Spontaneous Decomposition from Grouped Network Pathways

Satchel Grant Matheus Dias Sahil Kulkarni
Stanford University
{grantsrb, mattdias, sahil1}@stanford.edu

Abstract

There have been many recent breakthroughs in self-supervised learning (SSL), i.e. unsupervised techniques used to obtain general purpose image features for downstream tasks. However, these methods often require large amounts of computational resources, and much is still unknown about how architectural choices affect the quality of self-supervised learned representations. There is still a lack of understanding of why compositional features spontaneously arise in previous self-supervised publications. In this work, we propose a class of models that is reminiscent of an ensemble. We show how this class of models can greatly reduce the number of parameters needed for learning robust representations in a self-supervised setting. Additionally, we show that sparsely connected network pathways spontaneously create decomposed representations.

1. Introduction

Recent work [3] [9] [1] has shown that it is possible to create robust image representations for downstream tasks without supervision. Such methods are desirable due to the high cost to create labelled datasets and the computational costs of training neural networks from scratch. Self-supervised methods offer another advantage in that they remove the biases that come from human tailored features. Automatically learned features are one of the great successes of deep learning [7]. Features learned automatically require less human expertise and can be far more detailed than humans could practically create [17].

In addition to the value of representations for downstream tasks, we were particularly interested in the spontaneous image segmentation that occurred in [1]. In that work, the authors used a vision transformer [5] in conjunction with a variant of the self-supervised training scheme from BYOL [9]. They found that the attention maps attended to interpretable features from the image. The attention maps showed that the model learned to semantically segment the images simply from self-supervised training. One may reasonably assume that this phenomenon is a result of the attention mechanism within the vision transformer architecture. In this work, we demonstrate that this assumption does not capture the whole story.

We were inspired by the performance and interpretability of these recent self-supervised methods. Much is still unknown, however, about why different architectures perform differently, and often these methods require immense computational resources. In this work, we seek to expand upon these methods to reduce computational complexity as well as understand the training dynamics that give rise to compositional learned representations. Taking inspiration from grouped convolutions, ensembles, and multi-head attention [20], we propose a class of models that improves the representational power of existing architectures while holding parameter counts constant. Our method also demonstrates the ability of grouped network pathways to spontaneously decompose signals from the input data in interpretable ways.

More specifically, we begin by utilizing the training scheme in BYOL [9], while using an ensemble of networks trained in conjunction - drawing inspiration from results in DINO [1]. We define a set of student learners—which we call leaf nodes—and a corresponding set of teacher leaf nodes. We then aggregate the outputs of the leaf nodes to create a single representation - one for the student and one for the teacher. We call the the resulting architecture a TreeNet due to the architecture's resemblance to pictures of computational trees. Finally, we use these representations to train the students in the conventional BYOL scheme.

We find that the separation of pathways results in greater interpretability of the models, allows for greater flexibility in model creation, and improves the models' latent representations for classification tasks.

1.1. Inputs and Outputs

At training time the student leaf nodes all receive the same augmented image v, while the teacher leaf nodes all receive a distinct augmented image of the same underlying image v'. The outputs are then two vectors corresponding to the student and teacher aggregates. At test time we pick the teacher encoder to perform evaluation or downstream tasks.

Here, the selected encoder receives an unaugmented image as input and outputs a single encoded representation.

2. Related Work

2.1. Self-Supervised Learning

Learning effective visual representations without human supervision is a long-standing problem. In general, unsupervised methods for representation learning can be categorized as either contrastive or non-contrastive. Here we describe their essential mechanisms and their respective state-of-the-art methods.

2.2. Contrastive Methods

Contrastive approaches learn to discriminate across representations by bringing representations of different views (augmentations) of the same image closer ('positive pairs'), and spreading representations of views from different images ('negative pairs') apart.

SimCLR is an example of a contrastive SSL method. In this paradigm, there is a single network used to generate the representations. The pipeline is as follows: 1. sample a batch $x^{(i)}$, $i = 1, ..., N$, 2. for each input $x^{(i)}$ sample and apply two transformations $t_i \sim \mathcal{T}$ and $t'_i \sim \mathcal{T}$, 3. generate $2N$ corresponding output representations z_i and z'_i, $i = 1, ..., N$, 4. backpropagate using the contrastive loss, $\mathcal{L} = \frac{1}{2N} \sum_{i=1}^{N} \{\ell(z_i, z'_i) + \ell(z'_i, z_i)\}$, where ℓ is the normalized temperature-scaled cross entropy loss. In other words, the loss encourages the network to make the representations of transformed inputs from the same image similar, and those from distinct images (i.e. negative samples) dissimilar.

2.3. Non-Contrastive Methods

Non-contrastive methods rely solely on positive samples, that is, on variations (i.e. augmentations) from the class to be predicted. Unlike contrastive learning, non-contrastive approaches seek to train a model capable of extracting similar feature vectors for similar images only by minimizing some metric between these two views of the same image [8]. Non-contrastive SSL methods tend to be less dependent on large batch sizes and memory queues. Therefore, they are generally more efficient and conceptually simple while maintaining state-of-the-art performance [18].

Some examples of non-constrastive methods include DINO [1], BYOL [9], SimSiam [4] and MultiSiam [2]. The biggest hurdle to such methods is the risk of what is known as network collapse, that is, of having the network output a constant solution where all inputs map to the same output. To avoid this, most of these methods use a stop-gradient operation.

In BYOL, two networks of the same architecture but distinct random initializations are defined, the student network f and teacher network g. The pipeline is as follows: 1. Produce $2N$ transformed images as in steps 1,2 above. 2. generate $2N$ corresponding output representations, where z_i is produced by the student network and z'_i by the teacher network. 3. backpropogate only on the student network using the ℓ_2 loss between z_i and z'_i, i.e. $\mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} ||z_i - z'_i||_2^2$. 4. update the teacher weights as an exponential moving average of the student weights, $g_W \leftarrow \tau g_W + (1 - \tau) f_W$.

DINO follows largely the same BYOL training paradigm. The primary distinctions are 1. the student and teacher networks are now Vision Transformers rather than convolutional networks, and 2. the output representations are put through a softmax layer which are then used to compute cross entropy loss rather than ℓ_2.

Comparing each of SimCLR, BYOL and DINO, we note SimCLR requires relatively the largest batch sizes and compute. BYOL is less computationally expensive and less dependent on large batch sizes than SimCLR. Finally, DINO is requires the least amount of compute out of all three, however generally requires longer pre-training (transformers in general do). Moreover, DINO offers an implicit compositionality of the learning of each of its attention heads allowing for better interpretation of the learned representations.

Knowledge Distillation The student/teacher networks used in BYOL were inspired by the target networks developed by Deep Reinforcement Learning researchers to stabilize the bootstrapping updates provided by the Bellman equation [14]. While most RL methods use fixed target networks, BYOL uses a weighted moving average of previous networks (momentum encoder) in order to provide smoother changes in the target representation.
This approach of improving the quality of features by propagating a small initial set of soft annotations to a large set of unlabeled instances is also known as knowledge distillation. DINO [1] extends this concept to the scenario with no labels. While combining self-supervised learning and knowledge distillation had already been done [6], DINO innovates in dynamically building the teacher during training.

2.4. Grouped Convolutions

Grouped convolutions (or Filter Groups) are convolution operations in which the model's filters are separated into groups, and each of these filter groups is convolved with only part of the previous layer's feature maps. Moreover, as long as these pathways are not interconnected, group convolution creates a deep network with some number of layers and then replicate it so that there is more than one pathway for convolutions on a single image.

Grouped convolutions not only drastically reduce the number of computations by allowing for data and model parallelization, but also help the model learn better representations of the input [11] [13][17].

Our models apply the principle of group convolution in

the sense that each of our leaves is restricted to convolving to a separate subset of the feature map from the previous layer.

3. Methods

We begin by defining a set of transformations $\mathcal{T}$, a common CNN architecture, a number of leaves m, an aggregation function $A(\cdot)$, and a loss function $\mathcal{L}$. Then, using the common CNN architecture, we create m student networks f_j, i.e. student leaves, and m teacher networks g_j, i.e. teacher leaves. Each student has a distinct random initialization. The teacher leaf nodes are initialized as a one-to-one copy of the student leaf nodes. We now describe our training procedure for a single image:

1. Sample an image $x \sim \mathcal{X}$
2. Apply a transformation $t \sim \mathcal{T}$ and $t' \sim \mathcal{T}$ resulting in transformed images $v := t(x)$ and $v' := t'(x)$, respectively.
3. For each student leaf $f_j \in \{f_1, ..., f_m\}$ input v to generate its corresponding output representation $z_j := f_j(v)$.
4. Apply the aggregation function over the students' outputs to generate an aggregate student output representation $Z := A(z_1, ..., z_m)$. As is common in the literature, we call this student pipeline our encoder.
5. Follow the same procedure for the teacher leaf nodes, resulting in a final teacher output $Z' := A(z'_1, ..., z'_m)$
6. Backpropagate only on the student encoder using the loss function $\mathcal{L}$
7. For each teacher leaf $g_j \in \{g_1, ..., g_m\}$ update its weights as an exponential moving average of the corresponding the student leaf's weights, i.e. $g_{jW} \leftarrow \tau g_{jW} + (1 - \tau) f_{jW}$.

The aggregation function can take many forms. For example, it may be a simple average of leaf outputs, or a more complex function with learnable parameters, e.g. a feed forward neural network. The loss function may also take a number of forms. From 6 and 7 it is clear that the training procedure follows that of BYOL, so naturally one may use ℓ_2 or cross entropy loss. However, we also note that the choice of aggregation function may allow us to train analogously to SimCLR. For example, suppose we choose $A(\cdot)$ to be the concatenation of leaf outputs. Then we may choose the loss function to resemble the contrastive loss - where we seek to maximize z_j, z'_j similarity and minimize otherwise. This contrasts SimCLR in that we now use generated negative samples from the same image, rather than sampled negative samples from distinct images. This would in effect attempt to not only bypass the need for large batch sizes, but also explicitly encourage student leaf nodes to learn diverse representations. We hypothesize that the latter already occurs implicitly in the case of using ℓ_2 or cross entropy loss.

3.1. Evaluation Techniques

In the self-supervised setting we do not use labels during training, so it is not immediately obvious how to evaluate the quality of the learned representations. We focus on image classification as our evaluation task. There are a two primary techniques commonly used to evaluate the models' representations.

The first is to train a linear mapping from the model's representations to the corresponding output labels. In this scheme we freeze the weights of the encoder, collect the features from an inner layer, and train a linear classifier to use these features to predict labels.

The second technique is to simply take a number of training examples, store their corresponding encoder output and label pairs, then evaluate new examples using a k-nearest neighbor classification.

In both of these evaluation settings, we explore different hidden layers of the models and select the best performing layer from each model. We also explore using a concatenation of the best subset of leaf-CNN output vectors. The best subset is selected based on individual classification accuracy.

3.2. Tested Architectures

3.2.1 Leaf Architectures

Here we list the architectures of the leaf networks used in this paper. We considered and explored many architectures for the leaf nodes in the initial stages of this work. The two most important characteristics that informed our final choices were that we wanted to maintain the models' ability to process any size input, and we wanted the trainings to take a reasonable amount of time. Each leaf node can take the form of any existing model type. Additionally, the convolutional features of each leaf node can be aggregated together in much the same way that we aggregate the leaves together later in the model. We explored global max pooling, global average pooling, and a form of recurrent attention. We settled on global average pooling as it sped up training duration, did not significantly harm training performance, and was capable of processing variable input sizes.

Each leaf architecture consists of a series of convolutions with depths 8, 16, 32, 48, and 64, each followed by a layer normalization and a ReLU activation function. After the convolutions, the feature map is averaged across spatial dimensions and is then processed by a single linear projection layer. These outputs are then aggregated in the aggregation function.

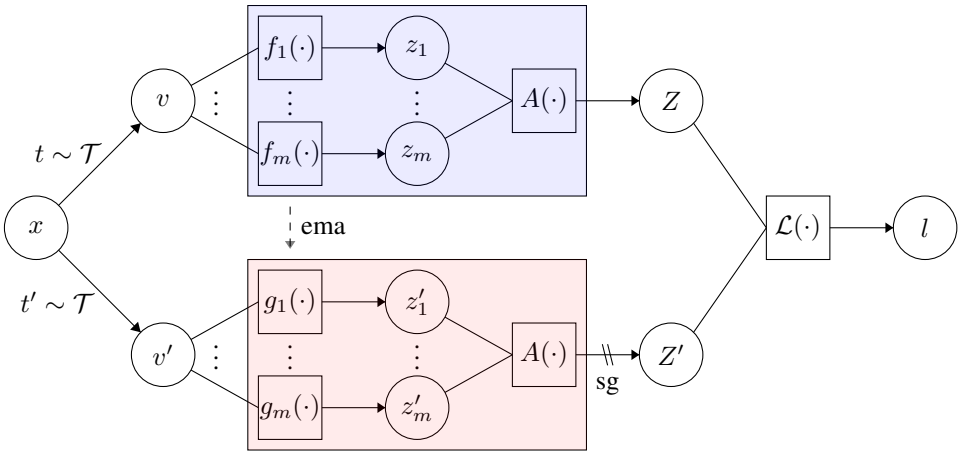

Figure 1. TreeNet Architecture. Teacher leaves, g_j's, are updated using an exponential moving average (ema) and sg means stop-gradient. The student and teacher encoders are shown in blue and red, respectively. The naming "TreeNet" is then apparent from the colored regions.

3.2.2 Aggregation Methods

We tested multiple ways to combine the outputs of each of the leaves into an aggregated representation.

1. **Average Join**: We initially explored a simple average of the outputs of each of the leaf nodes. This approach had the advantage of fewer parameters per leaf which enabled us to try as much as 128 leaves in a single model. This method, however, lacks explicit informed control over leaf node representations.

2. **Dense Join**: In this method we concatenated each of the outputs from the leaf nodes and processed this vector through a 3 layer densely connected neural network. This method has improved control over its use of the individual leaf nodes.

3. **Attentional Join**: This method applied a single layer of multi-head attention over the outputs of the leaves. We expected that this would be the best method as it allows for variable numbers of leaves and enables a degree of flexibility over how to process each of the leaves' features. We found that it did not perform as well as the Dense Join, but we believe this is likely due to the small size of the CIFAR10 dataset.

4. **Max Pool Join**: We explored taking the maximum value over the outputs of each leaf node. The results of these models were significantly worse than the other listed Join methods, so we did not pursue this join method further than a few preliminary models.

3.2.3 Baseline Architectures

TreeNets can incorporate any existing model architecture. Thus we selected baseline architectures from the same architecture class as the leaf nodes. We expanded the number of channels in the baseline CNNs to control for increasing numbers of parameters without changing architecture depth. We chose each baseline CNN model to be structurally equivalent to a DenseJoin TreeNet with a single leaf node.

3.3. Loss Functions

For all of our trainings we used the DINO loss function that seeks to minimize the cross entropy between a softmax distribution over the student outputs and a softmax over the teacher outputs. This loss is formalized as:

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} H(P_t(x_i), P_s(x'_i))$$

Where $H(P_t(x), P_s(x'))$ is the cross entropy between the teacher and student distributions P_t and P_s respectively. x and x' are different augmenations of the same data sample i from the training batch with N samples.

We feel it is worth noting, however, that our proposed class of models are not limited to the DINO loss function. We expect that many self-supervised methods would benefit from our proposed architectural changes.

We wanted to explore ways to encourage the leaf networks to learn diverse features. We settled on adding an additional loss term to minimize the cosine similarity between each of the leaf networks in select trainings. Using the outputs from each of the K leaf nodes $f_i(x)$ we derive the following loss:

$$\mathcal{L}_{cos} = \frac{1}{2K^2} \sum_{i=1}^{K} \sum_{j=1}^{K} \mathbf{1}_{i \neq j} \frac{f_i(x) \cdot f_j(x)}{\|f_i(x)\| \|f_j(x)\|}$$

$$\mathcal{L} = \beta \mathcal{L}_{cos} + \mathcal{L}_{DINO}$$

Figure 2. Examples of image augmentations performed.

Where β is a scaling term to balance the two losses. Cosine similarity ignores magnitude, thus, the leaf nodes would be unable to solve this additional loss by a simple scaling of their output.

3.4. Additional Implementation Details

To complete this project, we started from the official DINO github repository implemented in PyTorch. To build our model architectures most efficiently, we used grouped convolutions and grouped normalizations to implement the leaf nodes. This enabled up to 20x faster training speeds. We used the default hyperparameters from the existing DINO code base in all reported trainings. We slightly explored changing the image augmentations but quickly found that we did not have enough time/compute to properly explore these parameters.

4. Data

We use the CIFAR-10 dataset [12] with the standard 50,000-10,000 train/test split as our baseline upon which the transformations are applied.

4.1. Pre-Processing Pipeline

Previous work has demonstrated the importance of the types of transformations used. In particular, it has been shown that the choice of transformations used has a significant effect on the quality of learned representations and thereby performance in downstream tasks [3]. As a result, we adopt the same set of transformations that state-of-the-art methods use. Specifically, this set includes flip and color jitter, normalization, and global and local cropping.

The transformations are performed at training time. For each image $x^{(i)}$ in a batch, each leaf j samples a transformation $t_{ij} \sim \mathcal{T}$. The corresponding input is then $t_{ij}(x^{(i)})$. During evaluation, however, we are interested in the performance on downstream tasks, therefore transformations are not used.

One limitation of using CIFAR-10 is that while some augmentations maintain the semantic information in the images, many times this is not the case - as seen in the second image of Figure 2.

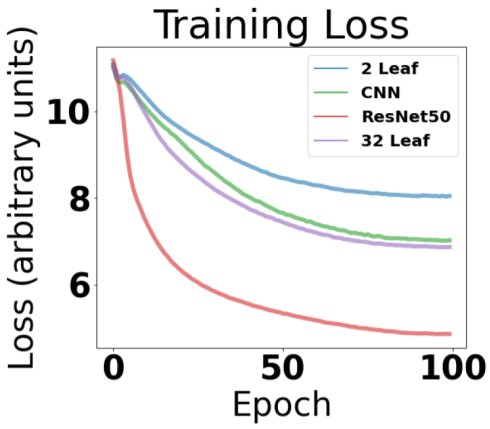

Figure 3. Loss curves over DINO training

5. Experiments

We now describe our current implementation details used to generate the results ahead.

5.1. Hyperparameters

As alluded to before, we adopt the same set of transformations from [1]. For the common CNN architecture we use 5 convolutional layers each with 2x2 kernel size, 0 padding, and a stride of 1. We chose a 2x2 kernel size for reduced model size. After the first layer, each convolution is preceded by a Layer Normalization and followed by a ReLU layer. After the 5 convolutional layers, the features are averaged across the spatial dimensions and processed by a Layer Norm, dense layer, ReLU, and a final dense layer. We used pytorch's Adam optimizer, with batch sizes of 384, and a starting learning rate of 5e-4 with a linear warm-up and a cosine learning rate schedule with a minimum of 1e-6. These were chosen based on DINO's defaults.

5.2. Quantitative Evaluation

To evaluate the quality of our self-supervised encoder, we use both a k-NN evaluation and a linear classification evaluation on the latent representations using the default train-test split of CIFAR-10. We use top-1 accuracy for both metrics and compare our model performance against relevant CNN models and ResNet50. As seen in Figure 4, our models achieve a higher total accuracy in both the k-NN and linear classification metrics. Our 48-Leaf model achieved a k-NN accuracy of 65.5% and a linear accuracy of 70.2% compared to the best performing baseline of 55.1% and 62.0% and ResNet50's performance of 34.2% and 61.0% respectively.

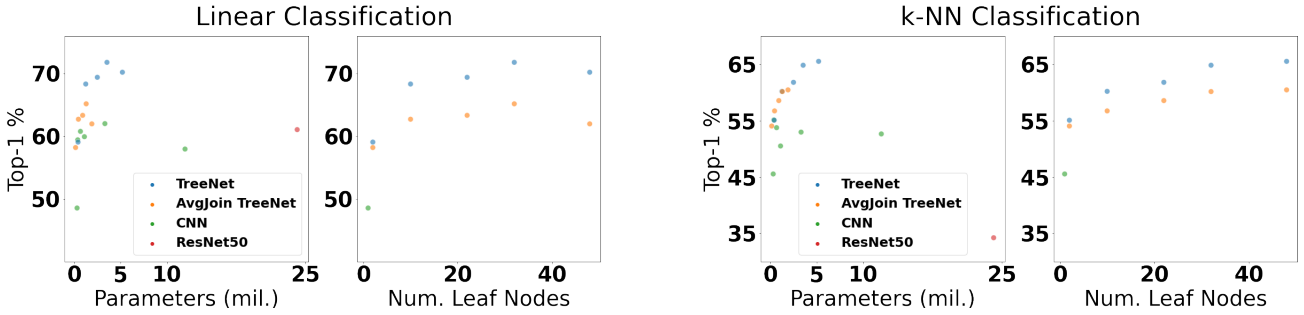

Figure 4. Linear and k-NN classification accuracy on CIFAR10 using latent features of each model. The blue marker TreeNet models use the DenseJoin aggregation method. Each TreeNet variant only changes in its number of leaf nodes whereas each CNN has progressively larger numbers of channels at each convolutional layer. Each right-side panel shows the number of leaf nodes for each of the TreeNets plotted in each of the left-side panels. Note the brown-looking data point in the lower left panel is caused by an overlay of blue and orange data points.

Type	Conv Channels	N Leaf	Accuracy
DenseJoin	8 - 16 - 32 - 48 - 64	32	**71.8**
AvgJoin	8 - 16 - 32 - 48 - 64	32	**65.2**
CNN	32-128-256-512-1028	1	62.0
ResNet50			61.1

Table 1. Linear evaluation accuracies of the best performing model of each type in Figure 4

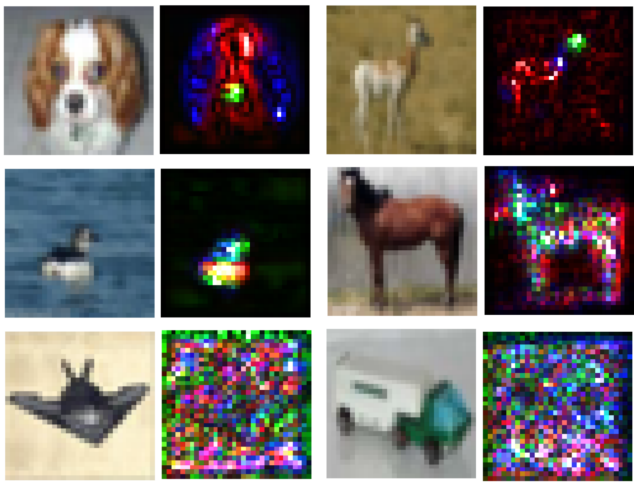

Figure 5. Overlayed saliency maps, each containing three unique leaf nodes - loosely categorized as specialists, generalists, and failed learners.

5.3. Qualitative Evaluation

5.3.1 Saliency Maps

Here we seek to further justify the motivation for our proposed models. Particularly, we use saliency maps as a method to evaluate our hypothesis that grouped network pathways lead to leaf nodes learning compositional feature representations, thereby allowing for a more robust performance in downstream tasks. To do so we present a number of overlayed saliency maps, each containing three unique leaf nodes (corresponding to red, green, and blue), of a common non-transformed image.

Throughout our experiments we observe that leaf nodes appear to fall into three broad categories, which we denote "specialists", "generalists", and "failed learners" - as shown in Figure 5. Specialist leaf nodes appear to encode specific semantic features in the image. For example, in the top-right image in Figure 5, three leaf nodes appear to focus the deer's body, neck, and head respectively. In contrast, generalist leaf nodes appear to encode the entire object of interest, as seen in the second row of Figure 5. Finally, we note that depending on the architecture, there are a number of leaf nodes that fail to learn meaningful or interpretable features, as seen in the third row of Figure 5. To expand on these three insights, we note that there are often more generalists when the number of leaf nodes is small, e.g. $<$ 30. Conversely, there are often more specialists and failed learners as the number of leaf nodes increases.

Finally, we note that specialist nodes often fall into two subcategories, which we denote "inconsistent" and "consistent". Inconsistent nodes appear to encode distinct, specific semantic features within the same context, i.e. class, while consistent nodes appear to attempt to encode the same semantic features - as seen in Figure 7.

5.3.2 Representation Visualizations

To further understand the behavior of our models, we perform two dimensionality reduction methods to visualize the topological structure of learned representations. Specifically, we use the Uniform Manifold Approximation and Projection (UMAP) [15] and t-Distributed Stochastic Neighbor Embedding (t-SNE) [19] methods to visualize the representations of models utilizing the attentional join, average join, dense join, and dense join with cosine similarity loss, on the CIFAR-10 test set. We note a few key obser-

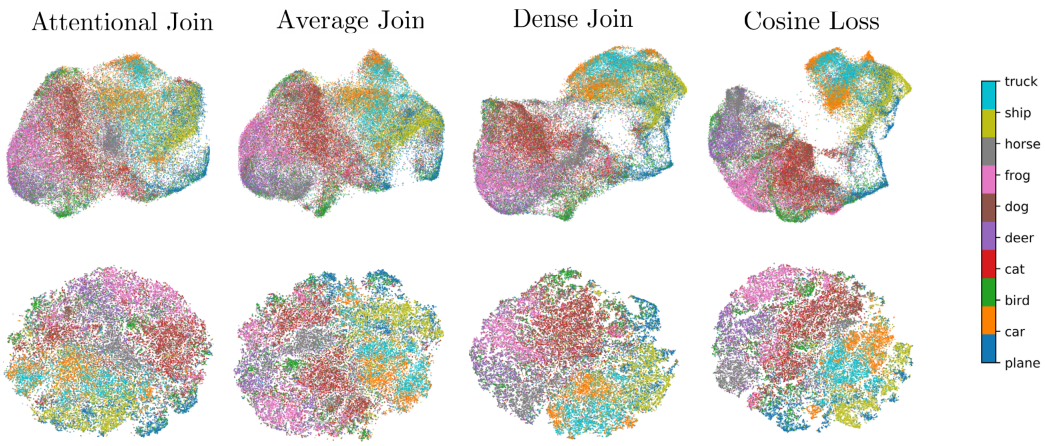

Figure 6. UMAP (first row) and t-SNE (second row) visualization of image representations by architecture.

vations. First, from the UMAP plots we see that the dense join and dense join with cosine loss model representations appear to be more well separated than those of the attentional join and average join models. The former is consistent with the our quantitative evaluation results, however the latter appears to contrast them. Next, it appears that across model architectures, representations for vehicles (car, truck, plane, ship) appear all relatively closer than the representations for animals (horse, frog, dog, deer, cat, bird). Finally, we note that the cat and dog class representations are almost always overlapping, regardless of model architecture or dimensionality reduction method. This may be due to their similar visual characteristics after augmentations.

5.4. Architecture Discussions

We explored a number of architectures during the initial stages of the project. Here we limit our discussion to the architectures presented in the in the methods.

We achieved a positive result in demonstrating the power of leaf nodes as a tool for feature representation. We can clearly see the performance improvements that come from the TreeNet architectures in Figure 4. Furthermore, the saliency maps and SHAP plots demonstrate the interpretability of each of the leaf nodes. This demonstrates a naturally occurring decomposition of the data into each of the parallel pathways.

From Figure 4 we can see that the DenseJoin aggregation method produced more useful representations than the AverageJoin method. We believe this can be explained by the ability of the DenseJoin method to intelligently integrate each of the leaf nodes based on information from other leaf nodes. The AverageJoin method has no ability to synthesize information from multiple leaf nodes in an intelligent way. The MaxJoin method performed poorly in early explorations, so we did not pursue further analysis. We regret that we did not have the time and resources required to explore the AttentionalJoin method more for this project. We expect our relatively low preliminary results of 57% linearly classification accuracy were due to insufficient attentional layers and the relatively small amount of data in the CIFAR10 dataset.

The performance of the ResNet50 architecture was particularly interesting to us in that the training loss was clearly the lowest, and the model likely has the greatest potential given its parameter count and performance in other papers/settings [9, 3, 10], but performs relatively poorly in our evaluation methods. We noticed that the features from earlier layers were better for linear classification. We chose the best layer for the reported results which in the end was only the 11th out of 50 total layers. The final layer had a 10% linear classification accuracy which is equivalent to guessing. We believe this poor performance has to do with the ResNet overfitting to the dataset. We suspect that the model managed to create such divergent pathways for each of the data samples that the features lost much of their general purpose use. Even looking at the training accuracy from the linear evaluation on the ResNet50 at the best layer, it only achieved 71.7% accuracy.

We believe it is important to note the fact that although TreeNets may have the same number of trainable parameters as a baseline CNN, TreeNets can have a larger number of total activation units. This occurs due to the nonlinear increase in number of parameters for a linear increase in the number of channels in a convolutional filter. This increase in the number of activation units may help explain the greater expressivity of the TreeNet models.

5.5. Cosine Similarity Loss

Models trained using the inter-leaf node cosine similarity loss function consistently performed worse on both the k-NN and linear classification metrics compared to models trained without the cosine similarity term. The best performing cosine similarity model had a linear classification accuracy of 51.2%.

Qualitatively, we noticed that the saliency maps of the models using the cosine loss term appeared more sparse. The focus of the saliency maps, however, did not appear to be interpretable. Additionally, the classes within the t-SNE and UMAP plots appeared meaningfully and consistently separated between models trained with unique seeds. So, we were surprised that the linear evaluation performed so poorly. We believe the reason that the cosine similarity loss term caused worse performance is ultimately due to the fact that satisfying the cosine loss term does not require semantically meaningful representations. We believe that the cosine loss term actively harms the semantics of the representations learned by the self-supervised training.

6. Conclusion

In this project, we proposed an novel Self-Supervised Learning method expanding on recent results, while reducing compute requirements. We explored different architectures, parameters and loss functions and reported our results on the CIFAR-10 dataset using two different evaluation techniques. Our results indicate superior Top-1 Accuracy for this dataset when compared to reasonable baselines. Finally, we used qualitative methods and explainability techniques to shed light on the inner workings of the learning process proposed. From these methods, we showed the spontaneously occurring phenomenon of decomposition arising simply from imposing an inductive bias of disconnected pathways within the model.

As next steps, we believe there is space for further architectural improvement on our model. First, our model lacks translational granularity over the input data. This could be limiting the learned representations of the model, especially for images in which translational variance is important. Utilizing some form of attention mechanism or separating the information that is given to each leaf according to some form of translational ordering could help solve this problem.

Moreover, there is room for improvement in our TreeNet architecture; according to preliminary tests we conducted, using ResNet leaf nodes would likely help our models learn more smoothly.

Finally, we would be interested in training our model with larger datasets such as ImageNet so we could compare our results to more comparable baselines, such as those reported in DINO and BYOL.

7. Appendix

7.1. Additional Saliency Plots

Figure 7 depicts the saliency maps of consistent specialist leaf nodes. In the first row, for example, three leaf nodes appear to consistently look for the head, body, and background of birds. In the second row, there are three unique leaf nodes again similarly encoding consistent semantic information, namely the body, saddle/rider, and background of a horse. We note that in this case however, the leaf node still encodes this information even though there may not be a saddle/rider present. Finally, the third row depicts that these leaf nodes need not be consistent within a specific class. As we see the first two images correspond to cars, whereas the third image corresponds to a truck - yet the same semantic features are being encoded consistently.

7.2. DeepSHAP

In this section we present an explainability method developed by Shrikumar et al. [16]. Our goal is to bring in a different method based on gradient integration to complement the Saliency Maps and to have a view over the model's representations as a whole (not only focused on each leaf). DeepShap decomposes the output prediction of a neural network on a specific input by backpropagating the contributions of all neurons in the network to every feature of the input. In that sense, it computes importance scores based on explaining the difference of the output from a 'reference' output in terms of differences of the inputs from their 'reference' inputs. We applied this method to the CIFAR-10 dataset using our 2 leaf model.

In the figure above, the left image shows the original image from CIFAR-10 and the right image shows the image's pixels colored in red (increase the probability of predicting the image to be of the correct class) and blue (decreased the probability of predicting the image to be from the correct class). We were not sure how to interpret the results. We would like to understand why we got this result, but due to computational and temporal constraints, we leave this to future work.

8. Contributions Acknowledgements

We'd like to thank Stanford's PDP Lab for allowing us to use their nodes on the CCN cluster for trainings.

We'd like to thank the wonderful TA's for all their hard work and the instructors for their wonderful lectures.

Sahil created the saliency maps and conducted an analysis of their behavior with respect to model architecture changes. He also conducted the analysis of learned representations using the dimensionality reduction methods.

Matheus conducted literature review, worked on the saliency maps and DeepSHAP explainability method.

Figure 7. Saliency maps of consistent specialist nodes.

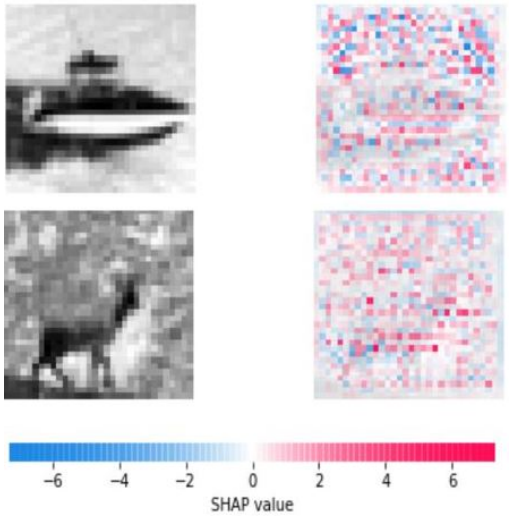

Figure 8. SHAP Plots for CIFAR-10 images combining all leaf nodes

Satchel did the model trainings as he had access to the CCN cluster. He also created the loss and accuracy plots.

All authors contributed equally to the writing of the paper.

References

[1] M. Caron, H. Touvron, I. Misra, H. Jégou, J. Mairal, P. Bojanowski, and A. Joulin. Emerging properties in self-supervised vision transformers, 2021. 1, 2, 5

[2] K. Chen, L. Hong, H. Xu, Z. Li, and D.-Y. Yeung. Multisiam: Self-supervised multi-instance siamese representation learning for autonomous driving, 2021. 2

[3] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton. A simple framework for contrastive learning of visual representations, 2020. 1, 5, 7

[4] X. Chen and K. He. Exploring simple siamese representation learning, 2020. 2

[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, 2020. 1

[6] Z. Fang, J. Wang, L. Wang, L. Zhang, Y. Yang, and Z. Liu. Seed: Self-supervised distillation for visual representation, 2021. 2

[7] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org. 1

[8] A. Gopani. Contrastive vs non-contrastive self-supervised learning techniquesnbsp;, Mar 2022. 2

[9] J.-B. Grill, F. Strub, F. Altché, C. Tallec, P. H. Richemond, E. Buchatskaya, C. Doersch, B. A. Pires, Z. D. Guo, M. G. Azar, B. Piot, K. Kavukcuoglu, R. Munos, and M. Valko. Bootstrap your own latent: A new approach to self-supervised learning, 2020. 1, 2, 7

[10] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition, 2015. 7

[11] Y. Ioannou. A tutorial on filter groups (grouped convolution), Aug 2017. 2

[12] A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009. 5

[13] A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. *Neural Information Processing Systems*, 25, 01 2012. 2

[14] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning, 2015. 2

[15] L. McInnes, J. Healy, and J. Melville. Umap: Uniform manifold approximation and projection for dimension reduction, 2018. 6

[16] A. Shrikumar, P. Greenside, and A. Kundaje. Learning important features through propagating activation differences. In D. Precup and Y. W. Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 3145–3153. PMLR, 06–11 Aug 2017. 8

[17] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions, 2014. 1, 2

[18] Y. Tian, X. Chen, and S. Ganguli. Understanding self-supervised learning dynamics without contrastive pairs, 2021. 2

[19] L. van der Maaten and G. Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008. 6

[20] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need, 2017. 1