Physics Aware Training

Jyotirmaya Shivottam

S-Lab
Aug 31, 2022

Article | Open Access | Published: 26 January 2022

Deep physical neural networks trained with backpropagation

Logan G. Wright, Tatsuhiro Onodera, Martin M. Stein, Tianyu Wang, Darren T. Schachter, Zoey Hu & Peter L. McMahon

Nature **601**, 549–555 (2022) | Cite this article

Outline

- The *Why?*
- The *What?*
- The *How?*
- Architecture
- Examples
- Design Considerations
- Limitations
- Potential Applications
- *References*

The *Why?*

- Nvidia & Amazon estimate that inferencing takes up to 80 - 90% of the energy costs associated with ML.
- *Deep-learning accelerators*, e.g. Tensor cores with fused multiply-add $\rightarrow$ Built around exact mathematical isomorphisms to increase training speed $\longrightarrow$ Does not always result in energy savings during inference.
- *Quantization-Aware Training* $\rightarrow$ Uses low precision models during inference leading to faster and energy-efficient models.
- *Physical Reservoir Computing* (PRC) $\rightarrow$ Black-box physical systems connected to a trainable output layer.
- ❗ **Issue**: None of these allow tunable / trainable, hierarchical computation.

The *What?*

- **Problem**: How to design machines that can be trained like current DNNs and can perform the inference task energy-efficiently?
- **Intuition**: Hard to simulate physical systems digitally $\rightarrow$ Why not use physical systems to perform expensive computations?
- **Concept**: Train parameters of physical system so that the system acts as layers of a *physical* NN.
- **Challenge**: Making an arbitrary physical system behave as a NN.

The *What?* — Intuition

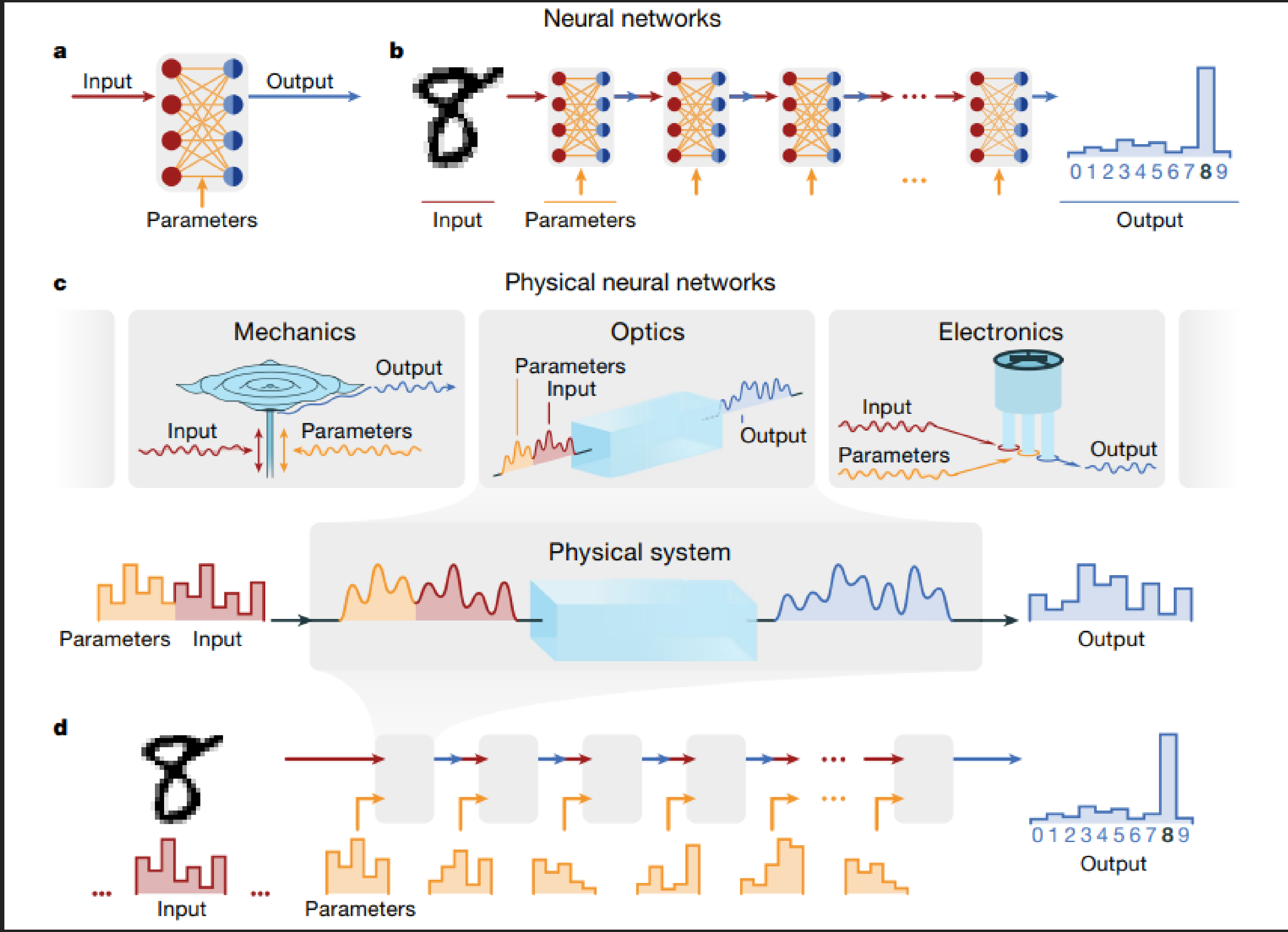
- Physical systems essentially map some input to some output, generally continuous.
- We also want to use auto-differentiation.
- Treat a physical system as a function, f_p:

$$\mathrm{y} = f_p(\mathrm{x}, \theta)$$

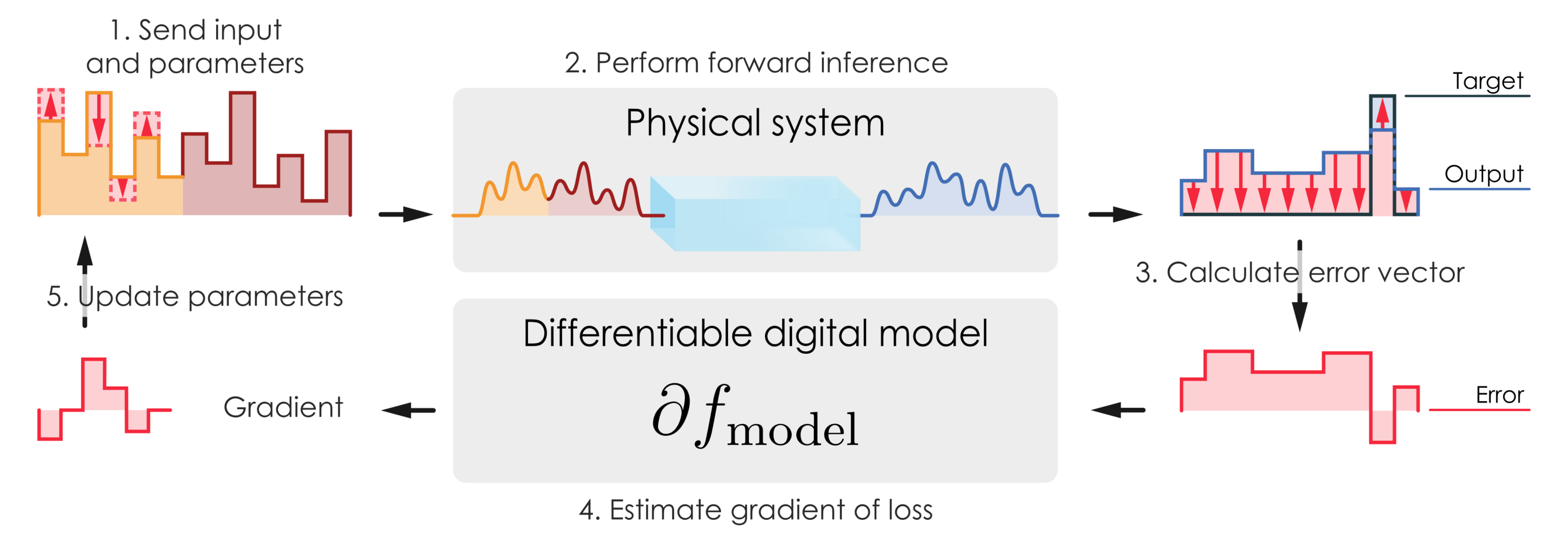
- ‼ **Issue**: f_p cannot be auto-differentiated. Finite difference is an option, but it's inefficient and noisy.
- **Solution**: Use a differentiable digital model, f_m (only to approx. backward pass) $\longrightarrow$ Generalization of Quantization-Aware Training.

The *How?*

- Usual NN ⟶
- Some physical systems ⟶
- *Physics-Aware Training* ⟶
- ⚠️ It is a gradient-descent algorithm.



The *How?*



1. Send input
and parameters
2. Perform forward inference
Physical system
Target
Output
3. Calculate error vector
5. Update parameters
Differentiable digital model
$\partial f_{\text{model}}$
Gradient
Error
4. Estimate gradient of loss

The *How?* — Differentiable Digital Models

- How to plug in physical systems?
  - Mathematical models
  - Black-box models ✅
  - Physics-Informed NN
- PNNs are inherently robust to noise (*Go to Aside*), since the physical system's output is used to calculate the errors.
- Updates are made using gradients calculated using the digital model.
- ⚠️ No digital activation functions are needed.
- ⚠️ Output layers are optional as well, since the physical system's response *is* the output.
- Physical system can be reused multiple times to form a multi-layer network.

The *How?* — Full Training Loop

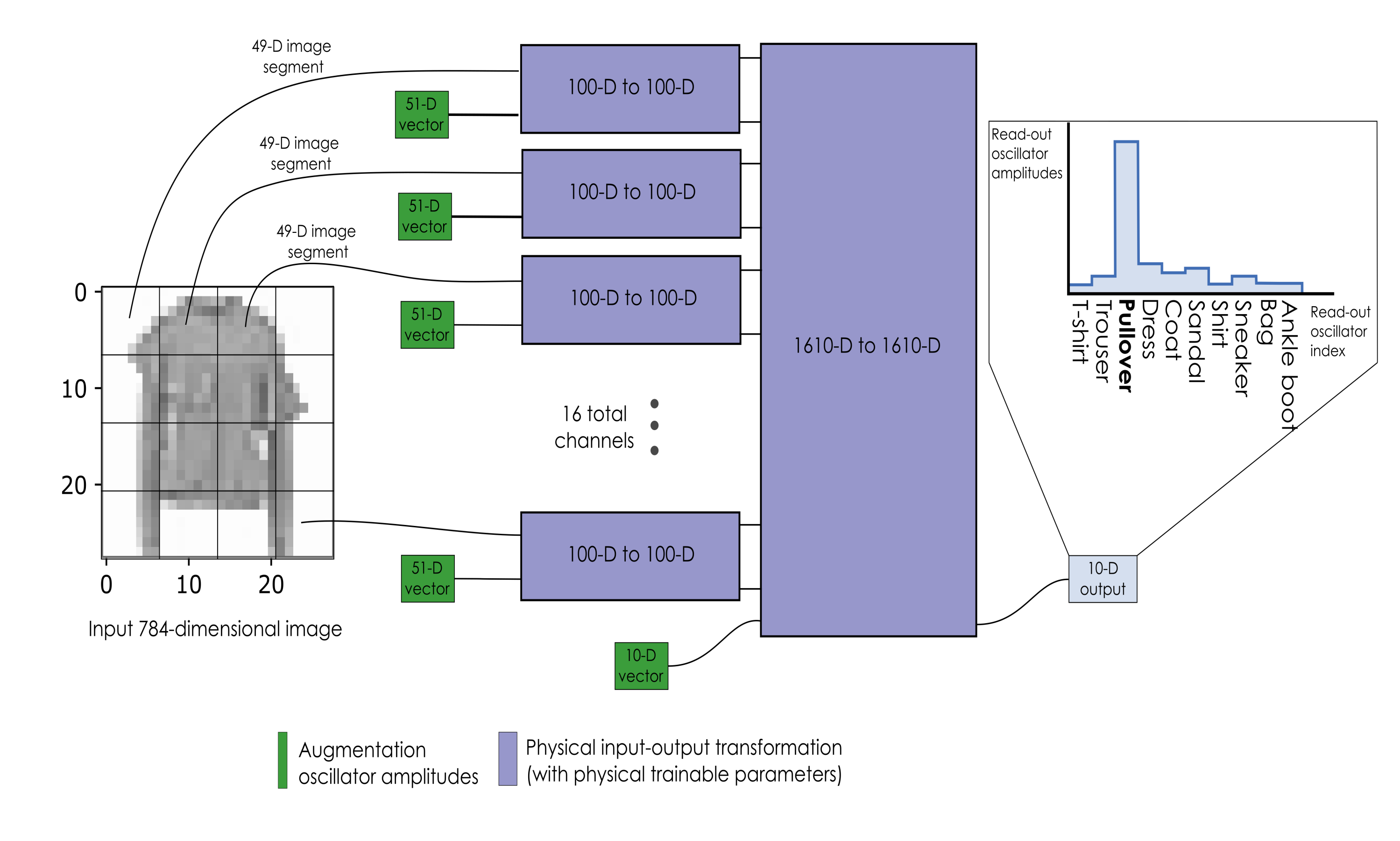
- Perform forward pass: $x^{l+1} = y^l = f_p(x^l, \theta^l)$
- Compute exact error vector: $g_y^N = \frac{\partial L}{\partial y^N} = \frac{\partial \mathcal{L}}{\partial y^N}(y^N, y_{\text{target}})$
- Perform backward pass:

$$g_{y^{l-1}} = \left[\frac{\partial f_m}{\partial x}(x^l, \theta^l)\right]^T g_{y^l}$$

$$g_{\theta^{l-1}} = \left[\frac{\partial f_m}{\partial \theta}(x^l, \theta^l)\right]^T g_{y^l}$$

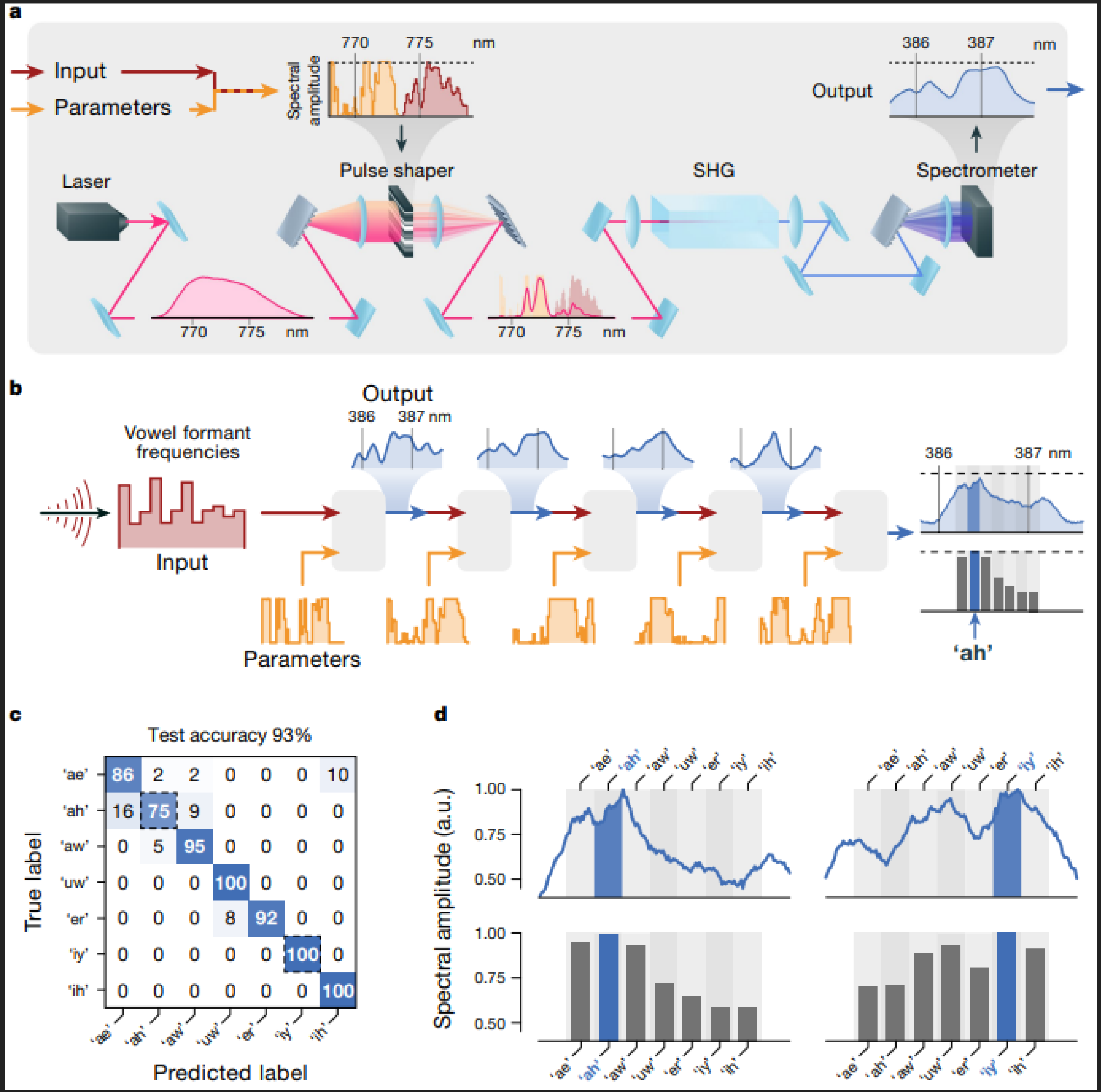
- Update parameters: $\theta^l \to \theta^l - \eta \frac{1}{N_{\text{data}}} \sum_k g_{\theta^l}^{(k)}$

Architecture with Oscillators on FashionMNIST



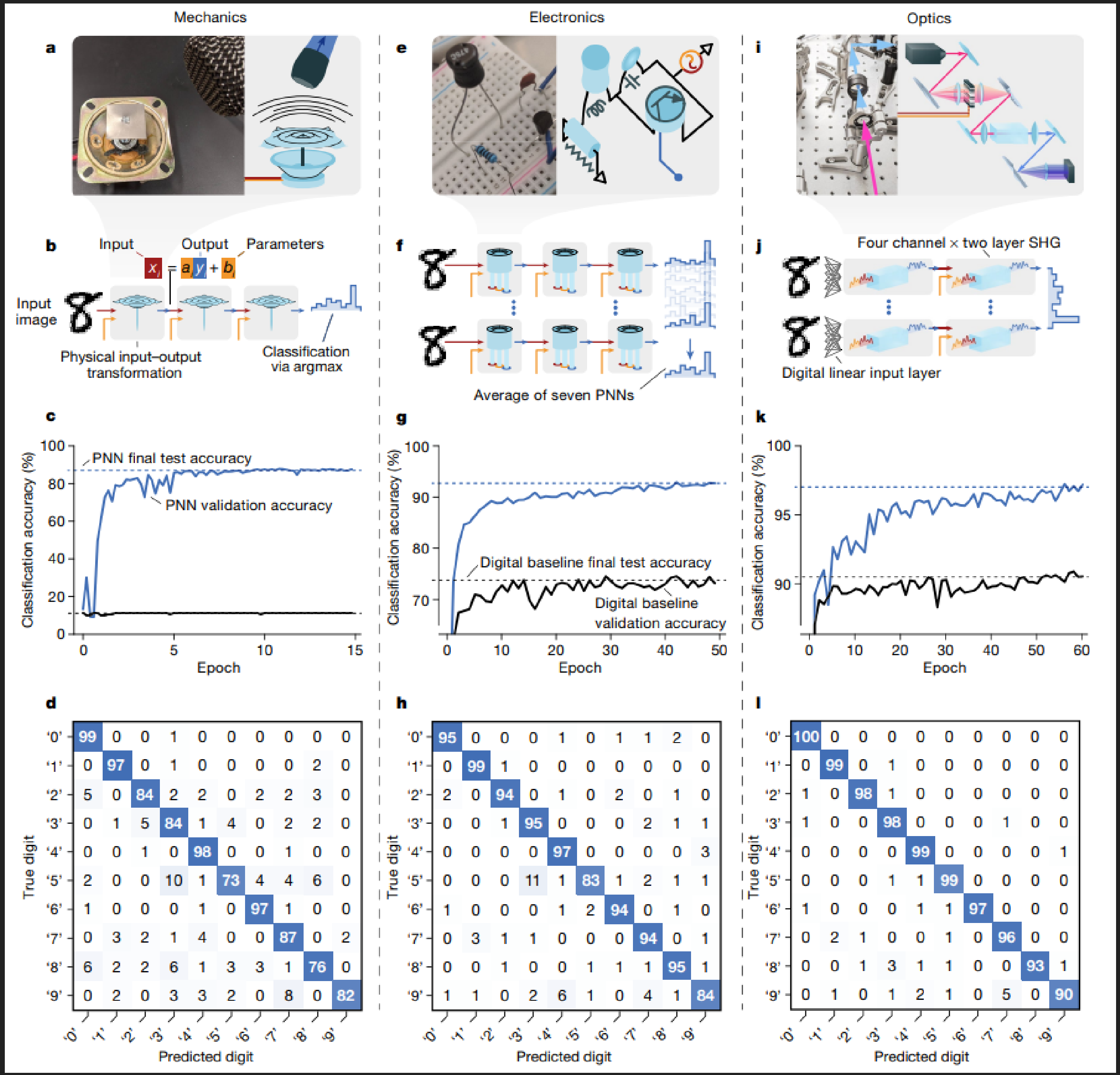
Examples

- Vowel classification using Second Harmonic Generation
- **Note**: Has no intuitive physical relation with vowel classification
- ⚠️ *Generality*



Examples

- Physical systems ↔ DNNs $\implies$ learned physical algorithms might be explainable by analyzing the network of DNNs.
- ⚠️ *Explainability*

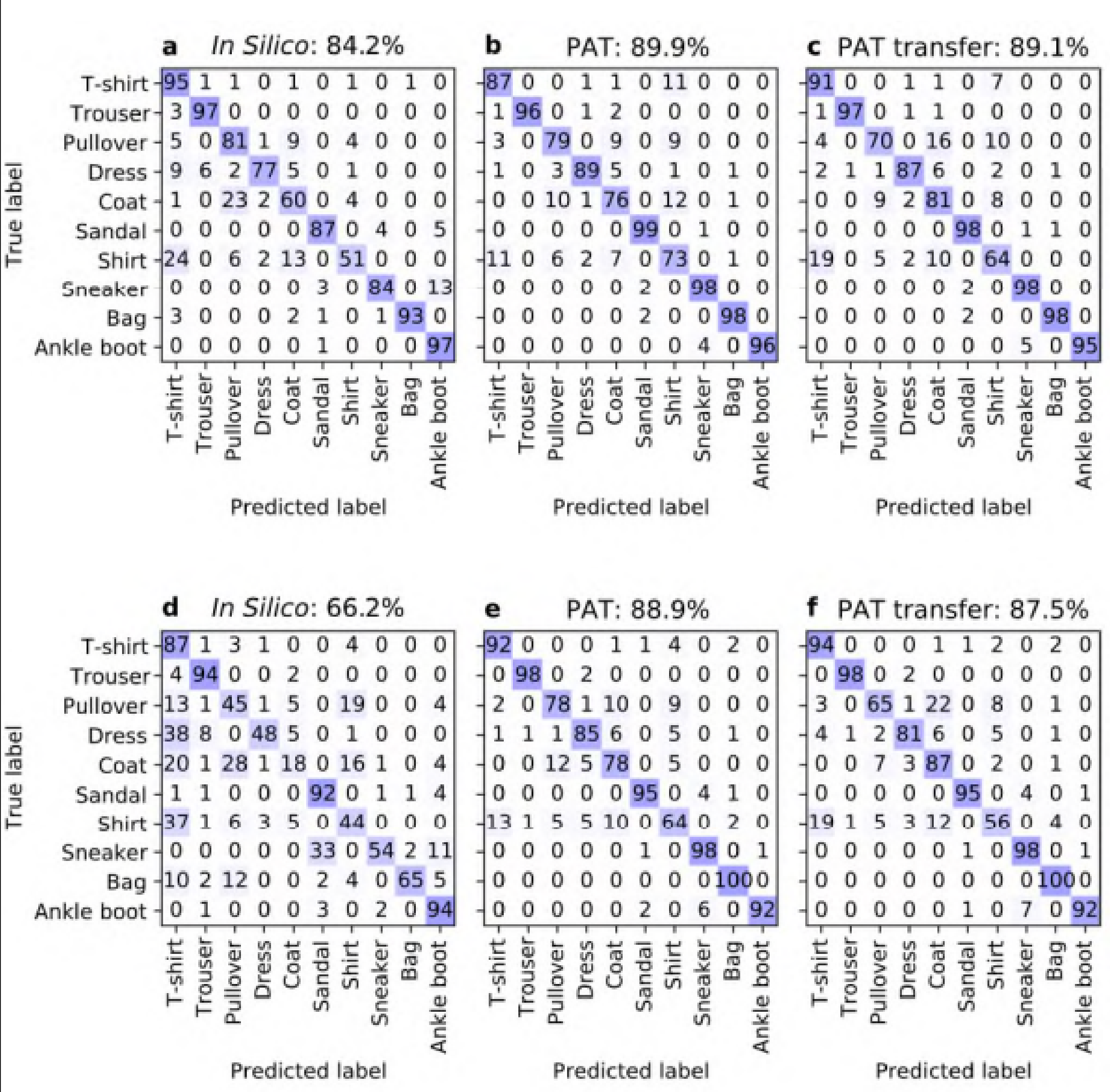


Design Considerations

- Any system that has a way to *input* data and *read out* data, and is auto-differentiable, is a potential candidate.
- **Guiding Principle**: *Controllability*.
- Hard to simulate digitally $\rightarrow$ instrinsic complexity $\rightarrow$ natural parallelism / high bandwidth.
- "*More parameters* $\implies$ *better results*" holds here as well.
- Some non-linearity is present. This might make the digital model more complex, but should produce better results.
- Neural architecture search and hyperparameter optimization are necessary.
- A good discussion is given in Supplementary information.

Aside

- Transferability & Noise
  - (a-c) CM for 20% mismatch & 6% variation
  - (d-f) CM for 30% mismatch & 9% variation



Limitations

- PNNs need a possibly non-exact differentiable digital model to exist.
- There might be some relation between underlying symmetries and constraints in a physical system, limiting the class of computations that they can accelerate.
- Significant energy savings only apply to the inference stage, since a digital model is used while training.
- ❗ **Note**: PNNs do not supplant existing training hardware.
- The complexity of physical systems doesn't necessarily translate to better *controllability* and hence better PNNs.

Potential Applications

- PNNs unlock possibilities for massively parallel natural computations, as a *co-processor*.
- Wherever quantization loss is an issue, PNNs can help.
- Can allow signal pre-processing in ex-silico domains, such as optical, microfluidic or chemical, that can then be converted to electrical signals $\rightarrow$ *smarter* sensors.
- Frozen parts of large DL models, such as large-scale language models, can be made more efficient using PAT (during inference).
- Can help with early-stage research and development of new computing platforms $\rightarrow$ NISQ applications.
- *Possible future research direction*: Analyzing "inductive biases" of physical processes.

References

- Original paper and supplementary documents
  Demos on GitHub
- Heavily inspired by Peter McMahon's Talk @ Qiskit Seminar Series on YouTube

Thank you! Any Questions?