Master thesis
Embedded Systems
Software and Technology

ARC: Anchored Representation Clouds for High-Resolution INR Classification

Joost Luijmes

A thesis submitted to the Delft University of Technology in partial fulfilment of the requirements for the degree of Master of Science in Embedded Systems.

Delft University of Technology
The Netherlands
December, 2024

Joost Luijmes. ARC: Anchored Representation Clouds for High-Resolution INR Classification (2024)

Supervisors	Dr. J.C. van Gemert MSc. A.S. Gielisse
Thesis Committee	Dr. J.C. van Gemert Prof. Dr. E. Eisemann

TU Delft

I Acknowledgements

I would like to express my gratitude to my supervisors Jan van Gemert and Sander Gielisse. Thank you for your invaluable guidance throughout this thesis. In our meetings, both formal and informal, I gained so much insight in what meaningful research entails. This thesis often raised unexpected results but your drive to dig deeper and find answers was always a source of motivation. In extension, I would like to thank Elmar Eisemann for agreeing to be a part of the thesis committee. Lectures from both you and Jan on the intersection of visual media and computer science inspired me to pursue this topic.

Many thanks go out to my family, especially my parents, who never doubted my pursuit of a MSc degree at a university, a journey which seemed so daunting at first. Your love and support was always tangible to me. Thank you Linsey, for being there in moments of joy, despair, and otherwise. I would not have managed without your support.

Joost Luijmes,
Delft, December 2024

Contents

1 Introduction

Image classification is a foundational computer vision problem that over the past 15 years has symbiotically driven major advancements in the field of deep learning. However, unresolved challenges persist, particularly with the growing prevalence of high-resolution images. These images demand significant computational resources to process, whether they come from mobile phones, medical diagnostics, or autonomous vehicles.

Alongside image classification research, the advancement of computer vision has led to the emergence of the Implicit Neural Representation (INR) [18, 14, 15]. An INR is a neural network which, given a pixel coordinate, learns to output the corresponding colour. After learning how to reproduce the image, the INR has effectively encoded the entire image inside its weights. In contrast to images, an INR can freely distribute its capacity over the image content. This way, the INR allows for a compressed representation of the original image.

This raises the question whether INRs can be used as alternative representations of images in image classification tasks. By transforming our objective from image classification to INR classification, we may mitigate the typical issues that image classifiers face. Current INR classifiers are limited to low-resolution images and possess limited robustness against image translation. In finding a way to classify images outside of image-space, we have developed a novel type of INR called ARC, paired with an associated INR classification method. In brief, we transform images into point clouds, coupling the INR's latent space to the image coordinate space. This spatial locality provides several advantages, including improved interpretability, enhanced data augmentation opportunities, and compatibility with point cloud classification methods, which serve as indirect image classifiers.

The main findings in this thesis are presented in Chapter 3 in the form of a scientific paper in CVPR format. Though the paper is a stand-alone work, Chapter 2 discusses foundational concepts and terminology from the paper which the reader may find useful.

2 Background

2.1 Machine Learning

Like any other algorithm, a machine learning (ML) algorithm aims to produce relevant outputs when given some inputs. For a regular algorithm, we would define a series of instructions that describe exactly how some input becomes an output. In machine learning, things work differently. A structure is defined wherein variable parameters affect how the input is manipulated. A change in these parameters will bring about a change to the outputs of the algorithm. As such, in machine learning, one of the main objectives is to obtain a set of parameters that enable the machine learning algorithm to perform optimally at its task.

How this set of parameters is found leads us to *supervised* machine learning. In supervised ML, we iteratively improve the parameters via a process called 'training' or 'fitting' whereby the ML algorithm seemingly 'learns' to produce better outputs each iteration in which the parameters are updated. The term supervised refers to how the algorithm is given feedback; we let the algorithm compute an output given some input. Then output is compared to the expected output, whereby we quantify the error. This error is then used to update the parameters. In other words, we continually *supervise* the algorithm during its training phase. To perform this supervision we need a dataset; a set of inputs and corresponding outputs with which we can supervise and train the algorithm.

For instance, a dataset may be a collection of cat and dog images with corresponding labels. To humans it is obvious which image we would classify as 'dog' or 'cat', but to the machine learning algorithm, all these images are just sets of RGB pixels. When the data is labelled, the algorithm can receive useful feedback allowing it to learn what image characteristics differ one set of pixels from another, and consequently, what differentiates dogs from cats. The way such feedback leads to changes in the algorithm's parameters is described in *Training Mechanics*, page 4.

Machine learning is a broad field that embodies many types of algorithms and learning methods. In this thesis, the scope is limited to a particular type of ML method named neural networks.

2.2 Neural Networks

Neural networks (NNs) are a class of machine learning algorithms. Put simply, an NN consists of many simple functions that are chained together to form a complex composite function, parametrised by *learnable* parameters. A particular composition of functions is named an 'architecture'. The architecture is typically fixed in place, whereas the parameters will change during training. In the particular case where a set of parameters is not iterated on during training, they are 'frozen'.

The simple function that a typical neural network is built from, is a first-degree polynomial $ax+b$ wrapped in a non-linear function ϕ.

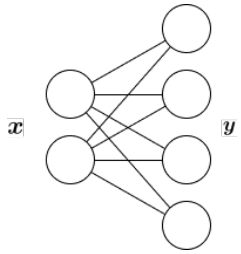
$$h(x) = \phi(ax + b) \tag{1}$$

The parameters a and b are 'trainable', *i.e.* it is *these* parameters that will be updated throughout the neural network training process (see *Training Mechanics*, page 4). ϕ is a non-linear function called the 'activation function'. The activation function is typically predefined and contains no trainable properties. Without ϕ, h would be a linear function regardless of the values that parameters a and b attain. Linear functions are not very flexible however, so for non-trivial tasks ϕ is required to obtain a useful NN.

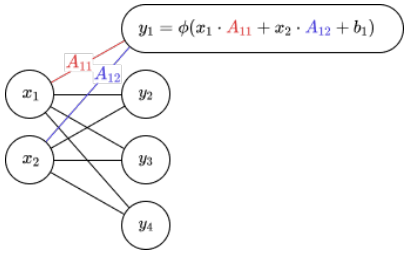
h can easily be extended to multi-variable inputs and outputs.

$$h : \mathbb{R}^{d_{\text{in}}} \mapsto \mathbb{R}^{d_{\text{out}}} \quad h(\boldsymbol{x}) = \phi(\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}) \tag{2}$$

Where $\boldsymbol{A} \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$ is called the 'weight' matrix and $\boldsymbol{b} \in \mathbb{R}^{d_{\text{out}}}$ is called the 'bias' vector. Again, these are the parameters that will be updated throughout the training process. h can now take in a vector, like an RGB colour $\boldsymbol{x} \in \mathbb{R}^3$ or an entire (but flattened) black-and-white image $\boldsymbol{x} \in \mathbb{R}^{\text{Height} \cdot \text{Width}}$, producing a vector output $\boldsymbol{y} \in \mathbb{R}^{d_{\text{out}}}$. Eq. (2) describes a single 'linear layer' with an activation function ϕ. A common visualisation of a linear layer is depicted in Fig. 1a, where an input vector $\boldsymbol{x} \in \mathbb{R}^2$ is mapped to an output vector $\boldsymbol{y} \in \mathbb{R}^4$. The output of a single node, *e.g.* $\boldsymbol{y}_0$, is called an 'activation'.



(a) A single linear layer.



(b) The underlying operations hiding beneath the typical depiction of a single linear layer.

Figure 1: An illustration of h Eq. (2). A linear layer maps an input vector $\boldsymbol{x} \in \mathbb{R}^2$ to an output vector $\boldsymbol{y} \in \mathbb{R}^4$.

Though the representation of a neural network layer in Fig. 1a is commonplace in literature and education, it hides some intricacies (Fig. 1b). Nodes denote a scalar value, like how the leftmost two nodes depict the input vector $\boldsymbol{x}$. The next layer of nodes, and the edges towards them, depict the application of h Eq. (2). Edges represent multiplication with a value from the weight matrix $\boldsymbol{A}$. A node depicts an activation: the summation of incoming values followed by adding a bias term from the bias vector $\boldsymbol{b}$ and the subsequent application of the activation function ϕ.

As alluded to at the start of this section, a neural network is formed when functions h (Eq. (2)) are composed together. A simple neural network can hence be defined as $f_\theta(\boldsymbol{x}) = h^L(h^{i-L}(...(h^2(h^1(\boldsymbol{x})))))$. Each layer has its own learnable weight and bias parameters, which are collectively referred to as the 'weights' of a network, or θ.

Straightforward chaining of linear layers creates a type of NN called a multi-layer perceptron (MLP). An MLP mapping an input vector $\boldsymbol{x} \in \mathbb{R}^2$ to an output vector $\boldsymbol{y} \in \mathbb{R}^4$ is visualised in Fig. 2.

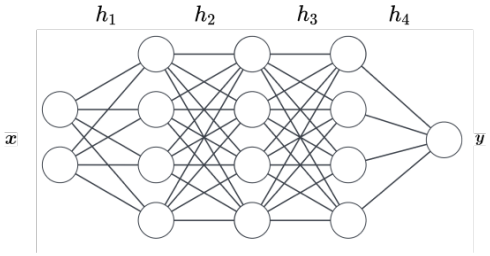


Figure 2: An MLP mapping $\boldsymbol{x} \in \mathbb{R}^2$ to an output vector $\boldsymbol{y}$.

Input data $\boldsymbol{x}$ is 'fed' into the first layer h^1 of the neural network, producing an intermediary result. This intermediary result then serves as the input to the second layer h^2, and so on. When the final layer is reached, the network's output is a heavily transformed version of what was input into the neural network. This act of propagating a unit of data from beginning to end is called a 'forward pass'. Generally, adding more layers to a network or increasing the nodes per layer increases an NN's ability to model complex functions. This ability is referred to as the 'capacity' of an NN. An NN with several layers makes for a 'deep' network, giving rise to the term 'deep learning'. The high-dimensional space that transforms the neural network input is also referred to as the latent- or weight-space.

An MLP is a straightforward architecture and sufficiently powerful to solve complex tasks, such as classifying digits from images. For this example, let us consider the MNIST dataset [12] of 60k images of single handwritten digits from 0 to 9 inclusive. We need to somehow input a 2D image and obtain a single value from the MLP. As the in- and outputs of the MLP must be vectors, the input image $\mathbb{R}^{28\times28}$ is flattened to $\mathbb{R}^{784}$. We can let the final layer map to a scalar output $y \in \mathbb{R}^1$, but this would imply that an image depicting '1' is more similar to '0' and '2' than *e.g.* '7'. In contrast, an input image of a '1' looks more like a '7' than either a '0' or a '2'. In classification problems, it would be bad practice to imprint some type of ordering into the classes. Instead, classification problems such as this one convert the label to a *one-hot encoding* (Figure 3). For our MNIST classification task, the final layer of the network should now map to $y \in \mathbb{R}^{10}$, which can be interpreted as the NN computing the likeliness of each class depicting the digit. On MNIST, a simple MLP can obtain a classification accuracy of 97.2% [21].

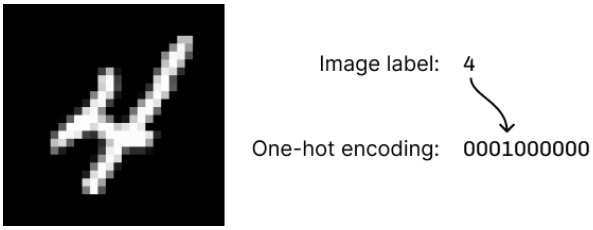


Figure 3: Depiction of how a one-hot encoding is obtained. The image label serves as an index into a zero-vector of length equal to the number of classes in the dataset.

2.2.1 End-to-end Learning

In the example of digit classification, we passed the entire image to the NN and expected it to learn why the image denotes a particular digit. Which image features are relevant to decide the digit type is left entirely to the NN. This is an example of end-to-end learning, whereby we do not explicitly tell the NN what features to look for but instead trust the NN's flexibility to learn these during training. Before deep learning took on a prominent role in computer vision, classification systems would be provided with hand-crafted features designed to capture relevant patterns in the input data such as edge detection, histogram of oriented gradients (HOG), or scale-invariant feature transform (SIFT). These features were then fed into a separate classifier, such as a support vector machine (SVM) or a decision tree, to perform the final classification. In end-to-end learning, feature extraction and feature classification are intertwined. By optimising the NN to minimise a loss function, the NN learns which features are relevant to the task at hand. For example, in digit classification, the NN might learn that round strokes are common in images depicting 8s and 0s, whereas vertical strokes pertain to 1s.

Notably, the topic of this thesis – INR classification – violates the end-to-end learning mechanism. This is described in more detail in *Implicit Neural Representations*, page 8.

2.3 Training Mechanics

Training a neural network is performed by repeating the following procedure.

1. Perform a forward pass to compute the output $\hat{\boldsymbol{y}}$
2. Compute the loss value $\mathcal{L}(\hat{\boldsymbol{y}}, \boldsymbol{y})$
3. Compute how each weight contributes to $\mathcal{L}(\hat{\boldsymbol{y}}, \boldsymbol{y})$ using back-propagation
4. Update all weights proportional to their influence on $\mathcal{L}(\hat{\boldsymbol{y}}, \boldsymbol{y})$ by following the optimiser's rules

Steps 2, 3 and 4 will be expanded upon in this chapter.

2.3.1 Loss values

For a given input, an untrained network will likely produce an output $\hat{\boldsymbol{y}}$ that is far off of the ground truth label $\boldsymbol{y}$. The difference between $\hat{\boldsymbol{y}}$ and $\boldsymbol{y}$ can be quantified by a loss function $\mathcal{L}(\hat{\boldsymbol{y}}, \boldsymbol{y})$. For instance, if $\hat{y}, y \in \mathbb{R}^1$, the mean squared error (MSE) could be used.

$$\mathcal{L}(\hat{y}, y) = (\hat{y} - y)^2 \tag{3}$$

If $\hat{y} = y$, the loss value is 0 *i.e.* the output could not be more correct. However, if $\hat{y}$ deviates from y, the error will grow quadratically, thereby punishing large deviations more than small deviations.

If $\hat{\boldsymbol{y}}, \boldsymbol{y} \in \mathbb{R}^n$, $n > 1$, the multivariable MSE loss is defined as follows.

$$\mathcal{L}(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 = \frac{1}{n} \|\hat{\boldsymbol{y}} - \boldsymbol{y}\|_2^2 \tag{4}$$

What remains is how a loss value is converted into a meaningful change to the network's parameters, such that it improves on its task. The missing link between loss value and parameters back-propagation.

2.3.2 Back-propagation

Say we have performed a forward pass for a given input and have obtained an output. With the loss function, the 'goodness' of this output is computed. How has each weight a of the NN contributed to this 'goodness'? The goal of back-propagation is to quantify this, so we can later update each weight into the right direction. More specifically, how does a particular weight a affect $\hat{\boldsymbol{y}}$, and consequently, the loss function $\mathcal{L} = (\hat{\boldsymbol{y}}, \boldsymbol{y})$. This study of change naturally leads to derivatives, or more specifically, partial derivatives as a is just one of many weights that contribute to the loss.

The derivative of the loss function (abbreviated to L) with respect to a can be expressed as:

$$\frac{\partial L}{\partial a} = \frac{\partial}{\partial a} \frac{1}{n} \| f_\theta(\boldsymbol{x}) - \boldsymbol{y} \|_2^2 \tag{5}$$

where $f_\theta(\boldsymbol{x})$ is the output of the neural network parametrised by θ where $a \in \theta$.

Say a is not in the final layer of the NN, but instead in the second-to-last layer. a will not directly influence $\hat{\boldsymbol{y}}$ and L but instead only affect the inputs of the final layer. This indirection naturally translates to the chain rule of derivates:

$$\frac{\partial L}{\partial a} = \frac{\partial L}{\partial z} \cdot \frac{\partial z}{\partial a} \tag{6}$$

where z represents any intermediate node between a and the loss value L. By recursively applying the chain rule, back-propagation computes how changes in a ripple through the network and affect the loss. Back-propagation Eq. (6) also enforces an order of operations: if a is a parameter in the first layer of the MLP, all intermediary partial derivatives of the parameters between a and L are to be computed first. Whereas we go forwards in the network during a forward pass, we go backwards during the back-propagation algorithm, computing all partial derivates for each learnable parameter along the way. To generalise this to multi-variable cases, we consider the partial gradient. For the sake of keeping the explanation simple, we omit this step. Importantly, to compute the derivatives, we require the activation of each intermediate node that was computed during the forward pass. This can lead to particularly high memory demands for large models, large intermediate values, or large input data.

2.3.3 Optimisers

Informed by the back-propagation algorithm, the optimiser will update all the parameters to take a step in the right direction and reduce the loss. The magnitude with which the parameters are updated is controlled by the *learning rate*. This important hyperparameter affects how quickly, if at all, the NN converges to a set of parameters that minimises the loss. Optionally, the learning rate can be altered during training by a learning rate scheduler, benefitting convergence.

Different types of optimisers exist such as stochastic gradient descent (SGD) or Adam [9]. The Adam optimiser is generally more robust than SGD against suboptimal choices for the learning rate [30]. A downside of Adam is that it maintains state, occupying at least $2\times$ the model size [28], which can be problematic for large models. Nevertheless, Adam is the most used optimiser in INR literature, where model size is usually not an issue.

2.3.4 Epoch

To summarise, as introduced at the beginning of this chapter, the following steps constitute a singular improvement to the neural network parameters.

1. Performing a **forward pass** to compute output $\hat{\boldsymbol{y}}$
2. Computing the **loss value** $\mathcal{L}(\hat{\boldsymbol{y}}, \boldsymbol{y})$
3. Computing the influence of all weights on l using **back-propagation**
4. Updating all parameters following the **optimiser**'s rules

This sequence represents a single improvement to the NN, referred to as a training step.

In practice, the above steps are rarely performed on a single input-output pair $(\boldsymbol{x}, \boldsymbol{y})$. Instead, training data is divided into small subsets called mini-batches. A mini-batch is a small group of samples from the dataset that are processed together during a single training step. Using mini-batches requires sufficient memory to propagate multiple samples through the NN, and retain their activations for the optimiser step. However, the loss and gradient are averaged over the batch dimension. Hence, by considering mini-batches, the gradient will

be more smooth compared to when each sample gets its own gradient. A large batch size reflects the gradient over the whole dataset more accurately and make the training loss converge in fewer epochs. However, when combatting overfitting (*Overfitting and generalisation*, page 6) a small batch size can be used on purpose, to make the model generalise better to unseen data.

2.4 Training Considerations

2.4.1 Train, validation and test data

Consider a classification problem where we measure the NN's performance by its classification accuracy. For an appropriately sized NN and dataset, one may find that the NN will fairly quickly classify the entire dataset without errors. It seems like the NN is performing outstandingly but we are likely observing overfitting. One has to consider that an NN is very flexible and may just recall some unique spurious feature for each entry in the dataset which tells the NN its class. Instead, to fairly assess how well the NN works, we must assess the NN on data which it has not yet seen before. We typically set aside some training data as a validation set. After each epoch on the training data, the NN is evaluated on the validation dataset. When overfitting, we will observe that the train loss decreases whereas the validation loss increases which is similarly reflected in the train and validation accuracy. This signifies that the NN is simply recalling spurious features about the train data such that the train loss minimises but which bear no relevance to unseen data. In fairness, this is a reasonable consequence of the objective function which requires the NN to minimise the training loss, however, the resulting NN is not useful to us as it would be useless on new, unseen data. Instead, we tune our model to perform well on the validation set. An issue is that, whilst we do not openly use the validation set for training, we *do* use it to steer our design and tuning decisions. In a way, the validation set *is* used in training the model as it influences it indirectly. For this reason, the test set exists. Like the validation set, the test set is not used in training, but even more strictly, the test set must not inform any tuning decisions. It provides an unbiased estimate of the model's performance on unseen data.

2.4.2 Overfitting and generalisation

Overfitting is when the NN learns shortcuts rather than meaningful features to obtain a low loss. Subsequently, the NN will perform very poorly on unseen data. This often stems from the model exploiting correlations or biases in the training set rather than the true underlying structure of the data. For example, in the Waterbirds dataset, models can mistakenly rely on the background, *e.g*. water or land, rather than the actual bird's features to classify images [20]. Instead, we want our model to generalise well to unseen data.

Regularisation To combat overfitting, so-called regularisation techniques are employed. These techniques aim to make it somewhat more difficult for the model to overfit on the train set.

One class of regularisation techniques actively hinder the ability of the model to memorise the training set. For example, 'weight decay' discourages large weights which lead to smoother and more generalisable solutions. 'Dropout' is another such technique which temporarily disables a random subset of nodes during each training step. Doing this, the model cannot rely on specific pathways *i.e*. features, encouraging it to distribute its capacity over shared, meaningful features.

The second class of regularisation techniques is called 'data augmentation'. Data augmentation generates variations of the training samples. By *e.g*. flipping every image in our train dataset, we can double the dataset size. Flipping an image generally keeps the label intact, so our augmented data remains valid. By exposing the model to more varied data, it learns more robust features. Additionally, techniques such as CutMix [27] work on a higher level; *e.g*. by splicing parts of different image samples together.

Both classes of regularisation techniques are often employed in tandem.

2.5 Convolutional Neural Networks

Convolutional neural networks (CNNs) are a well-researched neural network architecture that is typically used in image processing. CNNs have appealing parameter sharing and equivariance properties, which will be expanded upon in this section.

Similar to how an MLP is built from linear layers, a CNN consists of *convolutional* layers. Whereas a linear layer models interaction between all input values, a convolutional layer takes a localised approach by means of a 'kernel'. Taking the example of image processing, a kernel is a 2D matrix of learnable parameters which is laid on top of the image, such that each value of the kernel overlaps with a pixel of the image. All overlapping values are multiplied and averaged together, yielding a single scalar value that is placed in the

layer output to a corresponding pixel position. The kernel is then shifted with a stride of one pixel, where the computation is repeated. By repeatedly moving the kernel and repeating the operation over the entire image, we produce an output named the 'feature map'. A single kernel might learn a single particular feature. To enhance the flexibility of a CNN, we can introduce several kernels for each convolutional layer. That is, k kernels, produce k feature maps. Note that only the pixels which the kernel overlaps can interact with each other, *i.e.*, convolution is a strictly local operation. This is beneficial in image analysis as pixels that are next to each other generally carry more relevance than pixels on either side of an image. Still, it would be useful to consider image features at a larger scale. To achieve this, we downsample the feature maps. This process is referred to as pooling, and takes *e.g.* the maximum value of each group of 4 pixels in the feature map, thereby aggregating 4 features into 1. When this aggregated feature is considered in a subsequent layer, that layer is implicitly operating on the initial group of 4 features. By repeatedly pooling feature maps, we grow the so-called receptive field of the convolutional layers, allowing us to model long-range dependencies in the image content.

The number of parameters of a CNN is not inherently dependent on the image resolution, because the size of convolutional kernels does not scale with the input resolution. However, high-resolution images still pose challenges. For instance, a depicted object is spread out over more pixels, so relevant image features are spread out accordingly. Additionally, on a larger image, a kernel must 'travel' further to cover the image, increasing latency. While the kernel size remains fixed, the feature map dimensions scale with image size, and the memory requirements grow accordingly. During training, every intermediate feature map must be stored for back-propagation, leading to significantly higher memory consumption as input size increases. Consequently, the computational cost, memory usage, and—indirectly—the number of parameters needed to process larger receptive fields all escalate when working with high-resolution images.

2.5.1 Equivariance, invariance, robustness

Consider an image depicting a cat sitting in the bottom left corner and an NN that successfully labels this image as 'cat'. If we were to shift all pixels such that the cat is not positioned in the bottom left corner, we would expect the NN to still classify the image as 'cat'. The fact that the cat is now positioned elsewhere does not influence the label we give the image. This refers to translation invariance, whereby a transformation applied to the input does not affect the function output. More specifically,

$$\mathcal{T}(f(x)) = f(x) \tag{7}$$

where x is the input image, f a function and $\mathcal{T}$ a particular transformation.

Translation equivariance refers to the property that a transformation to the input is reflected in a transformation to the output. That is,

$$f(\mathcal{T}(x)) = \mathcal{T}(f(x)) \tag{8}$$

which we can observe in convolutional layers. If a particular image feature is shifted, the corresponding convolutional layer's feature map will shift accordingly. That is because the convolutional layer's kernel will be slid over the image feature regardless of its absolute position in the image, whereby the activations in the feature map are shifted accordingly.

Robustness is a weaker property. A robust NN can handle certain perturbations in the input without significant performance degradation. When a non-translation invariant NN is trained on many images whereby the relevant features are shifted randomly, the NN will learn to be robust against it. However, it would be more efficient if we did not have to teach this property to the NN. Unlike invariance and equivariance, robustness does not imply that the NN will completely ignore the transformations it was exposed to during training.

2.6 Transformers

Like CNNs and MLPs, transformers are a particular neural architecture. Whereas an MLP stands out for its simple linear layer, and CNNs for their convolutional layers, a transformer is set apart by their so-called 'attention' mechanism. Transforms are widely used in natural language processing [2] and have also been successfully employed in image processing [5, 13].

One of the major advantages of transformers is their ability to handle input sequences of varying lengths. Unlike MLPs, which require a fixed input size, or CNNs, constrained by their receptive fields, transformers can process sequences of arbitrary lengths, such as paragraphs of text or long video frames. This adaptability makes them very versatile.

2.6.1 Attention

Attention is the mechanism that powers transformers, enabling them to determine the relevance of parts of a sequence from their context. The attention mechanism relies on three components: queries, keys, and values. A query vector represents the item of interest, while key and value vectors represent the other elements in the sequence. Attention computes the relationships between these components, weighting each key-value pair's relevance to the query. Mathematically, this is expressed as:

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^\top}{\sqrt{d_k}})V$$

where Q (queries), K (keys), and V (values) are produced by passing each sequence element through a corresponding query, key and value linear layer. QK computes the similarity between the query and key value. $\sqrt{d_k}$, is a scaling factor that stabilises training, where d_k is the dimension of the keys. The softmax operation scales all values such that they sum to 1. The attention weights are then applied to the value vectors V. In short, attention uniquely models how values in the input relate to each other given their context.

Local attention. Traditional attention mechanisms operate globally, modelling interactions between every element in the input data. Local attention constrains the attention mechanism to focus only on a set neighbourhood. This restriction significantly reduces computational complexity, and like the locality bias in CNNs, can actually be harmonious with the data modality in question. Nevertheless, considering global context remains important. Similar to CNNs, we can build up a receptive field whereby we pool features to aggregate broader information. Subsequent local attention layers operate on the pooled features and thus model long-range interactions.

2.7 Implicit Neural Representations

An Implicit Neural Representation (INR) is a neural network that is trained to represent a signal by mapping signal coordinates to signal values. For example, an INR may be trained to map pixel coordinates to their respective RGB values. During training, the INR is supervised by comparing the predicted colour to the original signal colour at that pixel. Once trained, the entire signal can be reconstructed by querying the INR at all signal coordinates, producing a reconstruction.

Put formally, an original signal can be represented as a mapping between discrete coordinates and discrete signal value $s : \mathbb{N}^{d_{\text{in}}} \mapsto \mathbb{N}^{d_{\text{out}}}$ and is stored in memory as a set of linked input and output values $\{(\boldsymbol{x}_i \in \mathbb{N}^{d_{\text{in}}} \mid s(\boldsymbol{x}_i) \in \mathbb{N}^{d_{\text{out}}})\}_{i=1}^{n}$ where $d_{\text{in}} = 2$, $d_{\text{out}} = 3$ for RGB images. An INR $f_\theta : \mathbb{R}^{d_{\text{in}}} \mapsto \mathbb{R}^{d_{\text{out}}}$ learns a continuous function to replicate $s(\boldsymbol{x})$ to the best of its ability.

Though the INR is trained on the discrete intervals where the signal is defined, the INR is inherently a continuous function. This continuity enables us to query the INR at unseen coordinates, allowing for tasks like image upscaling. In such tasks, the INR generalises to higher-resolution coordinates which exist in-between the coordinates used to supervise the INR. The INR effectively fills in finer details that were not present in the original signal [3]. The generalisation abilities of INRs are similarly leveraged in novel view synthesis, whereby an INR captures the structure of a 3D scene by training on images from different viewpoints. Once trained, the INR is prompted to generate views from new angles, producing photorealistic renders [15]. The examples of super-resolution and view synthesis show that INRs possess powerful compression and generalisation abilities.

In this thesis, we train INRs to capture images. The quality of the INR reconstructions is quantified as the peak signal-to-noise ratio (PSNR). When an INR produces reconstructions of a high enough PSNR, the signal is seemingly captured inside the weights of the INR. We can then treat the INR as an alternative representation of the original signal.

2.7.1 Common INR architectures

The principle of 'let an NN fit to a signal' does not prescribe a particular architecture. Early research therefore used a simple MLP [15, 18] but found that this architecture produced blurry reconstructions. The tendency of MLPs to learn low-frequency content is referred to as 'spectral bias' [24, 15] and had been identified before its prominence in INR literature [19]. A way of solving this issue is by introducing harmonics; transforming original values by sine functions of several frequencies. They can either be applied directly to the input coordinates [24, 15] or be incorporated into the activation function as done in SIREN [23].

SIRENs [23] are a foundational INR architecture which are commonly used in INR classification [16, 29, 10, 8]. A SIREN is an MLP where sine functions are used as activation functions as opposed to the typical ReLU activation function. If properly initialised, a SIREN mitigates spectral bias issues successfully.

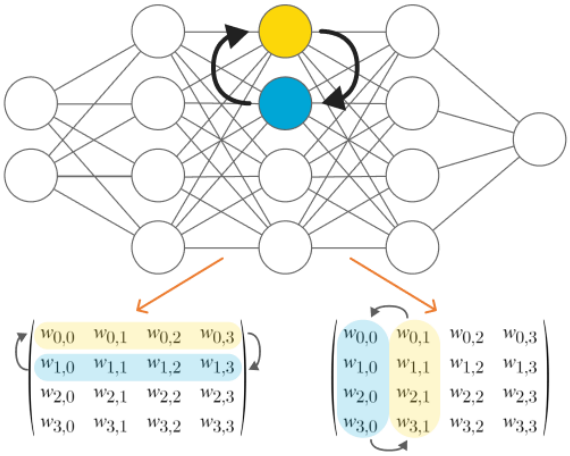


Figure 4: Permutation symmetries. In a layer, a node can be swapped. This does not change the NN's output whatsoever but the underlying weight matrices undergo a permutation.

2.7.2 INRs for downstream use

If an INR is able to reconstruct a signal from its coordinates, it should be possible to elicit the original signal's characteristics by analysing the INR's weight-space. If this can be done successfully, INRs can be used as an alternative representation to the original signals, mitigating challenges associated with traditional grid-based data such as images or voxel grids. Whereas we would usually classify image datasets, we now consider INR datasets, whereby each INR represents a single image. It is natural to assume that an INR that captures an image with high fidelity would be the best candidate for INR classification. However, there is no straightforward link between reconstruction quality and the INR's interpretability as both very low and very high PSNR values result in a similarly poor classification accuracy when used in a classification task [17].

In INR classification literature, many works focus on analysing SIRENs. As SIREN is a derivative MLP, we analyse the INR weight matrices and bias vectors. It is not obvious how such a weight-space can be processed. Early approaches simply extract statistics about the weight matrices [6], or flatten the weights into vector [4, 25], which are then processed by an NN. As NNs are flexible and the training process is stochastic, it is possible that sets of different weights yield a similar reconstruction. It is up to the INR classifier to discern between relevant and irrelevant weight-space features. A set of irrelevant weight-space differences is captured in weight-space symmetries; transformations to the weight-space which affect the weight representation but leave the INR function intact. INR classifiers which incorporate equivariance against such symmetries can automatically rule out a set of irrelevant weight-space features. Increasing the amount of such equivariances, increases the INR classification accuracy [16, 29, 10, 8]. One such symmetry is NN permutation, whereby two nodes in a layer swap position (Figure 4). This changes the underlying weight matrices but does not affect the NN function.

Beyond symmetries, similar signals may still be encoded by widely different weight-spaces. This is evident in the overfitting issues which INR classifiers suffer from. A simple solution is fitting redundant INRs for each image. This shows the classifier in what ways a weight-space may differ whilst representing the same signal and leads to an increase in test accuracy [16, 1, 8]. Fitting redundant INRs from scratch can be a resource-intensive process. Instead, augmenting INRs in the data-loading pipeline is preferred. In SIRENs, such augmentation methods include perturbing weights with noise, transforming the first layer to resemble image transformations, masking weights, and mixing INR instances [22, 16]. When applied to an INR, it unclear how exactly the INR function *i.e.* the reconstruction is affected. Put simply, there is little intuition involved in *e.g.* adding noise to an INR's weights if we cannot understand how exactly these weights contribute to the reconstruction.

2.8 Data

2.8.1 Point cloud data

Point clouds typically consist of a collection of points in three-dimensional space. Next to coordinates, each point may contain additional features such as colour or surface normals. Point clouds are widely used to represent geometry such as in LiDAR systems or depth cameras. Unlike images, points in a point cloud are

not distributed equidistantly. This has given rise to a unique set of point cloud classification models that accommodate this property. Typical point cloud problems include assigning labels to individual points, *e.g*. semantic segmentation, or whole point clouds, *i.e*. point cloud classification, as done in this thesis.

2.8.2 Image data

Images are a type of grid-based data; a data modality made up of equally sized elements. Whether or not an image actually depicts a highly-detailed object or a blank canvas, in memory they are the same size. This presents challenges to neural networks which process images. Especially in specific cases where small details carry significant meaning, as is the case in the medical domain.

2.8.3 Datasets

MNIST. MNIST (Modified National Institute of Standards and Technology database) is a dataset of images depicting handwritten digits and corresponding labels ranging from 0 to 9. All images are greyscale and standardised to 28×28 pixels. 60k training images and 10k testing images are provided. A sample of the data is shown in Figure 5. MNIST has historically been used in pivotal machine learning research [12]. Nowadays, it is considered a low-complexity dataset without much challenge in the way of inter- and intraclass variability or background noise. It is easy to obtain high test accuracies using modern deep learning techniques [7, 21].

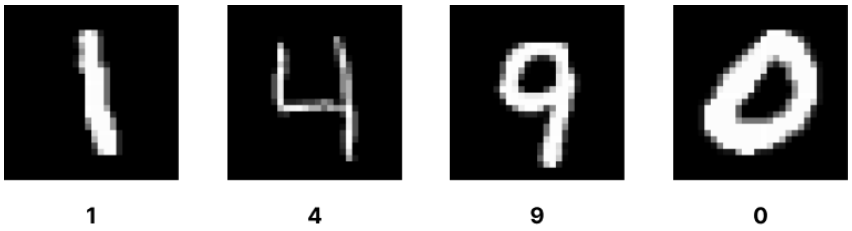


Figure 5: Samples from the MNIST dataset with their respective labels.

Fashion-MNIST Fashion-MNIST (FMNIST) is a dataset that was inspired by MNIST but aims to pose a more challenging task for classification algorithms [26]. Similar to MNIST, FMNIST contains greyscale images of 28×28 pixels. 60k train images and 10k test images are provided. Samples from FMNIST are shown in Figure 6.

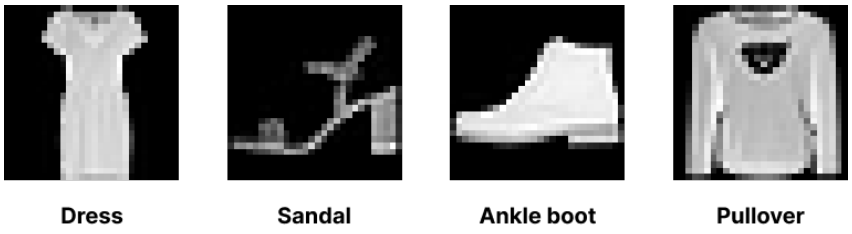


Figure 6: Samples from the Fashion-MNIST dataset with their respective labels.

Imagenette Imagenette is a subset of Imagenet. The original Imagenet contains more than a million images depicting 1,000 different classes [11]. Imagenette contains just 10 classes, of around 9.5k training images and a predefined validation set of around 4k images. Notably, Imagenette images do not have a standard size. As shown in Figure 7, image resolutions vary wildly. As this may pose issues in standard image classification pipelines, Imagenette variants exist which have resized the image such that the shortest side is *e.g*. 320 pixels long.

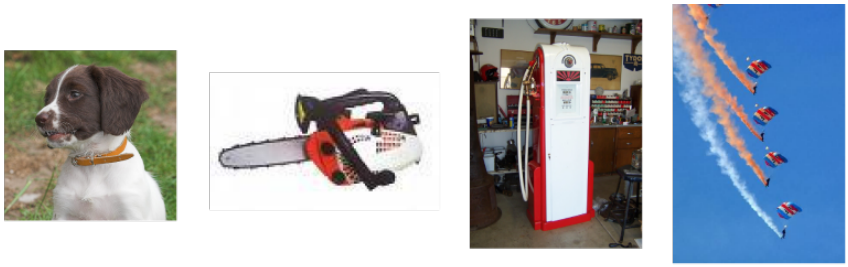


Figure 7: Samples from the Imagenette dataset with their respective labels and resolution Height $\times$ Width

2.8.4 Toy datasets

In academia and education, a toy problem is a purposefully simple problem that serves as a clear and unambiguous context in which a problem-solving technique can be demonstrated. The aim is to highlight particular capabilities and properties of the technique rather than its aptitude at solving real-world problems.

In deep learning, the properties of a model can be aptly demonstrated using a toy dataset. For instance, we may shift the hue of a dataset to strictly blue, red and green images to demonstrate a potential bias towards any colour by the model.

References

[1] M. Bauer, E. Dupont, A. Brock, D. Rosenbaum, J. R. Schwarz, and H. Kim. Spatial functa: Scaling functa to imagenet classification and generation. *arXiv preprint arXiv:2302.03130*, 2023.

[2] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language models are few-shot learners, 2020.

[3] Y. Chen, S. Liu, and X. Wang. Learning continuous image representation with local implicit image function. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8628–8638, 2021.

[4] L. De Luigi, A. Cardace, R. Spezialetti, P. Zama Ramirez, S. Salti, and L. Di Stefano. Deep learning on implicit neural representations of shapes. In *International Conference on Learning Representations (ICLR)*, 2023.

[5] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2021.

[6] G. Eilertsen, D. Jönsson, T. Ropinski, J. Unger, and A. Ynnerman. Classifying the classifier: dissecting the weight space of neural networks. *ArXiv*, abs/2002.05688, 2020.

[7] S. Greydanus and D. Kobak. Scaling down deep learning with mnist-1d, 2024.

[8] I. Kalogeropoulos, G. Bouritsas, and Y. Panagakis. Scale equivariant graph metanetworks. *arXiv preprint arXiv:2406.10685*, 2024.

[9] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization, 2017.

[10] M. Kofinas, B. Knyazev, Y. Zhang, Y. Chen, G. J. Burghouts, E. Gavves, C. G. Snoek, and D. W. Zhang. Graph neural networks for learning equivariant representations of neural networks. *arXiv preprint arXiv:2403.12143*, 2024.

[11] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[12] Y. LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

[13] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo. Swin transformer: Hierarchical vision transformer using shifted windows, 2021.

[14] L. Mescheder, M. Oechsle, M. Niemeyer, S. Nowozin, and A. Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[15] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng. Nerf: representing scenes as neural radiance fields for view synthesis. *Commun. ACM*, 65(1):99–106, Dec. 2021.

[16] A. Navon, A. Shamsian, I. Achituve, E. Fetaya, G. Chechik, and H. Maron. Equivariant architectures for learning in deep weight spaces. In A. Krause, E. Brunskill, K. Cho, B. Engelhardt, S. Sabato, and J. Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 25790–25816. PMLR, 23–29 Jul 2023.

[17] S. Papa, R. Valperga, D. Knigge, M. Kofinas, P. Lippe, J.-J. Sonke, and E. Gavves. How to train neural field representations: A comprehensive study and benchmark. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 22616–22625, 2024.

[18] J. J. Park, P. Florence, J. Straub, R. Newcombe, and S. Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[19] N. Rahaman, A. Baratin, D. Arpit, F. Draxler, M. Lin, F. Hamprecht, Y. Bengio, and A. Courville. On the spectral bias of neural networks. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 5301–5310. PMLR, 09–15 Jun 2019.

[20] S. Sagawa, P. W. Koh, T. B. Hashimoto, and P. Liang. Distributionally robust neural networks for group shifts: On the importance of regularization for worst-case generalization, 2020.

[21] Z. SE. (fashion-)mnist leaderboard. `http://fashion-mnist.s3-website.eu-central-1.amazonaws.com`, 2017.

[22] A. Shamsian, A. Navon, D. W. Zhang, Y. Zhang, E. Fetaya, G. Chechik, and H. Maron. Improved generalization of weight space networks via augmentations. In *Forty-first International Conference on Machine Learning*, 2024.

[23] V. Sitzmann, J. N. Martel, A. W. Bergman, D. B. Lindell, and G. Wetzstein. Implicit neural representations with periodic activation functions. In *arXiv*, 2020.

[24] M. Tancik, P. P. Srinivasan, B. Mildenhall, S. Fridovich-Keil, N. Raghavan, U. Singhal, R. Ramamoorthi, J. T. Barron, and R. Ng. Fourier features let networks learn high frequency functions in low dimensional domains. *NeurIPS*, 2020.

[25] T. Unterthiner, D. Keysers, S. Gelly, O. Bousquet, and I. Tolstikhin. Predicting neural network accuracy from weights. *arXiv preprint arXiv:2002.11448*, 2020.

[26] H. Xiao, K. Rasul, and R. Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

[27] S. Yun, D. Han, S. J. Oh, S. Chun, J. Choe, and Y. Yoo. Cutmix: Regularization strategy to train strong classifiers with localizable features. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 6023–6032, 2019.

[28] Y. Zhang, C. Chen, Z. Li, T. Ding, C. Wu, D. P. Kingma, Y. Ye, Z.-Q. Luo, and R. Sun. Adam-mini: Use fewer learning rates to gain more, 2024.

[29] A. Zhou, K. Yang, K. Burns, A. Cardace, Y. Jiang, S. Sokota, J. Z. Kolter, and C. Finn. Permutation equivariant neural functionals. *Advances in neural information processing systems*, 36, 2024.

[30] P. Zhou, J. Feng, C. Ma, C. Xiong, S. C. H. Hoi, et al. Towards theoretically understanding why sgd generalizes better than adam in deep learning. *Advances in Neural Information Processing Systems*, 33:21285–21296, 2020.

3 Scientific article

The scientific article starts on the next page.

ARC: Anchored Representation Clouds for High-Resolution INR Classification

Joost Luijmes Alexander Gielisse Roman Knyazhitskiy Jan van Gemert
Delft University of Technology, The Netherlands

Abstract

Implicit neural representations (INRs) exhibit exceptional compression and generalisation abilities that have enabled striking progress across a variety of applications. These properties have fuelled a growing interest in leveraging INRs for traditional classification tasks as a memory-efficient alternative representation of images, breaking the persistent link between image resolution and associated resource costs. Current INR classification methods face limitations such as a restriction to low-resolution data and sensitivity to image-space transformations. We attribute these issues to the employed INR architecture which lacks mechanisms for local representation, thereby disregarding spatial structure within the data and furthermore limiting their ability to capture high-frequency details. In this work, we propose ARC: Anchored Representation Clouds, a novel INR architecture that explicitly anchors latent vectors in image-space. By introducing spatial structure to the latent vectors, ARC can capture local image data which in our testing leads to state-of-the-art implicit image classification of both low- and high-resolution images and increased robustness against image-space translation.

1. Introduction

From novel view synthesis to inverse problems, implicit neural representations (INRs) have enabled leaps in accuracy across a variety of problems and domains due to their unique data compression and generalisation capabilities [7, 9, 13, 18, 25, 36, 49, 61]. As such, interest has grown in whether INRs can similarly enrich conventional computer vision tasks like image classification; the focus of this research.

An INR is a neural network (NN) that learns a mapping from coordinates in the signal's domain to the signal's values, *e.g.* from 2D pixel coordinates to RGB colours. After training, the INR can reconstruct an approximation of the original signal when queried on all signal coordinates, implying that the signal is encoded inside the INR weights. INRs are able to compress signals in a variety of domains [10, 14, 21, 45, 48], and exhibit excellent generalisation capabilities outside of the signal's domain [3, 36, 60]. Alternatively, INRs can be interpreted as an alternative view of the original signal. An INR data modality can potentially mitigate persistent issues associated with the original signal modality.

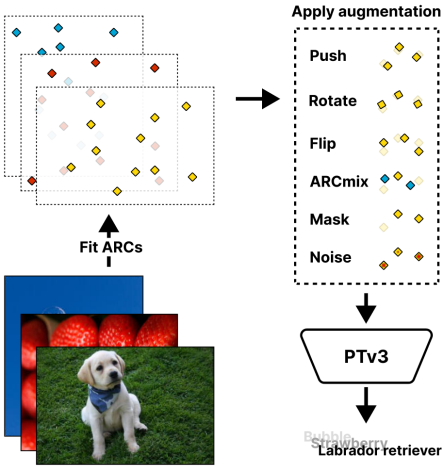


Figure 1. ARC anchors latent vectors directly in image coordinate space, preserving the spatial image structure within the INR weight-space. Once trained, ARC can be processed by a point cloud classifier, effectively recasting the task of INR classification into the domain of point cloud processing.

In image classification, one such issue has to do with the nature of the grid-based image modality. In grid-based data, elements of equal size are positioned equidistantly over the signal domain, *e.g.* RGB pixels in an image or occupancy bits in a voxel grid. This uniform data representation results in memory costs that scale exponentially with signal dimensionality and resolution. To make training on image datasets feasible, images are typically downsampled, which may eliminate the relevant features [19, 27] or lead to decreased performance [23, 51]. While these issues can be mitigated [19, 27], an alternative data representation that more efficiently encodes high-resolution data may offer a more effective basis for resource-efficient learning, which suggests INRs as a promising solution.

Current INR classification methods face several shortcomings [24, 26, 68]. First, these methods are only demon-

strated on low-resolution image datasets such as MNIST [29], Fashion-MNIST [58], and CIFAR10 [28]. Second, the employed INR architecture learns entirely different representations under image-space transformations such as translation. Equivariance to the position of image features is a fundamental property in traditional image classifiers [6, 63], especially because higher-resolution image datasets allow for less restricted object positions and present challenges in terms of background noise [50]. Third, INR classifiers typically suffer from overfitting [24, 26, 46], with restricted data augmentation methods to combat this [38, 46], or through the resource-intensive process of fitting several redundant INRs per image [1, 38, 46, 68].

In response to these issues, we introduce a novel type of INR named ARC: Anchored Representation Clouds, along with a flexible classification pipeline (Figure 1). ARC consists of 1) an image-specific encoder which anchors a cloud of latent vectors in the image coordinate space, and 2) an MLP decoder that is shared among ARC instances. The encoder decouples the image domain from the MLP, allowing local latent representations. This way, ARC retains local image features, making it more robust against image translation and more capable of high-resolution implicit image classification. Furthermore, as the latents can be positioned freely, they can be anchored more densely in high-frequency image regions, biasing model capacity towards complex regions rather than having to encode the image globally. By increasing the number of anchored latents, ARC can trivially scale to larger image complexity. By converting images to a set of ARCs, each image is effectively represented by a cloud of latent vectors. This intuitively gives rise to the application of point cloud architectures for downstream use, along with intuitive and effective data augmentation methods which eliminate the need for redundant INR fitting. To our knowledge, this is the first work to utilise an INR's entire weight-space on a high-resolution dataset like Imagenette [20], achieving a classification accuracy of 75.92%. Through controlled experiments, we further demonstrate ARC's robustness to image-space transformations and its ability to capture high-resolution images.

Our contributions include:

- A novel INR architecture that anchors latent vectors in the image coordinate space, preserving spatial locality.
- An accompanying classification pipeline that enables effective and intuitive weight-space augmentation methods.
- Enhanced robustness to image-space translations in INR classification.

2. Related Work

Implicit neural representations. An implicit neural representation (INR) [47] is a neural network trained to represent a signal. Given a pixel coordinate as input, the INR aims to produce the corresponding signal value as accurately as possible. By learning this mapping for all coordinates where the signal is defined, the INR learns to capture the signal inside its weights. Alternative names for implicit neural representations are neural fields [39, 56, 59], coordinate networks [31, 34, 67] and coordinate-based neural representation [53].

The premise of INRs does not prescribe a particular architecture, prompting early work to assess the suitability of the simple MLP [35, 36, 40]. Such models struggle to capture the high-frequency components of the signal, a phenomenon referred to as *spectral bias* which in this application results in blurry image reconstructions [42, 47, 52]. This issue inspired a large set of diverse solutions, such as transforming the input using sinusoids [36, 52, 67], modifying the activation function [5, 15, 33, 43, 44, 47] or by positioning learnable elements in the signal coordinate space to represent local image regions [2, 4, 16, 22, 30, 32, 34, 37, 41].

We posit that positioning learnable elements in signal coordinate space can aid INR classification, as by coupling latent information to local image regions, we retain image structure in the latent space and the image features encoded therein. Other methods which position latents freely in image coordinate space, focus on improving the signal reconstruction quality [4, 16]. These methods hybridise INR methods [4], and require intricate initialisation and latent decoding schemes [4, 16]. In contrast, simplicity is a core design principle in ARC; reducing computational complexity ensures that fitting an entire image dataset remains efficient.

INR classification. INR literature typically emphasises training and parameter efficiency [4, 10, 13, 21, 37], or utilises INRs as a component in a broader method for their compression and generalisation abilities [7, 9, 9]. In our approach, INRs are treated as singular units of data, where they function as an alternative representation to images in a classification task.

Interpreting an INR as a singular unit of data centres on analysing its weights learned during the signal fitting process to intuit the represented signal. Early works on weight-space analysis studied characteristics of classification networks from their flattened weight matrices [54] or their weights' statistics [12]. Flattened representations remain in use as low-dimensional embeddings are trained along with [1, 11], or after [8], the INR fitting. These methods do not generalise well to image classification [8, 24] or larger-scale classification tasks [1] however. In the context of INRs however, a key insight to process *whole* weight-spaces was to consider a neural network's symmetries; transformations which alter the weights but preserve the INR's function [17]. Architectures which incorporate equivariances to such symmetries significantly increase INR classification accuracy [24, 26, 38, 69]. With ARC, we propose an architecture that anchors low-dimensional latent embeddings in image-space. This ties the learnt encodings directly to local image content, effectively

compressing an image into a latent point cloud. A similar observation is made and implemented in a concurrent work with an attention-based architecture [56]. Their method is demonstrated across various domains but does not address the shortcomings of INR classification concerning image classification, such as confinement to low-resolution image datasets and sensitivity to image-space translations.

Improving the interpretability of INRs. The flexibility of NNs allows for wildly varying weight-spaces that faithfully capture a signal. This variability makes it difficult for downstream models to capture consistent image features across INR instances [24, 26, 46]. Previous work has shown that establishing a form of alignment or 'common ground' among INRs improves classification accuracy. Such methods include sharing the INR weight initialisation [38, 39], introducing shared learnable elements to the fitting process [4, 16, 56], learning low-dimensional shifts to an established INR base network [1, 11] or sharing a part of the INR over all instances [55]. In this spirit, ARC shares a decoder over the whole dataset, which is jointly pretrained on a subset of the data and then frozen.

Even if INRs share a form of alignment, overfitting remains a persistent issue in INR classification [24, 26, 46]. A limited set of weight-space augmentations are available to combat this [38, 46]. Hence, INR classification methods fit redundant INRs for each image in the dataset, which is a resource-intensive task [1, 24, 38, 66, 68]. Instead, we can leverage the unique weight-space of ARC to apply intuitive data augmentation methods on-the-fly which we demonstrate to be competitive in regularisation effectiveness to redundant INR fitting, at a fraction of the computational cost.

3. Method

3.1. Preliminaries

The analysis presented in this chapter benefits from a more rigorous formulation of the problem setting. We interpret a sampled signal s as a set of equidistant discrete observations $\{(\boldsymbol{x}_i \in \mathbb{N}^{d_{\text{in}}}, s(\boldsymbol{x}_i) \in \mathbb{N}^{d_{\text{out}}})\}_{i=1}^{n}$. For instance, an RGB image would be a set of pixel locations ($d_{\text{in}} = 2$) with corresponding RGB colours ($d_{\text{out}} = 3$). An INR learns parameters θ by supervising the mapping between the signal's domain and codomain $f_\theta : \mathbb{R}^{d_{\text{in}}} \mapsto \mathbb{R}^{d_{\text{out}}}$, supervising on $s(\boldsymbol{x})$. The mean squared error (MSE) loss is used to supervise the INR.

As the main focus of this paper is on image classification, we will forego the more general 'signal' terminology and instead refer to images, pixels, colours, and so on throughout this chapter.

3.2. ARC

ARC consists of an encoder and a decoder. The encoder is composed of a cloud of latent vectors and retrieval logic to obtain the n nearest latent vectors for a given input coordinate. These vectors are then concatenated and passed to the decoder, which maps it to the signal's codomain. A visualisation of this process is shown in Figure 2 and is elaborated upon in the remainder of this section.

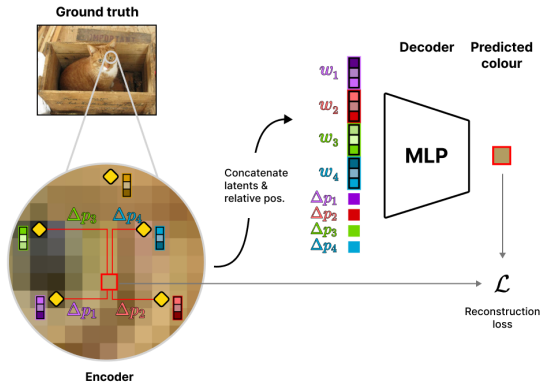


Figure 2. When queried on an image coordinate $\boldsymbol{x}$, ARC finds the 4 nearest latent vectors. These vectors, along with their relative position to $\boldsymbol{x}$, are concatenated into a long descriptive vector. The decoder maps this vector to the corresponding RGB colour.

Let us analyse in more detail how ARC maps a coordinate to a colour. Given a coordinate $\boldsymbol{x} \in \mathbb{R}^{d_{\text{in}}}$, U_n retrieves the n nearest anchored latents and their relative positions to $\boldsymbol{x}$ from $\mathcal{P} = \{(\boldsymbol{p}_i, \boldsymbol{w}_i)\}_{i=1}^{k}$. These latents and their relative positions are concatenated. The concatenated vector is then passed through the MLP decoder g, mapping it to an RGB colour. ARC can be specified as follows.

$$f_\theta(\boldsymbol{x}) = g_\psi(e(\boldsymbol{x})) \qquad \text{ARC} \quad (1)$$

$$\mathbb{R}^{d_{\text{in}}} \mapsto \mathbb{R}^{d_{\text{out}}} \quad (2)$$

$$e(\boldsymbol{x}) = \text{Concat}(U_n(\boldsymbol{x})) \qquad \text{Encoder} \quad (3)$$

$$(\mathbb{R}^z \times \mathbb{R}^{d_{\text{in}}})^n \mapsto \mathbb{R}^{n\cdot(z+d_{\text{in}})} \quad (4)$$

$$g_\psi(\boldsymbol{v}_{\boldsymbol{x}}) = \text{MLP}_\psi(\boldsymbol{v}_{\boldsymbol{x}}) \qquad \text{Decoder} \quad (5)$$

$$\mathbb{R}^{n\cdot(z+d_{\text{in}})} \mapsto \mathbb{R}^{d_{\text{out}}} \quad (6)$$

$$U_n(\boldsymbol{x}) = \{(\Delta\boldsymbol{p}_i, \boldsymbol{w}_i)\}_{i=1}^{n} \qquad \text{Indexing function} \quad (7)$$

$$\mathbb{R}^{d_{\text{in}}} \mapsto (\mathbb{R}^{d_{\text{in}}} \times \mathbb{R}^z)^n \quad (8)$$

$$\mathcal{P} = \{(\boldsymbol{p}_i, \boldsymbol{w}_i)\}_{i=1}^{k} \qquad \text{Latent cloud} \quad (9)$$

$$\text{where } \boldsymbol{p}_i \in \mathbb{R}^{d_{\text{in}}},\ \boldsymbol{w}_i \in \mathbb{R}^z \quad (10)$$

$$\theta = \{\psi, \{\boldsymbol{w}_i\}_{i=1}^{k}\} \qquad \text{Learnable parameters} \quad (11)$$

3.2.1 Encoder

The encoder consists of a cloud of learnable latent vectors that are anchored in the image coordinate space (Eq. (9)), and aggregation logic (Eq. (7)). The latent dimension z and the number of latents anchored in the image are hyperparameters which can be tuned to trade off memory footprint versus reconstruction quality.

Latent vector positions. In determining the position of the latent vectors, we follow [4, 30], whereby learnable elements are positioned in signal-space near high-frequency content so as to bias the model capacity to more difficult to encode content. To this end, the latents' positions are determined by sampling the image gradient norm. The latents' positions remain fixed. This way, the indexing function U_n can cache the index of nearest latents upon ARC initialisation, significantly decreasing training latency.

Indexing and aggregation. Given an input coordinate $\boldsymbol{x}$, U_n retrieves the n nearest latent vectors along with their relative position to $\boldsymbol{x}$. n can be any value but we found 4 nearest neighbours to be sufficiently expressive. In contrast to other methods which predefine an interpolation function to aggregate the latent vectors [4, 16], ARC defers to the decoder to learn this from the latent vectors and relative positions, similar to [3]. The latent vectors and their relative positions are thus simply concatenated and fed to the decoder. Consequently, we do not require Fourier features [4, 52], which, in line with our design objectives, keeps the model simple.

3.2.2 Decoder

The decoder is a simple MLP with ReLU activations, as opposed to more elaborate activation functions [4]. To align latent vectors across different ARC instances, we let instances share a single decoder. This decoder is pre-trained on a subset of the data and then frozen for the remaining ARCs, requiring all latent clouds to capture image features in an aligned manner. Consequently, the memory cost of the decoder can be amortised across the whole dataset as only the anchored latent cloud is required for classification.

3.3. Downstream processing

Due to its unique weight-space, ARC transforms the problem of INR classification into point cloud classification. This allows us to leverage well-studied downstream architectures for ARC classification. No mechanisms against weight-space equivariances are needed, in contrast to SIREN classifiers [24, 26, 38].

Point Transformer v3. Any point cloud architecture that supports arbitrary point feature dimensions can naturally process ARCs. However, since the anchored latent vectors encode local information, an architecture that emphasises local interaction is preferred. To this end, we select Point Transformer v3 (PTv3) [57], a state-of-the-art method that performs local attention. When using PTv3, we provide it only the learnt latent vectors. The latent positions are used only for the relative positional encoding and pooling operations. PTv3 allows for a varying number of points within a batch, enabling ARCs to adjust the number of latents to the image complexity. Note however that in this work we follow [4] by letting the number of latent vectors be proportional to image size. Modifications made to PTv3 for ARC compatibility are discussed in Appendix A.

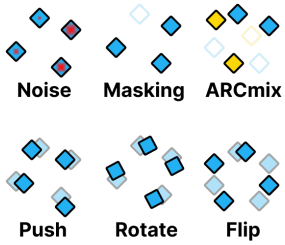


Figure 3. Data augmentation techniques for ARCs. Noise, Masking, and ARCmix manipulate the latent cloud to enhance data diversity, while Push, Rotate, and Flip change just the latent vector coordinates. These augmentations operate directly in the ARC weight-space.

Data augmentation. We can leverage the unique weight space of ARC to apply intuitive data augmentation methods which we show to be almost as effective as using redundant ARC instances in our experiments. These data augmentations are applied 'on-the-fly' on the anchored representation cloud. Figure 3 depicts the augmentation methods: Noise applies random Gaussian noise to the latent content, Masking omits a specified fraction of the points, Push, Rotate and Flip only augment the latent vector coordinates within a single ARC instance. Furthermore, ARCmix, a method inspired by CutMix [62, 64], mixes two ARC by combining their latent clouds.

4. Experiments

In our experiments, we compare INR classification pipelines. In INR classification literature, the seminal SIREN architecture remains the most prominent, and has seen incremental classification improvements over recent years [24, 26, 38, 68, 69]. We focus on two representative baselines: the foundational DWSnets [38] and the state-of-the-art ScaleGMN [24]. Details of the experiments can be found in Appendix D.

4.1. Experiment 1. High-resolution image classification

How well do ARCs and existing INR classification pipelines perform as image resolution increases? To answer this question, we analyse INR classification accuracy on Fashion-MNIST (FMNIST) [58] whereby we scale the image resolution by padding the images with zeroes to a 100×100 and 1024×1024 resolution (Figure 4). The toy datasets are then converted into SIRENs and ARCs and classified by their respective methods. The test accuracy is reported in Table 1.

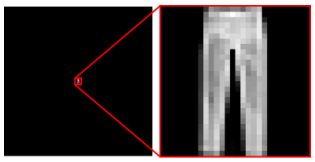


Figure 4. **Exp. 1:** Left, a sample from the 1024×1024 padded FMNIST dataset. Right, a zoomed in view of the depicted object. The high-resolution of the image, transforms the relatively simple task of FMNIST classification into a challenge for baseline INR classification methods.

Side Length	28	100	1024	INR #param increase
DWSNets	67.06$^\diamond$	67.60	53.28	$\times$33
ScaleGMN	**80.78**$^\diamond$	74.50	48.77	$\times$33
Ours	80.42	**79.36**	**73.57**	$\times$**1**

Table 1. **Exp. 1:** Test accuracy (%↑) on the padded FMNIST datasets and the required increase in INR parameters (↓) to produce recognisable reconstructions. The SIREN 1024×1024 dataset is a third of the size of the corresponding ARC dataset due to steep fitting costs as a consequence of the increased number of parameters. Entries marked with $^\diamond$ are taken from their original publication ([24, 38]). Next to being a more parameter efficient representation, our method is more resilient against increasing image size.

While both SIREN and ARC classification pipelines show a degradation in classification accuracy, ARC is demonstrably stronger. This difference can be attributed to how each method handles increasing image resolution. A SIREN's input domain is fixed to $[-1, 1]^{d_{\text{in}}}$, regardless of image size. On higher resolutions, the number of pixels mapped within $[-1, 1]^{d_{\text{in}}}$ grows, requiring SIRENs to produce higher frequency outputs. Subsequently, spectral bias effects re-emerge, resulting in overly smooth reconstructions. To mitigate this, the SIREN architecture is increased in width and depth, yielding a $\times$33 increase in the number of parameters between 100×100 and 1024×1024. This substantial increase significantly increased fitting time, requiring us to limit the SIREN dataset size to approximately a third of that used for ARC. For ARC, the image resolution is decoupled from the latent representation. Consequently, as image size grows but image complexity remains low, the number of latent vectors does not have to be adapted.

We continue our investigation into high-resolution image INR classification with Imagenette [20]. Imagenette is a dataset of natural images with a median resolution of 375×500 and a maximum of 4268×2912. As image complexity increases, INR capacity must increase proportionally. For ARC, this scaling is straightforward as we can simply increase the number of anchored latent vectors. For SIRENs, the scaling is performed in either the number of hidden layers or the hidden dimension. Neither DWSnets nor ScaleGMN support variable SIREN architectures, so a fixed size must be picked. This invariably leads to undercapacity or overcapacity on a subset of the image data. To decrease discrepancies that may arise from this, we use the prescaled dataset variant Imagenette320 [20], and apply a centre-crop to standardise all images to 320×320. These images are converted into SIRENs and ARCs, and subsequently classified. This procedure is additionally performed on the original, full-resolution Imagenette dataset with ARCs. As no test set is provided, we report validation accuracy in Table 2. We were not able to train ScaleGMN on this dataset due to instability issues. We go into more detail about ScaleGMN fitting difficulties in Appendix D.

	Imagenette 320x CenterCrop	Imagenette full resolution
DWSnets	41.05	-
Ours	**71.71**	**75.92**

Table 2. **Exp. 1:** Validation accuracy (%↑) on Imagenette. ARC sets a new watermark in classifying full-resolution image data through their INR representation.

ARC demonstrates high classification accuracy on Imagenette. We hypothesise that this performance is due to the higher resolution of Imagenette images, where relevant features are distributed across larger regions of the image. With latent vectors anchored in these regions, ARC can represent the features with greater precision and redundancy, all while respecting the spatial integrity of these features. This, in turn, enables PTv3 to learn a richer representation and yield a high classification accuracy. The improvement of ARC between the centre-crop and full-resolution sets can be attributed to the resolution bias present in Imagenette which may be exploited by the relative positional encoding in PTv3. We found that a simple ReLU-MLP of dimensions $[2, 64, 64, 64, 10]$ can obtain a validation accuracy of up to 23.46% on Imagenette based on image dimensions alone.

4.2. Experiment 2. Image classification benchmarks

How does ARC compare to established INR classification benchmarks? We follow other INR literature in using MNIST [29], Fashion-MNIST [58], and CIFAR10 [28]. Results are presented in Table 3. On low-complexity grey-scale datasets, ARC performs similarly to current state-of-the-art methods. On the more complex CIFAR10 dataset, ARC obtains state-of-the-art accuracy. CIFAR10 is more challenging than MNIST and FMNIST, as objects are presented in natural images, leading to background noise. We expect that image features are better represented among ARC instances on these more complex images, leading to superior accuracy compared to SIRENs.

	MNIST	FMNIST	CIFAR10
DWSnets [38]	85.71 ± 0.6	67.06 ± 0.3	-
NG-GNN [26]	91.40 ± 0.6	68.00 ± 0.2	36.04$^\diamond$ ± 0.44
NG-T [26]	92.40 ± 0.3	72.70 ± 0.6	-
ScaleGMN [24]	**96.59** ± 0.2	**80.78** ± 0.2	38.82 ± 0.1
Ours	92.69 ± 1.2	**80.42** ± 0.4	**58.47** ± **0.4**

Table 3. **Exp. 2:** Test classification accuracy (%↑) on various image classification datasets. We train our method on 3 different seeds and report the mean and std. Entries marked with $^\diamond$ are taken from reproductions by [24]. ARC classification accuracy is similar to baselines on low-complexity datasets and outperforms them on the more complex CIFAR10 dataset.

#INRs per image	1	20
NG-GNN [26]	36.04$^\diamond$	45.70$^\diamond$
ScaleGMN [24]	**38.82**	**56.95**
Ours	38.12	55.87

Table 4. **Exp. 3:** CIFAR10 test accuracy (%↑) after training on either 1 or 20 INRs per CIFAR10 image. No further data augmentation is employed. Similar to the baselines, ARC classification accuracy improves significantly when trained on redundant INRs. Entries marked with $^\diamond$ are taken from reproductions by [24].

4.3. Experiment 3. Data augmentation

In SIREN classification methods, an increasingly used technique to reduce overfitting is to generate redundant INRs for each image in the dataset [1, 24, 38, 66, 68], but this increases the resource-intensive INR fitting process. Are ARC weight-space data augmentation methods as effective as redundant INR fitting? To establish a baseline, we fit 20 ARCs per image on a 10k subset of CIFAR10. Without any additional augmentations, PTv3 is trained on two conditions: using a single ARC per image and using all 20 ARCs. We report the test accuracy per image Table 4. ARC classification accuracy is on par with the state-of-the-art which has the advantage of being trained on the full CIFAR10 dataset.

We now ask, are data augmentation methods that operate on the ARC weight-space (Figure 3) as effective as using redundant INRs during training? We convert the entire CIFAR10 dataset to ARC instances and evaluate the different data augmentation methods in Table 5. When comparing the test accuracies obtained under weight-space data augmentations to those obtained with redundant INR fitting Table 4, we observe that our data augmentations do not fully close the gap. In fact, as we leverage the entire CIFAR10 in the weight-space augmentation experiments and just a 10k subset in the redundant INR experiment, the gap may be slightly larger. Regardless, our data augmentation methods offer a compelling alternative that requires significantly less computational resources and time to execute.

Latent noise	Point space	ARC masking	ARCmix	Acc.
-	-	-	-	38.12
✓	-	-	-	39.08
-	✓	-	-	51.69
-	-	✓	-	50.25
-	-	-	✓	54.55
-	-	✓	✓	54.56
-	✓	✓	✓	50.34
✓	✓	✓	✓	51.03

Table 5. **Exp. 3:** Test accuracy (%↑) on different ARC augmentation methods. The 'Point space' column applies the push, flip and rotation augmentation methods. Compared to fitting and training on redundant INRs, these more efficient weight-space augmentations yield competitive test accuracy.

4.4. Experiment 4. Feature locality

In this experiment we test our claim that the anchored latent vectors of ARC represent local image features. If so, applying a transformation to the ARC coordinates should yield a similarly transformed reconstruction. Furthermore, in classifying ARCs, the latent vector positions should prove to be relevant. As a reminder, the absolute position of the points is not given as a feature PTv3. Instead, PTv3 uses a *relative* position encoding.

We first consider transformations on the ARC coordinates. Given a trained ARC, we manipulate only the latent vector positions. The index function cache is refreshed and a forward pass is performed. In Figure 5, we perform several such transformations and depict the resulting reconstructions. We can indeed verify that transforming latent vector positions yield correspondingly transformed reconstructions. If the anchored latent represent local image content, it is possible to mix ARCs. We can *e.g.* select parts of different ARCs or simply stack them. To demonstrate this, two ARCs were jointly trained with a shared decoder. As shown in Figure 6, mixing these ARCs produces interesting reconstructions. As the index function cache is refreshed, and image coordinates retrieve their nearest latent vectors, they receive a mix of latent features. Moreover, since the latent vectors are more densely positioned on the image gradient, we can clearly make up edges and other discontinuous jumps in the mixed reconstruction. Mixing ARCs is leveraged in our ARCmix data augmentation method.

We further investigate the significance of local feature representations by considering their influence on ARC classification accuracy. For this experiment, we reuse the FMNIST dataset that was padded to 100×100 in Experiment 1. We push the latent vectors of this dataset into a random direction once, and use this data to 1) evaluate a regularly trained PTv3 instance, and 2) train a PTv3 instance from scratch. In

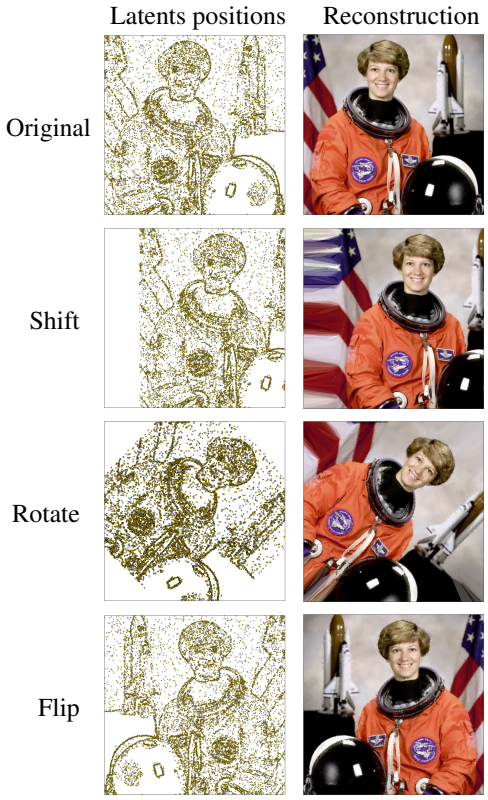


Figure 5. **Exp. 4:** Each row depicts a transformation on the latent vector positions. No other changes are made to the ARC. The resulting reconstruction is displayed in the right column. The correspondence between the latent cloud transformation and the new reconstructions demonstrates how ARC encodes image features locally.

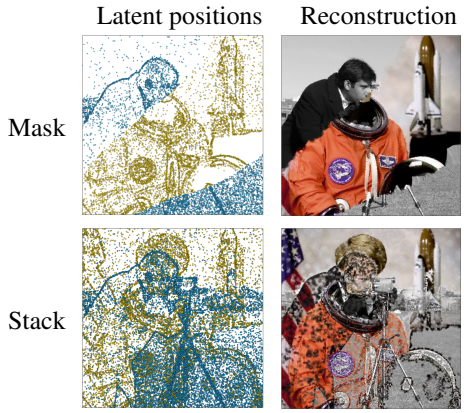


Figure 6. **Exp. 4:** We demonstrate how mixing two ARC retains their local image features. In the top row, latent vectors are masked in a specific shape. In the bottom row, the latent clouds are stacked. The ARCs were jointly trained with a shared decoder.

		Train	
		Intact	Pushed
Test	Intact	79.32	51.79
	Pushed	17.39	69.06

Table 6. **Exp. 4:** FMNIST test accuracy (%↑) if we push each latent vector to a random position during either train or test time. Leaving the latent positions intact yields the highest accuracy. Conversely, training on displaced latent vectors leads to a drop in accuracy when faced with intact ARCs, which we attribute to PTv3 learning incorrect relationships in its relative positional encoding mechanisms.

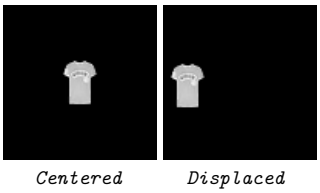


Figure 7. **Exp. 5:** Samples from the *Centered* and *Displaced* datasets.

Table 6 we list the resulting test accuracies. Unsurprisingly, not touching the latent vector positions leads to the highest test accuracy. When the regularly trained PTv3 instance is evaluated on the randomly pushed ARC dataset, a large drop in test accuracy ensues as the learned relative positional encoding within PTv3 becomes meaningless. Conversely, when we train on pushed ARC data, the classifier still works relatively well. PTv3 builds up a global context through layers of local attention and pooling. We hypothesise that the latent features are descriptive enough that, through pooling alone, they are relatively informative. When evaluated on intact ARCs, accuracy drops, which we attribute to PTv3 learning inconsistent or incorrect relationships in its relative positional encoding mechanisms due to the displaced training data.

4.5. Experiment 5: Image translation robustness

For regular image classifiers, translation invariance means that the classifier's predictions are unaffected by shifts in pixels. This runs counter to the typical use of INRs where any change in the pixels should be accurately captured in the reconstruction. An INR classifier should thus recognise that a particular change in the INR weight-space stems from an benign image shift.

As ARC learns local image features, we hypothesise that the weight-space remains largely intact when fitting to a shifted image, making the ARC classification pipeline more robust to such transformations. To test this, we return to the 100×100 padded Fashion-MNIST objects, where, alongside centred FMNIST objects (named *Centered*), we fit INRs to

Method	Test on *Centered*	Test on *Displaced*
ScaleGMN	74.50	13.00
Ours, PTv3	79.36	47.61

Table 7. **Exp. 5:** Test accuracy (%↑) when trained on INRs of *Centered* and evaluated on INRs of *Displaced*. SIREN-based methods (ScaleGMN) experience a complete collapse in classification accuracy due to significant differences in weight-space among the datasets. ARC paired with PTv3 demonstrates improved robustness. The drop in PTv3 is still significant enough to warrant extra experiments.

randomly displaced FMNIST images (named *Displaced*), depicted in Figure 7. SIRENs share the same initialisation, while ARCs use the same decoder over all datasets. Both ScaleGMN and PTv3 are trained on their respective INRs of the *Centered* dataset, and subsequently evaluated on *Displaced* INRs. Test accuracies are listed in Table 7. In SIREN-based methods, evaluating on *Displaced* is almost equivalent to random guessing, as the displaced FMNIST images have induced a drastic change to the SIREN weight-space. For ARC, we observe a much smaller, but still significant drop in generalising to *Displaced* ARCs. Like with SIRENs, the shifted image content may have induced significant differences to the latent content. However, an additional cause may be at play, where PTv3 is overfitting to the absolute positions of the ARC latent vectors. Although PTv3 does not explicitly use absolute latent positions, its pooling and local attention mechanisms *do* rely on them. We test the trained PTv3 model on *Centered* ARCs where the entire latent cloud is shifted. In Table 8, this shift causes a significant drop in accuracy degrades, confirming that PTv3 is not translation invariant. We therefore also consider Point Transformer v1 (PTv1) which uses a simpler nearest neighbour mechanism relying on relative position [65]. PTv1 shows no significant changes in accuracy drop under the same conditions as PTv3.

To address the issue of translation sensitivity, we retrain PTv3 but randomly shift the ARCs during training. This forces PTv3 to become more robust to absolute position differences. When evaluated on *Displaced*, the accuracy drop is significantly reduced compared to the original setup, as shown in Table 9. With the increased robustness against absolute position differences, the observed performance gap can be attributed to latent content differences between *Centered* and *Displaced*. PTv1 is also tested but shows significantly weaker generalisation properties.

4.6. Experiment 6. Ablation

In this experiment, we ablate various aspects of ARC and PTv3 to observe their impact on classification accuracy. For each ablation, ARCs are fit on a subset of 10k CIFAR10 images.

	No shift	Shift
PTv3	79.32	57.87
PTv1	76.37	76.46

Table 8. **Exp. 5:** Impact of absolute latent position shifts on PTv3 and PTv1 test accuracy (↑%). PTv3 shows a substantial drop in accuracy when the latent cloud is shifted, demonstrating its sensitivity to absolute latent positions. PTv1 shows no noticeable accuracy degradation which is in line with its relative position mechanisms.

Method	Test on *Centered*	Test on *Displaced*
PTv3	78.58	62.78
PTv1	76.37	49.36

Table 9. **Exp. 5:** Test accuracy (↑%) when training PTv3 on *Centered* with random latent cloud shifts. Naturally, training with latent cloud shifts improves PTv3's robustness to translation, and allows to analyse the accuracy gap due to latent content differences between *Centered* and *Displaced*. PTv1 had this property built-in but shows significantly weaker generalisation across the datasets.

Latent normalisation	Val. Acc.
None	52.67
Normalise Whole	54.58
Normalise Per-dim	58.68

Table 10. **Exp. 6:** Validation accuracy (%↑) under different latent normalisation strategies. We show how normalising each latent dimension independently yields the highest increase in performance.

First, we compare latent vector normalisation techniques and their impact on validation accuracy. Three normalisation strategies are compared. None: where no normalisation is applied, Normalise Whole: where the latent vectors are normalised using a scalar mean and standard deviation which are precomputed on a subset of the ARCs, and Normalise Per-dim: where the mean and standard deviation are precomputed per latent dimension on a subset of the ARCs. The effect of these techniques on test accuracy is listed in Table 10. Generally, applying normalisation to the latent vectors improves classification accuracy. This is in line with findings in other INR classification literature [26, 38, 68, 69]. Furthermore, applying normalisation across individual latent dimensions yields the highest improvement. This suggests that capturing variations specific to each latent dimension provides a more robust representation in classifying ARCs. We hypothesise that certain latent dimensions specialise in capturing distinct image features. This type of alignment would be induced by sharing the decoder. A similar property is introduced manually by means of harmonics of different frequencies, which results in better INR reconstructions [4].

		Latent dimension		
		8	**16**	**32**
# Latents	**10**	30.95	32.91	29.01
	25	41.12	40.05	41.43
	50	49.14	48.62	48.02

Table 11. **Exp. 6:** Validation accuracy (%↑) under different combinations of the number of latents versus the latent dimension. Increasing the number of latents has a clear positive impact on classification accuracy, whereas high latent dimensions provide diminishing results, particularly when there are few latents.

Next, we explore the trade-off between the number of latent vectors and the latent dimensionality of each latent vector in ARC, and aim to analyse their impact on classification accuracy. We fit a subset of 10k CIFAR10 images to each combination. In Table 11 the validation accuracies which PTv3 converges to are listed. Accuracy improves significantly as we increase the number of latent vectors in the image. Conversely, increasing the latent dimension yields diminishing returns on accuracy.

5. Conclusion

In this paper, we introduced ARC: Anchored Representation Clouds, a novel type of INR that couples latent space and image coordinate space. This allows ARC to leverage point cloud classification architectures to obtain state-of-the-art classification results and process image datasets that were previously unattainable for INR-based classification methods. Additionally, the unique weight-space of ARC provides efficient data augmentation techniques. We believe a key advancement in INR classification entails the unification of INR fitting and classification processes, which are currently treated as distinct stages. End-to-end coupling of the INR fitting and classification processes could lead to more competitive classification performance compared to their image-space counterparts whilst being more memory-efficient.

References

[1] Matthias Bauer, Emilien Dupont, Andy Brock, Dan Rosenbaum, Jonathan Richard Schwarz, and Hyunjik Kim. Spatial functa: Scaling functa to imagenet classification and generation. *arXiv preprint arXiv:2302.03130*, 2023.

[2] Rohan Chabra, Jan E Lenssen, Eddy Ilg, Tanner Schmidt, Julian Straub, Steven Lovegrove, and Richard Newcombe. Deep local shapes: Learning local sdf priors for detailed 3d reconstruction. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXIX 16*, pages 608–625. Springer, 2020.

[3] Yinbo Chen, Sifei Liu, and Xiaolong Wang. Learning continuous image representation with local implicit image function. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8628–8638, 2021.

[4] Zhang Chen, Zhong Li, Liangchen Song, Lele Chen, Jingyi Yu, Junsong Yuan, and Yi Xu. Neurbf: A neural fields representation with adaptive radial basis functions. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 4182–4194, 2023.

[5] Shin-Fang Chng, Sameera Ramasinghe, Jamie Sherrah, and Simon Lucey. Gaussian activated neural radiance fields for high fidelity reconstruction and pose estimation. In *Computer Vision – ECCV 2022*, pages 264–280, Cham, 2022. Springer Nature Switzerland.

[6] Taco Cohen and Max Welling. Group equivariant convolutional networks. In *Proceedings of The 33rd International Conference on Machine Learning*, pages 2990–2999, New York, New York, USA, 2016. PMLR.

[7] Elijah Cole, Grant Van Horn, Christian Lange, Alexander Shepard, Patrick Leary, Pietro Perona, Scott Loarie, and Oisin Mac Aodha. Spatial implicit neural representations for global-scale species mapping. In *International Conference on Machine Learning*, pages 6320–6342. PMLR, 2023.

[8] Luca De Luigi, Adriano Cardace, Riccardo Spezialetti, Pierluigi Zama Ramirez, Samuele Salti, and Luigi Di Stefano. Deep learning on implicit neural representations of shapes. In *International Conference on Learning Representations (ICLR)*, 2023.

[9] Johannes Dollinger, Philipp Brun, Vivien Sainte Fare Garnot, and Jan Dirk Wegner. Sat-sinr: High-resolution species distribution models through satellite imagery. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 10:41–48, 2024.

[10] Emilien Dupont, Adam Golinski, Milad Alizadeh, Yee Whye Teh, and Arnaud Doucet. COIN: COmpression with implicit neural representations. In *Neural Compression: From Information Theory to Applications – Workshop @ ICLR 2021*, 2021.

[11] Emilien Dupont, Hyunjik Kim, S. M. Ali Eslami, Danilo Jimenez Rezende, and Dan Rosenbaum. From data to functa: Your data point is a function and you can treat it like one. In *39th International Conference on Machine Learning (ICML)*, 2022.

[12] Gabriel Eilertsen, Daniel Jönsson, Timo Ropinski, Jonas Unger, and Anders Ynnerman. Classifying the classifier: dissecting the weight space of neural networks. *ArXiv*, abs/2002.05688, 2020.

[13] Amer Essakine, Yanqi Cheng, Chun-Wun Cheng, Lipei Zhang, Zhongying Deng, Lei Zhu, Carola-Bibiane Schönlieb, and Angelica I Aviles-Rivero. Where do we stand with implicit neural representations? a technical and performance survey, 2024.

[14] Elizabeth Fons, Alejandro Sztrajman, Yousef El-Laham, Alexandros Iosifidis, and Svitlana Vyetrenko. Hypertime: Implicit neural representations for time series. In *NeurIPS 2022 Workshop on Synthetic Data for Empowering ML Research*, 2022.

[15] Rui Gao and Rajeev K. Jaiman. H-siren: Improving implicit neural representations with hyperbolic periodic functions, 2024.

[16] Simon Giebenhain and Bastian Goldlücke. Air-nets : An attention-based framework for locally conditioned implicit representations. In *2021 International Conference on 3D Vision, 3DV 2021 : , virtual conference ; 1-3 December 2021 : proceedings*, pages 1054–1064, Piscataway, 2021. IEEE.

[17] Charles Godfrey, Davis Brown, Tegan Emerson, and Henry Kvinge. On the symmetries of deep learning models and their internal representations. *Advances in Neural Information Processing Systems*, 35:11893–11905, 2022.

[18] Ayaan Haque, Matthew Tancik, Alexei A Efros, Aleksander Holynski, and Angjoo Kanazawa. Instruct-nerf2nerf: Editing 3d scenes with instructions. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 19740–19750, 2023.

[19] Le Hou, Dimitris Samaras, Tahsin M Kurc, Yi Gao, James E Davis, and Joel H Saltz. Patch-based convolutional neural network for whole slide tissue image classification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2424–2433, 2016.

[20] Jeremy Howard. Imagenette.

[21] Langwen Huang and Torsten Hoefler. Compressing multidimensional weather and climate data into neural networks. *arXiv preprint arXiv:2210.12538*, 2022.

[22] Chiyu Jiang, Avneesh Sud, Ameesh Makadia, Jingwei Huang, Matthias Nießner, Thomas Funkhouser, et al. Local implicit grid representations for 3d scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6001–6010, 2020.

[23] Huaizu Jiang, Ishan Misra, Marcus Rohrbach, Erik Learned-Miller, and Xinlei Chen. In defense of grid features for visual question answering. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10267–10276, 2020.

[24] Ioannis Kalogeropoulos, Giorgos Bouritsas, and Yannis Panagakis. Scale equivariant graph metanetworks. *arXiv preprint arXiv:2406.10685*, 2024.

[25] Sosuke Kobayashi, Eiichi Matsumoto, and Vincent Sitzmann. Decomposing nerf for editing via feature field distillation. *Advances in Neural Information Processing Systems*, 35:23311–23330, 2022.

[26] Miltiadis Kofinas, Boris Knyazev, Yan Zhang, Yunlu Chen, Gertjan J Burghouts, Efstratios Gavves, Cees GM Snoek, and David W Zhang. Graph neural networks for learning equivariant representations of neural networks. *arXiv preprint arXiv:2403.12143*, 2024.

[27] Fanjie Kong and Ricardo Henao. Efficient Classification of Very Large Images with Tiny Objects. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2374–2384, New Orleans, LA, USA, 2022. IEEE.

[28] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[29] Yann LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

[30] Tianyang Li, Xin Wen, Yu-Shen Liu, Hua Su, and Zhizhong Han. Learning deep implicit functions for 3d shapes with dynamic code clouds. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022.

[31] David B Lindell, Dave Van Veen, Jeong Joon Park, and Gordon Wetzstein. Bacon: Band-limited coordinate networks for multiscale scene representation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 16252–16262, 2022.

[32] Lingjie Liu, Jiatao Gu, Kyaw Zaw Lin, Tat-Seng Chua, and Christian Theobalt. Neural sparse voxel fields. *Advances in Neural Information Processing Systems*, 33:15651–15663, 2020.

[33] Zhen Liu, Hao Zhu, Qi Zhang, Jingde Fu, Weibing Deng, Zhan Ma, Yanwen Guo, and Xun Cao. Finer: Flexible spectral-bias tuning in implicit neural representation by variable-periodic activation functions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2713–2722, 2024.

[34] Julien N. P. Martel, David B. Lindell, Connor Z. Lin, Eric R. Chan, Marco Monteiro, and Gordon Wetzstein. Acorn: adaptive coordinate networks for neural scene representation. *ACM Trans. Graph.*, 40(4), 2021.

[35] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[36] 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. *Commun. ACM*, 65(1):99–106, 2021.

[37] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Trans. Graph.*, 41(4):102:1–102:15, 2022.

[38] Aviv Navon, Aviv Shamsian, Idan Achituve, Ethan Fetaya, Gal Chechik, and Haggai Maron. Equivariant architectures for learning in deep weight spaces. In *Proceedings of the 40th International Conference on Machine Learning*, pages 25790–25816. PMLR, 2023.

[39] 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. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 22616–22625, 2024.

[40] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[41] Songyou Peng, Michael Niemeyer, Lars Mescheder, Marc Pollefeys, and Andreas Geiger. Convolutional occupancy networks. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part III 16*, pages 523–540. Springer, 2020.

[42] Nasim Rahaman, Aristide Baratin, Devansh Arpit, Felix Draxler, Min Lin, Fred Hamprecht, Yoshua Bengio, and Aaron Courville. On the spectral bias of neural networks. In *Proceedings of the 36th International Conference on Machine Learning*, pages 5301–5310. PMLR, 2019.

[43] Sameera Ramasinghe and Simon Lucey. Beyond periodicity: Towards a unifying framework for activations in coordinate-mlps. In *Computer Vision – ECCV 2022*, pages 142–158, Cham, 2022. Springer Nature Switzerland.

[44] Vishwanath Saragadam, Daniel LeJeune, Jasper Tan, Guha Balakrishnan, Ashok Veeraraghavan, and Richard G Baraniuk. Wire: Wavelet implicit neural representations. In *Conf. Computer Vision and Pattern Recognition*, 2023.

[45] Jonathan Richard Schwarz, Jihoon Tack, Yee Whye Teh, Jaeho Lee, and Jinwoo Shin. Modality-agnostic variational compression of implicit neural representations. In *Proceedings of the 40th International Conference on Machine Learning*. JMLR.org, 2023.

[46] Aviv Shamsian, Aviv Navon, David W. Zhang, Yan Zhang, Ethan Fetaya, Gal Chechik, and Haggai Maron. Improved generalization of weight space networks via augmentations. In *Forty-first International Conference on Machine Learning*, 2024.

[47] Vincent Sitzmann, Julien N.P. Martel, Alexander W. Bergman, David B. Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. In *arXiv*, 2020.

[48] Yannick Strümpler, Janis Postels, Ren Yang, Luc Van Gool, and Federico Tombari. Implicit neural representations for image compression. In *Computer Vision – ECCV 2022*, pages 74–91, Cham, 2022. Springer Nature Switzerland.

[49] Yu Sun, Jiaming Liu, Mingyang Xie, Brendt Wohlberg, and Ulugbek S Kamilov. Coil: Coordinate-based internal learning for imaging inverse problems. *arXiv preprint arXiv:2102.05181*, 2021.

[50] Gergely Szabó and András Horváth. Mitigating the bias of centered objects in common datasets. In *2022 26th International Conference on Pattern Recognition (ICPR)*, pages 4786–4792, 2022.

[51] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR, 2019.

[52] 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. *NeurIPS*, 2020.

[53] Matthew Tancik, Ben Mildenhall, Terrance Wang, Divi Schmidt, Pratul P Srinivasan, Jonathan T Barron, and Ren Ng. Learned initializations for optimizing coordinate-based neural representations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2846–2855, 2021.

[54] Thomas Unterthiner, Daniel Keysers, Sylvain Gelly, Olivier Bousquet, and Ilya Tolstikhin. Predicting neural network accuracy from weights. *arXiv preprint arXiv:2002.11448*, 2020.

[55] Kushal Vyas, Ahmed Imtiaz Humayun, Aniket Dashpute, Richard G. Baraniuk, Ashok Veeraraghavan, and Guha Balakrishnan. Learning transferable features for implicit neural representations, 2024.

[56] David R Wessels, David M Knigge, Samuele Papa, Riccardo Valperga, Sharvaree Vadgama, Efstratios Gavves, and Erik J Bekkers. Grounding continuous representations in geometry: Equivariant neural fields. *arXiv preprint arXiv:2406.05753*, 2024.

[57] Xiaoyang Wu, Li Jiang, Peng-Shuai Wang, Zhijian Liu, Xihui Liu, Yu Qiao, Wanli Ouyang, Tong He, and Hengshuang Zhao. Point transformer v3: Simpler faster stronger. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4840–4851, 2024.

[58] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

[59] Yiheng Xie, Towaki Takikawa, Shunsuke Saito, Or Litany, Shiqin Yan, Numair Khan, Federico Tombari, James Tompkin, Vincent sitzmann, and Srinath Sridhar. Neural fields in visual computing and beyond. *Computer Graphics Forum*, 41(2): 641–676, 2022.

[60] Alex Yu, Vickie Ye, Matthew Tancik, and Angjoo Kanazawa. pixelNeRF: Neural radiance fields from one or few images. In *CVPR*, 2021.

[61] Yu-Jie Yuan, Yang-Tian Sun, Yu-Kun Lai, Yuewen Ma, Rongfei Jia, and Lin Gao. Nerf-editing: geometry editing of neural radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 18353–18364, 2022.

[62] Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Youngjoon Yoo. Cutmix: Regularization strategy to train strong classifiers with localizable features. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 6023–6032, 2019.

[63] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *Computer Vision – ECCV 2014*, pages 818–833, Cham, 2014. Springer International Publishing.

[64] Jinlai Zhang, Lyujie Chen, Bo Ouyang, Binbin Liu, Jihong Zhu, Yujin Chen, Yanmei Meng, and Danfeng Wu. Pointcutmix: Regularization strategy for point cloud classification. *Neurocomputing*, 2022.

[65] Hengshuang Zhao, Li Jiang, Jiaya Jia, Philip Torr, and Vladlen Koltun. Point transformer, 2021.

[66] Zelin Zhao, Fenglei Fan, Wenlong Liao, and Junchi Yan. Grounding and Enhancing Grid-based Models for Neural Fields, 2024. arXiv:2403.20002 [cs].

[67] Jianqiao Zheng, Sameera Ramasinghe, Xueqian Li, and Simon Lucey. Trading positional complexity vs deepness in coordinate networks. In *European Conference on Computer Vision*, pages 144–160. Springer, 2022.

[68] Allan Zhou, Kaien Yang, Kaylee Burns, Adriano Cardace, Yiding Jiang, Samuel Sokota, J Zico Kolter, and Chelsea Finn. Permutation equivariant neural functionals. *Advances in neural information processing systems*, 36, 2024.

[69] Allan Zhou, Kaien Yang, Yiding Jiang, Kaylee Burns, Winnie Xu, Samuel Sokota, J Zico Kolter, and Chelsea Finn. Neural functional transformers. *Advances in neural information processing systems*, 36, 2024.

A. ARC implementation details

Provided with an image, the latent position are determined by sampling image gradient norm. Like [4], the number of latent vectors is proportional to the number of pixels in the image. We experimented with different combinations of the number of latents and latent dimension and found that # latents $= 0.05 \cdot$ # image pixels, combined with a feature dimension of $z = 32$ works well for most cases. This configuration is used in all experiments, unless otherwise noted. Like [4], the feature vectors are drawn from $U(-1\text{e}{-4}, 1\text{e}{-4})$. Upon initialisation, the indexing function U_n caches the index and relative position to the n nearest latent vectors. In all our experiments, we use a decoder of size $[n \cdot (z+2), n \cdot (z+2), d_{\text{out}}]$. The decoder is pre-trained by jointly training 100 ARC instances, aggregating the loss over all instance for each step. All our ARC instances, as well as the pretrained decoder, are trained for 500 steps. In fitting ARC we make use of mini-batching, whereby only 25% of the image coordinates are supervised each iteration. We found that this makes qualitatively little difference in reconstruction quality but makes fitting significantly faster. An ADAM optimiser is used with a learning rate of 0.005. We normalise images to [0,1] range.

B. Point Transformer v3 implementation details

Point Transformer v3 (PTv3) [57] is not intended to be used outside of point-cloud tasks that are typified by a 3D coordinate with optional features like colour or normals. Hence, PTv3 requires a few tweaks in order to be compatible with ARC. For instance, the latent coordinates are made three-dimensional by appending a 0 to them. To make PTv3 suitable for classification tasks, we replace the standard upsampling blocks by a global pooling layer, followed by a single linear layer that maps the feature dimension to the number of classes.

C. SIREN implementation details

Several of our experiments required custom SIREN datasets. We aimed to follow DWSnets implementation but found it to be erroneous compared to the regular SIREN specification [47]. Specifically, a 0.5 offset is added to the network's output, the first layer does not follow the prescribed initialisation scheme, and the bias layer is initialised to zero rather than the prescribed weight initialisation. We found that these errors are not readily apparent in low-resolution images such as the ones commonly used in INR classification. In fitting our larger resolution images, we observe heavy blurring in the reconstruction Figure 8. We opted to use a correct SIREN implementation instead.

We use default SIREN hyperparameters; $\omega_0 = 30.0$, no final layer activation function, ADAM optimiser and learning rate of 0.0005. The SIRENs are trained for 1000 steps.

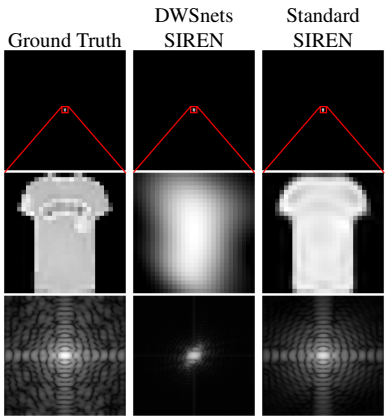


Figure 8. Comparison of 1024×1024 FMNIST objects reconstructed using DWSnets' SIREN implementation and a corrected SIREN implementation. Rows depict (from top to bottom) the full image, a zoomed-in view, and the FFT of the reconstruction. The left column shows the original image, highlighting its increased frequency content. From the FFTs, it is clear that spectral bias phenomena re-emerge in the SIREN INRs.

D. Experiment details

ScaleGMN. In utilising the ScaleGMN baseline [24], it proved to be rather unstable in training. The authors acknowledge this issue and take measures such as layer normalisation and skip connections to mitigate it. ScaleGMN is designed to work on SIREN datasets introduced by the DWSnets paper [38]. We found that a faulty SIREN implementation was used in creating these datasets (Appendix C). The SIREN datasets that we created therefore present an extra challenge as the provided ScaleGMN settings were created with DWSnet-SIRENs in mind. To quantify this error, we train ScaleGMN using the provided MNIST settings on 10k SIRENs that we fit ourselves. The resulting test accuracy is 88.89% after 71 epochs, which is 7.68% lower than the test accuracy which the authors obtained 96.57% on the whole MNIST dataset. We proceed to use ScaleGMN in our experiments where we use our own SIREN datasets, but are aware of the possibly suboptimal performance.

Toy datasets. In most experiments with toy datasets FMNIST was chosen as a basis. FMNIST poses a non-trivial classification challenge for current methods [24, 26, 38] and can easily be padded to increase the image size.