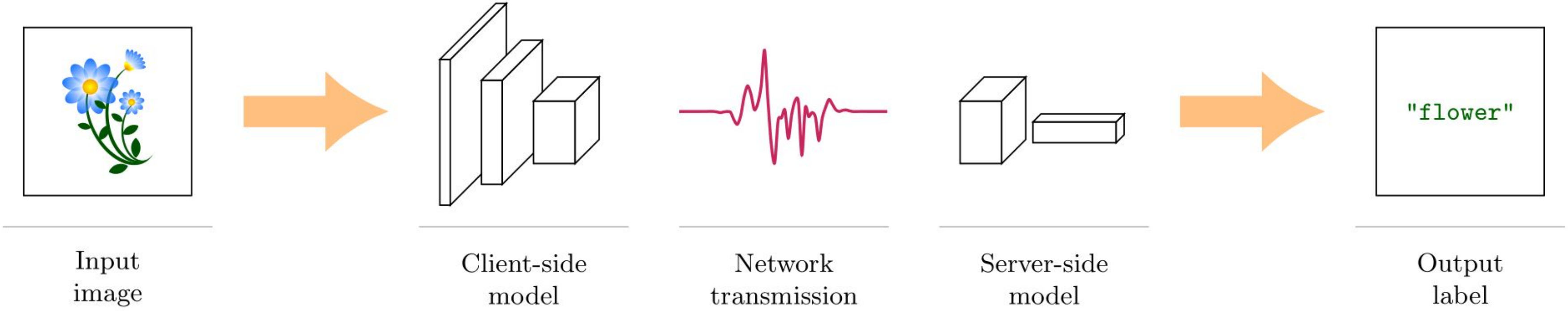
Shared Mobile-Cloud Inference for Collaborative Intelligence

Mateen Ulhaq



Outline

1. Background
2. Single tensor compression
3. Reusing image codecs
4. Towards tensor stream compression
5. Error concealment
6. Future work

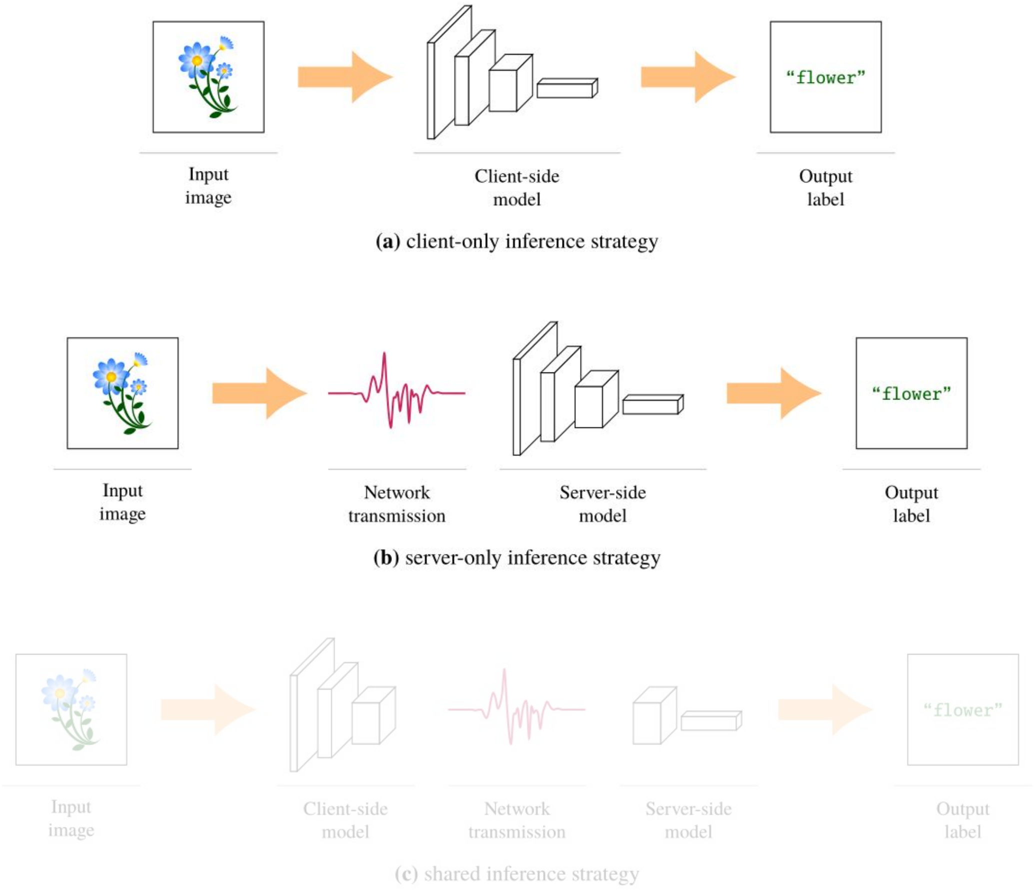
Background

Inference strategies

Inference of deep learning models is traditionally done directly on the mobile device (“client-only”) or in the cloud (“server-only”).

Client-only inference:
slower hardware than server;
limited to small models

Server-only inference:
depends on network connection quality;
consumes bandwidth and energy;
possible privacy concerns



Shared inference

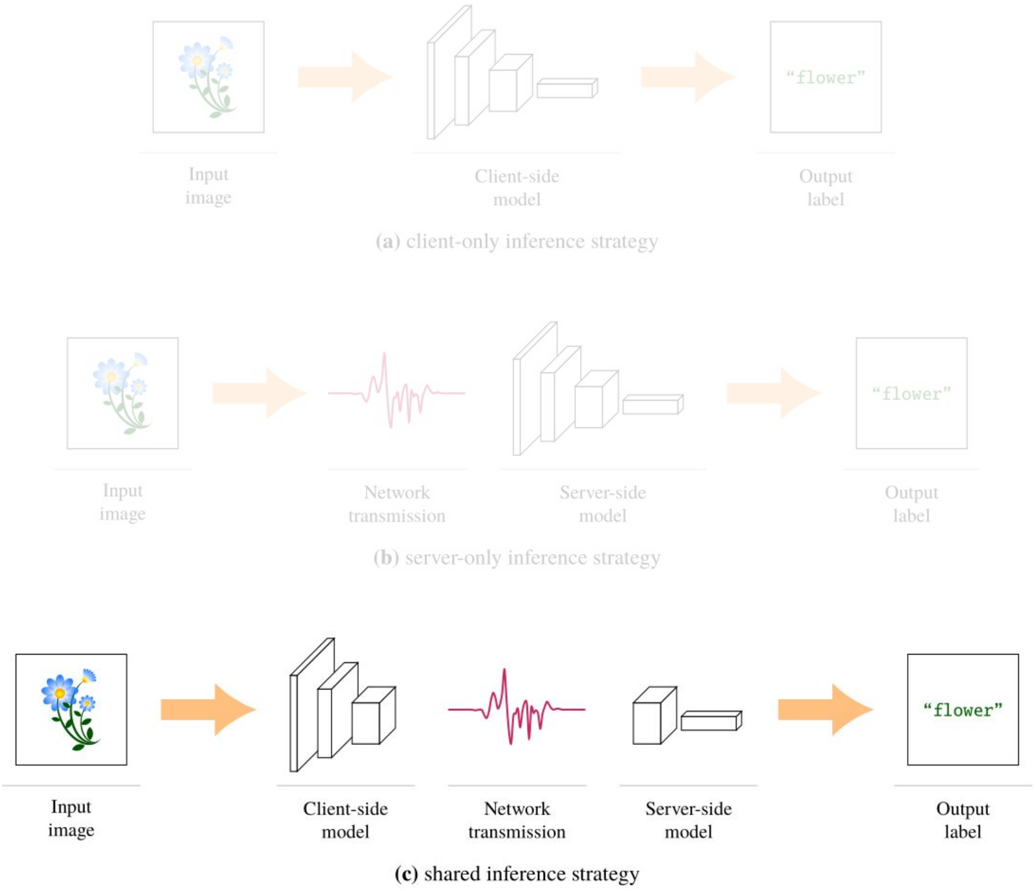
Key idea: reduce amount of data transmitted

Compared with client-only:

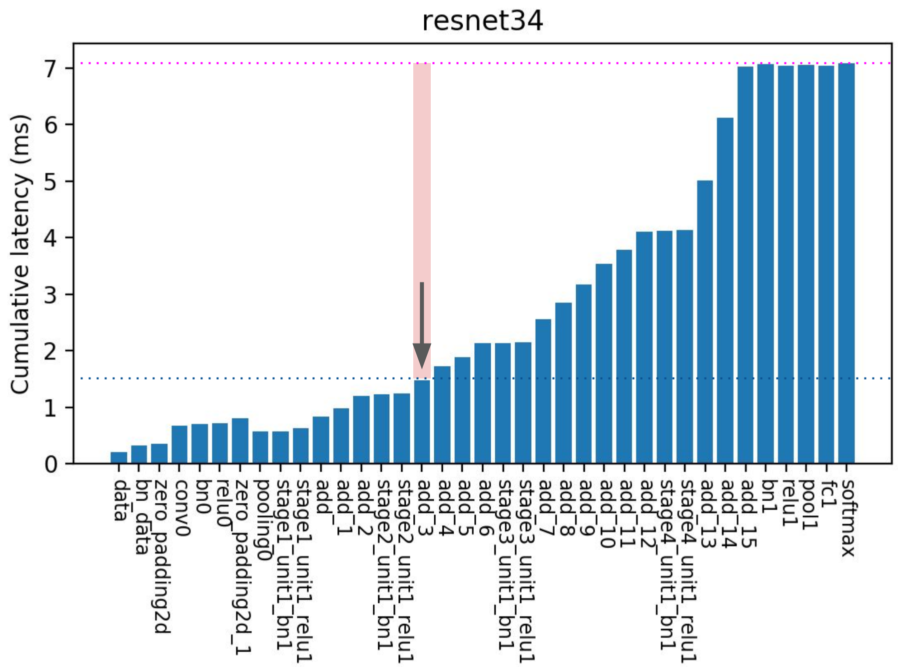
- Reduce inference times
- Reduce computational load

Compared with server-only:

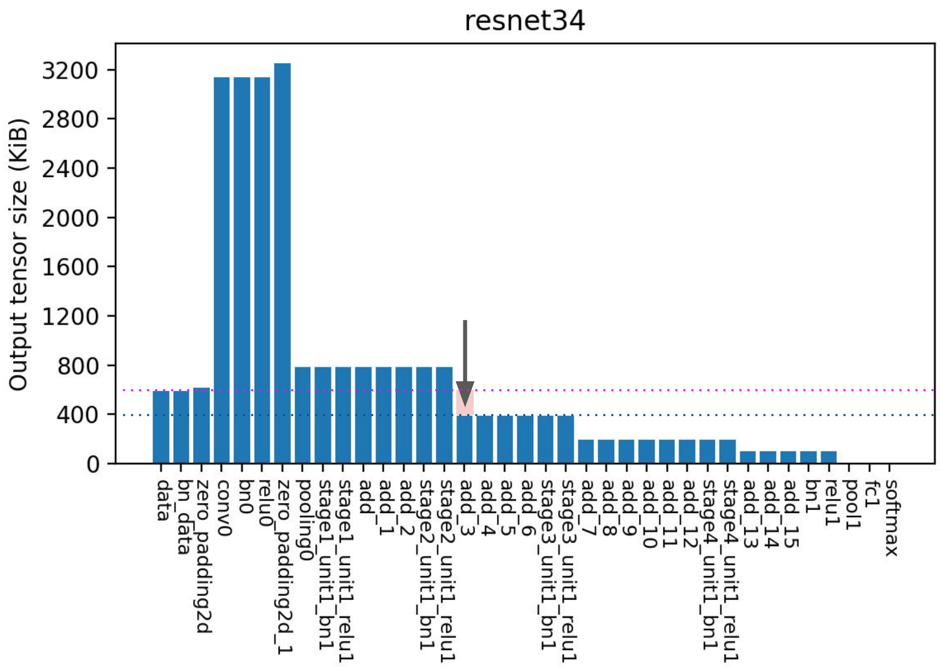
- Reduce inference times
- Save bandwidth
- Save device energy
- Better privacy



Layers of a deep learning model

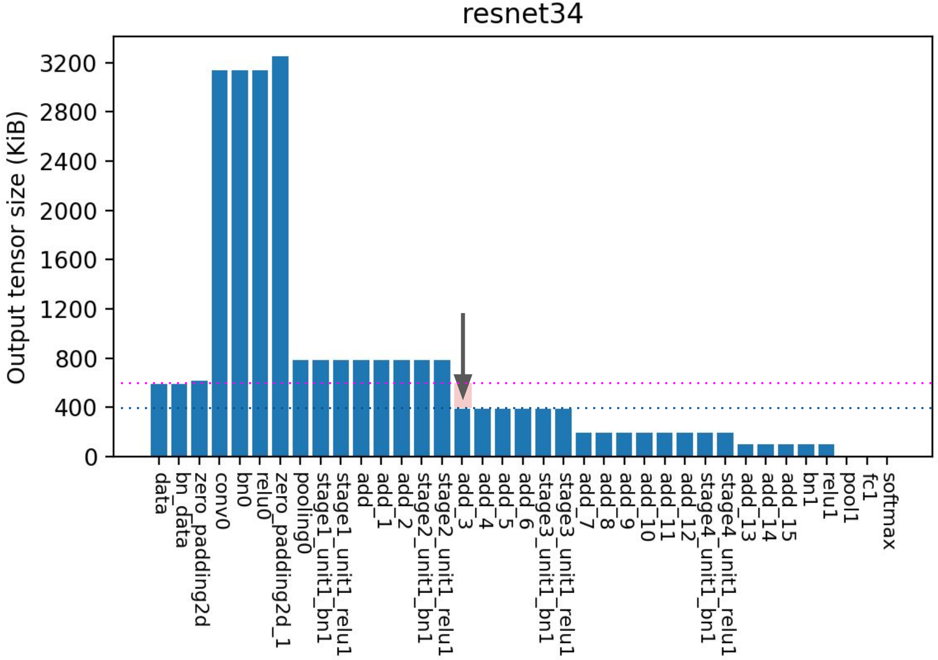


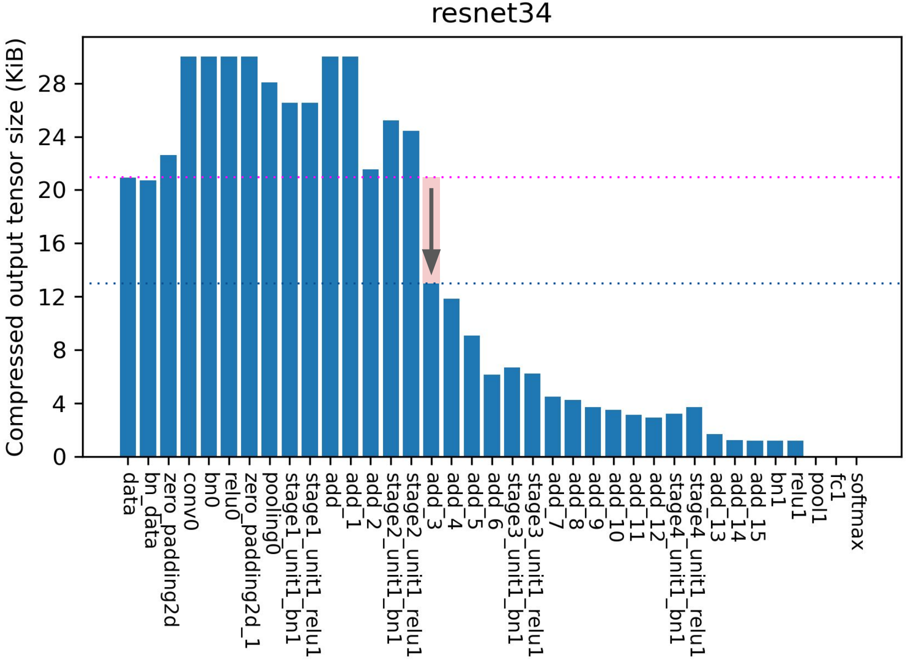
Cumulative inference time at layer



Uncompressed data size by layer

Layers of a deep learning model





Uncompressed data size by layer

Compressed data size by layer

Total inference time

$$I_t = \underbrace{I_c + E_c}_{\text{client-side}} + \underbrace{\frac{D}{B} + \mathrm{RTT}}_{\text{network}} + \underbrace{E_s + I_s}_{\text{server-side}} + \epsilon$$

I_t = total inference time

I_c = inference time of client-side model

I_s = inference time of server-side model

E_c = serialization/encoding time for client output tensor

E_s = deserialization/decoding time for server input tensor

D = size of serialized/compressed tensor data

B = rate of data transfer (bandwidth)

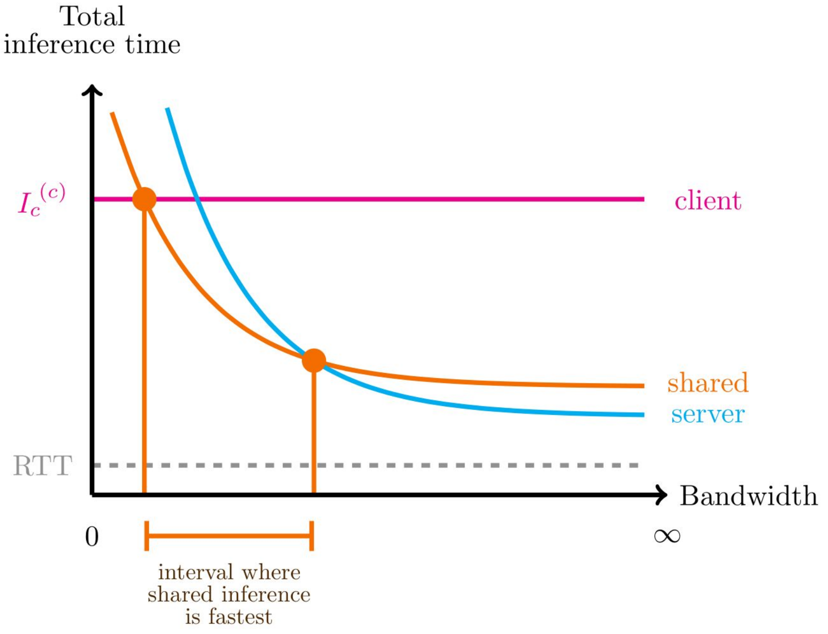
RTT = round trip time

ϵ = other latencies (negligible)

Total inference time

I_t = total inference time
I_c = inference time of client-side model
I_s = inference time of server-side model
E_c = serialization/encoding time for client output tensor
E_s = deserialization/decoding time for server input tensor
D = size of serialized/compressed tensor data
B = rate of data transfer (bandwidth)
RTT = round trip time
ϵ = other latencies (negligible)

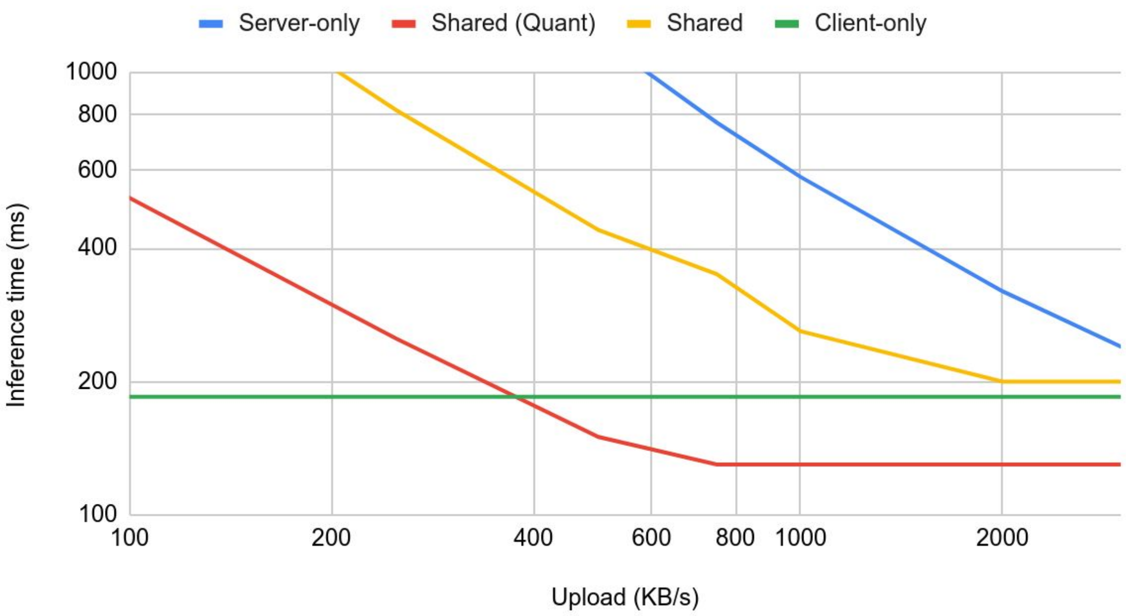
$$I_t = \underbrace{I_c + E_c}_{\text{client-side}} + \underbrace{\frac{D}{B} + \text{RTT}}_{\text{network}} + \underbrace{E_s + I_s}_{\text{server-side}} + \epsilon = b + \frac{D}{B}$$



Reasonable assumptions:

Amount of data transmitted: $0 = D^{(c)} < D^{(l)} < D^{(s)}$

Horizontal asymptotes: $b^{(s)} < b^{(l)} < b^{(c)}$



- In-lab experiments on Android device and remote server 5 km away with uncompressed tensors
- Shows similar trends as modelled on previous slide

Experimental tests

Prototype

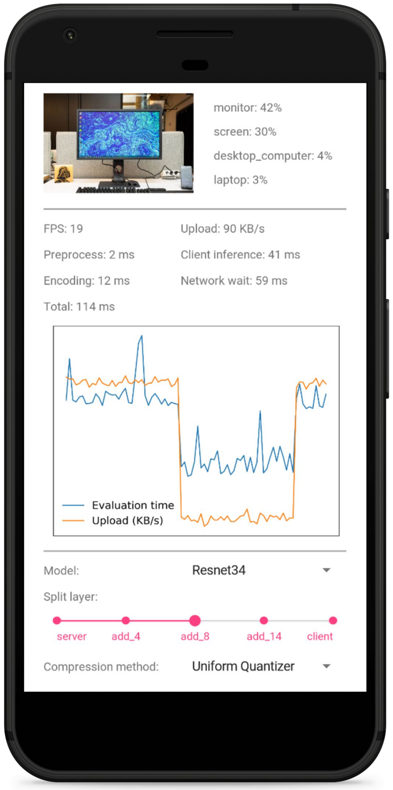
Demoed at NeurIPS 2019 conference.

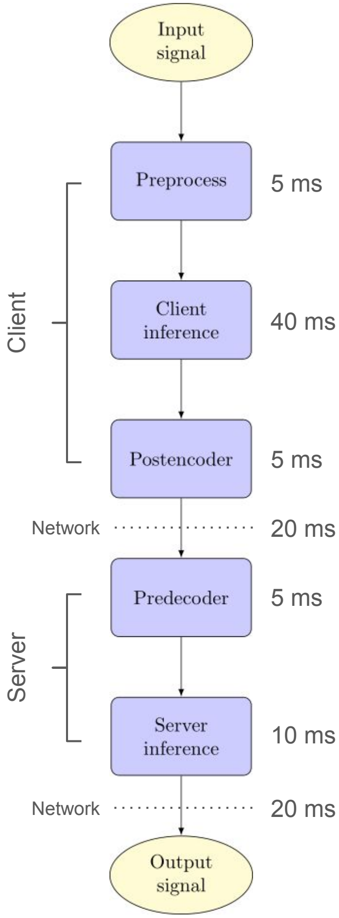
Client: Android; Kotlin, Tensorflow Lite

Server: Remote PC; Python, Tensorflow

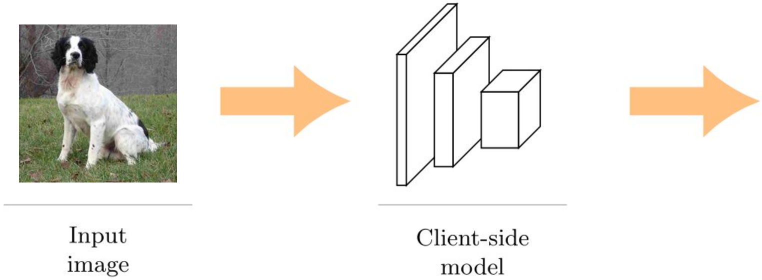
Low-latency, high-throughput shared inference:

- Process more than one frame at a time
- Synchronize to avoid “backpressure”





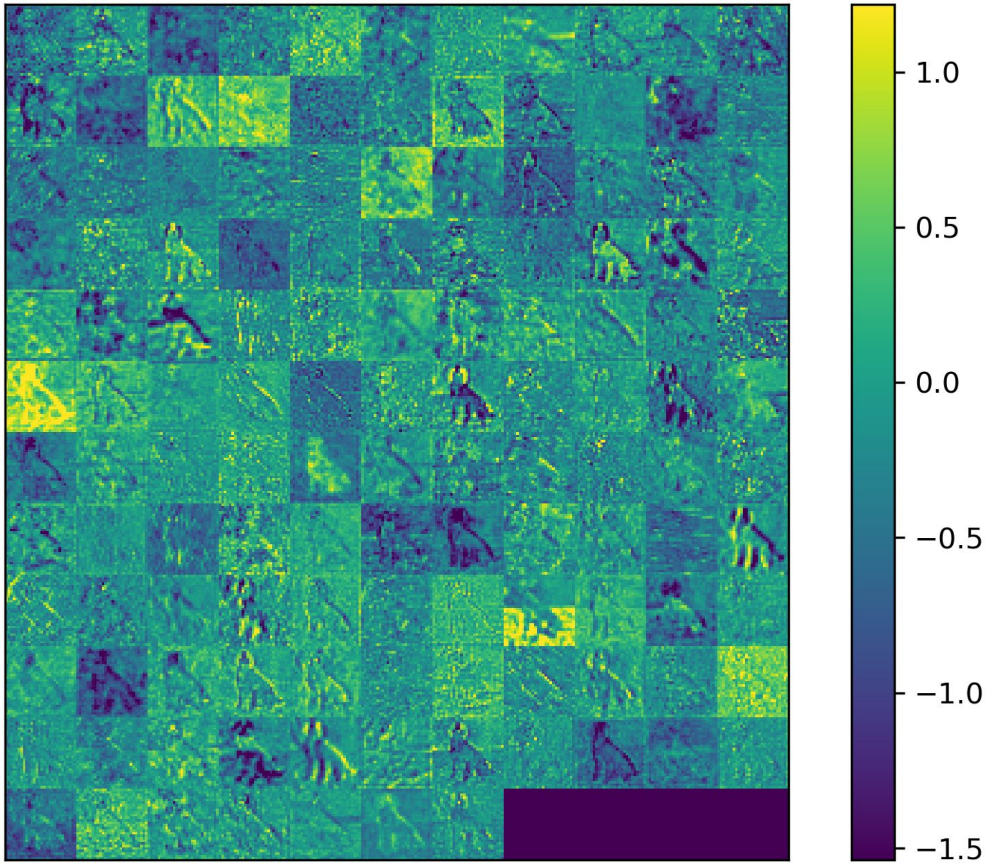
Single tensor compression



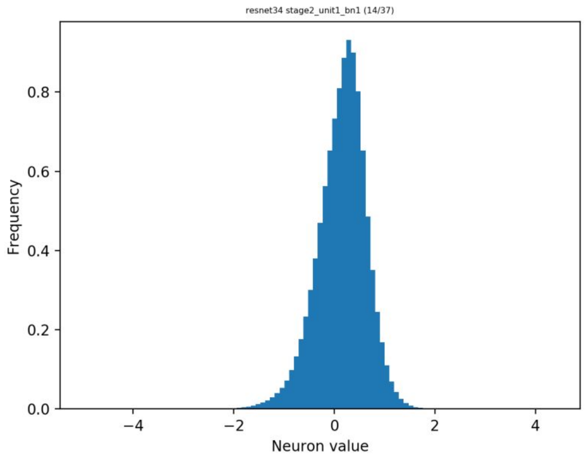
$$T(y, x, c) \approx T(y + \Delta y,\ x + \Delta x,\ c) \ \ \text{(intra-channel)}$$

$$S(c, c') \geq d\left(\frac{1}{\sigma_c}[T(c) - \mu_c],\ \frac{1}{\sigma_{c'}}[T(c') - \mu_{c'}]\right) \ \ \text{(inter-channel)}$$

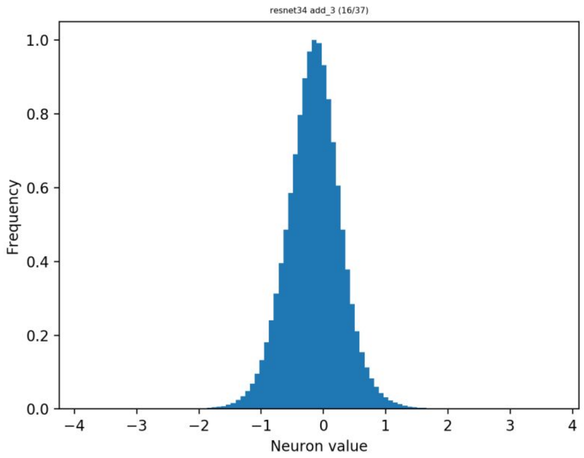
resnet34 add_3 (16/37)



Client-side inference featuremap



(a) BatchNorm layer



(b) post-BatchNorm layer

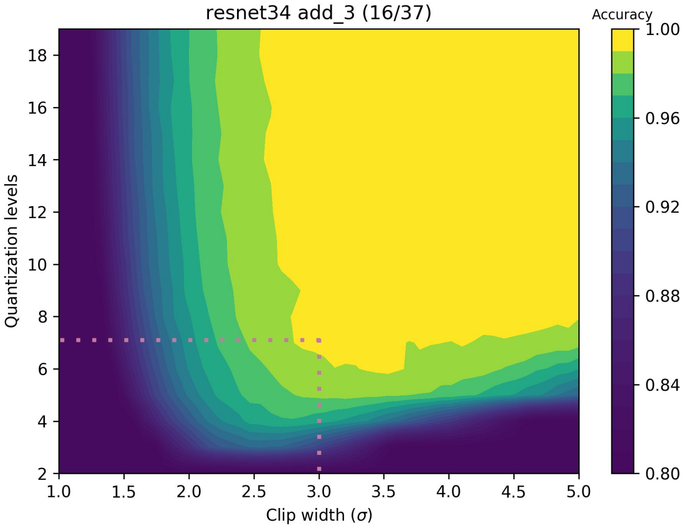
- Plotted: Mixture distribution of neuron output values at a layer
- Experimentally appears normal
- Most values are within 3 stddev about the mean

Distributions of neuron output values

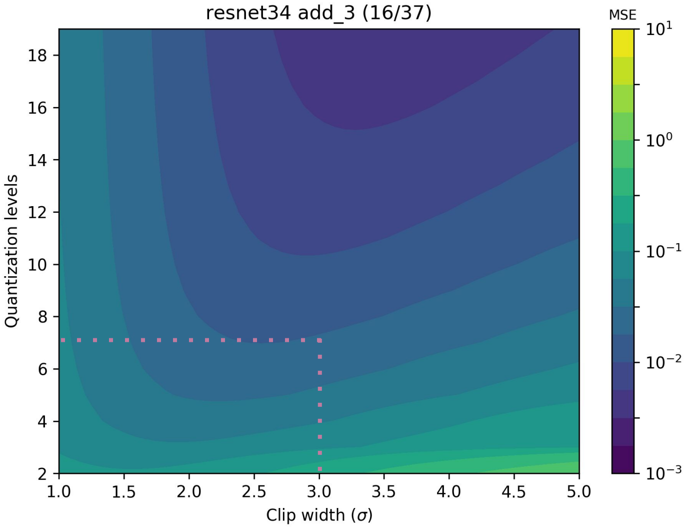
Top-1 accuracy

Reconstruction error (MSE)

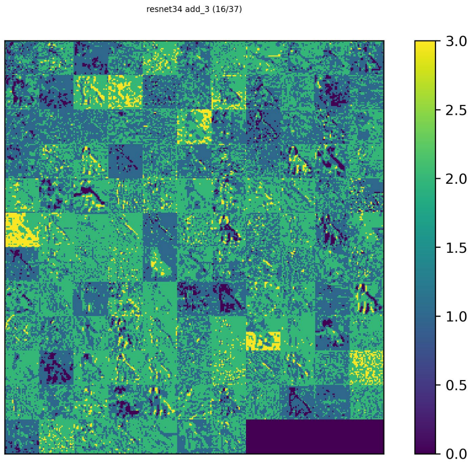
Quantized tensor



calculated over 16k samples from dataset



calculated over 16k samples from dataset



Uniform quantization

Reusing image codecs

Process

Client-side:

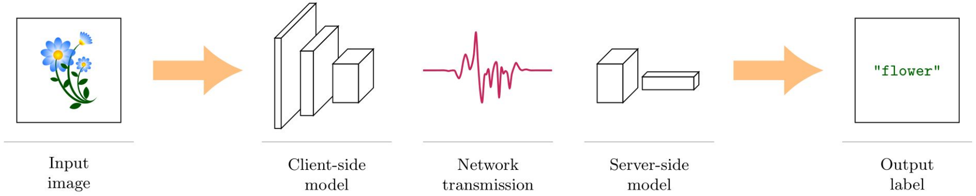
1. **Input** is a 224x224 image
2. **Client model inference** on image
3. **Quantize** 3D tensor
4. **Reshape** into 2D tensor
5. **Encode** via image codec

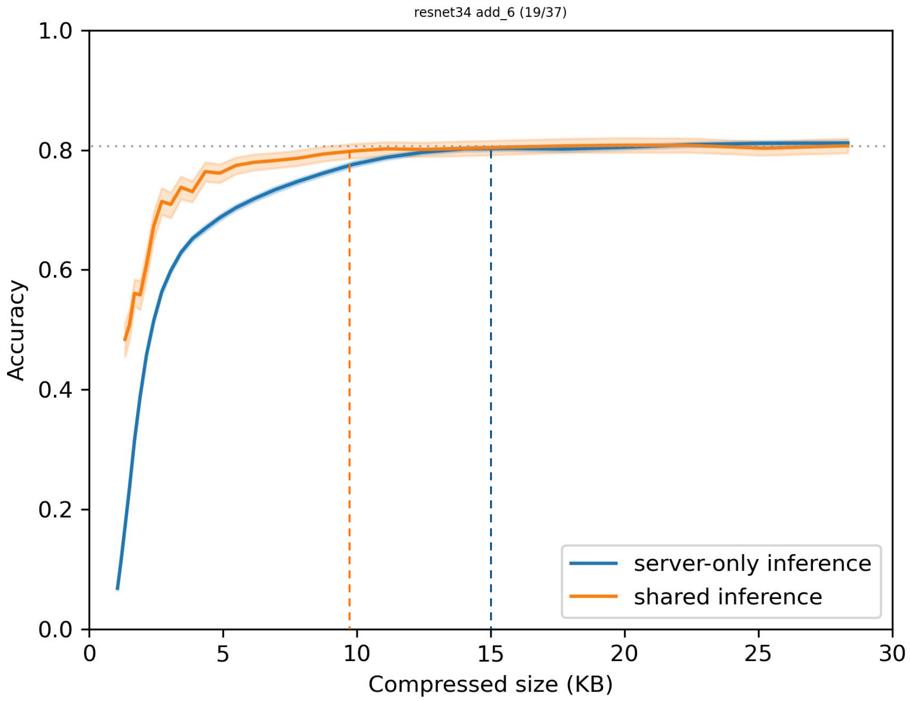
Compression

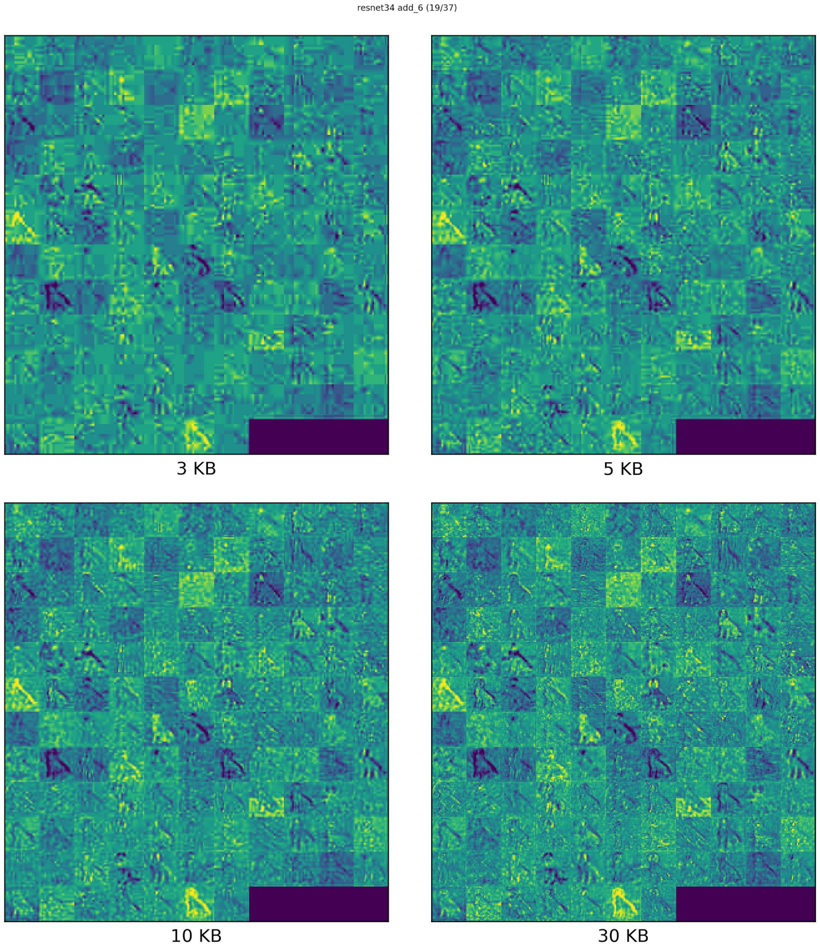
Server-side:

6. **Decode** via image codec
7. **Reshape** into 3D tensor
8. **Dequantize** 3D tensor
9. **Server model inference** on 3D tensor
10. **Output** is a probability vector

Decompression





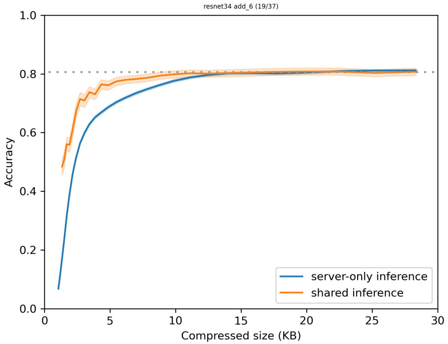


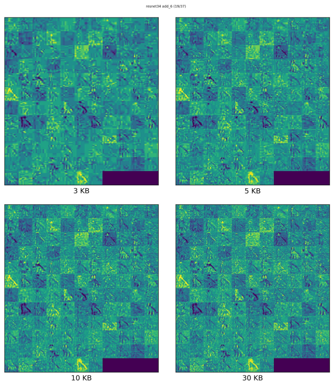
Accuracy vs compressed size: JPEG

Accuracy vs compressed size: experimental setup

Dataset generation

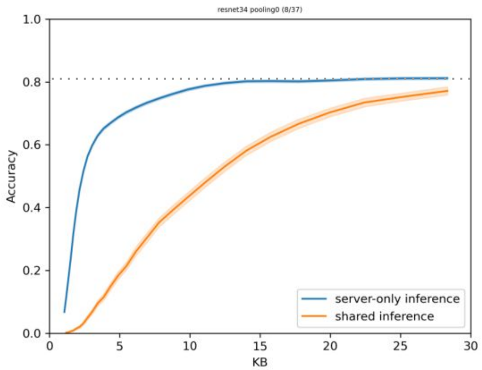
1. **ILSVRC 2012** (ImageNet, 1000 classes)
2. **Crop** to 1:1
3. **Downscale** to 224x224
4. **Save** as JPEG
5. **Keep** if file size is 30 ± 0.3 KB
6. **Keep** 16384 images



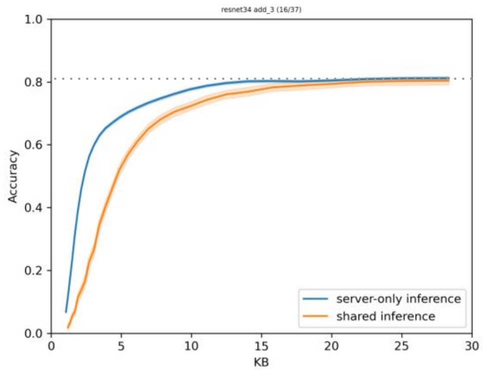


Experiment

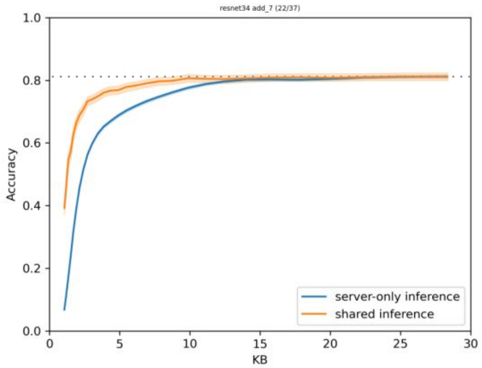
1. **Run client model inference** on each image
2. **Compress** each tensor via codec at various quality/bitrate settings, generating 100,000 different compressed tensors
3. **Bin by size** into logarithmically spaced bins
4. **Decompress** tensors
5. **Run server model inference**
6. **Compute top-1 accuracy** for each bin
7. **Plot** top-1 accuracy vs compressed size



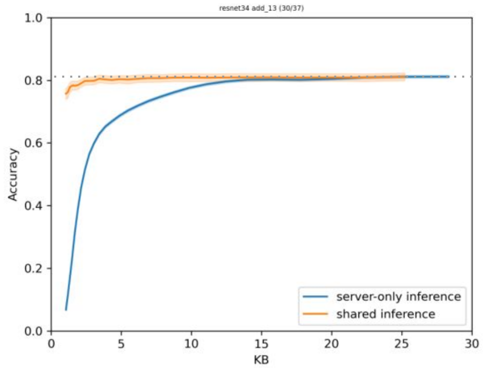
(a) ResNet-34, `pooling0` layer, $56 \times 56 \times 64$



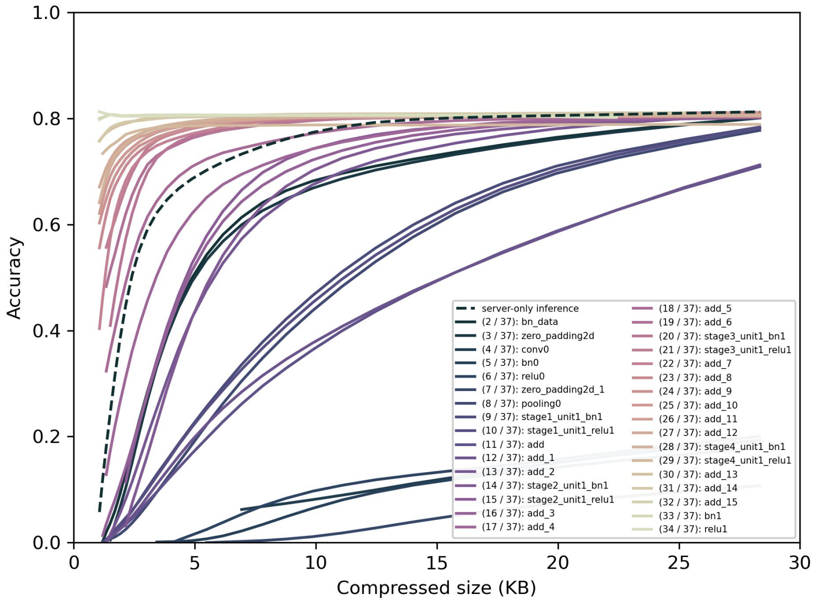
(b) ResNet-34, `add_3` layer, $28 \times 28 \times 128$



(c) ResNet-34, `add_7` layer, $14 \times 14 \times 256$

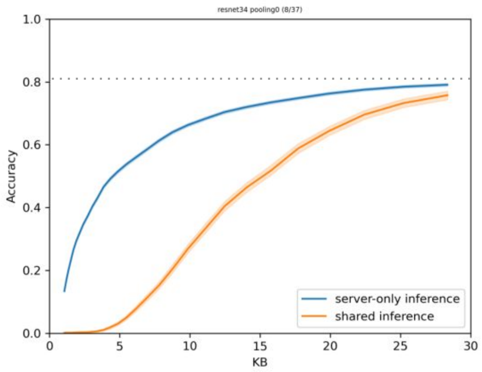


(d) ResNet-34, `add_13` layer, $7 \times 7 \times 512$

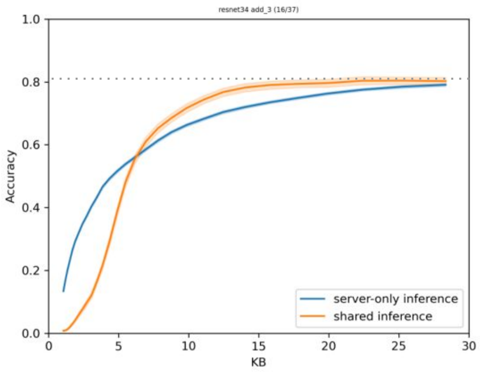


Shared inference accuracy curves generally improve as we go through deeper and deeper layers

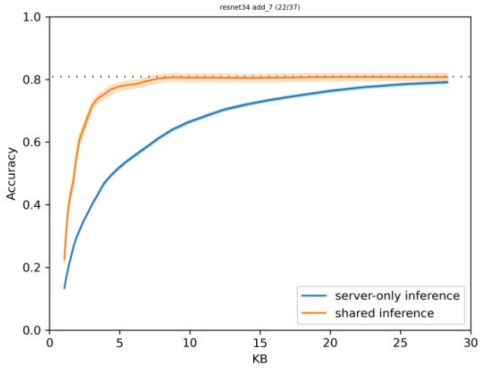
Accuracy vs compressed size: JPEG



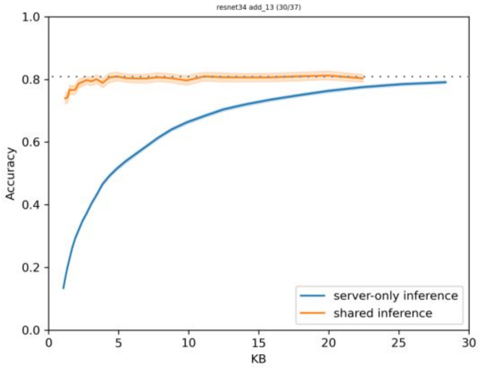
(a) ResNet-34, `pooling0` layer, $56 \times 56 \times 64$



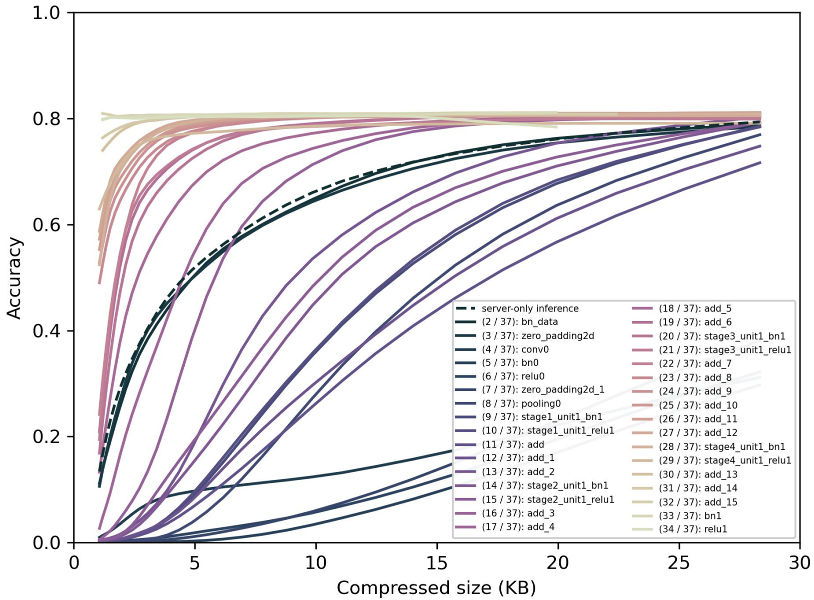
(b) ResNet-34, `add_3` layer, $28 \times 28 \times 128$



(c) ResNet-34, `add_7` layer, $14 \times 14 \times 256$



(d) ResNet-34, `add_13` layer, $7 \times 7 \times 512$



Shared curve seems to sustain maximum accuracy slightly longer than JPEG

Accuracy vs compressed size: JPEG 2000

Towards tensor stream compression

- Global translation of input by $\boldsymbol{x}$ pixels corresponds to $\boldsymbol{x}$ / 2^3 px translation in tensor

  16 px → 2 px
  32 px → 4 px
  48 px → 6 px

- Motion compensation w.r.t. reference tensor

$$\hat{T}(y,x,c) = T_{\text{ref}}(y + v_y(x,y),\ x + v_x(x,y),\ c)$$

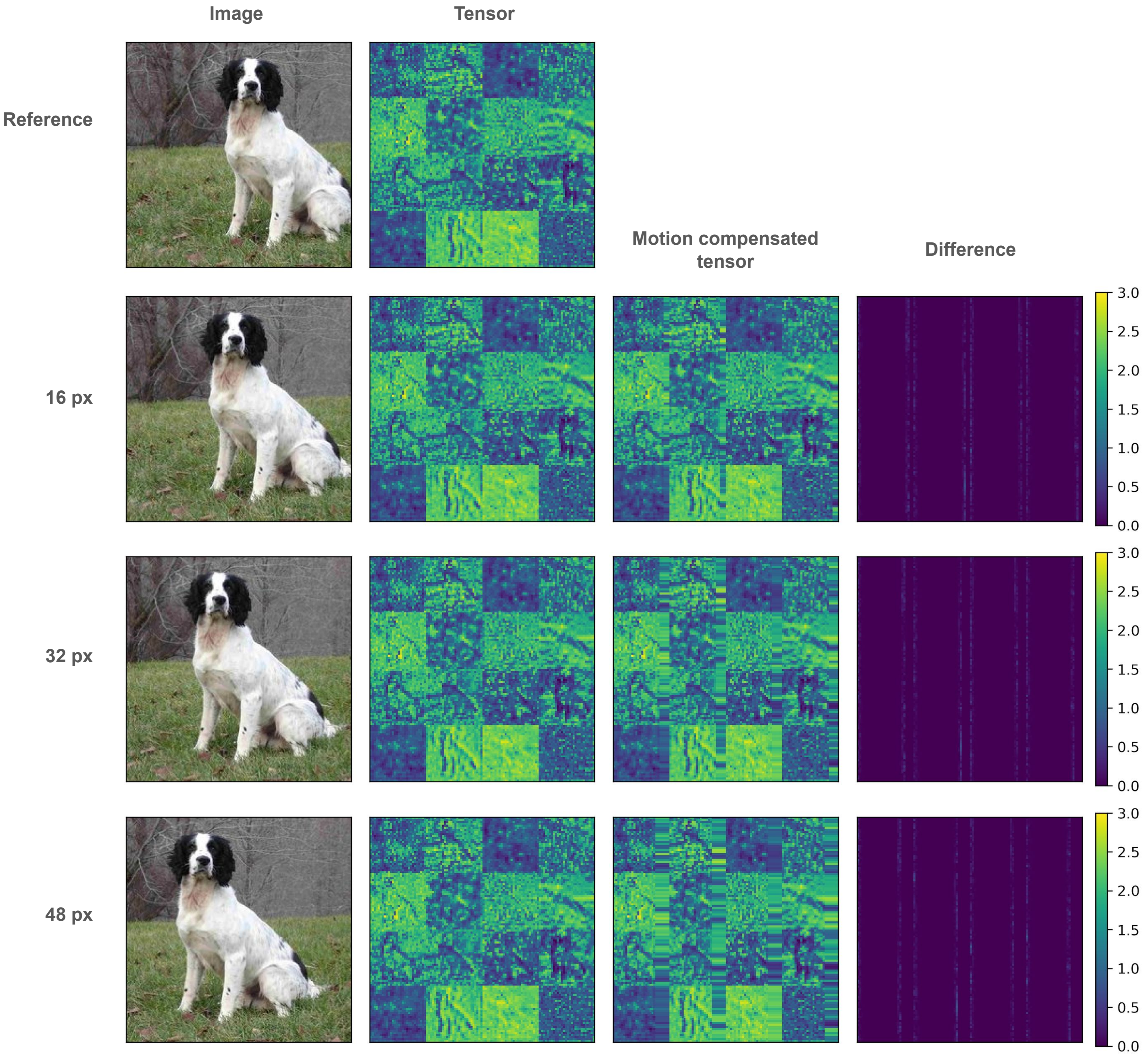
- PSNR: 90 dB

$$\text{MSE} = \|\hat{T} - T\|_2^2 = \frac{1}{HWC} \sum_{x=1}^{W} \sum_{y=1}^{H} \sum_{c=1}^{C} \left(\hat{T}(y,x,c) - T(y,x,c)\right)^2$$

$$R = \max T - \min T$$

$$\text{PSNR} = 10 \log \frac{R^2}{\text{MSE}}$$

Motion compensation



- Global translation of input by x pixels corresponds to x / 2^3 px translation in tensor

  18 px → 2.25 px
  34 px → 4.25 px
  50 px → 6.25 px

- Motion compensation w.r.t. reference tensor

$$\hat{T}(y,x,c) = T_{\text{ref}}(y+v_y(x,y),\ x+v_x(x,y),\ c)$$

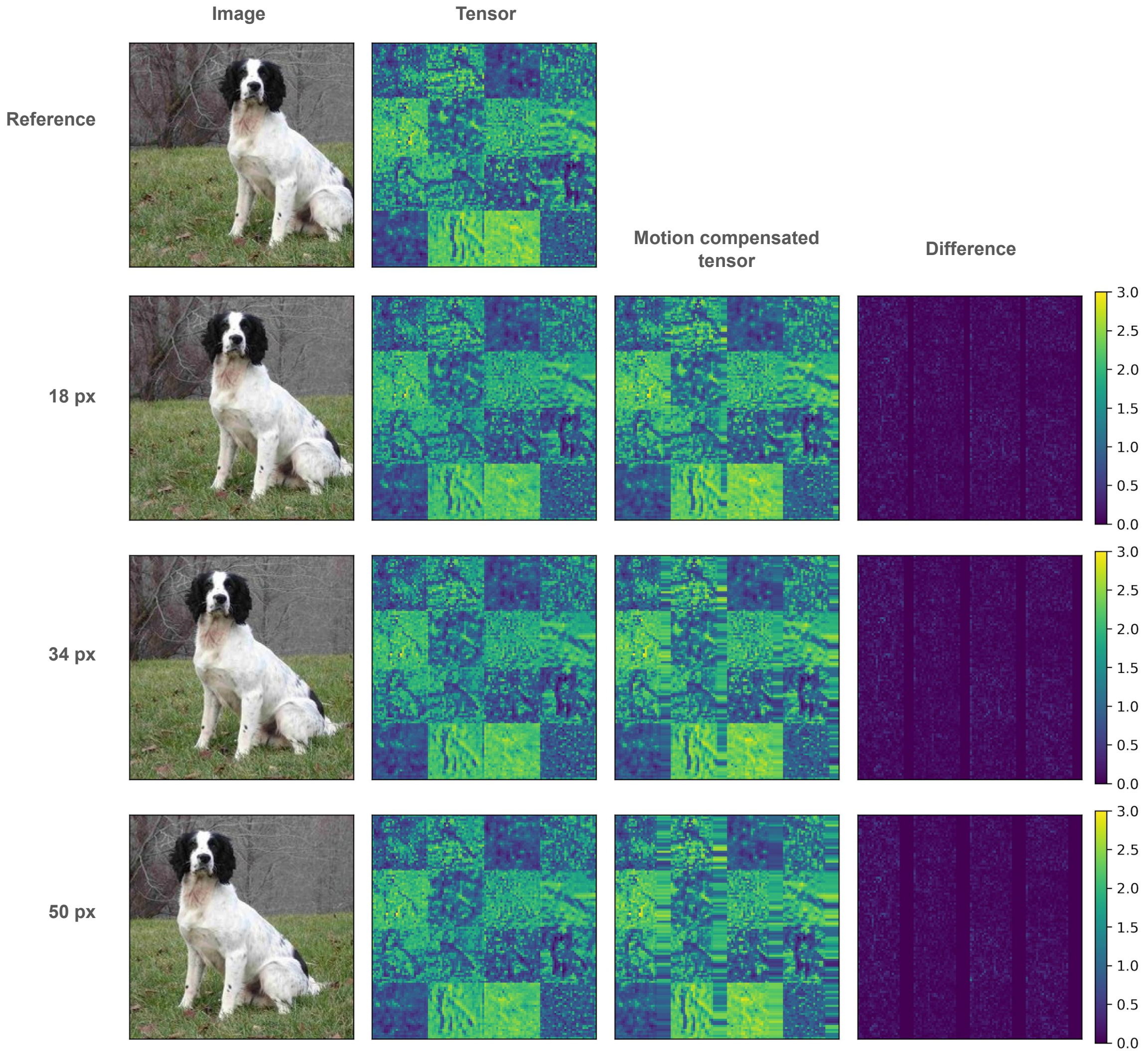
- PSNR: 75 dB

$$\text{MSE} = \|\hat{T} - T\|_2^2 = \frac{1}{HWC}\sum_{x=1}^{W}\sum_{y=1}^{H}\sum_{c=1}^{C}\left(\hat{T}(y,x,c) - T(y,x,c)\right)^2$$

$$R = \max T - \min T$$

$$\text{PSNR} = 10\log\frac{R^2}{\text{MSE}}$$

Motion compensation



Error concealment

Error concealment: experimental setup

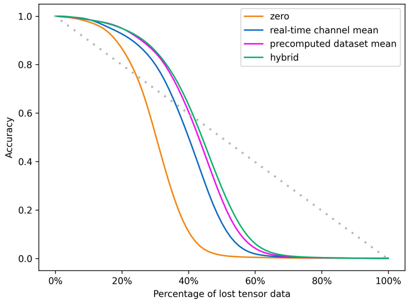
Goal: fill missing tensor entries in a way that minimizes drop in accuracy.

1. **Reuse** generated dataset from earlier.
2. **Randomly select** i.i.d. proportion of tensor entries.
3. **Fill** missing entries with best guess.
4. **Calculate** top-1 accuracy.
5. **Plot** average over many samples.

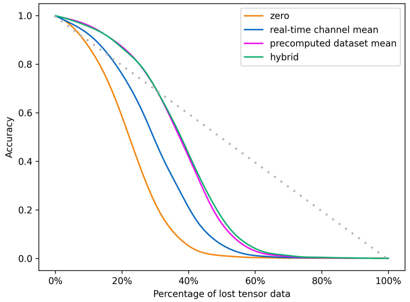
Results:

- 5% missing elements → 0.2% drop in accuracy
- 10% missing elements → 2% drop in accuracy
- 20% missing elements → 5% drop in accuracy

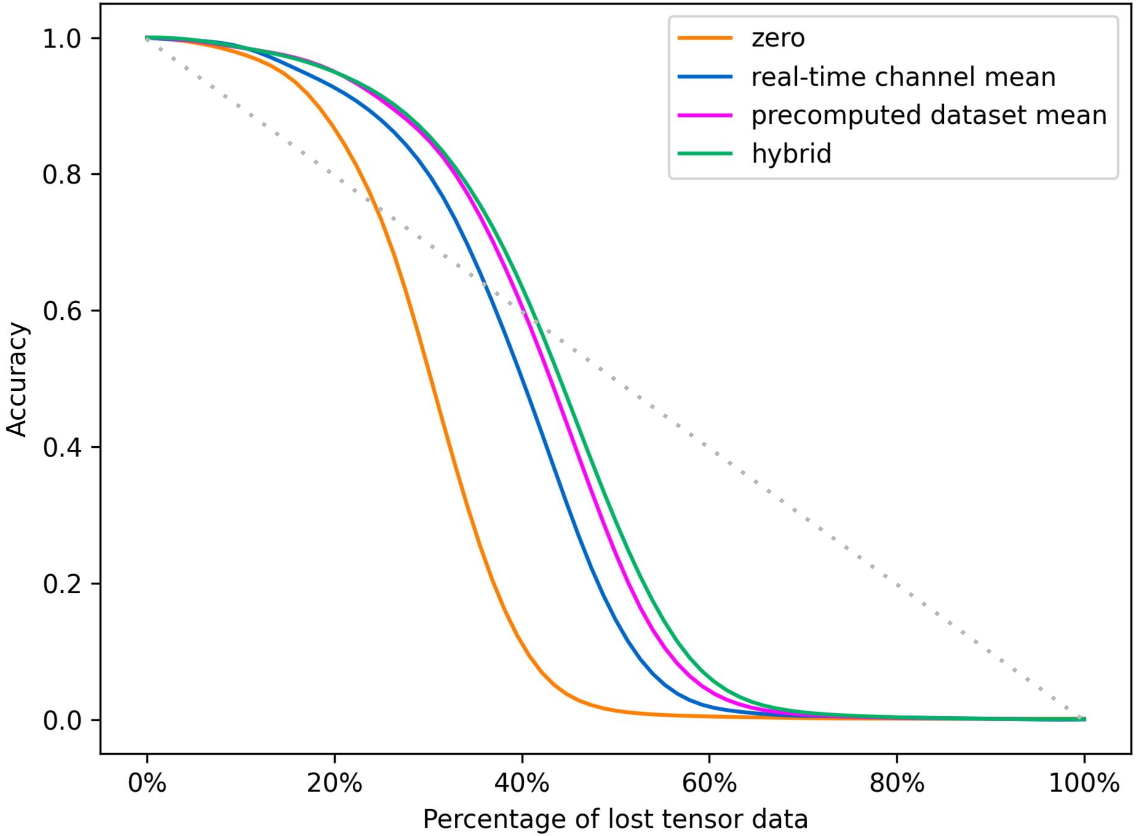
Comparison of error concealment methods

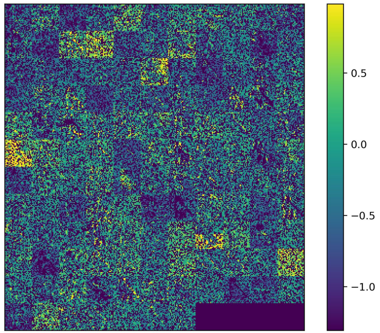


Randomly missing tensor elements



Randomly missing tensor channels





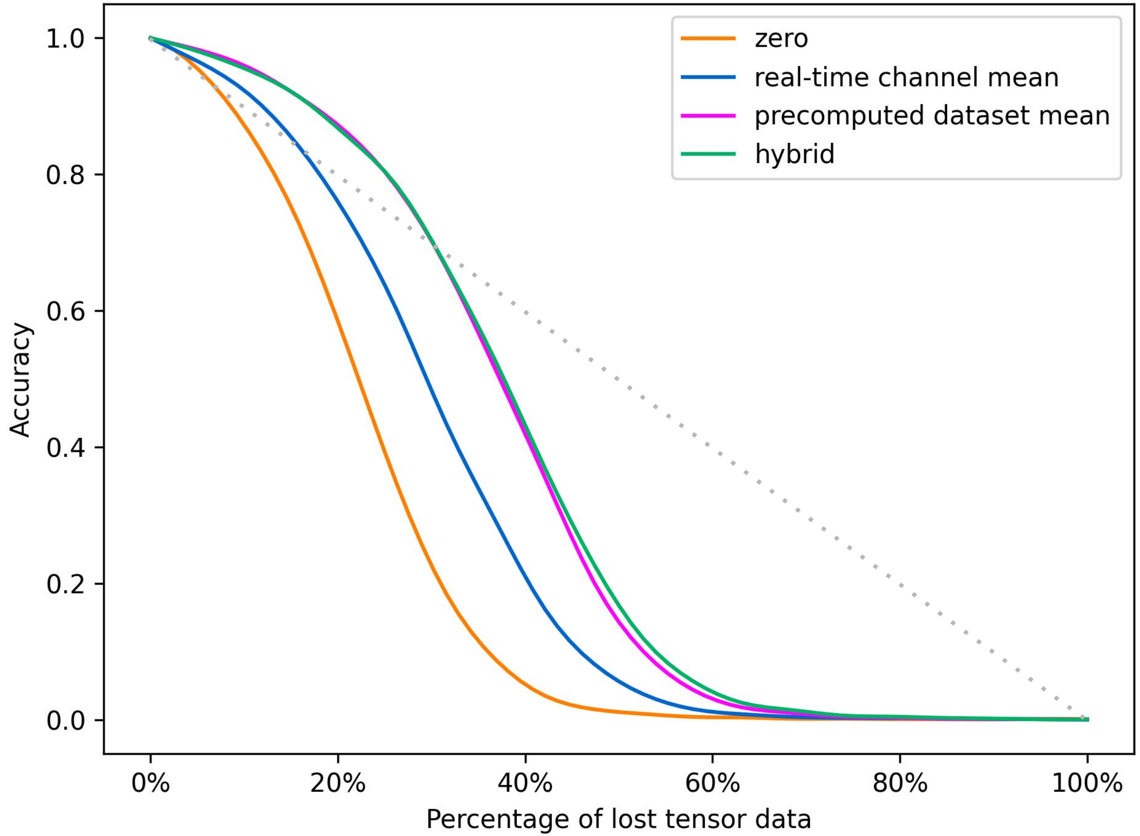
Tensor elements randomly “missing” from tensor.

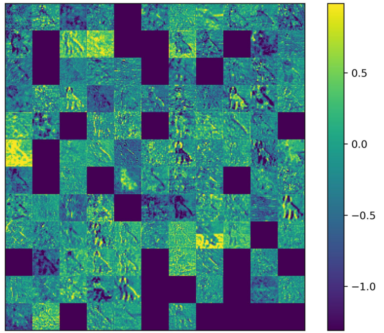
Recovery methods.
Set missing elements to:

1. zero
2. realtime mean of channel
3. mean element over large, representative dataset
4. “hybrid” of (2) and (3)

$$\hat{T}(y,x,c) = \mu(y,x,c) + \left(T_\mu(c) - \frac{1}{HW} \sum_{i,j} \mu(i,j,c) \right)$$

Error concealment of missing tensor elements





Tensor elements randomly "missing" from tensor.

Recovery methods.
Set missing elements to:

1. zero
2. realtime mean of channel
3. mean element over large, representative dataset
4. "hybrid" of (2) and (3)

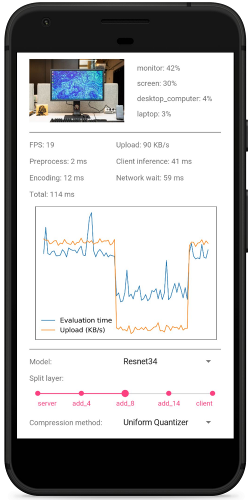
$$\hat{T}(y,x,c) = \mu(y,x,c) + \left(T_\mu(c) - \frac{1}{HW} \sum_{i,j} \mu(i,j,c) \right)$$

Error concealment of missing tensor channels

Future work

Future work: libraries

- **Ease of use** for non-experts (e.g. mobile app developers)
- **Extensibility** for researchers
- **Well-documented** API



Proposed features:

- **backpressure**-aware shared inference **pipelines** for real-time data processing
- combinators for methods used in **tensor compression** (e.g. quantization, tiling/weaving to convert between 3D and 2D tensor shapes, and interfacing with image/video/tensor codecs)
- real-time **tensor streaming protocol**
- real-time **monitoring** statistics for analysis
- real-time **inference strategy selection** based on network conditions

Future work

Improved model architectures

- Heavy computation reserved for server-side model
- High compressibility (lossily) of intermediate tensor

Compression

- Better tensor stream compression through reuse of motion vectors
- Better usage of redundancies in tensor (esp. for single tensor compression)

Networking

- Network protocol for low-latency, real-time compressed tensor streams

Thank you

References

[1] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge", *SIGARCH Comput. Archit. News*, vol. 45, no. 1, pp. 615–629, Apr. 2017, ISSN: 0163-5964. DOI: 10.1145/3093337.3037698. [Online]. Available: https://doi.org/10.1145/3093337.3037698.

[2] A. E. Eshratifar, M. S. Abrishami, and M. Pedram, "JointDNN: An efficient training and inference engine for intelligent mobile cloud computing services", *IEEE Transactions on Mobile Computing*, 2019, to appear.

[3] H. Choi and I. V. Bajić, "Deep feature compression for collaborative object detection", in *Proc. IEEE ICIP'18*, Oct. 2018, pp. 3743–3747. DOI: 10.1109/ICIP.2018.8451100.

[4] T. M. Cover and J. A. Thomas, "Elements of Information Theory", 2nd Edition. Wiley, 2006.

[5] J. Zhang, T. Liu, and D. Tao, "An information-theoretic view for deep learning", 2018. arXiv: 1804.09060 [stat.ML].

[6] L. Weng, "Anatomize deep learning with information theory", Sep. 28, 2017. [Online]. Available: https://lilianweng.github.io/lil-log/2017/09/28/anatomize-deep-learning-with-information-theory.html.

[7] N. Tishby and N. Zaslavsky, "Deep learning and the information bottleneck principle", 2015. arXiv: 1503.02406 [cs.LG].

[8] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition", in *Proc. ICLR*, 2015.

[9] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation", in *Proc. MICCAI*, 2015.

[10] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition", in *Proc. IEEE CVPR'16*, Jun. 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90.

[11] M. Ulhaq and I. V. Bajić, "Shared mobile-cloud inference for collaborative intelligence", NeurIPS'19 demo, 2020. arXiv: 2002.00157 [cs.AI].

[12] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift", 2015. arXiv: 1502.03167 [cs.LG].

[13] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry, "How does batch normalization help optimization?", 2018. arXiv: 1805.11604 [stat.ML].

[14] K. Sayood, "Introduction to Data Compression", 5th Edition. Morgan Kaufmann, 2018.

[15] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge", *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015. DOI: 10.1007/s11263-015-0816-y.

[16] Independent JPEG Group, "Libjpeg", version 9d, Jan. 12, 2020. [Online]. Available: http://libjpeg.sourceforge.net.

[17] ITU-T, "Information technology – digital compression and coding of continuous-tone still images – requirements and guidelines", International Telecommunication Union, Geneva, Recommendation T.81, Sep. 1992, p. 143.

[18] J. D. Kornblum, "Using jpeg quantization tables to identify imagery processed by software", *Digit. Investig.*, vol. 5, S21–S25, Sep. 2008, ISSN: 1742-2876. DOI: 10.1016/j.diin.2008.05.004. [Online]. Available: https://doi.org/10.1016/j.diin.2008.05.004.

[19] E. Kee, M. K. Johnson, and H. Farid, "Digital image authentication from jpeg headers", *IEEE Transactions on Information Forensics and Security*, vol. 6, no. 3, pp. 1066–1075, 2011.

[20] P. Yakubovskiy, "Image-classifiers", version 1.0.0, Oct. 4, 2019. [Online]. Available: https://github.com/qubvel/classification_models.

[21] D. Commander, "Libjpeg-turbo", version 2.0.4, Dec. 31, 2019. [Online]. Available: https://libjpeg-turbo.org.

[22] D. S. Taubman and M. W. Marcellin, "JPEG2000: Image Compression Fundamentals, Standards, and Practice". Kluwer Academic Publishers, 2002.

[23] U. d. L. Image and Signal Processing Group, "OpenJPEG", version 2.3.1, Apr. 2, 2019. [Online]. Available: https://www.openjpeg.org/.

[24] J. W. Woods, "Multidimensional Signal, Image, and Video Processing and Coding", 2nd Edition. Elsevier - Academic Press, 2012.

[25] L. Bragilevsky and I. V. Bajić, "Tensor completion methods for collaborative intelligence", *IEEE Access*, vol. 8, pp. 41 162–41 174, 2020.

[26] A. E. Eshratifar, A. Esmaili, and M. Pedram, "Towards collaborative intelligence friendly architectures for deep learning", 2019. arXiv: 1902.00147 [cs.DC].

[27] J. Nagle, "Congestion control in IP/TCP internetworks", RFC Editor, RFC 896, Jan. 1984. [Online]. Available: https://tools.ietf.org/html/rfc896.

[28] S. R. Alvar and I. V. Bajić, "Multi-task learning with compressible features for collaborative intelligence", in *Proc. IEEE ICIP'19*, 2019, pp. 1705–1709.

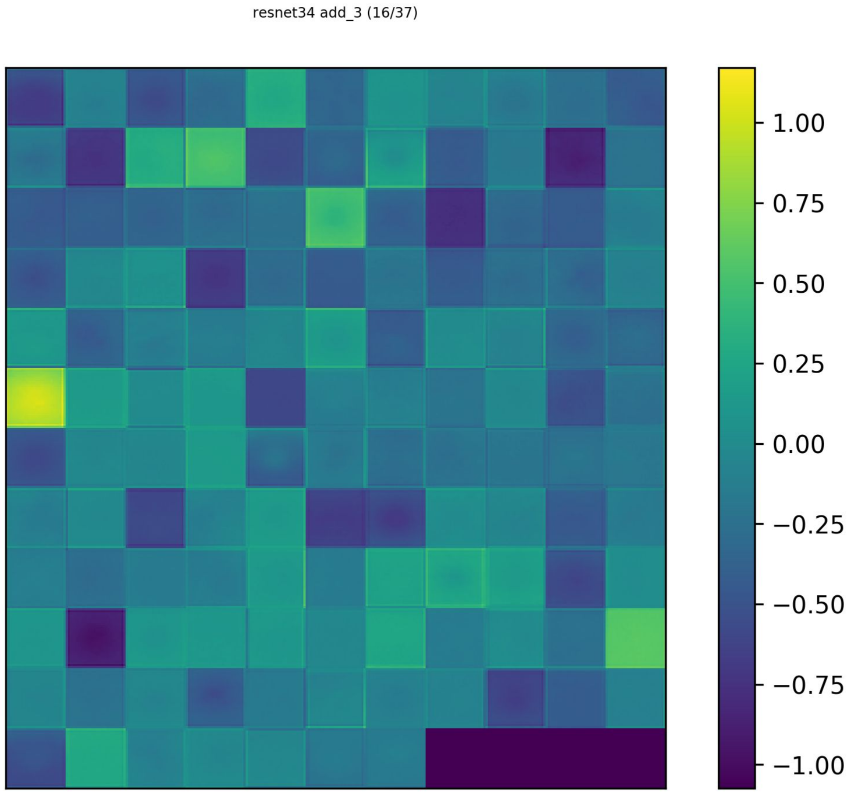
[29] H. Choi and I. V. Bajić, "Near-lossless deep feature compression for collaborative intelligence", in *Proc. IEEE MMSP'18*, Aug. 2018, pp. 1–6. DOI: 10.1109/MMSP.2018.8547134.

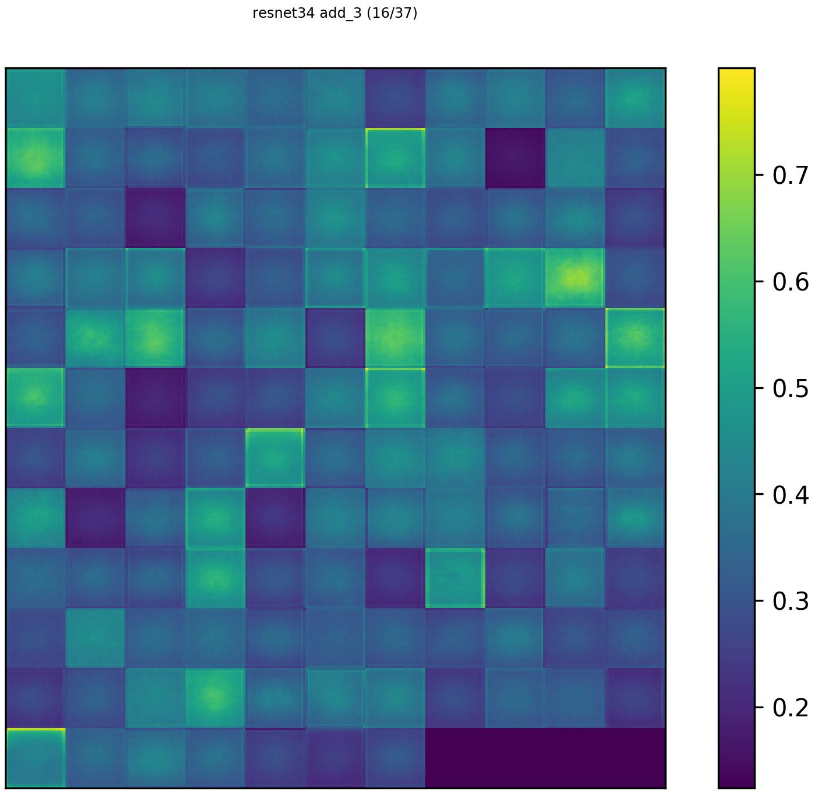
[30] H. Choi, R. A. Cohen, and I. V. Bajić, "Back-and-forth prediction for deep tensor compression", in *Proc. IEEE ICASSP'20)*, 2020, pp. 4467–4471.

[31] R. A. Cohen, H. Choi, and I. V. Bajić, "Lightweight compression of neural network feature tensors for collaborative intelligence", in *Proc. IEEE ICME'20*, to appear, 2020.

[32] Z. Chen, K. Fan, S. Wang, L. Duan, W. Lin, and A. C. Kot, "Toward intelligent sensing: Intermediate deep feature compression", *IEEE Transactions on Image Processing*, vol. 29, pp. 2230–2243, 2020.

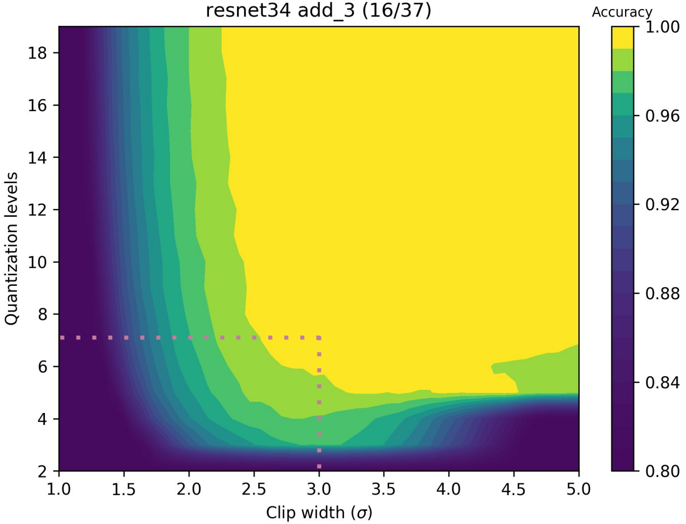
Bonus slides





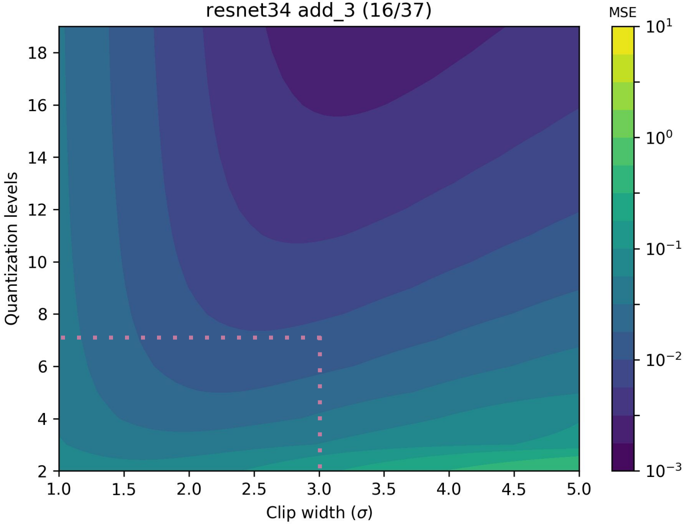
Mean and stddev (avg of 4096 samples)

Top-1 accuracy



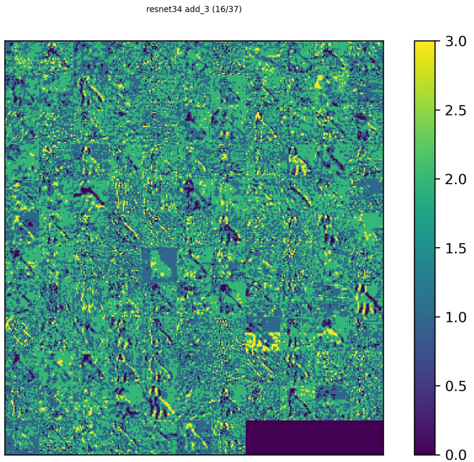
calculated over 16k samples from dataset

Reconstruction error (MSE)



calculated over 16k samples from dataset

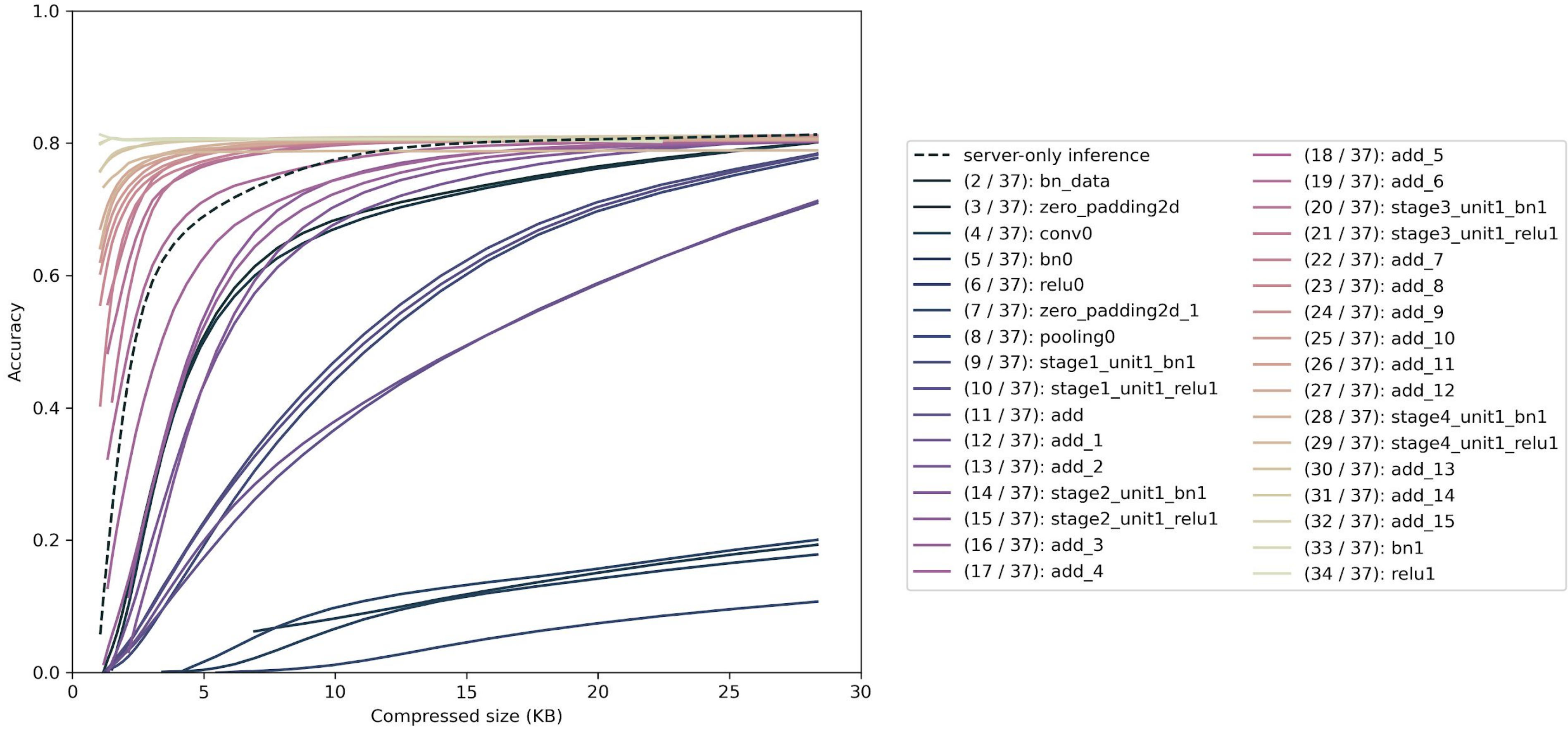
Quantized tensor



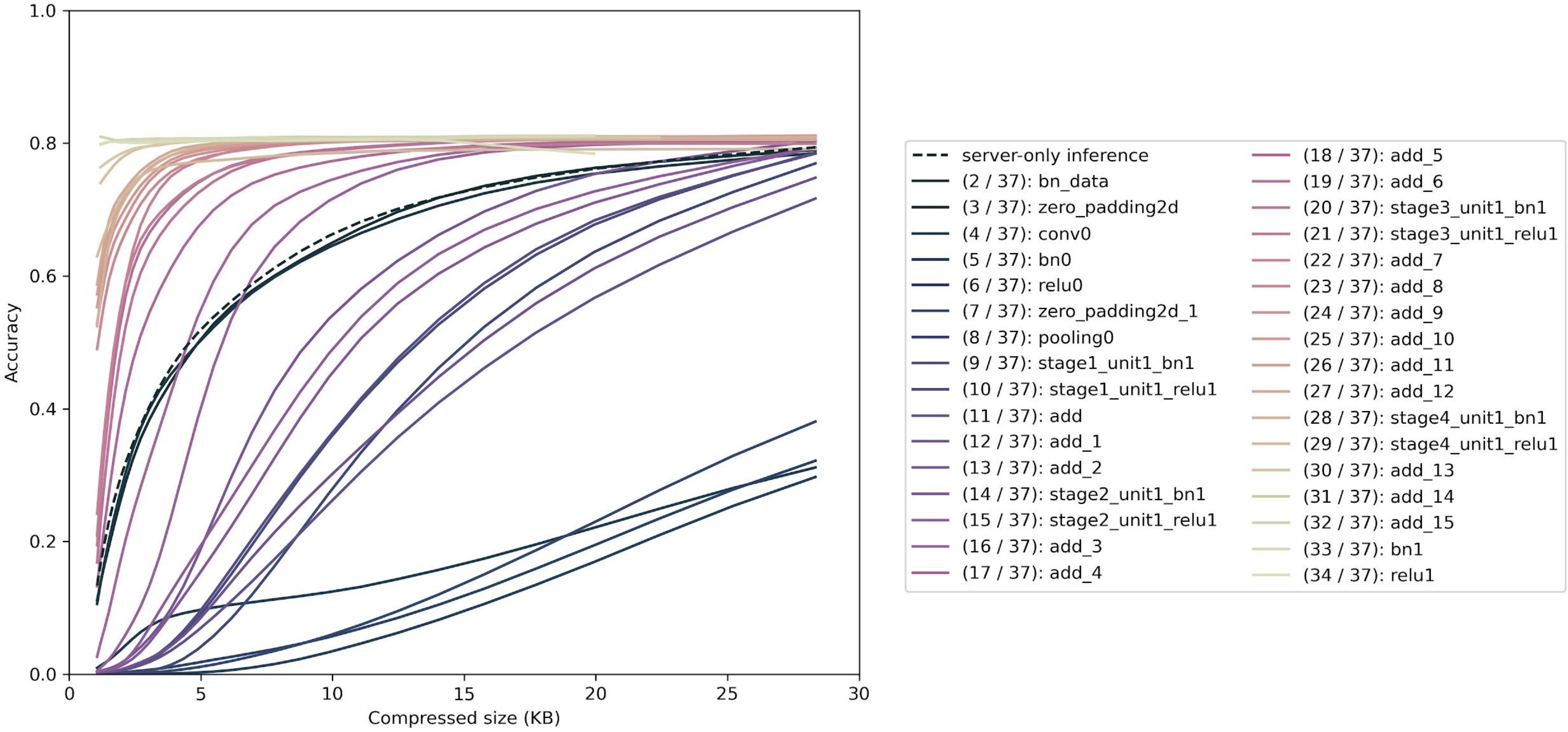
Uniform quantization, per-neuron distribution

Total inference time

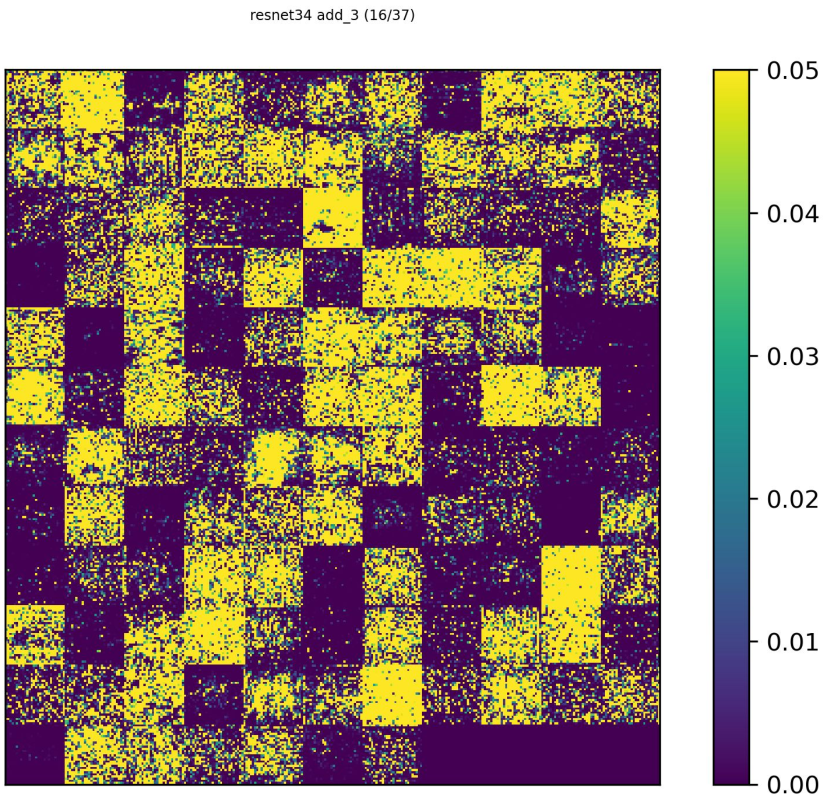
$$
\begin{aligned}
I_t^{(l)} &= I_c^{(l)} + E_c^{(l)} + E_s^{(l)} + I_s^{(l)} + \mathrm{RTT} + \frac{D^{(l)}}{B} &&= b^{(l)} + \frac{D^{(l)}}{B} && \text{split layer } l \\
I_t^{(s)} &= E_c^{(s)} + E_s^{(s)} + I_s^{(s)} + \mathrm{RTT} + \frac{D^{(s)}}{B} &&= b^{(s)} + \frac{D^{(s)}}{B} && \text{server-only} \\
I_t^{(c)} &= I_c^{(c)} &&= b^{(c)} && \text{client-only}
\end{aligned}
$$



Accuracy vs KB: JPEG



Accuracy vs KB: JPEG 2000



- Plotted: p-values of Shapiro-Wilk test for normality (frequentist)
- p-values clipped to [0, 0.05]
- 4096 input samples

Most neuron outputs are normally distributed (i.e. p >= 0.05) in response to input samples

Normality of output neuron values

Graveyard

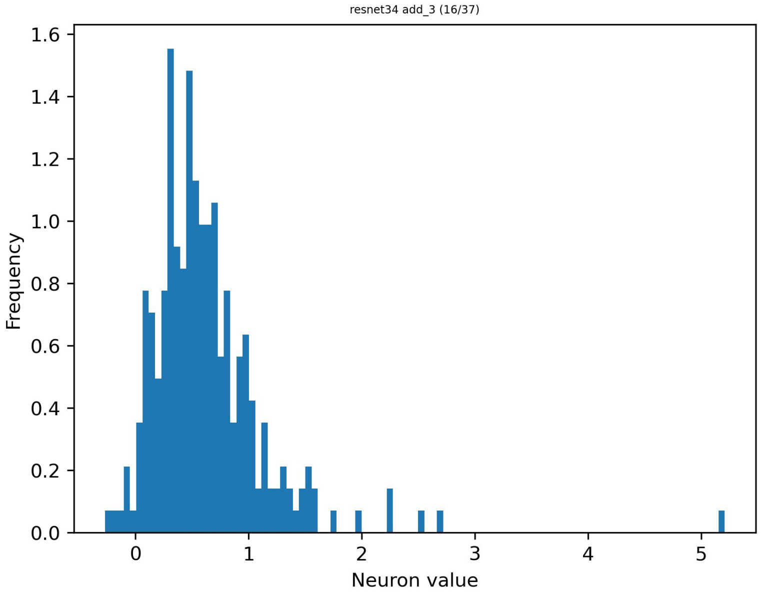
Script

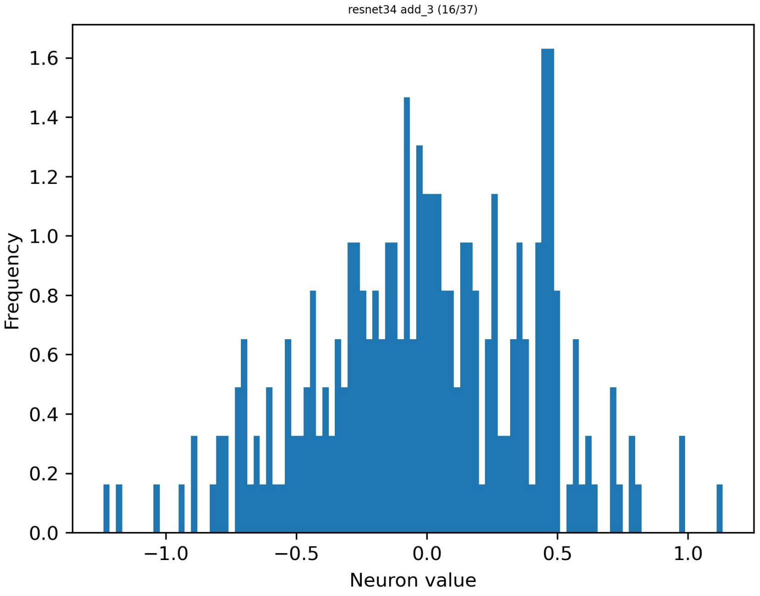
Target audiences: - Thesis committee - Lab members - "General" engineering audience (outside field of deep learning) - Future interested people looking at recording of defence > 1: Title - Hi everyone. - Thank you for joining me today. - I'll be presenting my thesis defence on the topic of "Shared ..." > 2: Outline - We'll start by looking at the background. (...???) - Then, we'll [look] at how to compress the data that is transmitted to the cloud. - Specifically, we will [talk about] the compression of single tensors, reusing image codecs for compression, developing techniques for compression of tensor streams. - Next, we will discuss some simple error concealment strategies for tensor data. - Finally, we will [discuss] possible extensions to this thesis and important work that remains to be done in the field of collaborative intelligence. > 3: Background - Let's start with the background. > 4: Inference strategies - Traditionally, there are two primary strategies that have been used to perform inference of deep learning models on mobile devices. - The first strategy is to perform the entire inference on the mobile device itself. This is shown in the diagram on the top. An image is inputted into the deep learning model running on the client, and the resulting label is then outputted by the model. - The second strategy is to perform the entire inference in the cloud. This is shown in the next diagram. The input image is transmitted over the network to the server, which then performs the inference. - These strategies both have their own drawbacks. - With the client-only inference strategy, the inference tends to run slower than server-only inference simply because servers tend to have better hardware than mobile devices. As a result, this puts a limit on the size and complexity of the model that can be run completely on the mobile client. - On the other hand, server-only inference requires transmission of the input image over the network, and is thus dependent on the quality of the network connection, upload bandwidth caps, and consumes energy due to radio transmission. Furthermore, it also introduces privacy concerns since the input image must be visible to the server in order to perform the inference. > 5: Shared inference - A third alternative to the previous two inference strategies is "shared inference". This concept has recently been introduced (in the Neurosurgeon paper) and is under active research. - This is visualized in the diagram on the bottom-right. With shared inference, the deep learning model is split partway through so that the first part of the inference is done on the client device. The information from this inference is then sent over the network to the server, which performs the second half of the inference. The result can then be transmitted back to the client. - The key idea behind this method is reducing the amount of data that is transmitted over the network. - In comparison to client-only inference, this strategy offers reduction in inference latency and the computational load put on the mobile device. - In comparison to server-only inference, this strategy offers reduction in inference times on slower network connections, saves bandwidth and mobile device energy, and can offer better privacy since the original signal never leaves the mobile. > 6: Layers of a deep learning model - On this slide we look at the layers of a deep learning model. - The figure on the left plots the total amount of time required to compute a given layer. The leftmost layer is the input layer and the rightmost is the output layer. The amount of time needed to compute and retrieve from GPU memory the contents of each layer is roughly increasing as we move deeper through the model. If we split early enough in the model, the larger portion of the remaining computation can be offloaded to the server. For instance, if we split on the layer pointed to by the arrow, the client only needs to perform approximately a quarter of the workload, and the remaining three quarters is given to the server. The question is then if we can find a good location to split early enough in the model. - The figure on the right plots the size of the data outputted at the given layer. As we move deeper through the layers, the size of the data volume increases and then decreases until it finally becomes a 4 KB probability vector. The size of the tensor data outputted at the arrow layer is for instance smaller than the size of the tensor outputted by the input layer. A good choice for a split point is when the data size is sufficiently lower than the input, such as the location pointed at by the arrow. However, this data is uncompressed so it does not show the whole story. In fact, the data that is outputted by these layers can often be lossily compressed significantly. As we will see, part of this occurs because the accuracy of inference is fairly resistant to errors in reconstruction, and this resilience often improves as we go deeper into the network. TODO fix script > 7: Layers of a deep learning model - ... > 8: Total inference time - This equation models the total inference time for shared inference as a sum of various terms. The first group of terms is the time spent on the client. The primary contributing factors are the inference time for the client-side model and the amount of time needed to compress the data. The next group represents the time taken to transfer the data across the network. The amount of time taken to upload the data is given by this term, which is the amount of data divided by the upload bandwidth of the connection. The other term represents the round trip time or ping. The terms in the last group represents the time taken to decompress the data and run server-side inference. All these terms summed together give the total inference time. > 9: Total inference time - If we vary the bandwidth and fix all the other terms to a constant, we can write the equation as a sum of some constant b and the data size divided by the upload bandwidth. We now quite reasonably assume then that the amount of data transmitted is zero for client-only inference, somewhat larger for shared inference, and even larger for server-only inference. Also, we assume that asymptotically, server-only inference is faster than shared inference, and shared inference is faster than client-only inference. Then, we can plot curves for client-only, shared, and server-only inference. - The client-only inference is constant w.r.t. bandwidth since no network transmission is needed. It has the lowest total inference time when upload bandwidth speed is zero. As the upload speed increases, shared inference becomes the strategy with lowest total inference time. As the upload speed further increases, server-only inference becomes the dominant strategy. So, there is some interval of upload bandwidth speed within which shared inference is the fastest. > 10: Experimental tests - This graph was generated from real-world experiments with uncompressed data. It exhibits the same sort of characteristics as were described in the previous slide. > 11: Prototype - A prototype was developed to demonstrate collaborative intelligence and help compare between client-only, server-only, and shared inference. An android device functioned as the mobile client, and it communicated with a remote server using TensorFlow for inference. This prototype was demoed at the NeurIPS 2019 conference. - The figure on the far right side shows the shared inference pipeline. - This shows various example latencies that occur at each block throughout the pipeline. - In order to conduct low-latency, high-throughput shared inference, more than one frame must be processed by the pipeline at a time. For instance, if the client is currently waiting for the server to reply, it may begin processing the next frame. - The pipeline must be properly synchronized to avoid backpressure due to blocks becoming overloaded with more frame requests than they can process. > 12: Single tensor compression - ... > 13: Client-side inference featuremap - After feeding the input image into the client-side model, we end up with an intermediate tensor. This is the featuremap for that tensor. - Compression techniques make use of redundancies within the data or the removal of unimportant information. - In this case, there are many spatial redundancies within each channel, similar to natural images. Nearby pixels are roughly equal to each other. - There is also some redundancy between channels; for instance, in many of the channels, the outline of a dog is visible. - One last thing to note the range of values, which is roughly between -1.5 and 1.2, which means most tensor elements lie within some small finite interval. > 14: Distributions of neuron output values - In order to apply entropy coding or image compression techniques, it is helpful to work with a small number of discrete values or symbols. The actual values of the tensor are floating point values, but we can introduce some error to them without losing too much inference accuracy. - These two plots show the distribution of possible values that the tensor elements can take. - On the left is the distribution of tensor values at the output of a BatchNorm layer. And on the right is the distribution of tensor values from a layer that follows some BatchNorm layer. Both seem to contain most of their values within some finite interval. This makes it easy to quantize and bin everything without introducing too much reconstruction error. - Furthermore, there is also a statistical distribution here that can be taken advantage of by entropy coding. > 15: Uniform quantization - This slide shows what happens when we uniformly quantize the approximately normal distribution from the previous slide. - For a quantization level of 7 and clipping interval width of approximately 3-sigma from the **mean of the distribution**, we get no drop in the top-1 accuracy, computed over 16000 samples from the dataset. - It is also interesting to note that the MSE in tensor reconstruction only roughly corresponds to the complete inference accuracy. It's important to measure the total inference accuracy, rather than just optimizing for minimal reconstruction error. TODO: bits/neuron > 16: Reusing image codecs - ... > 17: Process - The process for using an image codec is as follows: - First, the client-side model inference is run on the input and then the resulting 3D tensor is quantized, and then reshaped into a 2D tensor. This can then be fed into the image codec. - The compressed bitstream can be transferred to the server, which then reconstructs the tensor and performs the remaining inference on it. > 18: Accuracy vs KB: JPEG - Here, we plot the accuracy of the complete inference versus the size of the compressed data with JPEG compression applied. - The accuracy falls as more compression is applied. - The blue curve shows how the accuracy of server-only inference changes as compression is applied to the input image. - The orange curve represents the accuracy of shared inference with respect to the amount of compression applied to the intermediate tensor. - Some examples of compressed tensors of various compressed sizes are shown on the right. At 30 KB, the tensor is visually indistinguishable from the uncompressed tensor. But as more compression is applied, it becomes blockier and more distorted. At 3 KB, each channel within the featuremap is unrecognizable from its original. Despite this, such heavily compressed tensors still can produce reasonably accurate inferences. For instance, here, 3 KB tensors give an average drop in inference accuracy by 10%. - Furthermore, to allow no less than a drop in accuracy by 1%, one must not compress the data to less than 15 KB for server-only inference; whereas, for shared inference, one can compress to 10 KB to achieve the same accuracy threshold. TODO: show where in model this was cut!!! (small figures + arrow) > 19: Accuracy vs KB: experimental setup - This slide describes the experimental setup for the curves shown in the previous slide. - The dataset is generated from ImageNet, and consists of images cropped and resized to 224x224. - The experiment is done by running client-side inference on each image. Then, it is compressed to various sizes. These are then decompressed and server-side inference is run. The top-1 accuracy is then computed on the results. > 20: Accuracy vs KB: JPEG - This slide shows various plots for different layers of ResNet-34. - The figures on the left show a comparison of various layers with different dimensions of output tensors. - As we progress through the layers, the shared inference curves tend to improve. - However, this is not entirely due to reduced dimensionality. The figure on the right plots shared inference curves for a large number of layers. The darker, bottommost curves represent earlier layers; whereas, the later layers are given by the lighter curves above. Evidently, there is gradual improvement in inference accuracy, even as we move through layers with equal dimensionality. This shows that later layers are more resistant to reconstruction errors. > 21: Accuracy vs KB: JPEG 2000 - This slide shows the same layers as the previous slide, but using the JPEG 2000 codec instead. - The shared inference curves seem to sustain their maximum accuracy slightly longer than JPEG, though they also drop off more sharply. - One of the benefits of JPEG 2000 over JPEG is that the bitrate controls are more predictable, which is a desirable property for reliability. TODO: > 22: Towards tensor stream compression - ... > 23: Motion compensation - For many input sequences, there is often some significant amount of similarity between successive frames. For instance, the temporal redundancies between frames of a video give rise to motion estimation and motion compensation techniques. - On this slide, we demonstrate that many similar techniques should also be applicable to convolutional neural networks. - In this example, we translate the reference image by 16, 32, and 48 pixels. We claim that this results in a corresponding translation of the tensor by 2, 4, and 6 pixels. - The second column shows the ground truth tensors which are produced by running inference on the translated images. - The third column shows the difference between the ground truth and the motion compensated reconstructions of the tensor. - Many of the interior region of the channels are in fact a perfect reconstruction. This is likely because we are translating by what correspond to integer amounts within the tensor. This does not give the max pool operations in earlier layers an opportunity to introduce slight translational non-linearities. - The PSNR of this reconstruction is 90 dB, which is quite good. > 24: Motion compensation - On this slide, we do the same as the previous slide, but with translation amounts that should give non-integer translations in the tensor. - In this case, the interior regions are no longer an exact reconstruction and the PSNR has fallen to 75 dB. This is still quite good, however. > 25: Error concealment - ... > 26: Error concealment: experimental setup - Often in network transmission, some amount of the data may be lost or become corrupted. Error concealment seeks to reduce the negative effects of lost data by filling in lost tensor elements with values that minimize the drop in inference accuracy. Note that this is slightly different from attempting to minimize reconstruction error, though the objectives often align. - We reuse the dataset from earlier and run the client-side inference model to obtain a collection of tensors. Then, we randomly select some proportion of tensor elements and assume that they've been lost. We then fill them with a best guess, and then calculate the top-1 inference accuracy. - The figures on the right show that as more and more of the tensor goes missing, the more the accuracy drops. - The results show that if 5% of the tensor is missing, then using the concealment strategies tested, we only suffer a 0.2% drop in inference accuracy. Similarly, for 10% missing elements, a 2% drop; and for 20% missing, only a 5% drop. > 27: Error concealment of missing tensor elements - This figure is a close up of one of the figures from the previous slide. TODO: visualization of "black" tensor/channels > 28: Error concealment of missing tensor channels - ... > 29: Future work - ... > 30: Future work: libraries - ... > 31: Future work - ... TODO: describe what better architectures would look like (properties) TODO: strike out #3? Backup slides: TODO slides with information theory? (in case Jie asks question on size reduction) Ivan review: Cut out most of #10 and say we'll discuss later if questions on prototype For slide #17 note main points and don't spend too much time, then move to next slide where experimental details are

Changes

- Add citations small font in slide (e.g. neurosurgeon, jointdnn)
- Add actual video of demo
- ~~Mention work has been published; earlier in slides~~
- ~~Add notes (or text) for remembering what to say~~
- ~~Move some technical figures (“details”) to other hidden area/etc~~
- ~~Page numbers~~
- ~~Slide overview (to know what’s coming)~~

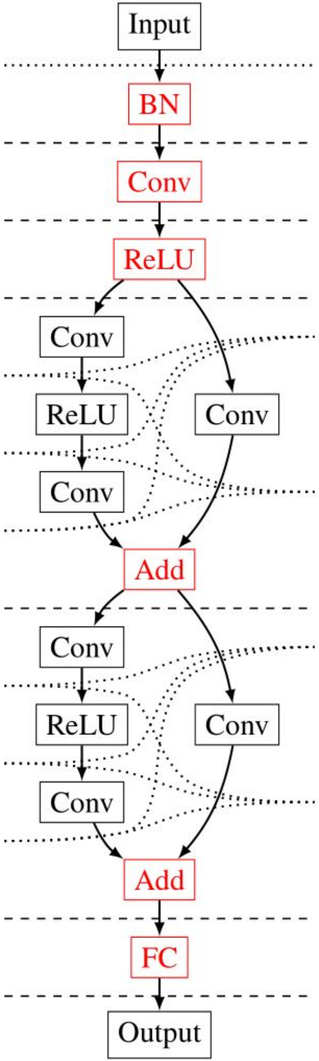
(TODOs from speaker notes too)



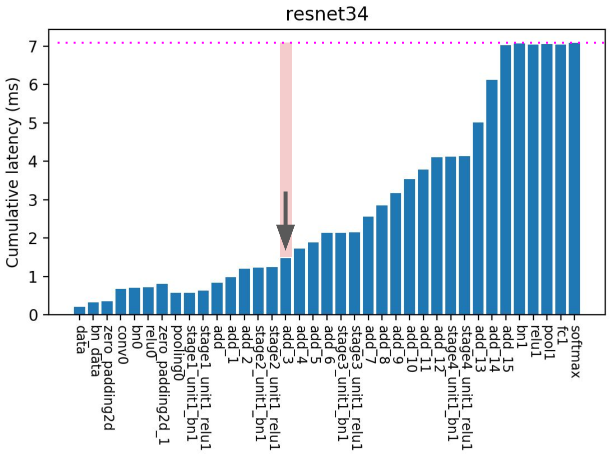


Individual neuron output distributions over dataset

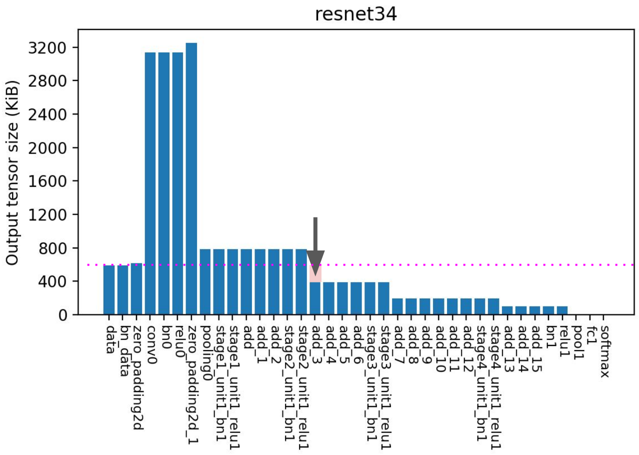
Example cut points



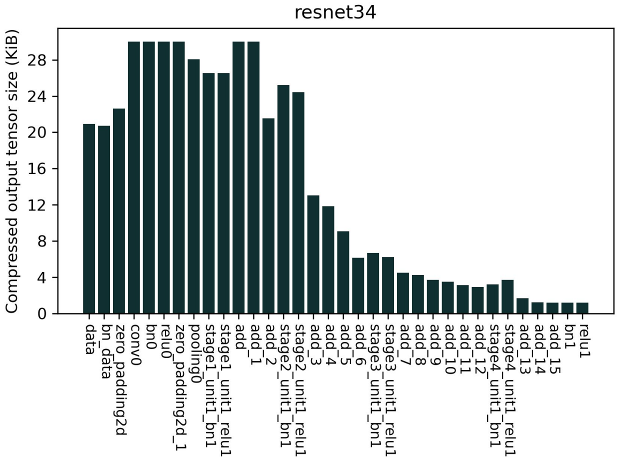
Layers of a deep learning model



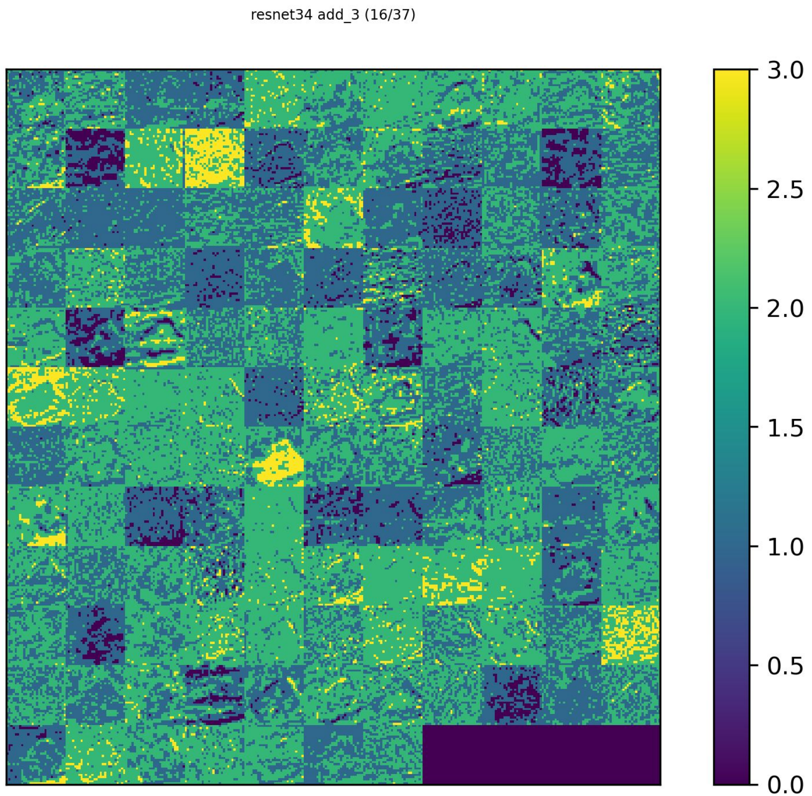
Cumulative inference time at layer

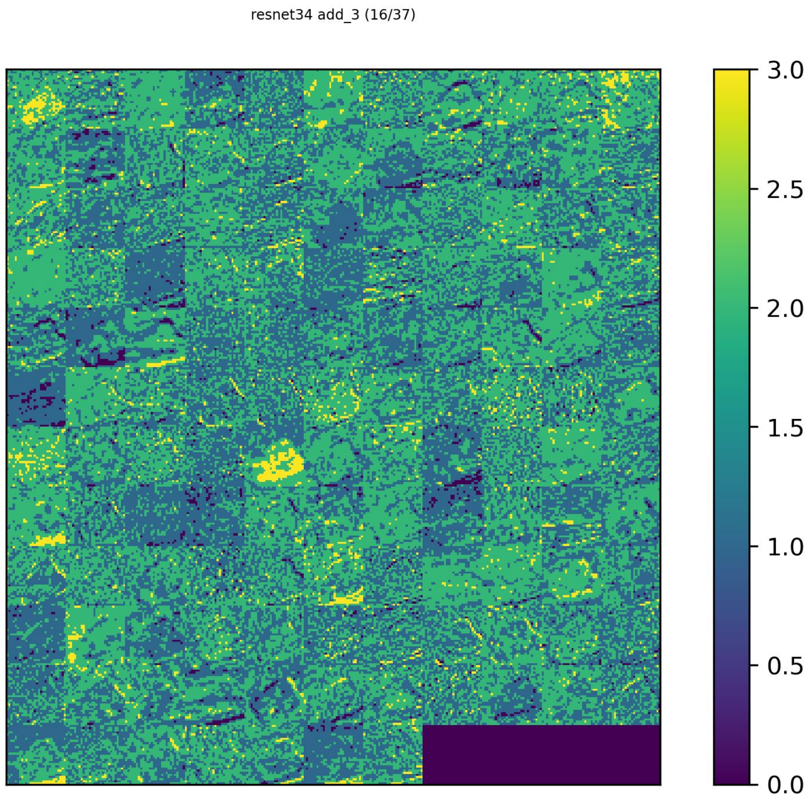


Size of data outputted by layer

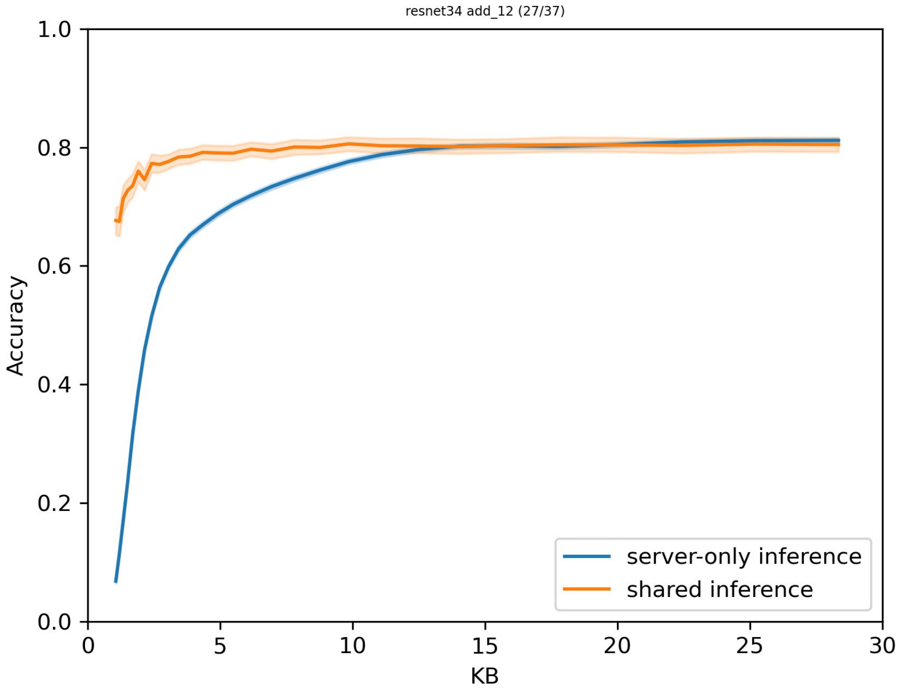


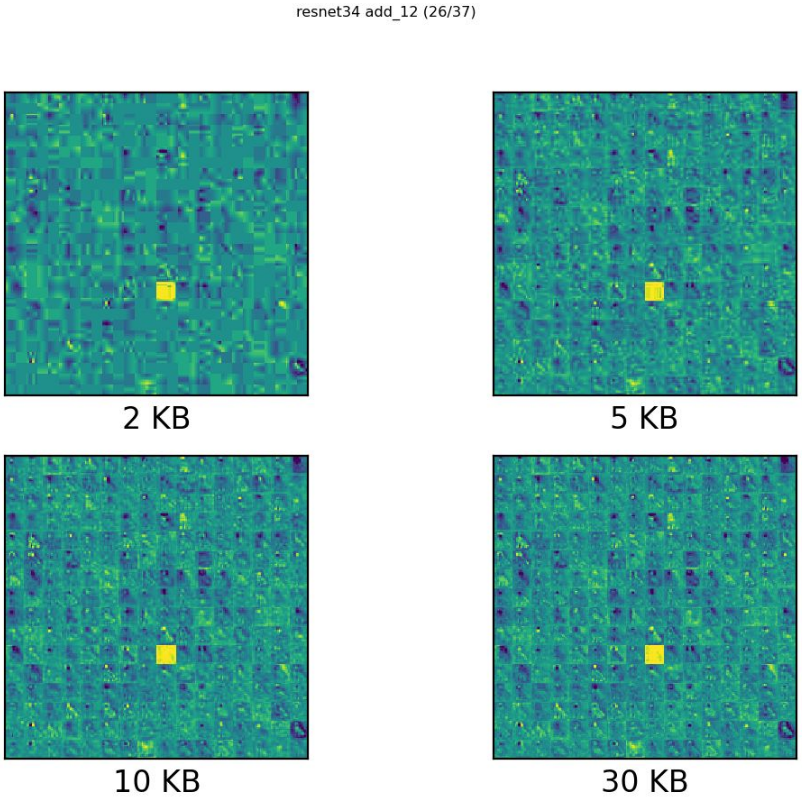
Compressed data size by layer



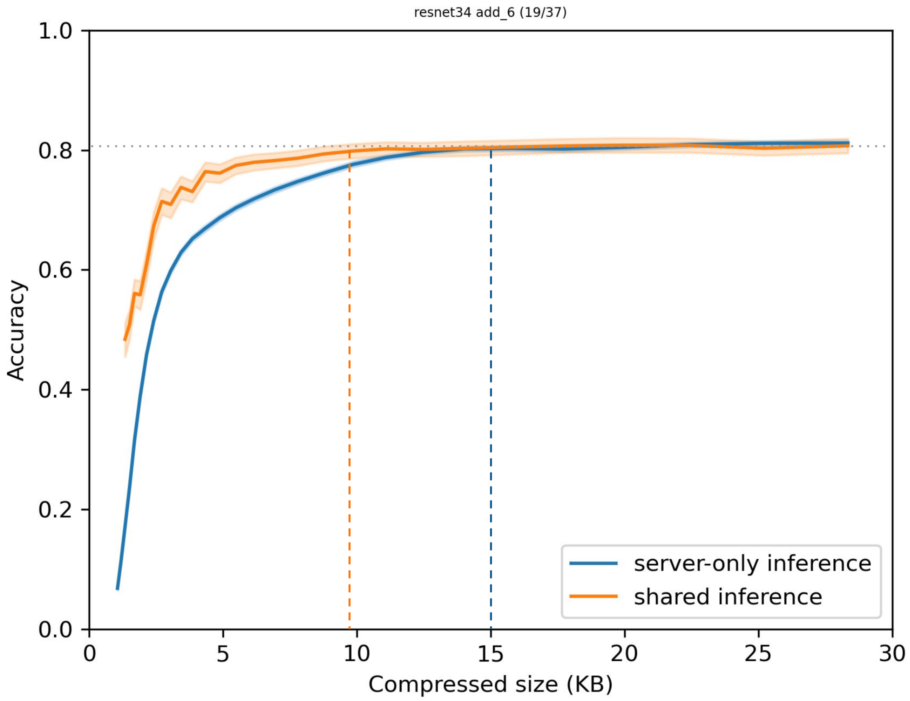


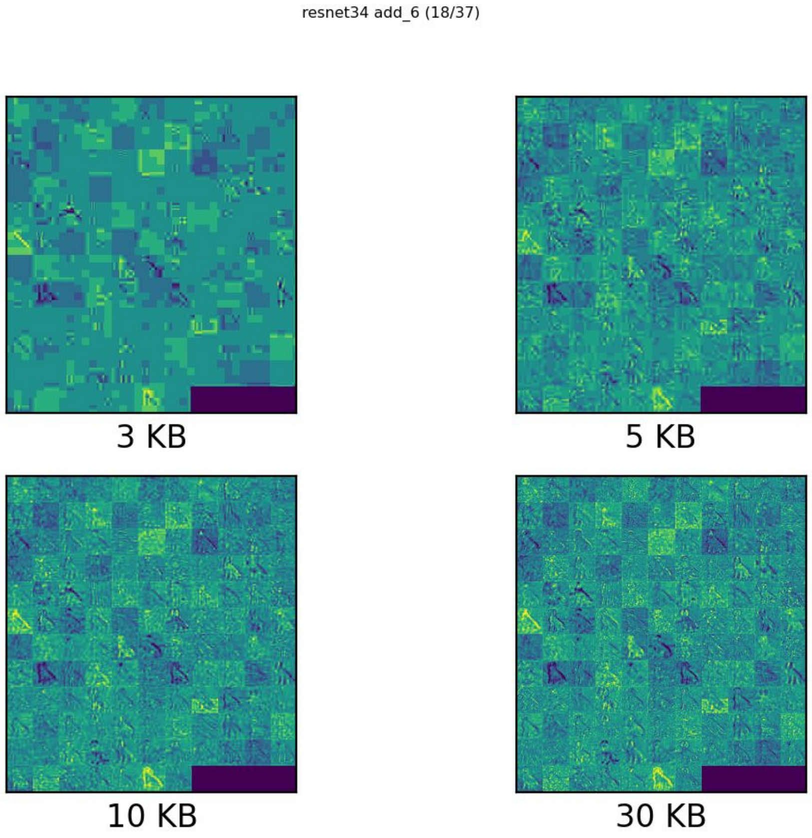
Featuremaps for clip_sigma=3, levels=4





Accuracy vs KB (JPEG)





Accuracy vs KB: JPEG

- Global translation of input by x pixels corresponds to x / 2^3 px translation in tensor

  16 px → 2 px
  32 px → 4 px
  48 px → 6 px

- Motion compensation w.r.t. reference tensor

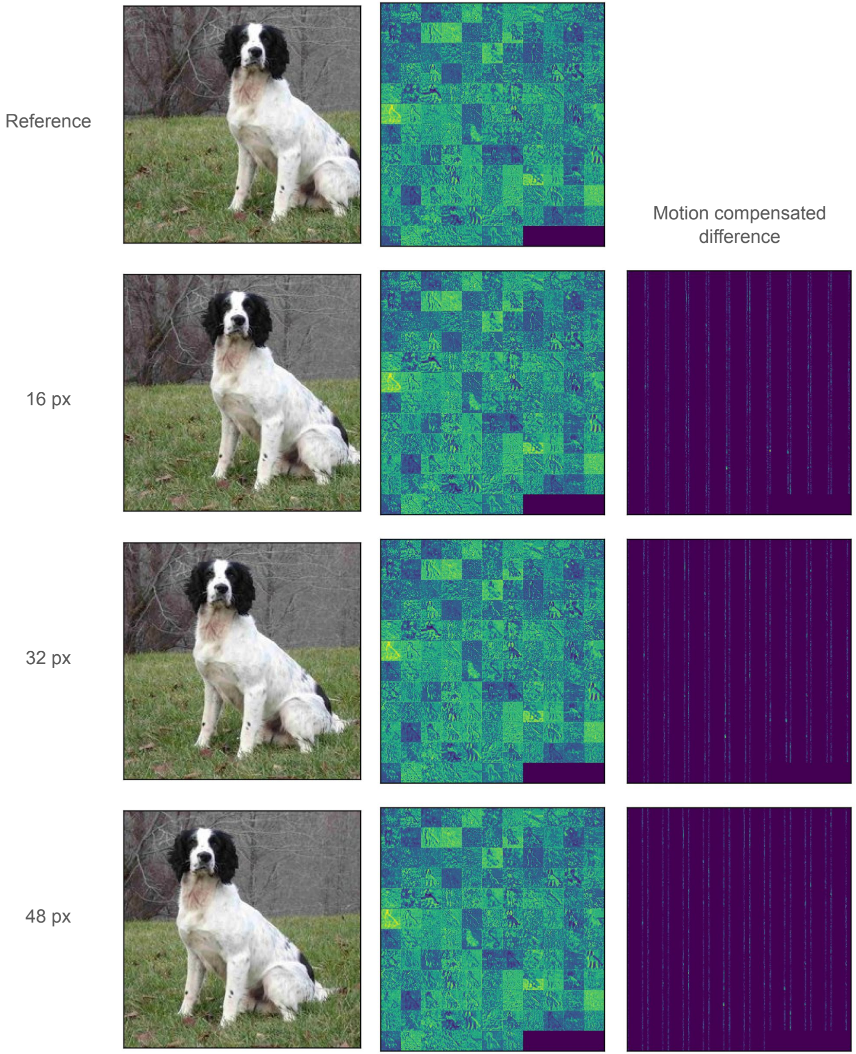
$$\hat{T}(y, x, c) = T_{\text{ref}}(y + v_y(x, y),\ x + v_x(x, y),\ c)$$

- PSNR of reconstruction: 90 dB

$$\text{MSE} = \|\hat{T} - T\|_2^2 = \frac{1}{HWC} \sum_{x=1}^{W} \sum_{y=1}^{H} \sum_{c=1}^{C} \left(\hat{T}(y, x, c) - T(y, x, c)\right)^2$$

$$R = \max T - \min T$$

$$\text{PSNR} = 10 \log \frac{R^2}{\text{MSE}}$$



Motion compensation

Motion compensation

- Global translation of input by x pixels corresponds to x / 2^3 px translation in tensor

  18 px → 2.25 px
  34 px → 4.25 px
  50 px → 6.25 px

- Motion compensation w.r.t. reference tensor

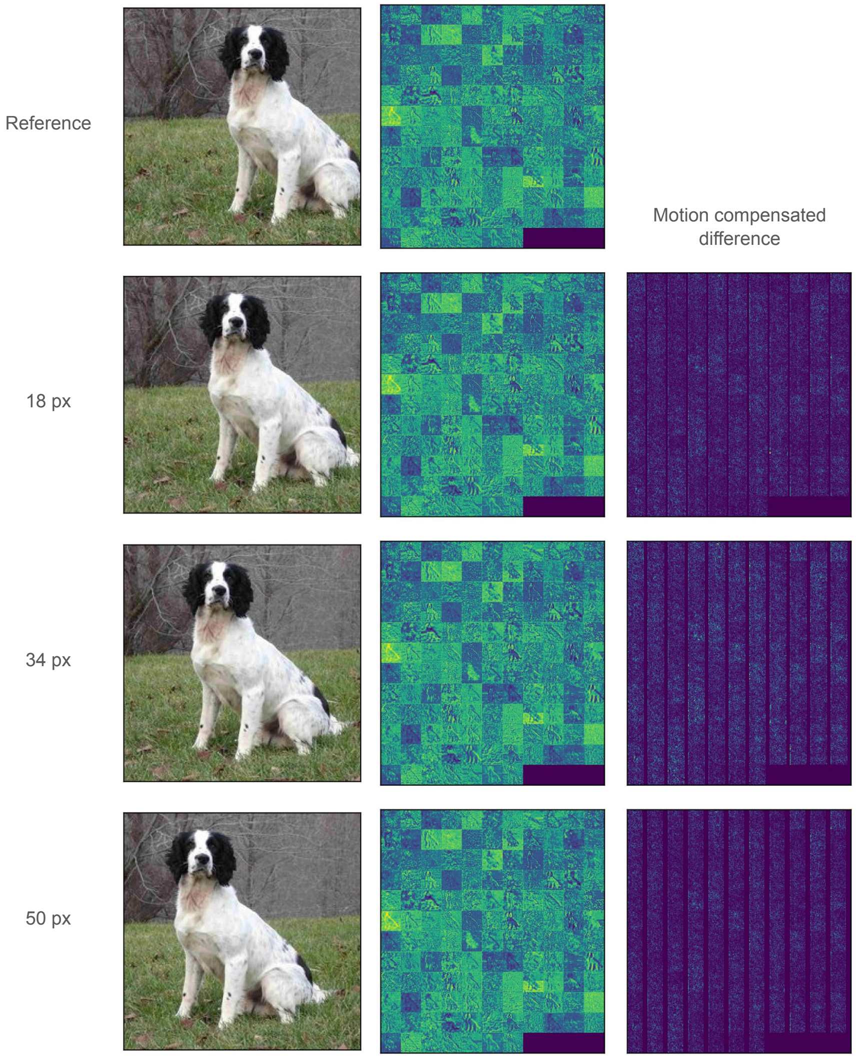
$$\hat{T}(y, x, c) = T_{\text{ref}}(y + v_y(x, y),\ x + v_x(x, y),\ c)$$

- PSNR of reconstruction: 75 dB

$$\text{MSE} = \|\hat{T} - T\|_2^2 = \frac{1}{HWC} \sum_{x=1}^{W} \sum_{y=1}^{H} \sum_{c=1}^{C} \left(\hat{T}(y, x, c) - T(y, x, c)\right)^2$$

$$R = \max T - \min T$$

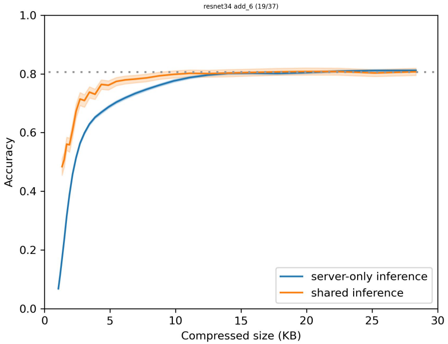
$$\text{PSNR} = 10 \log \frac{R^2}{\text{MSE}}$$

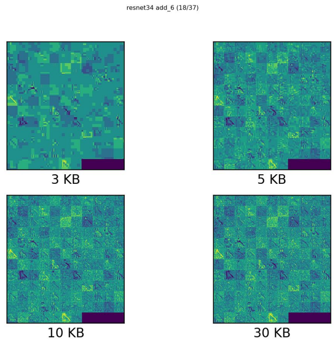


Accuracy vs KB: experimental setup

Dataset generation

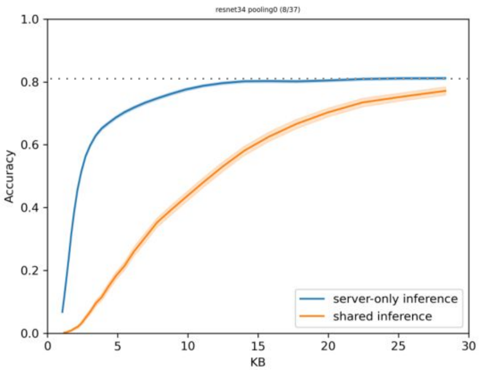
1. **ILSVRC 2012** (ImageNet, 1000 classes)
2. **Crop** to 1:1
3. **Downscale** to 224x224
4. **Save** as JPEG
5. **Keep** if file size is 30 ± 0.3 KB
6. **Keep** 16384 images



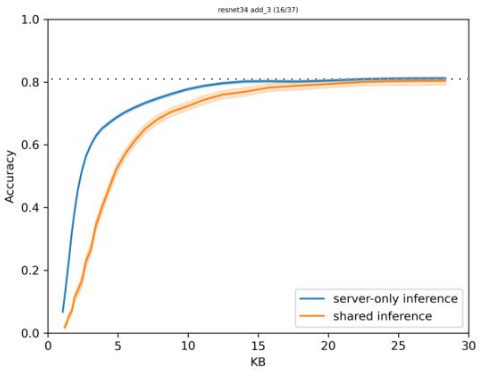


Experiment

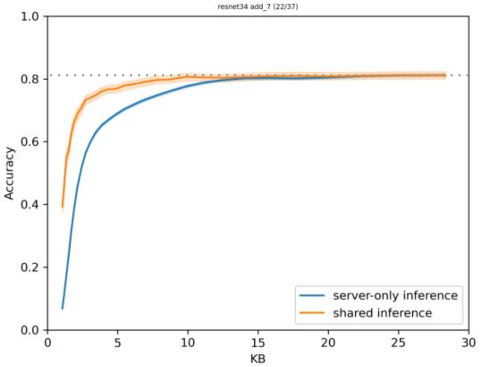
1. **Run client model inference** on each image
2. **Compress** each tensor via codec at various quality/bitrate settings, generating 100,000 different compressed tensors
3. **Bin by size** into logarithmically spaced bins
4. **Decompress** tensors
5. **Run server model inference**
6. **Compute top-1 accuracy** for each bin
7. **Plot** top-1 accuracy vs compressed size



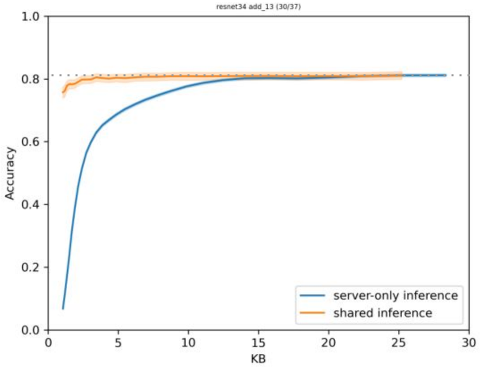
(a) ResNet-34, `pooling0` layer, $56 \times 56 \times 64$



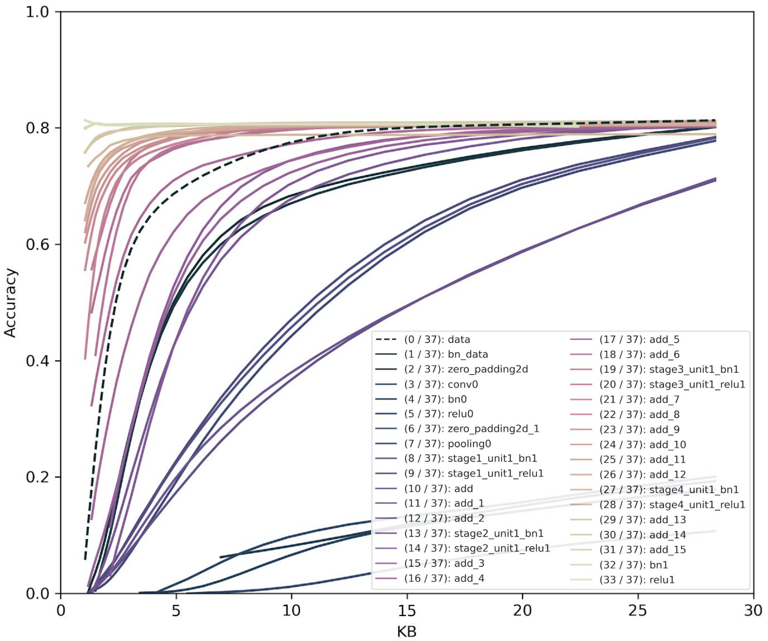
(b) ResNet-34, `add_3` layer, $28 \times 28 \times 128$



(c) ResNet-34, `add_7` layer, $14 \times 14 \times 256$

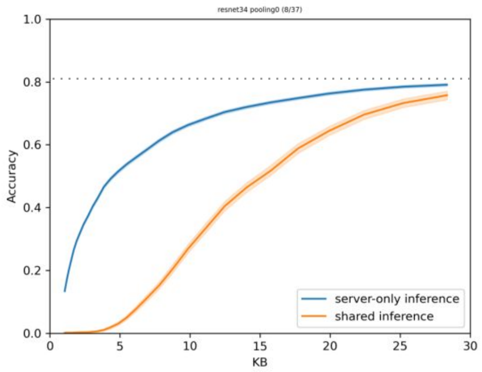


(d) ResNet-34, `add_13` layer, $7 \times 7 \times 512$

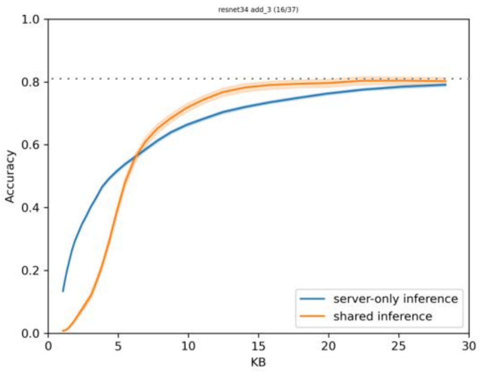


- Discrete Cosine Transform (DCT) on 16x16 macroblocks
- Shared inference accuracy curves generally improve as we go through deeper and deeper layers

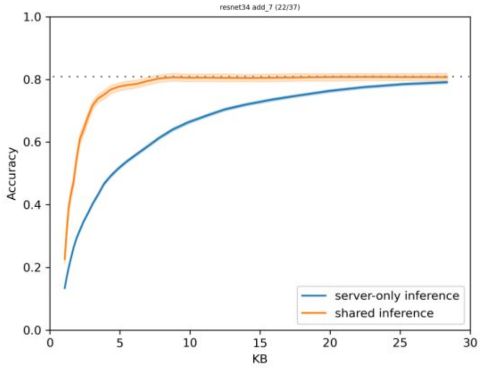
Accuracy vs KB: JPEG



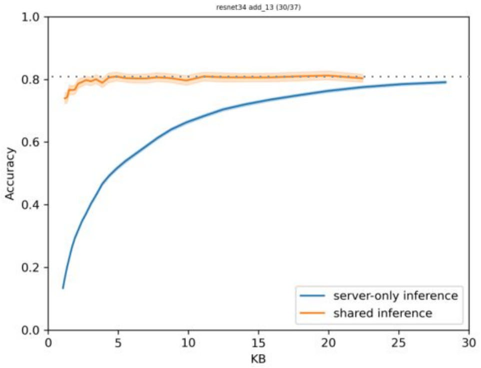
(a) ResNet-34, `pooling0` layer, $56 \times 56 \times 64$



(b) ResNet-34, `add_3` layer, $28 \times 28 \times 128$



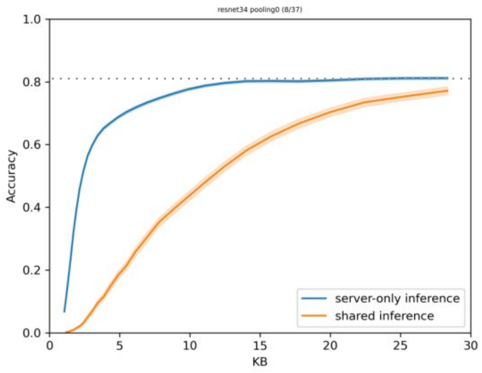
(c) ResNet-34, `add_7` layer, $14 \times 14 \times 256$



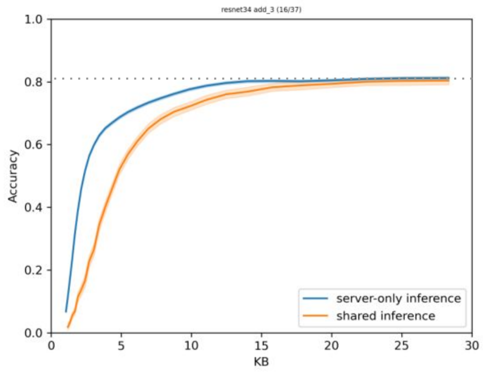
(d) ResNet-34, `add_13` layer, $7 \times 7 \times 512$

Accuracy vs KB: JPEG 2000

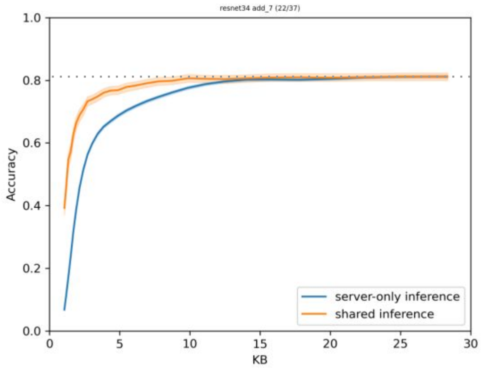
- Discrete Wavelet Transform (DWT)
- Shared curve seems to sustain maximum accuracy slightly longer than JPEG
- Server-only curve worse than JPEG (possibly because re-encoding JPEG to JPEG of a different quality level merely corresponds to a rescaling of quantization table)



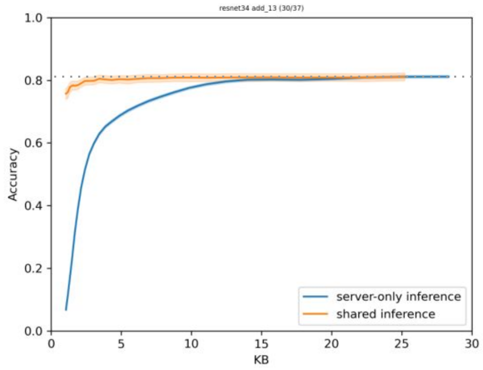
(a) ResNet-34, `pooling0` layer, $56 \times 56 \times 64$



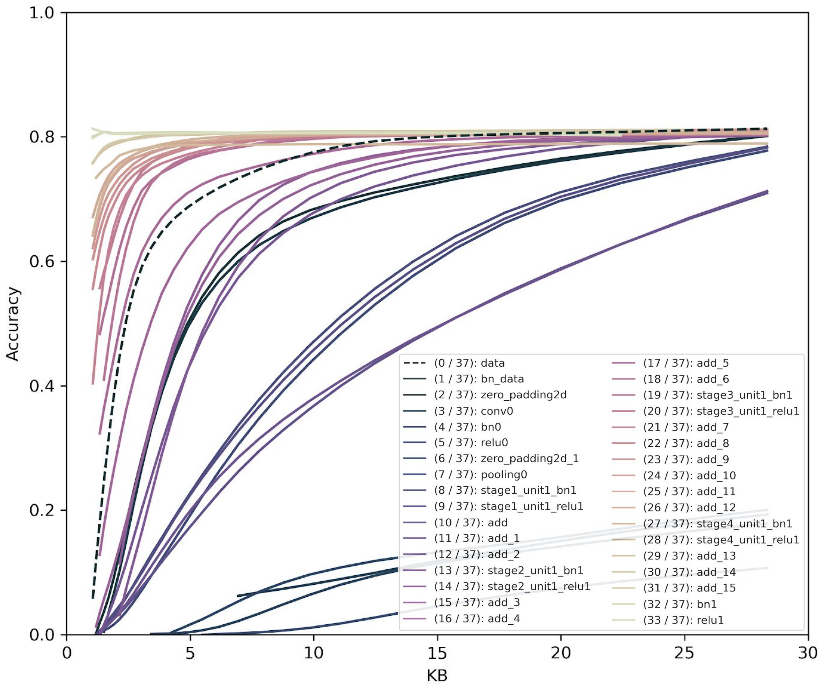
(b) ResNet-34, `add_3` layer, $28 \times 28 \times 128$



(c) ResNet-34, `add_7` layer, $14 \times 14 \times 256$

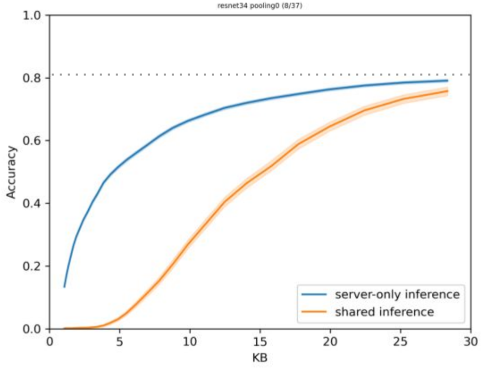


(d) ResNet-34, `add_13` layer, $7 \times 7 \times 512$

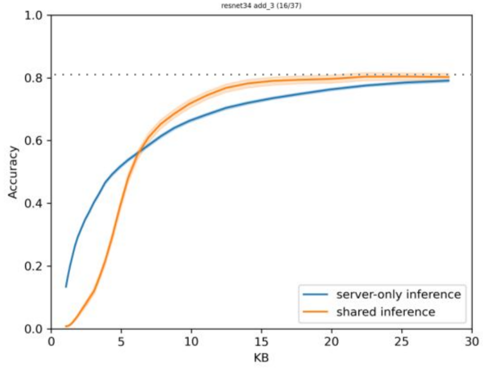


Shared inference accuracy curves generally improve as we go through deeper and deeper layers

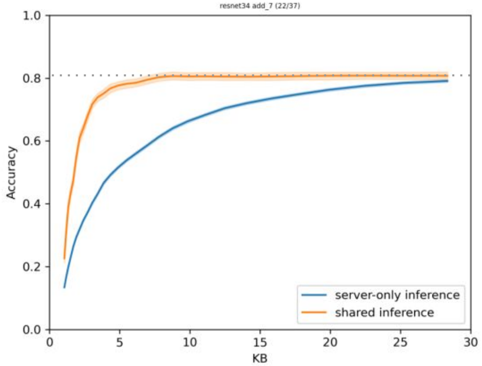
Accuracy vs KB: JPEG



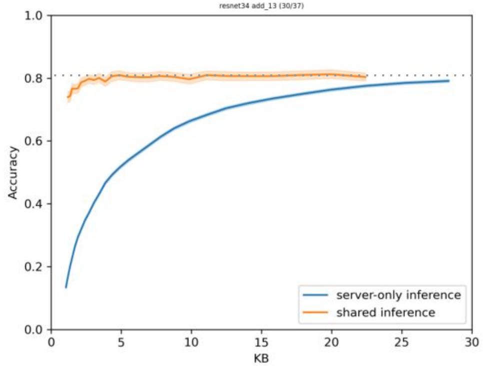
(a) ResNet-34, `pooling0` layer, $56 \times 56 \times 64$



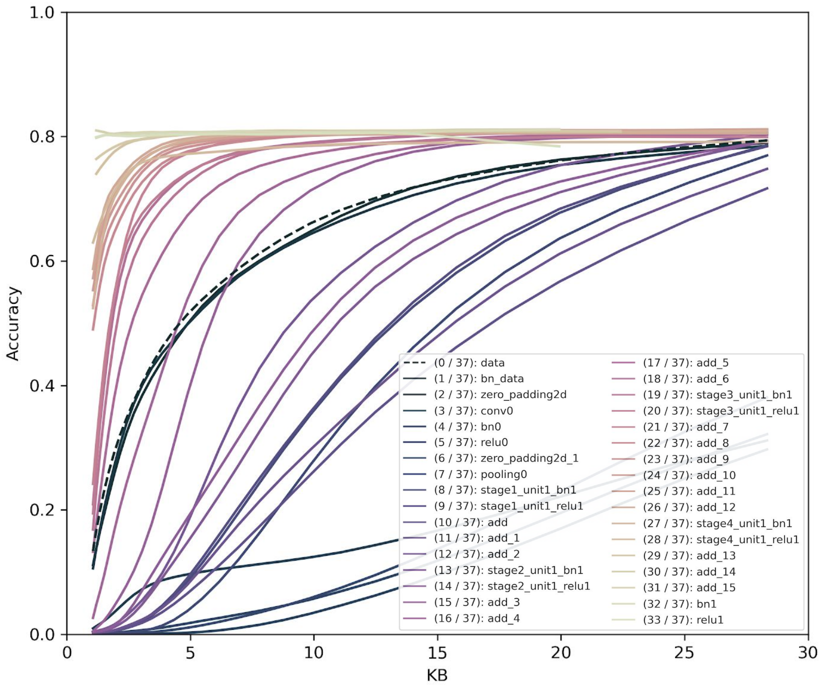
(b) ResNet-34, `add_3` layer, $28 \times 28 \times 128$



(c) ResNet-34, `add_7` layer, $14 \times 14 \times 256$

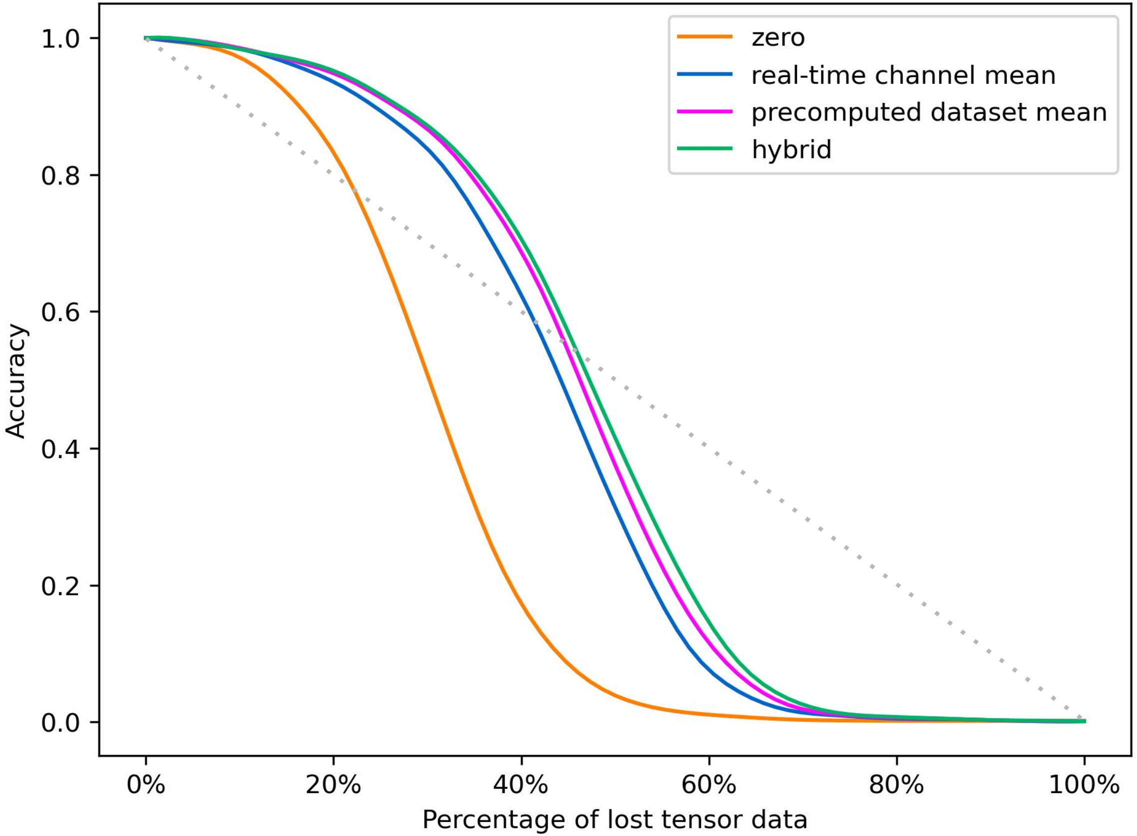


(d) ResNet-34, `add_13` layer, $7 \times 7 \times 512$



Shared curve seems to sustain maximum accuracy slightly longer than JPEG

Accuracy vs KB: JPEG 2000



Tensor channels randomly "missing" from tensor.

Recovery methods.
Set missing elements to:

1. zero
2. realtime mean of channel
3. precomputed mean element over large, representative dataset
4. "hybrid" of (2) and (3)

$$\hat{T}(y,x,c) = \mu(y,x,c) + \left(T_\mu(c) - \frac{1}{HW}\sum_{i,j}\mu(i,j,c)\right)$$

Error concealment of missing tensor channels (WRONG)

```json
{
  "frameNumber": "<int>",
  "inferenceTime": "<int>",
  "predictions": {
    "label": {"name": "<str>", "description": "<str>", "score": "<int>"},
    "label": {"name": "<str>", "description": "<str>", "score": "<int>"},
    ...
  }
}
```

```python
shape = model_client.output_shape[1:]
dtype = model_client.dtype
tensor_layout = ci.TensorLayout.from_shape(shape, "hwc", dtype)
postencoder = ci.JpegPostencoder(tensor_layout, quality=20)
tiled_layout = postencoder.tiled_layout
predecoder = ci.JpegPredecoder(tiled_layout, tensor_layout)
```

```python
def process_send_buffer():
  if send_buffer.is_empty():
    return

  if send_buffer.size() >= mss:
    data = send_buffer.pop_bytes(mss)
    send(Packet(data))
    return

  if send_buffer.contains_end_of_tensor():
    data = send_buffer.pop_until_end_of_tensor()
    send(Packet(data))
```

```python
def send_request_to_server(request):
  while True:
    # Wait until server rate limit is satisfied.
    if now() < last_request_time + server_rate_limit:
      Thread.sleep(0)
      continue

    # Wait until we expect not to exceed bandwidth limits.
    if estimate_unreceived_bytes() > 0:
      Thread.sleep(0)
      continue

    write(request.data)
    break
```

```python
def should_process_frame(frame):
  client_remain = estimate_client_process_latency()
  server_remain = last_request_time + server_rate_limit - now()
  bandwidth_remain = estimate_unreceived_bytes() / estimate_bandwidth()

  # Drop frame if server or bandwidth won't be available in time
  if (client_remain < server_remain or client_remain < bandwidth_remain):
    return False

  return True
```