Representing CNN Feature Maps with Implicit Neural Representations

A Proof-of-Concept Study Using SIRENs

CS5000: MSc CS Thesis project EEMCS

Bryan Yunlong He

Delft University of Technology

TU Delft

Representing CNN Feature Maps with Implicit Neural Representations

Master Thesis

Thesis report

by

Bryan Yunlong He

to obtain the degree of Master of Science

at the Delft University of Technology

to be defended publicly on **Thursday, 11 December, 2025 at 14:00**

Supervisor:	Sander Gielisse
Thesis advisor:	Jan van Gemert
Thesis Committee:	
Chair:	Dr. Jan van Gemert
	Dr. Klaus Hildebrandt
Project Duration:	September, 2024 - December, 2025
Faculty:	Faculty of Electrical Engineering, Mathematics and Computer Science, Delft

Preface

At the start of this project, I was intimidated by the complexity of a task I had never done before. Throughout the process, there were many moments when I wanted to abandon this work because I thought my own work was not good enough. During my first PRB student meeting, Dr. Chirag Raman, who moderated the session, gave a memorable piece of advice about learning to separate one's research or work from one's ego. Hearing and agreeing with this advice is one thing, but actually aligning one's behavior, emotions, thoughts, and attitude with it was a much harder challenge. I got to try and fail many times, and through that process I developed a better understanding of the attitude that is necessary for conducting research.

After completing this large and daunting project, I have come to truly appreciate the value and possibilities of collaboration. Before this experience, I had never worked closely with others over such an extended period. I am grateful for the mentorship and supervision of my supervisor, Sander Gielisse, and thesis advisor Prof. Dr. Jan van Gemert. Their critical feedback was humbling and invaluable as it forced me to question my reasoning, helped me further develop my critical thinking, and taught me how to separate my shared ideas from my ego. This project also gave me the opportunity to practice my communication skills with them in real time, to realize how much I needed to improve, and I gradually started to enjoy the humbling process of becoming better at communication. I am thankful for their guidance, patience, and for sharing their knowledge on how to conduct empirical research, even when I was far from a perfect student.

I would also like to thank my mother, father, and siblings for believing in me and encouraging me to continue when I wanted to give up. Their support carried me through the moments of self-doubt and frustration that naturally come with tackling something never attempted before.

I am grateful to Michel Rodrigues, Academic Counsellor for the MSc EEMCS program, for believing in me when I doubted myself and for encouraging me to practice seeing myself as a capable computer scientist instead of letting myself fall prey to imposter's syndrome.

Lastly, I want to thank myself for not giving up. Having seen this project through to the end, I feel more confident in my ability to handle stress, emotions, and the uncertainty that accompanies complex, open-ended challenges. I cherish the experience of practicing writing, argumentation, and engaging in discussion throughout this journey.

Bryan Yunlong He Delft, December 2025

Contents

1 Background Information

1.1. Convolutional Neural Networks

Convolutional Neural Networks (CNNs) [10, 3] are neural networks designed to process data with grid-like structure, such as 1D time series or 2D images. They consist of successive convolutional layers that apply small filters to local regions of the input to extract spatial features. Traditional neural networks, where every neuron in one layer connects to every neuron in the next layer (fully connected layers), struggle with image data because they treat each pixel independently, requiring an enormous number of parameters and failing to exploit the spatial structure inherent in images. CNNs address these limitations by introducing operations that respect the spatial relationships between pixels.

A key component of CNNs is the convolution operation, which applies small learnable filters to local regions of the input. Unlike fully connected layers that connect every input to every output, convolutional layers only connect each output to a small spatial neighborhood of the input. This design choice is motivated by two observations about natural images: first, nearby pixels are more strongly correlated than distant ones (spatial locality), and second, useful visual features like edges or textures can appear anywhere in an image (translation invariance).

A typical CNN architecture consists of several types of layers arranged in sequence: convolutional layers extract spatial features, pooling layers reduce spatial dimensions while retaining important information, and optional fully connected layers at the end perform classification or regression. By stacking multiple convolutional layers, CNNs learn hierarchical representations where early layers detect simple patterns like edges and colors, while deeper layers combine these into more complex structures like object parts and complete objects. This hierarchical feature learning has enabled CNNs to achieve state-of-the-art performance on various tasks ranging from image classification to object detection and semantic segmentation.

1.1.1. Convolution Operation

The convolution operation is the fundamental building block of CNNs. It applies a small learnable kernel (also called a filter) to local regions of the input. A kernel is a small matrix of learnable weights, typically 3×3 or 5×5 in size, that acts as a pattern detector. For a 2D input $\mathbf{x} \in \mathbb{R}^{H \times W}$ (an image with height H and width W) and kernel $\mathbf{w} \in \mathbb{R}^{k \times k}$ (a weight matrix of size $k \times k$, where k is the kernel size), the convolution at output position (i, j) is computed as:

$$y_{i,j} = \sum_{m=0}^{k-1} \sum_{n=0}^{k-1} w_{m,n} \cdot x_{i+m,j+n} \tag{1.1}$$

where:

- $y_{i,j}$ is the output value at spatial position denoted by (i, j) in the resulting feature map
- $w_{m,n}$ is the kernel weight at position (m, n)

- $x_{i+m,j+n}$ is the input value at position $(i+m, j+n)$
- m, n are indices that iterate over the kernel dimensions

This operation slides the kernel across the input image, computing a weighted sum at each position. The stride determines how many pixels the kernel moves at each step, while padding adds borders of zeros around the input to control the output size. The complete output of applying one kernel across the entire input is called a feature map.

1.1.2. Feature Maps

A feature map is the output produced when a convolutional kernel is applied across an entire input image or previous feature map. Each feature map is a 2D grid of activations (output values) that as a whole represents the presence and location of a particular pattern detected by its corresponding kernel.

When a convolutional layer processes an input, it typically applies multiple different kernels simultaneously. Each kernel learns to detect a different pattern: one might respond to horizontal edges, another to vertical edges, another to specific textures or colors. Each of these kernels produces its own feature map, so a convolutional layer with C kernels produces C feature maps. These feature maps are stacked together to form a 3D volume with dimensions $H' \times W' \times C$, where H' and W' are the spatial dimensions (which may differ from the input size due to stride and padding), and C is the number of channels (the number of different kernels applied).

Feature maps serve as the input to the next layer in the network. In early layers, feature maps typically represent simple, low-level patterns like edges and corners. As we move deeper into the network, feature maps represent increasingly abstract and complex patterns. For example, a feature map in a middle layer might activate strongly when it detects a wheel or a window, while a feature map in a deep layer might respond to entire objects like cars or faces.

The term **activation** refers to the output value at a specific spatial position in a feature map. An activation is the result of applying a kernel to a local input region, followed by a non-linear activation function (such as ReLU, which replaces negative values with zero). When we refer to an activation at position (h, w), we mean the value stored at spatial coordinate (h, w) in a particular feature map.

1.1.3. Inductive Biases of CNNs

CNNs leverage several inductive biases that make them particularly effective for visual tasks:

Spatial Locality: Convolutional layers process local neighborhoods of pixels rather than the entire image at once, based on the principle that nearby pixels are more strongly related than distant ones [8, 6]. This is why kernels are small (e.g., 3×3) rather than being the same dimension as the input.

Weight Sharing: The same convolutional kernel is used across all spatial locations in the input, drastically reducing the number of parameters compared to fully connected layers while enabling the detection of the same feature regardless of its position in the image [8, 9]. For example, an edge detector kernel uses the same weights whether it's examining the top-left or bottom-right corner of an image.

Translation Equivariance and Invariance: Due to weight sharing, if the input is translated, the output feature map is translated by the same amount; a property called translation equivariance. When convolutional layers are followed by pooling layers, information about absolute position is discarded, resulting in some translation invariance [8, 4]. This means the network can recognize an object regardless of where it appears in the image.

1.1.4. Receptive Fields

Receptive fields are key to understanding how CNNs process information. The receptive field of an activation refers to the region of the input image or prior feature map that influences its value. We distinguish between two types:

Receptive Field at Previous Layer: For a given activation at spatial coordinate $(h^{(i)}, w^{(i)})$ in layer i's feature map, its receptive field in the previous layer $i-1$ is determined by the kernel size k_i, stride s_i, and padding p_i of the convolution operator that produced the feature map at layer i :

$$R^{(i-1)}(h^{(i)}, w^{(i)}) = [h^{(i)} \cdot s_i, \ldots, h^{(i)} \cdot s_i + k_i - 1] \times [w^{(i)} \cdot s_i, \ldots, w^{(i)} \cdot s_i + k_i - 1] \tag{1.2}$$

where:

- $R^{(i-1)}(h^{(i)}, w^{(i)})$ denotes the receptive field of the activation at $(h^{(i)}, w^{(i)})$ of the i-th feature map; it is a collection of activations from the feature map at layer $i-1$
- $(h^{(i)}, w^{(i)})$ are the spatial coordinates (height and width indices) in layer i
- k_i is the kernel size of the convolution operator that outputs the feature map at layer i
- s_i is the stride (step size) of the convolution operator that outputs the feature map at layer i, which controls how many pixels the kernel moves at each step
- p_i is the padding at layer i; amount of zero-padding added to input borders prior to convolution operation
- The bracket notation $[a, \ldots, b]$ denotes the range of indices from a to b
- The Cartesian product $\times$ forms all possible pairs of height and width indices from the two ranges; that is, each element of $R^{(i-1)}(h^{(i)}, w^{(i)})$ is a 2-tuple (h', w') corresponding to a spatial coordinate in the previous layer.

This describes the immediate spatial neighborhood in layer $i-1$ that contributes to computing the activation at $(h^{(i)}, w^{(i)})$ in layer i.

Hierarchical Receptive Field: The hierarchical receptive field of an activation in layer i refers to the region in the original input image that influences its value [7], see Figure 1.1. This receptive field grows with network depth as each layer aggregates information from increasingly larger spatial regions. Computing an activation in a deep layer requires processing its entire hierarchical receptive field through all previous layers, which becomes computationally expensive for high-resolution inputs.

The hierarchical nature of receptive fields is both a strength and a limitation of CNNs. While it enables the learning of complex, semantically meaningful features, it also means that memory requirements grow quadratically with input resolution, as all intermediate feature maps must be stored in memory to update the kernel weights of the CNN.

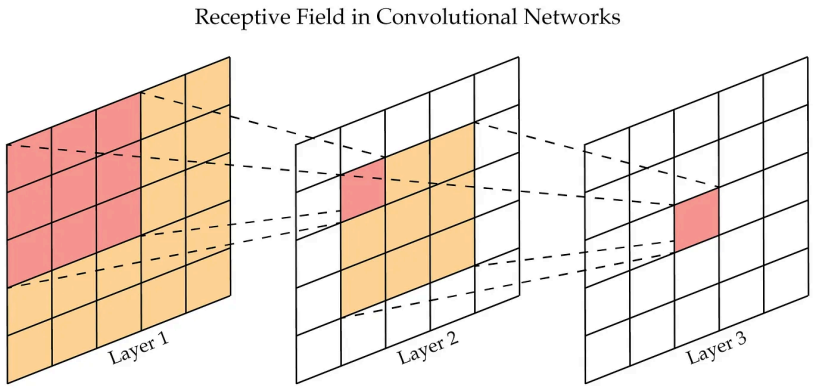

Figure 1.1: Illustration of receptive fields in a convolutional neural network from [2], showing how deeper layers aggregate information from increasingly larger spatial regions in the input.

1.2. Implicit Neural Representations

Implicit Neural Representations (INRs) represent a different method of storing and processing signals compared to traditional discrete representations. While traditional approaches store images in 2D arrays of pixel values, INRs encode signals as continuous mathematical functions. This shift from discrete

samples to continuous functions offers some advantages: signals can be queried at arbitrary resolutions, the representation becomes independent of the original sampling rate, and memory requirements scale more favorably with resolution.

Instead of storing values at each spatial coordinate, an INR requires train a neural network to learn the underlying function that maps coordinates to values. For an image, this means learning a function $f : \mathbb{R}^2 \to \mathbb{R}^3$ that takes spatial coordinates (x, y) as input and outputs the corresponding RGB color values. Once trained, this neural network is the image representation. To reconstruct the image at any desired resolution, the network is queried at the corresponding set of coordinates.

INRs have found success across diverse domains. In computer vision, Neural Radiance Fields (NeRFs) [11] use INRs to represent 3D scenes and enable novel view synthesis. In shape modeling, DeepSDF [12] represents 3D geometry as signed distance functions. For image representation, networks like SIREN [13] can fit high-quality images with surprisingly compact models.

A key advantage of INRs becomes apparent when considering multi-resolution requirements. Traditional array-based representations must store separate arrays for each resolution level, with memory growing quadratically with resolution. INRs, however, use the same set of network parameters regardless of the query resolution. As shown in Figure 1.2, this leads to much more favorable scaling properties.

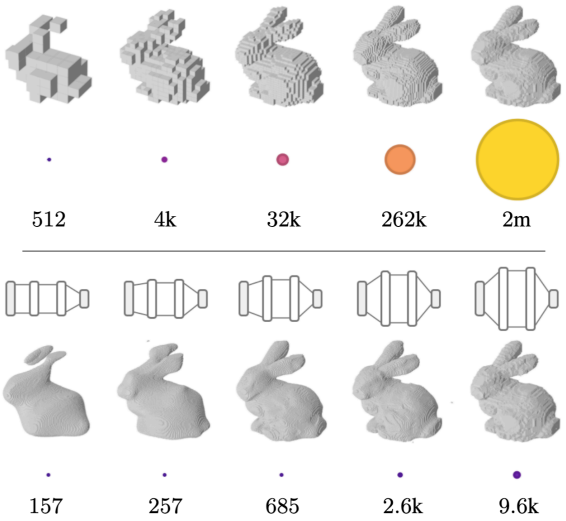

Figure 1.2: Comparison of storage requirements for array-based versus function-based (INR) representations at different resolutions from[1]. Circle area represents the numerical size of the array (top) versus the learned function parameters (bottom). INRs scale much more gracefully with resolution than array representations.

1.2.1. Sinusoidal representation networks

Sinusoidal representation networks (SIRENs)[13] are a specific type of INR that use periodic sinusoidal activations instead of ReLU. This design allows them to capture high-frequency details that ordinary multilayer perceptrons often struggle to learn due to problem known as spectral bias. The sine activations make SIRENs more expressive, enabling them to capture fine spatial structures such as sharp edges, textures, that are common in natural images and signals.

1.2.2. Training SIRENs

In a standard SIREN training setup, the dataset is constructed directly from a single image. The input set $\mathbf{X}$ consists of all pixel coordinates, typically normalized to the range $[-1, 1]^2$, while the target set $\mathbf{Y}$ contains the corresponding (RGB) pixel values at those coordinates. Training a SIREN therefore amounts to learning a continuous function that maps spatial coordinates to pixel values. Once trained, the network can reconstruct the entire image and even render it at arbitrary resolutions by evaluating the learned function at any set of coordinates.

Each SIREN is trained using the mean squared error (MSE) loss between the predicted and true pixel values. The Adam optimizer [5] is commonly used to update the SIREN weights based on gradients computed from the MSE loss. Training is performed with a single batch per epoch, where the batch contains the entire dataset, resulting in one optimization step per epoch. This approach is natural for signal fitting tasks where the objective is to learn a single continuous function that represents the entire signal, and using the complete set of coordinates in each update provides accurate gradient information for the entire domain. Training typically proceeds for several thousand epochs, although convergence often occurs much earlier.

2

Research Paper

Representing CNN Feature Maps with Implicit Neural Representations: A Proof-of-Concept Study Using SIRENs

Bryan Y. He Alexander Gielisse Jan C. van Gemert
Delft University of Technology, The Netherlands

Abstract

High-resolution image analysis using deep Convolutional Neural Networks (CNNs) faces significant memory constraints due to the quadratic growth of intermediate feature maps with input resolution. This paper investigates whether Implicit Neural Representations (INRs), specifically SIRENs, can effectively represent CNN feature maps to reduce memory footprint during training. We address the unique challenge that CNN feature maps are not static signals but evolve continuously as network weights are updated through gradient-based optimization. Through three experiments on a modified All-CNN architecture trained on MNIST, we validate that: (1) SIRENs can fit static feature maps from frozen CNNs with high fidelity (PSNR $>$ 30 dB) regardless of weight initialization; (2) SIRENs can track evolving feature maps during training, though with reduced reconstruction quality compared to static targets; and (3) SIREN-assisted feedforward—where SIRENs predict missing activations in receptive fields—enables classification accuracy (20.97%) above random guessing (10%) but substantially below standard training (95%). While results demonstrate the feasibility of using SIRENs to represent dynamic feature maps, significant challenges remain in maintaining reconstruction fidelity when SIRENs are integrated into the training loop. This proof-of-concept study provides empirical insights into bridging continuous implicit representations with discrete deep learning pipelines and highlights promising directions for future research in memory-efficient high-resolution image analysis.

1. Introduction

High-resolution (HR) images contain a large number of pixels per unit area, enabling them to capture information from both the global context and local textural details. In the field of pathology, whole-slide images (WSIs) of human organ tissue can reach sizes of up to 100,000 by 200,000 pixels (e.g., CAMELYON17 dataset [2]) at the highest resolution level, and are routinely analyzed by medical professionals for detection and diagnosis [17, 31, 55]. In remote sensing, HR satellite images are analyzed for a variety of purposes, including urban landscape planning [39, 47, 57], wildlife tracking [8, 36, 45], military surveillance [20, 46, 64], weather forecasting [4, 13, 40], and the study of long-term climate change [19, 68, 72].

Due to the large size of these HR images, analysis by human experts is a highly subjective and labor-intensive task. Computer Vision systems can automate these time-consuming visual analysis tasks. For example, deep Convolutional Neural Networks (CNNs) have achieved remarkable success in automating image-based tasks such as classification [41, 65, 66], semantic segmentation [22, 56, 58], object detection [5, 29, 70], and regression tasks [69]. However, these advances have relied on training on downsampled images (e.g., 224 by 224 in ImageNet [9]) to accommodate hardware memory constraints, which can distort or remove relevant local details [50].

Training CNNs directly on high-resolution images significantly increases memory usage, because deep CNNs contain many layers that each produce intermediate feature maps that must be stored for backpropagation, and the spatial resolution of these feature maps grows quadratically with input resolution. Existing strategies for handling high-resolution inputs typically rely on patch-based training [3, 18], multi-scale processing [59, 71] , or aggressive downsampling [30, 35] to fit memory constraints, but these approaches inevitably lose global context or fine-grained details. Implicit Neural Representations (INRs), particularly SIRENs [60], have recently demonstrated effective compression of natural images by representing them as continuous coordinate-based functions, achieving memory usage that is independent of image resolution. However, such methods have so far been applied to static input images rather than to intermediate CNN feature maps. In this work, we propose representing feature maps implicitly with SIREN weights instead of storing them as dense arrays. To our knowledge, this is the first work to explore using implicit neural representations to store and predict intermediate CNN feature maps during training.

Using SIRENs to compress feature maps during the training of CNNs presents a distinct challenge. SIRENs

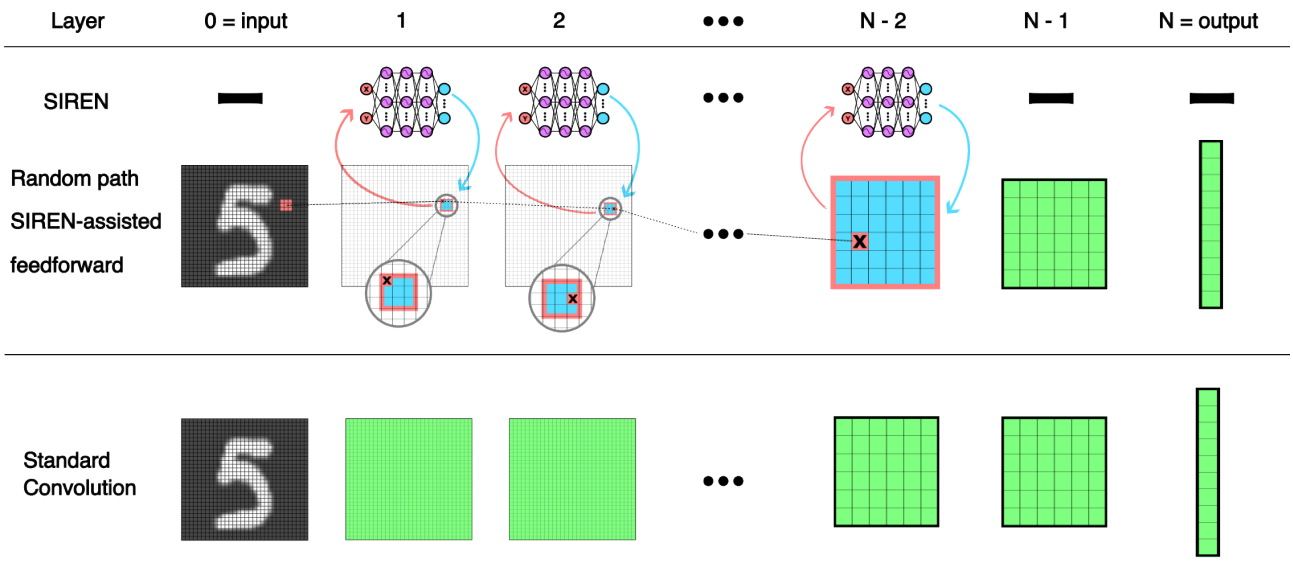

Figure 1. **Comparison of SIREN-assisted feedforward versus standard convolutional feedforward.** In SIREN-assisted feedforward, the network uses one dedicated SIREN per layer (layers 1 to $N-2$) to predict feature-map activations. Green feature maps are computed via standard convolution i.e. the entire previous layer is convolved. Blue regions indicate activations predicted by a SIREN at the spatial coordinates marked by red hollow squares. The red hollow squares represent the specific spatial locations where activations are needed for the subsequent convolution. An "X" in a red cell marks an activation computed by convolving the kernel with the activations in the red-square region in the previous layer. Although only a single random path is shown for clarity, multiple paths can be processed in parallel. In the bottom row, standard convolution computes all feature maps in their entirety at each layer by convolving the complete input or feature map at previous layer.

have been shown to effectively compress and reconstruct static signals, such as images and 3D scenes [60]. However, during training, a CNN's feature maps are not static signals. The feature maps computed for the same image from the training dataset are different across epochs because they are computed with the convolutional weights that are frequently updated through gradient-based optimization. Since CNN weights are updated at every training step, the corresponding feature maps change frequently rather than remain static. This raises the question whether SIRENs can effectively represent such evolving signals.

HR inputs not only increase memory requirements but also substantially raise the computational cost of training CNNs. To compute a single activation in a deep layer, the network must first process all activations in its hierarchical receptive field from previous layers. As the input resolution grows, these receptive fields expand at every layer, meaning that deeper activations require increasingly large portions of the input image to be processed. This leads to a rapid growth in the number of convolution operations, making training on high-resolution data computationally expensive.

In this paper, we investigate SIREN-assisted Feedforward, an alternative training method where intermediate feature maps are stored in SIRENs. Activations along a sampled path from input to output are explicitly computed and used to supervise the corresponding SIREN that stores the feature maps of that layer. This enables SIRENs to predict the missing activations in the receptive field required at the previous layer which is then used to compute an activation at the next layer, avoiding the need to process the entire hierarchical receptive field as in the conventional feedforward pass, see Figure 1 for a schematic comparison. We present this as a proof-of-concept, providing empirical insight into the feasibility of using SIRENs to represent evolving feature maps during CNN training.

The contributions of this paper are as follows:

- We introduce **SIREN-assisted Feedforward**, a novel training framework that uses implicit neural representations (SIRENs) to store and predict intermediate CNN feature maps, exploring an alternative to conventional approach of training CNNs.
- To our knowledge, this is the first work to apply implicit neural representations to represent *dynamic* feature maps that evolve during CNN training, rather than to static input images.
- We empirically validate that SIRENs can fit both static feature maps from frozen CNNs and evolving feature

maps from CNNs with trainable weights with high fidelity.

- We provide a proof-of-concept showing that the proposed SIREN-assisted Feedforward framework enables All-CNN to achieve accuracy above random guessing, validating its functional feasibility despite higher computational cost.

2. Related Work

Implicit Neural Representations INRs model signals such as images, 3D shapes, or radiance fields as continuous functions that map spatial coordinates to signal values [48, 49, 52, 60]. However, early implementations of INRs based on coordinate-based MLPs were found to exhibit spectral bias, which causes them to learn low-frequency components more readily than high-frequency detail [54]. Periodic activation functions as in SIREN [60], as well as Fourier features [67] and positional encodings [49], can be employed to mitigate spectral bias and enable INRs to reconstruct high-frequency details more accurately.

Beyond reconstruction quality, INRs are able to represent signals continuously across arbitrary spatial scales, without being bound to a fixed grid resolution. This property allows them to store large signals independently of the resolution of the discrete data they are trained on and to reconstruct these signals at any desired resolution, which has motivated their use in image and video compression [10, 12, 63]. We use SIRENs to store and predict intermediate feature maps in our work because their straightforward design, ability to capture high-frequency details accurately, ability to compress feature maps efficiently.

INRs as Data Representations Recent work has explored how INRs can be used as an alternative data modality for downstream tasks [11, 51]. To address the challenge of efficiently representing multiple data discrete instances with INRs, approaches that use a shared base network whose weights are modified with small instance-specific modulation vectors have emerged [11, 32]. The modulation vectors are then used as input to downstream models [11].

To enable the integration of INRs into standard deep learning pipelines, inr2vec [42] introduces an encoder–decoder framework that takes the weights of a trained INR as input and outputs a compact latent embedding that a jointly trained decoder uses to reconstruct the original signal. These embeddings can then be used in downstream tasks.

INR parameters exhibit permutation symmetry, since permuting neurons within a layer does not change the network's function [26]. Representing INRs as graphs allows permutation-equivariant graph neural networks or transformers to process them consistently across diverse architectures and parameter orderings [34].

Overall, these methods share a common pipeline: converting discrete datasets into collections of neural fields, then extracting features from modulation vectors, latent embeddings, or graph-based representations of INR parameters for downstream tasks. In contrast, our work uses INRs within the forward pass to predict activations of intermediate feature maps, without converting input images into implicit representations or using INR parameters as features.

Feature Map Compression and Restoration Semantically rich intermediate feature maps from deeper CNN layers are high-dimensional, memory- and compute-intensive [23,74], creating a need for efficient storage. Memory usage can be lowered by removing redundant information. Approaches such as quantization [7, 21, 73] represent activations and weights in lower precision such as 8-bit instead of the standard 32-bit floating point, enabling faster inference and smaller storage requirements. Pruning removes redundant filters or channels, thereby reducing the number of feature maps while preserving accuracy [25,38,43,44]. Knowledge distillation compresses networks by transferring feature knowledge from a large teacher to a smaller student, preserving task-relevant information in a more compact representation [6, 16, 27, 53].

Contrasting these compression methods that reduce information to save memory, INRs have been explored as a means to restore information that is lost in deep feature maps, e.g. FeatUp [14] fits INRs to CNN feature maps to reconstruct fine spatial detail lost through pooling or downsampling. Since INRs have shown potential to restore feature information, we extend this idea by using SIRENs to predict intermediate activations instead of explicitly computing all activations in the entire hierarchical receptive field needed to obtain the final network output.

3. Method

3.1. One SIREN Per Intermediate Feature Map

Given an input image $\mathbf{x}^{(0,m)} \in \mathbb{R}^{C_0 \times H_0 \times W_0}$ indexed by m from a training dataset $\{\mathbf{x}^{(0,m)}, \mathbf{y}^m\}_{m=1}^M$, and a CNN with L consecutive convolutional layers

$$\mathcal{C}^i(\,\cdot\,;\theta^i), \quad i = 1, \ldots, L \tag{1}$$

, where θ^i denotes the layer parameters, with kernel size k_i, stride s_i, padding p_i, input channels C_{i-1}, and output channels C_i, the intermediate feature maps are denoted as

$$\mathbf{x}^{(i,m)} \in \mathbb{R}^{C_i \times H_i \times W_i}, \qquad \mathbf{x}^{(i,m)} = \mathcal{C}^i(\mathbf{x}^{(i-1,m)}; \theta^i) \tag{2}$$

A single SIREN $\mathcal{S}(\,\cdot\,;\phi)$ has a fixed output dimensionality, so it cannot reconstruct all L intermediate feature maps when the output channel dimensions C_i differ across layers.

To avoid constraining C_i to be the same across all layers, we use separate SIRENs, each tasked with fitting the first $j \leq L$ intermediate feature maps of a training image:

$$\mathcal{S}(\,\cdot\,;\phi^{(i,m)}), \qquad m = 1,\ldots,M;\ i = 1\ldots j \tag{3}$$

Consequently, the memory usage scales linearly with both the number of SIREN-modeled layers j and dataset size M, which constrains the model depth and dataset size that can be used in our experiments.

3.2. Feature Maps from Modified All-CNN

All-CNN [61] comprises a relatively small number $L = 9$ of convolutional layers, which makes it feasible to use despite the memory constraints that come with the choice of assigning one SIREN per intermediate feature map of each training image.

Two modifications are made to All-CNN to make its feature maps more suitable for SIRENs to fit. In preliminary tests using the original All-CNN which does not have any normalization layers, we observed internal covariate shift [28], where the activations of intermediate feature maps became increasingly biased away from zero and often grew in magnitude. As a result, SIREN reconstructions of the feature maps are less accurate, because the initialization strategy and sinusoidal activations are designed to output values within the range $[-1, 1]$ [60]. The addition of layer normalization [1] helps stabilize the mean and scale of activations of feature maps across all layers, empirically improving the quality of SIREN reconstructions.

The second modification is the removal of dropout layers [62] originally present at the input and after the two convolutional layers with stride $s = 2$. However, unlike the gradual changes to feature maps that occur as CNN weights are updated during training, dropout introduces abrupt stochastic changes to the feature maps computed for the same image across epochs. Randomness propagates through all subsequent convolutional layers, altering downstream feature maps as well. If dropout layers are not omitted, SIRENs are tasked with fitting a stochastically varying target in each epoch, which increases the approximation difficulty and reduces the reliability of the feature maps they are tasked with reconstructing. An overview of the final All-CNN architecture used in our experiments is shown in Table 1.

Layer #	Layer description
0	input
1	conv(3x3,out=96,pad=1, stride=1), LayerNorm, ReLU
2	conv(3x3, 96, 1, 1), LayerNorm, ReLU
3	conv(3x3, 192, 1, 2), LayerNorm, ReLU
4	conv(3x3, 192, 1, 1), LayerNorm, ReLU
5	conv(3x3, 192, 1, 1), LayerNorm, ReLU
6	conv(3x3, 192, 1, 2), LayerNorm, ReLU
7	conv(3x3, 192, 0, 1), LayerNorm, ReLU
8	conv(1x1, 192, 0, 1), LayerNorm, ReLU
9	conv(1x1, #labels, 0, 1), LayerNorm, ReLU
10	average pool (6x6)

Table 1. Modifited architecture of All-CNN

3.3. Fitting Static Feature Maps

Given a complete feature map $\mathbf{x}^{(i,m)}$ with no missing activations, SIREN $\mathcal{S}(\,\cdot\,;\phi^{(i,m)})$ can fit it by optimizing $\phi^{(i,m)}$ to minimize the mean squared error loss:

$$\mathcal{L}_{\text{MSE}} = \sum_{i=1}^{n} \|\mathcal{S}([H_i, W_i]; \phi^{(i,m)}) - \mathbf{x}^{(i,m)}\|^2 \tag{4}$$

is minimized. Here, $[H_i, W_i]$ denotes the contiguous discrete interval of all spatial coordinates in the feature map at layer i.

3.4. Training All-CNN with SIREN-Assisted Random Path Feedforward

The feedforward procedure consists of two parts: random path sampling, then feedforward along the sampled paths. For clarity, we first describe the procedure for a single random path, and later generalize to the case of multiple paths.

3.4.1 Paths for SIREN-Assisted Feedforward

A random path is an ordered tuple of spatial coordinates

$$\mathcal{P} = \left(u^{(1)}, u^{(2)}, \ldots, u^{(j)}\right), \tag{5}$$

and it is constructed backwards, starting from the last element $u^{(j)}$ from the j-th intermediate feature map proceeding to $u^{(1)}$. The last element $u^{(j)} = (h^{(j)}, w^{((j)})$ is randomly sampled from the set

$$\{1, 2, ..., H_j\} \times \{1, 2, ..., W_j\} \subset \mathbb{N}^2 \tag{6}$$

$$j = \max\{i \in \{1, \ldots, L\} \mid k_i > 1 \wedge H_i > 1 \wedge W_i > 1\} \tag{7}$$

The condition of $k_i > 1$ is included because 1×1 convolutional kernels do not expand the receptive field and only require a single activation as input, so SIRENs are not needed to predict missing activations. The conditions $H_i > 1 \wedge W_i > 1$ are included to exclude the trivial case of 1x1 feature maps where there is only one coordinate to sample from.

The next element $u^{(j-1)}$ of the path $\mathcal{P}$ is randomly sampled from the spatial coordinates in the feature map $\mathbf{x}^{(j-1)}$

that make up the receptive field of the activation at $u^{(j)}$. The receptive field of $u^{(j)}$ in the previous layer is given by the discrete interval

$$\mathcal{R}^{(j-1)}(u^{(j)}) = [u^{(j)} \cdot s_j, \ldots, u^{(j)} \cdot s_j + k_j - 1], \quad (8)$$

where the bracket notation denotes the inclusive range of integer indices along each coordinate dimension, and the multiplication–addition operation is applied to each coordinate of the tuple separately, i.e.,

$$u^{(j)} \cdot s_j + k_j - 1 = (h^{(j)} \cdot s_j + k_j - 1,\ w^{(j)} \cdot s_j + k_j - 1).$$

A coordinate $u^{(j-1)}$ is then randomly sampled from $\mathcal{R}^{(j-1)}(u^{(j)})$, and this procedure is repeated until the first-layer coordinate $u^{(1)}$ and its receptive field $\mathcal{R}^{(0)}$ are obtained.

3.4.2 SIREN-assisted Feedforward

Given an input image $\mathbf{x}^{(0,m)}$, we compute the predicted label $\hat{\mathbf{y}}^m$ through SIREN-assisted feedforward by computing the estimate activations $(\tilde{\mathbf{x}}^1_{u^{(1)}}, \tilde{\mathbf{x}}^2_{u^{(2)}}, \ldots, \tilde{\mathbf{x}}^j_{u^{(j)}})$ at the spatial coordinates that comprise the path $\mathcal{P}$. For notational clarity, we omit the image index m from $\mathbf{x}^{(0,m)}$, $\mathbf{x}^{(i,m)}$ and $\hat{\mathbf{y}}^m$ in equations 9 up to and including 15. The index m is retained for the SIREN parameters $\phi^{(i,m)}$.

Computing $\tilde{\mathbf{x}}^1_{u^{(1)}}$ The slice of the input image $\mathbf{x}^0$ corresponding to the receptive field $\mathcal{R}^{(0)}$ is provided as input to the first convolutional layer:

$$\tilde{\mathbf{x}}^1_{u^{(1)}} = \mathcal{C}^1(\mathcal{R}^{(0)}; \theta^1) \quad (9)$$

Computing $\tilde{\mathbf{x}}^i_{u^{(i)}}$ We first approximate the activations in the receptive field $\mathcal{R}^{(i-1)}(u^{(i)})$ using the SIREN:

$$\hat{\mathbf{x}}^{(i-1)}_{\mathcal{R}^{(i-1)}(u^{(i)})} = \mathcal{S}\Big(\mathcal{R}^{(i-1)}(u^{(i)}); \phi^{(i-1,m)}\Big) \quad (10)$$

The SIREN-predicted activation at the coordinate $u^{(i-1)}$ is replaced with by $\tilde{\mathbf{x}}^{i-1}_{u^{(i-1)}}$ while keeping the SIREN-predicted values for the remaining receptive field coordinates:

$$\mathbf{z}^{(i-1)}_v = \begin{cases} \hat{\mathbf{x}}^{(i-1)}_v & \text{if } v \in \mathcal{R}^{(i-1)}(u^{(i)}) \setminus \{u^{(i-1)}\}, \\ \tilde{\mathbf{x}}^{(i-1)}_{u^{(i-1)}} & \text{if } v = u^{(i-1)} \end{cases} \quad (11)$$

with $v \in \mathcal{R}^{(i-1)}(u^{(i)})$. Here, $\tilde{\mathbf{x}}^{(i-1)}_{u^{(i-1)}}$ is the activation obtained via convolution, while $\hat{\mathbf{x}}^{(i-1)}_v$ denotes the SIREN-predicted activation (Eq. 10).

$\mathbf{z}^{(i-1)}_v$ is then provided as input to the i-th convolutional layer to compute the activation at $u^{(i)}$:

$$\tilde{\mathbf{x}}^i_{u^{(i)}} = \mathcal{C}^i\Big(\mathbf{z}^{(i-1)}_{\mathcal{R}^{(i-1)}(u^{(i)})}; \theta^i\Big) \quad (12)$$

Computing the predicted label $\hat{\mathbf{y}}$ We use the SIREN at layer j to approximate the full feature map:

$$\hat{\mathbf{x}}^{(j)}_{[H_j,W_j]} = \mathcal{S}\Big([H_j, W_j]; \phi^{(j,m)}\Big), \quad (13)$$

where $[H_j, W_j]$ denotes the contiguous discrete interval of all spatial coordinates in the feature map at layer j. The SIREN-predicted activation at the path coordinate $u^{(j)}$ is then replaced with the exact computed activation $\tilde{\mathbf{x}}^{(j)}_{u^{(j)}}$:

$$\mathbf{z}^{(j)}_v = \begin{cases} \hat{\mathbf{x}}^{(j)}_v & \text{if } v \in [H_j, W_j] \setminus \{u^{(j)}\}, \\ \tilde{\mathbf{x}}^{(j)}_{u^{(j)}} & \text{if } v = u^{(j)}, \end{cases} \quad v \in [H_j, W_j]. \quad (14)$$

This modified feature map $\mathbf{z}^{(j)}$ is then used as input to the subsequent convolutional layers to obtain $\hat{\mathbf{y}}$, without further SIREN assistance:

$$\hat{\mathbf{y}} = \mathcal{C}^L\Big(\cdots \mathcal{C}^{(j+1)}\Big(\mathbf{z}^{(j)}; \theta^{(j+1)}\Big) \cdots; \theta^{(L)}\Big) \quad (15)$$

Extension to multiple paths. In practice, multiple random paths are sampled. When paths intersect, the activations at the intersection are averaged before computing activations at the subsequent layers.

Feedforward during inference phase. Unlike during the training phase, there are no SIRENs trained for input images during the inference phase. Therefore, during inference the network performs standard feedforward without SIREN-assisted predictions instead of SIREN-assisted feedforward.

3.4.3 Loss Computation and Weight Updates

During SIREN-assisted feedforward the loss function used to supervise the SIRENs :

$$\mathcal{L}_{\text{SIREN}} = \sum_{b \in B} \sum_{i=1}^{j} \big\| \tilde{\mathbf{x}}^{(i,b)}_{u^{(i)}} - \hat{\mathbf{x}}^{(i,b)}_{u^{(i)}} \big\|^2, \quad (16)$$

where B is the set of indices of the training images of the current batch.

The CNN is supervised with the standard cross-entropy loss

$$\mathcal{L}_{\text{CE}} = \text{CE}(\hat{\mathbf{y}}, \mathbf{y}), \quad (17)$$

where $\hat{\mathbf{y}}$ is the predicted logit vector and $\mathbf{y}$ the ground truth label.

Gradients $\frac{\partial \mathcal{L}_{\text{CE}}}{\partial \phi}$ are not used to update the SIRENs parameters ϕ, they are discarded. Similarly, $\frac{\partial \mathcal{L}_{\text{SIREN}}}{\partial \theta}$ are not used to update the CNN weights θ.

4. Experiments

All SIREN models used in the experiments have three hidden layers with a width of 256. They are optimized using the Adam optimizer [33] with a learning rate of 1×10^{-4}. The SIREN hyperparameter ω_0 is set to 30, and the `outermost_linear` flag is set to `True`, meaning that the final layer is linear rather than sinusoidal. Results of the experiments are evaluated both qualitatively, through visual comparisons, and quantitatively, using standard numerical metrics.

4.1. Exp 1: How well can SIRENs fit static feature maps produced by frozen CNNs?

Prior work has demonstrated that SIRENs can effectively compress and reconstruct continuous signals from natural domains, such as real-world images, audio signals, and 3D scenes [60]. However, feature maps produced by CNNs differ from these natural signals. They are not directly captured from the physical world, but are instead computed with learned convolutional kernels and non-linear operations. Hence, it is unclear whether the same SIRENs are also suitable for fitting feature maps.

We validate how well SIRENs fit feature maps while CNN weights are frozen, focusing on two endpoints of training: (1) before training, when convolutional weights are initialized by sampling from the Kaiming normal distribution [24]; and (2) after training, when the weights have converged to minimize the CE-loss for classification tasks.

Implementation details The modified All-CNN architecture is used to generate a set of six intermediate feature maps for each sampled input image at two endpoints of training: before and after training. To obtain the static feature maps before training, input images are passed through the untrained All-CNN, where the convolutional weights are initialized using the Kaiming normal initialization strategy [24], and feature maps after ReLU layers are used as training targets for the SIRENs. Post-training feature maps are collected in the same manner but with a trained All-CNN. The modified All-CNN is trained for 250 epochs on a class-balanced one percent subset of the MNIST [37] training set and optimized with Adam [33] with a learning rate of 1×10^{-3}.

Each feature map is paired with a separate SIREN, which it supervises during training, resulting in six SIRENs trained in parallel for 250 epochs. Since all feature map activations are available, the loss function used to supervise the SIRENs in this experiment is given by Eq. 4 in Section 3.3. After each epoch, the Peak Signal-to-Noise Ratio [15] (PSNR) metric is used to quantify the difference between the target feature maps and the corresponding SIREN reconstructions, where a higher PSNR indicates a higher-quality reconstruction.

Results Figure 2 shows the average PSNR of SIREN reconstructions of intermediate feature map targets produced by each convolutional layer across epochs. Blue curves correspond to the PSNR of feature maps produced by the All-CNN with frozen randomly initialized weights, while orange curves correspond to feature maps from the frozen trained All-CNN. Across all layers, SIRENs achieve high-quality reconstructions, as evidenced by average PSNR scores exceeding 30 dB for both weight initialization conditions. With the exception of SIRENs corresponding to layer 1, the trained CNN feature maps result in both higher initial and final PSNR values, indicating that SIRENs reconstruct learned features more effectively than those from randomly initialized kernels. This gap is most pronounced in layer 6 which has the smallest resolution feature map of 8x8. These results suggest that the optimized convolutional kernels produce feature maps with lower complexity. Table 2 visually compares SIREN reconstructions at the 250 epoch mark with target feature maps for both initialization conditions. The reconstructions closely match their targets, demonstrating that SIRENs can effectively represent feature maps when supervised with Eq. 4, regardless of whether the CNN weights are trained or randomly initialized.

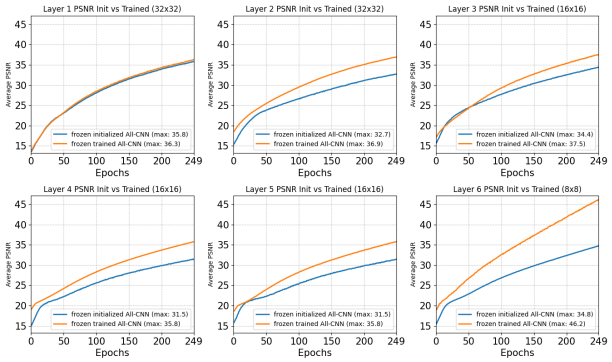

Figure 2. **Exp 1:** Average Peak Signal-to-Noise Ratio of SIREN reconstructions across training epochs for feature maps from the first six convolutional layers. Both initialization conditions yield high-quality reconstructions (PSNR $>$ 30dB), but feature maps from the frozen and trained All-CNN (orange) achieve higher PSNR values compared to those from the randomly initialized network (blue).

4.2. Exp 2: SIRENs fitting changing feature maps

During training, a neural network's weights are continuously updated to optimize a loss function, causing the feature maps produced for the same input to evolve over time. While Exp 1 evaluated SIREN reconstructions of fixed static feature maps from a frozen network, this ex-

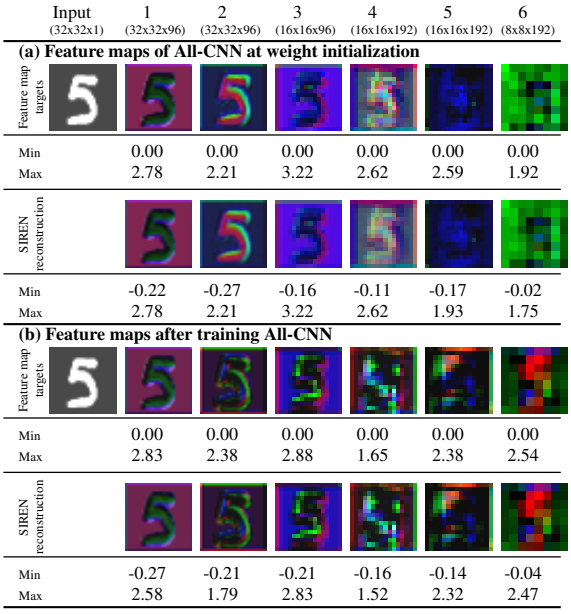

Table 2. **Exp 1:** Visual comparison of SIREN reconstructions and target feature maps. RGB visualization of the first three channels from each layer's feature map, with minimum and maximum activation values.**(a)** Feature maps from the All-CNN at weight initialization (Kaiming normal) and their SIREN reconstructions after 250 epochs of training. **(b)** Feature maps from the trained All-CNN and their SIREN reconstructions after 250 epochs of training. Minimum activation values of target feature maps are zero due to the ReLU nonlinearity. SIREN reconstructions closely match the target feature maps in cases **(a)** and **(b)**.

periment examines whether SIRENs can keep up with and reconstruct moving target feature maps from an All-CNN that is actively being trained.

Implementation Details The experimental setup follows that of Experiment 1, with one difference: instead of training SIRENs on static feature maps from a frozen network, the target feature maps now evolve during training. As the All-CNN's parameters θ^i are updated, the computed target feature maps $\mathbf{x}^{(i,m)} = \mathcal{C}^i(\mathbf{x}^{(i-1,m)}; \theta^i)$ change accordingly, creating moving training targets for the SIRENs (see Eq. 2). Both the All-CNN and the six SIRENs per input image are trained in parallel for 250 epochs, with the All-CNN optimized using Adam with learning rate 1×10^{-3} and SIRENs supervised by the loss function in Eq. 4. PSNR is computed at each epoch to measure reconstruction quality.

Results Figure 3 extends Figure 2 of Experiment 1 where SIRENs fit feature maps from frozen, trained (orange) or frozen, initialized All-CNN (blue) with PSNR of SIRENs trained on feature maps from the non-frozen, actively training All-CNN (green). With the exception of Layer 1, we can see that with moving targets, SIRENs have a lower initial PSNR value than in the setting of static targets. This is likely due to Adam optimizer making larger weight updates to the All-CNN weights θ at initial epochs [33]. These larger updates to θ cause the feature maps $\mathbf{x}^{(i,m)}$ to vary substantially across epochs, making them challenging targets for SIREN reconstruction during the initial training phase. By epoch 250, the green curves achieve lower final PSNR values than their static counterparts in Layers 2-5, likely because the feature maps continue changing throughout training. Layer 6 is an exception SIRENs fitting on feature maps from unfrozen All-CNN surpasses SIRENs fitting on feature maps from frozen initialized All-CNN (37.4 $>$ 34.8 dB), suggesting that by epoch 250, the feature maps have sufficiently converged to produce more smoother targets than those from random initialization. Table 3 provides visual comparisons at epoch 250, revealing that SIREN reconstructions are smoother than their targets. This suggest that SIRENs struggle to capture the high-frequency details when the targets are not fixed across epochs.

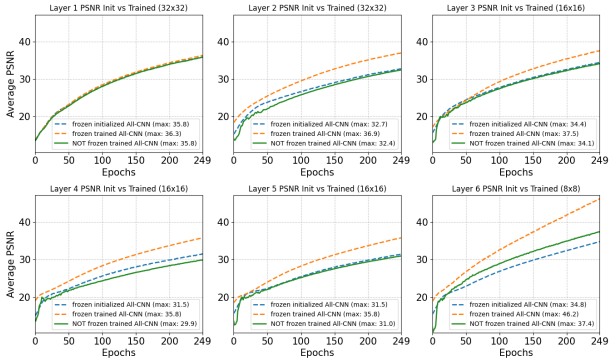

Figure 3. **Exp 2:** Average PSNR of SIREN reconstructions. SIRENs trained on moving targets from a non-frozen All-CNN (green) achieve lower PSNR than those trained on static targets from frozen networks (blue: initialized, orange: trained) in layers 2-5, but surpass frozen initialization in layer 6 where feature maps have converged.

4.3. Exp 3: Training All-CNN via SIREN-assisted feedforward

One of the key factors that makes CNNs effective is their ability to build hierarchical receptive fields, where deeper layers integrate information from increasingly larger spatial regions of the input image. We investigate the alternative SIRENs-assisted feedforward process, where SIRENs predict the activations required to progress through the CNN instead of building the receptive field hierarchically.

Implementation Details To evaluate the effectiveness of SIREN-assisted feedforward, we train two instances of the

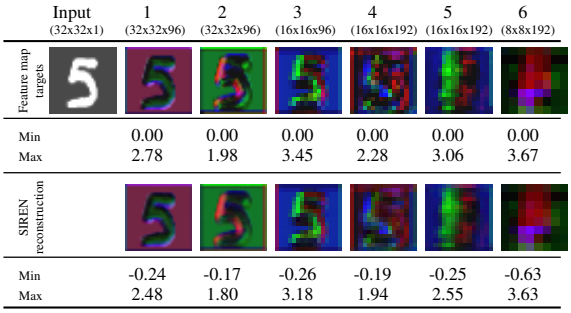

Table 3. **Exp 2:** Visual comparison of SIREN reconstructions trained with moving targets and target feature maps at the 250 epoch mark. Reconstructions closely approximate targets but exhibit slight smoothing due to loss of fine high-frequency details.

modified All-CNN on a class-balanced one percent subset of MNIST for 1500 epochs. Both are optimized using Adam with learning rate 1×10^{-5}. The first uses SIREN-assisted feedforward with 400 random paths per forward pass, and the second uses standard convolution as a baseline. For the SIREN-assisted feedforward run, SIRENs are supervised using the loss from Eq. 16 (Section 3.4.3), which differs from the complete feature map reconstruction loss in Eq. 4 used in Experiments 1 and 2.

Results Figure 4 shows the PSNR of SIREN reconstructions during All-CNN training with SIREN-assisted feedforward. Across all layers, reconstruction quality is substantially lower than in Experiments 1 and 2, with maximum PSNR values ranging from 22.8 dB to 26.6 dB compared to 30+ dB in static target scenarios. Table 4 provides visual confirmation: SIREN reconstructions are noticeably smoother and do not reconstruct the high frequency details present in the target feature maps. This progressive degradation is most evident in deeper layers, where reconstructions become increasingly blurred. The smoothing effect compounds across layers: each SIREN is supervised with activations computed from the smoothed predictions of the previous layer's SIREN, which then propagates to the next layer's input, resulting in cumulative information loss. By Layer 6, the SIREN reconstruction barely resembles the actual feature map.

Figure 5 compares training performance using pixels seen on the x-axis to account for different pixel coverage per epoch: SIREN-assisted feed forward uses 400 random coordinate paths while the baseline processes the entire input. The baseline loss decreases smoothly to 0.576 for training and 0.913 for test, indicating overfitting on the one percent subset of the MNIST training set. In contrast, the loss curve of SIREN-assisted feedforward remains high around 2.274 with minimal train-test gap. The poor accuracy and optimization instability likely stem from SIREN representations failing to track evolving feature maps during All-CNN training, creating a challenging optimization landscape. The baseline is more data-efficient as it achieves 100.00% training accuracy and 95.00% test accuracy with fewer pixels seen, while SIREN-assisted feedforward reaches 21.50% training accuracy and 20.97% test accuracy. Despite the gap in accuracy, the 20.97% test accuracy demonstrates that SIRENs implicitly capture meaningful representations that enable classification accuracy more than double the random guessing baseline of 10%, indicating the approach merits further exploration in future work.

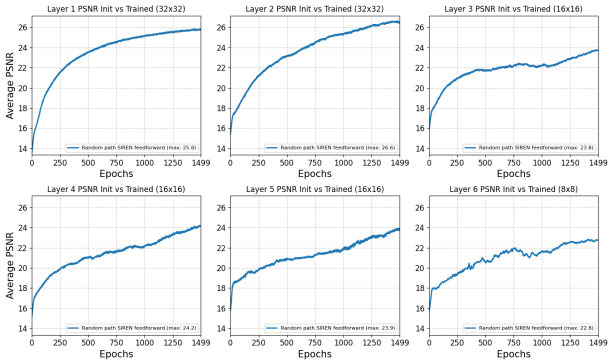

Figure 4. **Exp 3:** Average PSNR of SIREN reconstructions of feature maps across layers when training All-CNN with SIREN-assisted feedforward. At the end of training, the final PSNR of reconstructions ranges from 22.8 dB to 26.6, which indicate significant information loss in relation to target feature maps.

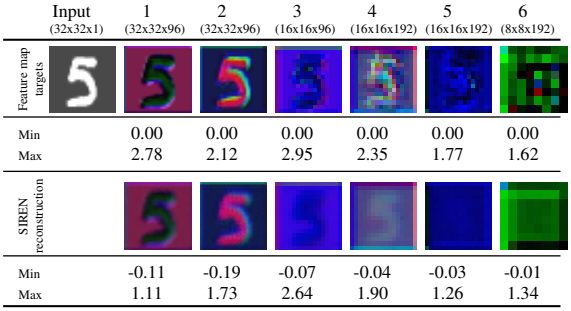

Table 4. **Exp 3:** SIREN reconstructions during SIREN-assisted feedforward training. Progressive smoothing across layers results from each SIREN being supervised with activations computed from the previous layer's blurred predictions. High-frequency details are lost, with degradation most severe in deeper layers.

5. Discussion

We explored the idea of representing intermediate CNN feature maps implicitly with SIRENs. The experimental results indicate that SIRENs can represent feature maps with high fidelity when trained with full supervision, i.e.,

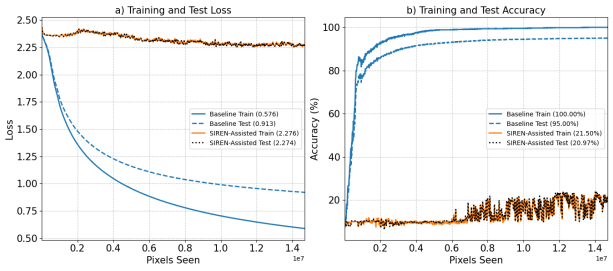

Figure 5. **Exp 3:** Training performance of SIREN-assisted versus baseline feedforward. a) Baseline has achieves lower loss while optimization of loss of SIREN-assisted is more unstable due to more challenging loss landscape. b) Baseline achieves 100% training and 95% test accuracy; SIREN-assisted reaches 21.5% training and 20.97% test accuracy, exceeding random guessing: 10%. X-axis shows pixels seen for fair comparison between methods.

when all pixel values of the target feature maps are available at each training step. When All-CNN is trained with SIREN-assisted feedforward, the classification performance is better than random guessing but substantially lower than when trained the standard way. We believe this gap arises from the increased difficulty of optimizing All-CNN when the SIRENs cannot approximate the evolving activations quickly enough.

Limitations While motivated by the goal of reducing memory usage, the current implementation of SIREN-assisted feedforward introduces significant memory overhead. Specifically, one SIREN is used per feature map per training image, which constrains training to a 1% subset of the MNIST dataset (600 training samples in our experiments). This makes the current proof-of-concept ironically less memory-efficient than standard convolution. Another limitation is that the method has only been conceptualized and tested during the *training phase*. Because the framework relies on the availability of feature maps at each epoch, it is not yet clear how SIREN-assisted feedforward could be used during inference or test time.

Future Work Future research should aim to design improved formulations where SIRENs can be shared or amortized across feature maps and training samples, eliminating the per-image memory overhead. It would also be valuable to extend the framework so that it can operate efficiently during inference, making SIREN-assisted feedforward applicable beyond the training phase. Additionally, performance comparisons are currently unfair because standard convolution benefits from optimized NVIDIA kernel implementations, while SIREN-assisted feedforward does not. Developing dedicated kernelized implementations for SIREN-assisted feedforward would enable a more meaningful comparison. Finally, orthogonal directions such as few-shot learning or meta-learning could further complement the current memory constraints of this framework, potentially allowing SIRENs to generalize feature map representations across images without retraining from scratch.

Our results highlight both the promise and the current challenges of integrating continuous implicit representations into discrete deep learning pipelines. Bridging this gap requires methods that allow implicit and discrete representations to co-evolve efficiently during training. A deeper understanding of this interaction could eventually lead to architectures that combine the expressive power of implicit networks with the optimization stability of standard CNNs.

References

[1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016. 4

[2] Peter Bandi, Oscar Geessink, Quirine Manson, Marcory Van Dijk, Maschenka Balkenhol, Meyke Hermsen, Babak Ehteshami Bejnordi, Byungjae Lee, Kyunghyun Paeng, Aoxiao Zhong, et al. From detection of individual metastases to classification of lymph node status at the patient level: the camelyon17 challenge. *IEEE transactions on medical imaging*, 38(2):550–560, 2018. 1

[3] Benjamin Bergner, Christoph Lippert, and Aravindh Mahendran. Iterative patch selection for high-resolution image recognition, 2023. 1

[4] Emily Berndt, Andrew Molthan, William W Vaughan, and Kevin Fuell. Transforming satellite data into weather forecasts. *Eos*, 98(3):26, 2017. 1

[5] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *European conference on computer vision*, pages 213–229. Springer, 2020. 1

[6] Jang Hyun Cho and Bharath Hariharan. On the efficacy of knowledge distillation. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 4794–4802, 2019. 3

[7] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1, 2016. 3

[8] Alexandre Delplanque, Jérôme Théau, Samuel Foucher, Ghazaleh Serati, Simon Durand, and Philippe Lejeune and. Wildlife detection, counting and survey using satellite imagery: are we there yet? *GIScience & Remote Sensing*, 61(1):2348863, 2024. 1

[9] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009. 1

[10] Emilien Dupont, Adam Goliński, Milad Alizadeh, Yee Whye Teh, and Arnaud Doucet. Coin: Compression with implicit neural representations, 2021. 3

[11] Emilien Dupont, Hyunjik Kim, SM Eslami, Danilo Rezende, and Dan Rosenbaum. From data to functa: Your data point

is a function and you can treat it like one. *arXiv preprint arXiv:2201.12204*, 2022. 3

[12] Emilien Dupont, Hrushikesh Loya, Milad Alizadeh, Adam Goliński, Yee Whye Teh, and Arnaud Doucet. Coin++: Neural compression across modalities, 2022. 3

[13] JR Eyre, W Bell, J Cotton, SJ English, M Forsythe, SB Healy, and EG Pavelin. Assimilation of satellite data in numerical weather prediction. part ii: Recent years. *Quarterly Journal of the Royal Meteorological Society*, 148(743):521–556, 2022. 1

[14] Stephanie Fu, Mark Hamilton, Laura Brandt, Axel Feldman, Zhoutong Zhang, and William T. Freeman. Featup: A model-agnostic framework for features at any resolution, 2024. 3

[15] Rafael Gonzalez and Richard Woods. *Digital Image Processing Global Edition*. Pearson Deutschland, 2017. 6

[16] Jianping Gou, Baosheng Yu, Stephen J Maybank, and Dacheng Tao. Knowledge distillation: A survey. *International journal of computer vision*, 129(6):1789–1819, 2021. 3

[17] Julien Guiot, Akshayaa Vaidyanathan, Louis Deprez, Fadila Zerka, Denis Danthine, Anne-Noelle Frix, Philippe Lambin, Fabio Bottari, Nathan Tsoutzidis, Benjamin Miraglio, et al. A review in radiomics: making personalized medicine a reality via routine imaging. *Medicinal research reviews*, 42(1):426–440, 2022. 1

[18] Deepak K. Gupta, Gowreesh Mago, Arnav Chavan, and Dilip K. Prasad. Patch gradient descent: Training neural networks on very large images, 2023. 1

[19] Dorothy K Hall. Assessment of polar climate change using satellite technology. *Reviews of Geophysics*, 26(1):26–39, 1988. 1

[20] Luwis Suryani Haloho and Asep Adang Supriyadi. Utilization of satellite technology in communication systems, disaster monitoring, border surveillance, and military intelligence: A literature review. *Remote Sensing Technology in Defense and Environment*, 1(1):36–44, 2024. 1

[21] Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding, 2016. 3

[22] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017. 1

[23] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015. 3

[24] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, 2015. 6

[25] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE international conference on computer vision*, pages 1389–1397, 2017. 3

[26] Robert Hecht-Nielsen. On the algebraic structure of feedforward network weight spaces. In Rolf ECKMILLER, editor, *Advanced Neural Computers*, pages 129–135. North-Holland, Amsterdam, 1990. 3

[27] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015. 3

[28] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015. 4

[29] Hanen Issaoui, Asma ElAdel, and Mourad Zaied. Object detection using convolutional neural networks: a comprehensive review. In *2024 IEEE 27th International Symposium on Real-Time Distributed Computing (ISORC)*, pages 1–6. IEEE, 2024. 1

[30] Chen Jin, Ryutaro Tanno, Thomy Mertzanidou, Eleftheria Panagiotaki, and Daniel C. Alexander. Learning to downsample for segmentation of ultra-high resolution images, 2022. 1

[31] Mohamed Khalifa and Mona Albadawy. Ai in diagnostic imaging: Revolutionising accuracy and efficiency. *Computer Methods and Programs in Biomedicine Update*, 5:100146, 2024. 1

[32] Chiheon Kim, Doyup Lee, Saehoon Kim, Minsu Cho, and Wook-Shin Han. Generalizable implicit neural representations via instance pattern composers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11808–11817, June 2023. 3

[33] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017. 6, 7

[34] Miltiadis Kofinas, Boris Knyazev, Yan Zhang, Yunlu Chen, Gertjan J. Burghouts, Efstratios Gavves, Cees G. M. Snoek, and David W. Zhang. Graph neural networks for learning equivariant representations of neural networks, 2024. 3

[35] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90, May 2017. 1

[36] Michelle A LaRue, Seth Stapleton, and Morgan Anderson. Feasibility of using high-resolution satellite imagery to assess vertebrate wildlife populations. *Conservation biology*, 31(1):213–220, 2017. 1

[37] Yann LeCun. The MNIST Database of Handwritten Digits. http://yann.lecun.com/exdb/mnist/, 1998. 6

[38] Shaohui Lin, Rongrong Ji, Chenqian Yan, Baochang Zhang, Liujuan Cao, Qixiang Ye, Feiyue Huang, and David Doermann. Towards optimal structured cnn pruning via generative adversarial learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2790–2799, 2019. 3

[39] Ting Liu and Xiaojun Yang. Monitoring land changes in an urban area using satellite imagery, gis and landscape metrics. *Applied geography*, 56:42–54, 2015. 1

[40] Yunxiang Liu and Shixuan Cai. Advancements in weather forecasting: A review of satellite imagery and ai integration. In *2024 9th International Conference on Intelligent Informatics and Biomedical Sciences (ICIIBMS)*, volume 9, pages 340–344. IEEE, 2024. 1

[41] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s, 2022. 1

[42] Luca De Luigi, Adriano Cardace, Riccardo Spezialetti, Pierluigi Zama Ramirez, Samuele Salti, and Luigi Di Stefano. Deep learning on implicit neural representations of shapes, 2023. 3

[43] Jian-Hao Luo and Jianxin Wu. An entropy-based pruning method for cnn compression. *arXiv preprint arXiv:1706.05791*, 2017. 3

[44] Jian-Hao Luo, Hao Zhang, Hong-Yu Zhou, Chen-Wei Xie, Jianxin Wu, and Weiyao Lin. Thinet: Pruning cnn filters for a thinner net. *IEEE transactions on pattern analysis and machine intelligence*, 41(10):2525–2538, 2018. 3

[45] Heather J Lynch. Satellite remote sensing for wildlife research in the polar regions. *Marine Technology Society Journal*, 57(3):43–50, 2023. 1

[46] Sushobhan Majumdar. The role of remote sensing and gis in military strategy to prevent terror attacks. *Intelligent Data Analytics for Terror Threat Prediction: Architectures, Methodologies, Techniques and Applications*, pages 79–94, 2021. 1

[47] K Malarvizhi, S Vasantha Kumar, and P Porchelvan. Use of high resolution google earth satellite imagery in landuse map preparation for urban related applications. *Procedia Technology*, 24:1835–1842, 2016. 1

[48] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space, 2019. 3

[49] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis, 2020. 3

[50] Jinlai Ning and Michael Spratling. The importance of anti-aliasing in tiny object detection. In Berrin Yanıkoğlu and Wray Buntine, editors, *Proceedings of the 15th Asian Conference on Machine Learning*, volume 222 of *Proceedings of Machine Learning Research*, pages 975–990. PMLR, 11–14 Nov 2024. 1

[51] Samuele Papa, Riccardo Valperga, David Knigge, Miltiadis Kofinas, Phillip Lippe, Jan-Jakob Sonke, and Efstratios Gavves. How to train neural field representations: A comprehensive study and benchmark, 2024. 3

[52] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation, 2019. 3

[53] Wonpyo Park, Dongju Kim, Yan Lu, and Minsu Cho. Relational knowledge distillation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 3967–3976, 2019. 3

[54] Nasim Rahaman, Aristide Baratin, Devansh Arpit, Felix Draxler, Min Lin, Fred A. Hamprecht, Yoshua Bengio, and Aaron Courville. On the spectral bias of neural networks, 2019. 3

[55] Naim Rochmawati, Chastine Fatichah, Bilqis Amaliah, Agus Budi Raharjo, Frédéric Dumont, Emilie Thibaudeau, and Cédric Dumas. Deep learning-based lesion detection in endoscopy: A systematic literature review. *IEEE Access*, 13:43532–43556, 2025. 1

[56] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015. 1

[57] Zubair Saing, Herry Djainal, and Saiful Deni. Land use balance determination using satellite imagery and geographic information system: case study in south sulawesi province, indonesia. *Geodesy and Geodynamics*, 12(2):133–147, 2021. 1

[58] Evan Shelhamer, Jonathan Long, and Trevor Darrell. Fully convolutional networks for semantic segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 39(4):640–651, 2016. 1

[59] Bharat Singh, Mahyar Najibi, and Larry S. Davis. Sniper: Efficient multi-scale training, 2018. 1

[60] Vincent Sitzmann, Julien N. P. Martel, Alexander W. Bergman, David B. Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions, 2020. 1, 2, 3, 4, 6

[61] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net, 2015. 4

[62] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014. 4

[63] Yannick Strümpler, Janis Postels, Ren Yang, Luc van Gool, and Federico Tombari. Implicit neural representations for image compression, 2022. 3

[64] Hadi Sulistyo, Unggul Satrio Yudhotomo, Hazen Alrasyid, Moh Fakhruddin Farhan, Kasim Kasim, Mala Utami, Winka Wino Yunanda, Fiorentina Nulhakim, Salsa Ayuning Tias, and George Royke Deksino. Remote sensing satellite technology to determine the center of the maritime defense logistics route for securing the indonesian capital city (ikn). In *2022 International Conference on Advanced Computer Science and Information Systems (ICACSIS)*, pages 225–234. IEEE, 2022. 1

[65] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions, 2014. 1

[66] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015. 1

[67] Matthew Tancik, Pratul P. Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan T. Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains, 2020. 3

[68] Beth Tellman, Jonathan A Sullivan, Catherine Kuhn, Albert J Kettner, Colin S Doyle, G Robert Brakenridge, Tyler A Erickson, and Daniel A Slayback. Satellite imaging reveals increased proportion of population exposed to floods. *Nature*, 596(7870):80–86, 2021. 1

[69] Mitko Veta, Yujing J. Heng, Nikolas Stathonikos, Babak Ehteshami Bejnordi, Francisco Beca, Thomas Wollmann, Karl Rohr, Manan A. Shah, Dayong Wang, Mikael Rousson, Martin Hedlund, David Tellez, Francesco Ciompi, Erwan Zerhouni, David Lanyi, Matheus Viana, Vassili Kovalev, Vitali Liauchuk, Hady Ahmady Phoulady, Talha Qaiser, Simon Graham, Nasir Rajpoot, Erik Sjöblom, Jesper Molin, Kyunghyun Paeng, Sangheum Hwang, Sunggyun Park, Zhipeng Jia, Eric I-Chao Chang, Yan Xu, Andrew H. Beck, Paul J. van Diest, and Josien P.W. Pluim. Predicting breast tumor proliferation from whole-slide images: The tupac16 challenge. *Medical Image Analysis*, 54:111–121, 2019. 1

[70] Zeyu Wang, Chen Li, Huiying Xu, and Xinzhong Zhu. Mamba yolo: Ssms-based yolo for object detection. *arXiv preprint arXiv:2406.05835*, 2024. 1

[71] Chao Yang, Xin Lu, Zhe Lin, Eli Shechtman, Oliver Wang, and Hao Li. High-resolution image inpainting using multi-scale neural patch synthesis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017. 1

[72] Jun Yang, Peng Gong, Rong Fu, Minghua Zhang, Jingming Chen, Shunlin Liang, Bing Xu, Jiancheng Shi, and Robert Dickinson. The role of satellite remote sensing in climate change studies. *Nature climate change*, 3(10):875–883, 2013. 1

[73] Sean I. Young, Wang Zhe, David Taubman, and Bernd Girod. Transform quantization for cnn compression. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(9):5700–5714, 2022. 3

[74] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks, 2013. 3

References

[1] Emilien Dupont et al. *From data to functa: Your data point is a function and you can treat it like one*. 2022. arXiv: 2201.12204 [cs.LG]. URL: https://arxiv.org/abs/2201.12204.

[2] Reza Ekalantar. *Receptive Fields in Deep Convolutional Networks*. https://medium.com/@rekalantar/receptive-fields-in-deep-convolutional-networks-43871d2ef2e9. Accessed: Thursday 4th December, 2025. 2024.

[3] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[4] Osman Semih Kayhan and Jan C. van Gemert. *On Translation Invariance in CNNs: Convolutional Layers can Exploit Absolute Spatial Location*. 2020. arXiv: 2003.07064 [cs.CV]. URL: https://arxiv.org/abs/2003.07064.

[5] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG]. URL: https://arxiv.org/abs/1412.6980.

[6] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira et al. Vol. 25. Curran Associates, Inc., 2012. URL: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[7] Y. LeCun et al. "Backpropagation Applied to Handwritten Zip Code Recognition". In: *Neural Computation* 1.4 (1989), pp. 541–551. DOI: 10.1162/neco.1989.1.4.541.

[8] Y. Lecun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. DOI: 10.1109/5.726791.

[9] Yann LeCun and Yoshua Bengio. "Convolutional networks for images, speech, and time series". In: *The handbook of brain theory and neural networks* (1998).

[10] Yann LeCun, Koray Kavukcuoglu, and Clement Farabet. "Convolutional networks and applications in vision". In: *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*. 2010, pp. 253–256. DOI: 10.1109/ISCAS.2010.5537907.

[11] Ben Mildenhall et al. *NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis*. 2020. arXiv: 2003.08934 [cs.CV]. URL: https://arxiv.org/abs/2003.08934.

[12] Jeong Joon Park et al. *DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation*. 2019. arXiv: 1901.05103 [cs.CV]. URL: https://arxiv.org/abs/1901.05103.

[13] Vincent Sitzmann et al. *Implicit Neural Representations with Periodic Activation Functions*. 2020. arXiv: 2006.09661 [cs.CV]. URL: https://arxiv.org/abs/2006.09661.