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.

<table>
<thead>
<tr>
<th></th>
<th>Dynamic Range</th>
<th>Min Positive Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FP32</td>
<td>-3.4 x 10^{38} ~ +3.4 x 10^{38}</td>
<td>1.4 x 10^{-45}</td>
</tr>
<tr>
<td>FP16</td>
<td>-65504 ~ +65504</td>
<td>5.96 x 10^{-8}</td>
</tr>
<tr>
<td>INT8</td>
<td>-128 ~ +127</td>
<td>1</td>
</tr>
</tbody>
</table>

- 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.

\[
\]
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:

\[
A = \text{scale}_A \times QA + \text{bias}_A \\
B = \text{scale}_B \times QB + \text{bias}_B
\]

Let’s multiply those 2 matrices:

\[
A \times B = \text{scale}_A \times \text{scale}_B \times QA \times QB + \\
\text{scale}_A \times QA \times \text{bias}_B + \\
\text{scale}_B \times QB \times \text{bias}_A + \\
\text{bias}_A \times \text{bias}_B
\]
Do we really need bias?

Two matrices:

\[ A = \text{scale}_A \times QA + \text{bias}_A \]
\[ B = \text{scale}_B \times QB + \text{bias}_B \]

Let’s multiply those 2 matrices:

\[ A \times B = \text{scale}_A \times \text{scale}_B \times QA \times QB + \text{scale}_A \times QA \times \text{bias}_B \]
\[ + \text{scale}_B \times QB \times \text{bias}_A \]
\[ + \text{bias}_A \times \text{bias}_B \]
Do we really need bias? No!

Two matrices:

\[ A = \text{scale}_A \times QA \]
\[ B = \text{scale}_B \times QB \]

Let’s multiply those 2 matrices:

\[ A \times B = \text{scale}_A \times \text{scale}_B \times QA \times QB \]
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 $|\text{max}|$ to 127

![Diagram showing quantization with no saturation](image)
Quantization

- **No saturation**: map $|\text{max}|$ to 127

- **Significant accuracy loss**, in general
Quantization

- **No saturation**: map $|\text{max}|$ to 127

- **Saturate above $|\text{threshold}|$ to 127

- **Significant accuracy loss**, in general
Quantization

- **No saturation**: map $|\text{max}|$ to 127

- **Saturate above $|\text{threshold}|$ to 127

- **Significant accuracy loss**, in general

- **Weights**: no accuracy improvement
- **Activations**: improved accuracy

- **Which $|\text{threshold}|$ 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.
  - $KL_{\text{divergence}}(P,Q):= \sum (P[i] \times \log(P[i]/Q[i]), i)$
- 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

googleNet: inception_5a/5x5

Normalized number of counts

Activation value
Results From Calibration #2

resnet-152: res4b30

- activations
- entropy method

Normalized number of counts vs. Activation value
Results From Calibration #2

Before saturation

After saturation
Results From Calibration #3
Results From Calibration #4

![Graph showing normalized number of counts vs. activation value for AlexNet pool2. The graph compares activations and entropy method.](image-url)
Results From Calibration #5

googlenet: inception_3a/pool

Normalized number of counts

Activation value

activations
entropy method
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 - Accuracy

<table>
<thead>
<tr>
<th>NETWORK</th>
<th>Calibrate with 5 batches</th>
<th>Calibrate with 10 batches</th>
<th>Calibrate with 50 batches</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Top1</td>
<td>Top5</td>
<td>Top1</td>
</tr>
<tr>
<td>Resnet-50</td>
<td>73.23%</td>
<td>91.18%</td>
<td>73.03%</td>
</tr>
<tr>
<td>Resnet-101</td>
<td>74.39%</td>
<td>91.78%</td>
<td>74.52%</td>
</tr>
<tr>
<td>Resnet-152</td>
<td>74.78%</td>
<td>91.82%</td>
<td>74.62%</td>
</tr>
<tr>
<td>VGG-19</td>
<td>68.41%</td>
<td>88.78%</td>
<td>68.42%</td>
</tr>
<tr>
<td>Googlenet</td>
<td>68.57%</td>
<td>88.83%</td>
<td>68.21%</td>
</tr>
<tr>
<td>Alexnet</td>
<td>57.08%</td>
<td>80.06%</td>
<td>57.00%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NETWORK</th>
<th>Diff Top1</th>
<th>Diff Top5</th>
<th>Diff Top1</th>
<th>Diff Top5</th>
<th>Diff Top1</th>
<th>Diff Top5</th>
<th>Diff Top1</th>
<th>Diff Top5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resnet-50</td>
<td>0.20%</td>
<td>0.03%</td>
<td>0.22%</td>
<td>0.13%</td>
<td>0.13%</td>
<td>0.12%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Resnet-101</td>
<td>-0.13%</td>
<td>0.14%</td>
<td>0.01%</td>
<td>0.09%</td>
<td>-0.01%</td>
<td>0.06%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Resnet-152</td>
<td>0.15%</td>
<td>0.01%</td>
<td>0.11%</td>
<td>0.01%</td>
<td>0.08%</td>
<td>0.05%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VGG-19</td>
<td>-0.02%</td>
<td>0.09%</td>
<td>-0.01%</td>
<td>0.10%</td>
<td>0.03%</td>
<td>0.07%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Googlenet</td>
<td>0.36%</td>
<td>0.16%</td>
<td>0.46%</td>
<td>0.25%</td>
<td>0.45%</td>
<td>0.19%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Alexnet</td>
<td>0.08%</td>
<td>0.08%</td>
<td>0.08%</td>
<td>0.07%</td>
<td>0.03%</td>
<td>-0.01%</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

Performance of INT8 vs FP32, Titan X (Pascal)

Performance of INT8 vs FP32, DRIVE PX 2 (dGPU)

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.
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)\n\[ [1 + 0 + 2 + 3, 5 + 3 + 1 + 7] = [6, 16] \]
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
\[ P /= \text{sum}(P) \quad Q /= \text{sum}(Q) \]
\[ \text{result} = \text{KL\_divergence}(P, Q) \]
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 ) )
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Resnet-50</td>
<td>562</td>
<td>415</td>
<td><strong>1.354</strong></td>
<td>1045</td>
<td>572</td>
<td><strong>1.825</strong></td>
<td>2572</td>
<td>938</td>
<td><strong>2.741</strong></td>
<td>3567</td>
<td>1126</td>
<td><strong>3.166</strong></td>
<td>3787</td>
<td>1156</td>
<td><strong>3.276</strong></td>
</tr>
<tr>
<td>Resnet-152</td>
<td>195</td>
<td>157</td>
<td><strong>1.242</strong></td>
<td>371</td>
<td>205</td>
<td><strong>1.807</strong></td>
<td>1017</td>
<td>357</td>
<td><strong>2.850</strong></td>
<td>1335</td>
<td>415</td>
<td><strong>3.220</strong></td>
<td>1437</td>
<td>436</td>
<td><strong>3.299</strong></td>
</tr>
<tr>
<td>VGG-16</td>
<td>393</td>
<td>261</td>
<td><strong>1.508</strong></td>
<td>606</td>
<td>257</td>
<td><strong>2.361</strong></td>
<td>984</td>
<td>382</td>
<td><strong>2.577</strong></td>
<td>1131</td>
<td>416</td>
<td><strong>2.722</strong></td>
<td>1178</td>
<td>426</td>
<td><strong>2.764</strong></td>
</tr>
<tr>
<td>VGG-19</td>
<td>345</td>
<td>221</td>
<td><strong>1.559</strong></td>
<td>523</td>
<td>222</td>
<td><strong>2.358</strong></td>
<td>812</td>
<td>311</td>
<td><strong>2.608</strong></td>
<td>916</td>
<td>336</td>
<td><strong>2.729</strong></td>
<td>946</td>
<td>339</td>
<td><strong>2.789</strong></td>
</tr>
<tr>
<td>Googlenet</td>
<td>945</td>
<td>913</td>
<td><strong>1.035</strong></td>
<td>1756</td>
<td>1163</td>
<td><strong>1.510</strong></td>
<td>4356</td>
<td>1737</td>
<td><strong>2.508</strong></td>
<td>6545</td>
<td>2300</td>
<td><strong>2.846</strong></td>
<td>7282</td>
<td>2499</td>
<td><strong>2.914</strong></td>
</tr>
<tr>
<td>Alexnet</td>
<td>972</td>
<td>823</td>
<td><strong>1.181</strong></td>
<td>1913</td>
<td>1534</td>
<td><strong>1.247</strong></td>
<td>6434</td>
<td>3638</td>
<td><strong>1.768</strong></td>
<td>13899</td>
<td>4758</td>
<td><strong>2.921</strong></td>
<td>18714</td>
<td>5882</td>
<td><strong>3.181</strong></td>
</tr>
</tbody>
</table>

TensorRT FP32 vs TensorRT INT8
Pascal TitanX
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Resnet-50</td>
<td>295</td>
<td>148</td>
<td>1.996</td>
<td>462</td>
<td>181</td>
<td>2.552</td>
<td>627</td>
<td>204</td>
<td>3.075</td>
<td>811</td>
<td>233</td>
<td>3.477</td>
<td>902</td>
<td>249</td>
<td>3.621</td>
</tr>
<tr>
<td>Resnet-152</td>
<td>110</td>
<td>59</td>
<td>1.871</td>
<td>179</td>
<td>68</td>
<td>2.617</td>
<td>239</td>
<td>79</td>
<td>3.039</td>
<td>318</td>
<td>91</td>
<td>3.496</td>
<td>356</td>
<td>97</td>
<td>3.674</td>
</tr>
<tr>
<td>VGG-16</td>
<td>130</td>
<td>47</td>
<td>2.757</td>
<td>189</td>
<td>62</td>
<td>3.029</td>
<td>229</td>
<td>71</td>
<td>3.220</td>
<td>286</td>
<td>84</td>
<td>3.411</td>
<td>DNR</td>
<td>DNR</td>
<td></td>
</tr>
<tr>
<td>VGG-19</td>
<td>114</td>
<td>41</td>
<td>2.797</td>
<td>162</td>
<td>52</td>
<td>3.117</td>
<td>191</td>
<td>58</td>
<td>3.296</td>
<td>233</td>
<td>67</td>
<td>3.464</td>
<td>DNR</td>
<td>DNR</td>
<td></td>
</tr>
<tr>
<td>Googlenet</td>
<td>497</td>
<td>306</td>
<td>1.622</td>
<td>777</td>
<td>364</td>
<td>2.131</td>
<td>1131</td>
<td>408</td>
<td>2.769</td>
<td>1576</td>
<td>497</td>
<td>3.170</td>
<td>1784</td>
<td>529</td>
<td>3.375</td>
</tr>
<tr>
<td>Alexnet</td>
<td>436</td>
<td>202</td>
<td>2.164</td>
<td>828</td>
<td>348</td>
<td>2.381</td>
<td>1458</td>
<td>570</td>
<td>2.561</td>
<td>3106</td>
<td>844</td>
<td>3.682</td>
<td>4853</td>
<td>1382</td>
<td>3.510</td>
</tr>
</tbody>
</table>

TensorRT FP32 vs TensorRT INT8
DRIVE PX 2, dGPU