Hyper-network based Implicit Neural Representations for Image Classification

Gokhan Egri
MIT
gokhaneg@mit.edu

Abstract

Implicit Neural Representations (INR) have recently become popular due to their storage efficiency and performance in 2D and 3D scene representation. The primary short-coming of INRs is that a new network needs to be trained separately for each scene as different scene networks do not share information. Recent works have proposed hyper-networks as a solution to the problem of isolated training, where the weights of an INR are initialized using a different network which is shared between different scenes. In this work, we show that hyper-network INRs not only exhibit the same desirable properties (inpainting, denoising, etc.) of single-scene INRs but also converge onto image representations which can be used for downstream tasks such as classification. All results and figures in this paper can be reproduced using our code provided at https://github.com/egrigokhan/hyperinr-classifier.

1. Introduction

Different from conventional discrete representations of signals (e.g. represent images using discrete pixel values, 3D shapes with point clouds, etc.), Implicit Neural Representations parametrize signals as a *continuous function* that maps the domain of the signal (e.g. pixel coordinate) to the desired features (e.g. RGB values) using a neural network. With INR, signals are no longer coupled with spatial resolution, and it is thus easier to perform upsampling and interpolation tasks for images. Meanwhile, the memory required to parametrize the signals is independent of spatial resolution, but relies on the complexity of the underlying signal.

In this work, we provide a brief survey of the methods in the field and demonstrate that INR representations can be used to generate meaningful embeddings that perform well for downstream tasks such as classification which we believe can lead to future research directions.

Our contributions in this work are as follows:

1. We train a HyperINR network using Sitzmann *et al.*'s SIREN architecture and show that these networks exhibit the same degree of applicability to certain operations such as upsampling, denoising and inpainting as regular INRs.
2. We demonstrate that the latent representations learned by the HyperINRs are meaningful and can be used for downstream tasks such as image classification.
3. We show that the convergence behaviour of HyperINRs can be improved by using pre-trained latent representations.
4. We propose a new training scheme for HyperINRs which encourage better latent representations and improves the convergence behaviour of HyperINRs further.

2. Related Work

Implicit Neural Representations (INR) Recent works [7, 6, 8, 5, 9], have explored the potential of using multilayer perceptrons (MLP) for continuous, memory-efficient implicit neural representations that map 2D/3D signals in low-dimensional vectors (coordinates, viewing angles, etc.) to scene features (pixel color, SDF, etc.) for various applications.

Sitzmann *et al.*'s SIREN [6] leverages periodic activation functions and sinusoidal networks to represent complex natural signals and their derivatives. Mildenhall *et al.*'s NeRF [5], uses a continuous scene representation for novel view synthesis. Specifically, they represent continuous scenes with complex geometry and materials as 5D neural radiance fields, parameterized by basic MLP networks.

Dupont *et al.* [2] presents an extension of Sitzmann *et al.*'s work to generative image modeling using INRs where training a generative model on a continuous function of low-dimensional scene representations rather than on a fixed grid of pixels allows the generative models to scale independently of signal resolution and dimension.

Hyper-network based Implicit Neural Representations While INRs have shown great promise on a lot of tasks, they need to be trained individually for each scene, which creates extensive computational over-head for training. Unlike in conventional CNN-based models, the learned features for INRs aren't shared between different scenes.

This constraint has previously been addressed [6, 2] by the use of a hyper-network Φ which transforms signals (images, 3D scenes, etc.) into the MLP-domain by directly predicting the weights θ of the INR-network ψ such that $I \approx \psi(\chi; \theta = \Phi(I))$ where χ is the set of queried low-dimensional vectors ((x, y)-coordinates for 2D images, a 5D-vector for NeRF).

Ha *et al.* [3] uses an external neural network to map a latent vector describing the weight parameters at any given layer (characteristic vector, weight index within layer, weight size, etc.) to the weights of a second CNN network and finds that the system achieves respectable results in image recognition tasks with fewer trainable parameters. Mildenhall *et al.* [4] uses a related mechanism called Kernel Prediction Networks which maps noisy input images to a set of kernels which are then multiplied with the noisy image and aggregated for denoising. The kernels in this context correspond to the weights of the predicted network.

The use of a hyper-network allows us to circumvent the problem of isolated training by having a shared CNN-based encoder which outputs a latent representation of the original signal and mapping this latent representation using a shared hyper-network to predict the weights of the INR.

3. Methodology

3.1. Implicit Neural Representation (INR) network

A standard implicit neural representation (INR) network is shown in Figure 1.

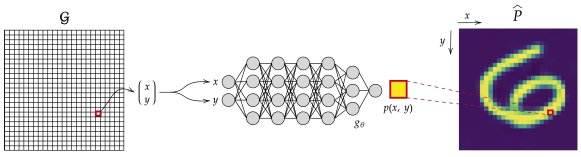

Figure 1. INR network.

The neural network $g_\theta(\cdot)$ with trainable parameters θ consists of L MLP layers and takes as input a k-dimensional coordinate vector and outputs an m-dimensional value vector. For 2D RGB images, we have $\{k, m\} = \{2, 3\}$, where the k-dimensional vector is sampled from a grid of positions $\mathcal{G}_{[W \times H \times k]}$ where W and H correspond to the width and height of the input image P respectively. By querying each individual coordinate pair $(x, y) \in \mathcal{G}$, we obtain an approximation $\hat{P}$ to the original image P.

The parameters θ of the network are trained using the MSE loss between the original image P and the approximation $\hat{P}$.

Samples from an INR trained on MNIST digits is shown in Figure 2.

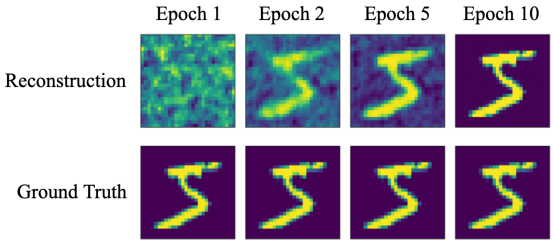

Figure 2. Single-scene INR outputs for the MNIST dataset.

3.2. INR with Hyper-network implementation

The INR with the hyper-network architecture is shown in Figure 3.

The entire model consists of three networks: the CNN encoder f_ϕ, the hyper-network h_{ϕ_i} and the INR g_θ with trainable parameters ψ, $\phi = \{\phi_1, \phi_2, \ldots, \phi_L\}$.

The CNN encoder f_ϕ comprises residual convolutional blocks and encodes the original image P to a 256-dimensional vector v.

The hyper-network h_{ϕ_i} comprises batch linear layers and maps the latent vector v to a weight vector θ_i.

The INR network g_θ has the same form as the INR network implementation from before where the trainable parameters $\theta = \{\theta_1, \theta_2, \ldots, \theta_L\}$ are initialized from the outputs of the hyper-network.

As the parameters θ of the INR network are predicted by the hyper-network, we train the parameters $\{\psi, \phi\}$ of only the CNN encoder and the hyper-network using the MSE loss between the approximate image $\hat{P}$ and the original image P.

Using only the MSE loss, we observe that the weights θ of the INR network does not converge. As such we impose two priors on the latent code v and the weights θ_i of the INR network which gives the loss as

$$\mathcal{L} = ||\hat{P} - P||_2^2 + \alpha_1 ||v||_2^2 + \alpha_2 ||\theta||_2^2$$

Following Sitzmann *et al.* [6], we set $\{\alpha_1, \alpha_2\} = \{0.1, 100\}$.

Samples from an HyperINR trained on MNIST digits is shown in Figure 4.

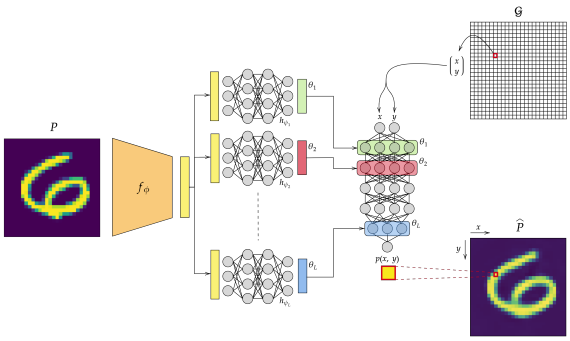

Figure 3. INR + Hyper-network diagram.

Figure 4. INR + Hyper-network outputs for the MNIST dataset.

3.3. Application of the HyperINR to different image operations

A detailed explanation of why INR and HyperINR representation of images are more suitable for some image operations such as upsampling, inpainting, denoising are provided in the Appendix (sec. A).

4. Experiments

In this project, we first train a HyperINR network which maps input images to the weights of an INR. After demonstrating that HyperINRs have the same reconstruction performance as individual INRs, we proceed with our primary three-stage experimentation:

1. **Can HyperINR be used for downstream tasks such as classification?:** As we are evaluating the viability of HyperINRs as canonical image representations, we are required to benchmark our architecture on common downstream tasks such as image classification which is an important benchmark for any canonical image representation. This experiment shows that in training the HyperINRs, the discovered latent vectors v and the weights of INR θ converge onto meaningful representations of the input.

2. **Can pre-trained classifier networks be used to generate HyperINR embeddings?:** Having established that the HyperINR converges onto meaningful embeddings, we experiment with initializing the CNN encoder f_ϕ with a pre-trained classifier without the final layer in order to obtain better convergence behaviour and to reduce training time.

3. **Can we embed classification as a secondary objective in HyperINR training?:** Finally we experiment with whether or not we can embed the classification task within the HyperINR training regime to generate embeddings better suited for classification while also improving the performance of the HyperINR reconstruction.

4.1. HyperINR Classifier

We train two image classifier networks on the MNIST dataset using (1) the latent embeddings v and (2) the INR weights θ generated by the HyperINR network.

Using the latent embeddings v. We create an image classifier using the trained embeddings v by constructing a two-layer MLP network which maps the 256-dimensional vectors v generated by the HyperINR to a 10-dimensional output as shown in Figure 5.

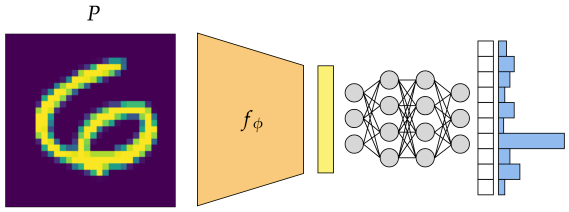

Figure 5. Classification using latent vector v.

We train the network with a cross-entropy loss using an Adam optimizer with learning rate $\lambda = 0.001$.

Using the INR weights θ. Using a 5-layer INR network, we observe that the hyper-network outputs a weight vector $\theta = \{\theta_1, \theta_2, \theta_3, \theta_4, \theta_5\}$ with the shapes $\{(256, 2), (256, 256), (256, 256), (256, 256), (1, 256)\}$.

As it is infeasible to flatten the entire set of weights, we combine all INR weights into a weight tensor θ^* with shape $(5, 256, 256)$ which concatenates all weight vectors θ (we repeat weights θ_1 and θ_5 in dimensions 2 and 1 respectively to achieve the shape $(256, 256)$).

We then construct a CNN classifier which uses four layers of convolutional layers followed by three MLP layers which map the weight matrix θ^* to a 10-dimensional output as shown in Figure 6.

We train the network with a cross-entropy loss using an Adam optimizer with learning rate $\lambda = 0.001$.

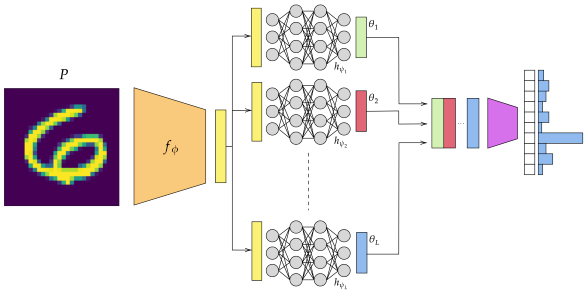

Figure 6. Classification using the weight tensor θ^*.

The training loss for both classifiers are shown in Figure 7. The latent vector classifier achieves a test accuracy of 98.42% and the weight classifier achieves a test accuracy of 97.96%.

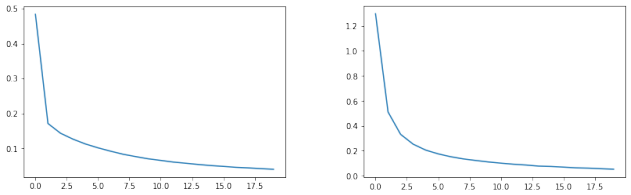

Figure 7. Training loss versus epoch. Left: using latent vectors v, right: using the weight tensor θ^*

4.2. HyperINR with Pre-trained Embeddings

We train a 2-layer MLP classifier on the MNIST dataset which maps the flattened images x to a 10-dimensional output with a hidden layer dimension of $d = 256$. After 3 epochs, we observe that our network converges to $\sim 99\%$ accuracy.

After the training, we remove the HyperINR encoder f_ϕ and train the HyperINR network by initializing the latent vectors v to the hidden layer representations of the pre-trained classifier.

The results for HyperINR and HyperINR with pre-trained embeddings are shown in Figure 8 (first two columns).

Here, we observe that the embeddings obtained from the pre-trained classifier allow the HyperINR network to converge faster and to clearer images. Note however that while the HyperINR (pre-trained embeddings) network is able to correctly identify the digit at the input, the reconstructions often converge to a prototypical representation of the input digit rather than replicating the form of the digit exactly (see the results for Iteration 2000, the reconstructed digit image is in fact a '5', however it does not accurately reconstruct the digit '5' in the input). This is likely due to the fact that classifier embeddings are trained to discriminate not between different images belonging to the same digit but rather between images belonging to different digits.

Figure 8. Sample HyperINR results throughout training iterations.

5. HyperINR with Classifier Loss

Having showed that (1) the HyperINR is capable of learning meaningful embeddings and that (2) using pre-trained embeddings improves convergence behaviour, we resolve the problem of prototypical convergence described previously by combining our previous two approaches.

We introduce an additional classifier network $t(\cdot)$ which consists of a 2-layer MLP which maps the embedding v to a 10-dimensional output with a hidden layer dimension of $d = 256$ which is shown in Figure 9.

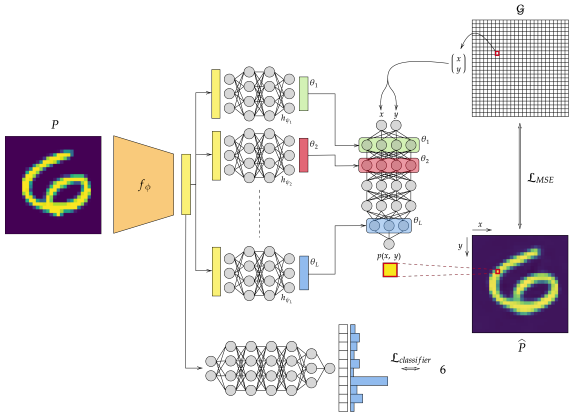

Figure 9. HyperINR with classifier loss diagram.

The classifier loss $\mathcal{L}_{classifier}$ is calculated using the cross-entropy loss between the predicted labels and the ground truth labels.

Then, our loss function becomes

$$\mathcal{L} = ||\hat{P} - P||_2^2 + 0.1||v||_2^2 + 100||\theta||_2^2 + 0.1 \times \mathcal{L}_{classifier}$$

We optimize the HyperINR and $t(\cdot)$ networks end-to-end using an Adam classifier with a learning rate of $\lambda = 0.01$ for the classifier and $\lambda = 0.00005$ for the rest of the network.

The sample results for the HyperINR trained with the classifier network are shown in Figure 8 (third column).

Here, we observe that training end-to-end with a classifier network improves the clarity of the reconstructions in the early iterations (observe that the HyperINR results are over-blurred, the HyperINR (pre-trained embeddings) results are noisy whereas the HyperINR (trained with classifier loss) results are clear). We also observe that the embeddings generated by the HyperINR (trained with classifier loss) network also mostly resolve the problem of prototypical convergence seen in the case of the HyperINR (pre-trained embedding networks)[1].

The embeddings generated by the HyperINR (with classifier loss) achieve a test accuracy of 98.84% which is an improvement over the classification accuracy of embeddings generated by the HyperINR and HyperINR (pre-trained embeddings) network.

6. Discussion

Some of our comments on the current state of the project are as follows:

- As far as we are aware, our work presents the first demonstration of INR and HyperINR performance in downstream tasks such as image classification. As such, in order to cover as much new ground as possible in a short amount of time, we have chosen to experiment with the MNIST dataset for fast-prototyping and sanity-checking, as stated in our original proposal. We however include early results on the CIFAR-10 dataset in the Appendix (sec. B).

- While training the HyperINR with the classifier, we have observed that the inclusion of an additional classifier network in the training loop increased the training time per-iteration as expected. This increase was however compensated by the better convergence behaviour we got in return, and as such our proposed method gave a net improvement in convergence. We note however that this may not be the case for more sophisticated datasets/downstream tasks which require more complex networks to be included such that the increase in training time may not be compensated by the improvements in convergence behaviour.

- While the results presented in this report are significant on their own and further demonstrate the viability of HyperINRs as canonical image representations, their primary importance stems from the fact that they point to a previously unexplored common ground between INRs and feature learning which can potentially lead to interesting research projects combining state-of-the-art feature learning methods such as contrastive learning [1] with the nascent INR literature.

[1]While the Iteration 1250 and Iteration 2000 rows in Figure 8 largely confirm this observation, the reader is encouraged to generate more results using our publicly available code to confirm that this is indeed the case.

7. Conclusion

We demonstrate the first instance of an hyper-network based implicit neural representation network being used for a downstream image recognition task such as classification. After showing that the feature representations learned by HyperINR methods can be used to get competitive classification scores, we further improve both the convergence behaviour and the quality of the network reconstructions by creating and benchmarking two new training schemes, using pre-trained embeddings and training embeddings with a secondary classification objective respectively.

While we believe that the results presented in this paper are strong on their own, we posit that their significance is compounded upon considering their role as an instance of the fingerpost towards future research directions which combine the extensive feature learning literature with the promising implicit neural representation framework.

References

[1] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations, 2020. 5

[2] Emilien Dupont, Yee Whye Teh, and Arnaud Doucet. Generative models as distributions of functions, 2021. 1, 2

[3] David Ha, Andrew Dai, and Quoc V. Le. Hypernetworks, 2016. 2

[4] Ben Mildenhall, Jonathan T. Barron, Jiawen Chen, Dillon Sharlet, Ren Ng, and Robert Carroll. Burst denoising with kernel prediction networks, 2018. 2

[5] 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. In *European Conference on Computer Vision*, pages 405–421. Springer, 2020. 1

[6] Vincent Sitzmann, Julien Martel, Alexander Bergman, David Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. *Advances in Neural Information Processing Systems*, 33, 2020. 1, 2

[7] Vincent Sitzmann, Justus Thies, Felix Heide, Matthias Nießner, Gordon Wetzstein, and Michael Zollhöfer. Deepvoxels: Learning persistent 3d feature embeddings, 2019. 1

[8] Vincent Sitzmann, Michael Zollhöfer, and Gordon Wetzstein. Scene representation networks: Continuous 3d-structure-aware neural scene representations, 2020. 1

[9] Matthew Tancik, Pratul P. Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan T. Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains, 2020. 1

[10] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Deep image prior. *International Journal of Computer Vision*, 128(7):1867–1888, Mar 2020. 6

A. HyperINRs on different image operations

In this section, we briefly describe the suitability of INR and HyperINR networks to certain operations by walking through how these operations are carried out in the INR domain compared to the canonical pixel domain:

1. **Upsampling**
   (a) **In the pixel domain**, upsampling is carried out by spreading out the pixel grid to the desired size and filling the spaces between the known pixels with values often generated as an interpolation of the surrounding known pixels.
   (b) **In the INR domain**, as the image is generated by "sampling" a continuous function at a $[-1, 1] \times [-1, 1]$ grid of coordinates, upsampling in this context refers to "sampling" the same continuous function with a denser grid of coordinates. In this sense, we observe that upsampling in the INR domain is in fact closer to the original meaning of the term "upsampling" in the signal literature.
2. **Inpainting**
   (a) **In the pixel domain**, inpainting of a part of an image which has been taken out is achieved by externally specifying a means by which the subtracted values are interpolated from the known pixel values.
   (b) **In the INR domain**, as the image is the output of a continuous implicit function, the interpolation method by which the removed area is inpainted is determined automatically by a learned prior on the weights of the INR (e.g. for the MNIST dataset, the hyper-network is able to fill in the removed part based on its knowledge of the general form of the digit '3').
3. **Denoising**
   (a) **In the pixel domain**, denoising an image requires us to separate the image into an image and a noise term whereby we try and subtract the latter. The means by which this noise term is generated is however again decided externally.
   (b) **In the INR domain**, denoising occurs similarly to Ulyanov *et al.*'s Deep Image Prior [10], where a convolutional neural network trained to reconstruct the image at its input is unable to reconstruct the higher-frequency (noise) terms in the input and therefore outputs a denoised, low-frequency (image) reconstruction of the input. This is caused by an intrinsic prior in convolutional neural networks towards low-frequency, "natural looking" images.

Sample results of these three operations using one of our trained HyperINR networks are shown in Figure 10.

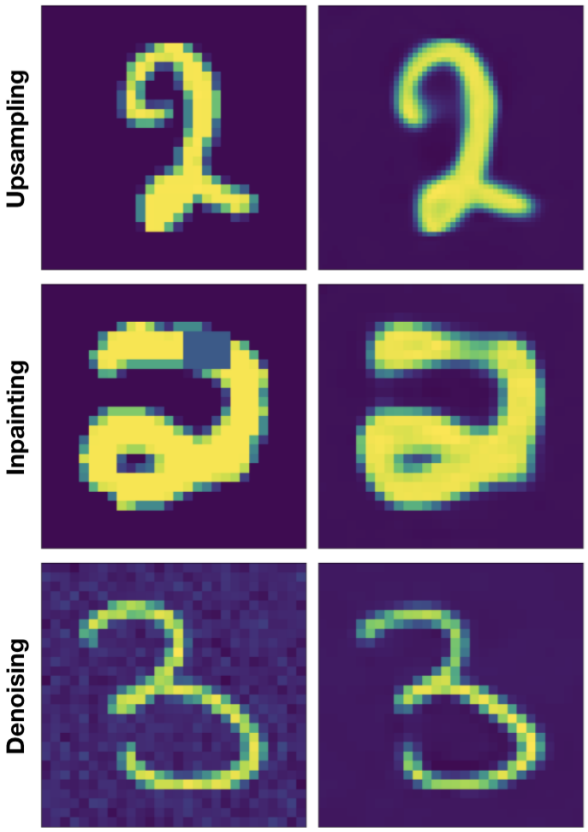

Figure 10. Sample results for the three image operations using the HyperINR.

B. CIFAR-10 Results

Here, we include some early results for the training of a HyperINR network on the CIFAR-10 dataset. As far as we

are aware, there is currently no paper which presents results of training a HyperINR on the CIFAR-10 dataset.

We have seen that the majority of papers in this area often use the MNIST and CelebA datasets for experimentation since (1) the framing of the images in these datasets are fairly consistent (in MNIST (CelebA), digits (faces) are always centered and are roughly on the same scale) and (2) the difference between any two images, even between images belonging to two different classes, is not significant and can easily be compressed into a latent representation.

By the same line of reasoning, we can argue that the CIFAR-10 dataset is a difficult benchmark for HyperINRs as the (1) framing, (2) scale and (3) position of the objects of interest are inconsistent between images and (4) there is a fair amount of variance between even the images of the same class.

Samples from a HyperINR network trained on the CIFAR-10 dataset are shown in Figure 11. The architecture for the HyperINR is the same as the HyperINR trained for the MNIST dataset in the main paper, only with 3-channel inputs and outputs for the RGB images.

Observe that while the reconstructions are not as clear as in the case of the MNIST dataset, the general outlines and identifiable features of the objects in each image are largely recognizable. The blurriness is likely due to the fact that we have chosen to use the same HyperINR architecture for the CIFAR-10 dataset as the MNIST dataset whereas the former is a considerably more difficult benchmark for the previously cited reasons. As such we believe that the network performance on the CIFAR-10 dataset can also be improved by increasing the number of layers or the trainable parameters of the networks in the HyperINR.

Using the latent vectors from the trained HyperINR network, an MLP-classifier is able to quickly converge to $\sim 70\%$ testing accuracy on the CIFAR-10 dataset which, while proving that our results are generalizable across datasets, is also likely to improve upon using more trainable parameters and a latent vector size higher than $d = 256$.

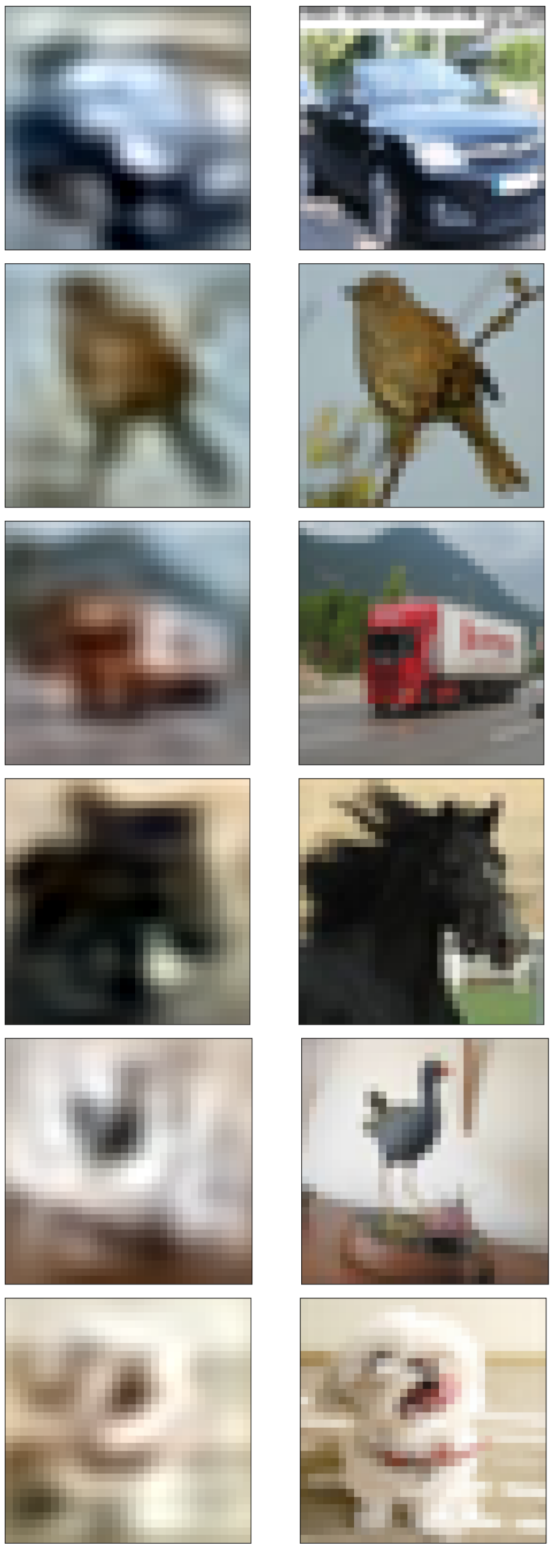

Figure 11. Sample results for the CIFAR-10 dataset.