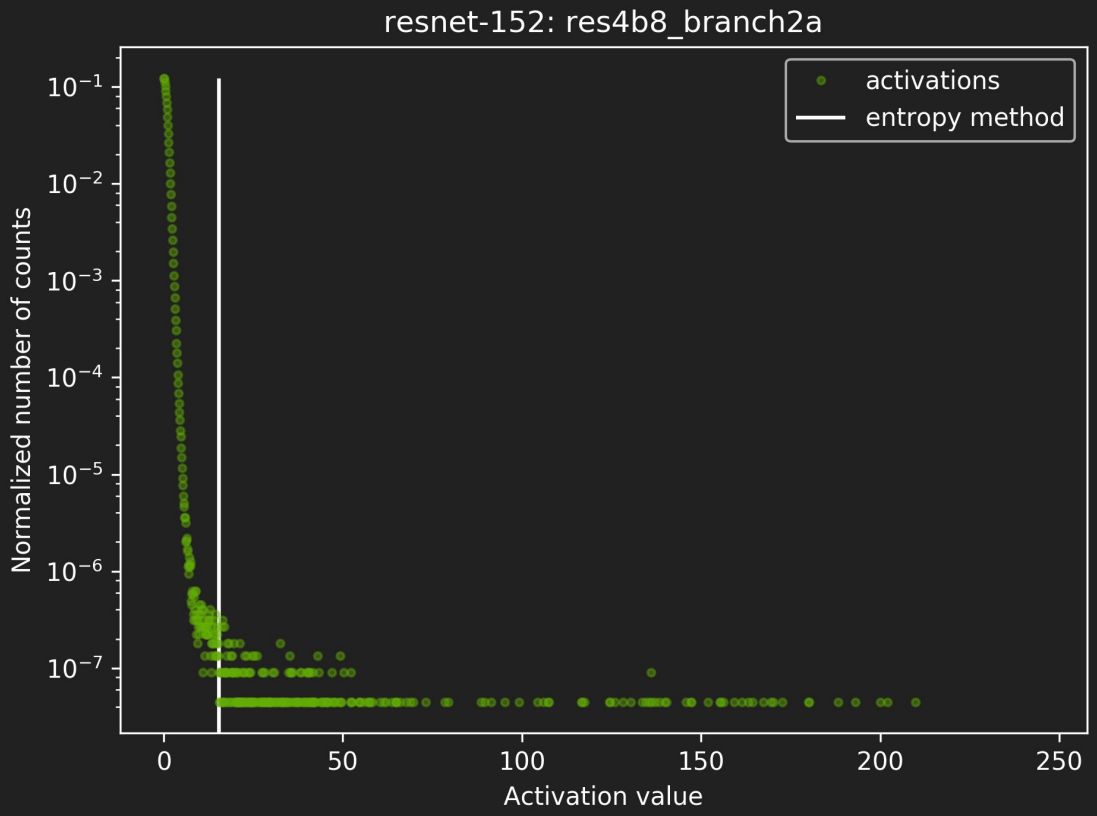
Before saturation



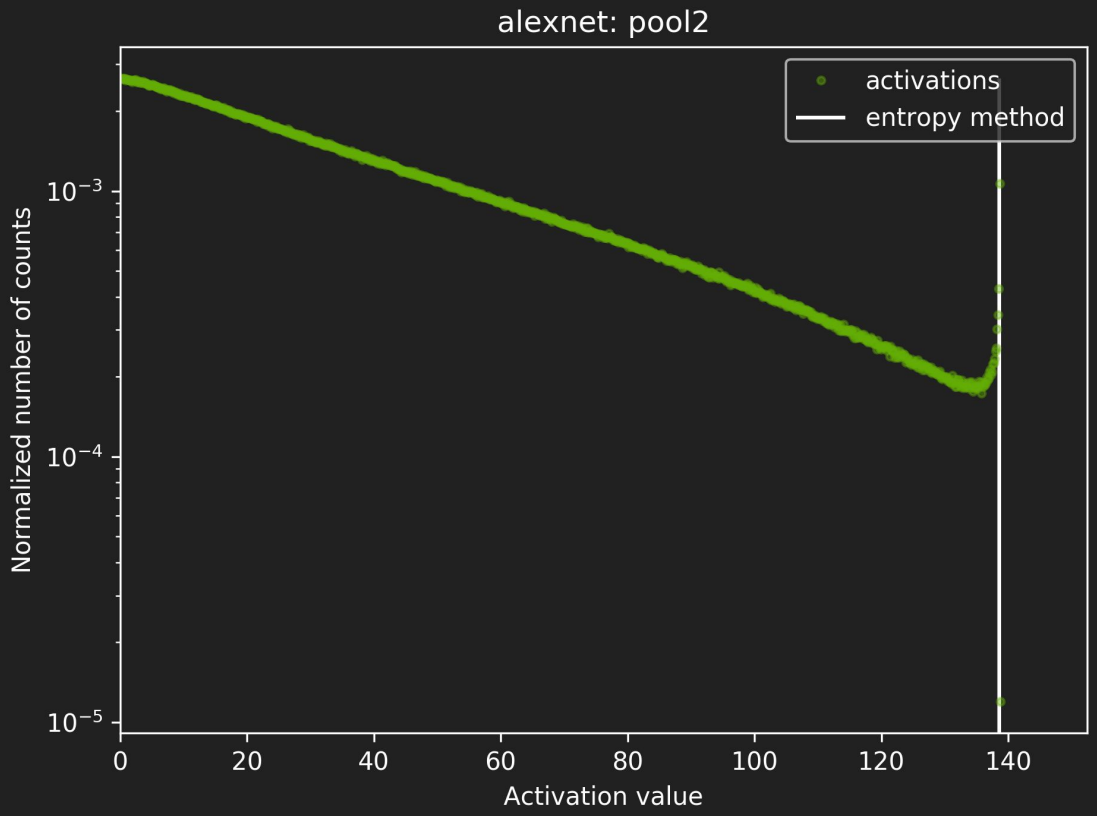
After saturation



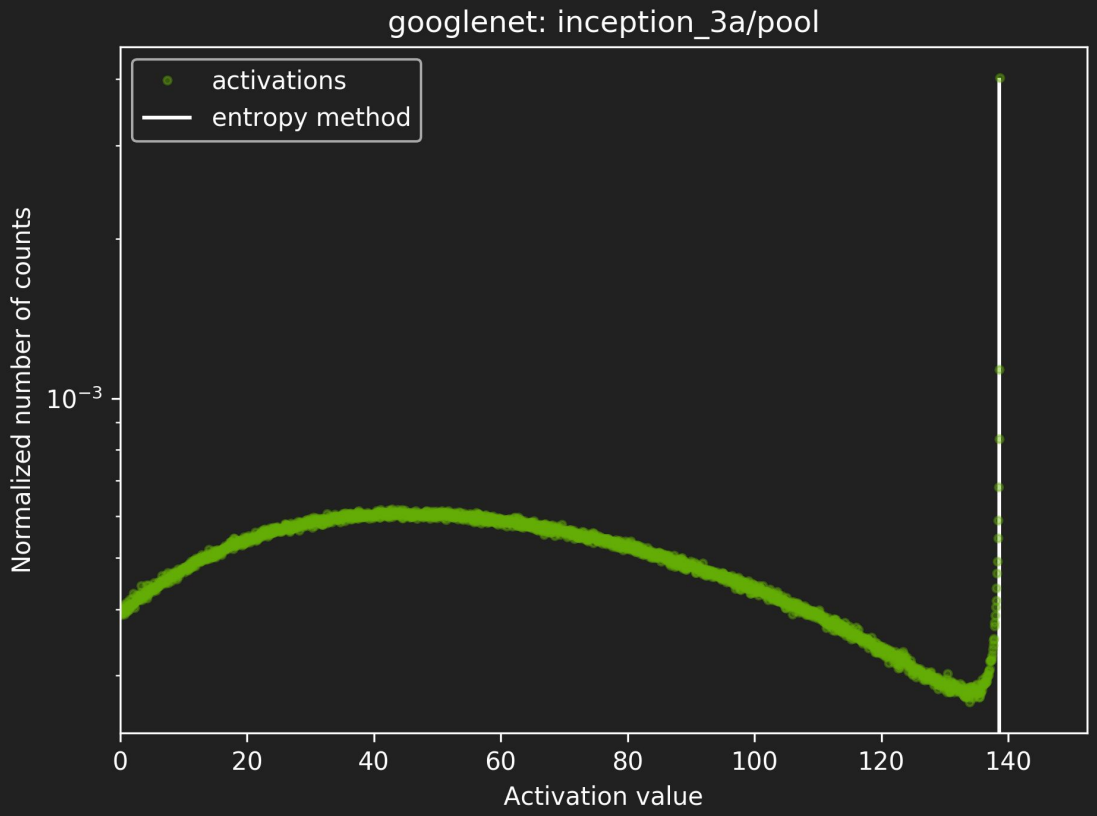
Results From Calibration #3



Results From Calibration #4



Results From Calibration #5



Workflow in TensorRT

Typical workflow in TensorRT

- You will need:
  - Model trained in FP32.
  - Calibration dataset.
- TensorRT will:
  - Run inference in FP32 on calibration dataset.
  - Collect required statistics.
  - Run calibration algorithm → optimal scaling factors.
  - Quantize FP32 weights → INT8.
  - Generate “CalibrationTable” and INT8 execution engine.

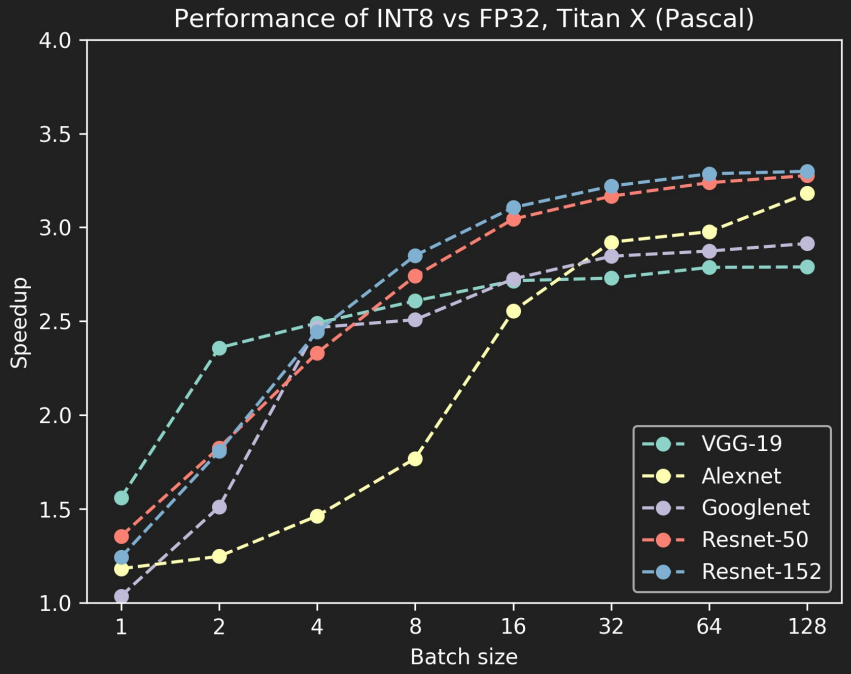
Results

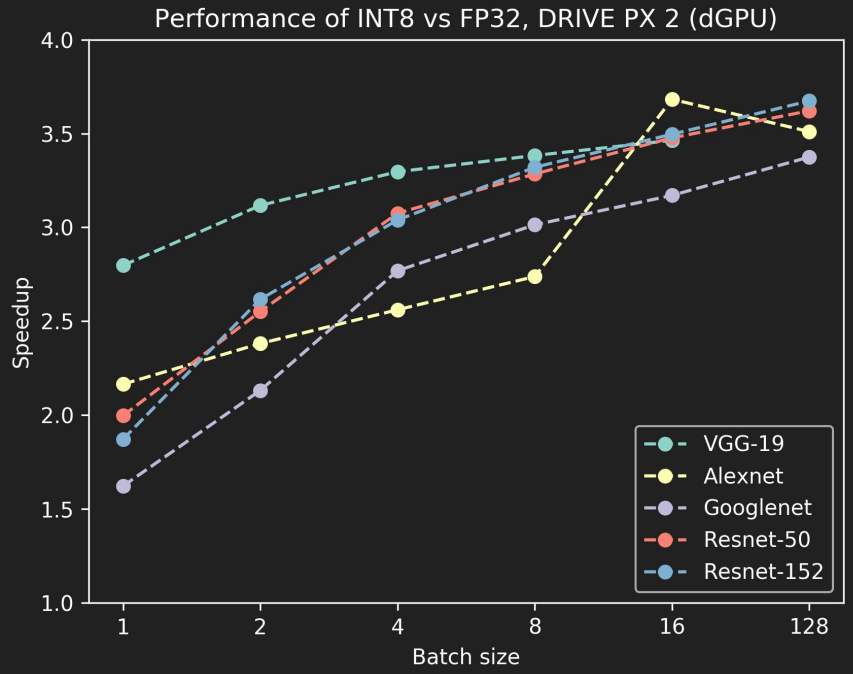
Results - Accuracy

	FP32		INT8					
			Calibration using 5 batches		Calibration using 10 batches		Calibration using 50 batches	
NETWORK	**Top1**	**Top5**	**Top1**	**Top5**	**Top1**	**Top5**	**Top1**	**Top5**
Resnet-50	73.23%	91.18%	73.03%	91.15%	73.02%	91.06%	73.10%	91.06%
Resnet-101	74.39%	91.78%	74.52%	91.64%	74.38%	91.70%	74.40%	91.73%
Resnet-152	74.78%	91.82%	74.62%	91.82%	74.66%	91.82%	74.70%	91.78%
VGG-19	68.41%	88.78%	68.42%	88.69%	68.42%	88.67%	68.38%	88.70%
Googlenet	68.57%	88.83%	68.21%	88.67%	68.10%	88.58%	68.12%	88.64%
Alexnet	57.08%	80.06%	57.00%	79.98%	57.00%	79.98%	57.05%	80.06%
NETWORK	**Top1**	**Top5**	**Diff Top1**	**Diff Top5**	**Diff Top1**	**Diff Top5**	**Diff Top1**	**Diff Top5**
Resnet-50	73.23%	91.18%	**0.20%**	**0.03%**	**0.22%**	**0.13%**	**0.13%**	**0.12%**
Resnet-101	74.39%	91.78%	**-0.13%**	**0.14%**	**0.01%**	**0.09%**	**-0.01%**	**0.06%**
Resnet-152	74.78%	91.82%	**0.15%**	**0.01%**	**0.11%**	**0.01%**	**0.08%**	**0.05%**
VGG-19	68.41%	88.78%	**-0.02%**	**0.09%**	**-0.01%**	**0.10%**	**0.03%**	**0.07%**
Googlenet	68.57%	88.83%	**0.36%**	**0.16%**	**0.46%**	**0.25%**	**0.45%**	**0.19%**
Alexnet	57.08%	80.06%	**0.08%**	**0.08%**	**0.08%**	**0.07%**	**0.03%**	**-0.01%**

TensorRT 2.1, all optimizations enabled. ILSVRC2012 validation dataset, batch = 25 images.
Accuracy was measured on 500 batches which were not used for the calibration.

Results - Performance





TensorRT 2.1, all optimizations enabled.

Open challenges / improvements

- Unsigned int8 for activations after ReLU.
- RNNs → open research problem.
- Fine tuning of saturation thresholds.
- Expose API for accepting custom, user provided scale factors.

Conclusion

- We introduced an automated, parameterless method for converting FP32 CNN models into INT8.
- Symmetric, linear quantization for weights and activations.
- Quantize original FP32 data such that the information loss is minimized.
- Popular, publicly available CNN models trained in FP32 can be converted to INT8, accuracy of INT8 models is comparable with the FP32 baseline.

Additional Resources

- We are going to publish whitepaper with description of the method.
- TensorRT 2.1 is going to be released soon.
- TensorRT 2.1 → sampleINT8.
- S7458 - DEPLOYING UNIQUE DL NETWORKS AS MICRO-SERVICES WITH TENSORRT, USER EXTENSIBLE LAYERS, AND GPU REST ENGINE.
  - Tuesday, May 9, 4:30 PM - 4:55 PM.
- Connect With The Experts:
  - Monday, May 8, 2:00 PM - 3:00 PM, Pod B.
  - Tuesday, May 9, 2:00 PM - 3:00 PM, Pod C.
  - Wednesday, May 10, 3:00 PM - 4:00 PM, Pod B.

Thank You

Backup slides

Entropy Calibration - pseudocode

Input: FP32 histogram H with 2048 bins: bin[0], …, bin[2047]

For i in range(128 , 2048):

```
    reference_distribution_P = [ bin[ 0 ] , ..., bin[ i-1 ] ]                  // take first ‘ i ‘ bins from H
    outliers_count = sum( bin[ i ] , bin[ i+1 ] , … , bin[ 2047 ] )
    reference_distribution_P[ i-1 ] += outliers_count
    P /= sum(P)                                                                // normalize distribution P
    candidate_distribution_Q = quantize [ bin[ 0 ], …, bin[ i-1 ] ] into 128 levels  // explained later
    expand candidate_distribution_Q to ‘ i ’ bins                              // explained later
    Q /= sum(Q)                                                                // normalize distribution Q
    divergence[ i ] = KL_divergence( reference_distribution_P,  candidate_distribution_Q)
```

End For

Find index ‘m’ for which divergence[m] is minimal

threshold = (m + 0.5) * (width of a bin)

Candidate distribution Q

- KL_divergence(P, Q) requires that len(P) == len(Q)
- Candidate distribution Q is generated after merging ‘ i ’ bins from bin[0] to bin[i-1] into 128 bins
- Afterwards Q has to be ‘expanded’ again into ‘i’ bins

Here is a simple example: reference distribution P consisting of 8 bins, we want to quantize into 2 bins:

P = [1, 0, 2, 3, 5, 3, 1, 7]

we merge into 2 bins (8 / 2 = 4 consecutive bins are merged into one bin)

[1 + 0 + 2 + 3 , 5 + 3 + 1 + 7] = [6, 16]

then proportionally expand back to 8 bins, we **preserve empty bins** from the original distribution P:

Q = [6/3, 0, 6/3, 6/3, 16/4, 16/4, 16/4, 16/4] = [2, 0, 2, 2, 4, 4, 4, 4]

now we should normalize both distributions, after that we can compute KL_divergence

P /= sum(P) Q /= sum(Q)

result = KL_divergence(P, Q)

Pseudocode for the INT8 conv kernel

```
// I8 input tensors: I8_input, I8_weights,                I8 output tensors: I8_output
// F32 bias (original bias from the F32 model)
// F32 scaling factors: input_scale,  output_scale, weights_scale[K]

I32_gemm_out = I8_input * I8_weights                      // Compute INT8 GEMM (DP4A)

F32_gemm_out = (float)I32_gemm_out                        // Cast I32 GEMM output to F32 float

// At this point we have F32_gemm_out which is scaled by  ( input_scale * weights_scale[K] ),
// but to store the final result in int8 we need to have scale equal to "output_scale", so we have to rescale:
// (this multiplication is done in F32, *_gemm_out arrays are in NCHW format)
For i in 0, ... K-1:
        rescaled_F32_gemm_out[ :, i, :, :] =  F32_gemm_out[ :, i, :, :]   * [ output_scale / (input_scale * weights_scale[ i ] ) ]

// Add bias, to perform addition we have to rescale original F32 bias so that it's scaled with "output_scale"
rescaled_F32_gemm_out _with_bias = rescaled_F32_gemm_out + output_scale * bias

// Perform ReLU (in F32)
F32_result = ReLU(rescaled_F32_gemm_out _with_bias)

// Convert to INT8 and save to global
I8_output = Saturate( Round_to_nearest_integer( F32_result ) )
```

Results - Performance - Pascal Titan X

	batchsize = 1			batchsize = 2			batchsize = 8			batchsize = 32			batchsize = 128		
Network	INT8 [img/s]	FP32 [img/s]	Ratio	INT8 [img/s]	FP32 [img/s]	Ratio	INT8 [img/s]	FP32 [img/s]	Ratio	INT8 [img/s]	FP32 [img/s]	Ratio	INT8 [img/s]	FP32 [img/s]	Ratio
Resnet-50	562	415	**1.354**	1045	572	**1.825**	2572	938	**2.741**	3567	1126	3.166	3787	1156	**3.276**
Resnet-152	195	157	**1.242**	371	205	**1.807**	1017	357	**2.850**	1335	415	3.220	1437	436	**3.299**
VGG-16	393	261	**1.508**	606	257	**2.361**	984	382	**2.577**	1131	416	2.722	1178	426	**2.764**
VGG-19	345	221	**1.559**	523	222	**2.358**	812	311	**2.608**	916	336	2.729	946	339	**2.789**
Googlenet	945	913	**1.035**	1756	1163	**1.510**	4356	1737	**2.508**	6545	2300	2.846	7282	2499	**2.914**
Alexnet	972	823	**1.181**	1913	1534	**1.247**	6434	3638	**1.768**	13899	4758	2.921	18714	5882	**3.181**

TensorRT FP32 vs TensorRT INT8
Pascal TitanX

Results - Performance - DRIVE PX 2, dGPU

	batchsize = 1			batchsize = 2			batchsize = 4			batchsize = 16			batchsize = 128		
Network	**INT8 [img/s]**	**FP32 [img/s]**	**Ratio**	**INT8 [img/s]**	**FP32 [img/s]**	**Ratio**	**INT8 [img/s]**	**FP32 [img/s]**	**Ratio**	**INT8 [img/s]**	**FP32 [img/s]**	**Ratio**	**INT8 [img/s]**	**FP32 [img/s]**	**Ratio**
Resnet-50	295	148	**1.996**	462	181	**2.552**	627	204	**3.075**	811	233	**3.477**	902	249	**3.621**
Resnet-152	110	59	**1.871**	179	68	**2.617**	239	79	**3.039**	318	91	**3.496**	356	97	**3.674**
VGG-16	130	47	**2.757**	189	62	**3.029**	229	71	**3.220**	286	84	**3.411**	DNR	DNR	
VGG-19	114	41	**2.797**	162	52	**3.117**	191	58	**3.296**	233	67	**3.464**	DNR	DNR	
Googlenet	497	306	**1.622**	777	364	**2.131**	1131	408	**2.769**	1576	497	**3.170**	1784	529	**3.375**
Alexnet	436	202	**2.164**	828	348	**2.381**	1458	570	**2.561**	3106	844	**3.682**	4853	1382	**3.510**

TensorRT FP32 vs TensorRT INT8
DRIVE PX 2, dGPU