THE OHIO STATE UNIVERSITY

Data-Aware Tuning of Deep Learning Models

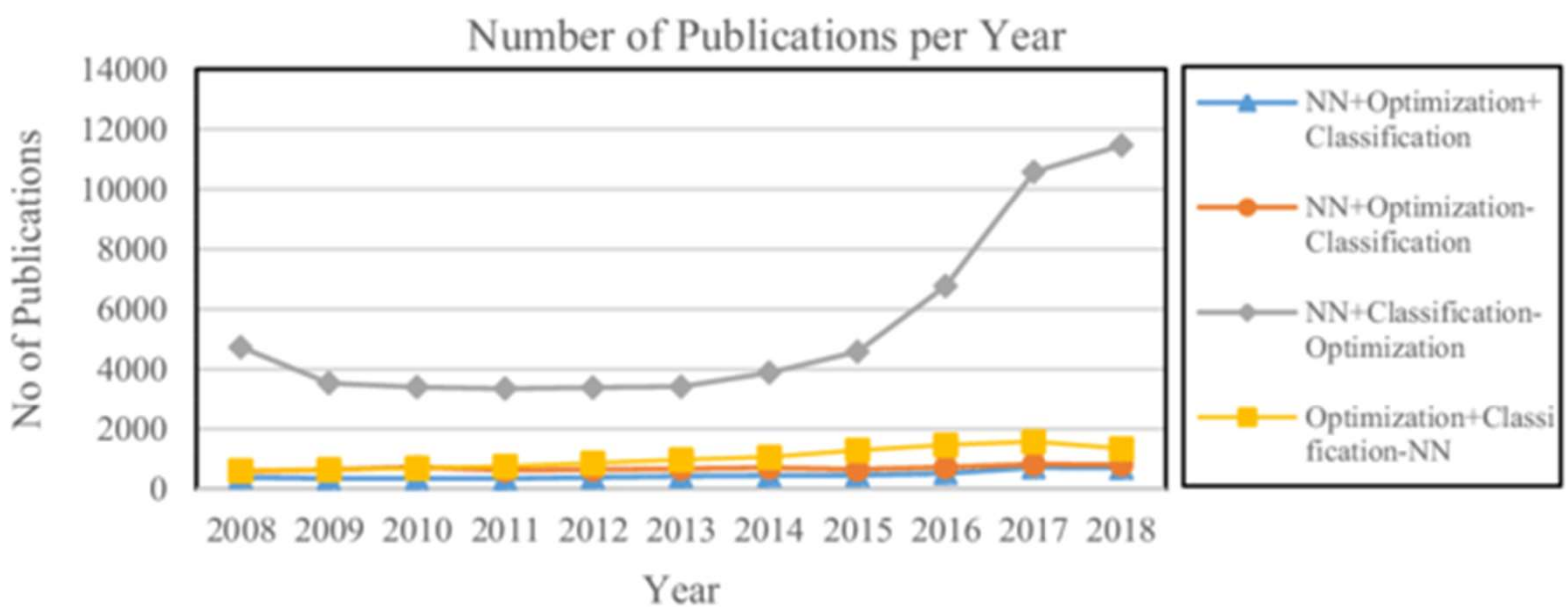
Anthony Baietto

Committee: Dr. Chris Stewart, Dr. Radu Teodorescu, Dr. Mi Zhang, Dr. Atanas Rountev, Dr. Trevor Bihl (AFRL)

Outline

- **Introduction**
  - Current Trends
  - Research Goals
- **Autowave**
  - Radar Waveform Design Overview
  - “Lean Neural Networks for Autonomous Radar Waveform Design”
  - “Lean Neural Networks for Real-time Embedded Spectral Notching Waveform Design”
- **SNN-GAN**
  - Spiking Neural Networks (SNNs) Overview
  - Generative Adversarial Networks (GANs) Overview
  - “Toward Robust Spiking Neural Networks”
  - “Dataset Augmentation for Robust Spiking Neural Networks”
- **Current & Future Work**

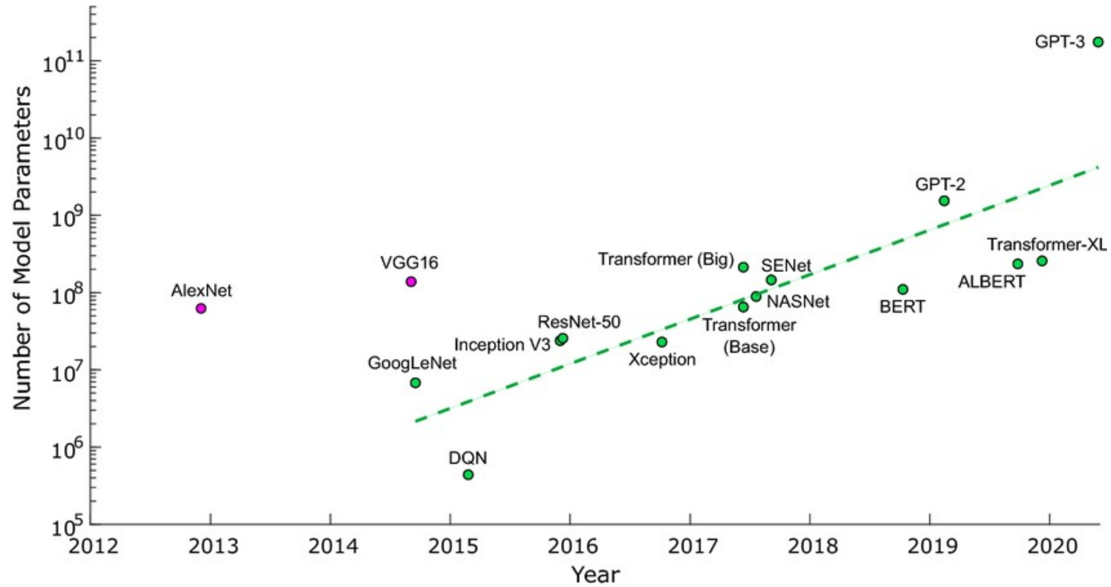
- Neural Network (NN) popularity growing very quickly [1]
- NN boasts superb non-linear function approximation [2, 3, 4] applied to many domains

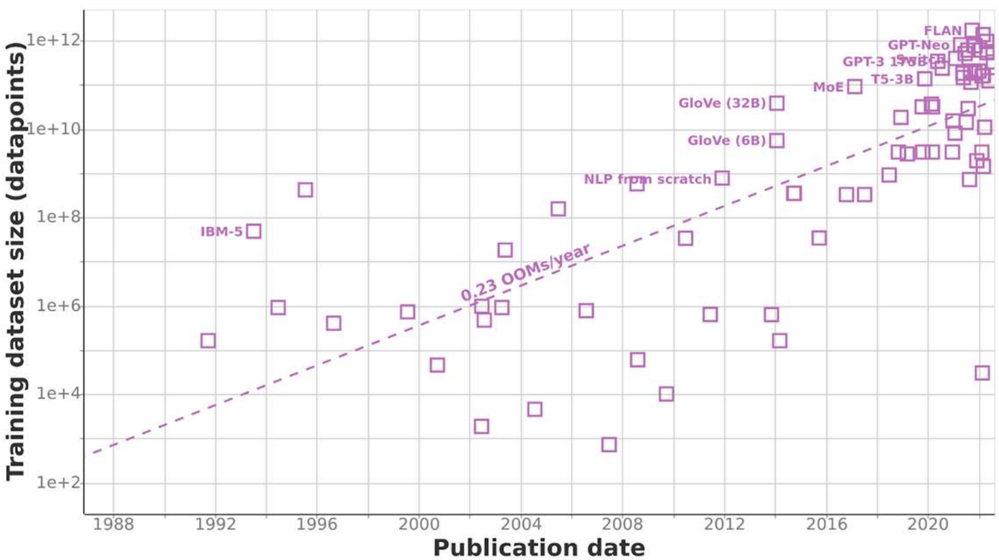


Introduction

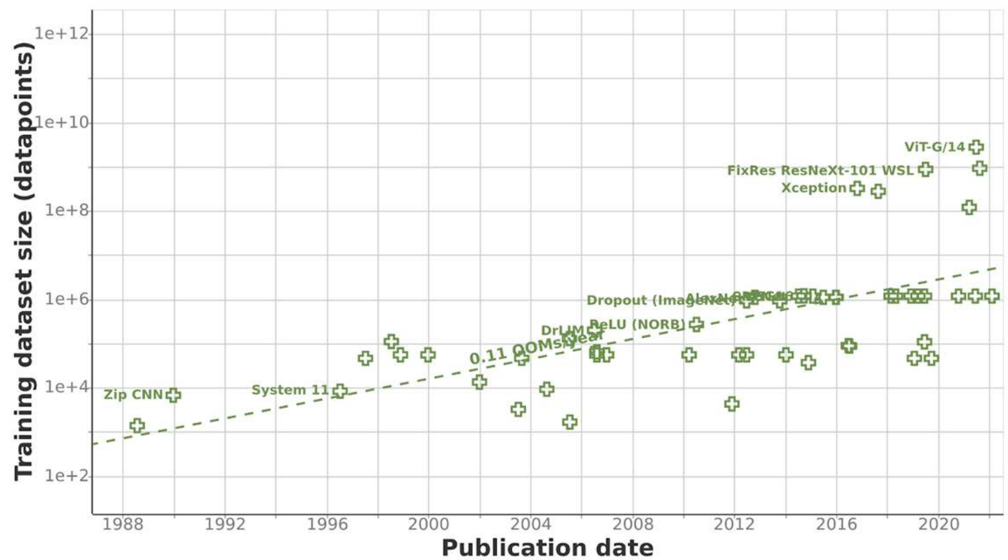
Current Trends

- NN dataset size growing [5]
- NN model size growing [6]



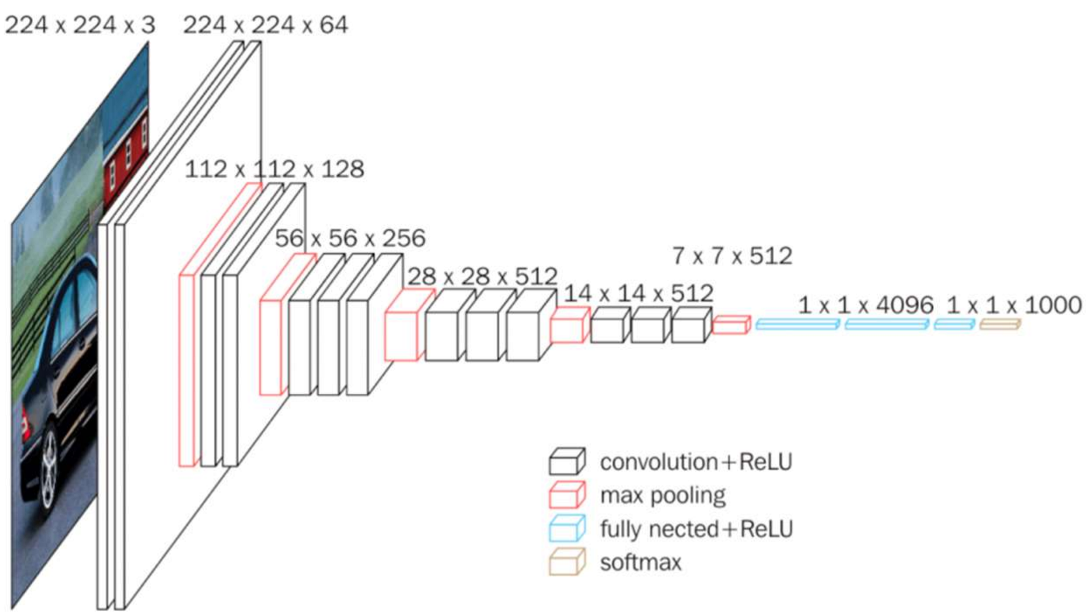


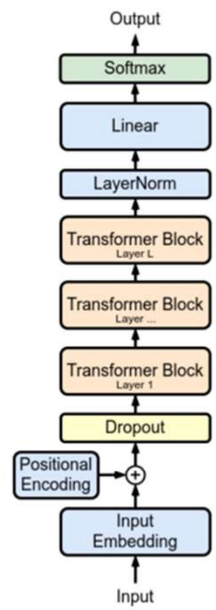
NLP

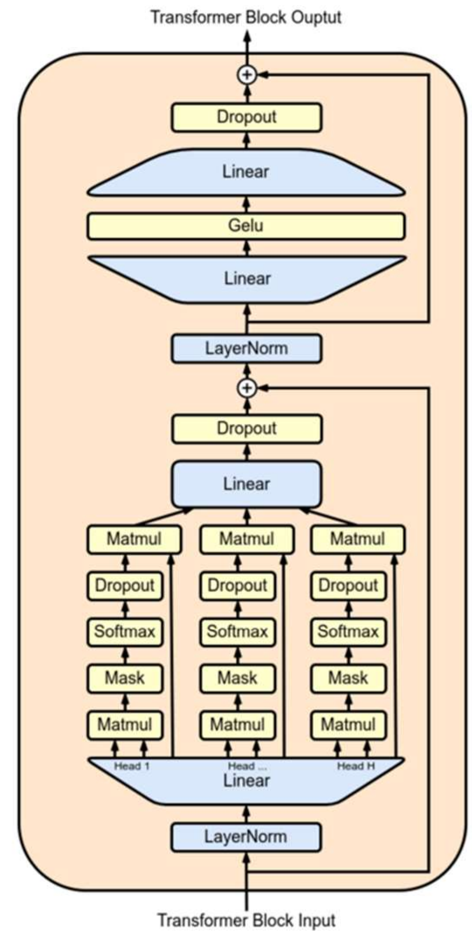


vision

- Harder problem → more data → larger model
- "1000 typewriting monkeys vs 1 Shakespeare"
  - Trend is to keep adding blocks to black-box model
  - **Are they <u>all</u> necessary?**
  - **Could design fewer more intelligent modules**







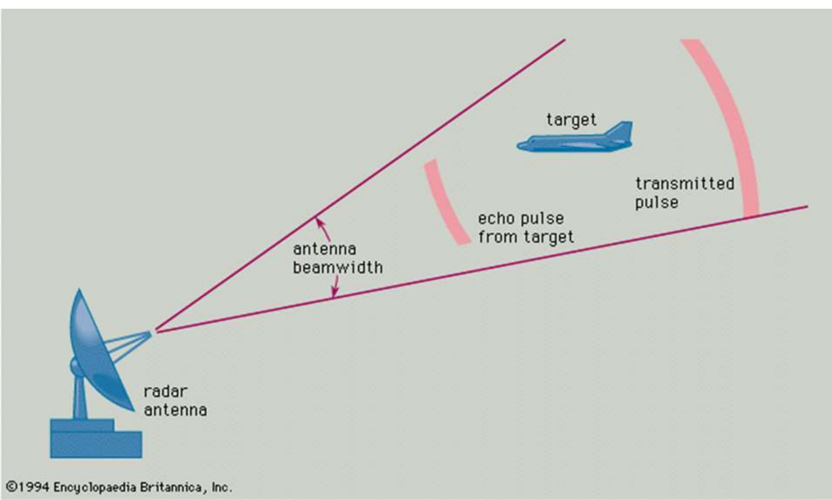
- Leverage extant problem information
- Reduce model footprint
- Reduce data footprint

Outline

- **Introduction**
  - Current Trends
  - Research Goals
- **Autowave**
  - Radar Waveform Design Overview
  - "Lean Neural Networks for Autonomous Radar Waveform Design"
  - "Lean Neural Networks for Real-time Embedded Spectral Notching Waveform Design"
- **SNN-GAN**
  - Spiking Neural Networks (SNNs) Overview
  - Generative Adversarial Networks (GANs) Overview
  - "Toward Robust Spiking Neural Networks"
  - "Dataset Augmentation for Robust Spiking Neural Networks"
- **Current & Future Work**

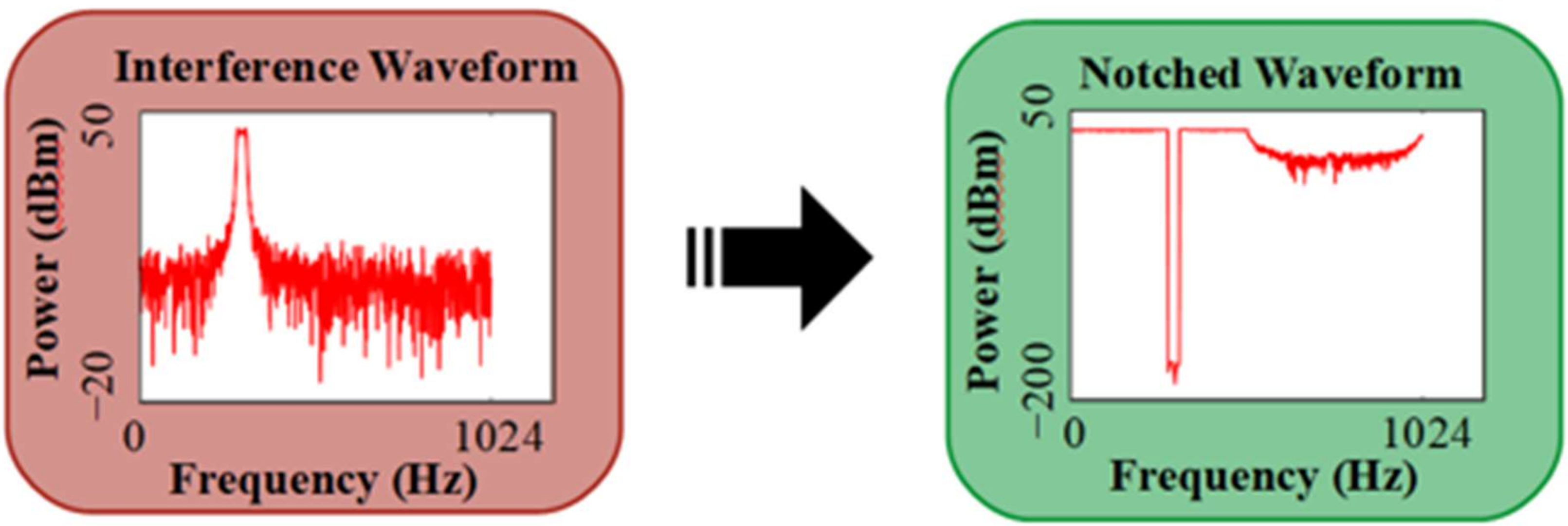
Increased Wireless Spectrum Interference

- 4G/5G telecommunication networks
- Mobile sensors
- IoT devices

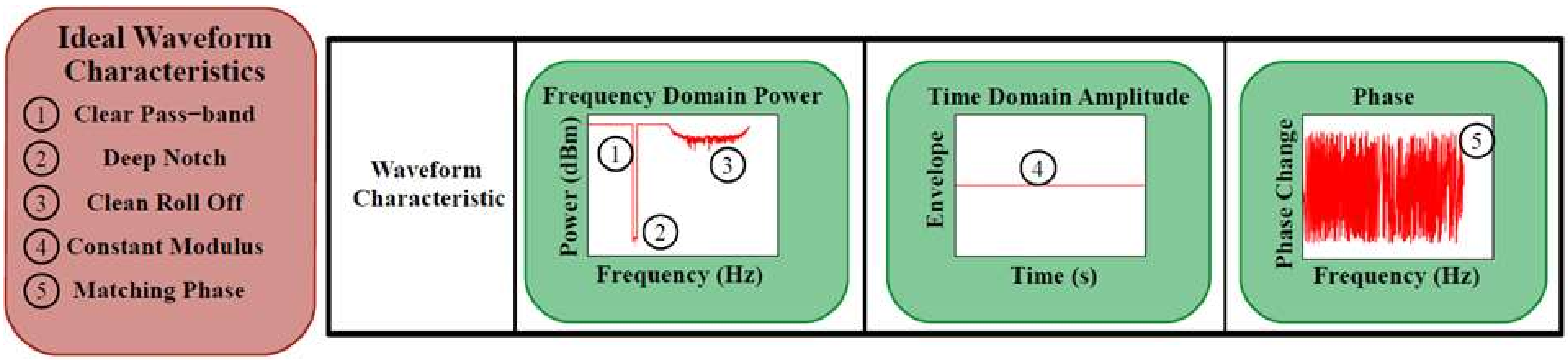


Interference Mitigation with Spectral Notching

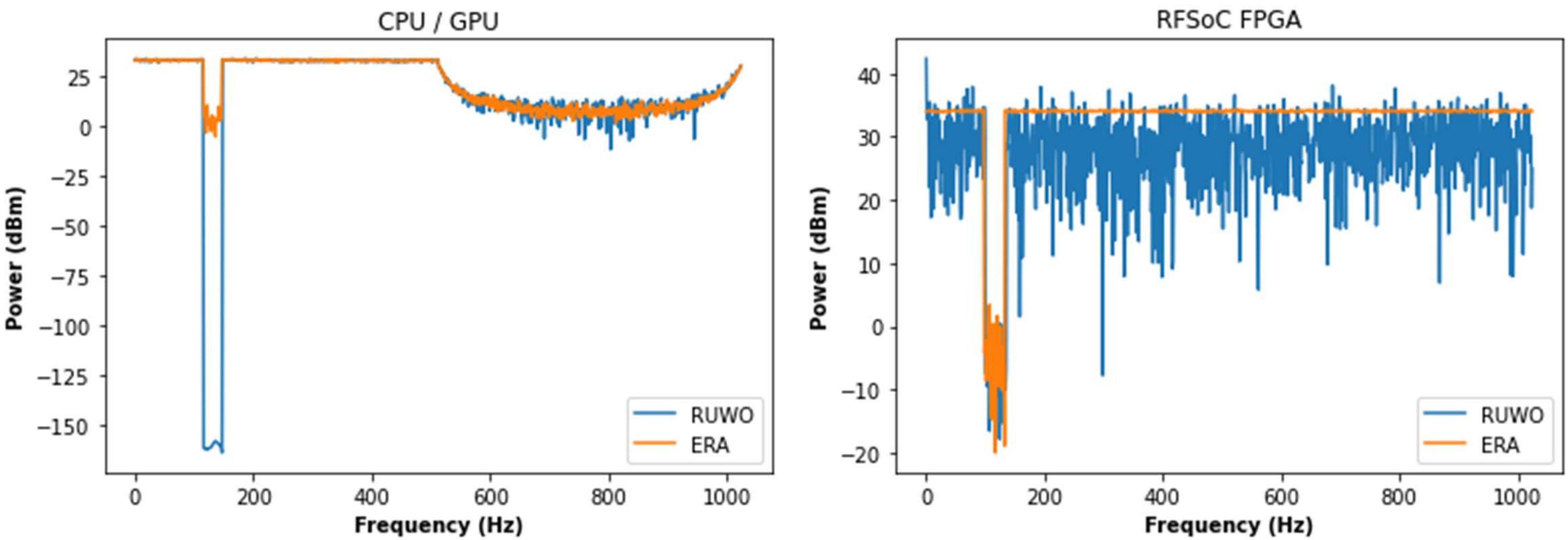
- Sample RF environment
- Determine interfered stopband
- Modify transmit waveform to avoid stopband



- Difficult task with multiple constraints that must be met for radar functionality and power efficiency
- Trade-off between runtime/power and precision
  - Want near real-time without sacrificing performance



- Solutions must also be portable to different hardware
  - RFSoC FPGA (Radio Frequency System on Chip Field-Programmable Gate Array) has fixed-point representation limit for example



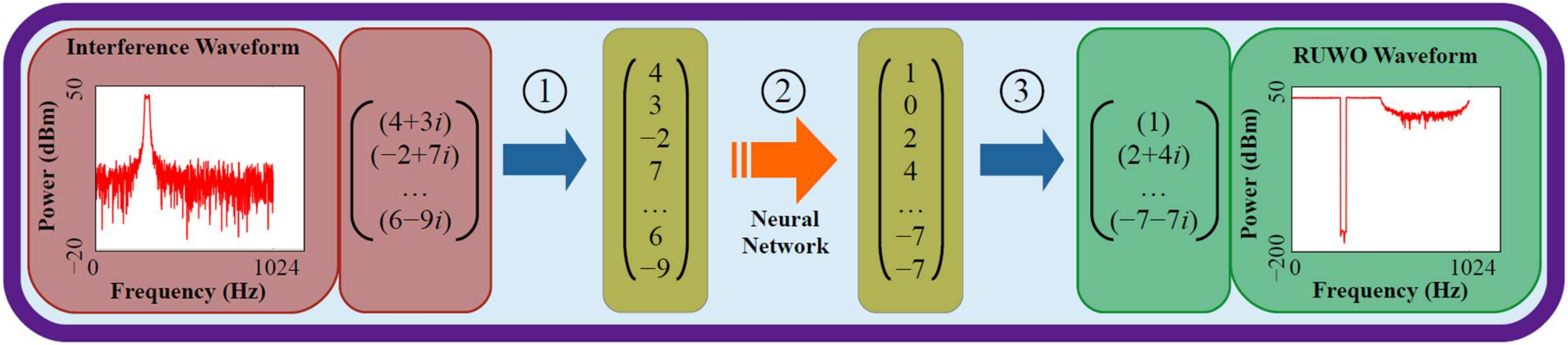
Work	Low SWaP	End-to-End	Hardware Portability	“Intelligent Design”
Error Reduction Algorithm (ERA) [7]		X		X
Re-Iterative Uniform Weight Optimization Algorithm (RUWO) [8]		X		X
MIMO GPU [9]		X	X	X
TCNRWR [9]	X		X	X
RVTDCNN [10]	X	Only output	X	
Autowave pre-computed [11, 12, 13]	X		X	
Autowave [AB1, AB2]	**X**	**X**	**X**	**X**

- AutoWave → Artificial Intelligence (AI) implementation of an adaptive radar system which uses NN to adjust transmitted waveforms to avoid sources of interference
  - Treat RUWO as absolute, train NN to learn RUWO
- Naïve approach of simply throwing a larger NN will not work

Algorithm	GPU Latency (μs)	CPU Latency (μs)	Cosine Similarity	Null Depth (dBm)
NN MSE 1 Layer	747.71 ± 5.23	786.44 ± 5.01	$0.9901 \pm 7.69 \times 10^{-5}$	28.54 ± 0.16
NN MSE 2 Layers	749.40 ± 5.61	797.92 ± 5.99	$0.9900 \pm 7.89 \times 10^{-5}$	29.17 ± 0.22
NN MSE 3 Layers	797.72 ± 10.03	855.34 ± 6.38	$0.9898 \pm 9.87 \times 10^{-5}$	26.57 ± 0.23

Moving away from Mean Squared Error (MSE)

- Numerical comparisons between coefficient vectors prone to errors

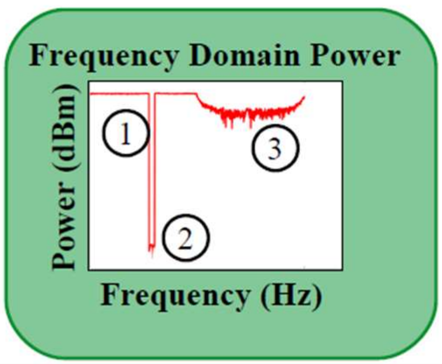
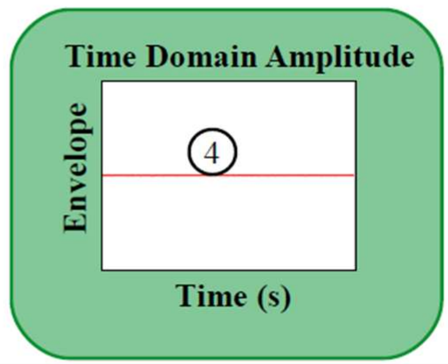
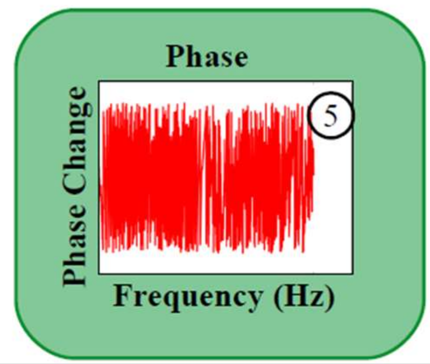


Tailor loss function to radar waveform design

1. Provide quicker learning to valid solutions compared to MSE
2. Encourage NN to always produce valid waveforms (even if not identical to RUWO)
3. Discourage "close enough" waveforms which are similar but not valid

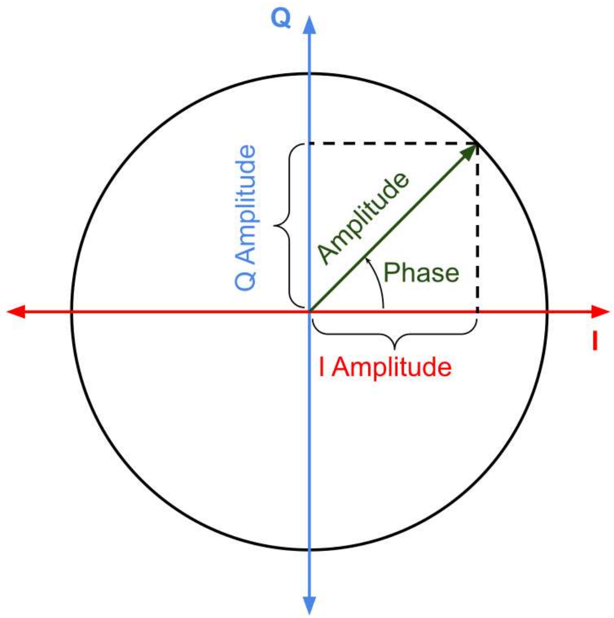
Ideal Waveform Characteristics

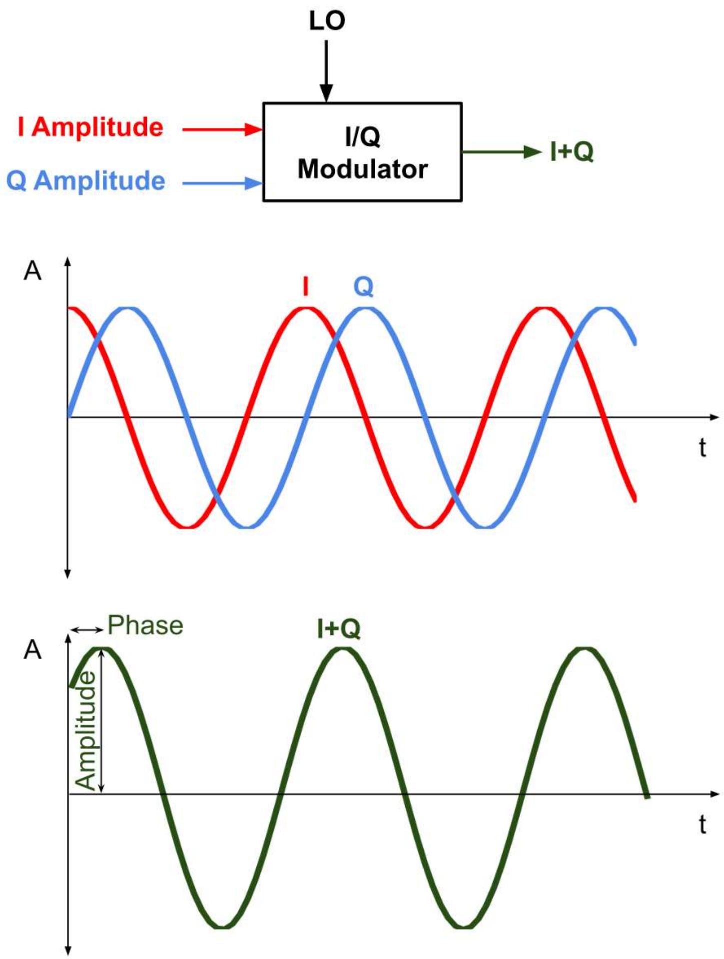
1. Clear Pass–band
2. Deep Notch
3. Clean Roll Off
4. Constant Modulus
5. Matching Phase

Waveform Characteristic			
Reflection in Loss Function	$\frac{1}{n}\sum_{i=1}^{n}(20*\log_{10}(FFT(\lvert z_i\rvert)) - 20*\log_{10}(FFT(\lvert\hat{z}_i\rvert)))^2$	$\frac{1}{n}\sum_{i=1}^{n}(IFFT(\lvert z_i\rvert) - IFFT(\lvert\hat{z}_i\rvert))^2$	$\frac{1}{n}\sum_{i=1}^{n}(\angle FFT(\lvert z_i\rvert) - \angle FFT(\lvert\hat{z}_i\rvert))^2$

Split processing into 2 parallel NN

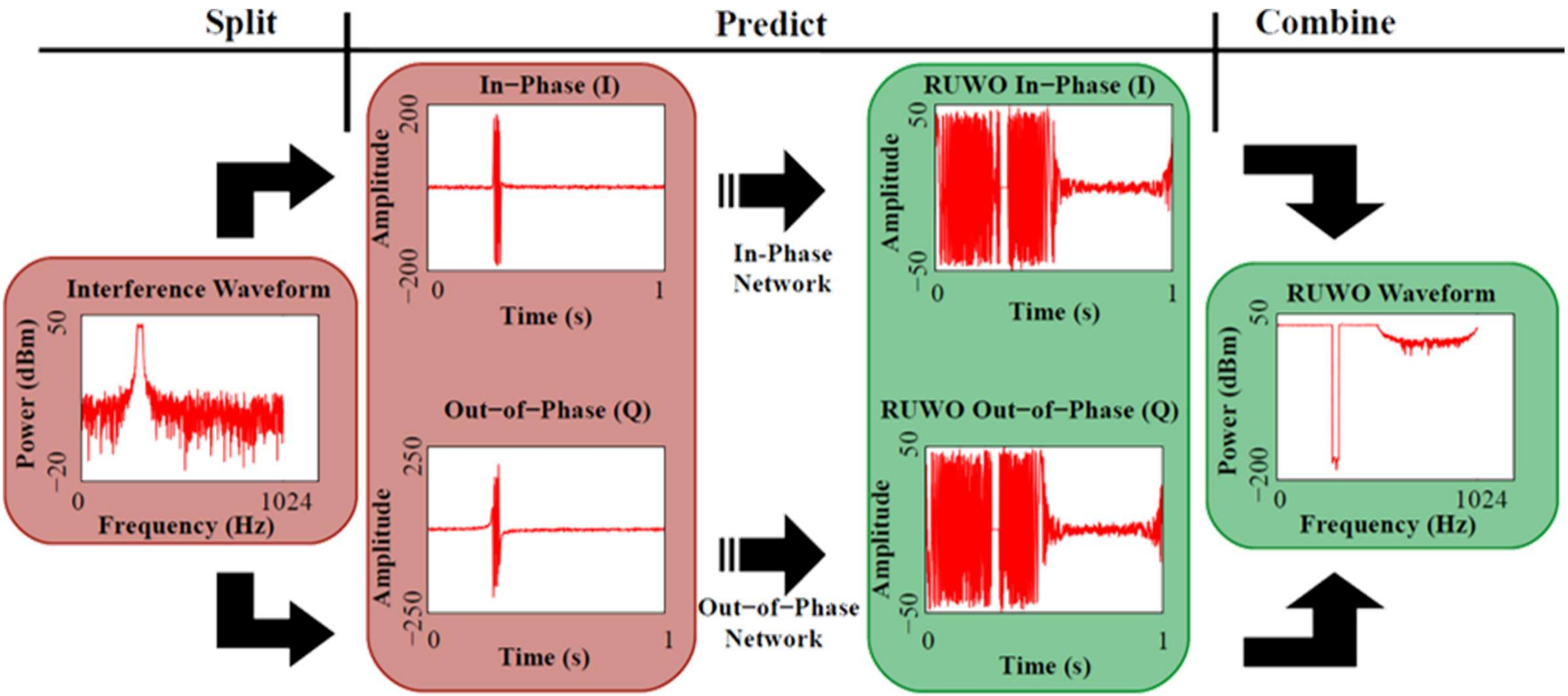
- Quadrature radar waveforms are separate





I Amplitude (V)	Q Amplitude (V)	I+Q	
		Amplitude (V)	Phase (°)
1	0	1	0
0	1	1	90
-1	0	1	180
0	-1	1	270

Split processing into 2 parallel NN (cont'd)



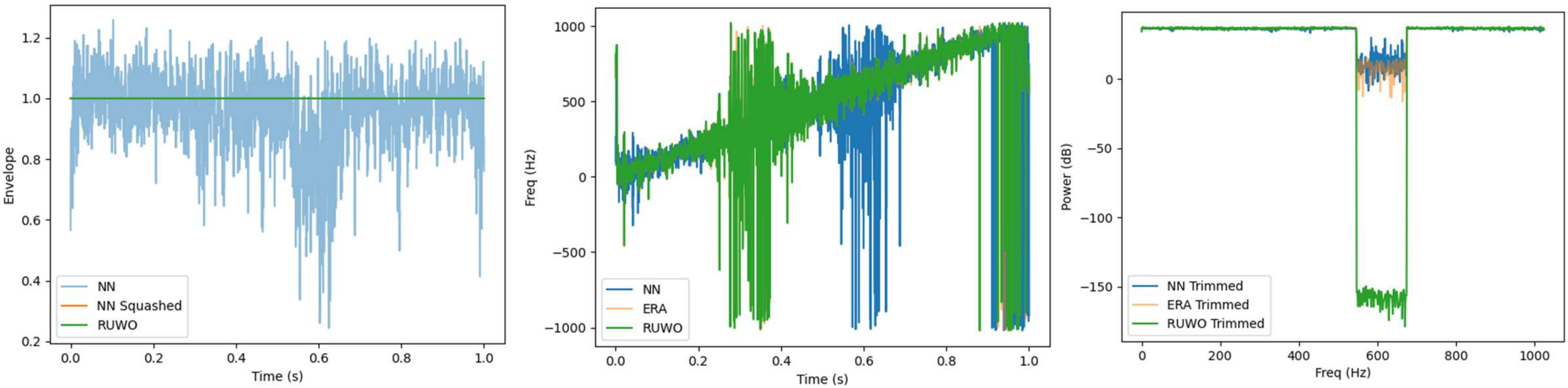
CPU / GPU Simulation

Algorithm	Cosine Similarity	Null Depth (dBm)
RUWO	1.0 ± 0.0	202.23 ± 0.0
ERA	0.9982 ± 0.0	31.89 ± 0.0
NN MSE	$0.9901 \pm 7.69 \times 10^{-5}$	28.54 ± 0.16
NN Custom Loss	$0.9789 \pm 9.53 \times 10^{-5}$	22.32 ± 0.13
NN Split	$0.9900 \pm 1.08 \times 10^{-4}$	29.75 ± 0.12

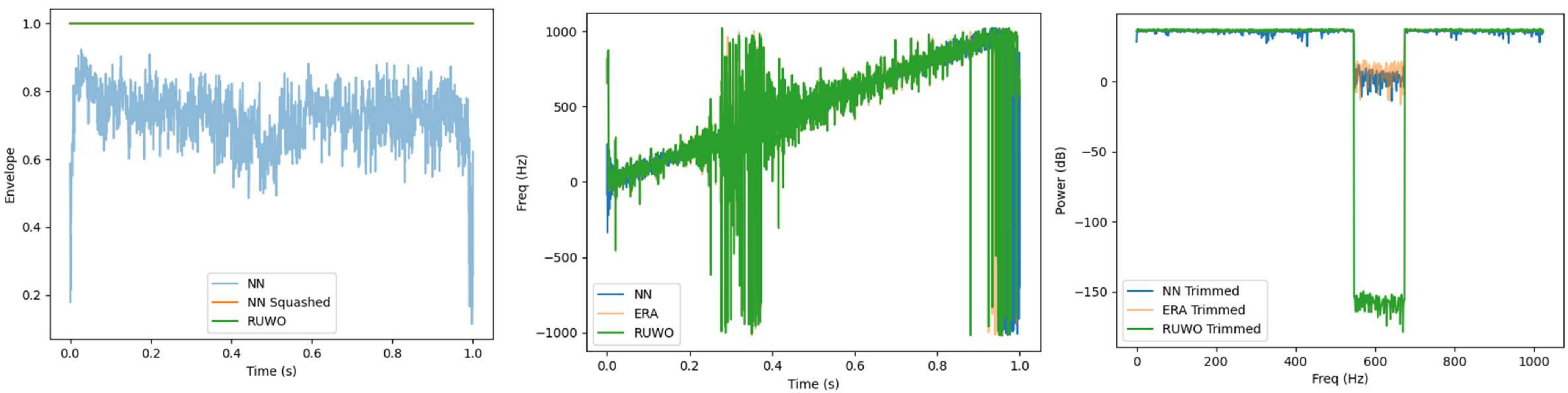
RFSoC FPGA Open-Air Trials

Algorithm	Null Depth (dBm)
RUWO	33.62 ± 0.041
ERA	37.13 ± 0.057
NN MSE	28.17 ± 0.213
NN Custom Loss	21.22 ± 0.138
NN Split	28.93 ± 0.285

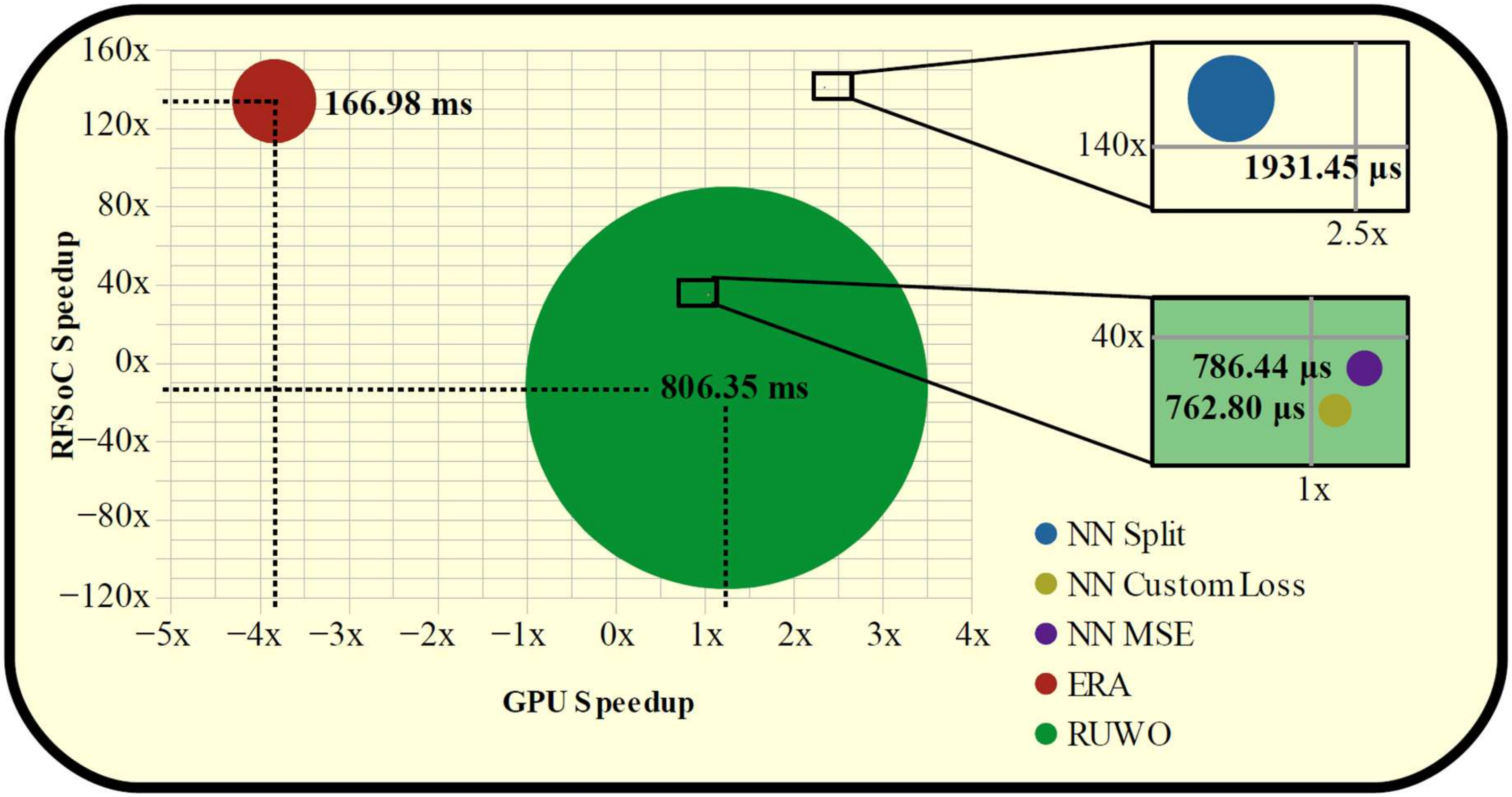
Custom Loss



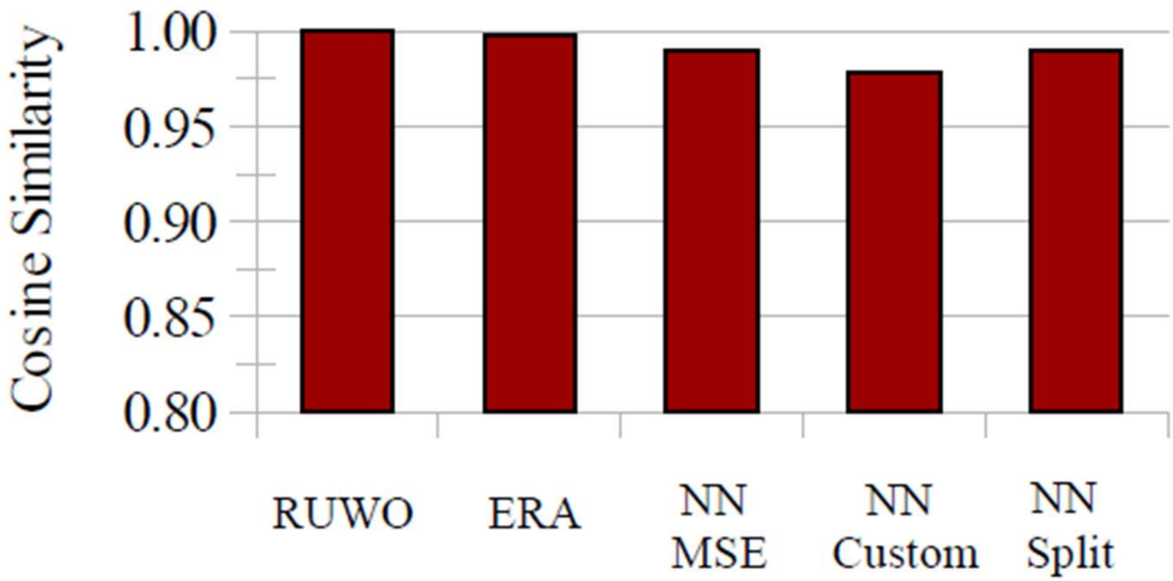
MSE



Algorithm	CPU Latency (µs)	GPU Latency (µs)	RFSoC Latency (µs)
RUWO	806,347.0 ± 11,860.82	649,581.0 ± 33,168.78	10,060,000.0 ± 999.0
ERA	166,982.0 ± 3465.06	641,441.0 ± 20,921.13	1246.0 ± 8.8
NN MSE	786.4 ± 5.01	747.71 ± 5.23	21.7 ± 0.0
NN Custom Loss	762.8 ± 5.04	735.68 ± 7.99	21.7 ± 0.0
NN Split	1931.5 ± 9.75	823.63 ± 6.76	13.7 ± 0.0



- Specifically target low power embedded devices (Raspberry Pi 3B)



Null Depth (dBm)	202.2	31.9	28.5	22.3	29.8

Algorithm	Dell r720 2x Intel E5-2670, NVIDIA GT 1030, 144GB RAM		Raspberry Pi 3B Broadcom BCM2837, 1GB RAM	
	Latency (ms)	Energy (J)	Latency (ms)	Energy (J)
RUWO	1064.98 ± 10.94	261.3 ± 6.5	453,965.43 ± 4131.61	1510.5 ± 14.8
ERA	185.47 ± 3.87	45.5 ± 1.4	1982.04 ± 29.27	6.5 ± 0.1
NN MSE	23.19 ± 1.86	**3.7 ± 0.3**	**230.98 ± 2.74**	**0.6 ± 0.01**
NN Tailored Loss Function	**20.72 ± 0.44**	**3.7 ± 0.1**	233.92 ± 3.16	**0.6 ± 0.01**
NN Tailored Network Architecture	23.35 ± 0.29	4.1 ± 0.6	250.90 ± 0.63	0.7 ± 0.01

Outline

- **Introduction**
  - Current Trends
  - Research Goals
- **Autowave**
  - Radar Waveform Design Overview
  - “Lean Neural Networks for Autonomous Radar Waveform Design”
  - “Lean Neural Networks for Real-time Embedded Spectral Notching Waveform Design”
- **SNN-GAN**
  - Spiking Neural Networks (SNNs) Overview
  - Generative Adversarial Networks (GANs) Overview
  - “Toward Robust Spiking Neural Networks”
  - “Dataset Augmentation for Robust Spiking Neural Networks”
- **Current & Future Work**

Spiking Neural Networks (SNNs)

- Biologically inspired 3rd generation neural networks
- Neurons communicate via discrete pulses over time
- Great for time-series data
- SNN processing consumes less power when realized on neuromorphic hardware such as Intel Loihi [15]

https://www.intel.com/content/www/us/en/newsroom/news/intel-unveils-neuromorphic-loihi-2-lava-software.html#gs.4ve63w

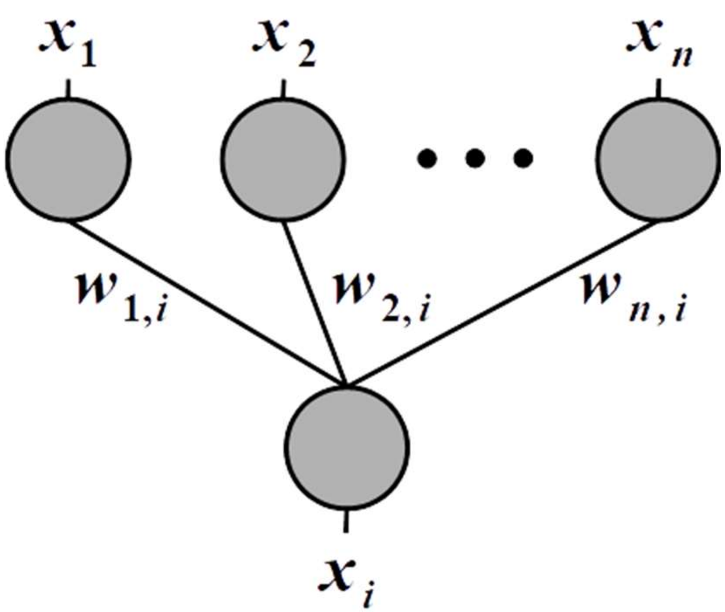
ANNs vs SNNs

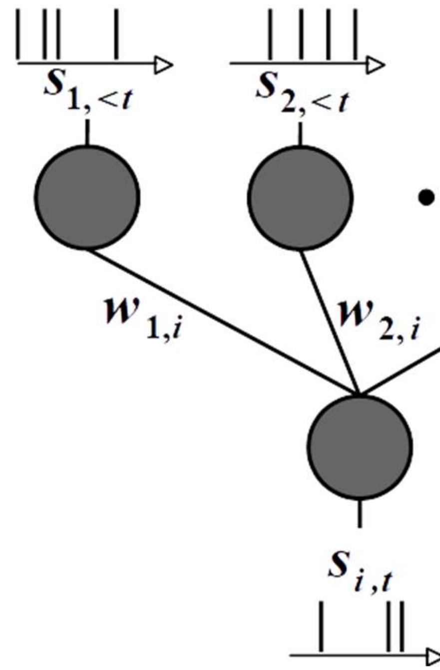
ANNs

- Operate on continuous values $x_1, x_2, \ldots, x_n$
- Information propagates instantaneously

SNNs

- Operate on discrete spike trains $S_1, S_2, \ldots, S_n$
- Must be run over a period of time





ANNs vs SNNs

$$E = L(y, \hat{y})$$

$$net_j = \sum_i w_{ij} x_i + b$$

$$o_j = \varphi(net_j)$$

$$\delta_j = \begin{cases} \frac{\partial L(y,o_j)}{\partial o_j} \frac{d\varphi(net_j)}{dnet_j} & j \text{ output} \\ \left(\sum_k w_{jk}\delta_k\right) \frac{d\varphi(net_j)}{dnet_j} & j \text{ hidden} \end{cases}$$

$$\Delta w_{ij} = -\eta o_j \delta_j$$

Input $s_i(t) = \sum_f \delta\left(t - t_i^{(f)}\right)$

$$a_i(t) = (\epsilon * s_i)(t) \quad \epsilon(t) = \frac{t}{\tau_s} exp\left(1 - \frac{t}{\tau_s}\right)\Theta(t)$$

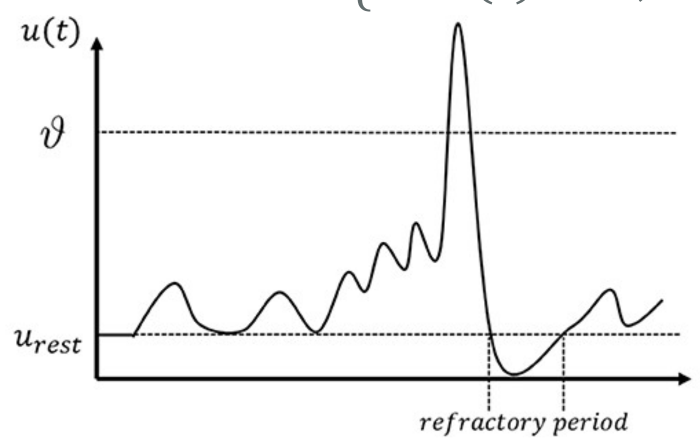
$$v_i(t) = (v * s)(t) \quad \mathrm{v}(t) = -2\vartheta exp\left(1 - \frac{t}{\tau_r}\right)\Theta(t)$$

$$u(t) = \sum_i w_i a_i(t) + v_i(t)$$

$$f_s(u) : u \to s$$

$$s(t) \coloneqq s(t) + \delta\left(t - t^{(f+1)}\right)$$

$$t^{(f+1)} = min\{t : u(t) = \vartheta, t > t^{(f)}\}$$



SNN Data

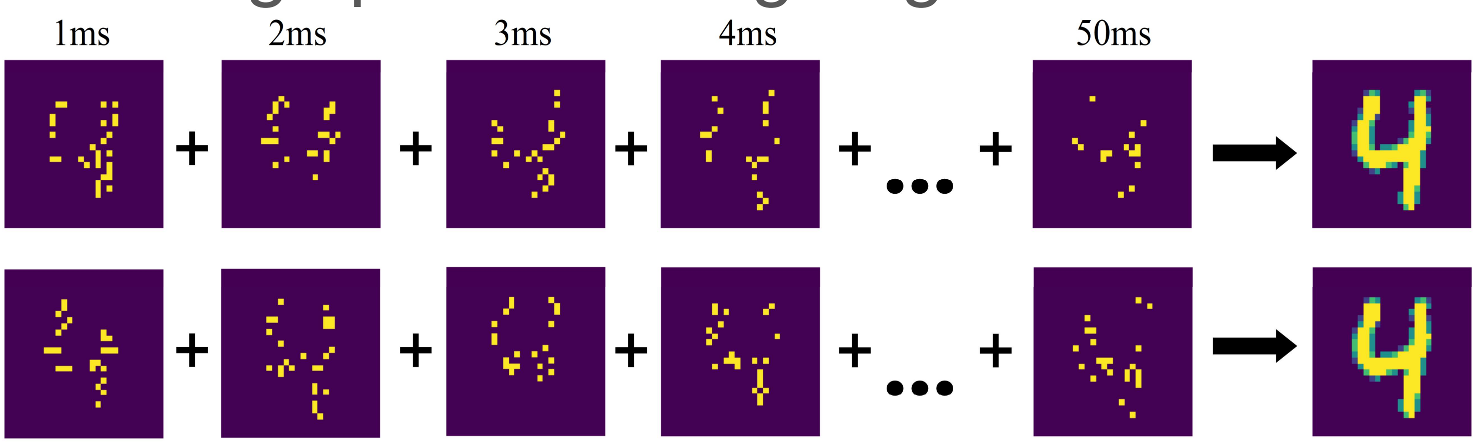
- SNNs operate on discrete spike trains
- Can be either generated from static data using integrate-and-fire (IF) neurons or captured directly using a Dynamic Vision Sensor (DVS) camera which produces event data:

https://inilabs.com/products/

$[\boldsymbol{x}\ \text{coordinate}, \boldsymbol{y}\ \text{coordinate}, \boldsymbol{t}\ \text{timestep}, \boldsymbol{p}\ \text{polarity of light} - \text{intensity change}]$

Spike Distribution Dependencies

- For a given static image, there are a copious number of valid spike trains which can be created/captured depending on IF neuron parameters, DVS camera settings, or lighting properties of the subject
- Surrogate gradient SNN training can fixate on the intervals of training spikes leading to generalization issues

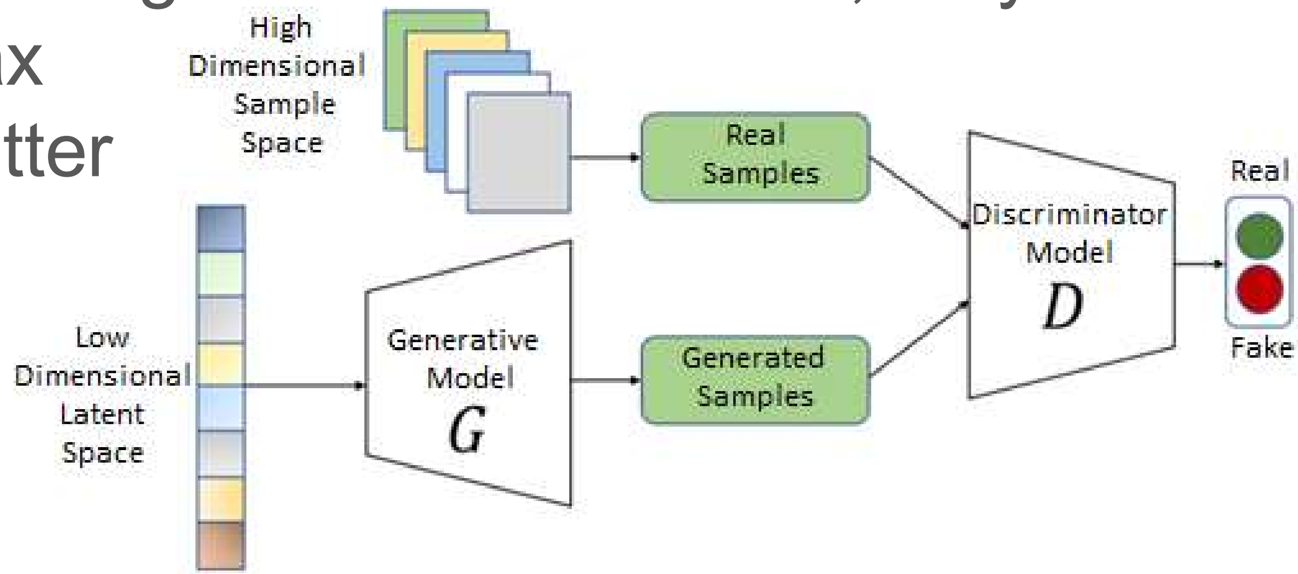


Our approach

- Using a spiking GAN, generate **valid** samples of **varying** spike distributions
- Augmented dataset provides additional robustness against samples different from the original training set
- Generated samples enrichen dataset without additional manual collection of data (and exponential growth for each different spike distribution)

Generative Adversarial Networks (GANs)

- Adversarial learning paradigm in which a generator model G synthesizes artificial samples, and a discriminator model D classifies samples as either real or fake
- G and D "compete" against each other i.e., they are playing a minimax game to each better themselves



Why GANs

- Most models are classifiers e.g., given an input $\boldsymbol{x}$, they predict a label $\boldsymbol{y}$ thus estimating $P(\boldsymbol{y}|\boldsymbol{x})$
- However, these models cannot estimate $P(\boldsymbol{x})$ and therefore cannot sample from $P(\boldsymbol{x})$ i.e., they cannot create new samples
- The combination of a traditional classifier model with a new generative model unlocks new functionality

Generator

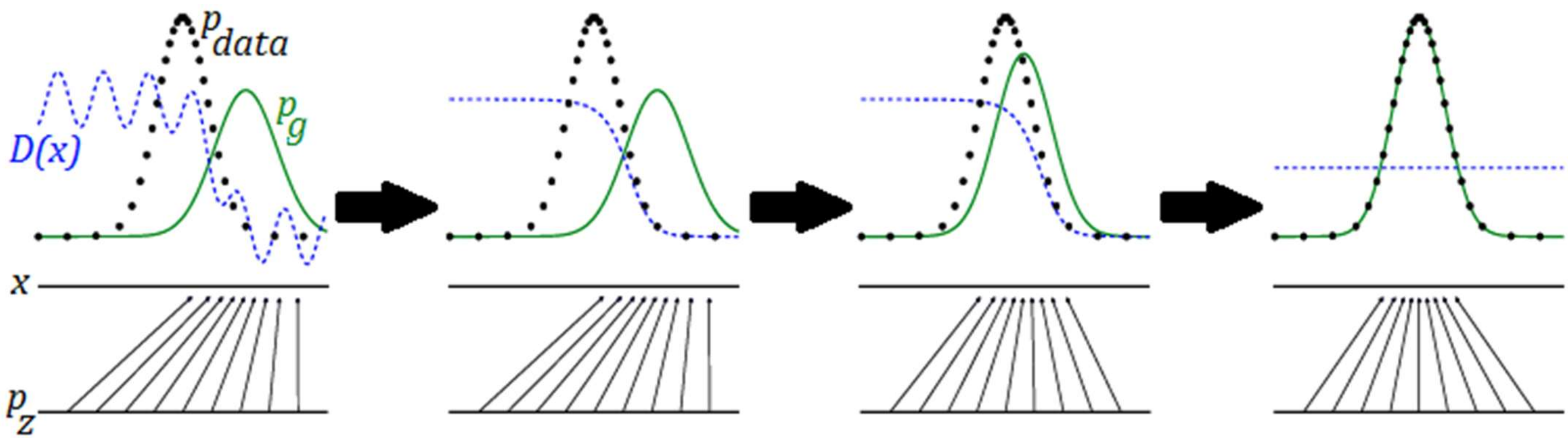
- The generator G, often a deep neural network, transforms an input noise vector into a realistic sample
  The goal of G is to learn a mapping from some noisy space p_z to p_g which approximates the real data distribution p_{data}
- Once sufficiently trained, $p_g \approx p_{data}$ e.g., G can generate arbitrary samples which appear to be real

Discriminator

- The discriminator D, often a deep neural network or Convolutional Neural Network (CNN) for images, classifies input samples as either real (coming from the real distribution) or fake (coming from G)
- The goal of D is to learn a mapping from an input space (containing potentially both real & fake samples) to [0, 1] where 0 asserts a sample is fake and 1 asserts a sample is real

GAN Training

- G and D are trained simultaneously in which D identifies areas in which it can more easily identify fake samples
- These areas then become the focus for where G updates its weights
- Given sufficient capacity, G and D converge to where $p_g \approx p_{data}$ and $D(x) = 0.5$ for all input
- $\min_G \max_D V(D,G) = \mathbb{E}_{x \sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}\left[\log\left(1 - D\left(G(z)\right)\right)\right]$



GAN Training (cont'd)

- For fixed G, training D to the optimal classifier $D_G^*(x) = \frac{p_{data}(x)}{p_{data}(x)+p_g(x)}$ is computationally expensive. Instead, training of G and D is **alternated** while keeping D near optimal to give G better gradients [16]
- **Notice:** G is never trained directly on the real data, it only learns from the gradient from D flowing backward. This prevents "overfitting" and instead allows G to branch beyond the real data

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**
  for k steps **do**
   • Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \dots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
   • Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
   • Update the discriminator by ascending its stochastic gradient:
$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)\right].$$
  end for
  • Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \dots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  • Update the generator by descending its stochastic gradient:
$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$
end for
The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

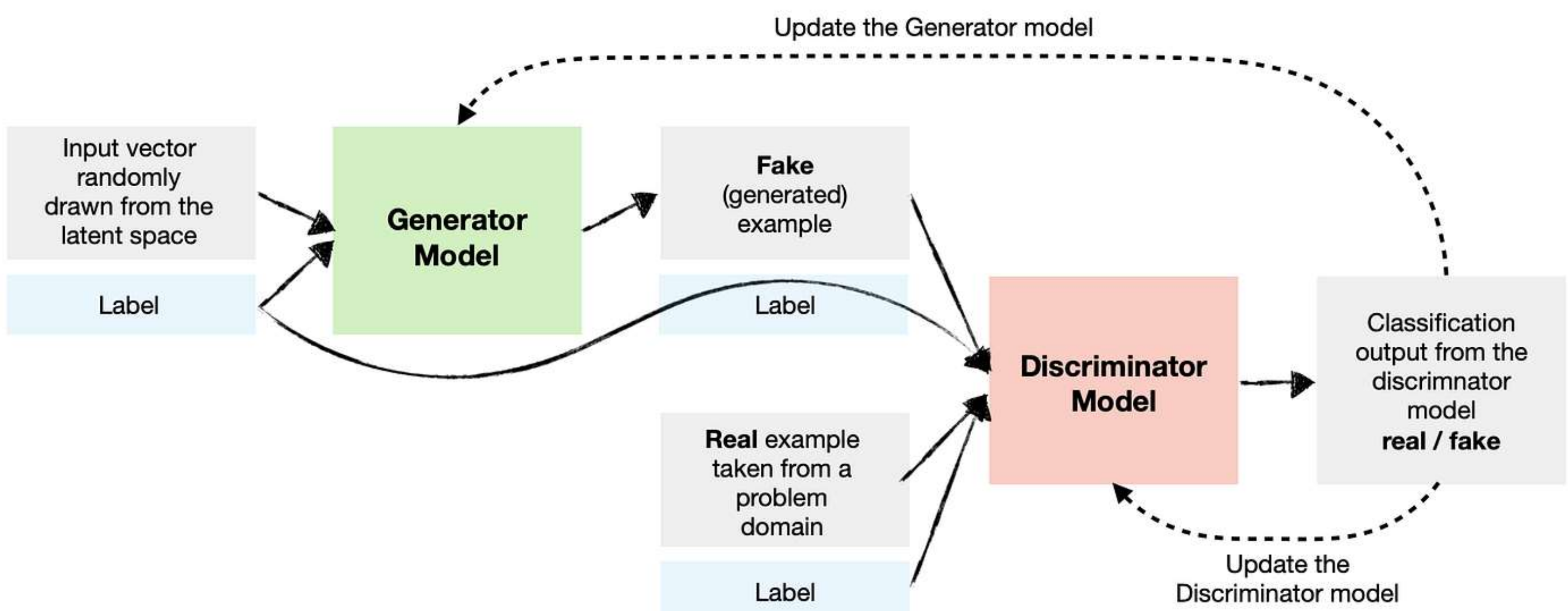
GAN Training Issues

- “Helvetica Scenario” aka Mode Collapse
  - Situation in which G fails to generalize its outputs and instead produces all similar samples
  - **Often caused by G learning too fast relative to D**, and so G begins to favor generating a subset of the data but is unable to escape the local minima once D continues learning
- Vanishing Gradient
  - **Often caused by D learning too fast relative to G** in which D can perfectly distinguish real/fake samples leaving G unable to “catch up” in its generating ability leading to stagnation in G

Balancing G and D is difficult!

Extensions

- Conditional GAN (CGAN)
  - Allows for specifying which class of data to be generated by attaching a label to the latent input vector for G



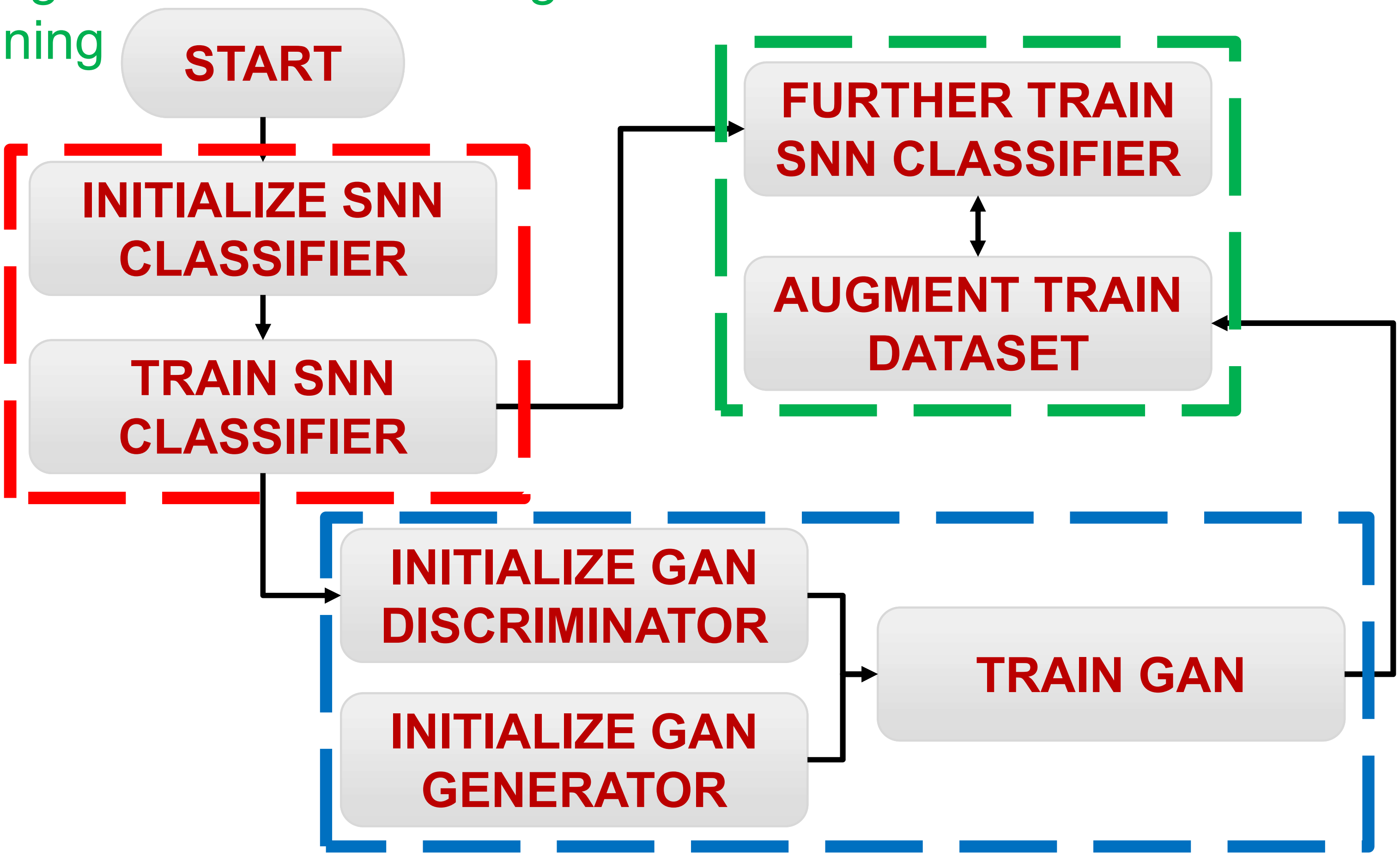
Our approach (rep.)

- Using a spiking GAN, generate **valid** samples of **varying** spike distributions
- Augmented dataset provides additional robustness against samples different from the original training set
- Generated samples enrichen dataset without additional manual collection of data (and exponential growth for each different spike distribution)

(1) SNN classifier trained to convergence

(2) GAN trained using classifier weights to seed discriminator

(3) Trained GAN generator used to augment train dataset for further classifier training



Our approach (cont'd)

- During augmentation, samples are generated on an as-needed basis determined by the relative class performances
- Difficulty of correct classification is not uniform across all classes of data → disproportionate number of samples can achieve same overall accuracy

Our approach (cont'd)

- Three schemes used to determine the number of additional samples needed for the next iteration:

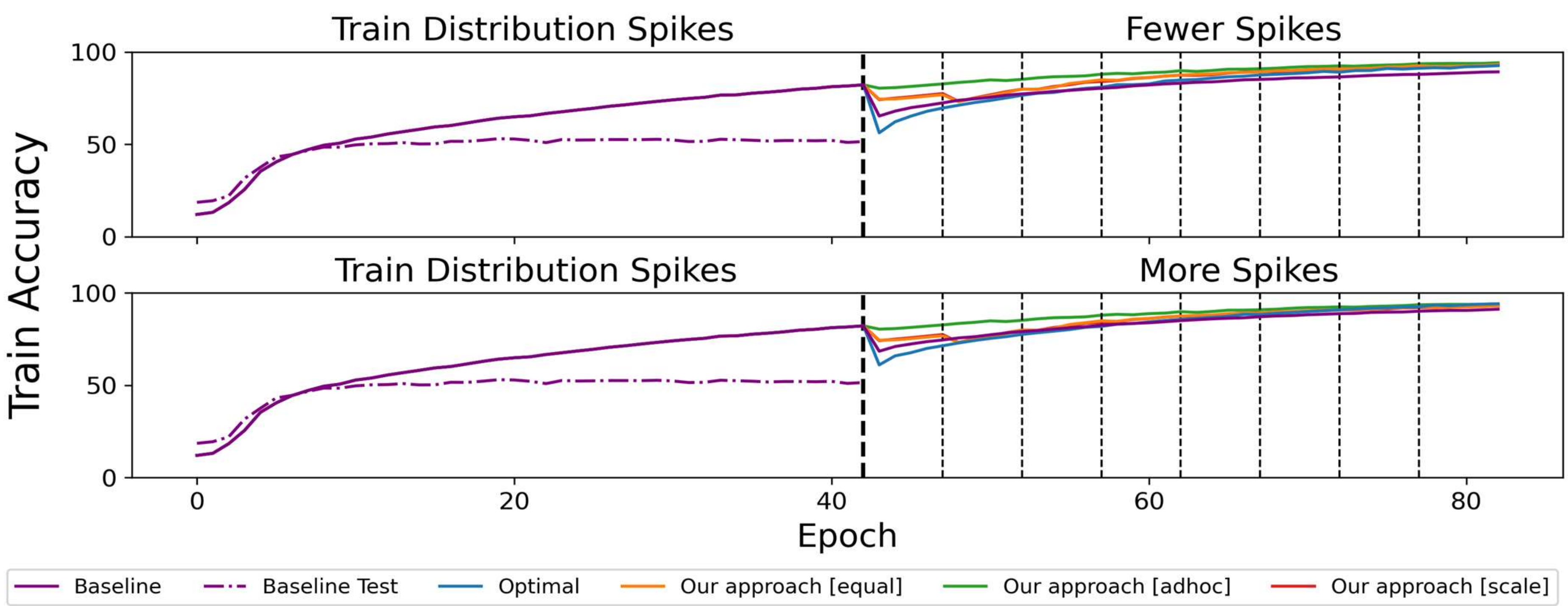
1) **equal**: same number of samples across all classes
2) **adhoc**: only samples from the 3 worst performing classes added
3) **scale**: number of samples added correlated to relative performance of each class

Setup

- SLAYER [17] SNN training platform used
- CIFAR-10 training spike trains generated from $X \sim U(100, 200)$ firing rate distribution using LIF (leaky integrate-and-fire neuron) in Nengo [18] simulator
- Models evaluated on **fewer** spikes and **more** spikes distributions → half ($X \sim U(50, 100)$) and double ($X \sim U(200, 400)$) the number of spikes compared to training distribution

Training

- Our models quickly responded to the changing spike distribution



Testing

- All models perform worse as samples drifted further from the training distribution
- Our models outperformed baseline classifier by an average **1.80%** and had an average **1.02%** lesser reduction in accuracy

Model	Testing Spike Distribution		
	Fewer Spikes	Train Dist. Spikes	More Spikes
Baseline	37.76 ± 0.34	52.67 ± 0.31	42.73 ± 0.52
Our approach [equal]	39.09 ± 0.25	**54.57 ± 0.27**	44.07 ± 0.52
Our approach [adhoc]	**39.33 ± 0.28**	54.25 ± 0.30	**44.67 ± 0.52**
Our approach [scale]	38.52 ± 0.39	54.05 ± 0.32	43.51 ± 0.29

Outcomes

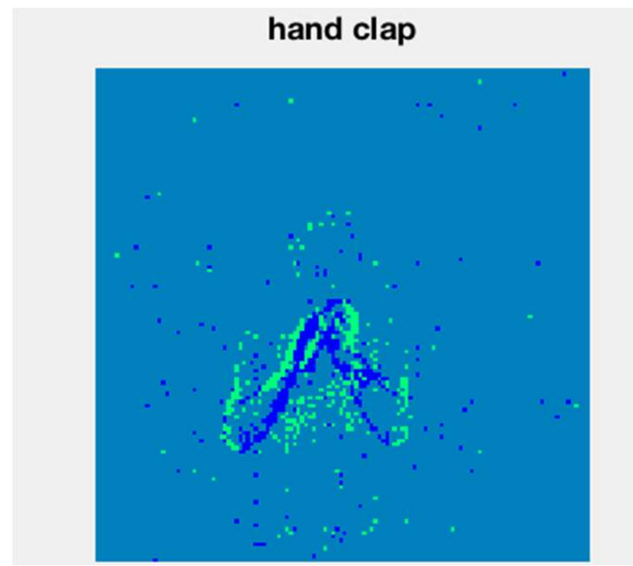
- Conventional SNN training methods do **not ensure generalization capabilities** for temporal data
- Our results show **improvements in model robustness** against dissimilar samples from the training data

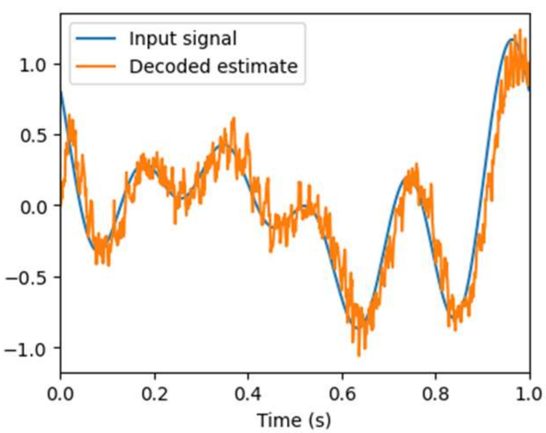
Outline

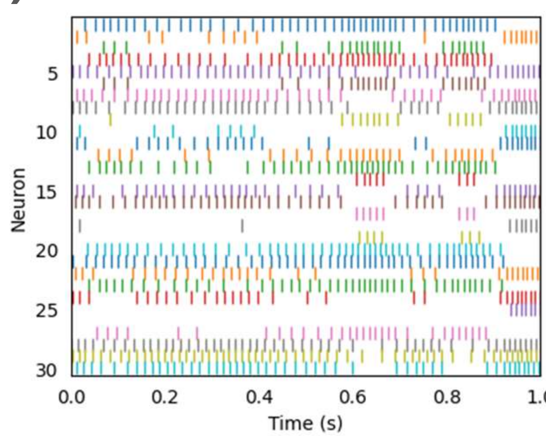
- **Introduction**
  - Current Trends
  - Research Goals
- **Autowave**
  - Radar Waveform Design Overview
  - “Lean Neural Networks for Autonomous Radar Waveform Design”
  - “Lean Neural Networks for Real-time Embedded Spectral Notching Waveform Design”
- **SNN-GAN**
  - Spiking Neural Networks (SNNs) Overview
  - Generative Adversarial Networks (GANs) Overview
  - “Toward Robust Spiking Neural Networks”
  - “Dataset Augmentation for Robust Spiking Neural Networks”
- **Current & Future Work**

"Generative Datasets for Training Spiking Neural Networks"

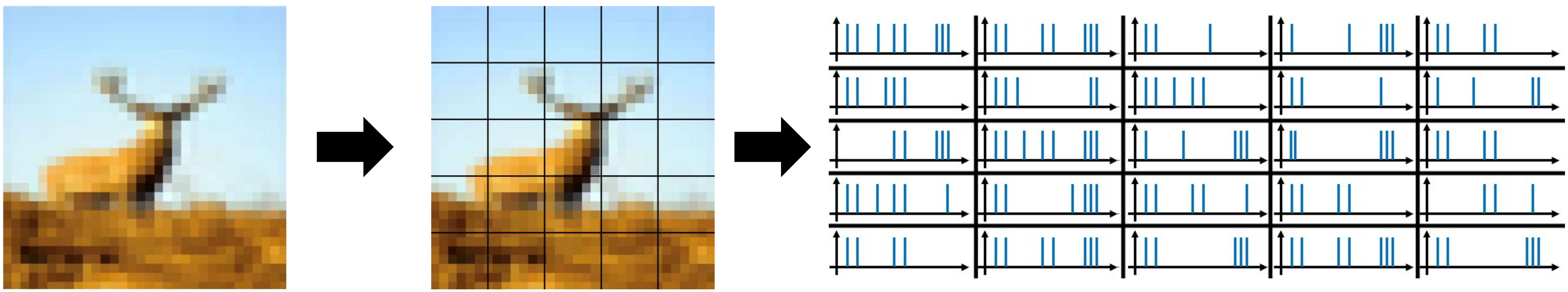
- Expand datasets (N-MNSIT, CIFAR10-DVS, DVSGesture)
- Expand frameworks (Nengo Neural Engineering Framework)



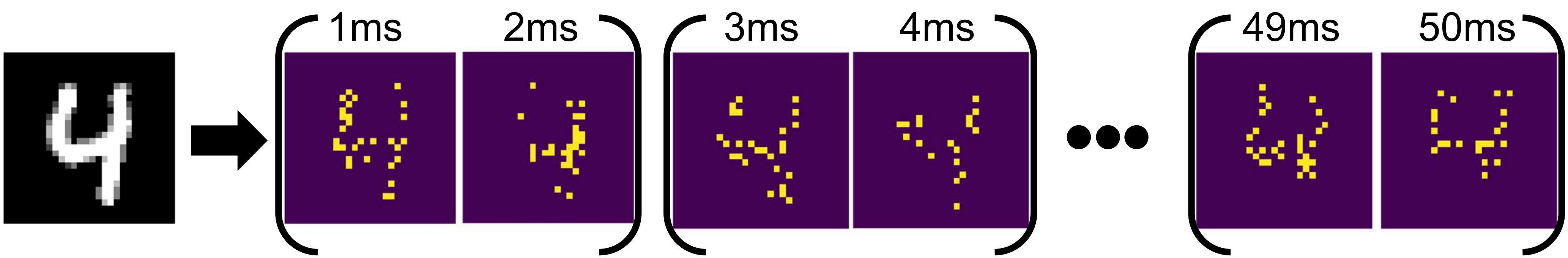




Spatial Splitting



Temporal Splitting



- Why do NN models (including SNNs) need so much data?
- NN training has many pitfalls [19, 20], better algorithm must exist (Foutse Khomh)
- **MAJOR dependency on training data**
  - Model performs poorly
    ↓
    Test set representative?

$$E = L(y, \hat{y})$$

$$net_j = \sum_i w_{ij} x_i + b$$

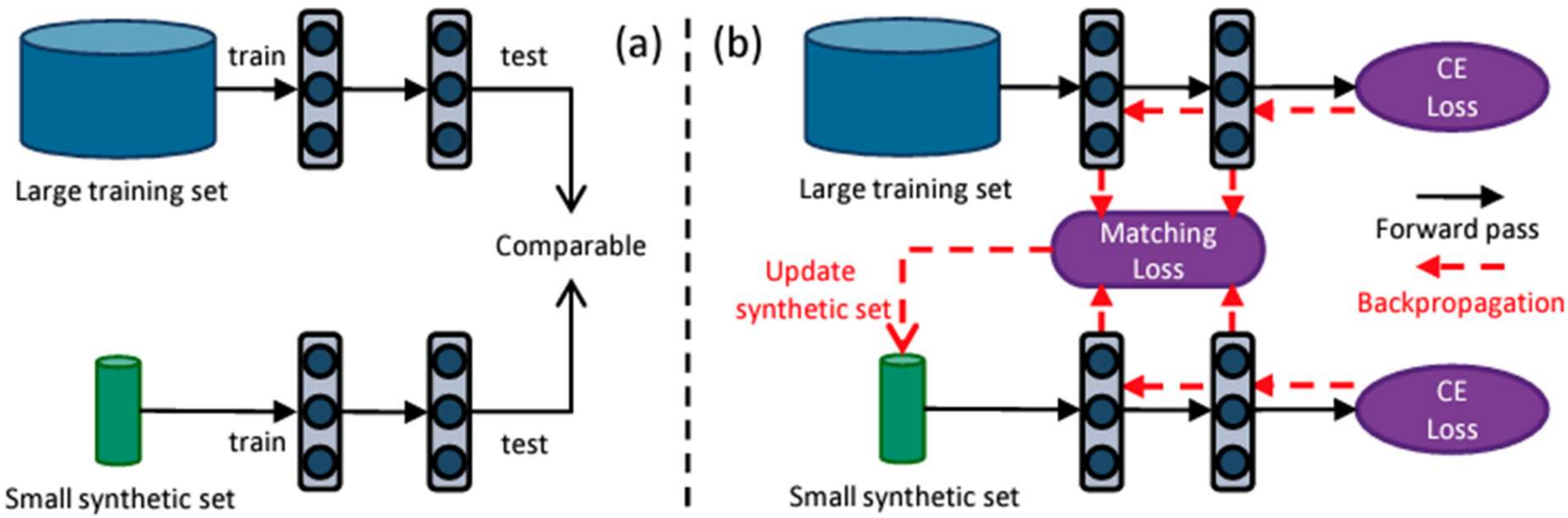
$$o_j = \varphi(net_j)$$

$$\delta_j = \begin{cases} \frac{\partial L(y, o_j)}{\partial o_j} \frac{d\varphi(net_j)}{dnet_j} & j \text{ output} \\ \left(\sum_k w_{jk} \delta_k\right) \frac{d\varphi(net_j)}{dnet_j} & j \text{ hidden} \end{cases}$$

$$\Delta w_{ij} = -\eta o_j \delta_j$$

Dataset condensation [21]

- Rather than generating realistic samples, generate "super" samples which don't necessarily look real, but produce identical gradient updates faster



Future Work

What about for SNNs?

- Can our GAN approach be combined with dataset condensation to generate “super” samples?
- Can this approach be applied to neuromorphic computing to compress the already larger data-space?
- **Can this approach condense different spike distributions?**

THE OHIO STATE UNIVERSITY

Questions?

References

[AB1] Baietto, A.; Boubin, J.; Farr, P.; Bihl, T.J.; Jones, A.M.; Stewart, C. Lean Neural Networks for Autonomous Radar Waveform Design. Sensors 2022, 22, 1317. https://doi.org/10.3390/s22041317

[AB2] A. Baietto, J. Boubin, P. Farr and T. J. Bihl, "Lean Neural Networks for Real-time Embedded Spectral Notching Waveform Design," 2022 IEEE 31st International Symposium on Industrial Electronics (ISIE), Anchorage, AK, USA, 2022, pp. 1121-1126, doi: 10.1109/ISIE51582.2022.9831772.

[AB3] Anthony Baietto, Christopher Stewart, Trevor J. Bihl. Toward Robust Spiking Neural Networks. Poster presented at: International Conference on Neuromorphic Systems, Santa Fe, NM, 2023

[AB4] Anthony Baietto, Christopher Stewart, Trevor J. Bihl. “Dataset Augmentation for Robust Spiking Neural Networks”. IEEE International Conference on Autonomic Computing and Self-Organizing Systems, Toronto, Canada, 2023

References

[1] A. Hemeida, S. Hassan, M. Al-Attar, S. Alkhalaf, M. Mahmoud, T. Senjyu, A. El-Din, and A. Alsayyari, “Nature-inspired algorithms for feed-forward neural network classifiers: A survey of one decade of research,” Ain Shams Engineering Journal, vol. 11, 01 2020.

[2] G. Cybenko, “Approximation by superpositions of a sigmoidal function,” Mathematics of Control, Signals, and Systems (MCSS), vol. 2, no. 4, pp. 303–314, Dec. 1989. [Online]. Available: http://dx.doi.org/10.1007/BF02551274

[3] K. Hornik, M. Stinchcombe, and H. White, “Multilayer feedforward networks are universal approximators,” Neural Networks, vol. 2, no. 5, pp. 359–366, 1989. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0893608089900208

[4] K. Hornik, “Approximation capabilities of multilayer feedforward networks,” Neural Networks, vol. 4, no. 2, pp. 251–257, 1991. [Online]. Available: https://www.sciencedirect.com/science/article/pii/089360809190009T

[5] P. Villalobos and A. Ho, “Trends in training dataset sizes,” 2022, accessed: 2023-10-3. [Online]. Available: https://epochai.org/blog/trendsin-training-dataset-sizes

[6] L. Bernstein, A. Sludds, R. Hamerly, V. Sze, J. Emer, and D. Englund, “Freely scalable and reconfigurable optical hardware for deep learning,” Scientific Reports, vol. 11, 02 2021.

[7] Gerchberg, R.W. (1972) A Practical Algorithm for the Determination of Phase from Image and Diffraction Plane Pictures. Optik, 35, 237-246.

References

[8] T. Higgins, T. Webster, and A. K. Shackelford, “Mitigating interference via spatial and spectra nulling,” IET Radar, Sonar & Navigation, vol. 8, no. 2, pp. 84–93, 2014. [Online]. Available: https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/iet-rsn.2013.0194

[9] Liu, G., et al.: A GPU-based real-time processing system for frequency division multiple-input-multiple-output radar. IET Radar Sonar Navig. 17(10), 1524–1537 (2023). https://doi.org/10.1049/rsn2.12439

[10] Xia, Y., Ma, Z., Huang, Z.: Radar waveform recognition based on a two-stream convolutional network and software defined radio. IET Radar Sonar Navig. 16(5), 837–851 (2022). https://doi.org/10.1049/rsn2.12224

[11] R. Michev, Y. Shu, D. Werbunat, J. Hasch and C. Waldschmidt, "Adaptive Compensation of Hardware Impairments in Digitally Modulated Radars Using ML-Based Behavioral Models," in IEEE Transactions on Microwave Theory and Techniques, doi: 10.1109/TMTT.2023.3285438.

[12] J. Boubin, A. M. Jones, and T. Bihl, “Neurowav: Toward real-time waveform design for vanets using neural networks,” in 2019 IEEE Vehicular Networking Conference (VNC), 2019, pp. 1–4.

[13] P. John-Baptiste, G. E. Smith, A. M. Jones, and T. Bihl, “Rapid waveform design through machine learning,” in 2019 IEEE 8th International Workshop on Computational Advances in Multi-Sensor Adaptive Processing (CAMSAP), 2019, pp. 659–663.

[14] P. Farr, A. M. Jones, T. Bihl, J. Boubin, and A. DeMange, “Waveform design implemented on neuromorphic hardware,” in 2020 IEEE International Radar Conference (RADAR), 2020, pp. 934–939.

References

[15] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, Y. Liao, C.-K. Lin, A. Lines, R. Liu, D. Mathaikutty, S. McCoy, A. Paul, J. Tse, G. Venkataramanan, Y.-H. Weng, A. Wild, Y. Yang, and H. Wang, "Loihi: A neuromorphic manycore processor with on-chip learning," IEEE Micro, vol. 38, no. 1, pp. 82–99, 2018.

[16] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, & Yoshua Bengio. (2014). Generative Adversarial Networks.

[17] S. B. Shrestha and G. Orchard, "Slayer: Spike layer error reassignment in time," 2018. [Online]. Available: https://arxiv.org/abs/1810.08646

[18] Bekolay, T., Bergstra, J., Hunsberger, E., DeWolf, T., Stewart, T., Rasmussen, D., Choo, X., Voelker, A., & Eliasmith, C. (2014). Nengo: a Python tool for building large-scale functional brain models. Frontiers in Neuroinformatics, 7(48), 1–13.

[19] Houssem Ben Braiek, & Foutse khomh. (2019). DeepEvolution: A Search-Based Testing Approach for Deep Neural Networks.

[20] Houssem Ben Braiek, & Foutse Khomh. (2022). Testing Feedforward Neural Networks Training Programs.

[21] Bo Zhao, Konda Reddy Mopuri, & Hakan Bilen. (2021). Dataset Condensation with Gradient Matching.