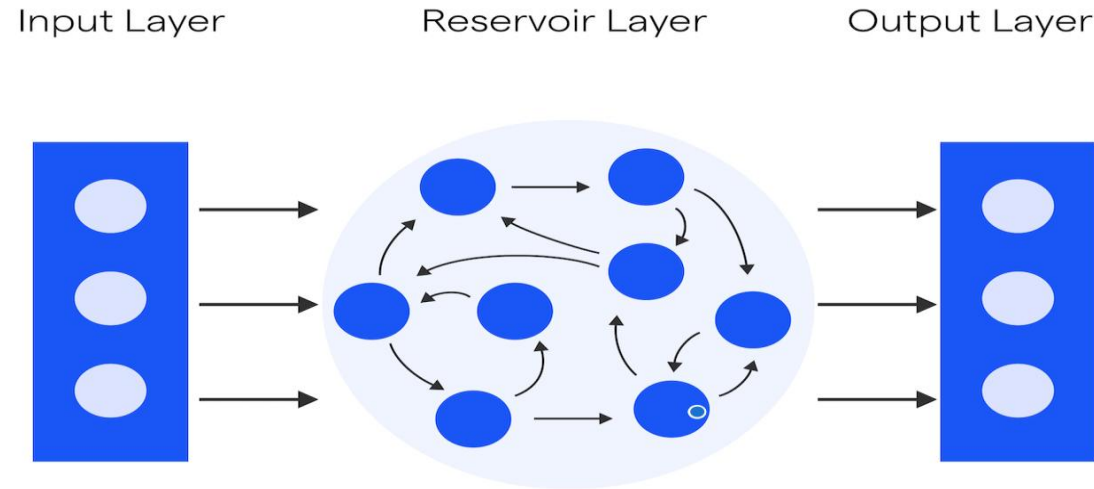


# Enhanced Rapid Training Echo Convolutional Network For Image Recognition

- 
- 01 Introduction to Echo State Network
  - 02 Introduction to Convolutional Neural Network
  - 03 Rapid Training Echo Convolutional Network For Image Recognition (RT-ECN)
  - 04 Efficient Rapid Training Echo Convolutional Network For Image Recognition (ERT-ECN)
  - 05 Result
  - 06 Summary

# Introduction to Echo State Network



### Definition:

- A computational framework derived from Recurrent Neural Networks (RNNs).
- Independently developed by Jaeger (2001), introducing the concept of Echo State Networks (ESNs).

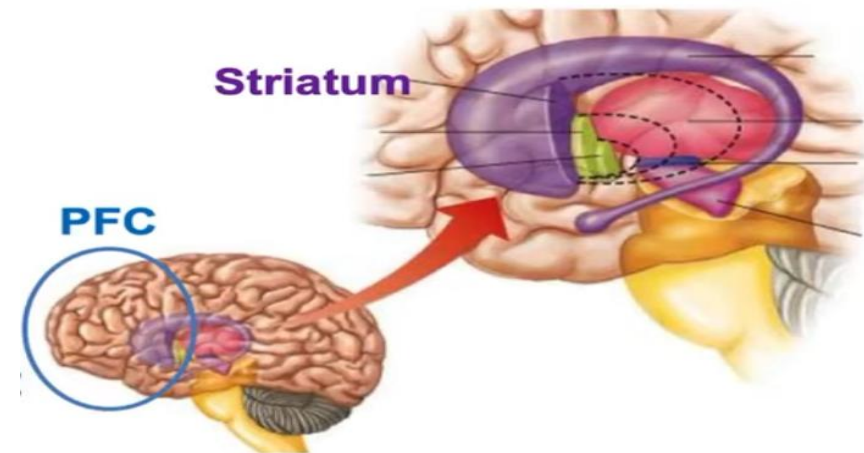
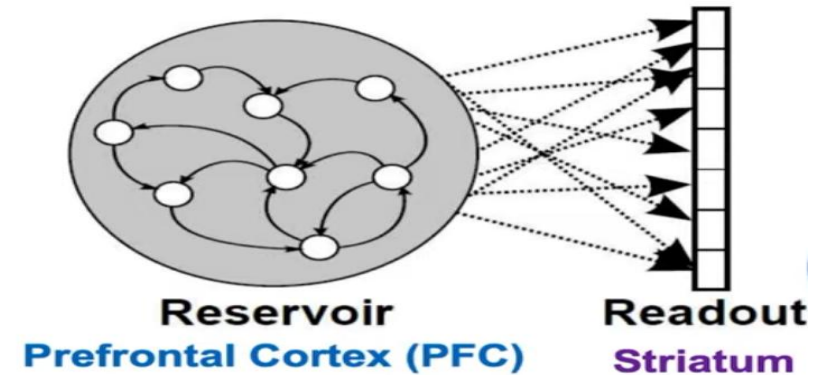
### Key Features of Reservoir Computing:

1. Random Reservoir Generation: The RNN, referred to as the reservoir, is randomly generated and remains unchanged during training.
2. Simplified Training: Only the readout layer is trained, significantly reducing computational complexity.

# Reservoir Computer Resemblances With Human Brain

## Biological Representation

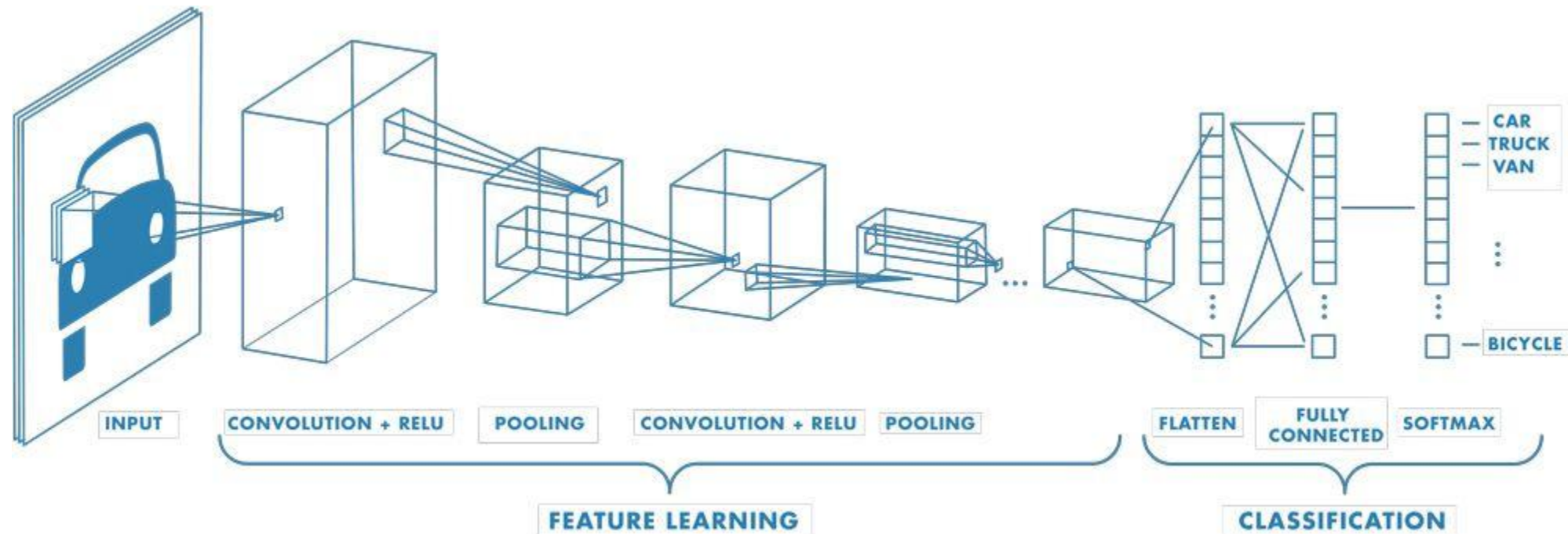
- **Reservoir (analogy: PFC/cortical microcircuits):** Recurrent dynamics maintain and mix input over time, yielding a rich, high-dimensional state useful for working memory and decision-making.
- **Readout:** Cortex (reservoir) maintains rich state; striatum learns state->action under dopamine; selected actions are issued via GPi/SNr ->thalamus ->motor areas.



# Introduction to Convolutional Neural Network

# Introduction to Convolutional Neural Network

## Overview



Convolutional layer: Feature extraction

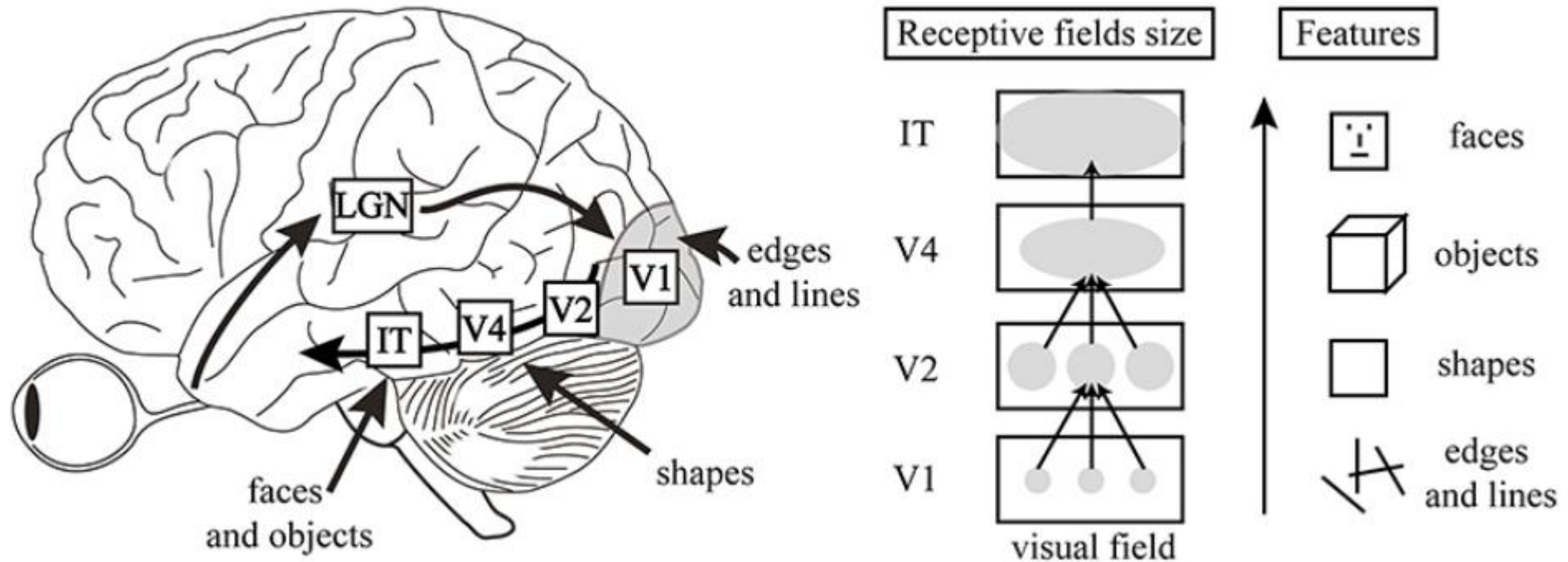
Activation function: Nonlinearity

Pooling layer: Compress and aggregate information, save parameters

Last layer: Fully-connected for classification

# Introduction to Convolutional layer

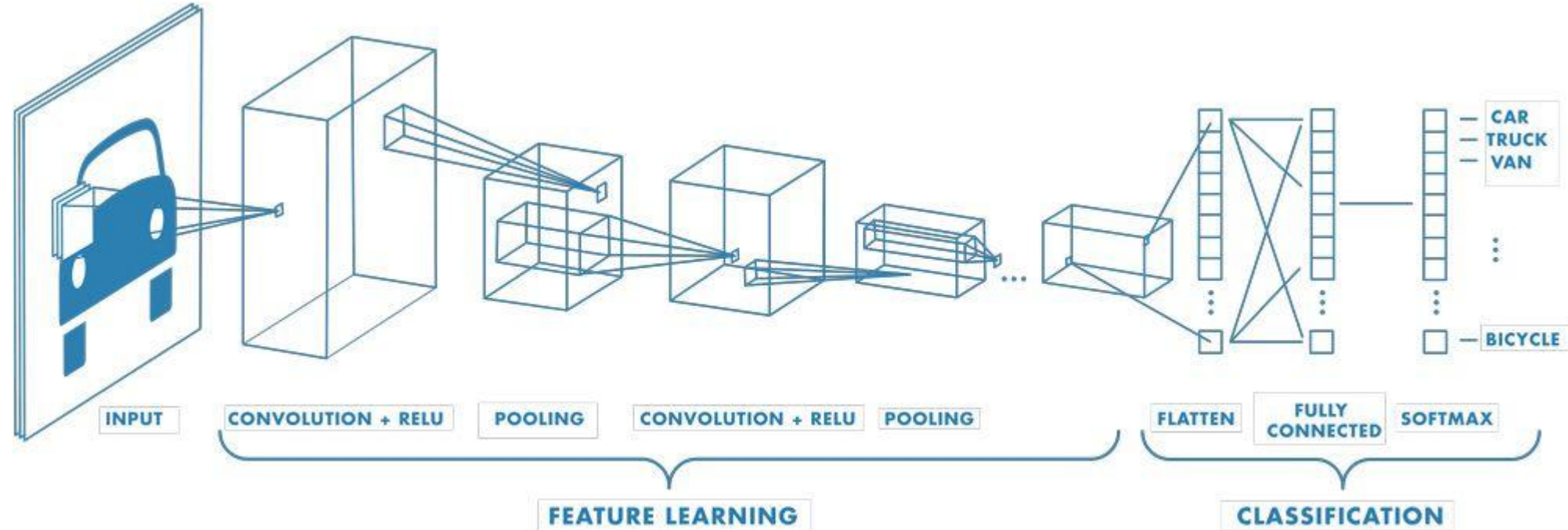
## Biological Representation



**Eye + visual cortex:** Visual information is processed hierarchically: retina (Captures raw light (pixels)) → Lateral geniculate nucleus (Relays & refines visual signals) → V1 (edges) → V2/V4 (patterns, colors) → IT cortex (complex objects, faces).

# Introduction to Convolutional Neural Network

Why to avoid FNN



Convolutional layer: Feature extraction

Activation function: Nonlinearity

Pooling layer: Compress and aggregate information, save parameters

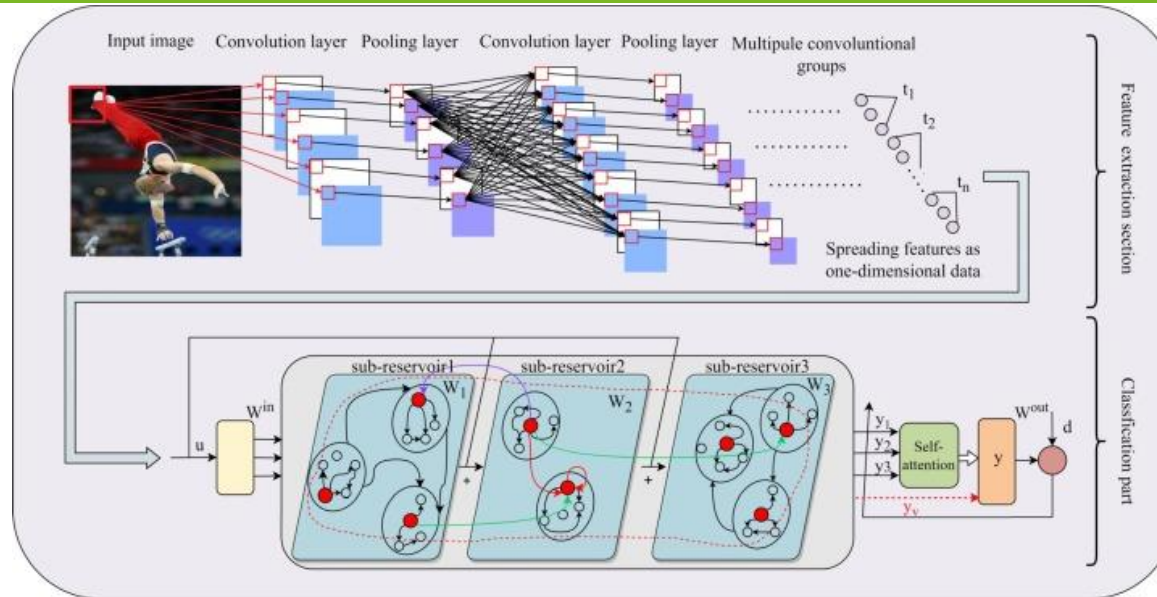
Last layer: Fully-connected for classification

---

# Rapid training echo convolution network(RT-ECN)

# Rapid training echo convolution network

CNN beginning



For an input image  $I \in \mathbb{R}^{M \times N \times C_{in}}$ , it can be represented as follows:

$$I = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1N} \\ a_{21} & a_{22} & \cdots & a_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ a_{M1} & a_{M2} & \cdots & a_{MN} \end{bmatrix} \quad (3)$$

where  $M$ ,  $N$ , and  $C_{in}$  represent the length, width, and the number of channels of the image, respectively.

$$K = \begin{bmatrix} k_{11} & k_{12} & k_{13} \\ k_{21} & k_{22} & k_{23} \\ k_{31} & k_{32} & k_{33} \end{bmatrix} \quad (4)$$

The output matrix  $H$  of convolution is:

$$H = \begin{bmatrix} H_{11} + b & H_{12} + b & \cdots & H_{1N} + b \\ H_{21} + b & H_{22} + b & \cdots & H_{2N} + b \\ \vdots & \vdots & \ddots & \vdots \\ H_{M1} + b & H_{M2} + b & \cdots & H_{MN} + b \end{bmatrix} \quad (5)$$

where  $b$  represents bias. And the specific calculation formula for  $H_{ij}$  is as follows:

$$H_{ij} = \left( \sum_{m=1}^3 \sum_{n=1}^3 k_{mn} * a_{(i+m-1)(j+n-1)} \right) + b \quad (6)$$

where  $f$  is the activation function, and the Leakey ReLU function is used in this paper. The formula of Leakey ReLU is as follows:

$$f = \begin{cases} x & \text{if } x \geq 0 \\ \alpha x & \text{if } x < 0 \end{cases} \quad (8)$$

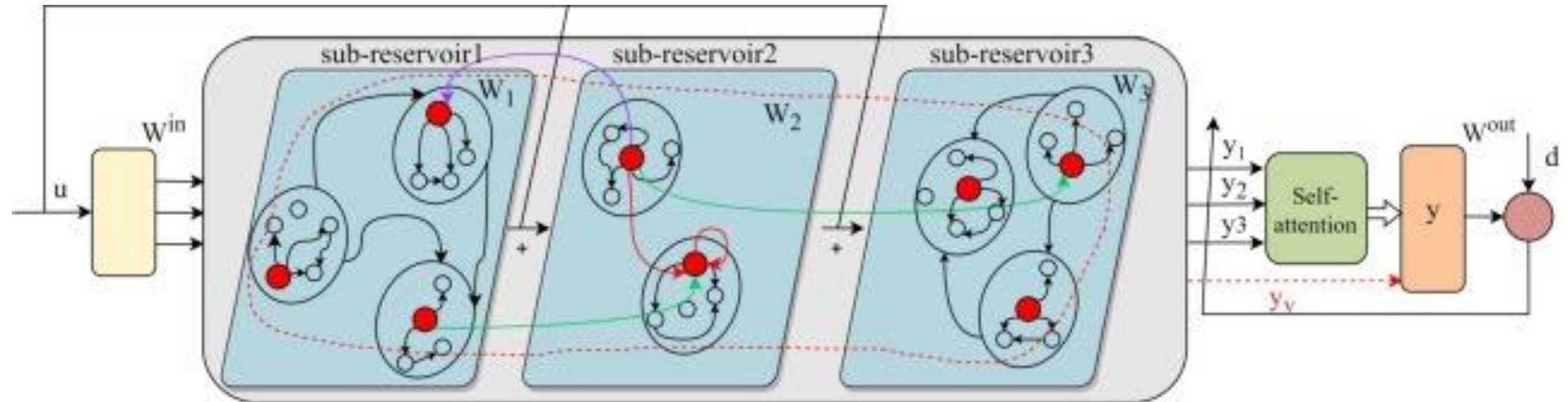
Following each convolution operation, a maximum pooling operation is applied to reduce the size of the feature map. Assuming that the pooling window size is  $2 \times 2$ , the pooling matrix  $P$  is defined as:

$$P = \begin{bmatrix} P_{11} & P_{12} \\ P_{21} & P_{22} \end{bmatrix} \quad (9)$$

$$P_{i,j} = \max(H_{i+p_1, j+p_2} | 0 \leq p_1 \leq 2, 0 \leq p_2 \leq 2) \quad (10)$$

After feature extraction, we obtain as many feature maps as there are convolutional kernels. These feature maps are then combined and flattened into one-dimensional vectors, which are fed into the M-ESN. The flatten operation can be expressed as:

$$v = \text{flatten}(H'_{i,j}) \in \mathbb{R}^{H'+W'+C'} \quad (11)$$

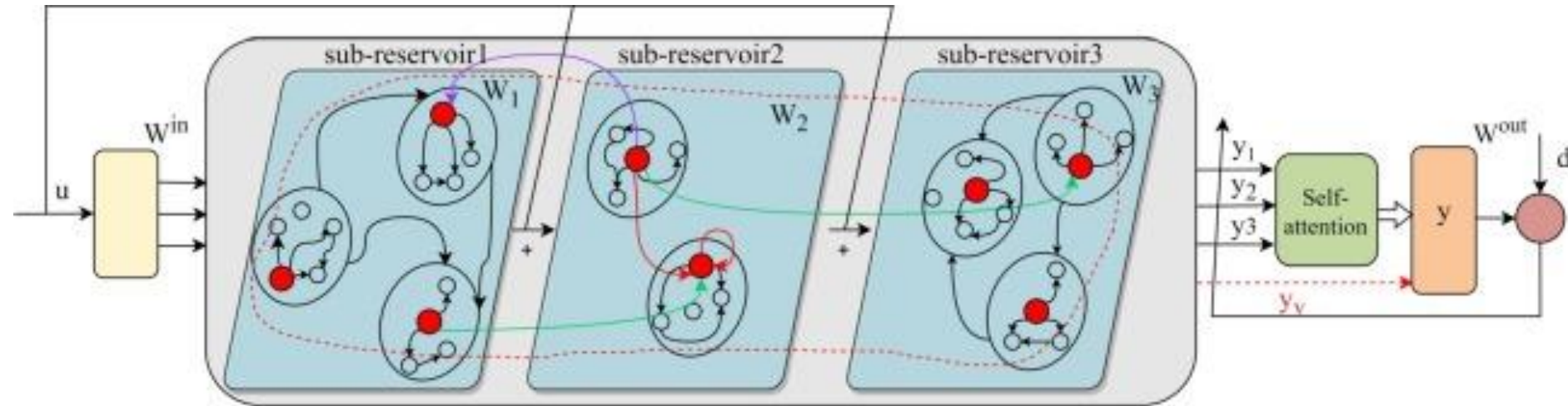


$$x_{d,w}(t+1) = \begin{cases} \tanh(W_{d,w}^{in} u(t+1) + W_{d,w} x_{d,w}(t)) & d = 1 \\ \tanh(W_{d,w}^{in} [u(t+1); x_{d-1,w}(t+1)] + W_{d,w} x_{d,w}(t)) & d > 1 \end{cases}$$

**Input:** Size of input layer  $S_{in}$ , Size of output layer  $S_{out}$ , Sparsity factor  $\sigma$

**Output:** Initialized reservoir weight matrix  $W_{d,w}$

- 1 Generate random weights matrix  $W_{d,w}$  of dimensions  $S_{in} \times S_{out}$  scaled by 0.5;
- 2 Generate random mask matrix  $M$  of dimensions  $S_{in} \times S_{out}$ ;
- 3 Set elements in  $W_{d,w}$  to 0 where corresponding elements in  $M$  are less than  $\sigma$ ;
- 4 Compute eigenvalues of  $W_{d,w}$  and find the maximum absolute eigenvalue  $\lambda_{(max)}$  i.e.,  $\rho(W_{d,w})$ ;
- 5 if  $\rho(W_{d,w}) > 1$  then
- 6     | Scale  $W$  by dividing all elements by  $\rho(W_{d,w})$ ;
- 7 end
- 8 Return  $W_{d,w}$ ;



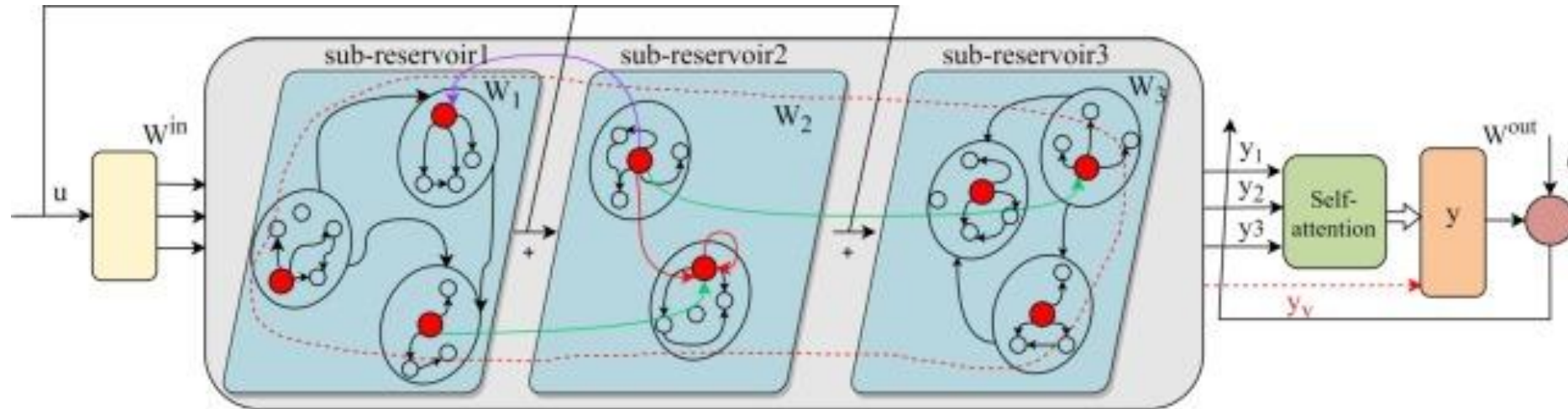
For the  $d$ -th deep sub-reservoir, it is obtained by connecting the states of all broad sub-reservoirs:

$$x_d(t+1) = [x_{d1}(t+1); x_{d2}(t+1); \dots; x_{dw}(t+1)] \quad (13)$$

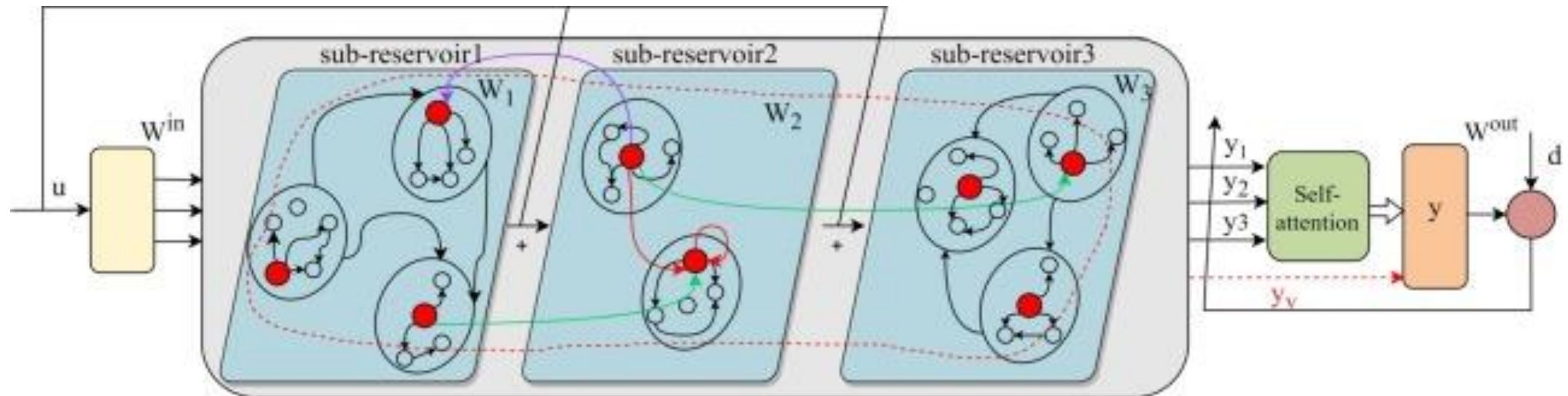
where  $x_d(t+1)$  denotes the state vector of the  $d$ -th deep reservoir.

For the  $w$ -th broad sub-reservoir in the  $d$ -th depth sub-reservoir, its reservoir state vector is:

$$x_{d,w}(t+1) = [(x_{d,w}^1); (x_{d,w}^2); \dots; (x_{d,w}^\theta)]^T \quad (14)$$

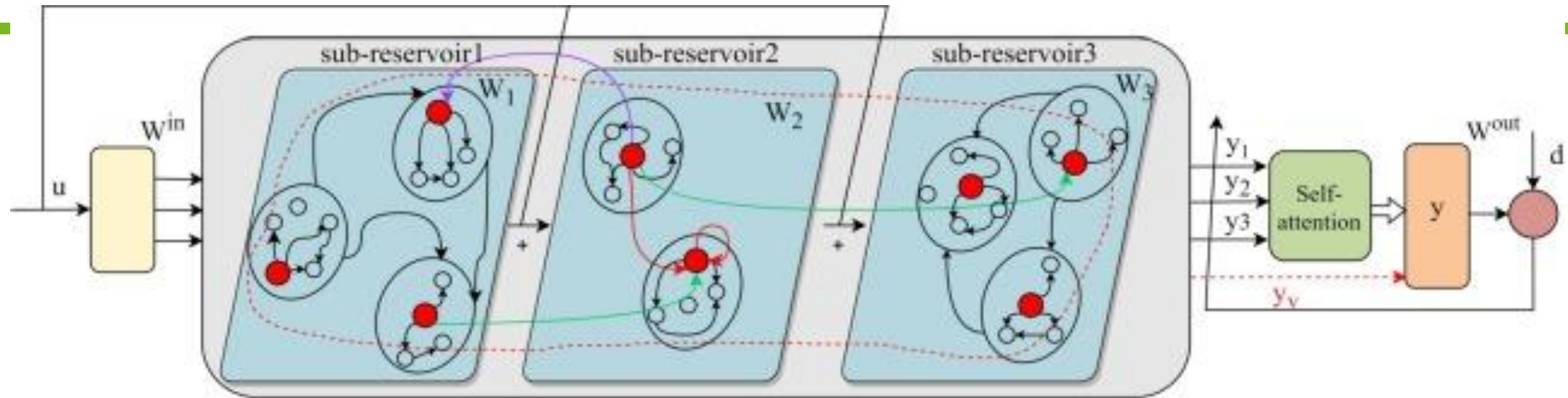


$$x_v(t+1) = (1 - \delta) x(t+1) + \delta * \tanh(W_v^{in} u(t+1) + W_v x_v(t)) \quad (15)$$



$$y_d(t+1) = W_d^{out} (Attn(x_d(t+1)); x_{vi}(t+1))$$

# Rapid training echo convolution network



After the CNN + reservoir, you get feature states  $X$ .

Shape:  $X \in \mathbb{R}^{N \times D}$

$D$  = feature dimension (reservoir + virtual states).

$N$  = number of training samples

$C$  = number of class

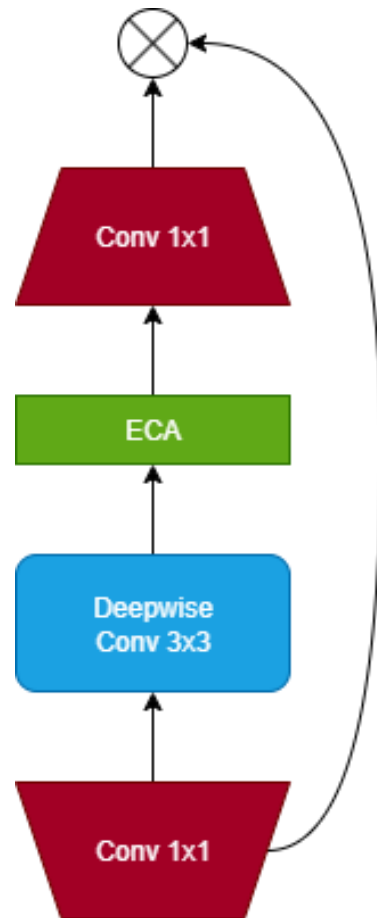
Shape  $Y \in \mathbb{R}^{N \times C}$  is the target output matrix, which in image recognition is the label of the image

Solved by  $W_{out} = (XX^T)^{-1}X^TY \rightarrow$  The closed-form least squares solution is the pseudo-inverse.

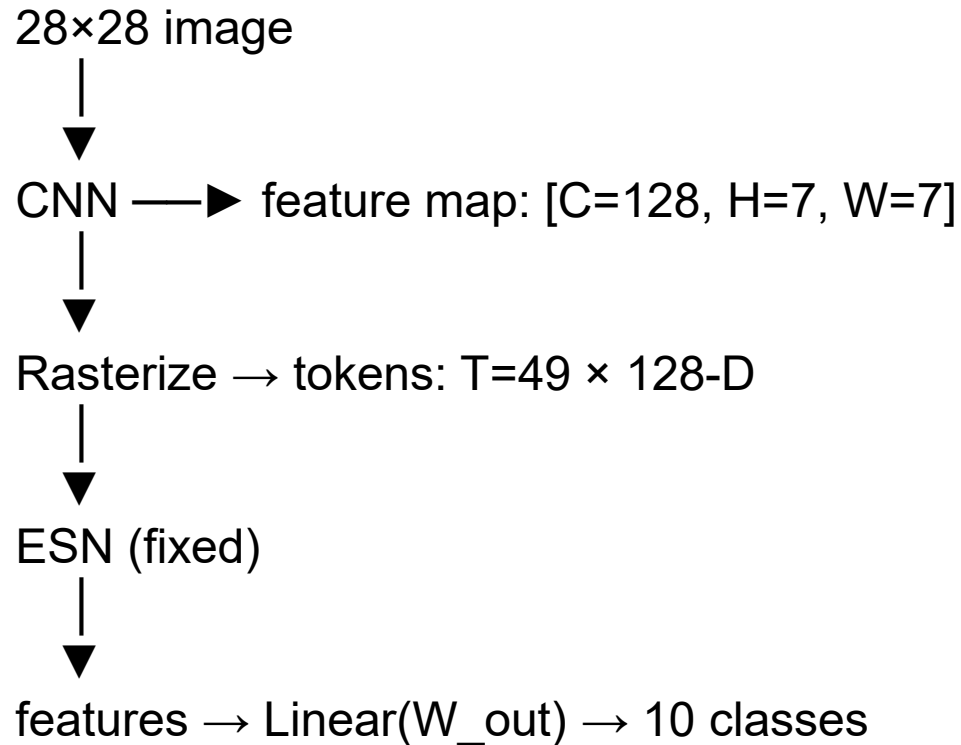
# Enhanced Rapid Training Echo Convolutional (ERTECNet)

# CNN Backbone

Efficient-Net style feature extractor



Layer (type:depth-idx)	Input Shape	Output Shape	Param #
RTECNet	[1, 1, 28, 28]	[1, 10]	--
CNNFeatureExtractor: 1-1	[1, 1, 28, 28]	[1, 128, 7, 7]	--
Sequential: 2-1	[1, 1, 28, 28]	[1, 32, 28, 28]	--
Conv2d: 3-1	[1, 1, 28, 28]	[1, 32, 28, 28]	288
BatchNorm2d: 3-2	[1, 32, 28, 28]	[1, 32, 28, 28]	64
SiLU: 3-3	[1, 32, 28, 28]	[1, 32, 28, 28]	--
Sequential: 2-2	[1, 32, 28, 28]	[1, 128, 7, 7]	--
MBConv: 3-4	[1, 32, 28, 28]	[1, 32, 28, 28]	--
Sequential: 4-1	[1, 32, 28, 28]	[1, 32, 28, 28]	9,923
Identity: 4-2	[1, 32, 28, 28]	[1, 32, 28, 28]	--
SiLU: 4-3	[1, 32, 28, 28]	[1, 32, 28, 28]	--
MBConv: 3-5	[1, 32, 28, 28]	[1, 64, 14, 14]	--
Sequential: 4-4	[1, 32, 28, 28]	[1, 64, 14, 14]	16,131
SiLU: 4-5	[1, 64, 14, 14]	[1, 64, 14, 14]	--
MBConv: 3-6	[1, 64, 14, 14]	[1, 64, 14, 14]	--
Sequential: 4-6	[1, 64, 14, 14]	[1, 64, 14, 14]	40,323
DropPath: 4-7	[1, 64, 14, 14]	[1, 64, 14, 14]	--
SiLU: 4-8	[1, 64, 14, 14]	[1, 64, 14, 14]	--
MBConv: 3-7	[1, 64, 14, 14]	[1, 128, 7, 7]	--
Sequential: 4-9	[1, 64, 14, 14]	[1, 128, 7, 7]	56,835
SiLU: 4-10	[1, 128, 7, 7]	[1, 128, 7, 7]	--
MBConv: 3-8	[1, 128, 7, 7]	[1, 128, 7, 7]	--
Sequential: 4-11	[1, 128, 7, 7]	[1, 128, 7, 7]	137,987
DropPath: 4-12	[1, 128, 7, 7]	[1, 128, 7, 7]	--
SiLU: 4-13	[1, 128, 7, 7]	[1, 128, 7, 7]	--
MESNReadout: 1-2	[1, 49, 128]	[1, 10]	10,590



Step 1 — CNN makes “feature tiles”: the image goes through a tiny CNN that turns it into a grid of features (size ~7×7, each tile has 128 numbers).

Step 2 — Make a sequence (spatial tokens): read the grid left-to-right, top-to-bottom. Each tile becomes one token = the 128-D channel vector at that spot. → Sequence length T, 49 (7×7), token size C = 128.

Step 3 — ESN summarizes: a fixed (non-trained) echo state network “scans” the 49 tokens, mixing nearby info over time.

Step 4 — Linear head predicts: concatenate the ESN states and feed a simple linear layer to get 10 logits (digits 0–9 → MNIST Dataset).

## **In the paper:**

- After the CNN + reservoir, you get feature states  $X$ .

Shape:  $X \in \mathbb{R}^{N \times D}$

$D$  = feature dimension (reservoir + virtual states).

$N$  = number of training samples

$C$  = number of class

Shape  $Y \in \mathbb{R}^{N \times C}$  is the target output matrix, which in image recognition is the label of the image

Solved by  $W_{out} = (XX^T)^{-1}X^TY$  -> The closed-form least squares solution is the pseudo-inverse.

## In my method:

Shape:  $X \in \mathbb{R}^{B \times D}$

$B$  = batch size,  $D$  = feature dimension

Augment with bias by appending a column of 1s:

$$X_{aug} = [X \ 1] \in \mathbb{R}^{B \times (D+1)}$$

Targets are one-hot:

$$Y \in \mathbb{R}^{B \times C}$$

$C$  = number of classes

Instead of storing all data, accumulate sufficient statistics:

$$S_{xx} = \sum X_{aug} X_{aug}^T \quad S_{xy} = \sum Y_{aug} X_{aug}^T$$

At the end of each epoch (or for warm-start), solve the ridge regression system:

$$W_{out} = (S_{xx} + \lambda I)^{-1} S_{xy}$$

Since the accumulation method allowed faster calculation. It was possible to also optimize CNN. In order to find initial  $W_{out}$ , you need to just apply warm-up your ESN without any CNN update: for the first time, give certain amount of the training data (can be equivalent to batch-size; I use this heuristic). Later:

1.  $x \rightarrow CNN \rightarrow \text{feature map} \rightarrow \text{spatial tokens}$
2.  $\text{tokens} \rightarrow ESN(\text{frozen}) \rightarrow \text{features } X$
3. Split into two lanes using the same  $X$  :
  - Stats lane (no grad):  
Meaning: take the *numbers* of  $X$ :  
Effect: update running sums:  
 $S_{xx} = \sum X_{aug} X_{aug}^T$  and  $S_{xy} = \sum Y_{aug} X_{aug}^T$
  - Learning lane (with grad):
    1. Compute  $Y$  with the **current, fixed  $w_{out}(\text{head})$** :
    2.  $Y = X_{aug} @ W_{out}$
    3. Compute loss (cross-entropy), backpropagation  $\rightarrow$  AdamW updates the CNN
    4. Gradients flow **only through the CNN** (ESN + head  $\rightarrow$  no updates).
    5. Result: **CNN updates every batch.**

## End of the epoch

Solve the ridge regression for the head:

$$W_{out} = (S_{xx} + \lambda I)^{-1} S_{xy}$$

## AMP (automatic mixed precision):

**What it is:** run most math in **16-bit** (half) instead of 32-bit when it's safe. Keep “tricky” parts and the **ESN** in **32-bit**. A **GradScaler** keeps gradients from going to zero.

**Why it helps you:** my **MBConv/convs + ECA** active on NVIDIA **Tensor Cores**, which are much faster in FP16 and use less memory. My **ESN** can stay in FP32 so it stays stable.

## torch.compile:

**What it is:** turns my model into an optimized graph and **fuses lots of tiny ops** (e.g., SiLU, DropPath, adds) into **fewer, bigger GPU kernels**.

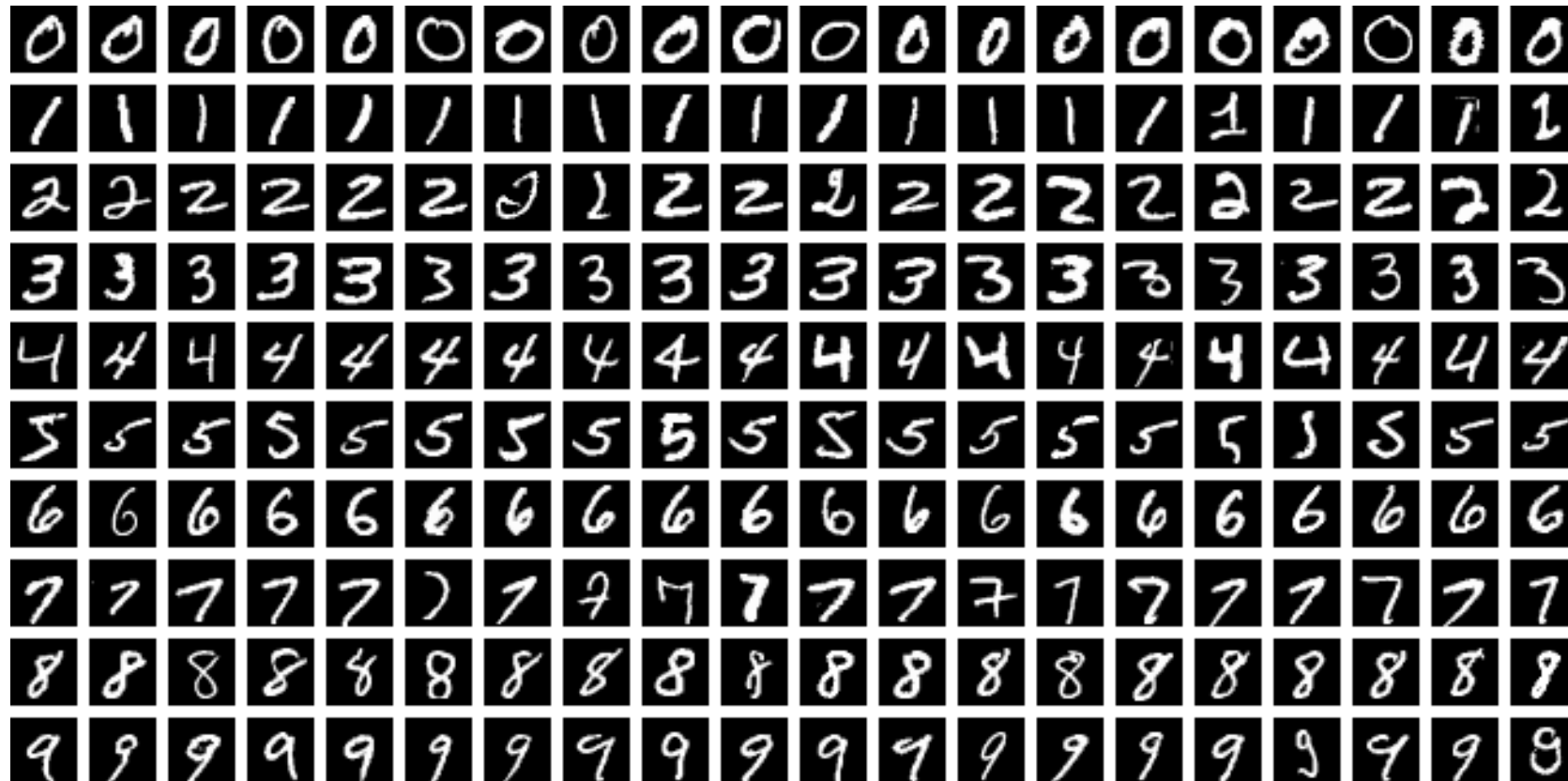
**Why it helps you:** fewer kernel launches + better memory use -> **faster**

- **Accumulation** ( $S_{xx}$ ,  $S_{xy}$ ) over batches and **ridge regression** solution, **Cholesky with jitter** and **eigendecomposition** fallback for robustness.
- **Uniform** initialization ( $W_{in} \sim U[-1,1]$ ,  $W \sim U[-0.5,0.5]$ ), apply a **sparse mask**, then **scale to target  $p=0.99$** . Same goal (ESP), but **different distributions** and an explicit target  $p$
- computes per-(**deep layer index** , **sub-reservoir index**) **mean over  $\theta$**  as the representative rather than the paper's "first neuron", then applies the same **leaky update** with  $Wv_{in} / Wv$ .
- My self-attention implementation adds **RMSNorm**, **dropout**, **residual + gating**, **layer embeddings**, and (optionally) **causal masking**—all **implementation choices** not specified in the paper
- Bigger picture -> different feature map: **28×28** image → about **7×7 tiles**, **128 channels (32,64,128)** each.  
**96×96** image → about **12×12 tiles**, **192 (48,96,192) channels** each.

# Result

# MNIST Dataset

Overview image



- My Implementation lead better result in shorter time around 15 minutes. It reached 99.66 accuracy in 29 epoch. It is faster and higher accuracy
- It trained with no attention layer, normalization exactly same as mentioned in the RT-ECN article.
- CNN Trainable params: 261,551. ESN parameter counter:10,590. Then 272,141 in total.
- My computer had i7-10750H CPU @ 2.60GHz (2.59 GHz) + RTX 2060 Laptop ,and in the article, laptop had i9-13900HX + RTX 4060 Laptop

```
ESN Config: L=3, S=4, neurons_per_deep=20, theta=5, R=60, Attention: none
Model CNN is on: cuda:0
Model ESN is on: cuda:0
Warm-starting W_out...
Epoch 1/37: Train Loss = 1.7710 | Test Loss = 1.5950 | Test Acc = 94.28% | F1 = 0.9420 | Prec = 0.9432 | Recall = 0.9430 | Time = 63.65s
Epoch 2/37: Train Loss = 1.5281 | Test Loss = 1.4974 | Test Acc = 97.56% | F1 = 0.9752 | Prec = 0.9762 | Recall = 0.9749 | Time = 20.16s
Epoch 3/37: Train Loss = 1.4660 | Test Loss = 1.4808 | Test Acc = 98.34% | F1 = 0.9833 | Prec = 0.9834 | Recall = 0.9834 | Time = 21.49s
Epoch 4/37: Train Loss = 1.4816 | Test Loss = 1.4777 | Test Acc = 98.89% | F1 = 0.9888 | Prec = 0.9890 | Recall = 0.9887 | Time = 21.90s
Epoch 5/37: Train Loss = 1.4868 | Test Loss = 1.4791 | Test Acc = 98.84% | F1 = 0.9883 | Prec = 0.9884 | Recall = 0.9882 | Time = 21.09s
Epoch 6/37: Train Loss = 1.4905 | Test Loss = 1.4810 | Test Acc = 98.66% | F1 = 0.9866 | Prec = 0.9869 | Recall = 0.9864 | Time = 22.39s
Epoch 7/37: Train Loss = 1.4863 | Test Loss = 1.4743 | Test Acc = 99.13% | F1 = 0.9912 | Prec = 0.9912 | Recall = 0.9913 | Time = 22.02s
Epoch 8/37: Train Loss = 1.4882 | Test Loss = 1.4768 | Test Acc = 99.13% | F1 = 0.9912 | Prec = 0.9913 | Recall = 0.9912 | Time = 21.53s
Epoch 9/37: Train Loss = 1.4848 | Test Loss = 1.4827 | Test Acc = 98.29% | F1 = 0.9830 | Prec = 0.9833 | Recall = 0.9831 | Time = 21.68s
Epoch 10/37: Train Loss = 1.4834 | Test Loss = 1.4739 | Test Acc = 98.98% | F1 = 0.9897 | Prec = 0.9900 | Recall = 0.9895 | Time = 21.14s
Epoch 11/37: Train Loss = 1.4800 | Test Loss = 1.4729 | Test Acc = 99.22% | F1 = 0.9921 | Prec = 0.9923 | Recall = 0.9921 | Time = 22.91s
Epoch 12/37: Train Loss = 1.4767 | Test Loss = 1.4716 | Test Acc = 99.27% | F1 = 0.9926 | Prec = 0.9926 | Recall = 0.9927 | Time = 21.68s
Epoch 13/37: Train Loss = 1.4830 | Test Loss = 1.4716 | Test Acc = 99.39% | F1 = 0.9939 | Prec = 0.9939 | Recall = 0.9939 | Time = 21.15s
Epoch 14/37: Train Loss = 1.4794 | Test Loss = 1.4765 | Test Acc = 98.93% | F1 = 0.9893 | Prec = 0.9895 | Recall = 0.9893 | Time = 22.97s
Epoch 15/37: Train Loss = 1.4756 | Test Loss = 1.4710 | Test Acc = 99.34% | F1 = 0.9934 | Prec = 0.9935 | Recall = 0.9934 | Time = 21.97s
Epoch 16/37: Train Loss = 1.4819 | Test Loss = 1.4719 | Test Acc = 99.21% | F1 = 0.9919 | Prec = 0.9919 | Recall = 0.9921 | Time = 21.18s
Epoch 17/37: Train Loss = 1.4770 | Test Loss = 1.4744 | Test Acc = 99.00% | F1 = 0.9898 | Prec = 0.9904 | Recall = 0.9894 | Time = 21.82s
Epoch 18/37: Train Loss = 1.4834 | Test Loss = 1.4776 | Test Acc = 99.06% | F1 = 0.9906 | Prec = 0.9908 | Recall = 0.9906 | Time = 21.62s
Epoch 19/37: Train Loss = 1.4699 | Test Loss = 1.4715 | Test Acc = 99.25% | F1 = 0.9924 | Prec = 0.9925 | Recall = 0.9924 | Time = 21.42s
Epoch 20/37: Train Loss = 1.4804 | Test Loss = 1.4702 | Test Acc = 99.24% | F1 = 0.9923 | Prec = 0.9923 | Recall = 0.9923 | Time = 22.05s
Epoch 21/37: Train Loss = 1.4823 | Test Loss = 1.4701 | Test Acc = 99.26% | F1 = 0.9925 | Prec = 0.9927 | Recall = 0.9924 | Time = 22.08s
Epoch 22/37: Train Loss = 1.4770 | Test Loss = 1.4713 | Test Acc = 99.21% | F1 = 0.9920 | Prec = 0.9922 | Recall = 0.9918 | Time = 21.45s
Epoch 23/37: Train Loss = 1.4747 | Test Loss = 1.4697 | Test Acc = 99.25% | F1 = 0.9924 | Prec = 0.9926 | Recall = 0.9923 | Time = 23.41s
Epoch 24/37: Train Loss = 1.4740 | Test Loss = 1.4699 | Test Acc = 99.34% | F1 = 0.9933 | Prec = 0.9935 | Recall = 0.9932 | Time = 22.03s
Epoch 25/37: Train Loss = 1.4778 | Test Loss = 1.4700 | Test Acc = 99.28% | F1 = 0.9927 | Prec = 0.9927 | Recall = 0.9928 | Time = 21.12s
Epoch 26/37: Train Loss = 1.4762 | Test Loss = 1.4701 | Test Acc = 99.33% | F1 = 0.9933 | Prec = 0.9934 | Recall = 0.9932 | Time = 22.79s
Epoch 27/37: Train Loss = 1.4744 | Test Loss = 1.4695 | Test Acc = 99.37% | F1 = 0.9937 | Prec = 0.9938 | Recall = 0.9936 | Time = 21.96s
Epoch 28/37: Train Loss = 1.4797 | Test Loss = 1.4720 | Test Acc = 99.14% | F1 = 0.9914 | Prec = 0.9914 | Recall = 0.9914 | Time = 21.44s
Epoch 29/37: Train Loss = 1.4715 | Test Loss = 1.4671 | Test Acc = 99.66% | F1 = 0.9966 | Prec = 0.9967 | Recall = 0.9965 | Time = 22.41s
Epoch 30/37: Train Loss = 1.4760 | Test Loss = 1.4731 | Test Acc = 98.91% | F1 = 0.9890 | Prec = 0.9894 | Recall = 0.9889 | Time = 22.77s
Epoch 31/37: Train Loss = 1.4768 | Test Loss = 1.4704 | Test Acc = 99.24% | F1 = 0.9924 | Prec = 0.9925 | Recall = 0.9923 | Time = 21.86s
Epoch 32/37: Train Loss = 1.4713 | Test Loss = 1.4695 | Test Acc = 99.40% | F1 = 0.9940 | Prec = 0.9940 | Recall = 0.9940 | Time = 22.03s
Epoch 33/37: Train Loss = 1.4755 | Test Loss = 1.4707 | Test Acc = 99.19% | F1 = 0.9918 | Prec = 0.9919 | Recall = 0.9917 | Time = 22.51s
Epoch 34/37: Train Loss = 1.4650 | Test Loss = 1.4678 | Test Acc = 99.46% | F1 = 0.9946 | Prec = 0.9947 | Recall = 0.9945 | Time = 21.26s
Epoch 35/37: Train Loss = 1.4768 | Test Loss = 1.4672 | Test Acc = 99.48% | F1 = 0.9948 | Prec = 0.9948 | Recall = 0.9947 | Time = 22.00s
Epoch 36/37: Train Loss = 1.4773 | Test Loss = 1.4678 | Test Acc = 99.56% | F1 = 0.9956 | Prec = 0.9956 | Recall = 0.9956 | Time = 22.42s
Epoch 37/37: Train Loss = 1.4706 | Test Loss = 1.4675 | Test Acc = 99.51% | F1 = 0.9951 | Prec = 0.9952 | Recall = 0.9950 | Time = 21.15s

=====
Total execution time: 871.19 seconds
Average time per epoch: 23.55 seconds
```

# MNIST Dataset

ESN comparison “Rapid training echo convolution network for image recognition

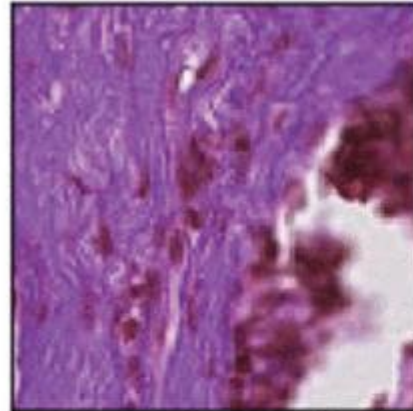
ESN Model	Neurons	Accuracy
<u>“Exploring deep echo state networks for image classification: a multi-reservoir approach”</u>	20000	98.43%
<u>“An image classification method based on echo state network”</u>	2000	92.75%
<u>“Protein structured reservoir computing for spike-based pattern recognition”</u>	2922	92.5%
<u>“Reservoir computing with untrained convolutional neural networks for image recognition”</u>	9000	99.25%
<u>“Image recognition model based on convolutional reservoir computing”</u>	4000	98.71%
<u>“Nonmasking-based reservoir computing with a single dynamic memristor for image recognition”</u>	936	98.44%
<u>“Energy-efficient edge and cloud image classification with multi-reservoir echo state network and data processing units”</u>	2000	97.20%
RT-ECN (the article)	60	99.57%
ERTECNet (my project)	60	99.66%

# MNIST Dataset

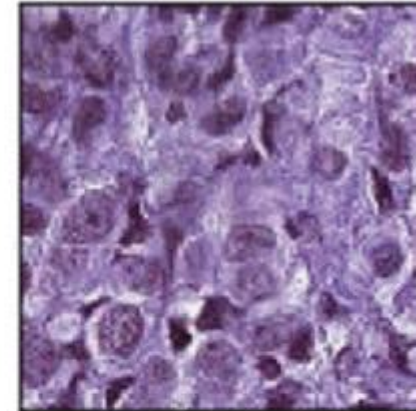
CNN comparison “Rapid training echo convolution network for image recognition

DL Methods	Prediction Accuracy	F1 Score
<u>Googlenet</u>	98.93	98.73
<u>LSTM</u>	99.37	99.25
<u>CNN with Hyperparameter optimization</u>	99.40	99.38
<u>Xception</u>	96.00	N/A
<u>ANN</u>	98.10	98.10
<u>CapsNet</u>	99.49	99.45
<u>Hybrid quantum neural networks</u>	99.21	99.24
<u>HeCapsNet</u>	99.51	99.50
<u>Alex Net</u>	99.08	99.08
RT-ECN	99.57	99.57
ERTECNet	99.66	99.66

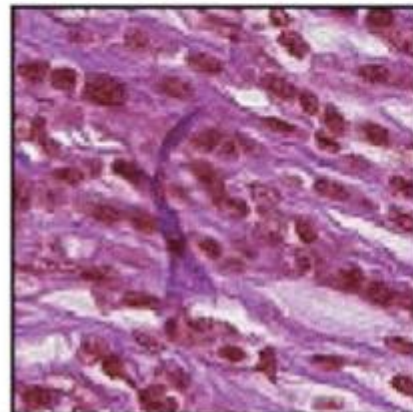
Label: 0



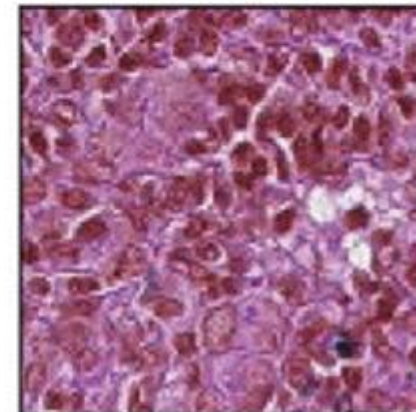
Label: 1



Label: 1



Label: 1



- My Implementation took roughly in 25 hours(100 epochs). It reached 97.73 accuracy in 91 epoch (23 hours)
- It trained with self-attention layer, normalization, RandomHorizontalFlip, RandomVerticalFlip, Color jitter. In the page, article data augmentation method will be provided
- CNN Trainable params: 570,487.ESN parameter counter:14,614. Then 585,101 parameter in total.
- My computer had i7-10750H CPU @ 2.60GHz (2.59 GHz) + RTX 2060 Laptop ,and the authors used The Google Colab Pro +

Epoch 70/100:	Train Loss = 0.2550	Test Loss = 0.3442	Test Acc = 97.45%	F1 = 0.9745	Prec = 0.9746	Recall = 0.9745	Time = 920.67s
Epoch 71/100:	Train Loss = 0.3209	Test Loss = 0.3467	Test Acc = 97.47%	F1 = 0.9747	Prec = 0.9748	Recall = 0.9747	Time = 942.07s
Epoch 72/100:	Train Loss = 0.3234	Test Loss = 0.3462	Test Acc = 97.54%	F1 = 0.9754	Prec = 0.9754	Recall = 0.9754	Time = 945.35s
Epoch 73/100:	Train Loss = 0.3207	Test Loss = 0.3449	Test Acc = 97.51%	F1 = 0.9751	Prec = 0.9751	Recall = 0.9751	Time = 915.76s
Epoch 74/100:	Train Loss = 0.2943	Test Loss = 0.3502	Test Acc = 97.53%	F1 = 0.9753	Prec = 0.9754	Recall = 0.9752	Time = 935.16s
Epoch 75/100:	Train Loss = 0.2957	Test Loss = 0.3447	Test Acc = 97.43%	F1 = 0.9743	Prec = 0.9744	Recall = 0.9743	Time = 927.07s
Epoch 76/100:	Train Loss = 0.3058	Test Loss = 0.3451	Test Acc = 97.60%	F1 = 0.9760	Prec = 0.9761	Recall = 0.9760	Time = 936.48s
Epoch 77/100:	Train Loss = 0.3306	Test Loss = 0.3440	Test Acc = 97.57%	F1 = 0.9757	Prec = 0.9757	Recall = 0.9756	Time = 924.50s
Epoch 78/100:	Train Loss = 0.3396	Test Loss = 0.3477	Test Acc = 97.62%	F1 = 0.9762	Prec = 0.9763	Recall = 0.9761	Time = 914.26s
Epoch 79/100:	Train Loss = 0.3198	Test Loss = 0.3443	Test Acc = 97.55%	F1 = 0.9755	Prec = 0.9755	Recall = 0.9755	Time = 919.43s
Epoch 80/100:	Train Loss = 0.3010	Test Loss = 0.3506	Test Acc = 97.61%	F1 = 0.9761	Prec = 0.9762	Recall = 0.9760	Time = 920.44s
Epoch 81/100:	Train Loss = 0.3319	Test Loss = 0.3564	Test Acc = 97.56%	F1 = 0.9756	Prec = 0.9757	Recall = 0.9756	Time = 920.89s
Epoch 82/100:	Train Loss = 0.2739	Test Loss = 0.3460	Test Acc = 97.60%	F1 = 0.9760	Prec = 0.9761	Recall = 0.9760	Time = 922.81s
Epoch 83/100:	Train Loss = 0.3002	Test Loss = 0.3440	Test Acc = 97.66%	F1 = 0.9766	Prec = 0.9767	Recall = 0.9766	Time = 923.14s
Epoch 84/100:	Train Loss = 0.2829	Test Loss = 0.3409	Test Acc = 97.60%	F1 = 0.9760	Prec = 0.9761	Recall = 0.9760	Time = 935.50s
Epoch 85/100:	Train Loss = 0.3075	Test Loss = 0.3430	Test Acc = 97.57%	F1 = 0.9757	Prec = 0.9757	Recall = 0.9757	Time = 925.78s
Epoch 86/100:	Train Loss = 0.3144	Test Loss = 0.3449	Test Acc = 97.54%	F1 = 0.9754	Prec = 0.9754	Recall = 0.9754	Time = 932.81s
Epoch 87/100:	Train Loss = 0.2841	Test Loss = 0.3429	Test Acc = 97.59%	F1 = 0.9759	Prec = 0.9759	Recall = 0.9759	Time = 940.34s
Epoch 88/100:	Train Loss = 0.3003	Test Loss = 0.3447	Test Acc = 97.58%	F1 = 0.9758	Prec = 0.9758	Recall = 0.9758	Time = 913.82s
Epoch 89/100:	Train Loss = 0.2980	Test Loss = 0.3418	Test Acc = 97.55%	F1 = 0.9755	Prec = 0.9756	Recall = 0.9755	Time = 915.84s
Epoch 90/100:	Train Loss = 0.3069	Test Loss = 0.3550	Test Acc = 97.67%	F1 = 0.9767	Prec = 0.9768	Recall = 0.9767	Time = 921.00s
Epoch 91/100:	Train Loss = 0.3276	Test Loss = 0.3456	Test Acc = 97.73%	F1 = 0.9773	Prec = 0.9774	Recall = 0.9772	Time = 912.32s
Epoch 92/100:	Train Loss = 0.3117	Test Loss = 0.3447	Test Acc = 97.67%	F1 = 0.9767	Prec = 0.9768	Recall = 0.9767	Time = 914.75s
Epoch 93/100:	Train Loss = 0.2429	Test Loss = 0.3425	Test Acc = 97.54%	F1 = 0.9754	Prec = 0.9755	Recall = 0.9754	Time = 917.48s
Epoch 94/100:	Train Loss = 0.2950	Test Loss = 0.3452	Test Acc = 97.53%	F1 = 0.9753	Prec = 0.9753	Recall = 0.9753	Time = 914.57s
Epoch 95/100:	Train Loss = 0.2813	Test Loss = 0.3400	Test Acc = 97.60%	F1 = 0.9760	Prec = 0.9760	Recall = 0.9761	Time = 922.83s

# Data Augmentation of the article

“Automatic detection of cancer metastasis in lymph node using deep learning”

Parameter	Value
Horizontal flip	True
Vertical flip	True
Rotation range	90 <sup>0</sup>
Maximum zoom	1.1
Maximum rotate	5
Maximum lighting	0.03
Maximum wrap	0.2
P_affine	0.75
P_lighting	0.75

# PCAM dataset

Result comparison “Automatic detection of cancer metastasis in lymph node using deep learning”

Study	Accuracy	Estimated number of parameters
Kassani et al. [1]	94.64 %	<b>171 million.</b>
ResNet-34 freeze	96.00 %	<b>21.5 million.</b>
Luz et al. [2]	96.90 %	<b>429 million</b>
Zheng et al. [3]	97.30 %	<b>26 million</b>
ResNet-50 freeze	97.30 %	<b>23.9 million</b>
ResNet-101 freeze	97.60 %	<b>42.8 million</b>
ERTECNet	97.73 %	585,101

# PCAM dataset

Result comparison “Automatic detection of cancer metastasis in lymph node using deep learning”

Study	Accuracy	Estimated number of parameters
Sun et al. <a href="#">[4]</a>	97.94 %	43 million
Wan et al. <a href="#">[5]</a>	98.10 %	7.21 million
ResNet-34 unfreeze	98.49 %	21.5 million.
ResNet-50 CBAM	98.58 %	23.9 million
ResNet-50 unfreeze	98.54 %	23.9 million
ResNet-101 unfreeze	98.60 %	42.8 million
Zheng et al. <a href="#">[6]</a> with TTA	98.80 %	23.9 million

# Summary

---

My models is a novel deep learning model. It is also unique in terms trainable feature extractor, which us non-existing for the other state-of-the-art ESN models.

Pro's:

- It is fast, and performs well despite small size parameter size
- Not demanding computationally as conventional CNNs

Con's:

- For medical applications, it is hard to make this model explainable (even darker black box)
- It is not as lightweight as the original article (it might not be suitable for embedded system which is indicated in the article)
- If Cholesky solver fails to solve the ridge regression, then it might be slower to calculate since it must use the eigenvalue decomposition method for the ridge regression. (So far, never occurred, also implement different methods other than adding jitter for Cholesky to make it robust).

# References

---

[Projections to the Basal Ganglia - Neuroscience - NCBI Bookshelf](#)

[Modulation of striatal projection systems by dopamine – PMC](#)

[Intrinsic and integrative properties of substantia nigra pars reticulata neurons – PMC](#)

[Explanation feedback](#)

[Frontiers | Alterations in neuronal activity in basal ganglia-thalamocortical circuits in the parkinsonian state](#)

---

[Convolutional Neural Networks as a Model of the Visual System: Past, Present, and Future | Journal of Cognitive Neuroscience | MIT Press](#)

[Convolutional neural networks for vision neuroscience: significance, developments, and outstanding issues - PMC](#)