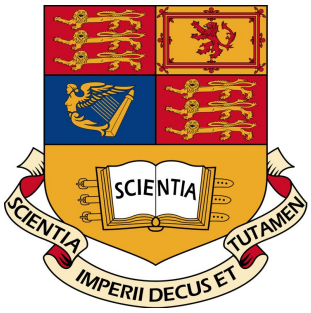


Imperial College London

Master of Research in Neurotechnology

Zifan Ning (CID: 02351834)
Department of Bioengineering

PlasticFlex: Learnable Path Switching between Selectivity and Invariance Inspired by Synaptic Plasticity

Andriy Kozlov, Ph.D
Thesis Adviser
Department of Bioengineering
Imperial College London

Claudia Clopath, Ph.D
Thesis Co-adviser
Department of Bioengineering
Imperial College London

Date of Submission:

September 2025

Word Count: 5899 words.

Abstract

PlasticFlex: Learnable Path Switching between Selectivity and Invariance Inspired by Synaptic Plasticity

Zifan Ning
Imperial College London
September 2025

Advisers:
Andriy Kozlov, Ph.D
Claudia Clopath, Ph.D

Biological systems balance information preservation with compression, while standard convolutional networks rely on a fixed convolution–pooling sequence. PlasticFlex introduces a per-location routing module that replaces convolutional layers with a learnable conv–pool gate. The module is drop-in: it preserves locality and downsampling semantics, unifies artificial and spiking networks in the same backbone, and remains compatible with both surrogate and non-surrogate training.

On MNIST, CIFAR-10, and ImageNet-100, PlasticFlex consistently improves robustness area-under-curve and training stability, with accuracy gains over strong baselines. Analysis of loss landscapes and Hessian spectra shows convergence to flatter and smoother minima. In spiking settings, routing promotes useful sparsity without loss of task accuracy. Measured routing indicators, such as convolution ratio and mask binariness, align with robustness and curvature trends, giving model-side interpretability. Taken together, these results position PlasticFlex as a certifiability-compatible adaptive routing mechanism rather than another variant of conditional gating. Details of code availability can be found in Appendix E.3

Acknowledgments

I would like to thank my primary supervisor, Professor Andriy Kozlov, for giving me the opportunity to study at Imperial College London and for his guidance throughout the past year. His advice has been invaluable for shaping this work. I am also grateful to my second supervisor, Professor Claudia Clopath, for her supervision and feedback during the project.

I would also like to thank Professor Simon Schultz and Professor Guang Yang for their valuable discussions during my mid-term poster presentation, which provided many helpful suggestions for improving this work.

This work used the Imperial College Research Computing Service, DOI: 10.14469/hpc/2232.

Table of Contents

Abstract i

Acknowledgments ii

List of Figures viii

List of Tables viii

Chapter 1 Introduction 1

1.1 Background & Motivation 1

1.2 Problem Statement 1

1.3 Research Objective 2

1.4 Contributions 2

Chapter 2 Related Works 3

2.1 Selectivity–Invariance Balance 3

2.2 Dynamic Routing: Mechanisms and Limits 4

2.3 Transformer-Era Dynamic Computation 4

2.4 Dynamic Mechanisms for Robustness 4

2.5 Neuroscience Signals for Flexible Routing 5

2.6 SNNs as a Substrate for Adaptive Computation 5

2.7 Positioning PlasticFlex 5

2.8 A Compact Comparison 6

2.9 Takeaway 6

Chapter 3 Methodology 7

3.1 Overview of the Approach 7

3.2 Network Architecture 8

3.3 Flex Routing Mechanism 8

3.4 Mask Behaviour Control 9

3.5 Spiking Behaviour 10

3.6 Training Procedure 11

3.7 Theoretical Analysis 12

3.7.1 Safety and Tightness . . . 12
3.7.2 Correlation-Aware Bounds . . . 13
3.7.3 Interval Propagation in Practice . . . 13
3.7.4 Linear Relaxations (CROWN-Compatible) . . . 13
3.7.5 Lipschitz View and Certified Stability . . . 14
3.7.6 Mask Sensitivity . . . 14
3.7.7 Complexity and Scalability . . . 14
3.7.8 Summary . . . 14

Chapter 4 Experiments . . . 15
4.1 Datasets . . . 15
4.1.1 MNIST . . . 15
4.1.2 CIFAR-10 . . . 16
4.1.3 ImageNet-100 . . . 16
4.2 Experimental Setup . . . 16
4.2.1 Training Details . . . 17
4.2.2 Baselines and Variants . . . 17
4.3 Evaluation Protocols . . . 18
4.4 Computational Resources . . . 19

Chapter 5 Results . . . 20
5.1 Classification Performance . . . 20
5.1.1 MNIST . . . 20
5.1.2 CIFAR-10 . . . 21
5.1.3 ImageNet-100 . . . 21
5.2 Robustness . . . 22
5.2.1 Adversarial Attack Results . . . 22
5.2.2 Loss Landscape Analysis . . . 23
5.2.3 Hessian Spectrum Analysis . . . 24
5.3 Interpretability of Flex Routing . . . 25
5.3.1 Binariness . . . 25
5.3.2 Conv Ratio–Binariness Evolution and Correlation . . . 26
5.4 Spiking Dynamics and Sparsity . . . 28
5.4.1 Firing Rate Analysis . . . 28
5.4.2 Raster Plot Analysis . . . 28

5.4.3 Temporal Sparsity Dynamics . . . 29

Chapter 6 Discussion . . . 31
6.1 Analysis of Results . . . 31
6.2 Neuroscience Connection . . . 32

Chapter 7 Conclusions and Future Work . . . 33
7.1 Summary and Conclusions . . . 33
7.2 Limitations and Future Works . . . 34

Appendix A Comparison with Prior Dynamic Routing . . . 35

Appendix B Theoretical Analysis . . . 36
B.1 Certifiability . . . 36
B.2 Interval Bound Propagation . . . 37
B.3 Mask Sensitivity . . . 38

Appendix C Model Architectures and Hyperparameters . . . 39
C.1 VGG Architecture Tables . . . 39
C.1.1 VGG-6 for MNIST . . . 39
C.1.2 VGG-11 for CIFAR-10 . . . 40
C.1.3 VGG-16 for ImageNet-100 . . . 40
C.2 Training Hyperparameters . . . 40
C.2.1 MNIST . . . 40
C.2.2 CIFAR-10 . . . 40
C.2.3 ImageNet-100 . . . 42

Appendix D Supplementary Experimental Results . . . 46
D.1 Top-1 Classification Accuracy Results . . . 46
D.2 Additional Robustness Results . . . 48
D.2.1 Adversarial Attacks . . . 48
D.2.2 Loss Landscapes . . . 49
D.3 Additional Interpretability Results . . . 51
D.3.1 Binariness . . . 51
D.3.2 Conv Ratio–Binariness Evolution and Correlation . . . 52
D.4 Other Results . . . 55

Appendix E Computational Resources and Reproducibility 56
E.1 Hardware and HPC Platform 56
E.2 Software Environment 57
E.3 Code Availability 57

List of References 58

List of Figures

2.1 Invariance vs. selectivity . . . 3

3.1 Framework of PlasticFlex . . . 7

4.1 MNIST dataset snapshot . . . 15
4.2 CIFAR-10 dataset snapshot . . . 16
4.3 ImageNet dataset snapshot . . . 17

5.1 MNIST balanced training curves . . . 20
5.2 CIFAR-10 balanced training curves . . . 21
5.3 ImageNet-100 balanced training curves . . . 22
5.4 Adversarial robustness on CIFAR-10 . . . 23
5.5 Comparison of loss landscapes across six model variants . . . 24
5.6 Hessian spectrum density of six model variants . . . 25
5.7 Binariness evolution on CIFAR-10 . . . 26
5.8 CIFAR10-ANN-FLEX Conv Ratio–Binariness Dashboard . . . 27
5.9 CIFAR10-SNN-FLEX Conv Ratio–Binariness Dashboard . . . 27
5.10 CIFAR10-SNN-BP-FLEX Conv Ratio–Binariness Dashboard . . . 27
5.11 Firing rate distributions of four spiking model variants . . . 28
5.12 Raster plots of spike activity across layers . . . 29
5.13 Raster plots of the first convolutional layer . . . 30
5.14 Temporal sparsity dynamics under surrogate backpropagation . . . 30

4.1 MNIST top-1 training curves . . . 47
4.2 CIFAR-10 top-1 training curves . . . 47
4.3 ImageNet-100 top-1 training curves . . . 48
4.4 Adversarial robustness on MNIST . . . 48
4.5 Adversarial robustness on CIFAR-10 . . . 49
4.6 Adversarial robustness on ImageNet-100 . . . 49
4.7 Comparison of loss landscapes across six model variants on MNIST . . . 50
4.8 Comparison of loss landscapes across six model variants on CIFAR-10 . . . 50
4.9 Comparison of loss landscapes across six model variants on ImageNet-100 . . . 51
4.10 Binariness evolution on MNIST . . . 51
4.11 Binariness evolution on CIFAR-10 . . . 52
4.12 Binariness evolution on ImageNet-100 . . . 52
4.13 MNIST-ANN-FLEX Conv Ratio–Binariness Dashboard . . . 52
4.14 MNIST-SNN-FLEX Conv Ratio–Binariness Dashboard . . . 53
4.15 MNIST-SNN-BP-FLEX Conv Ratio–Binariness Dashboard . . . 53
4.16 CIFAR10-ANN-FLEX Conv Ratio–Binariness Dashboard . . . 53
4.17 CIFAR10-SNN-FLEX Conv Ratio–Binariness Dashboard . . . 54
4.18 CIFAR10-SNN-BP-FLEX Conv Ratio–Binariness Dashboard . . . 54
4.19 ImageNet100-ANN-FLEX Conv Ratio–Binariness Dashboard . . . 54
4.20 ImageNet100-SNN-FLEX Conv Ratio–Binariness Dashboard . . . 55
4.21 ImageNet100-SNN-BP-FLEX Conv Ratio–Binariness Dashboard . . . 55

List of Tables

2.1 Dynamic families and corresponding selectivity–invariance trade-off . . . 6

1.1 Comparison with prior dynamic routing methods 35

3.1 Baseline VGG-6 architecture for MNIST 39
3.2 Flex-enabled VGG-6 for MNIST 40
3.3 Baseline VGG-11 architecture for CIFAR-10 41
3.4 Flex-enabled VGG-11 for CIFAR-10 42
3.5 Baseline VGG-16 architecture for ImageNet-100 43
3.6 Flex-enabled VGG-16 for ImageNet-100 44
3.7 MNIST variants and differing hyperparameters 45
3.8 CIFAR-10 variants and differing hyperparameters 45
3.9 ImageNet-100 variants and differing hyperparameters 46

Chapter 1

Introduction

1.1 Background & Motivation

Biological sensory systems face a recurring trade-off: they need to capture fine local details while also staying stable when inputs vary.[1, 2] In vision, early cortical neurons are tuned to precise spatial features, whereas downstream neurons often generalize across shifts in position or scale.[3, 4] This balance between selectivity and invariance is not fixed but adapts depending on the context, task, and statistics of the input.[5]

Convolutional neural network (CNN) borrows this hierarchical idea but implement it rigidly.[6] Standard architectures alternate between convolution and pooling layers.[7] Pooling discards detail to gain invariance, [8] while convolution keeps local precision.[9] Once trained, the information flow in a CNN becomes fixed: data always passes through the same sequence of operations regardless of input content. This is efficient but ignores the possibility of adapting the information pathway itself.[10]

From a neuroscience and information-theoretic perspective, this rigidity is a simplification. [11] Real circuits route signals dynamically, reallocating bandwidth to regions of interest while allowing other regions to undergo more abstract processing.[12] In effect, the brain does not treat information flow as a single predetermined channel but as something that can be reorganized depending on the stimulus. Introducing this property into artificial networks could improve robustness and also provide a more realistic platform for studying biologically inspired computation.[13, 14]

1.2 Problem Statement

Despite progress in adaptive mechanisms such as attention [15] and dynamic convolution [16], most CNNs still apply the same operation to all spatial locations within a layer.[17] This global uniformity prevents the network from balancing selectivity and invariance in a spatially adaptive way. From an information-theoretic angle, the model commits to a single path of transformation, with no capacity to alter routing once the weights are

fixed. Existing routing methods rarely quantify how such rigidity impacts robustness, interpretability, or formal verifiability.[18, 19] In spiking neural network (SNN), where computation unfolds over time, the lack of routing flexibility is even more restrictive, limiting both efficiency and the diversity of information flow.[20]

1.3 Research Objective

This work introduces PlasticFlex, a routing module that lets each spatial location decide between convolution and pooling. In contrast to a single fixed channel of transformation, PlasticFlex allows multiple candidate pathways and learns where to allocate information flow. The design runs in both ANN and SNN settings without changing the backbone, enabling controlled comparisons. The objectives are: (i) test whether adaptive routing improves accuracy and robustness across datasets; (ii) measure interpretability through mask sensitivity analysis; (iii) derive theoretical bounds showing routing does not break certifiability; (iv) examine how spiking activations interact with routing to affect sparsity and efficiency.

1.4 Contributions

Unlike prior dynamic routing modules that mainly adjust computation capacity, PlasticFlex places the routing decision at the convolution–pooling interface, directly governing the selectivity–invariance trade-off while remaining certifiable.

The main contributions are:

(i) PlasticFlex: a pixel-level routing layer between convolution and pooling, preserving locality and feature semantics, with closed-form bounds.

(ii) Unified ANN–SNN Backbone: one architecture supports spiking and non-spiking activations, enabling direct comparisons across training regimes.

(iii) Interpretable Routing Indicators: convolution ratio and mask binariness track robustness and curvature, linking routing behaviour to stability.

(iv) Robustness and Stability Gains: Flex improves robustness AUC, smooths convergence, and yields accuracy gains over baselines.

(v) Systematic Evaluation: ablations across datasets, depths, and activation types isolate the effect of local routing versus fixed conv–pool pipelines.

Chapter 2

Related Works

2.1 Selectivity–Invariance Balance

Selectivity captures sensitivity to diagnostic features; invariance preserves identity under nuisance transformations.[3, 7] Classical CNNs hard-code this balance via convolution followed by pooling; once trained, the same trade-off is applied everywhere.[21, 22] Biology does not fix this compromise: early areas emphasise selectivity, downstream areas accumulate invariance, and the allocation shifts with context.[23] This motivates architectures that let the network decide, per input and location, how much detail to keep versus discard.[3]

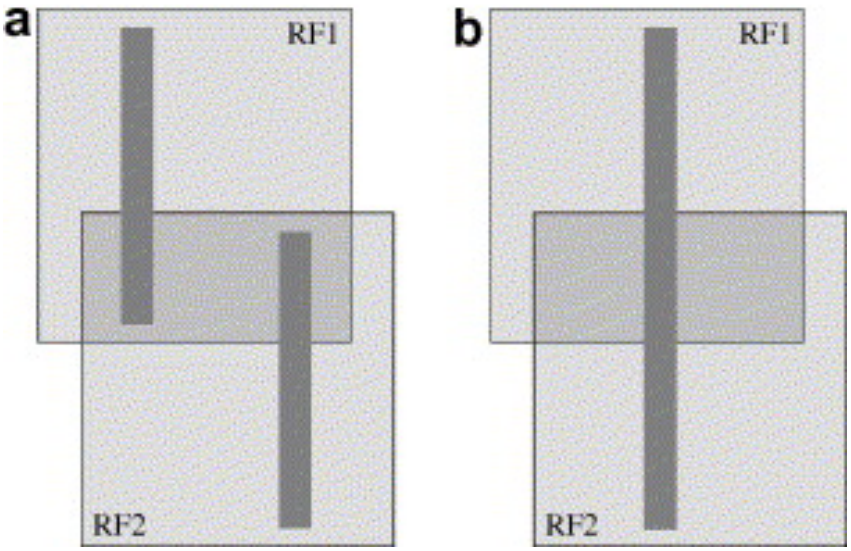


Figure 2.1 Invariance vs. selectivity.[24] Two units with phase-invariant responses can be indistinguishable under different arrangements, illustrating why fixed pooling can hide task-relevant structure.

2.2 Dynamic Routing: Mechanisms and Limits

Dynamic routing replaces uniform processing with input- or region-conditional paths.[16] Conditional computation activates filters or branches by learned gates.[25] Masking and attention mechanisms select kernels, scales, or resolutions on the fly.[26] Capsule-style routing aggregates part–whole votes to form equivariant groupings, but can be brittle and costly. Training such discreteness requires proxies: soft gates bias toward easy shortcuts,[27] straight-through tricks can be unstable,[28] and policy-based updates add variance and overhead.[29] Most methods improve adaptiveness yet offer no direct handle on *how* routing reallocates selectivity versus invariance across space and depth.

2.3 Transformer-Era Dynamic Computation

Vision Transformers [30] introduced token- and head-level decisions that prune patches, vary depth, and route tokens through experts. Adaptive Computation Time varies layer usage by confidence, decoupling budget from depth.[31] Mixture-of-Experts gating [32] sends different regions to specialized submodules, increasing capacity without uniform compute. These strategies demonstrate that routing is not only architectural but also budget-aware, and that decision granularity (token, head, block, expert) matters for stability and generalisation. However, they seldom make the selectivity–invariance tension explicit, nor do they quantify how routing choices reshape this balance. [33]

2.4 Dynamic Mechanisms for Robustness

Routing can improve robustness by preserving informative detail where needed and down-weighting spurious cues elsewhere.[30, 32] It can also introduce brittle decision boundaries if gates flip under small perturbations. Certifiability requires controlling Lipschitz constants and monotone structure; naive routing complicates both.[34] Information-theoretic views frame routing as regulating the flow and compression of task-relevant content, linking invariance to controlled information loss.[35]

2.5 Neuroscience Signals for Flexible Routing

Cortical systems redirect signals with task and attention, reweighting pathways to prioritise detail or invariance as demands change.[36, 5] Map-based navigation exhibits context-dependent remapping, highlighting adaptive circuit-level routing.[37]

2.6 SNNs as a Substrate for Adaptive Computation

Spiking networks communicate with events, introducing temporal coding and energy-friendly sparsity aligned with neuromorphic hardware.[38, 39] They integrate membrane potentials until thresholds trigger spikes, offering a natural substrate for time-varying gating and routing.[2, 14] Training remains hard due to non-differentiable spikes; surrogate gradients and conversions mitigate but obscure functional attributions.[40, 41, 42, 10] Combining SNN dynamics with learnable routing promises fine-grained control of detail preservation versus compression under resource constraints.[43]

2.7 Positioning PlasticFlex

PlasticFlex targets the trade-off directly: it routes between convolution and pooling within the same backbone at per-location granularity. The module exposes an explicit pathway that decides when to keep structure (selectivity) and when to compress (invariance). Gates are trained end-to-end and are compatible with both ANN and SNN activations, enabling a unified architecture without re-design.[40, 41, 42] Because routing choices sit exactly at the convolution–pooling interface, their effect on robustness and generalisation can be measured via invariance- and curvature-related probes.[7, 35] Compared to conditional computation and attention masks, PlasticFlex ties routing to the specific operation that historically creates invariance, rather than adding parallel branches that are harder to attribute.[25, 26] Compared to capsule-style agreement, PlasticFlex avoids iterative routing loops and reduces sensitivity to optimization instabilities.[16, 28] In SNN mode, the same routing interface couples with spike-driven sparsity to study joint effects on detail preservation, robustness, and efficiency.[38, 39]

2.8 A Compact Comparison

Table 2.1 Dynamic families and their treatment of the selectivity–invariance trade-off.

Family	**Decision granularity**	**What is routed**	**Main limitation**
Fixed CNNs	None	Conv → Pool (fixed)	Uniform trade-off; inflexible[21, 22]
Conditional Masks	Filter/branch	Kernels or compute per region	STE instability; unclear attribution[25, 27, 28]
Capsules	Part–whole votes	Part–whole grouping	Iterative; costly; brittle[16]
Transformer dynamics	Token/block/ expert	Depth, tokens, experts	SI balance implicit; uncertifiable[33, 30]
SNN dynamics	Spike timing/sparsity	Event-driven compute	Training difficulty; attribution gap[40, 41]
PlasticFlex	**Pixel-level**	**Detail vs. compression**	**Needs stability-aware training**

A broader comparison with representative dynamic routing methods is provided in Appendix A.

2.9 Takeaway

Most existing dynamic methods succeed in making networks adaptive, but they rarely address the core trade-off between selectivity and invariance directly. PlasticFlex is designed to intervene exactly at the convolution–pooling step where this trade-off is set. Because of this placement, its effect can be quantified, it integrates naturally with both ANN and SNN backbones, and its impact on robustness and stability can be examined in a straightforward way.

Chapter 3

Methodology

3.1 Overview of the Approach

PlasticFlex replaces a convolutional layer with a two-branch module: one branch performs convolution, the other pooling. At each spatial location, a mask decides the mix between the two outputs. The mask is trainable, so routing remains adaptive after training rather than collapsing to a single fixed path. Because both branches preserve locality and downsampling semantics, the module can drop into existing CNN backbones without altering dimensions. Its convex mixing form also admits closed-form output bounds, keeping compatibility with standard certification methods, as shown in Figure 3.1

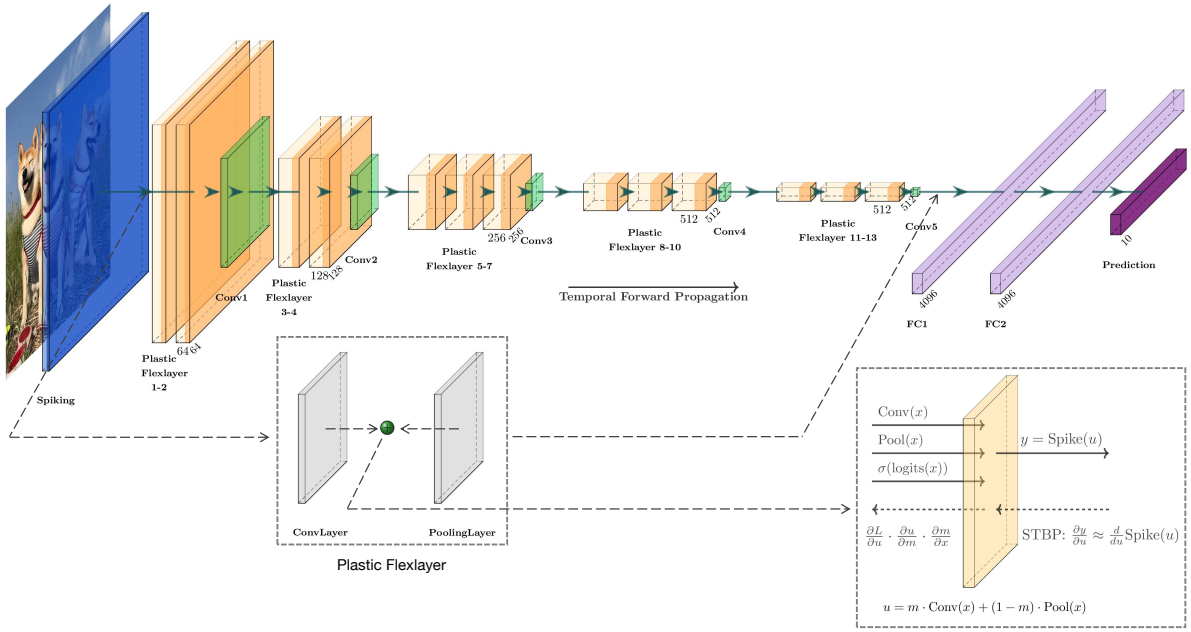


Figure 3.1 Framework of PlasticFlex.

The framework further separates routing from activation. A first switch controls routing behaviour, deciding whether the mask is frozen or learned. A second switch controls activation dynamics, selecting either a conventional ReLU or a spiking activation with surrogate gradients. The same backbone can therefore be run in ANN or SNN mode,

and with fixed or adaptive routing. This design allows systematic comparison of routing flexibility and spiking effects under identical architectures.

3.2 Network Architecture

The backbone follows VGG-style designs in three depth variants: VGG6, VGG11, and VGG16. Each convolutional block is implemented using PlasticFlex instead of a fixed convolution or max-pooling. The PlasticFlex module computes a standard convolution and a max-pooling operation in parallel, then combines their outputs via the spatial mask derived from local features. Fully connected layers are placed after the PlasticFlex blocks, projecting to the dataset's class space. For complete model architectures and hyper-parameters, please see Appendix C.

3.3 Flex Routing Mechanism

Each layer is implemented as a PlasticFlex module with two parallel branches: a convolution branch $F_{\text{conv}}(\cdot)$ and a pooling branch $F_{\text{pool}}(\cdot)$. A spatial routing mask $M \in [0,1]^{H\times W}$ decides, for every location (i,j), how much information flows through each branch. The mask is obtained from learnable logits Z derived from the input features, followed by an activation function $\sigma_{\text{mask}}(\cdot)$ such as sigmoid or hard-sigmoid:

$$M = \sigma_{\text{mask}}(Z), \quad Z = f_{\text{mask}}(X), \tag{3.1}$$

where X is the input, and $f_{\text{mask}}(\cdot)$ is a learnable mapping.

The output Y is a convex combination of the two branches:

$$Y = M \odot F_{\text{conv}}(X) + (1 - M) \odot F_{\text{pool}}(X), \tag{3.2}$$

with $\odot$ denoting element-wise multiplication. This formulation allows different spatial positions within the same feature map to follow different routes, making feature extraction adaptive rather than fixed.

To analyse routing behaviour, two statistics are monitored during training and evaluation:

(i) Convolution ratio: proportion of positions routed to the convolution branch,

$$r_{\text{conv}} = \frac{1}{HW} \sum_{i=1}^{H} \sum_{j=1}^{W} \mathbb{1}\{M_{i,j} > 0.5\}. \tag{3.3}$$

(ii) Binariness: degree to which mask values approach $\{0, 1\}$,

$$b_{\text{mask}} = 1 - \frac{4}{HW} \sum_{i=1}^{H} \sum_{j=1}^{W} M_{i,j}(1 - M_{i,j}). \tag{3.4}$$

PlasticFlex forward algorithm can be seen in Algorithm 1.

Algorithm 1: PlasticFlex Forward

Input: Input feature map X

Output: Output feature map Y

$C \leftarrow F_{\text{conv}}(X)$ `// convolution branch`

$P \leftarrow F_{\text{pool}}(X)$ `// pooling branch (aligned in channels)`

if *mask is frozen* **then**

  $M \leftarrow M_{\text{frozen}}$ `// stored binary mask`

else

  $Z \leftarrow f_{\text{mask}}(X)$ `// logits from features`

  $M \leftarrow \sigma_{\text{mask}}(Z)$ `// sigmoid / hard-sigmoid / STE`

$Y \leftarrow M \odot C + (1 - M) \odot P$

return Y

3.4 Mask Behaviour Control

Each PlasticFlex module produces its routing mask from learnable logits, transformed by an activation function $\sigma_{\text{mask}}(\cdot)$. Two activation types are supported:

(i) Sigmoid: a continuous mapping in $(0, 1)$, allowing smooth gradients for stable optimisation.

(ii) Hard-sigmoid: a piecewise-linear approximation that saturates near 0 and 1, encouraging more discrete routing behaviour during training.

The learnability of the mask can be configured in two modes:

(i) Trainable mask: the mask parameters remain learnable throughout training, enabling dynamic adaptation of routing at all epochs.

(ii) Frozen mask: the mask is trained for an initial number of epochs, after which its parameters are fixed, enabling testing whether ongoing adaptation improves performance or stability.

To prevent mask collapse—a degenerate case where all positions route through only one branch—we add a quadratic regularisation term that penalises deviation from a target

routing ratio:

$$\mathcal{L}_{\text{mask}} = \lambda_{\text{mask}} \cdot \frac{1}{HW} \sum_{i=1}^{H} \sum_{j=1}^{W} (M_{i,j} - \tau)^2, \tag{3.5}$$

where λ_{mask} controls the regularisation strength and τ is a target ratio (default 0.5). Compared to L_1 penalties, this L_2 form stabilises the global distribution without forcing the mask into a single branch.

Optionally, a binarisation regulariser can be introduced to encourage discrete routing decisions:

$$\mathcal{L}_{\text{bin}} = \lambda_{\text{bin}} \cdot \frac{1}{HW} \sum_{i=1}^{H} \sum_{j=1}^{W} 4\, M_{i,j}(1 - M_{i,j}), \tag{3.6}$$

where λ_{bin} is typically set to zero unless explicitly stated.

During analysis, the evolution of the mask is monitored through:

(i) Convolution ratio: r_{conv} over epochs, indicating routing preference changes.

(ii) Binariness: b_{mask} over epochs, reflecting the discreteness of routing decisions.

Tracking r_{conv} and b_{mask} across epochs reveals how routing preferences evolve and whether the mask stabilises under different control strategies.

3.5 Spiking Behaviour

The framework supports both ANN and SNN modes by selecting the activation function after each layer: ReLU for ANN, SpikingActivation for SNN. In ANN mode, a standard ReLU is used. In SNN mode, the activation is replaced by a SpikingActivation unit that introduces membrane potential dynamics and threshold-based firing.

We implement two operation modes:

(i) Surrogate gradient mode: At each time step t, the membrane potential U_t is compared against a threshold θ (set to 1.0 in all experiments). A spike is generated when the threshold is crossed:

$$S_t = \mathbb{1}\{U_t > \theta\}. \tag{3.7}$$

Because the derivative of this step function is zero almost everywhere, we use a surrogate gradient during training:

$$\frac{\partial S_t}{\partial U_t} \approx \alpha \cdot \tanh(U_t), \tag{3.8}$$

where α controls the slope, preserves binary spike dynamics in the forward pass while enabling gradient-based training.

(ii) Continuous mode: Instead of hard thresholding, the output is bounded through a clamping function:

$$S_t = \text{clip}(U_t, 0, 1). \tag{3.9}$$

This yields a continuous analogue of spiking behaviour, removes the need for surrogate gradients but loses the explicit event-driven character of spiking.

In both modes, the spiking activation is applied channel-wise at each spatial location. To evaluate temporal behaviour, the network is unrolled over T discrete steps, and outputs are averaged:

$$Y = \frac{1}{T} \sum_{t=1}^{T} f_{\text{SNN}}(X_t). \tag{3.10}$$

This design enables direct ANN–SNN comparison under the same backbone and highlights the trade-off between biological fidelity and training stability.

Algorithm 2: SNN Temporal Unrolling (if $T > 1$)

Input: Input x, steps T

Output: Prediction y

$y_{\text{acc}} \leftarrow 0$ **for** $t = 1, \ldots, T$ **do**

  $x_t \leftarrow \text{Encode}(x, t)$ `// rate/latency/identity`

  $h_t \leftarrow \text{Backbone} + \text{PlasticFlex}(x_t)$ $s_t \leftarrow \text{SpikingActivation}(h_t)$ `// STBP or non-STBP`

  $y_{\text{acc}} \leftarrow y_{\text{acc}} + \text{Head}(s_t)$

$y \leftarrow \text{Aggregate}(y_{\text{acc}})$ `// mean or sum`

return y

3.6 Training Procedure

All models are trained using stochastic gradient descent with momentum (SGD+Momentum), a momentum coefficient of 0.9, and weight decay of 5×10^{-4}. The initial learning rate η_0 is set to 0.1. A cosine annealing schedule gradually reduces the learning rate to zero over the course of training:

$$\eta_t = \eta_{\min} + \frac{1}{2}(\eta_0 - \eta_{\min}) \left[1 + \cos\left(\frac{\pi t}{T} \right) \right], \tag{3.11}$$

where t is the current epoch and T is the total number of epochs.

Data preprocessing follows standard protocols for each dataset:

(i) CIFAR-10 / ImageNet100: random cropping with 4-pixel padding, random horizontal flipping, and per-channel normalization.

(ii) MNIST: zero-padding to 32×32 resolution for compatibility with VGG backbones, followed by normalization to zero mean and unit variance.

All models are trained with cross-entropy loss:

$$\mathcal{L}_{\text{CE}} = -\frac{1}{N}\sum_{n=1}^{N}\sum_{c=1}^{C} y_{n,c}\log p_{n,c}, \tag{3.12}$$

where $y_{n,c}$ is the ground-truth label and $p_{n,c}$ is the predicted probability for class c.

For SNN variants trained with surrogate gradients, the backpropagation step applies the surrogate derivative defined in Eq. 3.8, replacing the zero derivative of the spike function. Non-surrogate SNNs and ANN variants use standard backpropagation without modification.

To enable interpretability analysis, mask statistics are recorded at the end of each epoch for all PlasticFlex modules. For robustness evaluation, adversarial attacks are applied post-training to the final model checkpoints.

Algorithm 3: Training Step with Optional Mask Freezing

Input: Mini-batch (x, y), epoch e

Output: Updated parameters

$\hat{y} \leftarrow \text{Model}(x)$ `// forward with PlasticFlex + activation`

$\mathcal{L} \leftarrow \mathcal{L}_{\text{CE}}(\hat{y}, y)$ **if** *mask regulariser is enabled* **then**

  $\mathcal{L} \leftarrow \mathcal{L} + \lambda_{\text{mask}} \cdot \frac{1}{HW}\sum_{i,j}(M_{i,j} - \tau)^2$ **if** $\lambda_{\text{bin}} > 0$ **then**

    $\mathcal{L} \leftarrow \mathcal{L} + \lambda_{\text{bin}} \cdot \frac{1}{HW}\sum_{i,j} 4M_{i,j}(1 - M_{i,j})$

Backprop($\nabla\mathcal{L}$) ; OptimizerStep()

if $e = E_f$ *and mask is learnable* **then**

  $M_{\text{frozen}} \leftarrow \mathbb{1}\{M \geq 0.5\}$ `// binarize and store`

  freeze mask parameters and skip logits branch henceforth

3.7 Theoretical Analysis

This section examines whether the flexibility introduced by PlasticFlex changes its fundamental safety properties, and whether such changes affect compatibility with standard certification methods. Complete derivations are provided in Appendix B.

3.7.1 Safety and Tightness

The PlasticFlex output at location i is

$$y_i \;=\; m_i C_i + (1 - m_i)P_i \;=\; P_i + m_i(C_i - P_i). \tag{3.13}$$

Let $D_i = C_i - P_i$. With intervals

$$P_i \in [l_p, u_p], \quad D_i \in [l_d, u_d], \quad m_i \in [l_m, u_m],$$

the closed-form certified bound is

$$y_i \in \big[l_p + \min\{l_m l_d, l_m u_d, u_m l_d, u_m u_d\},\ u_p + \max\{l_m l_d, l_m u_d, u_m l_d, u_m u_d\}\big]. \tag{3.14}$$

Monotonicity. From $\partial y/\partial C = m$, $\partial y/\partial P = 1 - m$, and $\partial y/\partial m = D$, one can determine which endpoints attain the extrema (e.g. if $D \geq 0$, the lower bound always uses $m = l_m$).

3.7.2 Correlation-Aware Bounds

Since C_i and P_i share the same receptive field, treating them as independent produces loose bounds. By tracking

$$y_i = P_i + m_i D_i,$$

and propagating intervals for (P_i, D_i, m_i) instead of (C_i, P_i, m_i), one obtains strictly narrower certified ranges. This requires no modification of IBP frameworks.

3.7.3 Interval Propagation in Practice

Given intervals

$$P_i \in [l_p, u_p], \quad D_i \in [l_d, u_d], \quad m_i \in [l_m, u_m],$$

the propagation rule is simply

$$[l_p, u_p],\ [l_d, u_d],\ [l_m, u_m] \quad \mapsto \quad [l_y, u_y] \tag{3.15}$$

with $[l_y, u_y]$ computed by Eq. (3.14). This closed-form update is composable across layers, ensuring scalability.

3.7.4 Linear Relaxations (CROWN-Compatible)

For linear relaxations, bilinear terms can be bounded with McCormick envelopes. For instance,

$$z = mC, \quad w = mP, \tag{3.16}$$

$$y = z + P - w, \tag{3.17}$$

and z, w each satisfy four linear inequalities determined by the box corners. Thus y inherits valid affine bounds with only constant overhead.

3.7.5 Lipschitz View and Certified Stability

Let the Lipschitz constants of the convolution, pooling, and mask branches be L_C, L_P, L_m. Then for perturbation δx,

$$|\delta y| \;\le\; u_m L_C \|\delta x\| + (1 - l_m) L_P \|\delta x\| + \sup|D|\, L_m \|\delta x\|, \tag{3.18}$$

where $\sup|D| = \max(|l_d|, |u_d|)$. This decomposition separates branch smoothness from routing smoothness, clarifying which factors improve robustness.

3.7.6 Mask Sensitivity

The basic sensitivity of PlasticFlex to its routing mask is

$$S_i := |C_i - P_i| = |D_i|. \tag{3.19}$$

Two refinements make this measure more diagnostic:

$$R_i := m_i(1 - m_i)\,|D_i|, \quad \text{(decision leverage)} \tag{3.20}$$

$$S_i^\star := \frac{|D_i|}{|C_i| + |P_i| + \varepsilon}, \quad \text{(scale-normalised)}. \tag{3.21}$$

A regulariser

$$\mathcal{L}_{\text{route}} = \lambda \sum_i R_i$$

can penalise uncertain, high-impact routing.

3.7.7 Complexity and Scalability

Each PlasticFlex unit can be bounded in $O(1)$ time with the (P, D, m) formulation. Linear relaxations add a fixed number of inequalities per site. Thus the method scales comparably to standard convolution and pooling layers.

3.7.8 Summary

PlasticFlex preserves certifiability, admits correlation-aware bounds, integrates seamlessly with IBP and CROWN, provides a Lipschitz-based stability decomposition, and introduces sensitivity measures that can guide interpretation and training. Full derivations are given in Appendix B.

Chapter 4

Experiments

4.1 Datasets

We conducted experiments on three datasets with increasing complexity to assess the performance, interpretability, and robustness of the proposed architecture.

4.1.1 MNIST

MNIST consists of 70,000 grayscale handwritten digit images (28×28), with 60,000 for training and 10,000 for testing. Despite its simplicity, it provides a clean setting to validate spiking activations and flexible routing under low-dimensional visual patterns.

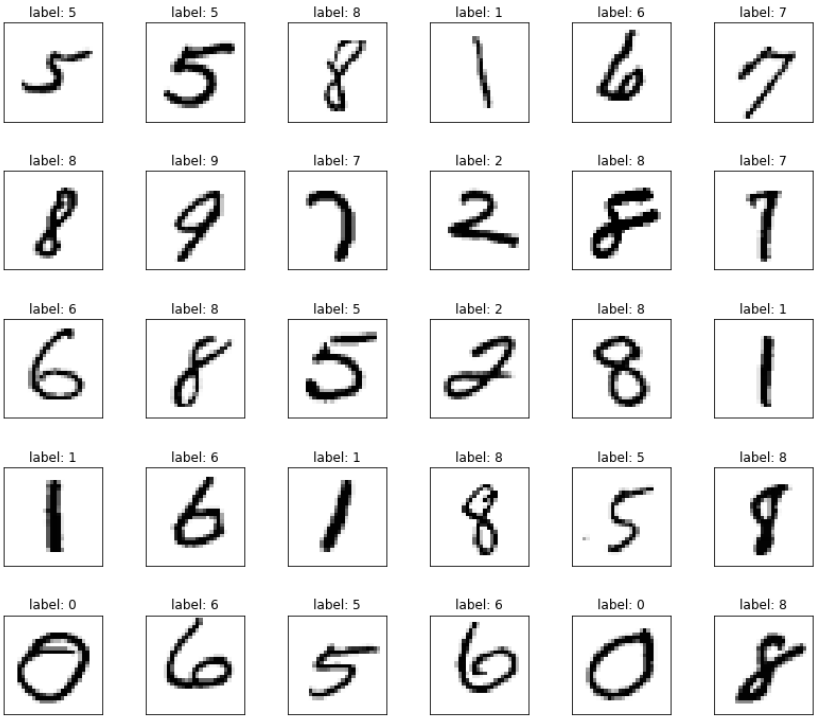


Figure 4.1 MNIST dataset snapshot.

4.1.2 CIFAR-10

CIFAR-10 contains 60,000 RGB images (32×32) evenly split across 10 categories, with 50,000 for training and 10,000 for testing. Compared to MNIST, it introduces colour and higher variability, providing a benchmark to test how Flex routing adapts under richer spatial statistics.

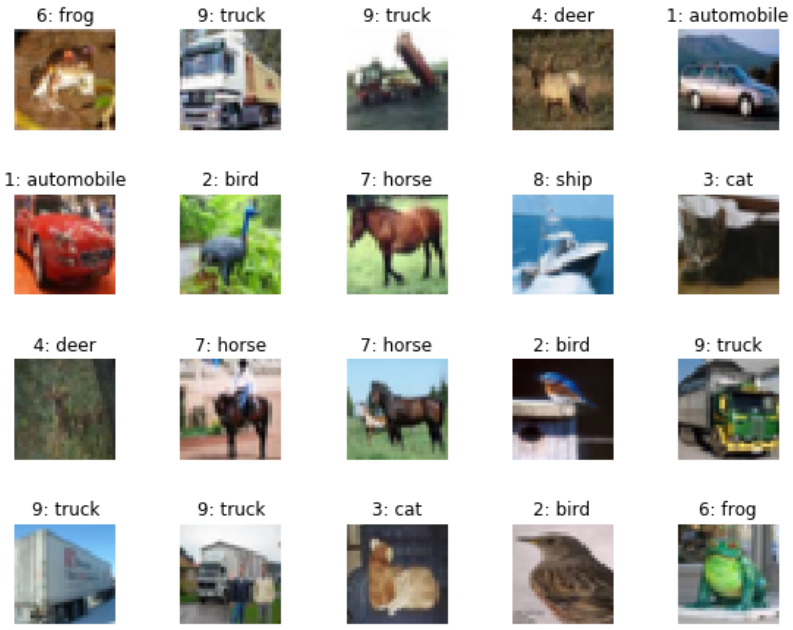


Figure 4.2 CIFAR-10 dataset snapshot.

4.1.3 ImageNet-100

ImageNet-100 is a balanced subset of 100 classes from ImageNet, with about 1,300 training and 50 validation images per class (224×224). It evaluates the scalability of Flex routing to higher-resolution inputs and a larger label space, and tests whether interpretability and robustness trends generalise beyond small-scale datasets.

4.2 Experimental Setup

We evaluate PlasticFlex under controlled settings by varying three main factors: (i) backbone depth (VGG6, VGG11, VGG16), (ii) routing flexibility (standard conv+pool vs. PlasticFlex with learnable or frozen masks), and (iii) activation mode (ANN with ReLU, SNN with surrogate gradients, or SNN with non-surrogate clamping). This design

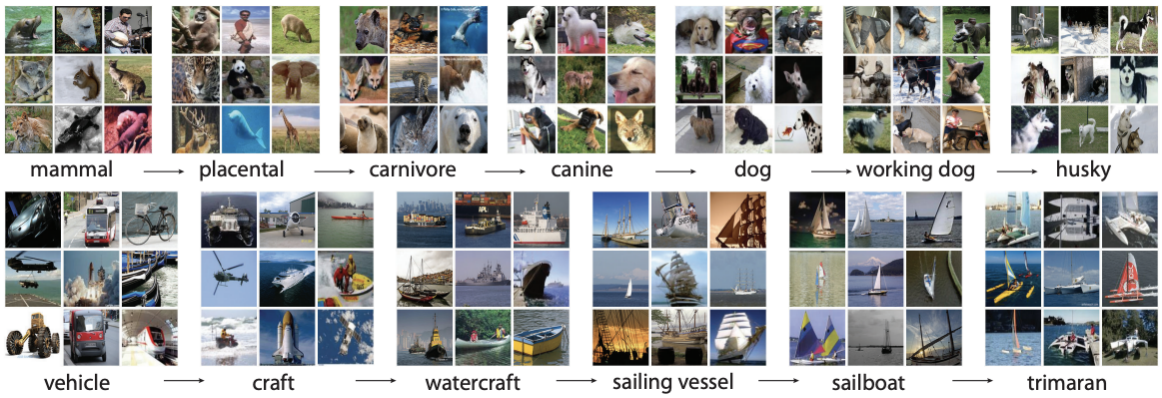


Figure 4.3 ImageNet dataset snapshot.

isolates the contributions of structural flexibility and spiking behaviour across different model capacities and dataset scales (MNIST, CIFAR-10, ImageNet-100).

4.2.1 Training Details

All models are trained with stochastic gradient descent (SGD) using an initial learning rate of 0.1, momentum 0.9, and weight decay 5×10^{-4}. The learning rate decays by 0.1 at scheduled epochs depending on dataset size. Training runs for 120 epochs on MNIST, 120 epochs on CIFAR-10, and 180 for ImageNet-100. Batch sizes are 128 (MNIST, CIFAR-10) and 256 (ImageNet-100). Weights are initialised with Kaiming normal initialisation.

For SNN variants, activations are unrolled over $T = 10$ timesteps with a spiking threshold $\theta = 1.0$. Surrogate training uses slope $\alpha = 0.3$, while non-surrogate mode replaces binary spikes with a continuous clamping function. PlasticFlex masks are initialised uniformly and regularised with an L_2 penalty to a target ratio $\tau = 0.5$ (see Sec. 3.4 for details). All experiments are run on NVIDIA A100 GPUs with mixed precision.

4.2.2 Baselines and Variants

We use a factorised design so that each choice can be isolated while keeping depth and channel widths fixed.

1. Backbone depth. VGG6 / VGG11 / VGG16.

2. Routing (fixed vs. flexible).

    - Standard CNN: all layers are conventional convolutions (with pooling where needed).

- PlasticFlex: every convolutional layer is replaced by PlasticFlex.

3. Mask parameterisation and schedule.
   - Parameterisation: sigmoid; hard-sigmoid with STE; deterministic binary mask.
   - Schedule: trainable for all epochs; or frozen after a pre-defined epoch E_f (binarise at 0.5 and stop updating).
4. Activation mode (ANN vs. SNN).
   - ANN: ReLU.
   - SNN (STBP): binary spikes with surrogate gradients.
   - SNN (continuous): bounded continuous surrogate (no surrogate gradient).
5. Datasets. MNIST, CIFAR-10, ImageNet-100.

This factorisation allows main effects and interactions (routing, mask behaviour, neural dynamics) to be assessed under a consistent backbone.

4.3 Evaluation Protocols

We evaluate models along four dimensions:

1. Accuracy and balanced accuracy. The standard top-1 accuracy is defined as

$$\mathrm{Acc} = \frac{1}{N}\sum_{n=1}^{N} \mathbb{1}\{\hat{y}_n = y_n\}, \tag{4.1}$$

   where $\hat{y}_n$ is the predicted class and y_n is the ground truth. Balanced accuracy averages recall across classes to address class imbalance:

$$\mathrm{Acc}_{\mathrm{bal}} = \frac{1}{C}\sum_{c=1}^{C} \frac{TP_c}{TP_c + FN_c}, \tag{4.2}$$

   where TP_c and FN_c denote true positives and false negatives for class c.

2. Robustness. Adversarial robustness is measured under FGSM, PGD, and SPGD attacks. The FGSM perturbation is given by

$$x' = x + \epsilon \cdot \mathrm{sign}\left(\nabla_x \mathcal{L}(f_\theta(x), y)\right), \tag{4.3}$$

where ϵ is the perturbation budget. PGD applies multiple projected gradient steps:

$$x^{t+1} = \Pi_{\mathcal{B}(x,\epsilon)} \left(x^t + \alpha \cdot \text{sign} \left(\nabla_x \mathcal{L}(f_\theta(x^t), y) \right) \right), \tag{4.4}$$

where $\Pi_{\mathcal{B}(x,\epsilon)}$ projects onto the ϵ-ball around x. SPGD uses stochastic gradient estimates with random direction vectors u_k:

$$\nabla_x \mathcal{L}(f_\theta(x), y) \approx \frac{1}{K} \sum_{k=1}^{K} \frac{\mathcal{L}(f_\theta(x + \delta u_k), y) - \mathcal{L}(f_\theta(x - \delta u_k), y)}{2\delta} u_k, \tag{4.5}$$

which enables gradient-free adversarial optimisation.

3. Interpretability. We monitor routing statistics (convolution ratio, mask binariness), gradient flow, and Hessian spectra to assess how routing decisions and spiking dynamics influence learning.

4. Sparsity and efficiency. For SNNs, the average spike rate is recorded. For PlasticFlex, the degree of mask binarisation indicates the extent of computational sparsity, serving as a proxy for efficiency.

4.4 Computational Resources

All experiments were conducted on the Imperial College London High Performance Computing (HPC) facility (Research Computing Service, DOI: 10.14469/hpc/2232). The compute nodes used in this study were equipped with dual-socket AMD EPYC 7742 processors (64 cores per socket, 256 hardware threads in total) and 1.0 TiB of system memory, running Linux kernel 4.18. GPU-accelerated training was performed on NVIDIA A100 GPUs (40 GB memory). Jobs were submitted via the PBS scheduler with typical resource requests of 2 CPU cores, 8 GB RAM, and 1 GPU per job, with a maximum walltime of 32 hours.

The complete hardware and software resources are listed in Appendix E

Chapter 5

Results

5.1 Classification Performance

5.1.1 MNIST

Figure 5.1 shows the training and validation curves for all MNIST variants. Across architectures, all models reached 99.6% balanced accuracy, and 100.0% top-1 accuracy (Appendix D.1) on the validation set, indicating that neither the spiking activation nor the flexible routing impaired classification performance on this dataset.

A consistent effect was observed in the loss dynamics: models with the PlasticFlex mechanism showed more stable validation loss, even after training accuracy saturated. This suggests that adaptive routing between convolution and pooling paths provides a regularising effect, reducing oscillations during optimisation.

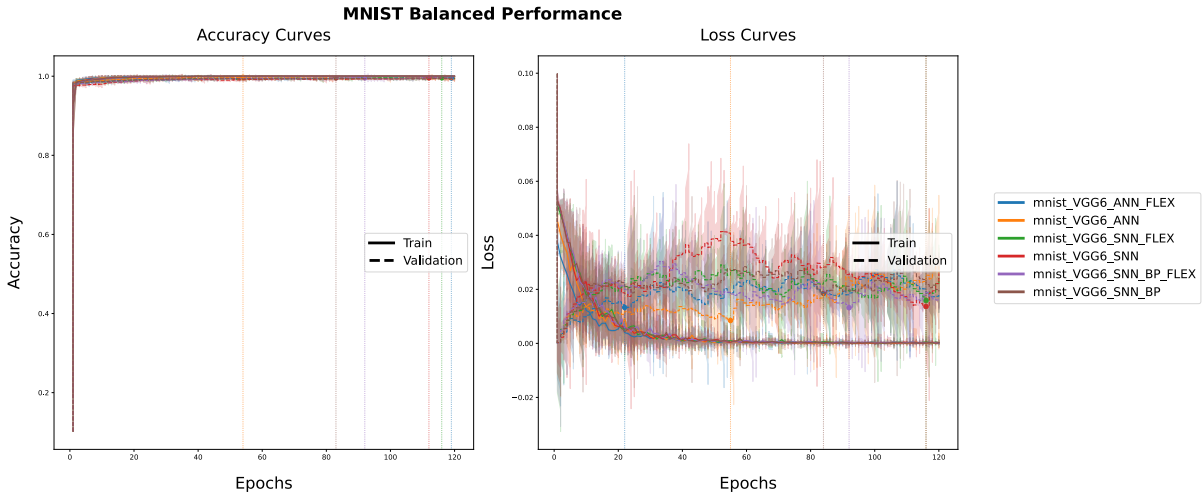


Figure 5.1 MNIST balanced training curves (VGG6 backbone). Left: balanced accuracy for training and validation. Right: balanced loss for training and validation.

5.1.2 CIFAR-10

Figure 5.2 shows the training dynamics on CIFAR-10 with VGG11 backbones. Among ANN models, the Flex-enabled variant reached the highest validation balanced accuracy (88.8%), slightly above the standard ANN baseline (87.9%). A similar trend appeared in SNNs: the spiking network with Flex achieved 86.0%, compared to 84.7% for the best non-Flex spiking baseline (SNN-BP).

Across both ANN and SNN settings, the flexible routing mechanism consistently improved balanced accuracy. Although the gains were modest, the effect was reproducible, suggesting that spatially adaptive routing between convolution and pooling helps preserve discriminative information. The loss curves further show smoother convergence for Flex variants, indicating that Flex stabilises optimisation while maintaining accuracy.

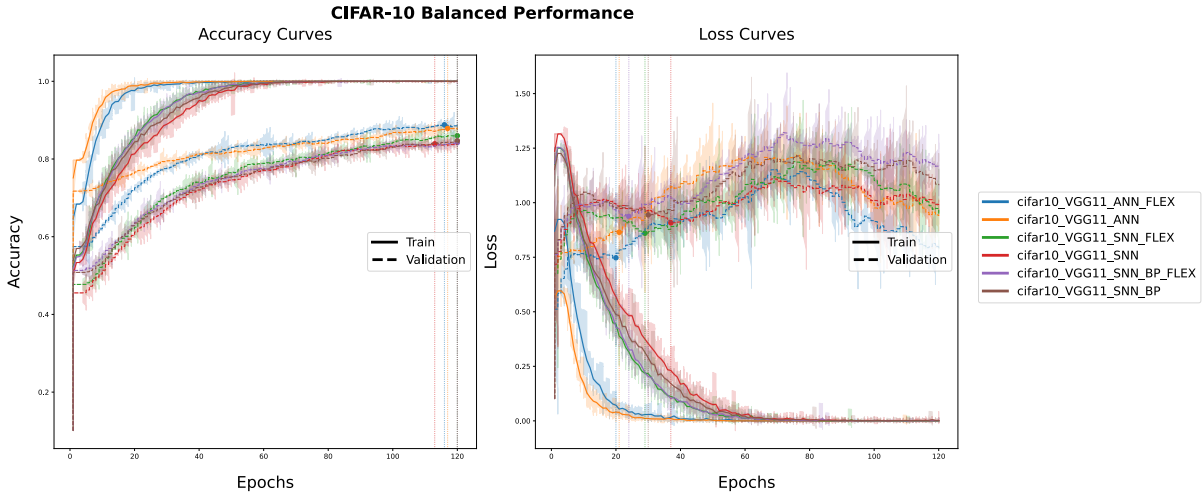


Figure 5.2 CIFAR-10 training curves (VGG11 backbone). Left: balanced accuracy for training and validation. Right: balanced loss for training and validation.

5.1.3 ImageNet-100

Figure 5.3 shows the training and validation curves on ImageNet-100. Validation accuracy saturates around 78.5%–80.6% for ANN variants and slightly lower for SNNs, showing that the models cannot fully capture the complexity of ImageNet-100. The gap between training and validation curves is larger than in MNIST and CIFAR-10, indicating weaker generalisation. Loss values converge but plateau at higher levels, consistent with the more difficult task.

Flex provides a modest but consistent benefit. ANN+Flex and SNN+Flex converge more smoothly and reach slightly higher validation accuracy than their baselines.

SNN+BP with Flex also trains more stably than SNN+BP without Flex, suggesting that routing reduces instability. Performance on ImageNet-100 remains limited, but Flex improves optimisation and gives small accuracy gains.

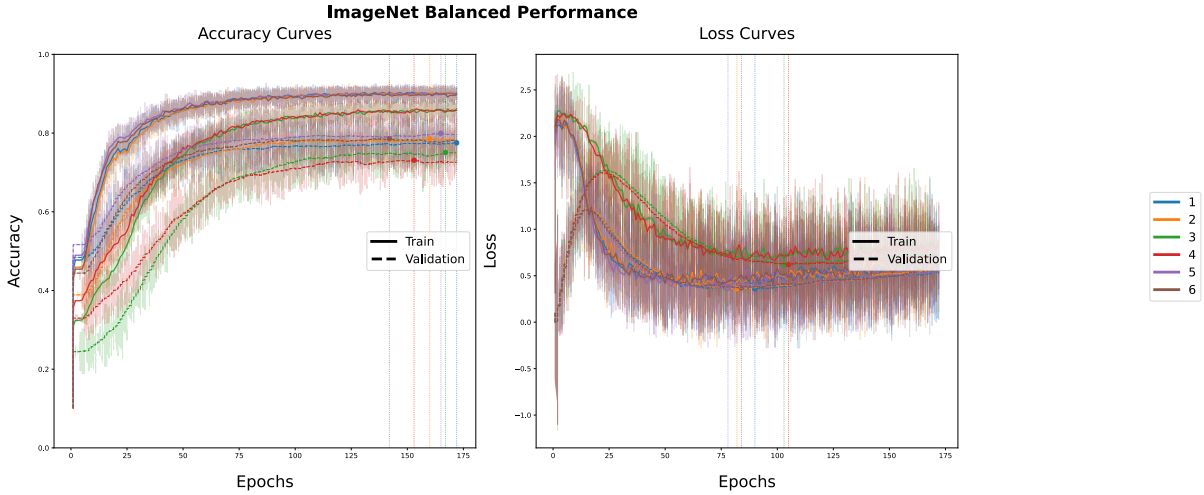


Figure 5.3 ImageNet-100 training curves (VGG16 backbone). Left: balanced accuracy for training and validation. Right: balanced loss for training and validation.

5.2 Robustness

This section reports robustness results. For clarity and space efficiency, we focus on CIFAR-10 as the main benchmark, since it balances complexity and tractability better than MNIST (too simple) and ImageNet-100 (too large-scale for detailed plots). Comprehensive results on MNIST and ImageNet-100 are provided in the Appendix D.2.

5.2.1 Adversarial Attack Results

Figure 5.4 shows adversarial accuracy on CIFAR-10 under FGSM, PGD, and SPGD for $\epsilon \leq 0.3$.

Spiking variants are more robust than ANN across attacks. Adding Flex improves both families; the gains are larger under iterative attacks (PGD, SPGD), where Flex models sustain higher accuracy over a wider ϵ range.

These results suggest complementary effects: temporal sparsity from spiking and adaptive routing from Flex. SNN-BP-Flex is the most robust, indicating that surrogate-gradient training combined with flexible routing better resists gradient-based perturbations.

For reporting clarity, we additionally compute the AUC over $\epsilon \in [0, 0.3]$ (shown below each plot), which provides a compact robustness measure consistent with the curve trends.

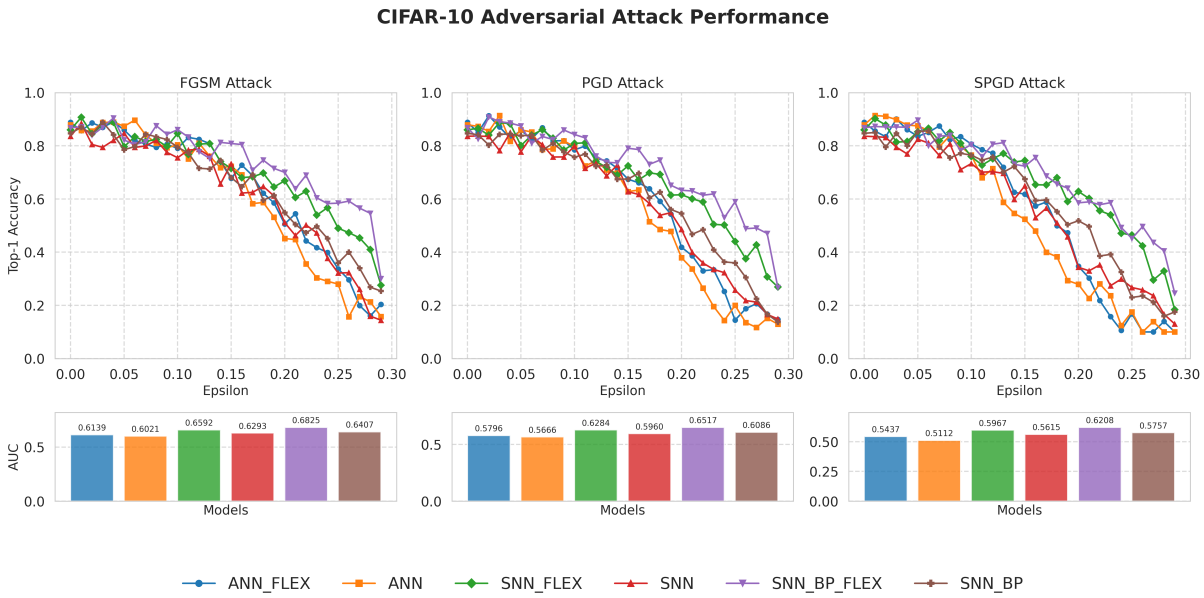


Figure 5.4 CIFAR-10 adversarial robustness. Top: accuracy under FGSM, PGD, and SPGD attacks with increasing perturbation ϵ. Bottom: AUC scores summarising robustness across ϵ. Flex-enhanced SNNs consistently achieve the highest robustness.

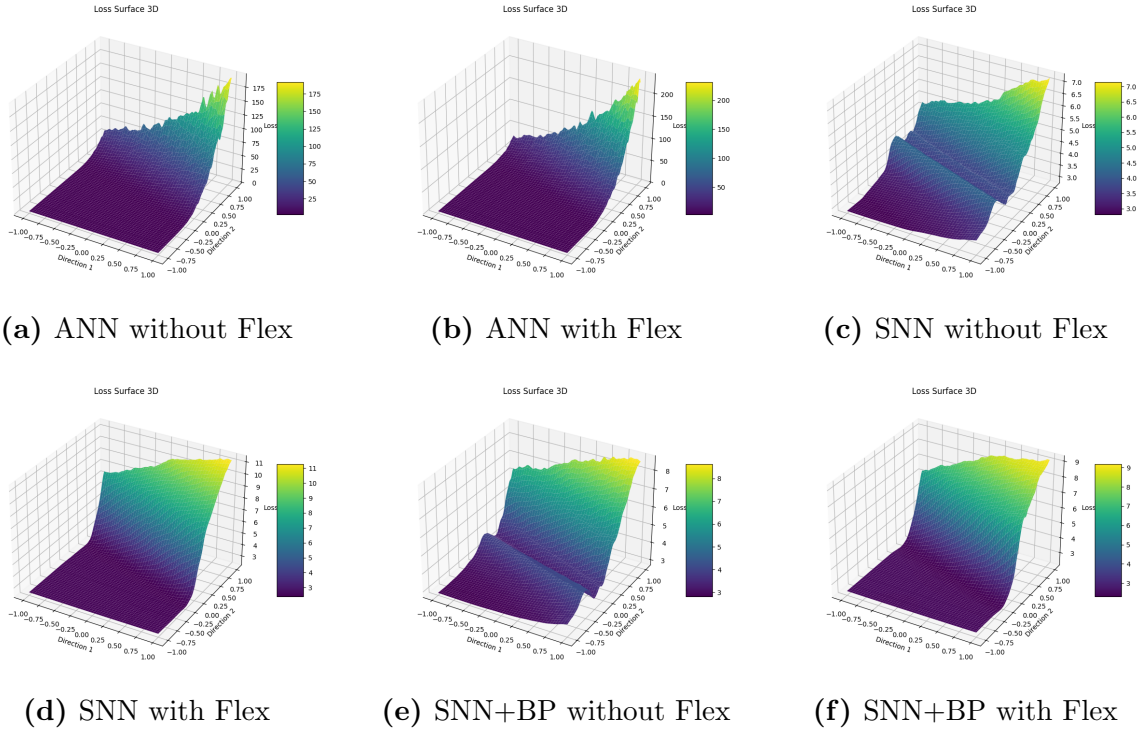
5.2.2 Loss Landscape Analysis

To examine stability and generalization, we visualized the loss surfaces of different variants along two random parameter directions (Figure 5.5).

The ANN baseline (Figure 5.5a) shows a broad basin but also sharp cliffs, revealing sensitivity in certain directions. Introducing Flex (Figure 5.5b) yields a smoother surface with gentler slopes, suggesting convergence to flatter minima.

For SNNs, the basic model (Figure 5.5c) exhibits ridge-like structures, indicating uneven sensitivity. Flex reduces these ridges (Figure 5.5d), while backpropagation-through-time (SNN+BP, Figure 5.5e) further regularizes the surface. The combination of BP and Flex (Figure 5.5f) produces the flattest and most stable basin among spiking models.

In general, Flex consistently promotes flatter landscapes in both ANN and SNN settings, and BP training further strengthens this effect in spiking networks.



(a) ANN without Flex **(b)** ANN with Flex **(c)** SNN without Flex

(d) SNN with Flex **(e)** SNN+BP without Flex **(f)** SNN+BP with Flex

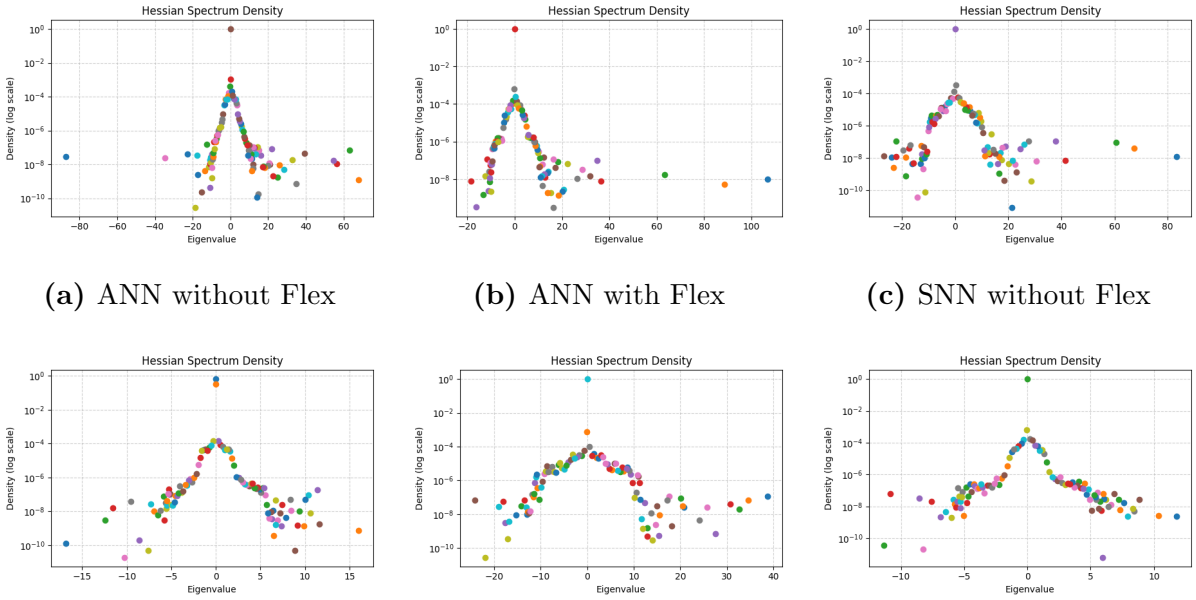
Figure 5.5 Comparison of loss landscapes across six model variants on CIFAR-10 dataset.

5.2.3 Hessian Spectrum Analysis

We analysed the Hessian eigenvalue distributions of trained models to characterise local curvature (Figure 5.6).

For ANN baselines (Figure 5.6a), the spectrum spans a wide range with long negative tails, reflecting sharp directions. Introducing Flex (Figure 5.6b) contracts the distribution toward zero, alleviating part of this instability.

In spiking networks, the plain SNN (Figure 5.6c) exhibits extended positive and negative tails, consistent with rugged curvature. Flex reduces this spread (Figure 5.6d), shifting most eigenvalues closer to zero. Backpropagation training (Figure 5.6e) further suppresses the negative tail, and its combination with Flex (Figure 5.6f) yields the most compact spectrum, with eigenvalues tightly clustered around zero. Flex consistently dampens extreme eigenvalues across both ANN and SNN families, while backpropagation provides an additional stabilizing effect in spiking models.



(a) ANN without Flex **(b)** ANN with Flex **(c)** SNN without Flex

(d) SNN with Flex **(e)** SNN+BP without Flex **(f)** SNN+BP with Flex

Figure 5.6 Hessian spectrum density of six model variants on CIFAR-10 dataset.

5.3 Interpretability of Flex Routing

This section reports interpretability analyses of Flex routing. Due to space constraints, we present results on CIFAR-10; full results on MNIST and ImageNet-100 are provided in the appendix D.4.

5.3.1 Binariness

Figure 5.7 compares the temporal evolution of activation binariness across ANN and SNN variants. In ANN_FLEX, binariness stabilizes around 0.22 with limited variation across layers, indicating that Flex layers in ANN mainly serve structural adjustment without amplifying sparsity effects. In contrast, both SNN_FLEX and SNN_BP_FLEX exhibit higher binariness (∼0.25–0.30), consistent with the spike-driven nature of SNNs. Notably, SNN_FLEX shows stronger divergence between layers, with several deeper layers maintaining binariness above 0.28, whereas SNN_BP_FLEX presents a more balanced distribution, suggesting that backpropagation mitigates extreme sparsification. Across all models, a hierarchical trend emerges: early layers retain lower binariness, while later layers gradually increase, in agreement with the biological selectivity–invariance principle, where shallow stages encode detailed selectivity and deeper stages abstract sparse, invariant representations.

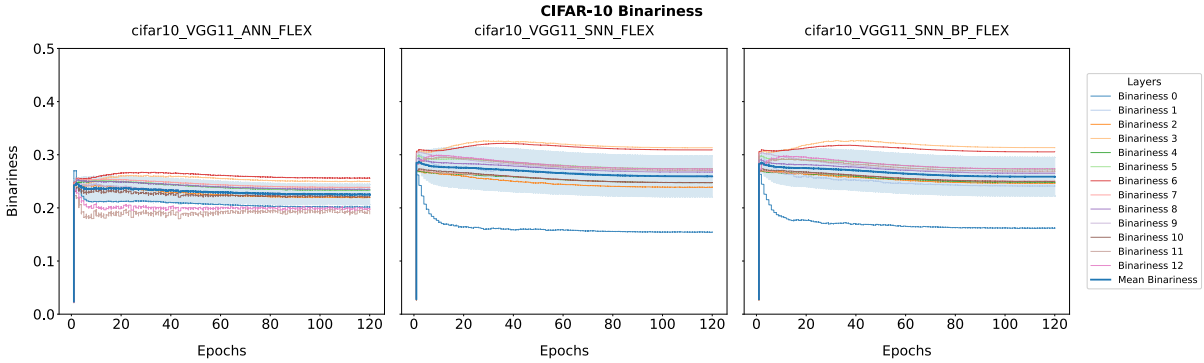


Figure 5.7 Binariness evolution on CIFAR-10 across different architectures.

5.3.2 Conv Ratio–Binariness Evolution and Correlation

Figure 5.8, Figure 5.9, Figure 5.10 compare the evolution of convolutional ratio (Conv Ratio) and activation binariness across ANN and SNN models. Several consistent patterns emerge. First, layer-wise trajectories reveal that early layers converge to higher Conv Ratios, while deeper layers stabilize at lower values. This hierarchical separation is in line with the biological principle of selectivity–invariance, where early stages prioritize feature selectivity and later stages promote invariant abstraction. Second, in the global trend, ANN_FLEX maintains an overall Conv Ratio around 0.4 with stable binariness near 0.23, and no clear coupling is observed (Pearson $r \approx 0.015$). By contrast, SNN models exhibit higher Conv Ratios ($\sim$ 0.5–0.6) together with increased binariness ($\sim$ 0.26–0.30). Notably, both SNN_BP_FLEX and SNN_FLEX show negative correlations between Conv Ratio and binariness (Pearson $r = -0.372$ and $r = -0.200$, respectively), suggesting that higher convolutional dominance suppresses binary activation patterns. Taken together, these results indicate that while Flex layers in ANN primarily act as structural regulators, in SNNs they additionally mediate the balance between convolutional processing and spike-based sparsity, with BP training accentuating this coupling effect.

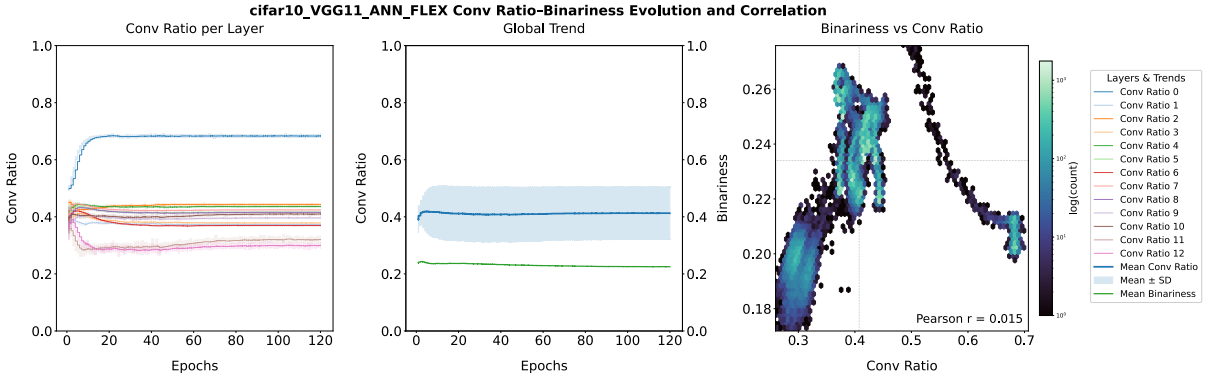


Figure 5.8 CIFAR10-VGG11-ANN-FLEX Conv Ratio–Binariness Dashboard. The left panel shows the evolution of convolution ratios across layers, the middle panel illustrates the global trend with mean ± SD and mean binariness, and the right panel depicts the correlation between binariness and convolution ratio.

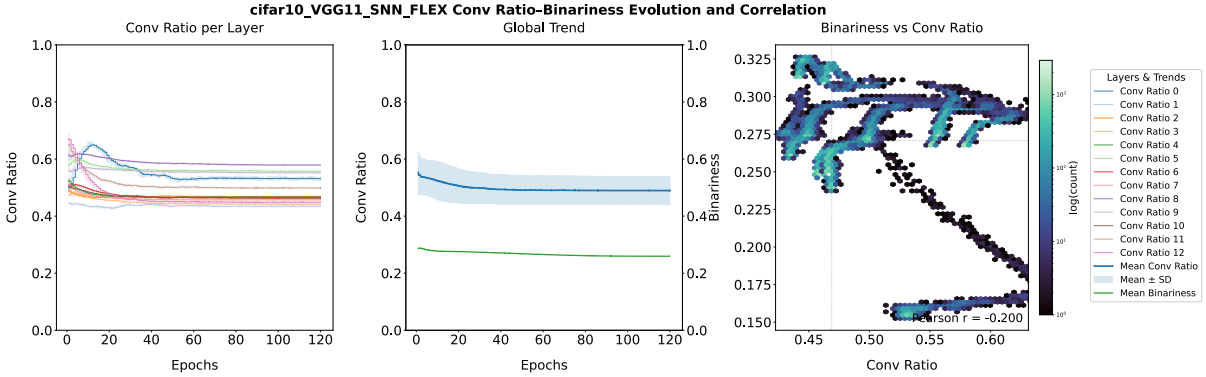


Figure 5.9 CIFAR10-VGG11-SNN-FLEX Conv Ratio–Binariness Dashboard.

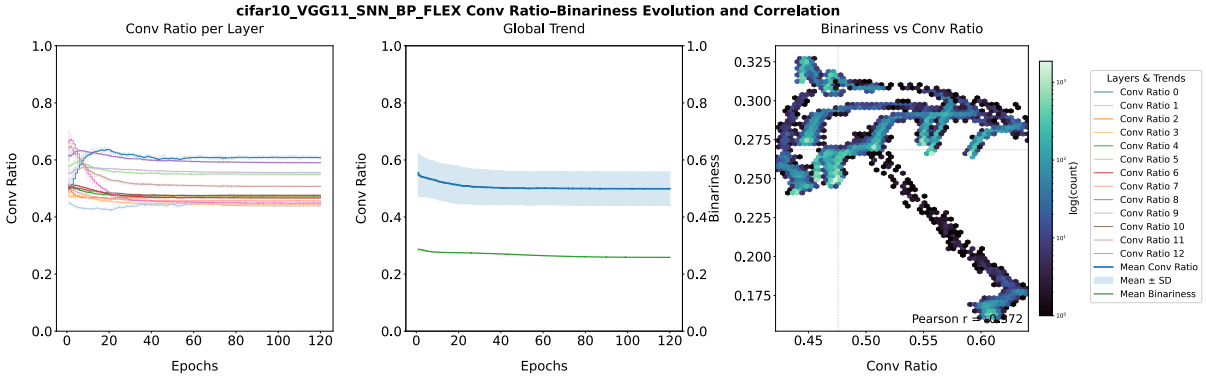
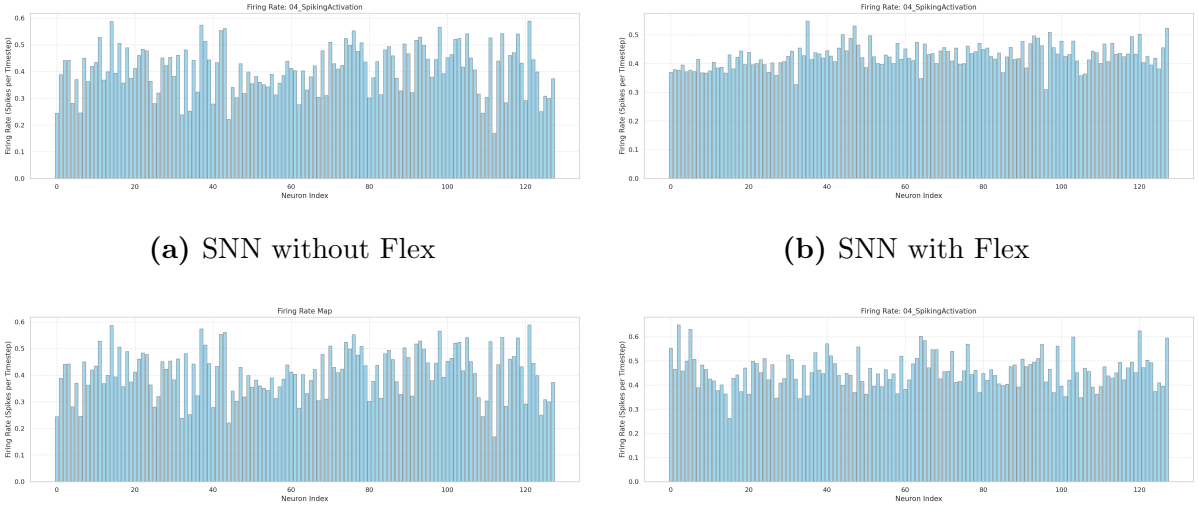


Figure 5.10 CIFAR10-SNN-BP-FLEX Conv Ratio–Binariness Dashboard.

5.4 Spiking Dynamics and Sparsity

5.4.1 Firing Rate Analysis

Figure 5.11 shows firing rate distributions across spiking variants. The plain SNN (Figure 5.11a) displays broad variability, while adding Flex (Figure 5.11b) narrows the spread. With backpropagation (Figure 5.11c), overall firing shifts upward and the tail extends to higher rates. When Flex is combined with BP (Figure 5.11d), the distribution remains elevated but more regularised, indicating that Flex reduces variability in firing while BP increases average activity, and their combination achieves higher yet more controlled firing.



(a) SNN without Flex

(b) SNN with Flex

(c) SNN+BP without Flex

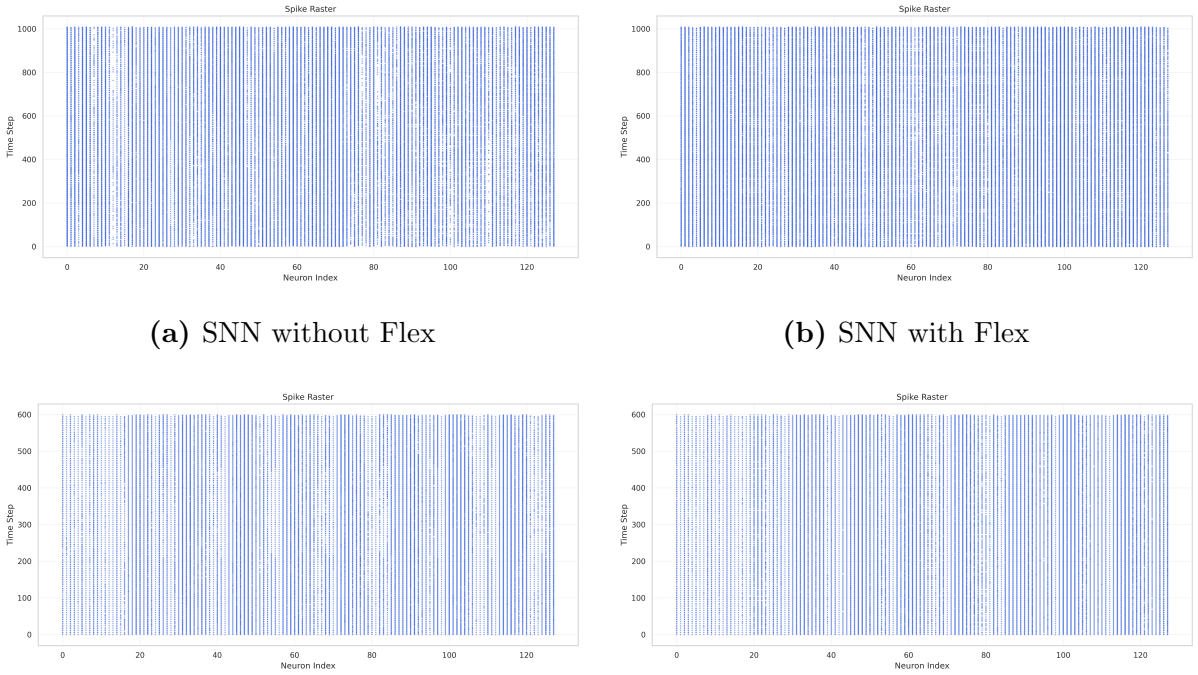
(d) SNN+BP with Flex

Figure 5.11 Firing rate distributions of four spiking model variants.

5.4.2 Raster Plot Analysis

We examined the spiking behavior of different network variants through raster plots. Figure 5.12 illustrates the spike activity across layers for four representative models. The baseline SNN without backpropagation (Figure 5.12a) exhibits dense firing with limited sparsity, while adding the Flex mechanism (Figure 5.12b) produces smoother and sparser activity across layers. When backpropagation with surrogate gradients is applied (Figure 5.12c), firing activity is heavily suppressed, with deeper layers showing almost no spikes. Introducing Flex in this setting (Figure 5.12d) restores a more balanced level of spiking, resulting in a structured and stable distribution across layers. These

observations suggest that Flex reduces redundant spikes and stabilises sparsity, whereas backpropagation alone tends to over-suppress activity.



(a) SNN without Flex

(b) SNN with Flex

(c) SNN+BP without Flex

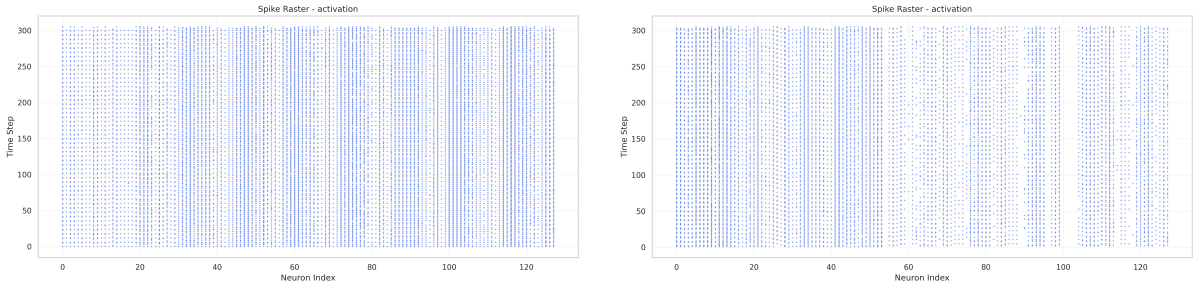
(d) SNN+BP with Flex

Figure 5.12 Raster plots of spike activity across layers. Flex reduces redundant firing and promotes structured sparsity, while backpropagation suppresses spikes more aggressively; combining both achieves a more balanced activity pattern.

To further examine layer-specific dynamics in more detail, we focused on the first convolutional layer of two models: SNN+BP without Flex and SNN+BP with Flex (Figure 5.13). This layer was chosen because it directly encodes input features and largely shapes the network's spiking behaviour. In the SNN+BP model without Flex, the raster plot shows relatively uniform firing across neurons, suggesting limited differentiation. By contrast, the SNN+BP with Flex variant exhibits a distinct "patchy" pattern, where certain neurons are highly active while others remain silent. This selective activation indicates that Flex not only regulates sparsity but also enhances functional differentiation within layers, potentially improving representational efficiency.

5.4.3 Temporal Sparsity Dynamics

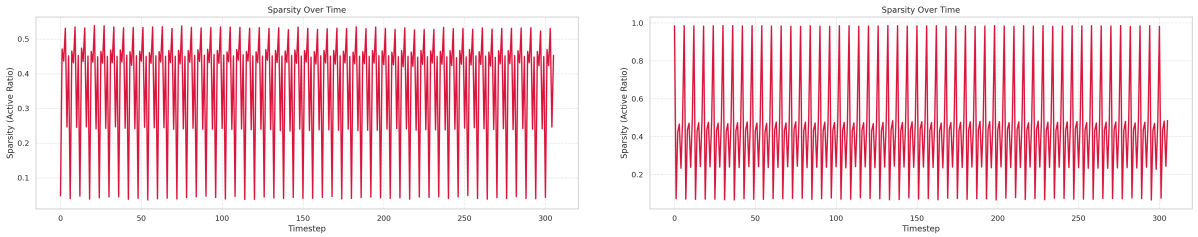
To analyse how Flex affects temporal behaviour under surrogate backpropagation, we compared the evolution of activity sparsity between SNN+BP and SNN+BP with Flex



(a) SNN+BP without Flex **(b)** SNN+BP with Flex

Figure 5.13 Raster plots of the first convolutional layer. Without Flex, neurons fire uniformly with little differentiation, whereas Flex induces a patchy pattern with selective activation, highlighting its role in promoting functional diversity.

(Figure 5.14). The baseline SNN+BP maintains a relatively stable active ratio around 0.4–0.5 across timesteps, indicating a strong global suppression that keeps firing sparse yet uniform. By contrast, the Flex-augmented model displays pronounced oscillations between near-silent and highly active states, suggesting that mask routing introduces temporal modulation of activity. This shows that Flex does not merely reduce spiking overall, but redistributes events selectively, creating a richer temporal structure in the activations.



(a) SNN+BP without Flex **(b)** SNN+BP with Flex

Figure 5.14 Temporal sparsity dynamics under surrogate backpropagation. Without Flex, firing remains uniformly sparse across time, while Flex introduces oscillatory modulation, alternating between suppressed and highly active states.

Chapter 6

Discussion

6.1 Analysis of Results

Across datasets and evaluation dimensions, the results consistently show that the proposed Flex mechanism enhances both stability and robustness without sacrificing accuracy. From a biological perspective, this effect can be linked to the principle of selectivity–invariance: early layers maintain higher convolutional ratios, preserving detailed selectivity of local features, while deeper layers favour pooling, promoting invariant abstraction. This hierarchical shift mirrors cortical processing, where early sensory areas encode fine-grained information and higher areas compress it into sparse, invariant codes.

The binariness and firing analyses further suggest that Flex acts as a regulator of sparsity. In artificial neural networks, Flex maintains moderate binariness, mainly shaping structural balance between convolution and pooling. In spiking networks, however, Flex interacts with spike-driven dynamics to modulate both spatial and temporal sparsity, reducing redundant firing while preserving discriminative signals. This behaviour reflects a form of efficient coding: by redistributing activity across neurons and timesteps, the network approximates information-theoretic principles such as entropy maximisation and redundancy reduction.

Loss landscape and Hessian analyses provide a complementary view. Models equipped with Flex consistently converge to flatter minima with spectra concentrated near zero, indicating reduced sensitivity to perturbations. In information-theoretic terms, such flatter solutions correspond to models that generalise by avoiding sharp decision boundaries, effectively trading representational capacity for robustness against input and parameter noise. The surrogate-gradient trained SNNs with Flex show the most compact spectra, suggesting that combining temporal sparsity with adaptive routing yields particularly stable solutions.

Taken together, these findings highlight that the Flex mechanism not only improves

classification and robustness in engineering terms, but also enforces constraints that resemble biological efficiency principles: selective routing of information, sparse and temporally modulated activity, and convergence to flatter, more stable representations. This alignment between computational performance and biological plausibility suggests that Flex-style adaptive modules may provide a principled way to bridge artificial and spiking neural systems under a unified framework.

6.2 Neuroscience Connection

The Flex routing mechanism parallels the role of cortical microcircuits that dynamically gate inputs through recurrent and feedforward pathways. Mask binariness reflects a process similar to synaptic switching, where local circuits stabilise into discrete routing states under repeated stimulation. The observed trade-off between convolutional ratio and binariness aligns with the balance of excitation and inhibition in cortical layers, which regulates both selectivity and sparsity. Temporal modulation of spiking activity resembles oscillatory gating in thalamocortical loops, where bursts and silences structure information flow across time. The flatter loss surfaces found in Flex models echo the stability of attractor states in recurrent cortical networks, supporting reliable computation under noisy conditions. Finally, the efficiency gains from reduced firing variability reflect an information-theoretic principle also seen in sensory coding, where redundancy is minimised while maintaining discriminability.

Chapter 7

Conclusions and Future Work

7.1 Summary and Conclusions

This thesis introduced PlasticFlex, a framework that extends conventional convolutional networks to support both ANN and SNN modes by replacing standard convolution layers with a dual-branch module—pooling and convolution combined via a learnable mask. This design allows the network to regulate the balance between information preservation and abstraction in a data-driven way.

Experiments on MNIST, CIFAR-10, and ImageNet-100 showed that Flex maintained accuracy with modest but consistent improvements in both ANN and SNN settings. In spiking models, Flex gave the highest robustness under gradient-based adversarial attacks, suggesting that temporal sparsity and spatially adaptive routing act as complementary defences.

Analysis of loss landscapes and Hessian spectra showed that Flex produced flatter minima and reduced extreme eigenvalues. These effects were present in both ANN and SNN models, most pronounced when combined with surrogate-gradient backpropagation. Stabilisation was also visible in firing rate distributions and raster plots: Flex reduced variability and encouraged structured sparsity.

Interpretability analyses revealed consistent layer-wise trends: early layers favoured convolution with lower binariness, later layers became sparser, in line with the selectivity–invariance principle. The negative correlation between convolution dominance and binariness in SNNs further indicated that Flex mediates a trade-off between detailed feature extraction and spike-based abstraction.

Taken together, these findings establish PlasticFlex as a design that integrates adaptive routing, spiking dynamics, and surrogate learning. Beyond accuracy, the framework contributes robustness, interpretability, and biologically grounded sparsity, suggesting a path towards architectures that balance efficiency, stability, and expressive power.

7.2 Limitations and Future Works

Although PlasticFlex shows consistent gains in robustness, interpretability, and stability, there are clear limitations. First, improvements on ImageNet-100 are modest: local routing helps on small datasets but does not close the generalisation gap on large-scale tasks. Second, adversarial robustness was only tested up to $\epsilon \leq 0.3$; stronger or adaptive attacks could expose weaknesses. Third, all benchmarks were image classification, so it remains unclear whether routing benefits carry over to sequential tasks such as speech decoding, reinforcement learning, or neuromorphic sensing. Fourth, the module was applied uniformly across layers; more selective placement or hierarchical parameterisation could improve efficiency without extra parameters. Finally, sparsity and efficiency were assessed only via firing rates and mask binarisation; a full account of energy efficiency would require neuromorphic deployment or detailed cost modelling.

Future work can build on these points. One direction is architectural: testing PlasticFlex in ResNet, ViT, or hybrid backbones to see whether the same advantages hold beyond VGG-style designs. Another is robustness: combining adaptive routing with adversarial training baselines (e.g. PGD-AT, TRADES) to assess whether gains persist under stronger defences. A third is hardware: implementing PlasticFlex on neuromorphic platforms to evaluate actual energy savings. Finally, closer integration with neuroscience is important. The observed trends in convolution ratio and binariness resemble cortical strategies for balancing selectivity and invariance, but this link remains speculative and should be tested against neural recordings.

An additional direction is a systematic horizontal comparison with other dynamic routing methods. This work focused on the SNN setting to examine biological plausibility and connect routing behaviour with neural computation. A fair comparison would require re-implementing other dynamic routing mechanisms in spiking form, which lies beyond the current scope but will be an important next step.

Appendix A

Comparison with Prior Dynamic Routing

Several dynamic routing mechanisms have been proposed in the literature, including dynamic convolution, CondConv, attention-based routing, mixture-of-experts, and capsule routing. Table 1.1 provides a concise comparison of these methods and highlights how PlasticFlex differs by placing the routing decision directly at the convolution–pooling interface, targeting the selectivity–invariance trade-off rather than only scaling computation.

Table 1.1 Comparison of PlasticFlex with representative dynamic routing mechanisms.

Method	**Decision granularity**	**What is routed**	**Main limitation**
Dynamic Conv[44]	Filter level	Kernel weights per input	High overhead; weak selectivity–invariance link
CondConv[44]	Filter/branch	Expert filters per sample	Extra parameters; limited interpretability
Attention Routing[15]	Token/patch	Features across heads or regions	Expensive; effect on invariance indirect
Mixture-of-Experts[32]	Expert layer	Subnetworks or blocks	Routing instability; high compute cost
Capsule Routing[45]	Part–whole votes	Grouped entity features	Iterative; brittle; costly optimisation
PlasticFlex	**Pixel-level**	Conv vs. pooling	Needs stability-aware training

Appendix B

Theoretical Analysis

Goal: To demonstrate that the PlasticFlex Layer maintains stability, interpretability, and certifiability despite its structural plasticity.

B.1 Certifiability

Step 1: Layer Definition. The PlasticFlex layer computes the output at each location $i \in \{1, \ldots, N\}$ as:

$$y_i = m_i \cdot C_i + (1 - m_i) \cdot P_i \tag{B.1}$$

where C_i and P_i denote the outputs of the convolutional and pooling paths, respectively, and $m_i \in [0, 1]$ is a routing mask.

Assume each component lies within known bounds:

$$C_i \in [l_c,\ u_c] \tag{B.2}$$

$$P_i \in [l_p,\ u_p] \tag{B.3}$$

$$m_i \in [l_m,\ u_m] \subseteq [0, 1] \tag{B.4}$$

Step 2: Affine Expansion. Rewriting Equation (B.1):

$$y_i = m_i(C_i - P_i) + P_i \tag{B.5}$$

Define $f(m, C, P) := m(C - P) + P$.

Step 3: Domain Geometry. The domain is the box

$$[m_l, u_m] \times [l_c, u_c] \times [l_p, u_p].$$

Since f is affine, its extrema occur at one of the $2^3 = 8$ vertices.

Step 4: Evaluate Vertices. Enumerating the 8 corner values:

$$
\begin{aligned}
v_1 &= f(l_m, l_c, l_p) = l_m(l_c - l_p) + l_p \\
v_2 &= f(u_m, l_c, l_p) = u_m(l_c - l_p) + l_p \\
v_3 &= f(l_m, u_c, l_p) = l_m(u_c - l_p) + l_p \\
v_4 &= f(u_m, u_c, l_p) = u_m(u_c - l_p) + l_p \\
v_5 &= f(l_m, l_c, u_p) = l_m(l_c - u_p) + u_p \\
v_6 &= f(u_m, l_c, u_p) = u_m(l_c - u_p) + u_p \\
v_7 &= f(l_m, u_c, u_p) = l_m(u_c - u_p) + u_p \\
v_8 &= f(u_m, u_c, u_p) = u_m(u_c - u_p) + u_p
\end{aligned}
$$

Step 5: Reduction to 4 Expressions. Because f is linear in both C and P, the extremal values can be summarised by 4 cases:

$$\mathcal{E}_1 = l_m \cdot l_c + (1 - l_m) \cdot l_p \tag{B.6}$$

$$\mathcal{E}_2 = u_m \cdot l_c + (1 - u_m) \cdot l_p \tag{B.7}$$

$$\mathcal{E}_3 = l_m \cdot u_c + (1 - l_m) \cdot u_p \tag{B.8}$$

$$\mathcal{E}_4 = u_m \cdot u_c + (1 - u_m) \cdot u_p \tag{B.9}$$

Step 6: Final Tight Bound.

$$y_i \in [\min(\mathcal{E}_1, \mathcal{E}_2, \mathcal{E}_3, \mathcal{E}_4),\ \max(\mathcal{E}_1, \mathcal{E}_2, \mathcal{E}_3, \mathcal{E}_4)] \tag{B.10}$$

This bound is closed-form, tight, and requires only 4 evaluations.

Step 7: Vectorisation. Element-wise generalisation to a tensor $\mathbf{y}$:

$$\mathbf{y} \in \Big[\min(\mathcal{E}_1, \mathcal{E}_2, \mathcal{E}_3, \mathcal{E}_4),\ \max(\mathcal{E}_1, \mathcal{E}_2, \mathcal{E}_3, \mathcal{E}_4)\Big]. \tag{B.11}$$

B.2 Interval Bound Propagation

Observation. The bound in Eq. (B.10) applies directly when the input intervals are obtained from a previous propagation step. Thus, the PlasticFlex unit can be inserted into an IBP pipeline without modification.

Propagation Rule. At each site:

$$[l_c, u_c],\ [l_p, u_p],\ [l_m, u_m] \ \mapsto\ [l_y, u_y] \text{ via Eq. (B.10)}. \tag{B.12}$$

Vectorisation. For tensor-valued features, this rule applies element-wise across (C, H, W). Hence the interval propagation through PlasticFlex is closed-form and composable across layers.

B.3 Mask Sensitivity

Step 1: Output Definition.

$$y_i = m_i C_i + (1 - m_i) P_i \tag{B.13}$$

Step 2: Local Derivative.

$$\frac{\partial y_i}{\partial m_i} = C_i - P_i \tag{B.14}$$

Step 3: Sensitivity Score.

$$S_i := |C_i - P_i| \tag{B.15}$$

Step 4: Global Average.

$$\bar{S} = \frac{1}{N} \sum_{i=1}^{N} |C_i - P_i| \tag{B.16}$$

Step 5: Gradient Signal.

$$\frac{\partial \mathcal{L}}{\partial m_i} = \frac{\partial \mathcal{L}}{\partial y_i} (C_i - P_i) \tag{B.17}$$

Hence S_i also governs mask trainability.

Step 6: Perturbation Bound. For perturbation $m_i \to m_i + \delta_i$:

$$|\Delta y_i| = |\delta_i| \cdot |C_i - P_i| = |\delta_i| \cdot S_i \tag{B.18}$$

Conclusion. The sensitivity score S_i captures both the interpretability of mask decisions and the magnitude of their effect under perturbations, linking directly to robustness and training dynamics.

Appendix C

Model Architectures and Hyperparameters

C.1 VGG Architecture Tables

This appendix lists the backbone architectures used in our experiments.

C.1.1 VGG-6 for MNIST

Table 3.1 Baseline VGG-6 architecture for MNIST.

Block	Layers
Block 1	Conv(3×3, 64) Conv(3×3, 128) MaxPool(2×2)
Block 2	Conv(3×3, 256) Conv(3×3, 256) MaxPool(2×2)
FC	FC1 (1024 units, Dropout) FC2 (512 units, Dropout) Output (10 classes)

Table 3.2 Flex-enabled VGG-6 for MNIST (all conv and pooling layers replaced by PlasticFlex).

Block	Layers
Block 1	PlasticFlex(64) PlasticFlex(128, stride 2)
Block 2	PlasticFlex(256) PlasticFlex(256, stride 2)
FC	FC1 (1024 units, Dropout) FC2 (512 units, Dropout) Output (10 classes)

C.1.2 VGG-11 for CIFAR-10

C.1.3 VGG-16 for ImageNet-100

C.2 Training Hyperparameters

C.2.1 MNIST

Shared settings. Unless otherwise noted, all MNIST runs use: VGG6 backbone; SGD with momentum 0.9 and weight decay 5×10^{-4}; cosine learning rate schedule with initial rate 0.1; batch size 128; dropout $p = 0.3$ after each fully connected layer; mask parameterisation: hard-sigmoid; logits mechanism: threshold; logits BatchNorm: enabled; spiking threshold 1.0 and membrane decay 0.95 (when spiking is enabled).

C.2.2 CIFAR-10

Shared settings. Unless otherwise noted, all CIFAR-10 runs use: VGG11 backbone; SGD with momentum 0.9 and weight decay 5×10^{-4}; cosine learning rate schedule with initial rate 0.1; batch size 128; dropout $p = 0.2$ after each fully connected layer; mask parameterisation: hard-sigmoid; logits mechanism: threshold; logits BatchNorm: enabled; spiking threshold 1.0 and membrane decay 0.95 (when spiking is enabled).

Table 3.3 Baseline VGG-11 architecture for CIFAR-10.

Block	Layers
Block 1	Conv(3×3, 64) MaxPool(2×2)
Block 2	Conv(3×3, 128) MaxPool(2×2)
Block 3	Conv(3×3, 256) Conv(3×3, 256) MaxPool(2×2)
Block 4	Conv(3×3, 512) Conv(3×3, 512) MaxPool(2×2)
Block 5	Conv(3×3, 512) Conv(3×3, 512) MaxPool(2×2)
FC	FC1 (2048 units, Dropout) FC2 (1024 units, Dropout) Output (10 classes)

Table 3.4 Flex-enabled VGG-11 for CIFAR-10.

Block	Layers
Block 1	PlasticFlex(64, stride 2)
Block 2	PlasticFlex(128, stride 2)
Block 3	PlasticFlex(256) PlasticFlex(256, stride 2)
Block 4	PlasticFlex(512) PlasticFlex(512, stride 2)
Block 5	PlasticFlex(512) PlasticFlex(512, stride 2)
FC	FC1 (2048 units, Dropout) FC2 (1024 units, Dropout) Output (10 classes)

C.2.3 ImageNet-100

Shared settings. Unless otherwise noted, all ImageNet-100 runs use: VGG16 backbone; SGD with momentum 0.9 and weight decay 5×10^{-4}; cosine learning rate schedule with initial rate 0.1; batch size 256; dropout $p = 0.2$ after each fully connected layer; mask parameterisation: hard-sigmoid; logits mechanism: threshold; logits BatchNorm: enabled; spiking threshold 1.0 and membrane decay 0.95 (when spiking is enabled).

Table 3.5 Baseline VGG-16 architecture for ImageNet-100.

Block	Layers
Block 1	Conv(3×3, 64) Conv(3×3, 64) MaxPool(2×2)
Block 2	Conv(3×3, 128) Conv(3×3, 128) MaxPool(2×2)
Block 3	Conv(3×3, 256) Conv(3×3, 256) Conv(3×3, 256) MaxPool(2×2)
Block 4	Conv(3×3, 512) Conv(3×3, 512) Conv(3×3, 512) MaxPool(2×2)
Block 5	Conv(3×3, 512) Conv(3×3, 512) Conv(3×3, 512) MaxPool(2×2)
FC	FC1 (4096 units, Dropout) FC2 (2048 units, Dropout) Output (100 classes)

Table 3.6 Flex-enabled VGG-16 for ImageNet-100.

Block	Layers
Block 1	PlasticFlex(64) PlasticFlex(64, stride 2)
Block 2	PlasticFlex(128) PlasticFlex(128, stride 2)
Block 3	PlasticFlex(256) PlasticFlex(256) PlasticFlex(256, stride 2)
Block 4	PlasticFlex(512) PlasticFlex(512) PlasticFlex(512, stride 2)
Block 5	PlasticFlex(512) PlasticFlex(512) PlasticFlex(512, stride 2)
FC	FC1 (4096 units, Dropout) FC2 (2048 units, Dropout) Output (100 classes)

Table 3.7 MNIST variants and differing hyperparameters. Abbreviations: BP (surrogate backprop).

Variant	Flex	SNN	BP	Epochs	Batch size	Init LR	spiking threshold	membrane decay
ANN (baseline)	–	–	–	120	128	0.10	–	–
ANN+Flex	✓	–	–	120	128	0.10	–	–
SNN	–	✓	–	120	128	0.10	1.0	0.95
SNN+Flex	✓	✓	–	120	128	0.10	1.0	0.95
SNN + BP	–	✓	✓	120	128	0.10	1.0	0.95
SNN + BP+Flex	✓	✓	✓	120	128	0.10	1.0	0.95

Table 3.8 CIFAR-10 variants and differing hyperparameters. Abbreviations: BP (surrogate backprop).

Variant	Flex	SNN	BP	Epochs	Batch size	Init LR	spiking threshold	membrane decay
ANN (baseline)	–	–	–	120	128	0.10	–	–
ANN+Flex	✓	–	–	120	128	0.10	–	–
SNN	–	✓	–	120	128	0.10	1.0	0.95
SNN+Flex	✓	✓	–	120	128	0.10	1.0	0.95
SNN + BP	–	✓	✓	120	128	0.10	1.0	0.95
SNN + BP+Flex	✓	✓	✓	120	128	0.10	1.0	0.95

Table 3.9 ImageNet-100 variants and differing hyperparameters. Abbreviations: BP (surrogate backprop).

Variant	Flex	SNN	BP	Epochs	Batch size	Init LR	spiking threshold	membrane decay
ANN (baseline)	–	–	–	180	256	0.10	–	–
ANN+Flex	✓	–	–	180	256	0.10	–	–
SNN	–	✓	–	180	256	0.10	1.0	0.95
SNN+Flex	✓	✓	–	180	256	0.10	1.0	0.95
SNN + BP	–	✓	✓	180	256	0.10	1.0	0.95
SNN + BP+Flex	✓	✓	✓	180	256	0.10	1.0	0.95

Appendix D

Supplementary Experimental Results

D.1 Top-1 Classification Accuracy Results

The following figures report the Top-1 accuracy results on MNIST, CIFAR-10, and ImageNet-100.

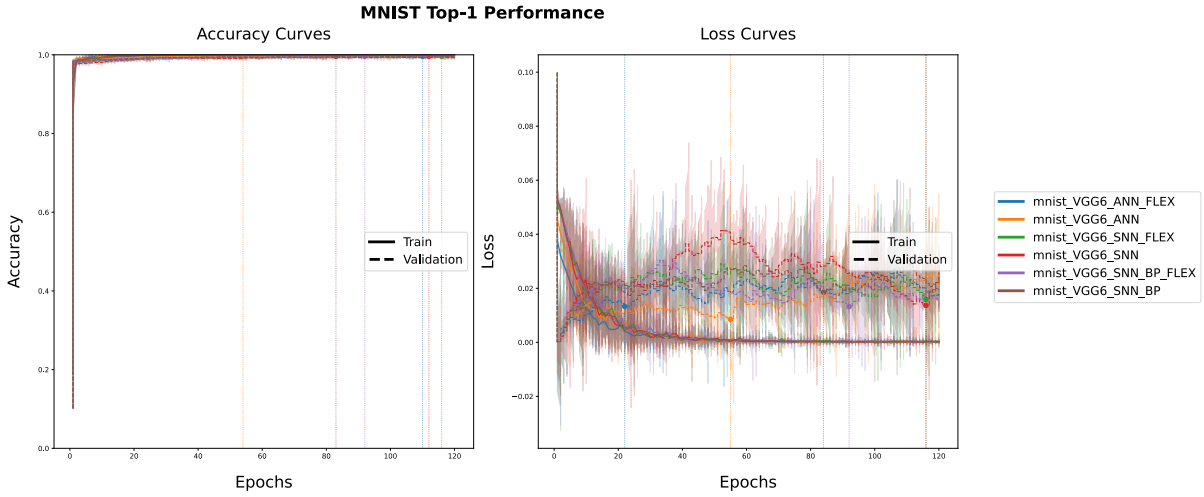


Figure 4.1 MNIST top-1 training curves (VGG6 backbone).

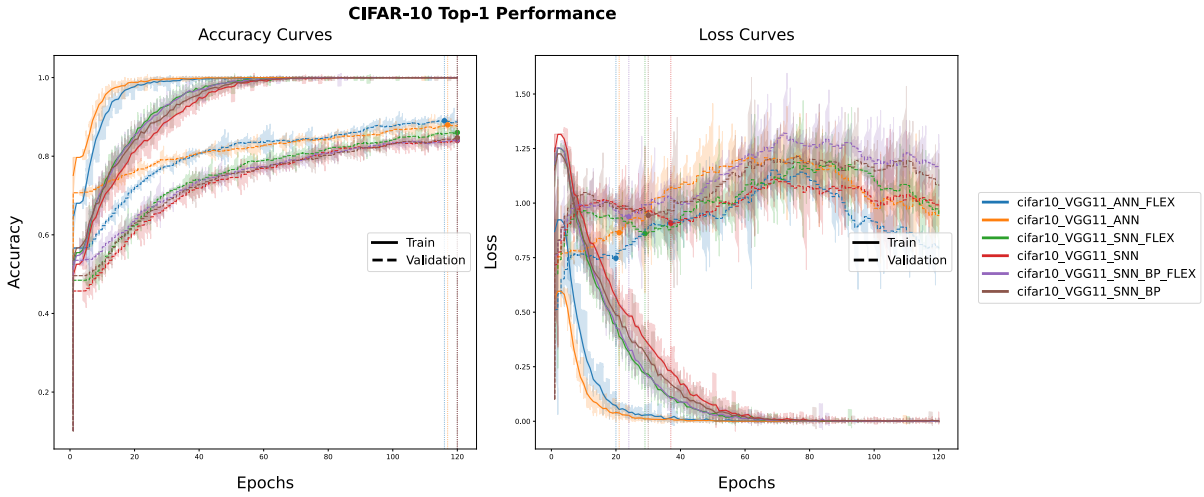


Figure 4.2 CIFAR-10 top-1 training curves (VGG11 backbone).

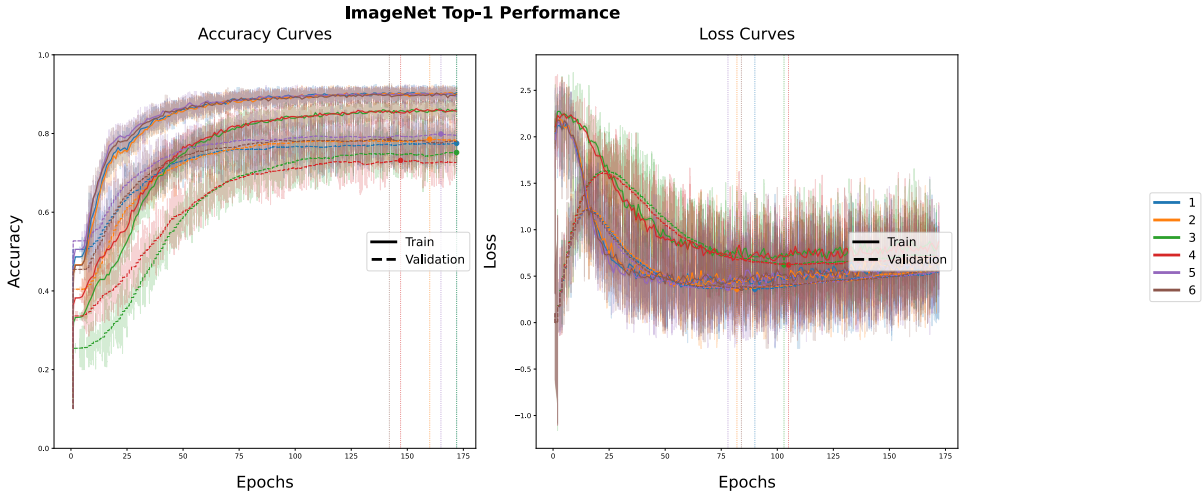


Figure 4.3 ImageNet-100 top-1 training curves (VGG16 backbone).

D.2 Additional Robustness Results

D.2.1 Adversarial Attacks

The following figures report the adverserial attack results on MNIST, CIFAR-10, and ImageNet-100.

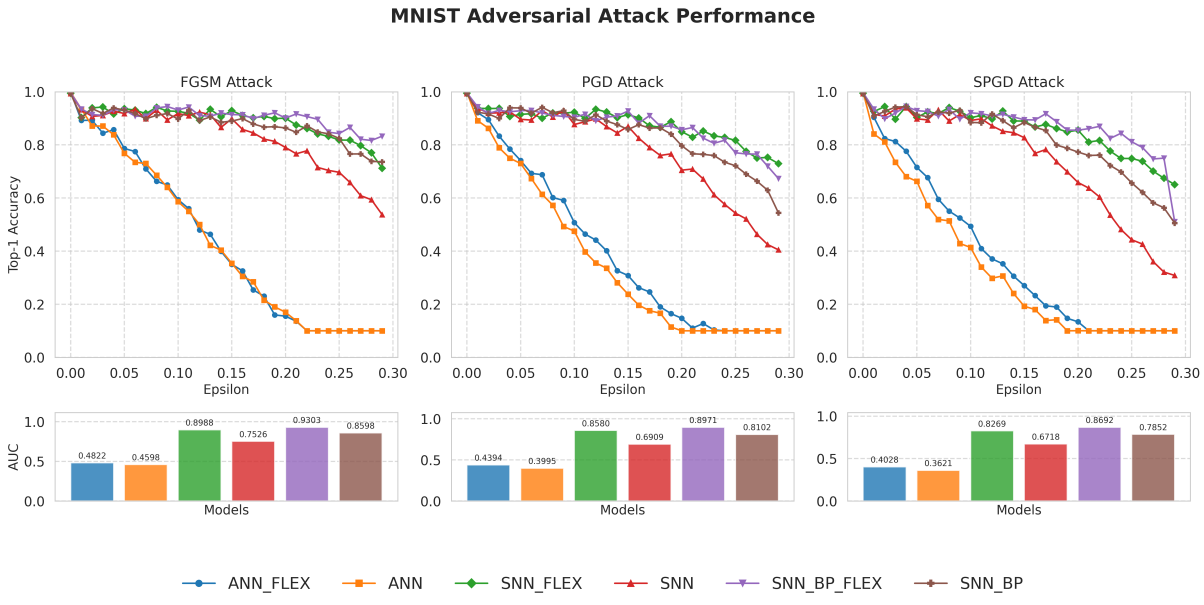


Figure 4.4 MNIST adversarial robustness. Top: accuracy under FGSM, PGD, and SPGD attacks with increasing perturbation ϵ. Bottom: AUC scores summarising robustness across ϵ. Flex-enhanced SNNs consistently achieve the highest robustness.

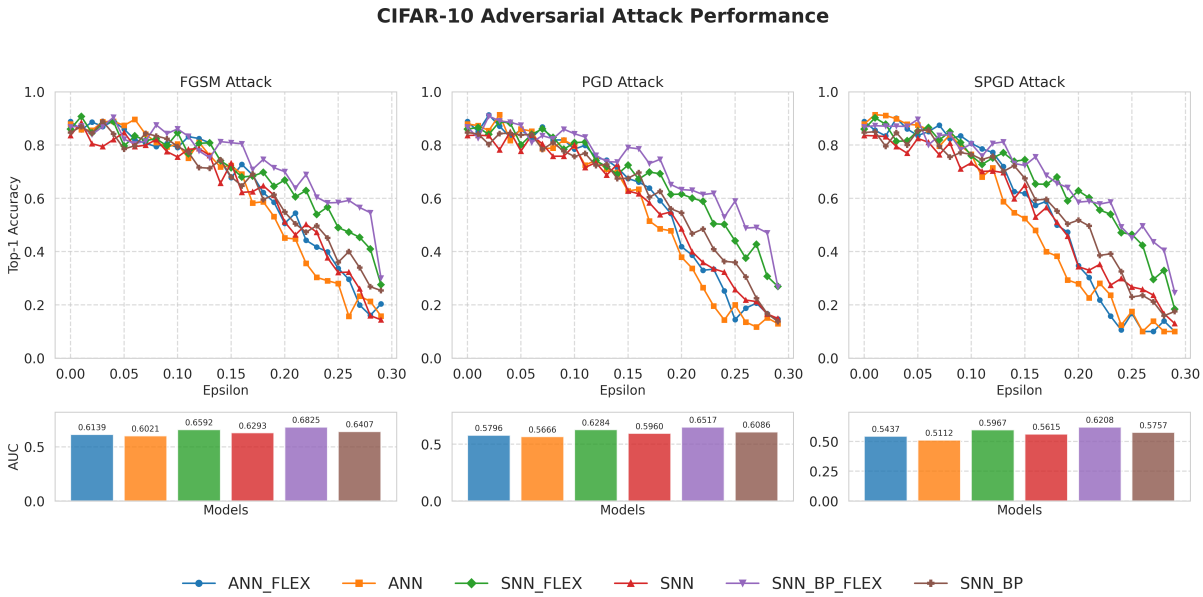


Figure 4.5 CIFAR-10 adversarial robustness.

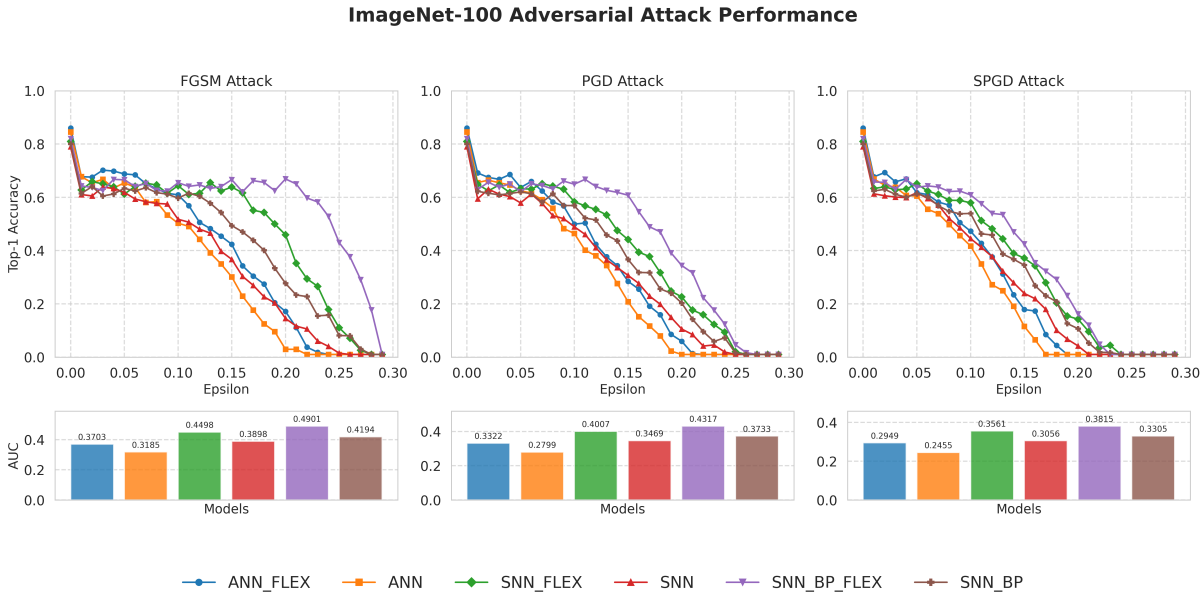
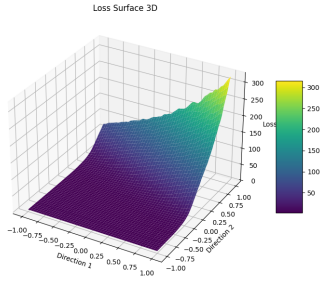


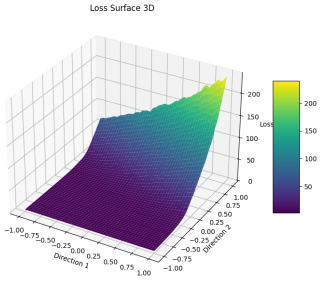
Figure 4.6 ImageNet-100 adversarial robustness.

D.2.2 Loss Landscapes

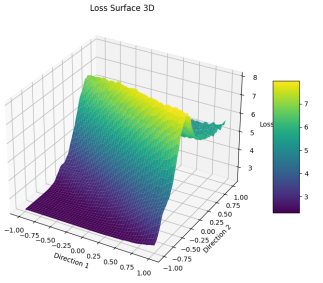
The following figures report the loss landscapes on MNIST, CIFAR-10, and ImageNet-100.



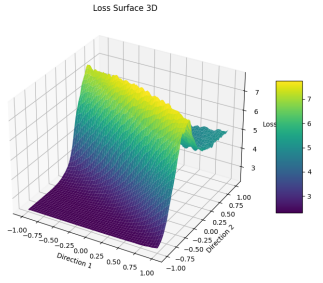
(a) ANN without Flex



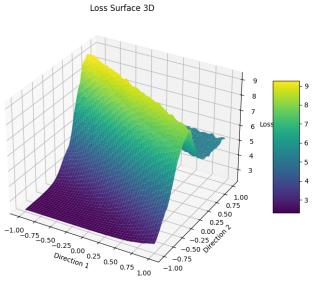
(b) ANN with Flex



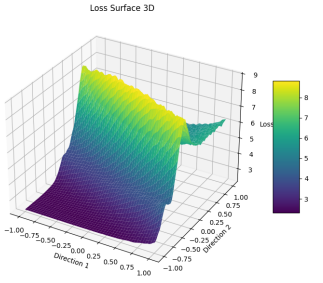
(c) SNN without Flex



(d) SNN with Flex

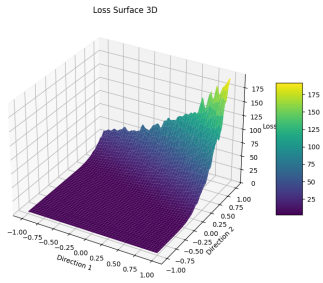


(e) SNN+BP without Flex

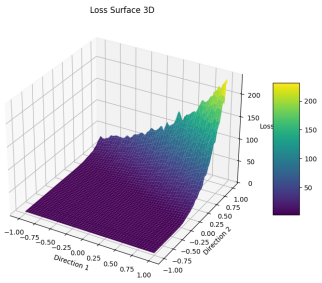


(f) SNN+BP with Flex

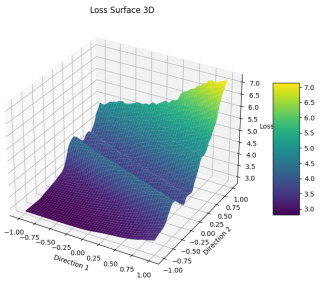
Figure 4.7 Comparison of loss landscapes across six model variants on MNIST dataset.



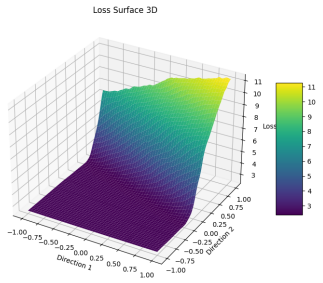
(a) ANN without Flex



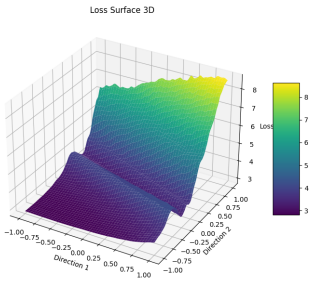
(b) ANN with Flex



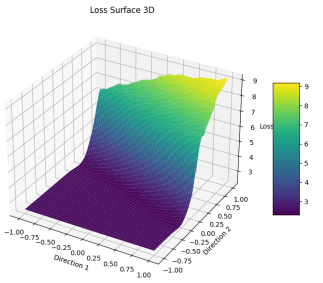
(c) SNN without Flex



(d) SNN with Flex

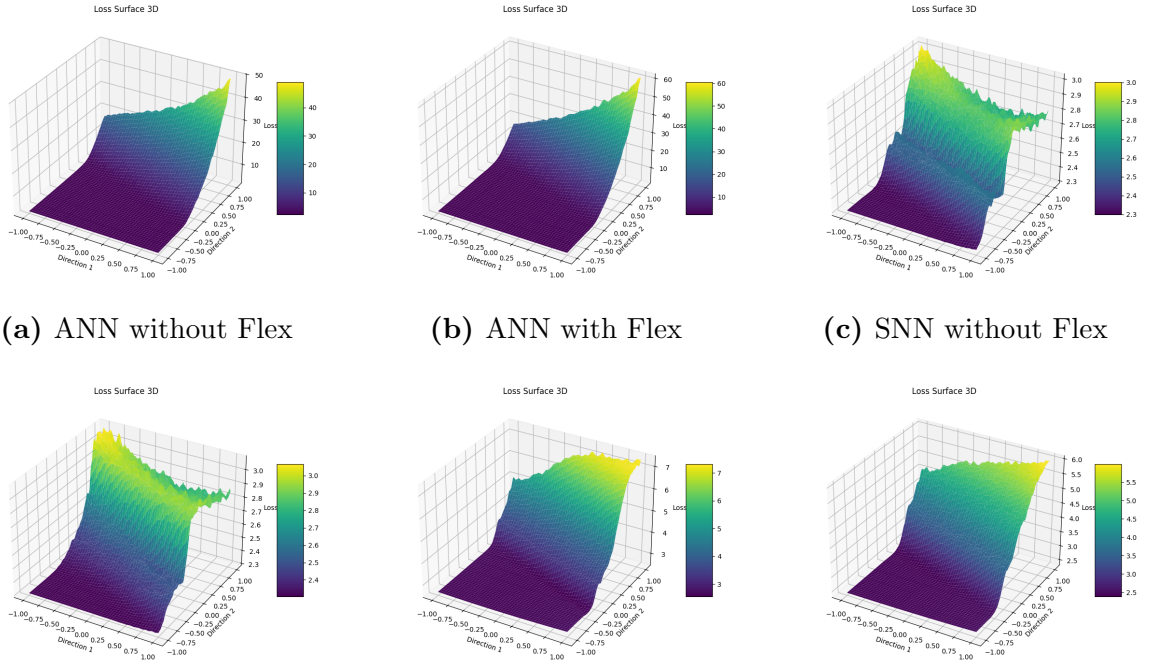


(e) SNN+BP without Flex



(f) SNN+BP with Flex

Figure 4.8 Comparison of loss landscapes across six model variants on CIFAR-10 dataset.



(a) ANN without Flex **(b)** ANN with Flex **(c)** SNN without Flex

(d) SNN with Flex **(e)** SNN+BP without Flex **(f)** SNN+BP with Flex

Figure 4.9 Comparison of loss landscapes across six model variants on ImageNet-100 dataset.

D.3 Additional Interpretability Results

D.3.1 Binariness

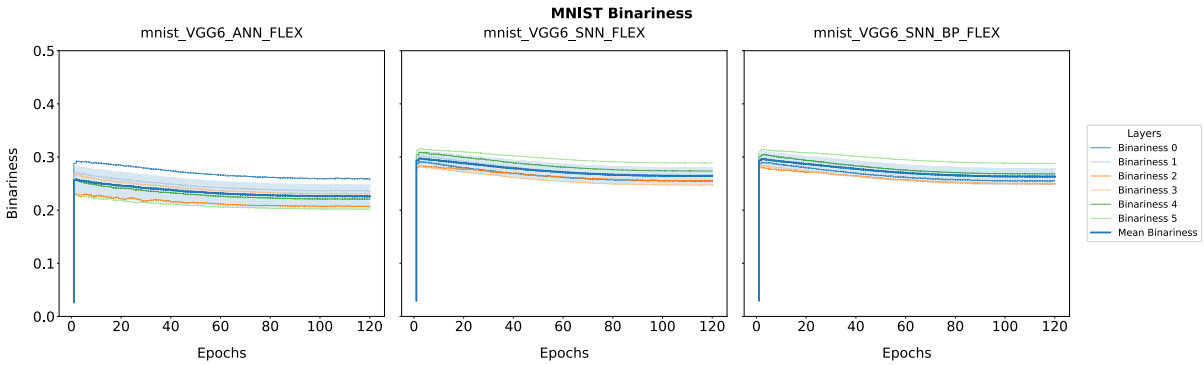


Figure 4.10 Binariness evolution on MNIST across different architectures.

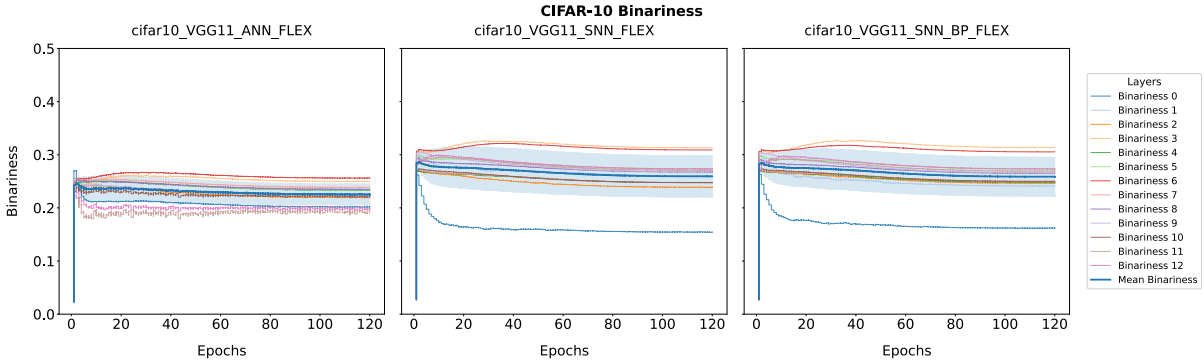


Figure 4.11 Binariness evolution on CIFAR-10 across different architectures.

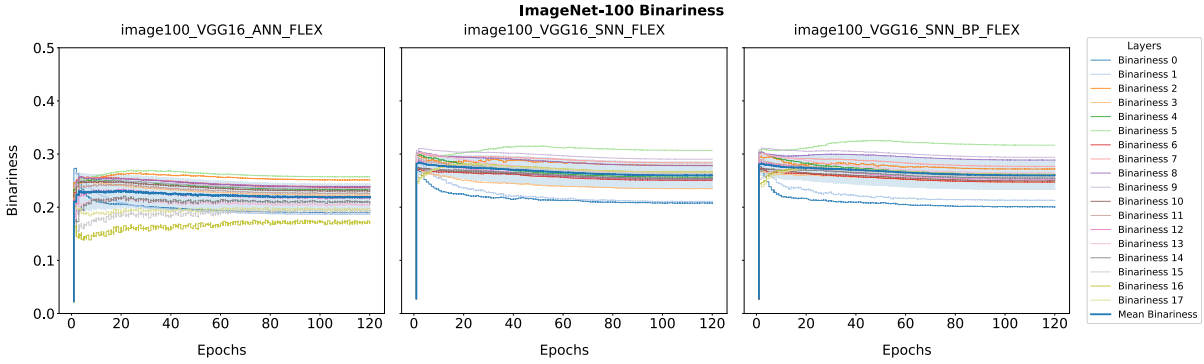


Figure 4.12 Binariness evolution on ImageNet-100 across different architectures.

D.3.2 Conv Ratio–Binariness Evolution and Correlation

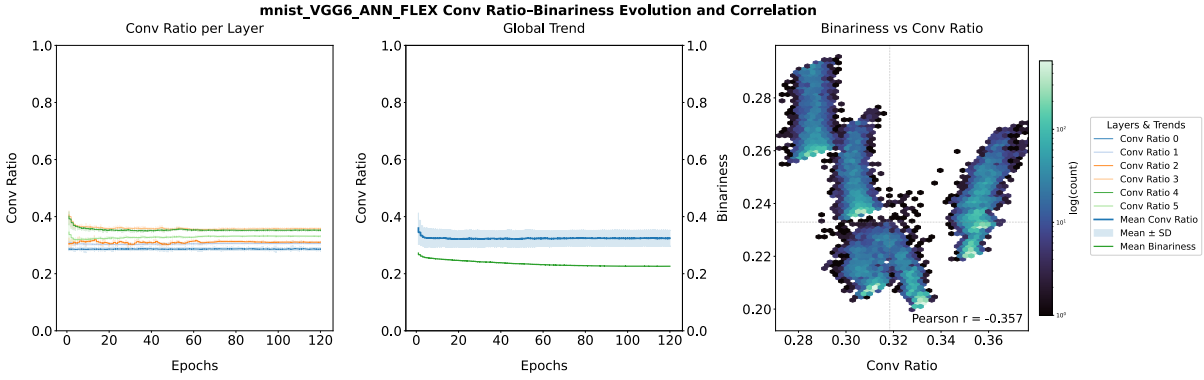


Figure 4.13 MNIST-VGG6-ANN-FLEX Conv Ratio–Binariness Dashboard. The left panel shows the evolution of convolution ratios across layers, the middle panel illustrates the global trend with mean ± SD and mean binariness, and the right panel depicts the correlation between binariness and convolution ratio.

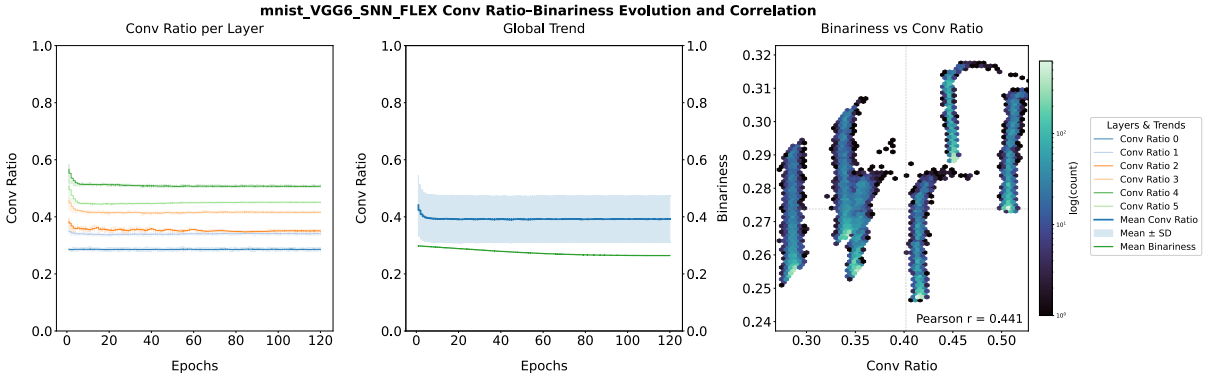


Figure 4.14 MNIST-VGG6-SNN-FLEX Conv Ratio–Binariness Dashboard.

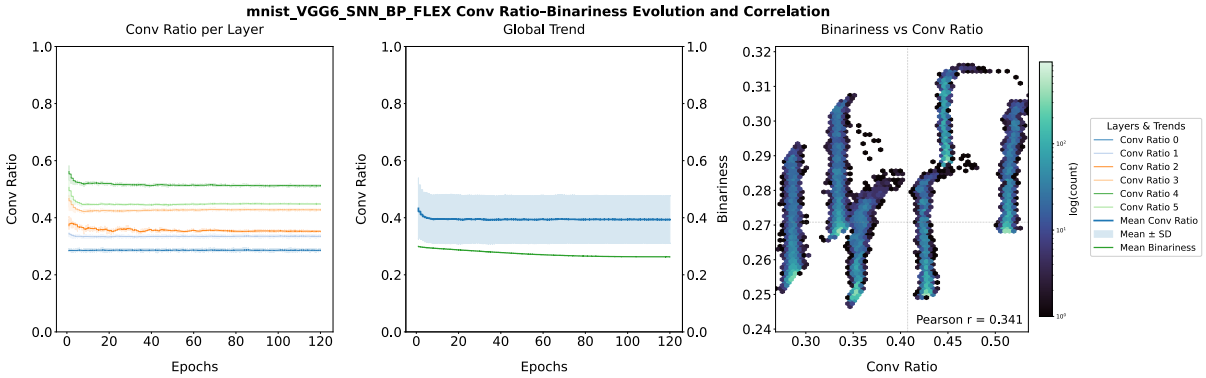


Figure 4.15 MNIST-VGG6-SNN-BP-FLEX Conv Ratio–Binariness Dashboard.

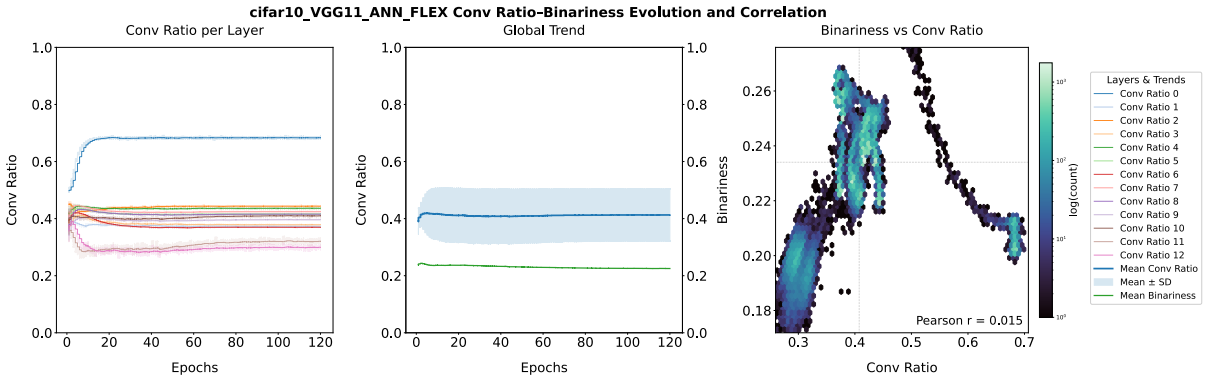


Figure 4.16 CIFAR10-VGG11-ANN-FLEX Conv Ratio–Binariness Dashboard. The left panel shows the evolution of convolution ratios across layers, the middle panel illustrates the global trend with mean ± SD and mean binariness, and the right panel depicts the correlation between binariness and convolution ratio.

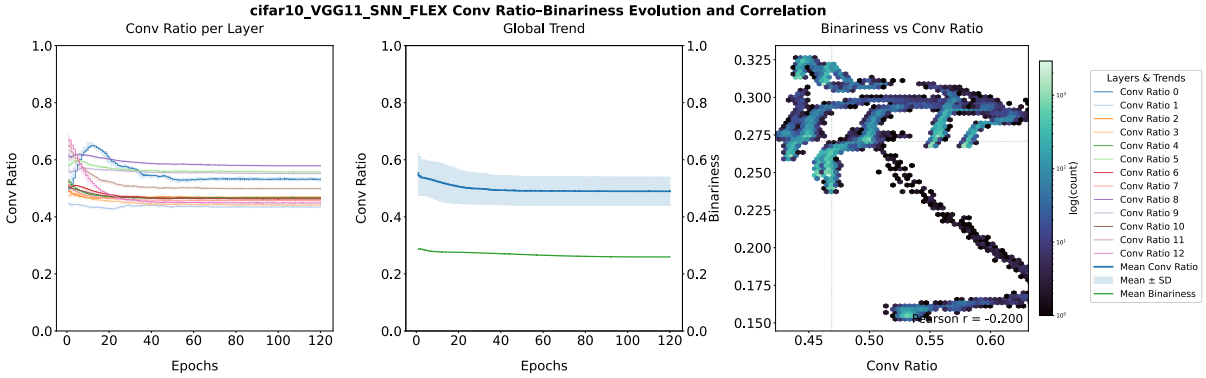


Figure 4.17 CIFAR10-VGG11-SNN-FLEX Conv Ratio–Binariness Dashboard.

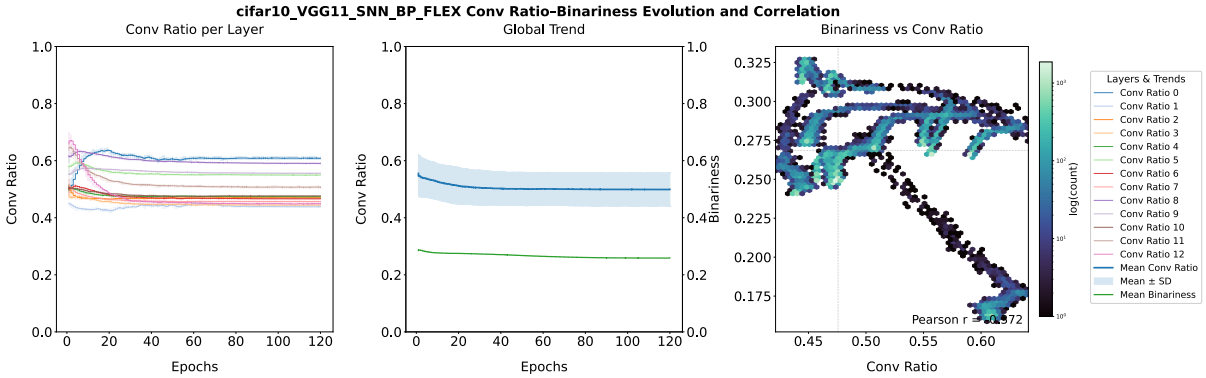


Figure 4.18 CIFAR10-SNN-BP-FLEX Conv Ratio–Binariness Dashboard.

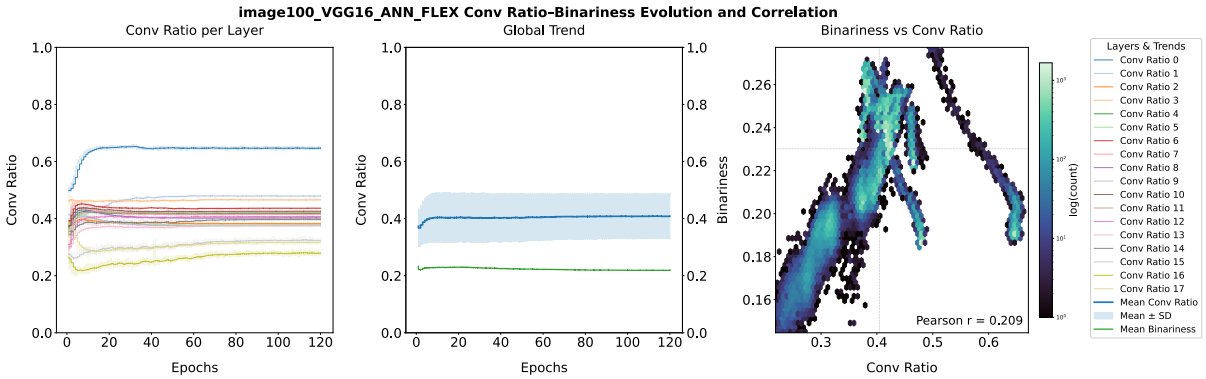


Figure 4.19 ImageNet100-VGG16-ANN-FLEX Conv Ratio–Binariness Dashboard. The left panel shows the evolution of convolution ratios across layers, the middle panel illustrates the global trend with mean ± SD and mean binariness, and the right panel depicts the correlation between binariness and convolution ratio.

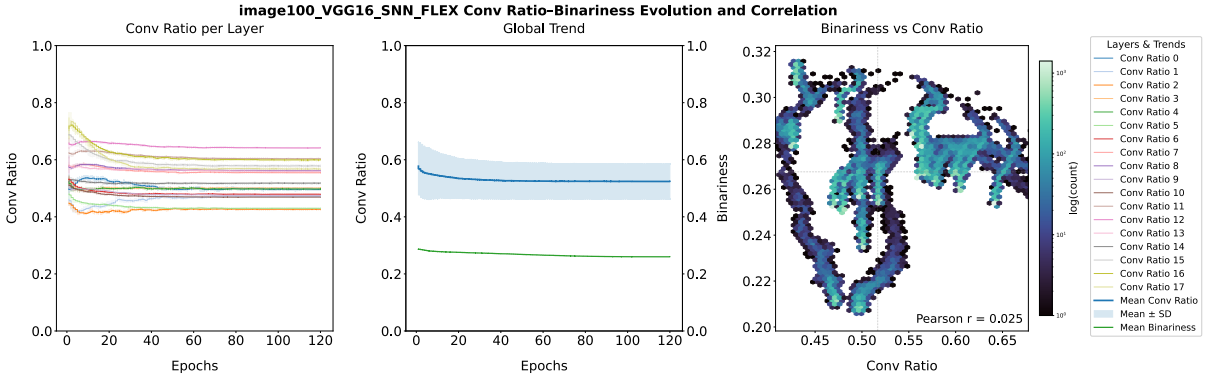


Figure 4.20 ImageNet100-VGG16-SNN-FLEX Conv Ratio–Binariness Dashboard.

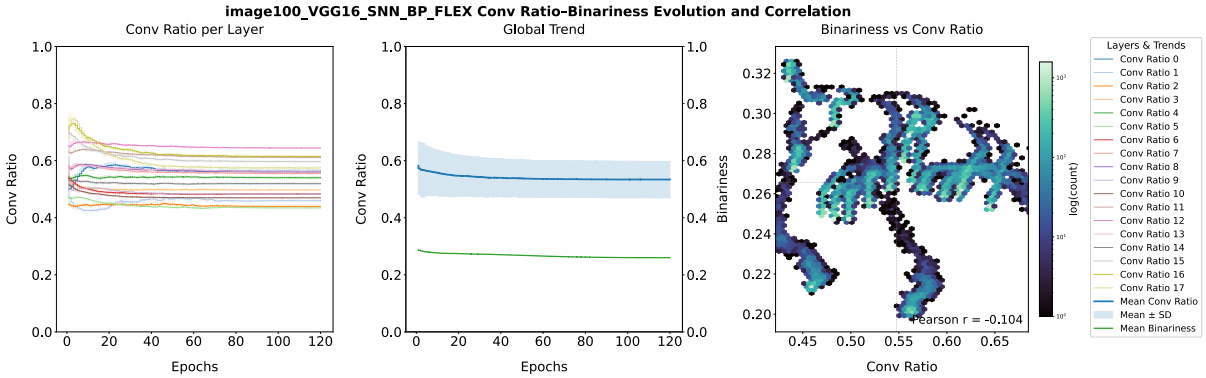


Figure 4.21 ImageNet100-VGG16-SNN-BP-FLEX Conv Ratio–Binariness Dashboard.

D.4 Other Results

Appendix E

Computational Resources and Reproducibility

E.1 Hardware and HPC Platform

All experiments were executed on the Imperial College London High Performance Computing (HPC) facility (Research Computing Service, DOI: 10.14469/hpc/2232). As described in the main text, compute nodes are equipped with dual-socket AMD EPYC 7742 processors (64 cores per socket, 256 hardware threads in total) and 1.0 TiB of system memory, running Linux kernel 4.18.

GPU-accelerated training was performed on NVIDIA Quadro RTX 6000 devices, each with 24 GB GDDR6 memory and CUDA 12.2 support (driver version 535.54.03). Typical jobs requested a single GPU, 2 CPU cores, and 16 GB of system memory, with maximum walltime up to 64 hours. Multi-GPU nodes hosting up to eight RTX 6000 devices were available; memory utilisation per process ranged between 4 GB and 24 GB depending on the experiment.

Jobs were scheduled via the PBS batch system. A representative submission script is shown below:

```
#!/bin/bash
#PBS -l select=1:ncpus=2:mem=16gb:ngpus=1
#PBS -l walltime=32:00:00
#PBS -N analysis
...
python src/train_initialiser.py --config configs/mnist1.json
```

To ensure reproducibility across runs, environment variables limited thread usage (`OMP_NUM_THREADS=1`, `MKL_NUM_THREADS=1`), and the PyTorch CUDA allocator was configured with `expandable_segments=True` to avoid fragmentation. Checkpoints were synchronised between local scratch (`$TMPDIR`) and the project directory at job completion.

E.2 Software Environment

All experiments were run under Red Hat Enterprise Linux 8.5 (Ootpa). The software environment was based on Python 3.12.10, managed with Conda. GPU drivers were provided by the NVIDIA driver version 535.54.03 with CUDA 12.2.

Model training and evaluation were implemented in PyTorch (with torchvision), and relied on standard scientific Python libraries including NumPy, scikit-learn, matplotlib, seaborn, tqdm, and rich. The complete environment configuration is available as a Conda environment file (environment.yml) in the project repository.

E.3 Code Availability

All source code related to this thesis is publicly available at:
https://github.com/ZephyrNing/BioFlexNet.
A copy of this thesis is also archived at:
https://github.com/ZephyrNing/BioFlexNet/blob/main/thesis/MRes_Thesis.pdf.
The mid-term poster can be found at:
https://github.com/ZephyrNing/BioFlexNet/blob/main/posters/BioFlexNet%20Poster.pdf.
Please note that the poster was prepared during the mid-term stage; due to different experimental settings and parameters, its results may differ from the final outcomes presented in this thesis. The results and hyperparameters reported in the thesis should be considered definitive.

List of References

[1] Elvis K Boahen et al. "Bio-Inspired Neuromorphic Sensory Systems from Intelligent Perception to Nervetronics". In: *Advanced Science* 12.1 (2025), p. 2409568.

[2] Jenna M Crowe-Riddell and Harvey B Lillywhite. "Sensory systems". In: *Health and welfare of captive reptiles*. Springer, 2023, pp. 45–91.

[3] Charles Cadieu et al. "A model of V4 shape selectivity and invariance". In: *Journal of neurophysiology* 98.3 (2007), pp. 1733–1750.

[4] Ali Almasi et al. "Mechanisms of feature selectivity and invariance in primary visual cortex". In: *Cerebral Cortex* 30.9 (2020), pp. 5067–5087.

[5] Jinming Su et al. "Selectivity or invariance: Boundary-aware salient object detection". In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2019, pp. 3799–3808.

[6] Leon O Chua. "CNN: A vision of complexity". In: *International Journal of Bifurcation and Chaos* 7.10 (1997), pp. 2219–2425.

[7] Fabio Anselmi, Lorenzo Rosasco, and Tomaso Poggio. "On invariance and selectivity in representation learning". In: *Information and Inference: A Journal of the IMA* 5.2 (2016), pp. 134–158.

[8] Nadeem Akhtar and U Ragavendran. "Interpretation of intelligence in CNN-pooling processes: a methodological survey". In: *Neural computing and applications* 32.3 (2020), pp. 879–898.

[9] Fu Xing et al. "Homeostasis-based cnn-to-snn conversion of inception and residual architectures". In: *International Conference on Neural Information Processing*. Springer. 2019, pp. 173–184.

[10] Ahmad Chaddad et al. "Modeling information flow through deep neural networks". In: *arXiv preprint arXiv:1712.00003* (2017).

[11] Alessia Celeghin et al. "Convolutional neural networks for vision neuroscience: significance, developments, and outstanding issues". In: *Frontiers in Computational Neuroscience* 17 (2023), p. 1153572.

[12] Tamás Roska et al. “The use of CNN models in the subcortical visual pathway”. In: *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications* 40.3 (1993), pp. 182–195.

[13] Grace W Lindsay. “Convolutional neural networks as a model of the visual system: Past, present, and future”. In: *Journal of cognitive neuroscience* 33.10 (2021), pp. 2017–2031.

[14] Beilun Wang et al. “How decisions are made in brains: Unpack “black box” of CNN with Ms. Pac-Man video game”. In: *IEEE Access* 8 (2020), pp. 142446–142458.

[15] Ashish Vaswani et al. “Attention is all you need”. In: *Advances in neural information processing systems* 30 (2017).

[16] Yinpeng Chen et al. “Dynamic convolution: Attention over convolution kernels”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 11030–11039.

[17] Ivet Rafegas et al. “Understanding trained CNNs by indexing neuron selectivity”. In: *Pattern Recognition Letters* 136 (2020), pp. 318–325.

[18] Shujian Yu et al. “Understanding convolutional neural networks with information theory: An initial exploration”. In: *IEEE transactions on neural networks and learning systems* 32.1 (2020), pp. 435–442.

[19] Jingwei Zhang, Tongliang Liu, and Dacheng Tao. “An information-theoretic view for deep learning”. In: *arXiv preprint arXiv:1804.09060* (2018).

[20] Wenrui Zhang and Peng Li. “Information-theoretic intrinsic plasticity for online unsupervised learning in spiking neural networks”. In: *Frontiers in neuroscience* 13 (2019), p. 31.

[21] Derek Xu et al. “Neural Network Pruning for Invariance Learning”. In: *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 1*. 2025, pp. 1691–1702.

[22] Valerio Biscione and Jeffrey S Bowers. “Convolutional neural networks are not invariant to translation, but they can learn to be”. In: *Journal of Machine Learning Research* 22.229 (2021), pp. 1–28.

[23] Tatyana O Sharpee. “How invariant feature selectivity is achieved in cortex”. In: *Frontiers in synaptic neuroscience* 8 (2016), p. 26.

[24] Stuart Geman. "Invariance and selectivity in the ventral visual pathway". In: *Journal of Physiology-Paris* 100.4 (2006), pp. 212–224.

[25] Shaofeng Cai, Yao Shu, and Wei Wang. "Dynamic routing networks". In: *proceedings of the IEEE/CVF winter conference on applications of computer vision*. 2021, pp. 3588–3597.

[26] Xingxing Wei and Shiji Zhao. "Boosting adversarial transferability with learnable patch-wise masks". In: *IEEE Transactions on Multimedia* 26 (2023), pp. 3778–3787.

[27] Prachi Garg et al. *Memorization and generalization in deep cnns using soft gating mechanisms*. 2019.

[28] Penghang Yin et al. "Understanding straight-through estimator in training activation quantized neural nets". In: *arXiv preprint arXiv:1903.05662* (2019).

[29] Yu He et al. "A trust update mechanism based on reinforcement learning in underwater acoustic sensor networks". In: *IEEE Transactions on Mobile Computing* 21.3 (2020), pp. 811–821.

[30] Kai Han et al. "A survey on vision transformer". In: *IEEE transactions on pattern analysis and machine intelligence* 45.1 (2022), pp. 87–110.

[31] Alex Graves. "Adaptive computation time for recurrent neural networks". In: *arXiv preprint arXiv:1603.08983* (2016).

[32] Saeed Masoudnia and Reza Ebrahimpour. "Mixture of experts: a literature survey". In: *Artificial Intelligence Review* 42.2 (2014), pp. 275–293.

[33] Xavier Amatriain et al. "Transformer models: an introduction and catalog". In: *arXiv preprint arXiv:2302.07730* (2023).

[34] Alexander Levine, Sahil Singla, and Soheil Feizi. "Certifiably robust interpretation in deep learning". In: *arXiv preprint arXiv:1905.12105* (2019).

[35] Ziv Goldfeld et al. "Estimating information flow in deep neural networks". In: *arXiv preprint arXiv:1810.05728* (2018).

[36] Chunwei Tian et al. "Attention-guided CNN for image denoising". In: *Neural Networks* 124 (2020), pp. 117–129.

[37] Benedetta Savelli et al. "A multi-context CNN ensemble for small lesion detection". In: *Artificial intelligence in medicine* 103 (2020), p. 101749.

[38] Shuangming Yang et al. “Self-supervised high-order information bottleneck learning of spiking neural network for robust event-based optical flow estimation”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2024).

[39] Quoc Trung Pham et al. “A review of SNN implementation on FPGA”. In: *2021 international conference on multimedia analysis and pattern recognition (MAPR)*. IEEE. 2021, pp. 1–6.

[40] Hoyoung Tang et al. “Spike counts based low complexity SNN architecture with binary synapse”. In: *IEEE Transactions on Biomedical Circuits and Systems* 13.6 (2019), pp. 1664–1677.

[41] Ruohong Zhou. “A method of converting ann to snn for image classification”. In: *2023 IEEE 3rd International Conference on Electronic Technology, Communication and Information (ICETCI)*. IEEE. 2023, pp. 819–822.

[42] Yuhang Li et al. “Error-aware conversion from ANN to SNN via post-training parameter calibration”. In: *International Journal of Computer Vision* 132.9 (2024), pp. 3586–3609.

[43] Yuling Luo et al. “An efficient, low-cost routing architecture for spiking neural network hardware implementations”. In: *Neural Processing Letters* 48.3 (2018), pp. 1777–1788.

[44] Brandon Yang et al. “Condconv: Conditionally parameterized convolutions for efficient inference”. In: *Advances in neural information processing systems* 32 (2019).

[45] Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. “Dynamic routing between capsules”. In: *Advances in neural information processing systems* 30 (2017).