Understanding Learning Dynamics of Neural Representations via Feature Visualization at Scale

Anonymous Author(s)
Affiliation
Address
email

Abstract

How does feature learning happen during the training of a neural network? We developed an accelerated pipeline to synthesize maximally activating images ("prototypes") for hidden units in a parallel fashion. Through this, we were able to perform feature visualization at scale and to track the emergence and development of visual features across the training of neural networks. Using this technique, we studied the 'developmental' process of features in a convolutional neural network trained from scratch using SimCLR with or without color jittering augmentation. After creating over one million prototypes with our method, tracking and comparing these visual signatures showed that the training with color-jitter augmentation led to constantly diversifying high-level features, while no color-jittering led to more diverse low-level features but less development of high-level features. These results illustrate how feature visualization can be used to understand hidden training dynamics under different training objectives and data distribution.

1 Introduction

The neural representation of images is often analyzed in a multi-dimensional vector space (4; 16; 7; 18), formed by the activation of neurons. Usually, the representations are analyzed in this neural space, for example, the representation of different object categories form "object manifolds" within the space(4; 6). An alternative perspective is to think about neural representation in their domain i.e. on the image manifold (29; 26). From this perspective, the tuning of each neuron is a function (i.e. landscapes) on the manifold, with peaks and troughs. The peaks of the landscape correspond to images that highly activate these neurons. Note that, the axes of the neural vector space are the tuning functions of these neurons, thus the highly activating images could be deemed as the meaning of these axes. Through this paper, we call the activation maximizing images for each neuron a "prototype" ((23)). Thus, obtaining the prototypes for all units in a neural network could provide a full basis set for understanding the representation of this network.

Feature visualization has been a prominent technique for finding and synthesizing prototypes in deep artificial neural networks (20; 21; 10), and the biological brain (22; 28; 12). But normally, these methods were applied to one unit at a time, hard for application at scale.

In this work, we developed an accelerated pipeline to extract "prototypes" in a parallel fashion. Through this, we were able to apply feature visualization on a large scale, tracking the emergence and change of "prototypes" across the whole training process of neural networks – creating a visual signature for each network checkpoint. We leveraged this method to study the 'development' of features in a convolutional neural network trained from scratch via self-supervised learning. The preliminary results illustrate how different training objectives and data distribution led to different "development" dynamics of these prototypes.

Submitted to 37th Conference on Neural Information Processing Systems (NeurIPS 2023). Do not distribute.

2 Methods

2.1 Feature Visualization at Scale

Our method is based on (19; 27; 25), where feature visualization is performed within the latent space of a pretrained generative adversarial network (GAN)(8). This GAN can be regarded as the natural image prior or the regularizer for the optimization, which counteracts the adversarial artifacts (20). For each target unit, we optimize its activation using a hybrid of CMA-ES (14) and gradient optimization: we performed 10s of CMA steps to search for an initialization that evoked non-zero activation in the unit, then we performed 100s of gradient ascent steps to visualize the features. We implemented both CMA and Adam optimization in a more paralleled fashion, which enables feature visualization for each and every channel in a layer in one run. This method increased our overall throughput by 33 fold (details in Sec. 6.2).

2.2 Experiment Setup for Self-Supervised Learning

For all our experiments, we used ResNet18(15) as our neural network architecture and trained it with a popular self-supervised learning algorithm (SimCLR, (2)) on STL10 (5) dataset for 100 epochs. These algorithms train a neural network to associate different augmented views of the same image as similar representations, and those of different images as dissimilar ones. One key component of this method is the augmentation pipeline, which determines what type of transformation should the neural network be invariant to. Here we had two training conditions with different augmentation pipelines and tested their effect on the development process of prototypes. 1) **Color jitter** (abbreviated as *clrjit*), the default augmentation pipeline of SimCLR; 2) **Keep Color** (*keepclr*), the same pipeline with color jittering and random grayscale augmentation disabled, which keeps the original color of the image. As the two conditions exposed the neural networks to different image statistics and pushed them with different objectives, we'd like to see if we can understand these differences better through the lens of prototype distribution.

Specifically, we completed three training runs of ResNet18 from scratch with random seeds 1,2,3 with color jittering and keep color augmentations; resulting in 6 training sequences of 101 epochs neural network checkpoints. For each checkpoint, we performed prototype extraction twice (details in Sec.6.1). Thus, all these prototypes can be indexed by (training condition, run number, evolution repeat, epoch number, layer, channel).

We evaluated the quality of their representations using the linear probe protocol (see Sec.6.1), namely fitting a linear classifier to see how well it classifies the test set images. The models trained with color jittering augmentation have far higher classification accuracy ($70.0 \pm 0.3\%$) than the models without ($49.8 \pm 0.3\%$) (Fig.4). This is consistent with the original observations of the importance of color augmentation in SimCLR (Fig.5 in (2)). From this perspective, the clrjit models have better feature representations for object classification. We want to dissect this difference of representation quality and link it back to the development of prototypes.

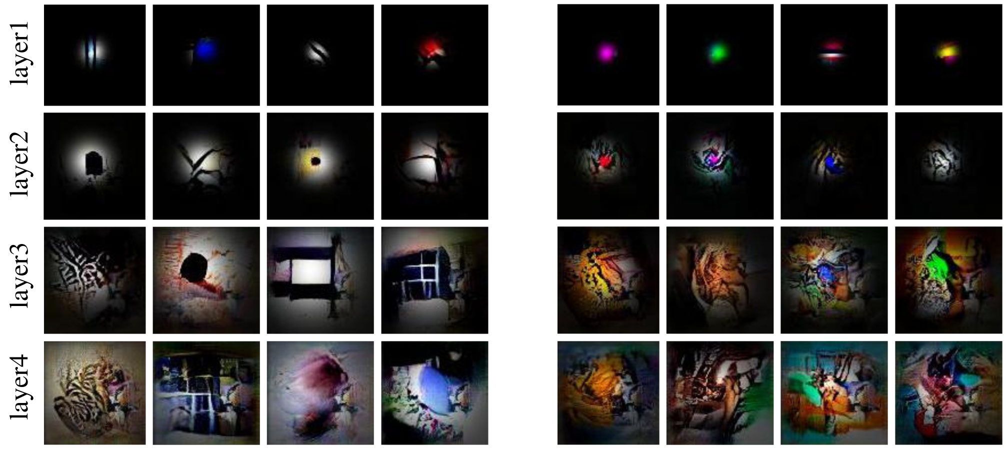
3 Results

3.1 Visual difference of the prototypes between conditions

How do the learned features differ between the two training conditions? We first visually inspected the distribution of prototypes in each layer for the two conditions (Fig.1).

For the color jittering condition (Fig.1a), in layer 1, the prototypes masked with their respective receptive fields primarily captured patterns like black stripes on a white background (a1-1,1-3[1]), and solid colors like Prussian blue (a1-2), white/off-white, black, and partial cyan, red, and green shades. In the second layer, more square-circle figures like squircles (a2-1,2-3) were observed along with intricate patterns like thick lines and irregular line figures that somewhat resembled a cracked earth texture or an abstract glass painting texture (a2-2,2-4). In layer 3, the features became finer, as high-frequency textures were observed (a3-1), along with black and white squircles (a3-2), rectangles (a3-3), and grids (a3-4). In layer 4, high-frequency textures (a4-1) were observed as well as distorted grid-like structures (a4-2). Few prototypes showed a gradient of colors resembling fur-like (a4-3) and watercolor textures (a4-4).

[1] 1-based row-col index in the grid a of Fig.1, same convention below



(a) SimCLR with color jittering (clrjit) (b) SimCLR without color jittering (keepclr)

Figure 1: Example prototypes for networks trained with color jittering (clrjit) and without (keepclr)

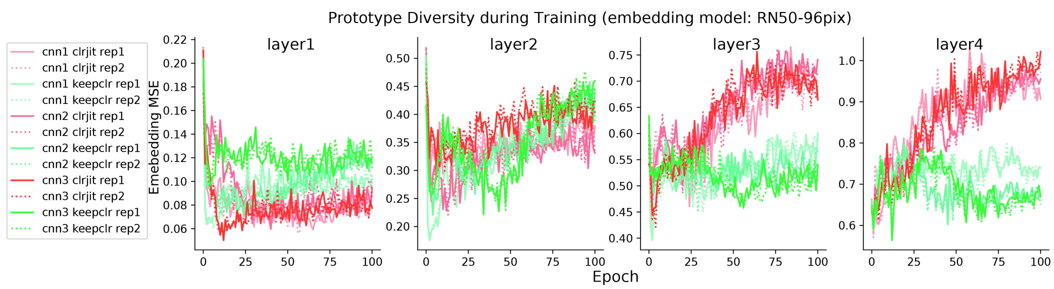


Figure 2: **Dynamics of prototypes diversity during training**.

In contrast, in the keep color condition (Fig.1 b), in layer 1, a vibrant array of colors were observed including magenta / pink (b1-1), green (b1-2), red, blues, cyan, and yellow along with colorful stripes (b1-4). In layer 2, high-frequency textures (b2-4) were present along with colored gemstone-like shapes embedded in high-frequency textures (b2-1,b2-2,b2-3). In higher layers (layer 3 and layer 4), there were a significant number of high-frequency textures present mainly in the warm hues like oranges and reds (row3,4). These texture patterns are perceptually more similar to each other than the ones in color-jittering conditions.

3.2 Developmental process of prototypes during training

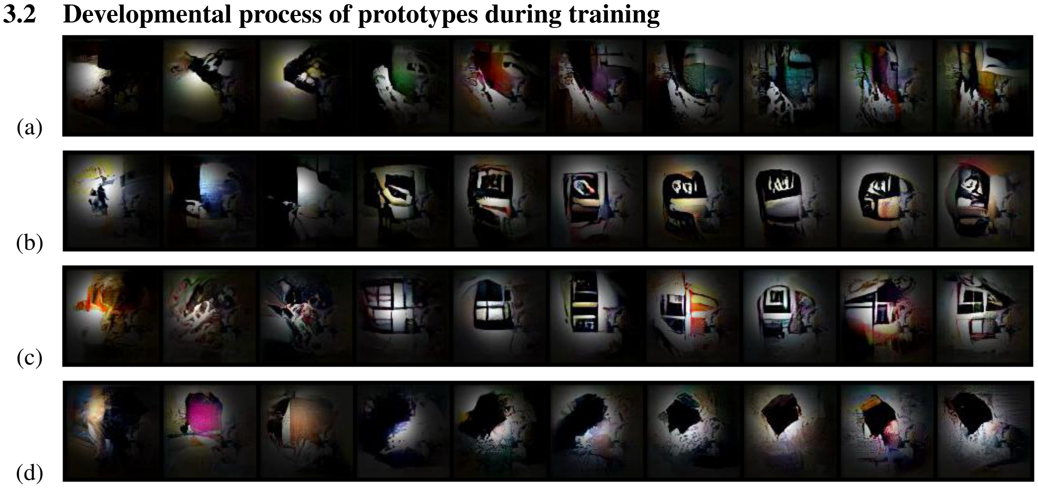


Figure 3: Development of prototypes through training for color jittering (clrjit) condition, layer 3. Columns denote 0,10,20,... to 90 epoch; Rows denote Units 1, 2, 12, and 98 (0-based index).

So how do the neural networks arrive at these features? We visualized the prototypes during training epochs as a row. For instance, for these example units in layer3 of a clrjit network (Fig.3, see Fig.8

for keepclr condition), it can be seen that each of these units goes through an initial stage of rapid erratic change, and then settles down to a primitive version of the final feature at around epoch 30, then elaborate this primitive form until the end. The latter half of epochs have more similarities between each other for each unit than the initial epochs however each of these individual units keeps diversifying with respect to each other throughout the training process as seen in Fig.2.

3.3 Distance structure between prototypes

Next, we quantified our perception by computing the distance structure between prototypes to understand their distribution and dynamics during training. We computed the Mean Squared Error (MSE) and Cosine distance in both pixel space and the embedding space of some pre-trained networks (detail in Sec.6.4). Further, we computed the prototype similarity with the images masked by their functional receptive field mask (Sec.6.3) to focus on the central feature.

We quantified the **diversity of prototypes** during training: for each epoch, we computed the pairwise distance matrix between prototypes of all channels, and then computed the mean distance between prototype pairs (Fig. 2). We found salient differences between the two training conditions (color jitter, keep color), and consistencies between repeated training runs and prototype evolutions. Here we showed results with ResNet50 as our embedding model and MSE as the distance metric. For layer 1, the diversity dropped drastically in the first few epochs, and then grew to a stable level. In the end, keepclr condition led to more diverse prototypes than clrjit. For layer 2, after the initial drop of diversity, the prototypes diversify again, and keepclr condition led to slightly higher diversity. However, for layers 3 and 4, the keepclr condition increased prototype diversity early on and then they plateaued; in contrast, the clrjit condition led to a constant increase in prototype diversity without plateau. When the cosine distance is used instead of MSE (Fig.5), a shift is observed in the dynamics, albeit the diverging trend between conditions remained similar to the MSE result. The consistency of the color-jittering networks being at the top tends to demonstrate how these networks develop more diverse prototypes through their evolutions. Further observations about the rate of change and the stability of prototype across re-evolution are noted in Sec. 7.2

This observation is intriguing. We interpreted it as follows, the color jittering augmentation constantly drives the network to find higher-level visual features to solve the instance classification task; while without color jittering, slightly more diverse lower-level features (layer1,2) suffice to solve the task. Intuitively, when SimCLR training doesn't randomly augment the color (keepclr), one simple way to find views of the same image is to look for similar color palettes. Thus, it's intuitive that the keepclr network needs to be more sensitive to image colors (Fig. 1). In comparison, with color jittering, the network cannot rely on color matching as a reliable strategy, and it needs to discover higher-level form consistencies, which may drive the diversification of deeper layer features.

4 Related Work

Understanding self-supervised representation Self-supervised learning (SSL) has been popular in vision for feature learning. In these paradigms, the pre-training uses different objectives to learn features and these features are directly used in the downstream application, with little fine-tunings. But what is a good feature representation? Usually, these features were evaluated based on the performance of the downstream task. One open question is to understand and evaluate representations learned by models without using a downstream task. Many works analyzed the representation similarity of SSL networks and supervised networks (13). Recent works have creatively used generative models to understand the "interpretation" of the same image by pre-trained networks to show their different biases (1).

5 Discussion

It has been noticed that the randomly initialized neural networks have lower dimensional representations, i.e. the activations of hidden units are more correlated across populations; and supervised training increased the dimensionality of the representation, and the increase is more salient in deeper layers (Fig. H.1A, (9)). In the other perspective, the units become less correlated to each other during training, which is consistent with our finding that the prototypes of the units become more and more diverse during training.

Prototype diversity seems like a promising proxy for the richness of neural representation, however, it may not be the full story, these prototypes need to be related to the training and testing distribution of images in a meaningful way to have high quality. Thus, one deep and open question is to elucidate the relationship between these prototypes and the training distribution of the network (11).

References

[1] BORDES, F., BALESTRIERO, R., AND VINCENT, P. High fidelity visualization of what your self-supervised representation knows about. *arXiv preprint arXiv:2112.09164* (2021).

[2] CHEN, T., KORNBLITH, S., NOROUZI, M., AND HINTON, G. A simple framework for contrastive learning of visual representations. In *International conference on machine learning* (2020), PMLR, pp. 1597–1607.

[3] CHEN, X., LIU, X., AND JIA, Y. Combining evolution strategy and gradient descent method for discriminative learning of bayesian classifiers. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation* (2009), pp. 507–514.

[4] CHUNG, S., LEE, D. D., AND SOMPOLINSKY, H. Classification and geometry of general perceptual manifolds. *Physical Review X 8*, 3 (2018), 031003.

[5] COATES, A., NG, A., AND LEE, H. An analysis of single-layer networks in unsupervised feature learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics* (2011), JMLR Workshop and Conference Proceedings, pp. 215–223.

[6] COHEN, U., CHUNG, S., LEE, D. D., AND SOMPOLINSKY, H. Separability and geometry of object manifolds in deep neural networks. *Nature communications 11*, 1 (2020), 746.

[7] DICARLO, J. J., AND COX, D. D. Untangling invariant object recognition. *Trends in cognitive sciences 11*, 8 (2007), 333–341.

[8] DOSOVITSKIY, A., AND BROX, T. Generating images with perceptual similarity metrics based on deep networks. *Advances in neural information processing systems 29* (2016).

[9] ELMOZNINO, E., AND BONNER, M. F. High-performing neural network models of visual cortex benefit from high latent dimensionality. *bioRxiv* (2022), 2022–07.

[10] ERHAN, D., BENGIO, Y., COURVILLE, A., AND VINCENT, P. Visualizing higher-layer features of a deep network. *Technical Report, Univeristé de Montréal* (01 2009).

[11] FIELD, D. J. Relations between the statistics of natural images and the response properties of cortical cells. *Josa a 4*, 12 (1987), 2379–2394.

[12] FÖLDIÁK, P. Stimulus optimisation in primary visual cortex. *Neurocomputing 38-40* (2001), 1217–1222. Computational Neuroscience: Trends in Research 2001.

[13] GRIGG, T. G., BUSBRIDGE, D., RAMAPURAM, J., AND WEBB, R. Do self-supervised and supervised methods learn similar visual representations? *arXiv preprint arXiv:2110.00528* (2021).

[14] HANSEN, N., MÜLLER, S. D., AND KOUMOUTSAKOS, P. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (cma-es). *Evolutionary computation 11*, 1 (2003), 1–18.

[15] HE, K., ZHANG, X., REN, S., AND SUN, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2016), pp. 770–778.

[16] HUNG, C. P., KREIMAN, G., POGGIO, T., AND DICARLO, J. J. Fast readout of object identity from macaque inferior temporal cortex. *Science 310*, 5749 (2005), 863–866.

[17] LOSHCHILOV, I. A computationally efficient limited memory cma-es for large scale optimization. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation* (2014), pp. 397–404.

[18] MAJAJ, N. J., HONG, H., SOLOMON, E. A., AND DICARLO, J. J. Simple learned weighted sums of inferior temporal neuronal firing rates accurately predict human core object recognition performance. *Journal of Neuroscience 35*, 39 (2015), 13402–13418.

[19] NGUYEN, A., DOSOVITSKIY, A., YOSINSKI, J., BROX, T., AND CLUNE, J. Synthesizing the preferred inputs for neurons in neural networks via deep generator networks, 2016.

[20] OLAH, C., MORDVINTSEV, A., AND SCHUBERT, L. Feature visualization. *Distill 2*, 11 (2017), e7.

[21] OLAH, C., SATYANARAYAN, A., JOHNSON, I., CARTER, S., SCHUBERT, L., YE, K., AND MORDVINTSEV, A. The building blocks of interpretability. *Distill 3*, 3 (2018), e10.

[22] PONCE, C. R., XIAO, W., SCHADE, P. F., HARTMANN, T. S., KREIMAN, G., AND LIVINGSTONE, M. S. Evolving Images for Visual Neurons Using a Deep Generative Network Reveals Coding Principles and Neuronal Preferences. *Cell 177*, 4 (may 2019), 999–1009.e10.

[23] ROSE, O., JOHNSON, J., WANG, B., AND PONCE, C. R. Visual prototypes in the ventral stream are attuned to complexity and gaze behavior. *Nature communications 12*, 1 (2021), 1–16.

[24] SUSMELJ, I., HELLER, M., WIRTH, P., PRESCOTT, J., AND ET AL., M. E. Lightly. *GitHub. Note: https://github.com/lightly-ai/lightly* (2020).

[25] WANG, B., AND PONCE, C. R. High-performance evolutionary algorithms for online neuron control. In *Proceedings of the Genetic and Evolutionary Computation Conference* (New York, NY, USA, 2022), GECCO '22, Association for Computing Machinery, p. 1308–1316.

[26] WANG, B., AND PONCE, C. R. Tuning landscapes of the ventral stream. *Cell Reports* (nov 2022).

[27] XIAO, W., AND KREIMAN, G. XDream: Finding preferred stimuli for visual neurons using generative networks and gradient-free optimization. *PLOS Computational Biology 16*, 6 (jun 2020), e1007973.

[28] YAMANE, Y., CARLSON, E. T., BOWMAN, K. C., WANG, Z., AND CONNOR, C. E. A neural code for three-dimensional object shape in macaque inferotemporal cortex. *Nature Neuroscience 11*, 11 (nov 2008), 1352–1360.

[29] ZAVATONE-VETH, J. A., YANG, S., RUBINFIEN, J. A., AND PEHLEVAN, C. Neural networks learn to magnify areas near decision boundaries. *arXiv preprint arXiv:2301.11375* (2023).

6 Extended Method

6.1 Details for Self-Supervised Learning

We used a popular self-supervised learning pipeline for training neural networks, SimCLR (2). We used the implementation in lightly-ai (24).

Augmentations We tested two augmentation conditions, 1) the default stochastic augmentation pipeline with color jittering, 2) the same pipeline with color jittering and random grayscaling disabled (`cj_prob=0.0`, `random_gray_scale=0.0`)

Model Architecture For the model backbone, we used the ResNet18 model (15), with 128d projection head.

Dataset For computational feasibility, we experimented with the SimCLR algorithm on the STL10 dataset (5) with the 96-pixel resolution, a classic testbed for self-supervised learning. Where the unlabeled training set has 100000 images, the training set has 500 images for each of the 10 classes and the testing set has 800 images for each of the 10 classes. The 10 classes are airplane, bird, car, cat, deer, dog, horse, monkey, ship, and truck, where 4 of those are animate man-made vehicles, and 6 of those are animate species.

Training hyperparameters For all models, we trained 100 epochs with Stochastic Gradient Descent with Cosine Annealing learning rate ($lr = 6 \times 10^{-2}, momentum = 0.9, weight\_decay = 5 \times 10^{-4}$)

Evaluation For evaluation, we used the linear probe protocol: We fixed all parameters of the CNN and used it to map images from the training and test set to feature vectors. Here no image augmentation was used, only RGB value normalization. Then we fit a linear classifier based on the training set features and evaluated the classifiers on the test set features. We used three ways to fit the linear classifier: Logistic regression (`LogisticRegression` from `sklearn`), Linear Support Vector Classifier (`LinearSVC` from `sklearn`), and gradient descent (Adam) on Cross Entropy Loss.

6.2 Scalable methods for synthesizing prototypes

This work requires a huge amount of highly activating images (prototypes) to be synthesized, so we developed a more scalable way to synthesize them efficiently. Specifically, we parallelized the hybrid of CMA-ES (14; 17) and gradient optimization (3) to optimize the images for each channel in a layer independently. for the major layers in each convolutional neural network.

As a concrete example, we need to synthesize, $101 \times (64 + 128 + 256 + 512) = 96960$ prototypes for all channels for each epoch of a training run in ResNet18. This will take around 269hrs on a single GPU using the previous non-parallelized CMA-ES algorithm per channel pipeline. Using our current method, it takes only 8hrs on a single GPU, which is a 33 times speed up. Using this method, we synthesized over 1 million prototypes in a reasonable time.

6.3 Methods for computing receptive field of units

We used gradient-based receptive field mapping for hidden units. We denote the hidden unit as $f : \mathbb{R}^{H\times W\times C} \to \mathbb{R}, \mathbf{x} \mapsto r$. We sample random white noise patterns $\mathbf{x}$ with image shape and then send the noise pattern through the neural network, and compute the gradient of f.

$$M_{raw} = \mathbb{E}_{\mathbf{x}\sim Unif[0,1]^{H\times W\times C}} \nabla_{\mathbf{x}} f(\mathbf{x}) \tag{1}$$

We averaged this gradient across 200 samples of $\mathbf{x}$ and then took the sum of squares over the channel C dimension as a $H \times W$ spatial mask. Finally, we fit this mask with a 2D Gaussian function, and the fitted mask is denoted as M_{fit}. This Gaussian mask was called the receptive field mask and was used to mask the prototypes and highlight the central features.

6.4 Methods for comparing image similarity

For comparing the image similarity across prototypes generated by various networks, we used this approach in the pixel space and the embedding space. In the pixel space, the images were directly used to compute the distance matrices. In the embedding space, the activations of ten pre-trained neural networks and the receptive field masks for the prototype's network's layer. The method to calculate the receptive field size and mask is mentioned in subsection 6.3. The image dataset of prototypes for one network had subdirectories for several layers spanning the network, each subdirectory corresponding to one layer in the network had images corresponding to each neuron/channel in that layer. Each of these subdirectories had a corresponding receptive field size and a receptive field mask. Each of these subdirectories containing images was sent into the ten pre-trained neural networks which were, ResNet50, ResNet101, ResNet152, InceptionV3, VGG16, VGG19, DenseNet121, DenseNet169, DenseNet201 and MobileNetV2. From here, the activations from the last fully connected layer of the network, like the 'avgpool' layer from ResNet50 were chosen for each network to extract the activations. These were then used to compute the distance matrices.

Note that, we used the receptive field mask to mask the prototype image, before computing their similarity. Because of this, the similarity or distance value might not be comparable across layers, as the units in different layers have different sizes of receptive field masks, thus the different masked prototypes will have different amounts of black backgrounds.

7 Extended Results

7.1 Linear Probe Evaluation of Learned Features

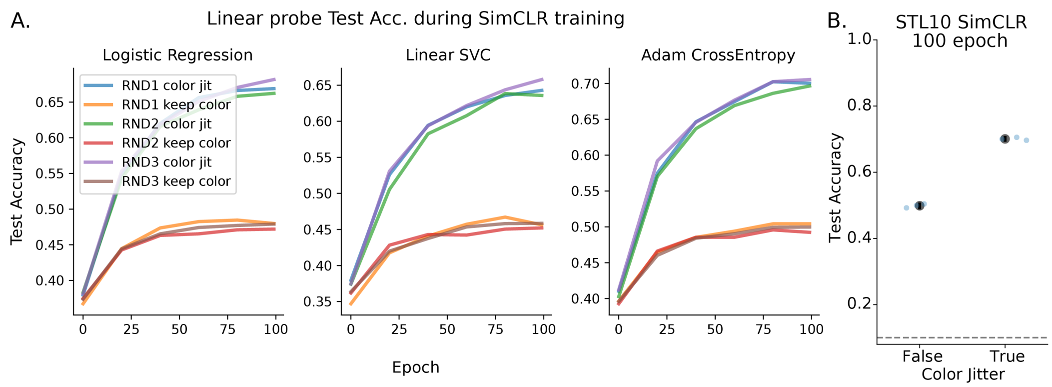


Figure 4: **Linear feature evaluation of representation during SimCLR training. A.** Test set accuracy during SimCLR training, Each panel corresponds to one way of fitting the linear readout layer: Logistic regression, Linear Support Vector Classifier, and gradient optimization (Adam) on Cross Entropy. **B.** Final test set accuracy for the self-supervised models, separated by whether they used color jittering augmentation.

7.2 Additional observations on the distance structure of prototypes

Aside from prototype diversity, we also examined the **rate of prototype change** during training: we computed the distances between the prototype of the same unit c during neighboring epochs Ep and $Ep + 1$. We averaged the distance for channels in each layer and showed it across epochs and networks (Fig.6). We can see, that all the networks experienced a transient peak at the first step, showing the drastic change of representation between randomly initialized network to network after one training epoch. Here we also saw clrjit networks experienced a higher rate of change of prototypes in layers 3 and 4, which might be the cause of their higher diversity.

Finally, we examined the **consistency of prototype across repeated Evolution**. This is related to the overall geometry of the landscape, i.e. how multimodal is tuning of the unit We computed the distances between the prototype of the same channel c for the two extractions. We averaged the distance for channels in each layer and showed it across epochs and networks (Fig.7). Generally, the distance between prototypes of the same channel (repeated evolution) is smaller than the distance between prototypes of different channels (cf. Fig.2). Intriguingly this distance between re-evolved prototypes is increasing through the training process, especially for the deeper layers of the models trained with color jittering. This highlights that for the same unit, repeated evolution led to increasingly different prototypes during training — thus the tuning functions of neurons are becoming more multimodal during training. This increase in multimodality may also benefit the final quality of representation.

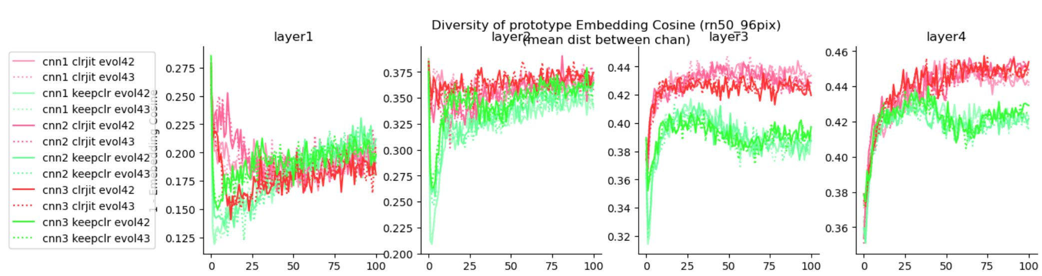


Figure 5: **Dynamics of prototypes diversity during training**. Cosine distance metric, resnet50 embedding

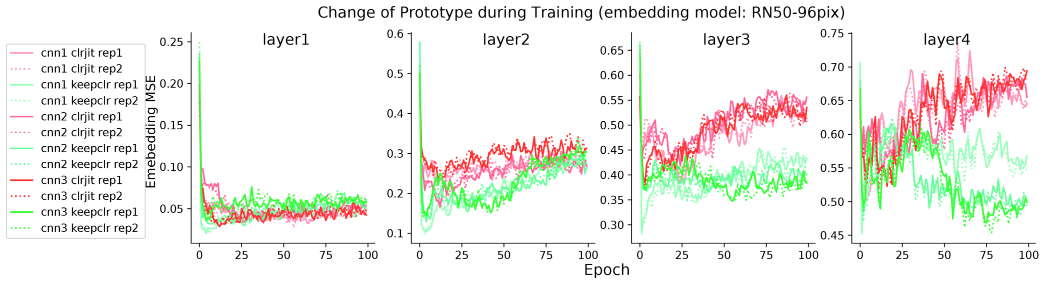


Figure 6: **Speed of prototype change during training**.

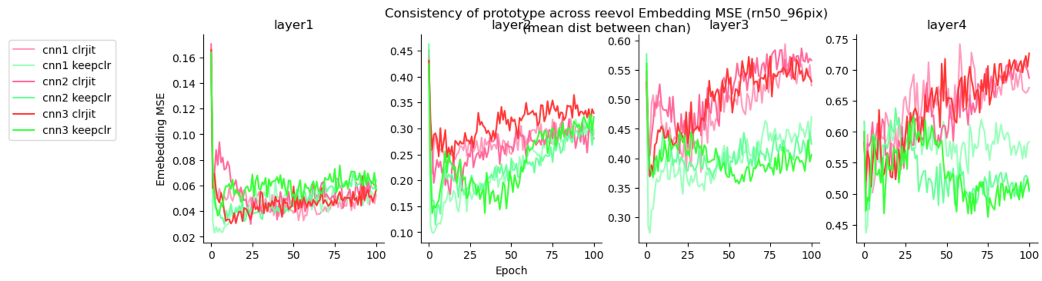
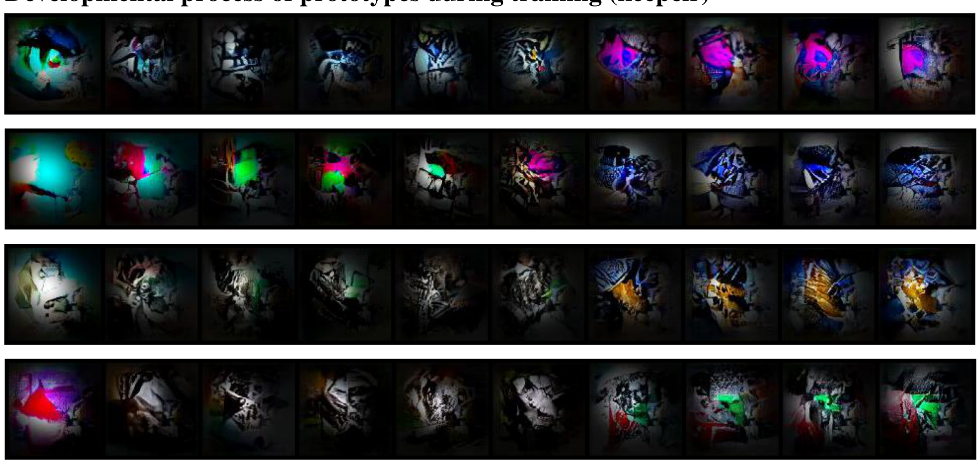


Figure 7: **Similarity of prototype between repeated evolution**.

7.3 Developmental process of prototypes during training (keepclr)



(a) (b) (c) (d)

Figure 8: Development of prototypes through training without color jittering (keepclr) condition, layer 3. Columns correspond to 0,10,20,... to 90 epoch; Rows correspond to Units 71, 93, 211, and, 248 (0-based index)