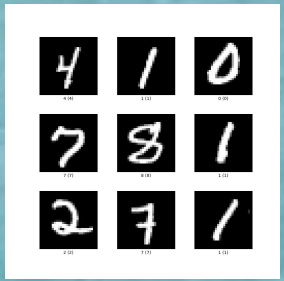
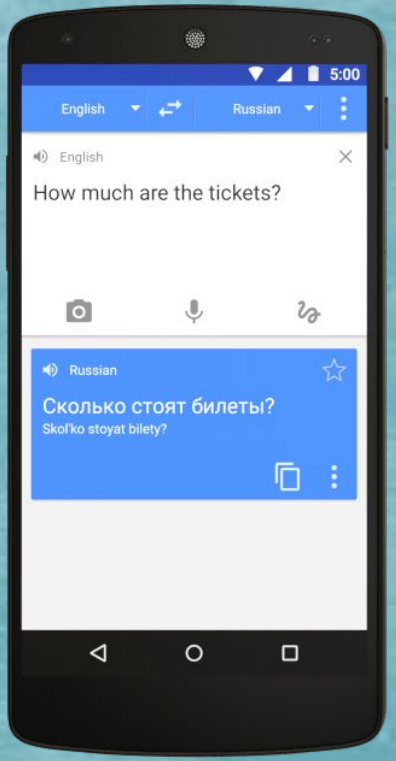
MINIMALISM IN DEEP LEARNING

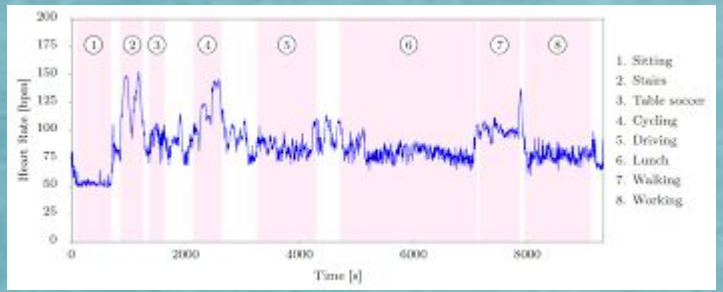
Dissertation Defense of Louis Jensen

Deep Learning Started a Revolution



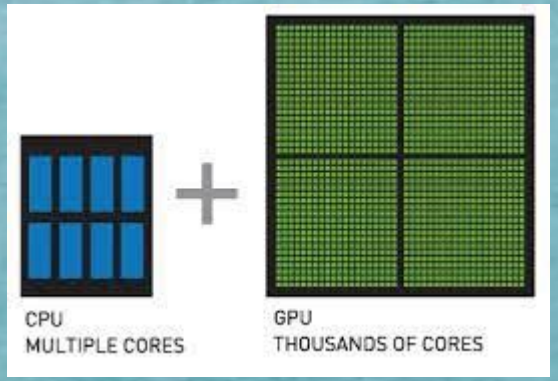


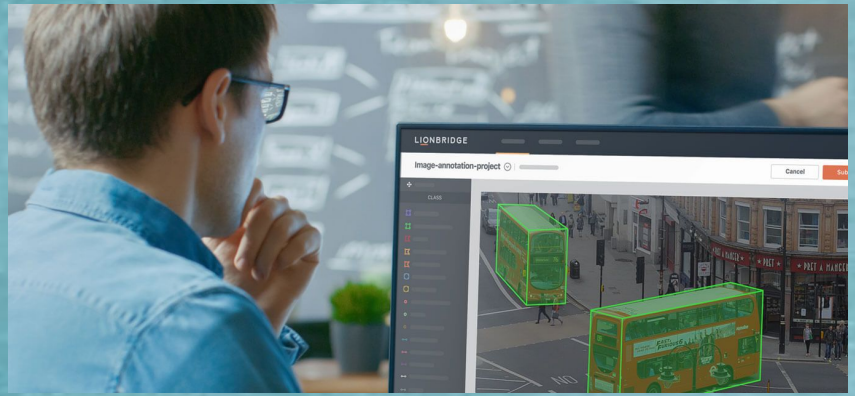
Deep Learning Started a Revolution



But Deep Learning is Still Limited

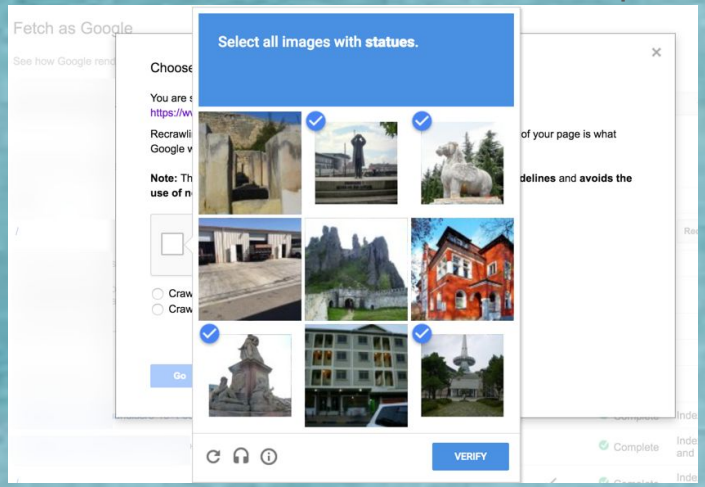
- Computational demands of training
- Need for labelled data
- Computational demands for serving on small devices
  - i.e. Internet of Things, embedded systems

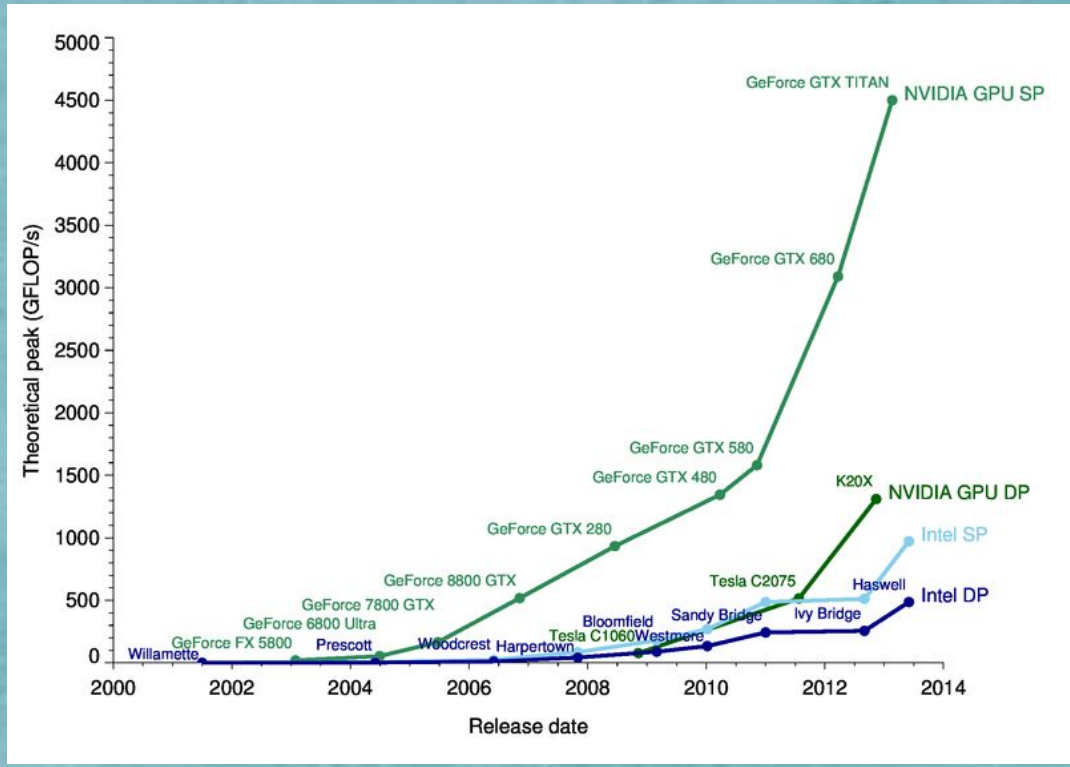




Deep Learning's Limitations come from Limited Resources

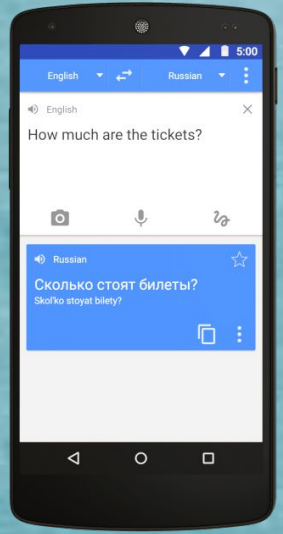
- Deep learning can advance in two ways
  - Improve availability of limited resources
  - Improve resource efficiency
    - Minimalism in Deep Learning





Deep Learning Resource Efficiency

- If deep learning can succeed with less, then deep learning can succeed at more
  - More devices will be equipped to operate Neural Networks (NNs)
  - More computationally efficient NNs will be able to solve more difficult tasks
  - More tasks will become feasible with NN's that learn more efficiently from measurements, data, and labels



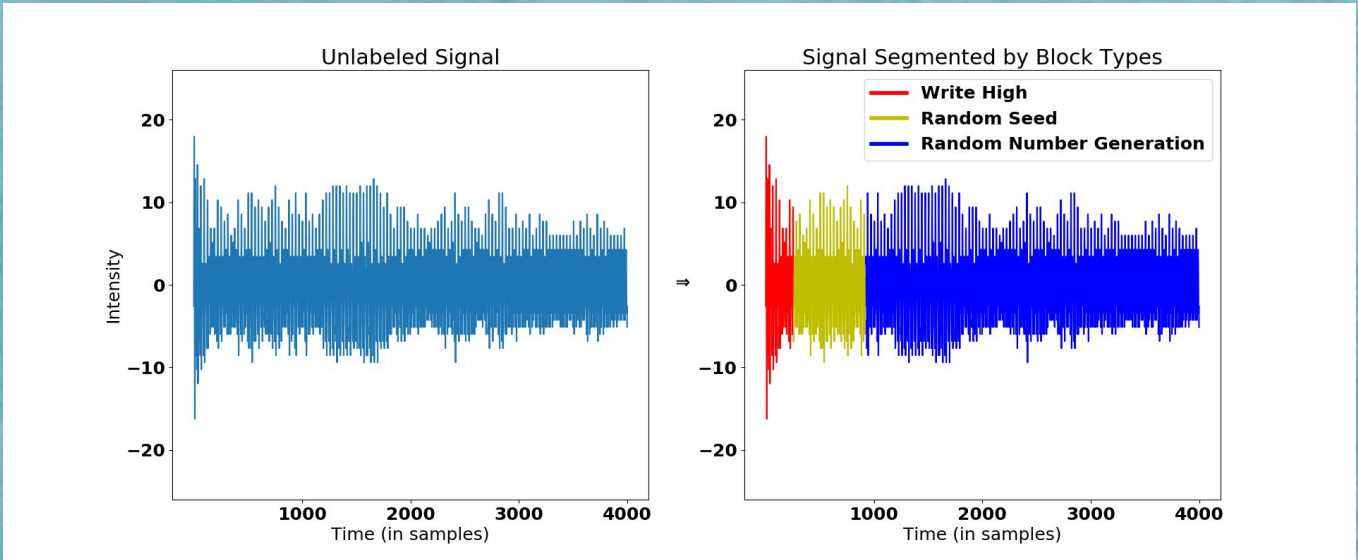
Deep Learning Minimalism Applications in this Work

- Minimal Context Block Tracking
  - Performs side-channel analysis with as few measurements as possible while maintaining performance
- NodeDrop
  - Reduces the memory footprint and computational cost of neural networks while maintaining performance
- Anomaly Detection
  - General purpose time-series anomaly detection without needing expensive domain specific labelled data
- Image Compression
  - A unique approach to using NNs for image compression, allows us to compare the explanatory power of NNs with more traditional compression algorithms

Deep Learning for Side-Channel Analysis

- A side channel is any unintended channel of information leakage from a device
  - Examples include power drain, electromagnetic (EM) radiation, fan sound, temperature output
- We use the power and EM side channels to analyze program execution on the device

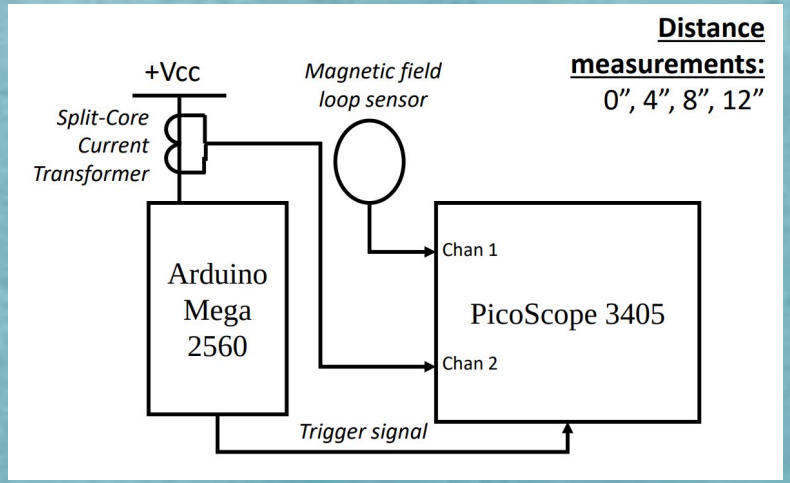
Block-Tracking: Intro



Label every time point with its corresponding block-type. Example shown above.
Note that blocks often repeat the same block-type consecutively.

Block-Tracking: Data Used

- Measuring two channels, power consumption and EM radiation
- Two programs tested with 3-5 block-types:



Program	**Random Number Gen.**	**Bit-Toggling**
Block-Types	-Write Low -Write High -Random Number Seed -Random Number Gen. -Loop	-Write Low -Write High -Loop

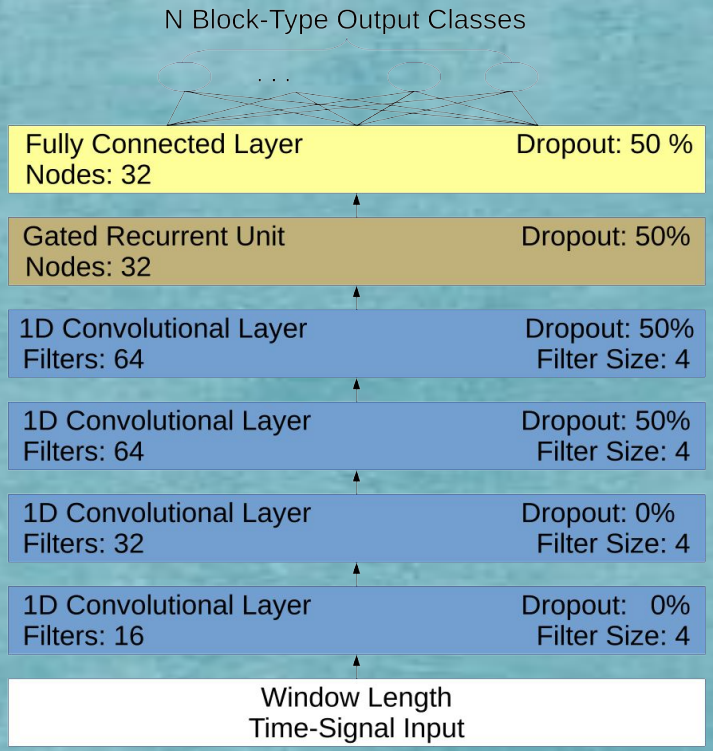
Minimal Context Block-Tracking

Why it Matters:

- Block-Tracking is valuable on its own as a tool for identifying program flow
- Block-Tracking can assist with other execution analysis tasks using blocks as a sort of “alphabet” or “dictionary”
- Finding a minimum required context for block-tracking gives guidance for reasonable window size when approaching future execution analysis tasks

Block-Tracking: Model

- Model Input: Sub-window of measured data from signal
- Early Layers: Multiple 1D convolution layers learn signal features
- Middle layer: Recurrent GRU layer will enable network to train on a different length window then tested
- Late Layers: Fully Connected layer leads to an output block-type prediction for a given window input
- Repeated: This network is applied to all sub-windows of a signal to predict block-type over the whole signal time



Block-Tracking: Minimum Context

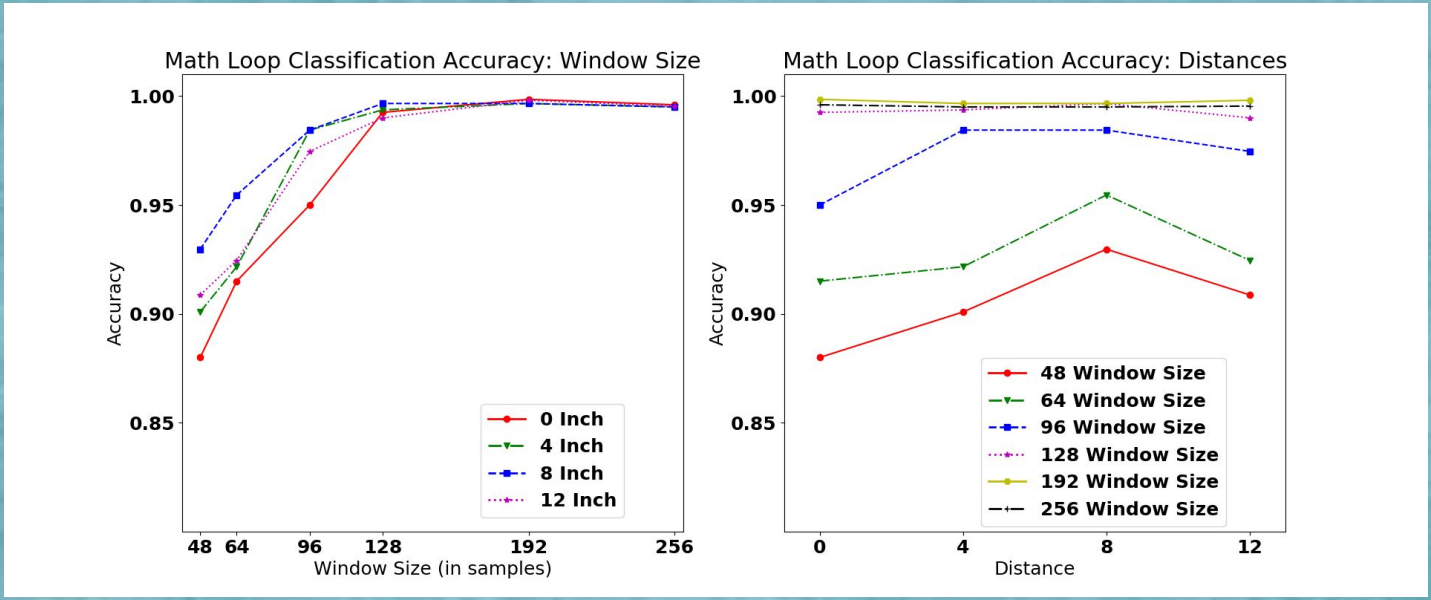
- Goal 1: Perform block-tracking with high accuracy
- Goal 2: Find the minimum window-size at which block-tracking does not lose significant accuracy
  - Gives us a maximum resolution estimate for block-tracking
  - Provides insight for window-size selection in future work (see later this slideshow). This is very beneficial!

Block-Tracking: Performance

- Model performed block-tracking with accuracies above 99% for sub-windows of size 128 and above.
- Using the Math (Random Number Generation) Program experiments we identify sub-window size of 128 as a good candidate
- Model performed well with sub-windows of size 128 for all distance and program setups

		Distance			
Program	Window	0”	4”	8”	12”
Math	48	87.9	90.1	93.0	90.9
	64	91.5	92.2	95.5	92.5
	96	95.0	98.4	98.4	97.5
	128	99.2	99.4	99.7	99.0
	192	99.9	99.7	99.7	99.8
	256	99.6	99.4	99.5	99.5
Toggle	128	99.7	98.3	100	99.7

Block-Tracking: Performance



Block-Tracking: Conclusion

- We are able to perform block-tracking with high accuracy
- We found that a sub-window of size 128, ~4 instruction cycles in our setup, was the smallest window size for which our model performed well
- For future execution analysis tasks requiring windows, we will start by using ~4 instruction cycles as a starting point

NodeDrop

- We want small neural networks
  - Cheaper storage, memory, and runtime
  - Expand opportunities for local neural network applications on phones, embedded systems, IoT devices
- … but not too small!
  - A network should still be appropriately sized to achieve desired task
- Choosing the number of features per layer can be a challenging and time intensive task for neural network architects.

Categories for Methods of Network Compression

- Iterative heuristic pruning and retraining
  - Training time is significantly extended by these techniques
- Quantized or binarized networks
  - Could be stacked with other methods relatively easily
- Regularization or sparsifying techniques
  - Our technique will be a part of this category

An Overview of our Technique

- We propose the NodeDrop conditions, one for vanilla and one for batchnorm, to provide a guarantee for when a neuron is not used
  - Take advantage of “Dying ReLU” effect of the flat region in ReLU
- Then we propose regularization to encourage meeting the condition above
  - The regularization will push nodes to satisfying our “dead” node conditions in the absence of gradients from the primary objective function
- All proofs apply to convolutional and fully connected layers
- We demonstrate results significantly reducing network sizes while maintaining performance

The Vanilla NodeDrop Condition

We propose the NodeDrop condtion:

1. Given a node with input vector $\vec{x} \in [0,1]^n$, a weight vector $\vec{w} \in \mathbb{R}^n$, bias b $\in \mathbb{R}$, and an activation function σ such that $\sigma(v) = 0 \ \forall v \leq 0$.
2. We wish to find the condition under which this node is dead, $\sigma(\vec{w} \cdot \vec{x} + b) = 0$ for all inputs $\vec{x}$.

The Vanilla NodeDrop Condition (cont.)

3. Since $\sigma(v) = 0 \; \forall v \leq 0$, we simply need to find the condition under which $\vec{w} \cdot \vec{x} + b \leq 0$. We have constrained the inputs to be within $[0, 1]$, $\vec{x} \in [0, 1]^n$, so we have:

$$\vec{w} \cdot \vec{x} + b \leq \| \max(\vec{w}, \vec{0}) \|_1 + b \leq \|\vec{w}\|_1 + b$$

4. Then, $\| \max(w_i, 0) \|_1 + b \leq 0 \Rightarrow \sigma(\vec{w} \cdot \vec{x} + b) = 0$

This leaves us with the NodeDrop condition:

$$\| \max(w_i, 0) \|_1 + b \leq 0 \qquad (1)$$

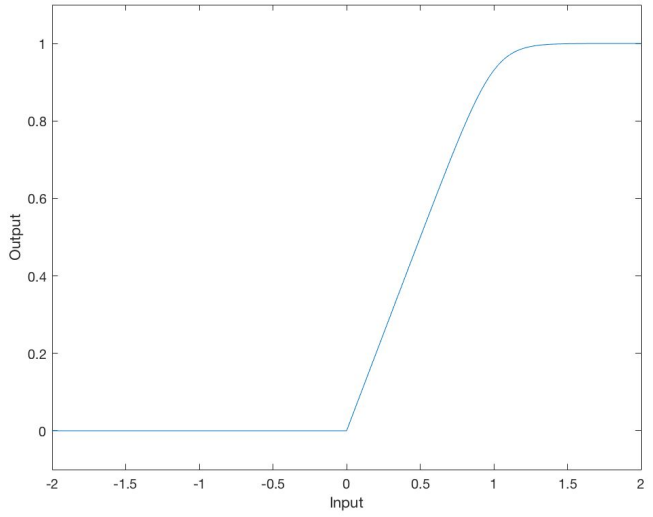
Activation function

1. Given a node with input vector $\vec{x} \in [0,1]^n$, a weight vector $\vec{w} \in \mathbb{R}^n$, bias b $\in \mathbb{R}$, and an activation function σ such that $\sigma(v) = 0 \ \forall v \leq 0$.

Conditions for Activation function:

$$\sigma(v) = 0 \ \forall v \leq 0 \quad (2) \qquad \sigma(v) \in [0,1] \ \forall v \quad (3)$$

We used SoftClampedReLU



Regularization

- We want to encourage the condition $\max(\vec{w}, \vec{0}) + b \leq 0,$
- Specifically, we will encourage $\|\max(\vec{w}, \vec{0})\|_1 + b = -C$ for some constant C
- Thus, we used the following regularization:

$$\lambda|\|\max(\vec{w}, \vec{0})\|_1 + b + C| = \lambda|\sum_i \max(w_i, 0) + b + C|$$

- L2 regularization is not appropriate for our goals, as it is cheaper to have multiple identical nodes summed than one unique node with this regularization.

Batch Norm Extension

- We make an added assumption that, in a batch of size m, if a node is guaranteed not to activate, then we can safely remove it.
- Lemma: $|\gamma|\sqrt{m} + \beta \leq 0 \implies y_i \leq 0$
  - γ and β are learnable parameters

$$\mu = \frac{1}{m}\sum_{i=1}^{m} x_i$$

$$\sigma^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu)^2$$

$$\hat{x}_i = \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

$$y_i = \gamma\hat{x}_i + \beta$$

$$\begin{aligned}
\text{given } \hat{x}_i \quad & = \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}} \\
\implies \hat{x}_i^2 \quad & = \frac{(x_i - \mu)^2}{\sigma^2 + \epsilon} \\
\implies \sum_{i=0}^{m} \hat{x}_i^2 \quad & = \frac{\sum_{i=0}^{m}(x_i - \mu)^2}{\sigma^2 + \epsilon} \\
& = \frac{m\sigma^2}{\sigma^2 + \epsilon}
\end{aligned}$$

Batch Norm Edition (cont.)

$$\implies \sum_{i=0}^{m} \hat{x}_i^2 = \frac{\sum_{i=0}^{m} (x_i - \mu)^2}{\sigma^2 + \epsilon}$$

$$= \frac{m\sigma^2}{\sigma^2 + \epsilon}$$

$$\implies \sum_{i=0}^{m} \hat{x}_i^2 \leq m$$

$$\implies -\sqrt{m} \leq \hat{x}_i \leq \sqrt{m}$$

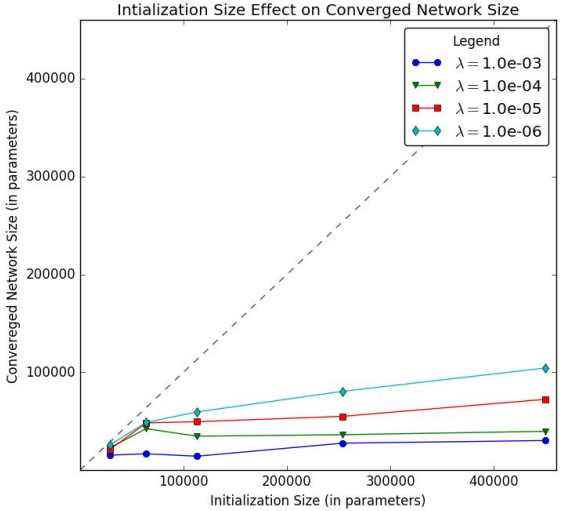
Together $|\hat{x}_i| \leq \sqrt{m}$ and $y_i = \gamma \hat{x}_i + \beta$ imply $y_i \leq |\gamma|\sqrt{m} + \beta$. Therefore

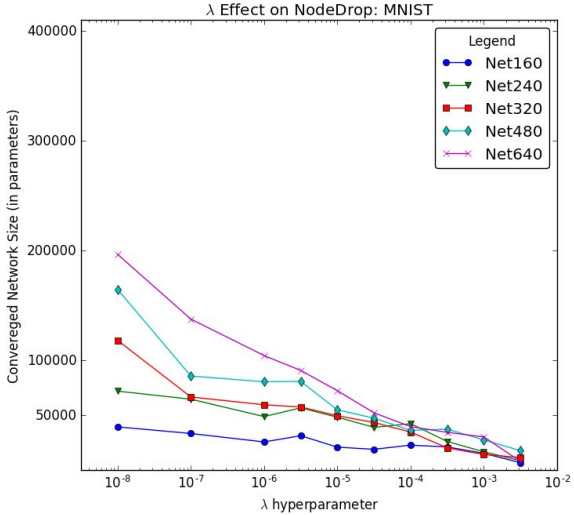
$$|\gamma|\sqrt{m} + \beta \leq 0 \implies y_i \leq 0$$

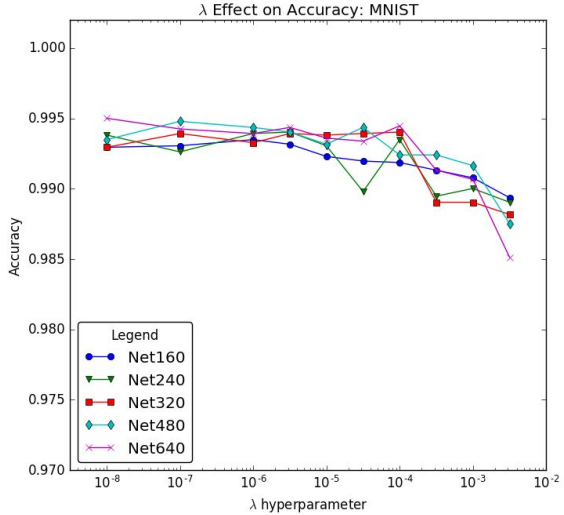
Batch Norm Condition and Regularization

- We assume an activation function that is “dead” for any value less than 0.
- Thus we use the following NodeDrop condition: $|\gamma|\sqrt{m} + \beta \leq 0$
- And regularization term: $|\gamma|\sqrt{m} + |\beta + C|$
- ReLU may be used for the batch norm version of NodeDrop!

MNIST Exploratory Experiments







CIFAR 10 Results

Table 2: Cifar10 Classification Results

Network	λ	Test Error	Parameters	Pruned %	Factor	Nodes	Pruned %
VGG 16 w/o BN	Baseline	13.01	15.04M	0.0	1.0	4736	0.0
	1.0×10^{-6}	14.14	0.45M	97.00	33.28	1115	76.46
	1.0×10^{-5}	13.27	0.31M	97.96	48.98	859	81.9
	3.2×10^{-5}	**13.76**	**0.13M**	**99.12**	**114.00**	**612**	**87.08**
	1.0×10^{-4}	90.00	0.0M	100.0	-	0	100.0
VGG 16	Baseline	6.50	15.04M	0.0	1.0	4736	0.0
	1.0×10^{-6}	6.88	8.88M	40.7	1.69	3624	23.48
	1.0×10^{-5}	7.36	1.39M	90.75	10.81	1164	75.42
	3.2×10^{-5}	**7.41**	**0.61M**	**95.96**	**24.76**	**751**	**84.14**
	1.0×10^{-4}	20.16	0.10M	99.35	152.84	308	93.50
DenseNet40 w/o BN	Baseline	14.94	1.04M	0.0	1.0	456	0.0
	1.0×10^{-6}	15.21	0.66M	35.69	1.55	363	20.39
	1.0×10^{-5}	14.74	0.41M	60.47	2.54	291	36.18
	1.0×10^{-4}	**14.99**	**0.08M**	**91.96**	**12.43**	**154**	**66.22**
DenseNet40	Baseline	6.80	1.05M	0.0	1.0	456	0.0
	1.0×10^{-6}	7.13	0.99M	4.19	1.04	447	1.97
	1.0×10^{-5}	6.75	0.98M	5.67	1.06	443	2.85
	1.0×10^{-4}	**7.79**	**0.55M**	**47.12**	**1.89**	**333**	**26.73**

CIFAR 100 and ImageNet Results

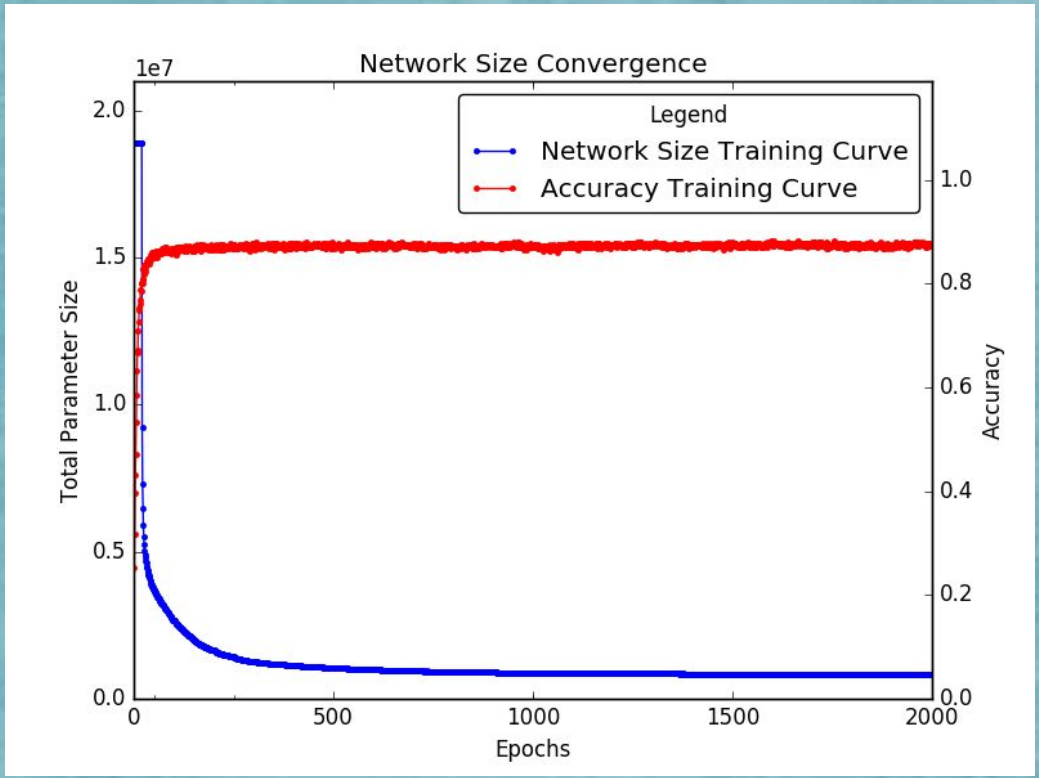
Table 3: Cifar100 Classification Results

NETWORK	λ	TEST ERROR	PARAMETERS	PRUNED %	FACTOR	NODES	PRUNED %
VGG 16	BASELINE	27.65	15.04M	0.0	1.0	4736	0.0
	1.0×10^{-6}	27.69	9.78M	34.99	1.54	3914	17.35
	1.0×10^{-5}	**28.04**	**1.83M**	**87.82**	**8.21**	**1623**	**65.73**
	1.0×10^{-4}	38.49	0.46M	96.93	32.58	729	84.6
DENSENET40	BASELINE	26.5	1.05M	0.0	1.0	456	0.0
	1.0×10^{-6}	26.92	1.05M	2.27	1.02	451	1.09
	1.0×10^{-5}	**27.01**	**1.03M**	**4.74**	**1.05**	**445**	**2.41**
	1.0×10^{-4}	29.38	0.744M	31.12	1.45	376	17.54

Table 4. ImageNet Classification Results

NETWORK	λ	TEST ERROR	PARAMETERS	PRUNED %	FACTOR	NODES	PRUNED %
VGG 19	BASELINE	33.79	143.65M	0.0	1.0	14696	0.0
	1.0×10^{-5}	34.85	23.75M	83.47	6.05	6670	54.61

CIFAR10 Convergence examples

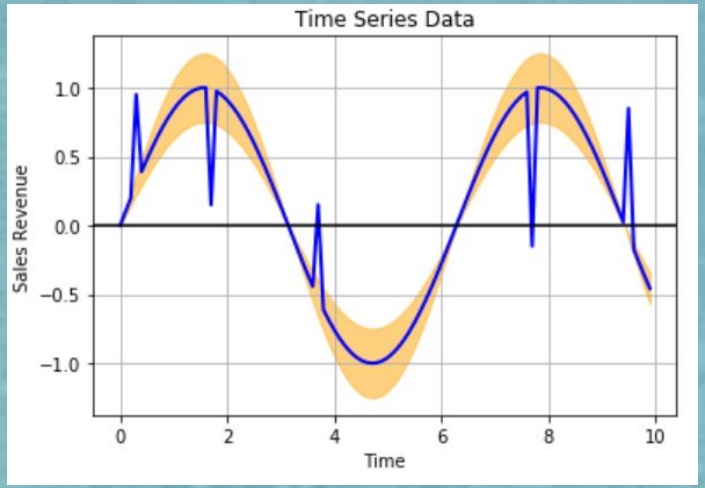


Conclusion

- NodeDrop is a powerful way to reduce the size of simple or complex networks
- Guarantee ensures that removal of nodes does not impact network performance
- Vanilla NodeDrop reduces network size by up to a factor of 100x while maintaining performance with the CIFAR10 dataset
- BatchNorm NodeDrop reduces network size by a factor 25x for CIFAR10, 8x for CIFAR100, and 6x for ImageNet datasets

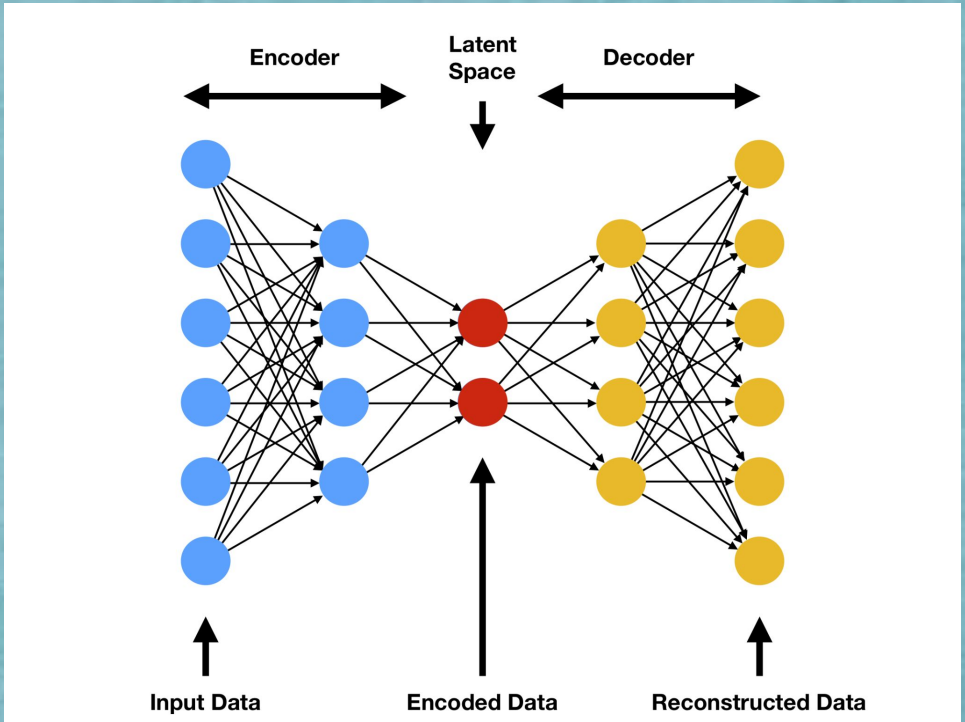
Autoencoders for State-of-the-art in Time-Series Anomaly Detection

- Identify anomalies in time-series signals
- What is an Anomaly?
  - Not normal
  - Cannot be explained by current understanding of Normal
    - Example: At CERN physicists require explanation for event 5 or more standard deviations outside current model
- Anomaly Detection requires an implicit understanding of “Normal”
- Prob[x is Anomaly] = 1 - Prob[x is Normal]
- Normal is explained simply



Autoencoders for Anomaly Detection

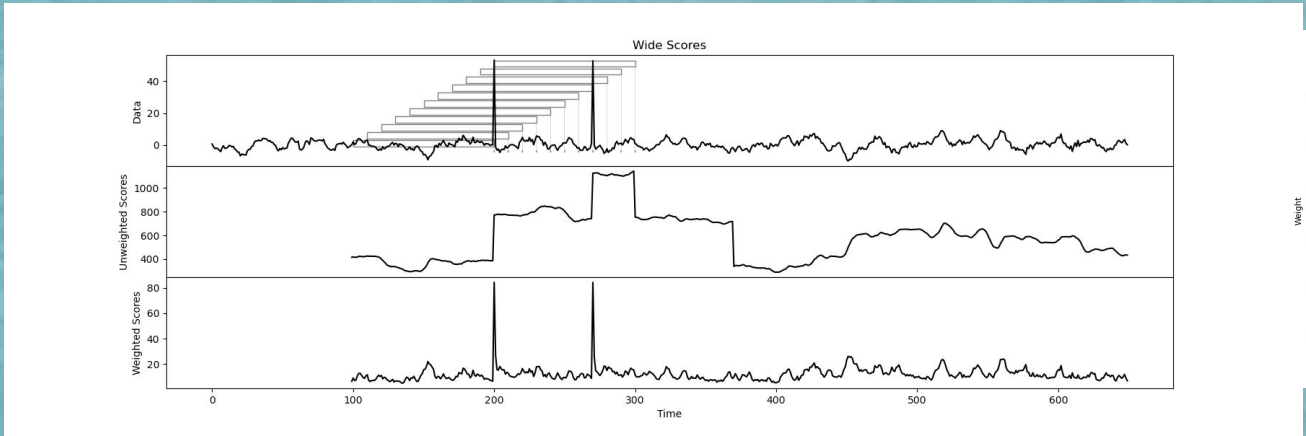
- Latent dimension as “simple” explanation of data
- Anomalous data not well explained or reconstructed
- Autoencoders are a component of many state-of-the-art systems

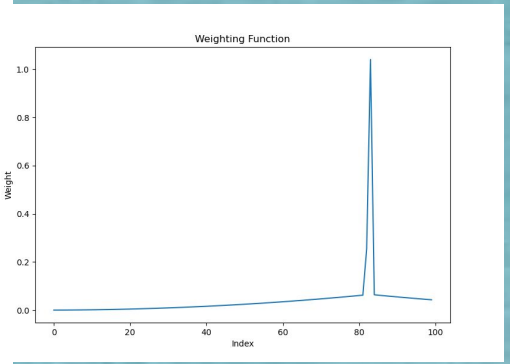


Techniques for Time-Series Anomaly Detection

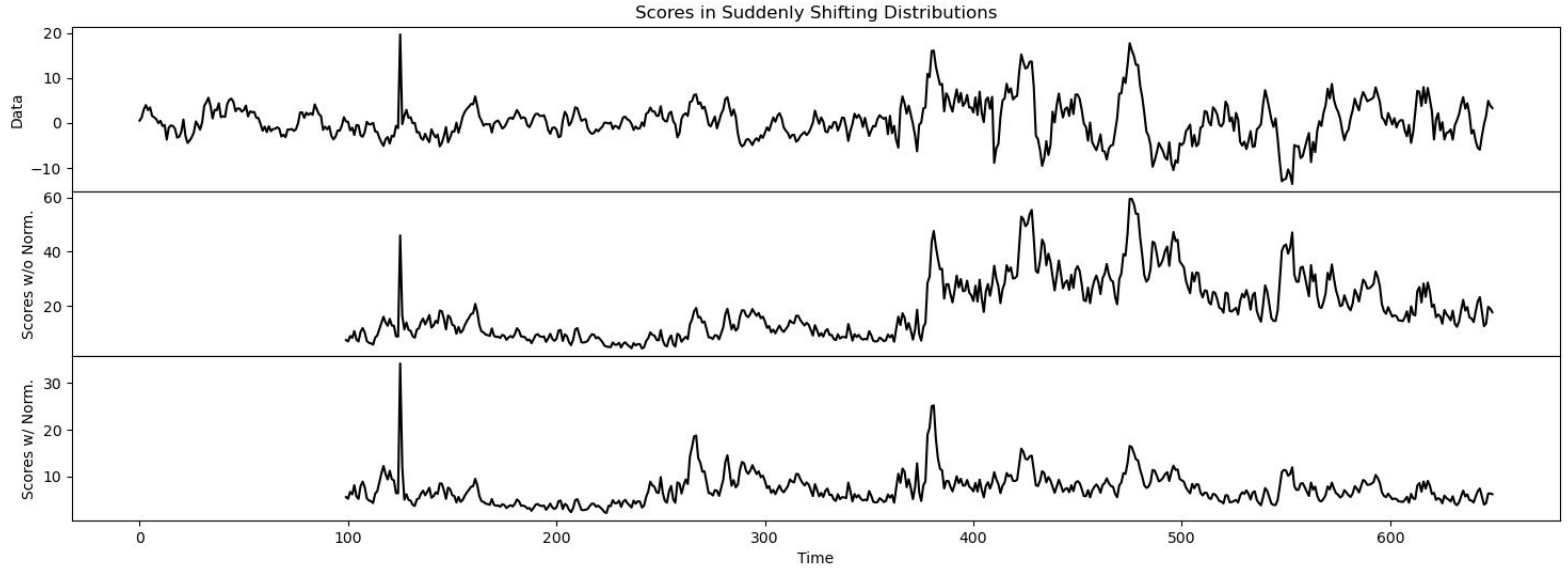
- Reconstruction
- Forecasting
  - Assumptions and Statistics Based
  - Learning Based
- Generative
- Combinations and Variations
  - Combine multiple techniques
  - Vary pre-processing, post-processing and loss

Weighted Window to improve event precision





Normalization to Account for Distribution Shift



Time-series Anomaly Detection data and Metric

- Numenta Anomaly Benchmark, Yahoo, NASA
- F1 = $\frac{TP}{TP+1/2*(FP+FN)}$
  -
- Time Series F1
  - Unfortunate lack of standardization
  - Overlapping windows?
  - Is time step event or a whole anomaly?
  - We use F1 scoring function of TADGAN

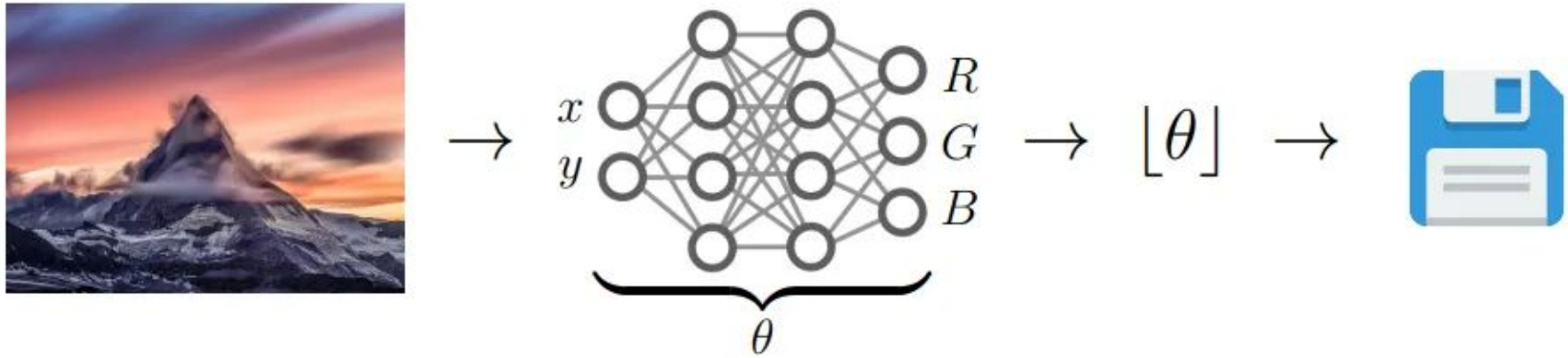
Results

Model	NASA		YAHOO				NAB						
	SMAP	MSL	A1	A2	A3	A4	Art	AdEx	AWS	Traf	Tweets	μ	σ
DenseAE w/ Post	0.623	**0.797**	**0.916**	**0.995**	**0.976**	**0.912**	**0.8**	0.762	0.762	**0.8**	**0.71**	**0.823**	0.115
DenseAE w/o Post	**0.655**	0.608	0.496	0.283	0.097	0.041	0.667	0.533	**0.764**	0.333	0.742	0.474	0.252
TADGAN [5]	0.623	0.704	0.8	0.867	0.685	0.6	**0.8**	**0.8**	0.644	0.486	0.609	0.693	0.114
LSTM [3]	0.46	0.69	0.744	0.98	0.772	0.645	0.375	0.538	0.474	0.634	0.543	0.623	0.171
ARIMA [2]	0.492	0.42	0.726	0.836	0.815	0.703	0.353	0.583	0.518	0.571	0.567	0.599	0.156
Deep AR [17]	0.583	0.453	0.532	0.929	0.467	0.454	0.545	0.615	0.39	0.6	0.542	0.555	0.142
HTM [9]	0.412	0.557	0.588	0.662	0.325	0.287	0.455	0.519	0.571	0.474	0.526	0.489	0.113
MADGAN [18]	0.111	0.128	0.37	0.439	0.589	0.464	0.324	0.297	0.273	0.412	0.444	0.35	0.144
MS Azure [19]	0.218	0.118	0.352	0.612	0.257	0.204	0.125	0.066	0.173	0.166	0.118	0.219	0.152

Neural Networks for Lossy Image Compression

- Encode an image to a compressed form
- Decode to reconstruct similar image
- Traditional approaches
  - Wavelets, Fourier Transforms, Color Depth Projection, etc.
- Deep learning approaches
  - Autoencoders
  - Sinusoidal Representation Networks (SIRENs)

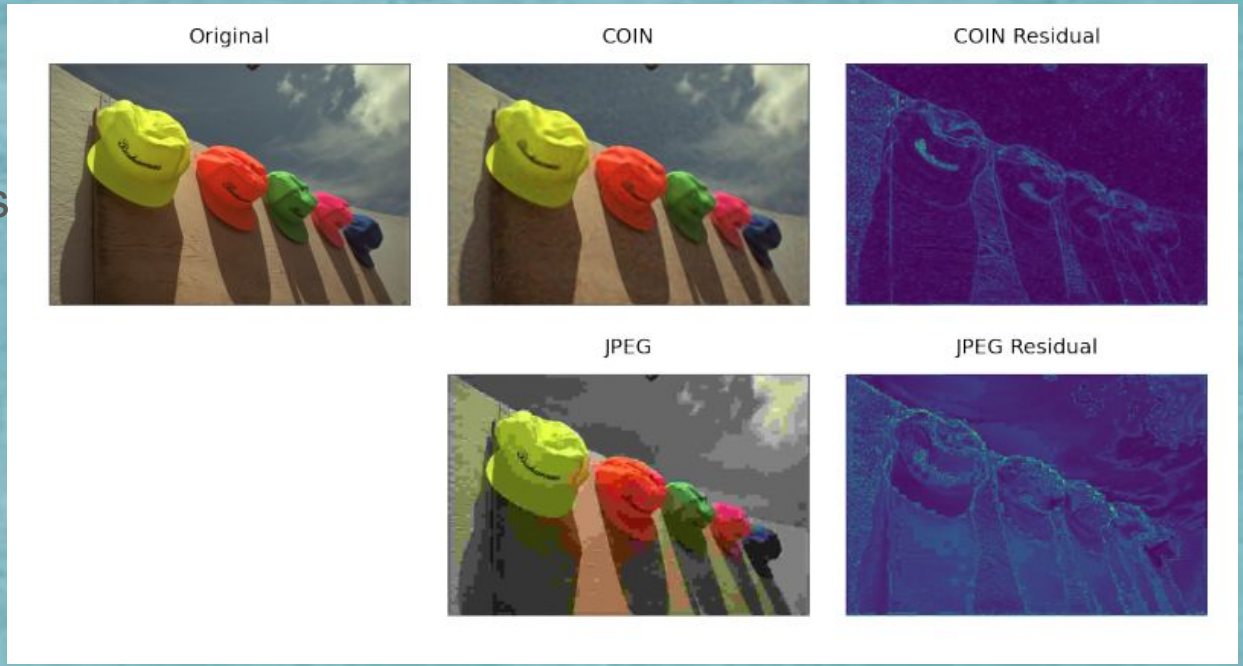
Neural Networks can Store Images



- Consider signals, images, videos, or 3D scenes as functions of space and time coordinates
- Train to learn function of image

Compressed Implicit Neural representation

- Use SIREN's to compress image and send quantized weights instead of image
- Benefits over reconstruction-based approach
  - Small model size
  - Gives concept of 'efficiency' of neural network

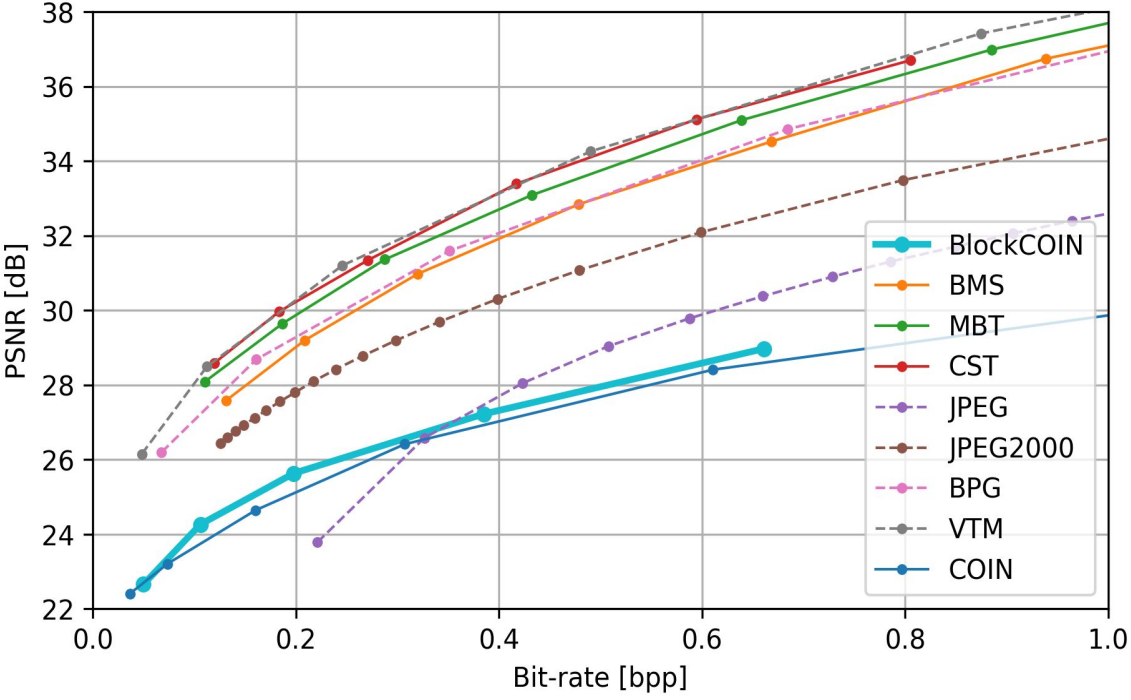


Block-Sparse Matrices

- Theory: Sparse NNs are more efficient per weight
- Set sparsity before training instead of after
- Block-sparse structure more efficient
- Experiment: Do block-sparse weights perform better per weight than full matrices for image compression?

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_1 & 0 & \cdots & 0 \\ 0 & \mathbf{A}_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \mathbf{A}_n \end{bmatrix}$$

Rate Distortion Plot



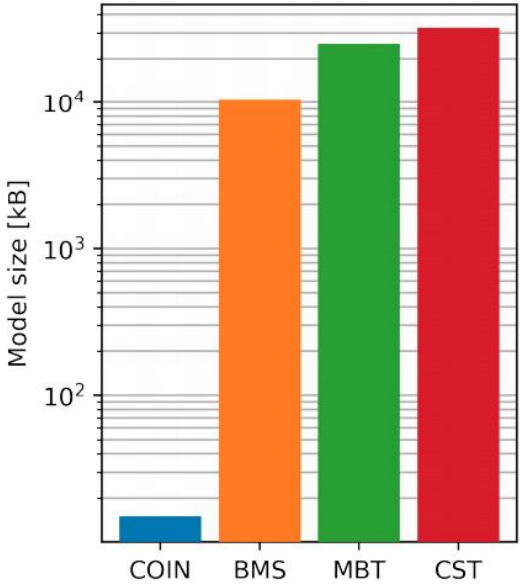


Figure 3: Model sizes at 0.3bpp.

Outlook

- Entropy coding, more complete quantization analysis
- Regularization combined with delta coding on the blocks
- Preprocessing images

In Conclusion

- Minimalism in deep learning allows for the expansion of deep learning applications through improved efficiency
- Minimalism in deep learning is a broad topic, in this talk we explored:
  - Minimizing number of input measurements for side-channel analysis
  - Minimizing number of nodes in general purpose neural networks
  - Using autoencoders for unsupervised, general purpose, and highly performant time-series anomaly detection
  - Improving an exciting new approach to image compression, and offering insight into a technique which could have broader implications to network efficiency