Final Project Report
Exploring the Efficacy of Self-Supervised Learning: A Deep Dive into the DINO Algorithm

Atharv Ramesh Nair EE20BTECH11006 | Nithish S EE20BTECH11044 | Ajit Shankar ES20BTECH11003 | Theresa Karra EE20BTECH11050

1. Introduction

In recent years, the field of deep learning has seen remarkable advancements, particularly within computer vision. One emerging approach that stands out is self-supervised learning, a paradigm that eliminates the need for labeled data, allowing models to benefit from the abundance of available unlabeled data.

Our investigation takes inspiration primarily from the work of [1] on the DINO (Self Distillation with No Labels) algorithm. The promising outcomes exhibited by DINO [1] and its subsequent version, DINO V2 [2], are noteworthy. Significantly, models trained using DINO and DiNO V2 have demonstrated competitive performance across several downstream tasks, such as image classification, segmentation, depth estimation, and instance retrieval. The DiNo approach stands in contrast to prior contrastive learning methods, which necessitated the identification of positive and negative samples for any given image. Our primary objective is to explore the capabilities of these self-supervised methods, contrasting them against traditional supervised techniques to comprehend their strengths and potential limitations. We aim to compare the differences between these self-supervised techniques on partially labeled datasets. Taking into account the computational constraints, we decided to test out this approach on the smaller imagenette dataset from FastAI [3] [1]

The rest of this report is as follows: Section 2 briefly explains the architecture of the Vision Transformer [4] which has been extensively used by the DiNo [1] to produce competitive results. Section 3 provides an overview of some previously used self-supervised methods and Section 4 takes a deeper dive into Boot Stap Your Latent (BYOL) [5] which is the predecessor of DiNo. In Section 5, the DINO algorithm is discussed comprehensively. In section 6, we present results produced using the DiNo algorithm on the imagenette dataset. In section 7, we show some results on downstream tasks using the trained model provided by [1]. Finally, Section 8 discusses future prospects. The code used to produce the results shown in this report is made available here [2].

[1] https://github.com/fastai/imagenette/

[2] https://github.com/AtharvRN/EE6380_Project

2. Vision Transformers

Vision Transformers (ViTs) [4] apply the transformer architecture, originally designed for Natural Language Processing tasks [6], to vision tasks. Instead of relying on convolutional layers, ViTs capitalize on the self-attention mechanism. Convolutional Neural Networks (CNNs) have been the mainstay in computer vision for some time since they exploit local spatial hierarchies through convolutional layers. On the contrary, Transformers are able to capture both local and global dependencies.

ViT Components:

- **Image Patching:** Images are divided into fixed-size patches, like 16×16 or 8×8, which once flattened become tokens, analogous to word embeddings in NLP.

- **Linear Projection and Positional Embedding:** The patches undergo linear embedding into a higher-dimensional space. A Class Token is added to the start, followed by Positional Embeddings for spatial context. Both the class token and positional embeddings are trainable.

- **Transformer Encoder:** Leveraging the transformer encoder from [6], it consists of alternate layers of multi-headed self-attention mechanisms and MLP blocks. Before and after each block, Layer normalization and residual connections are applied, respectively.

$$z_0 = [x_{\text{class}}; x_1 E; x_2 E; \ldots; x_{N-1} E] + E_{\text{pos}} \quad (1)$$

Z_o is the input to the encoder

$$z'_\ell = \text{MSA}(\text{LN}(z_{\ell-1})) + z_{\ell-1}, \quad \ell = 1, \ldots, L \quad (2)$$
$$z_\ell = \text{MLP}(\text{LN}(z'_\ell)) + z'_\ell, \quad \ell = 1, \ldots, L \quad (3)$$
$$y = \text{LN}(z^0_L) \quad (4)$$

- **Classification Head:** The output corresponding to the Class Token (y) is extracted post-transformer and passed to an MLP Layer for the final classification.

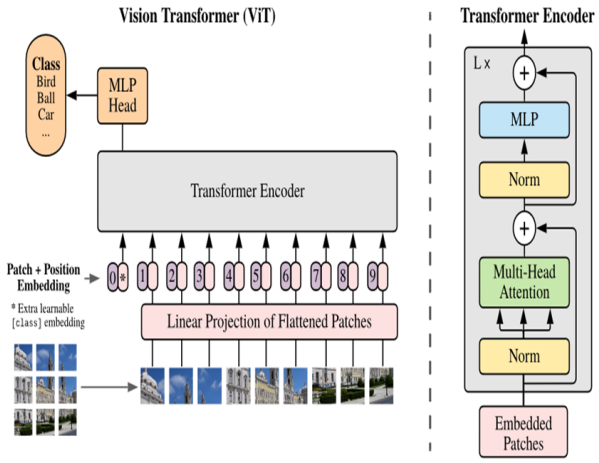


Figure 1. ViT Model Architecture

3. Self Supervised Learning

As stated earlier, self-supervised learning (SSL) [7] is an emerging field in machine learning that learns representations of data from unlabelled data, which gives a significant advantage over the supervised learning approaches that require labeled data for training, which isn't available handful these days. In the field of computer vision, SSL has led to the development of various methods like Bootstrap Your Own Latent (BYOL) and DINO, which learn all-purpose features of images that help in performing various downstream tasks efficiently. While constructing these all-purpose features, many methods have been developed and can be categorized under four categories respectively – the deep metric learning family, the self-distillation family, the canonical correlation analysis, and the masked image modeling family. One such popular method **SimCLR** [8] learns visual representations by encouraging similarity between two augmented views of an image, instead of using sampling techniques and then checking for similarity in the sampled images, which was done before SSL was introduced. **VICReg** [9], balances three objectives based on covariance matrix representations from two views: variance, invariance, and covariance. Regularizing variance along each dimension of the representation prevents collapse, the invariance ensures two views are encoded similarly, and the co-variance encourages different dimensions of the representation to capture different features. **MoCo** (Momentum Contrast) [10] uses a contrastive loss function, where the model is trained to maximize the similarity between positive pairs, i.e., different views of the same image and minimize the similarity between negative pairs, i.e., different images, where the negative pairs are sampled from a queue of negative examples, which is updated using a momentum encoder.

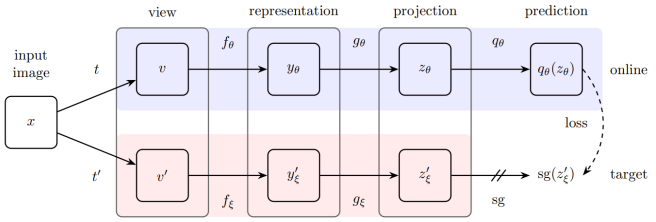


Figure 2: BYOL's architecture. BYOL minimizes a similarity loss between $q_\theta(z_\theta)$ and $\mathrm{sg}(z'_\xi)$, where θ are the trained weights, ξ are an exponential moving average of θ and sg means stop-gradient. At the end of training, everything but f_θ is discarded, and y_θ is used as the image representation.

Figure 2. BYOL Architecture

4. Bootstrap Your Own Latent

Unlike the state-of-art contrastive methods that are primarily built on reducing the distance between 'positive pairs' and increasing the distance between the 'negative pairs', a new self-supervised learning algorithm, called BYOL (Bootstrap Your Own Latent) introduced by Grill et al., [5] relies on two neural networks, referred to as online and target networks, that interact and learn from each other, to produce results comparable to the state-of-art performance on learning image representations, without using negative pairs. The online network is used to predict the target network representation of the same image under different augmented views. At the same time, the target network is updated with a slow-moving average of the online network. Using a combination of online network and target network helps the BYOL algorithm to avoid collapsing to trivial solutions, i.e., outputting the same vector for all images. Also, BYOL suffers a much smaller performance drop than SimCLR, when only using random crops for image segmentation.

The authors claim that "The core motivation for BYOL: from a given representation, referred to as target, we can train a new potentially enhanced sequence of representation, referred to as online, by predicting the target representation. From there, we can expect to build a sequence of representations of increasing quality by iterating this procedure, using subsequent online networks as new targets for further pretraining." By using a slow-moving average over the outputs of the online networks as the target network, the representations are refined.

The online network is defined by a set of weights θ consisting of three stages: an encoder f_θ, a projector g_θ, and a predictor q_θ. The target network, whose parameters are ξ are an exponential moving average of the online parameters θ and are updated as follows, given a decay rate $\tau \in [0, 1]$,

$$\xi \leftarrow \tau\xi + (1 - \tau)\theta.$$

Given a set of images $\mathcal{D}$, an image $x \sim \mathcal{D}$ sampled uniformly from $\mathcal{D}$ and two distributions of image augmentations $\mathcal{T}$ and $\mathcal{T}'$, two augmented views of the image $v \triangleq t(x)$

Algorithm 1: BYOL: **B**ootstrap **Y**our **O**wn **L**atent

Inputs :

$\mathcal{D}$, $\mathcal{T}$, and $\mathcal{T}'$	set of images and distributions of transformations
θ, f_θ, g_θ, and q_θ	initial online parameters, encoder, projector, and predictor
ξ, f_ξ, g_ξ	initial target parameters, target encoder, and target projector
optimizer	optimizer, updates online parameters using the loss gradient
K and N	total number of optimization steps and batch size
$\{\tau_k\}_{k=1}^K$ and $\{\eta_k\}_{k=1}^K$	target network update schedule and learning rate schedule

for $k = 1$ **to** K **do**
  $\mathcal{B} \leftarrow \{x_i \sim \mathcal{D}\}_{i=1}^N$ // sample a batch of N images
  for $x_i \in \mathcal{B}$ **do**
    $t \sim \mathcal{T}$ and $t' \sim \mathcal{T}'$ // sample image transformations
    $z_1 \leftarrow g_\theta(f_\theta(t(x_i)))$ and $z_2 \leftarrow g_\theta(f_\theta(t'(x_i)))$ // compute projections
    $z'_1 \leftarrow g_\xi(f_\xi(t'(x_i)))$ and $z'_2 \leftarrow g_\xi(f_\xi(t(x_i)))$ // compute target projections
    $l_i \leftarrow -2 \cdot \left(\frac{\langle q_\theta(z_1), z'_1 \rangle}{\|q_\theta(z_1)\|_2 \cdot \|z'_1\|_2} + \frac{\langle q_\theta(z_2), z'_2 \rangle}{\|q_\theta(z_2)\|_2 \cdot \|z'_2\|_2} \right)$ // compute the loss for x_i
  end
  $\delta\theta \leftarrow \frac{1}{N} \sum_{i=1}^N \partial_\theta l_i$ // compute the total loss gradient w.r.t. θ
  $\theta \leftarrow \text{optimizer}(\theta, \delta\theta, \eta_k)$ // update online parameters
  $\xi \leftarrow \tau_k \xi + (1 - \tau_k)\theta$ // update target parameters
end

Output : encoder f_θ

Figure 3. BYOL Algorithm

and $v' \triangleq t'(x)$ are produced from x by applying respective image augmentations $t \sim \mathcal{T}$ and $t' \sim \mathcal{T}'$. From the first augmented view v, the online network outputs a representation $y_\theta \triangleq f_\theta(v)$ and a projection $z_\theta \triangleq g_\theta(y)$. From the second augmented view v', the target network outputs $y'_\xi \triangleq f_\xi(v')$ and the target projection $z'_\xi \triangleq g_\xi(y')$. We then output a prediction $q_\theta(z_\theta)$ of z'_ξ and l_2-normalize both $q_\theta(z_\theta)$ to $\overline{q_\theta}(z_\theta) \triangleq \frac{q_\theta(z_\theta)}{\|q_\theta(z_\theta)\|_2}$ and z'_ξ to $\overline{z'}_\xi \triangleq \frac{z'_\xi}{\|z'_\xi\|_2}$. Also, note that the predictor is the only stage that is present in the online network and absent in the target network. Now, we calculate the mean squared error between normalized predictions and target projections,

$$\mathcal{L}_{\theta,\xi} \triangleq \|\overline{q_\theta}(z_\theta) - \overline{z'}_\xi\|_2^2 = 2 - 2 \cdot \frac{\langle q_\theta(z_\theta), z'_\xi \rangle}{\|q_\theta(z_\theta)\|_2 \cdot \|z'_\xi\|_2}.$$

The loss function is then symmetrized by feeding v' to the online network and v to the target network to compute $\tilde{\mathcal{L}}_{\theta,\xi}$. Only θ is updated using stochastic optimization of minimization of $\mathcal{L}_{\theta,\xi}^{\text{BYOL}} = \mathcal{L}_{\theta,\xi} + \tilde{\mathcal{L}}_{\theta,\xi}$. So the parameter updation of the online network and target network are as follows:

$$\theta \leftarrow \text{optimizer}(\theta, \nabla_\theta \mathcal{L}_{\theta,\xi}^{\text{BYOL}}, \eta),$$
$$\xi \leftarrow \tau\xi + (1 - \tau)\theta.$$

4.1. Code Implementation

Figure 3 shows the algorithm used to implement BYOL approach. STL-10 is used for training the BYOL block as well as training and testing the linear classifier. This dataset consists of 100000 unlabelled images used to train the BYOL block, 500 labeled images used to train the Linear classifier block, and 800 test images per class. It consists of 10 classes of images: airplane, bird, car, cat, deer, dog, horse, monkey, ship, and truck. It is similar to the CIFAR-10 dataset, but differs in the fact that it contains unlabelled images used for self-supervised learning tasks.

During implementation, the images are first passed through many data augmentation transforms with the intention of producing various augmented views such that they are similar and yet different from each other at the same time. As stated in the block diagram of BYOL architecture, an encoder, a projector, and a predictor together form the online network, and the target network consists of an encoder and a projector only. The encoder and projector are together constructed using the ResNet-18 model, taking into account the large dataset that needs to be used to train the model, in both the online network and the target network. The predictor is constructed using a Multi-layer perceptron head within features = number of out channels of the ResNet block, number hidden layer neurons = 128, out features = 128. The loss between the output features of the online network and the target network is calculated and then used to update the network parameters. At first, the target network and the online network are initialized to the same parameters. SGD optimizer with learning rate = 0.03, momentum = 0.9, weight decay = 0.0004 is used to update the parameters. The BYOL blosck is trained for 15 epochs and the loss at each epoch is calculated. Results show that the loss fast during the first iterations and decreases rather slowing in the coming iterations.

Now, to implement a linear classifier using the simplest Logistic Regression classifier as a downstream task, we take the output features of the encoder block as input to the Logistic Regression classifier. The performance increases gradually but decreases after a few epochs. The highest accuracy obtained with the above implementation was 42%. The reason behind the decrease in the accuracy after a few epochs on the downstream tasks could be due to over-training of the linear classifier block.

Thus, from this experiment, we can clearly see that using latent space features as input for downstream tasks reduces the pressure on training the model for particular downstream tasks. Rather, the latent features can be used for various downstream tasks, and a linear classifier is one of them. The model's accuracy can be improved by using a more sophisticated architecture than ResNet-18, to train the BYOL block for latent features, and also increasing the number of epochs. Newer approaches to producing all-purpose features and using them for various downstream tasks are being proposed, which give better performance. The sources for the implementation of the code are [11], [12].

5. Self Distillation with No labels (DiNo)

In this section, we will go through the Dino (Self **Di**stillation with **No** Labels) Algorithm. The approach can be clearly understood from Figure 4. The pseudocode 5 explains the Algorithm in detail. Both these have been taken directly from [1]. DiNo involves training a student network g_{θ_s} to emulate the outputs of a designated teacher network g_{θ_t}, with parameters dictated by θ_s and θ_t, respectively. This is a type of knowledge Distillation [13]. Given augmentations (x_1,x_2) of the input image x, both net-

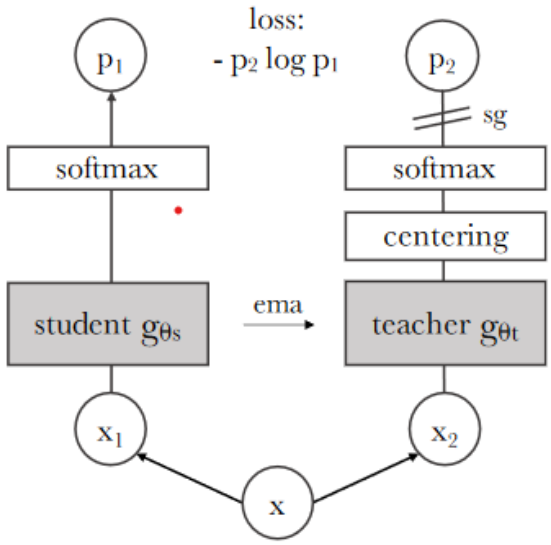


Figure 4. Self Distillation with No Labels

Algorithm 1 DINO PyTorch pseudocode w/o multi-crop.

```
# gs, gt: student and teacher networks
# C: center (K)
# tps, tpt: student and teacher temperatures
# l, m: network and center momentum rates
gt.params = gs.params
for x in loader: # load a minibatch x with n samples
    x1, x2 = augment(x), augment(x) # random views

    s1, s2 = gs(x1), gs(x2) # student output n-by-K
    t1, t2 = gt(x1), gt(x2) # teacher output n-by-K

    loss = H(t1, s2)/2 + H(t2, s1)/2
    loss.backward() # back-propagate

    # student, teacher and center updates
    update(gs) # SGD
    gt.params = l*gt.params + (1-l)*gs.params
    C = m*C + (1-m)*cat([t1, t2]).mean(dim=0)

def H(t, s):
    t = t.detach() # stop gradient
    s = softmax(s / tps, dim=1)
    t = softmax((t - C) / tpt, dim=1) # center + sharpen
    return - (t * log(s)).sum(dim=1).mean()
```

Figure 5. DiNo Algorithm

works yield probability distributions across K dimensions, denoted P_s and P_t. The probability P is obtained by normalizing the network's output g with a softmax function. More precisely, P_s is given by,

$$P_s(x)^i = \frac{\exp(g_s(x)^i/T_s)}{\sum_{k=1}^{K}\exp(g_s(x)^k/T_s)} \tag{5}$$

Similarly, for P_t:

$$P_t(x)^i = \frac{\exp(g_t(x)^i/T_t)}{\sum_{k=1}^{K}\exp(g_t(x)^k/T_t)} \tag{6}$$

with $T_s, T_t > 0$, temperature parameters that control the sharpness of the distribution.

Given a fixed teacher g_t, the objective of the algorithm is to optimize the parameters of the student network, θ_s, such that the distributions produced by the teacher and student networks are aligned. This is achieved by minimizing the cross-entropy loss between the output distributions of the two networks.

$$\min_{\theta_s} H(P_t(x), P_s(x)) \tag{7}$$

In the computation of the Cross-Entropy Loss, as illustrated in Algorithm 1 5, the teacher probabilities are centered before applying the softmax function. Specifically, in the function $H(t, s)$, the teacher's output t has the center C subtracted before it is passed through the softmax. The center C is updated with an exponential moving average as shown in Algorithm 1 5.

Now, let's discuss how the Cross Entropy is adapted for self-supervised learning. A set V of different views is generated from a given image. This set consists of two global views, x_1^g and x_2^g, and several local views (generally 4) of finer resolution. The student processes all crops, whereas only the teacher processes the global views. The Loss is computed as the aggregated cross-entropy between all pairs of augmented views, with the exception of comparisons involving identical augmentations. The loss is minimized as:

$$\min_{\theta_s} \sum_{x\in\{x_1^g, x_2^g\}} \sum_{x'\in V, x'\neq x} H(P_t(x), P_s(x')).$$

However, the standard setting for multi-crop utilizes two global views at a resolution of 224^2, covering a significant area (more than 50%) of the original image, and several local views at a resolution of 96^2, covering smaller sections (less than 50%) of the original image.

During training, the teacher network parameters are frozen. It is built using past iterations of the student network using an exponential moving average method (a.k.a momentum encoder).

$$\theta_t = \lambda\theta_t + (1-\lambda)\theta_s \tag{8}$$

The authors claim that the teacher has better performance than the student during training due to an ensembling effect due to the EWMA method. The neural network used for both the student and the teacher is the same. The neural network has a backbone (ViT or ResNet [14]) and a projection head. The projection head consists of a 3-layer multilayer perceptron (MLP) with a hidden layer followed by a l_2 normalization and a weight-normalized fully connected layer [15] with K dimensions. Batch-normalization is not used in the architecture.

As per the authors, teacher centering along with sharpening introduced through the temperature coefficients τ_t and τ_s is essential for avoiding collapse. There are two types

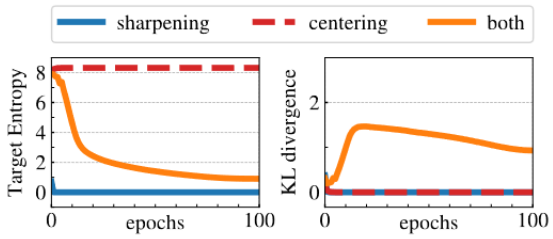


Figure 6. Analysis of Collapse

of collapse possible. One is where the output is uniform, irrespective of the input. The other is where the output is dominated by one feature. Both of these cases are not desirable. The centering avoids the collapse induced by a dominant dimension but encourages a uniform output. Sharpening induces the opposite effect. The cross entropy H can be broken into an entropy h term and a KL-Divergence D_{KL} term:

$$H(P_t, P_s) = h(P_t) + D_{KL}(P_t \| P_s). \tag{9}$$

When the KL value is zero, the output remains constant, leading to a scenario described as a collapse. As depicted in Fig 6, [1] plots the entropy and KL throughout the training process, both with and without the application of centering and sharpening. The absence of either operation causes the KL to approach zero, highlighting the collapse. On the other hand, the entropy, denoted as h, tends to vary in its values: it reaches 0 when devoid of centering and touches $-\log(1/K)$ without sharpening. This observation underscores the distinct collapse tendencies introduced by each operation. Combining both centering and sharpening provides a counterbalance to these tendencies.

6. Experiments with DINO SSL Algorithm

As a part of our experimentation, we tested the DiNo Algorithm on the Imagenette dataset from Fastai [3]. This smaller dataset contains 12,954 images of 10 classes with a 70/30 Train-Validation split. This dataset is ideal for running preliminary tests to help us understand the algorithm properly. The code has been taken from [16][3]. Significant changes have been made considering the limited resources at hand. We repeated the same exercise for ResNet18 and VIT base (DEIT Variant [17]) models. The DiNo hyperparameters used are listed as follows:

- total number of crops: 4 (2 local,2 global)
- momentum teacher (EWMA factor) = 0.995
- teacher temp = 0.04
- student temp = 0.1
- output dimension of the head (K) : 8192

[3] https://github.com/facebookresearch/dino

Model	Validation Set Accuracy (%)
Supervised (from scratch)	70.06
Supervised (ImageNet pre-trained weights)	96.31
DiNo (fine-tuned)	82.17
DiNo (transfer learning)	82.29

Table 1. Validation accuracy (ResNet backbone)

Model	Validation Set Accuracy (%)
Supervised (from scratch)	52.35
Supervised (ImageNet pre-trained weights)	98.34
DiNo (fine-tuned)	72.87
DiNo (transfer learning)	66.24

Table 2. Validation accuracy (ViT backbone)

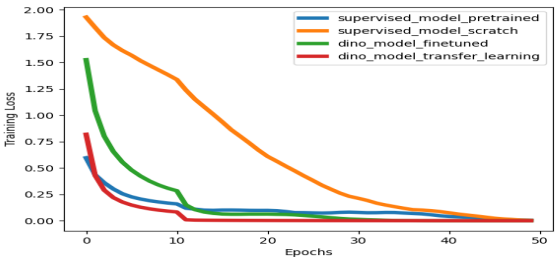
- centre momentum (EWMA factor) 0.9 (fixed)
- DiNo MLP Head with 3 layers (1000,512,1024) with GeLU [18] non-linearities

We changed some of these hyperparameters due to computational constraints. Originally, [1] used 8 crops in total (2 global,6 local). The value of K they used was 65,536. They also used more nodes higher for the middle layer in the DiNo head (4096).

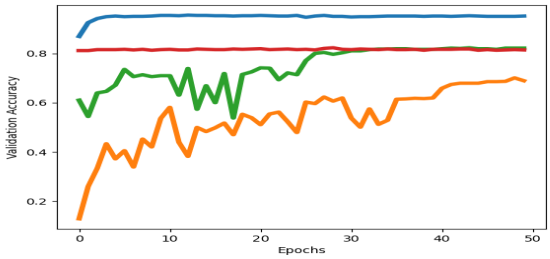
Self-supervised training using DiNo with no labels was done using the Adam optimizer with an initial learning rate of 2.5×10^{-4}, momentum of 0.9 (default), and weight decay of 0.4. The learning rate was manually adapted every 50-75 epochs by monitoring the validation and training loss. K-Nearest Neighbours Accuracy was used as a validation metric. With 160 epochs, the best model using the ResNet18 backbone had a KNN accuracy of 79 %. With nearly 200 epochs, the best model using a VIT base backbone achieved a 70.7 % accuracy.

To fairly compare the performance DiNo with the Supervised Learning method, we first train the ResNet18 [14] by attaching a linear classifier to it in a supervised fashion using only 20% of the dataset. For comparison, we also fine-tuned the models (ResNet18 and VIT base) pretrained on ImageNet [19]. For the DiNo model, we extract the backbone, add a simple linear classifier head, and train(both by finetuning and by freezing the backbone) using the same 20% dataset used for supervised learning. All supervised training was done for 50 epochs using the Adam-optimizer using different learning rates for different cases ($10^{-3}, 10^{-4}$), momentum of 0.9, and weight decay of 10^{-4}.

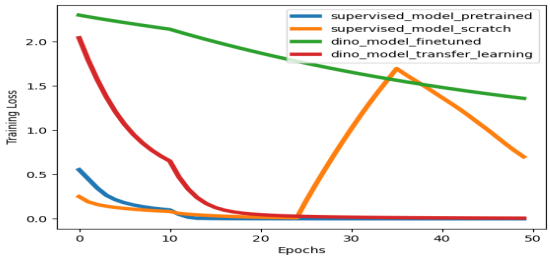
The Validation Scores for each technique and backbone used have been shown in Table 1 and 2. It can clearly seen that both the DiNo models outperform the supervised model by a significant margin (10% in the case of the ResNet backbone and 20% in the case of VIT Backbone). This increased



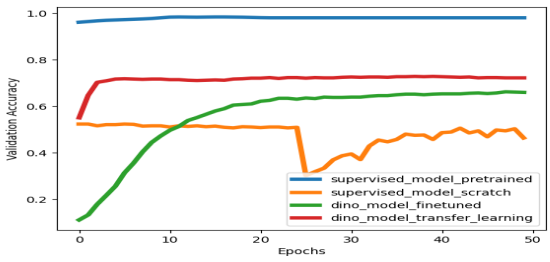
(a) ResNet Training Loss (Moving Average)



(b) ResNet Validation Accuracy



(c) VIT Training Loss (Moving Average)



(d) VIT Validation Accuracy

Figure 7. Plots for Training Loss and Validation Accuracy

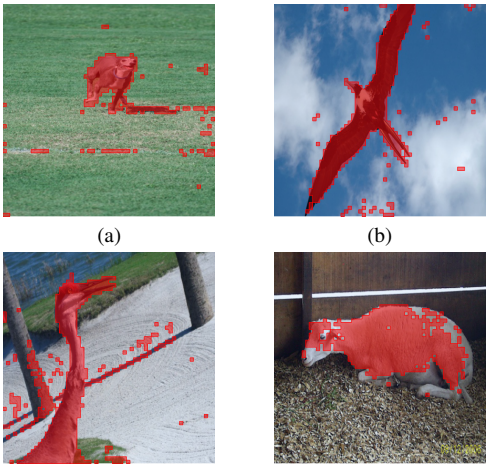
Supervised Model Pretrained
Supervised Model Scratch
DINO Model Transfer Learning
DINO Model Finetuned

performance with the VIT backbone could be because VITs require more training data to perform well than traditional CNNs.

Figure 7a and 7b depict the training loss and the validation set accuracy, respectively, for the ResNet backbone, whereas Figure 7c and 7d depict the training loss and the validation set accuracy, respectively for the VIT backbone. Interestingly, in the ResNet backbone case, the frozen DiNo model performed better during the initial epochs than the supervised model fine-tuned using ImageNet pre-trained weights. However, over time, there is hardly any significant improvement. Eventually, the fine-tuning model catches up and achieves a similar score. The significant difference between the frozen and non-frozen models in the VIT backbone could be attributed to the fact that training hasn't converged even after 50 epochs. More training or a different learning rate could have improved the performance. Another interesting thing is that the training loss converges somewhat despite the supervised model (trained from scratch) having a poor validation score. This could be because of the overfitting of the model on the small dataset (1900 images). Comparatively poorer performance of VITs could be attributed to the smaller nature of the dataset. It's important to note that each model can probably be further optimized by training for more epochs and choosing more appropriate hyper-parameters. However, the comparison should be consistent since we have used similar parameters for each case.

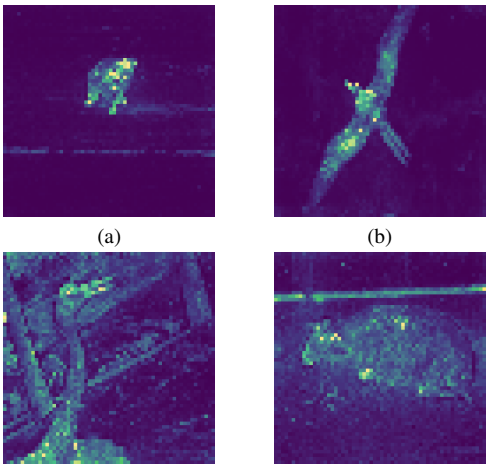
7. Downstream Tasks using DiNo

This section will discuss the downstream tasks we tried to implement. The code used has been primarily obtained from [16]. The authors of [1] report that one can effectively generate segmentation masks from attention maps. The segmentation masks are generated by retaining only a certain fraction of the highest attention values in an attention map; the values in the attention maps are sorted, and only a fraction of the attention values from the top are retained. As shown in 9, we can see that the segmentation masks generated from attention maps are quite accurate. We used ViTs on the PASCAL VOC 2012 dataset to get the segmentation maps and computed the Jaccard index between the predicted and ground truth segmentation maps. The ground truth segmentation map in the dataset has multiple classes, and the predicted segmentation map from Vision Transformers is a single-class segmentation map. The authors did not explicitly mention how to calculate the Jaccard Index, so we devised two methods of computing it. In the first method, we preprocess the ground truth segmentation map so that only the segmentation class with the highest number of pixels remains. In the second method, we combine the regions of different segmentation classes by taking union over all classes. While using the first method with ViT-S with patch size 8, we got a Jaccard index of 36.8. While using the second method with ViT-S with patch size



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

Figure 9. Segmentation Masks



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

Figure 10. Overall Attention Map

8, we got a Jaccard index of 40.65, and with a ViT-S with a patch size of 16, we got a Jaccard index of 48.1. However, the authors of the paper report a Jaccard index of 44.7 and 45.9 for ViT-S with a patch size of 8 and 16, respectively.

There are six heads in the attention modules. We get the overall attention map by taking the pixel-wise maximum across the attention maps to find all the regions in the image to which the transformer pays attention. We have displayed the overall attention maps for the same four images in 10.

We also used a pre-trained vision transformer using DiNo on ImageNet as a backbone and implemented a projection head to perform image classification. We freeze the weights of the backbone and fine-tuned the weights of the projection head(MLP). The authors of [1] report that they could achieve 99% accuracy while finetuning on the CIFAR10 Dataset. However, we could only achieve 93% accuracy when we tried to reproduce the result. This could be due to different hyperparameters and a lower number of epochs.

8. Future Work

Through our experiments, we could see performance improvement using DiNo Self-Supervised Learning on partially labeled datasets. Though our experimentation in this regard was limited to smaller datasets, a lot of current literature indicates that self-supervised learning is the future especially when there is a lack of proper labeling. Further, our primary objective was to harness DiNo-based self-supervised learning techniques on the OCT Dataset for the intricate task of biomarker classification [20]. This dataset has approximately 78,000 B-scan images, of which a mere subset, roughly 9,400, are labeled. Notably, Kokilepersaud et al. [21] previously employed contrastive learning as a self-supervised strategy on this identical dataset. We couldn't go ahead with this primarily because of computational constraints.

We also aim to explore certain downstream tasks further through this project. We have quantified the accuracy of the segmentation masks generated from the attention maps. More work can be done on applying image segmentation for image retrieval using kNN clustering. Since the current segmentation masks have some spurious noise, we would like to improve the accuracy of these segmentation masks by running the Minor Blob removal algorithm. Further, we would like to demonstrate how the different heads of the attention module focus on different regions of the image by visualizing the different regions to which the attention module pays attention.

9. Work carried out by team members

Atharv Ramesh	Vision Transformer, DINO - Literature Survey and Implementation
Nithish S	Image segmentation from ViTs, computing the Jaccard index
Ajit Shankar	Image segmentation from ViTs, literature survey
Theresa Karra	Self Supervised Learning, BYOL and Implementation

Table 3. Contributions of each team member

References

[1] M. Caron, H. Touvron, I. Misra, H. Jégou, J. Mairal, P. Bojanowski, and A. Joulin, "Emerging properties in self-

supervised vision transformers," in *Proceedings of the International Conference on Computer Vision (ICCV)*, 2021. 1, 3, 5, 6, 7

[2] M. Oquab, T. Darcet, T. Moutakanni, H. V. Vo, M. Szafraniec, V. Khalidov, P. Fernandez, D. Haziza, F. Massa, A. El-Nouby, R. Howes, P.-Y. Huang, H. Xu, V. Sharma, S.-W. Li, W. Galuba, M. Rabbat, M. Assran, N. Ballas, G. Synnaeve, I. Misra, H. Jegou, J. Mairal, P. Labatut, A. Joulin, and P. Bojanowski, "Dinov2: Learning robust visual features without supervision," 2023. 1

[3] J. Howard, "Imagewang." [Online]. Available: https://github.com/fastai/imagenette/ 1, 5

[4] A. Dosovitskiy *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020. 1

[5] J.-B. Grill, F. Strub, F. Altché, C. Tallec, P. Richemond, E. Buchatskaya, C. Doersch, B. Avila Pires, Z. Guo, M. Gheshlaghi Azar *et al.*, "Bootstrap your own latent-a new approach to self-supervised learning," *Advances in neural information processing systems*, vol. 33, pp. 21 271–21 284, 2020. 1, 2

[6] A. Vaswani *et al.*, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008. 1

[7] R. Balestriero, M. Ibrahim, V. Sobal, A. S. Morcos, S. Shekhar, T. Goldstein, F. Bordes, A. Bardes, G. Mialon, Y. Tian, A. Schwarzschild, A. G. Wilson, J. Geiping, Q. Garrido, P. Fernandez, A. Bar, H. Pirsiavash, Y. LeCun, and M. Goldblum, "A cookbook of self-supervised learning," *ArXiv*, vol. abs/2304.12210, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:258298825 2

[8] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations," in *International conference on machine learning*. PMLR, 2020, pp. 1597–1607. 2

[9] A. Bardes, J. Ponce, and Y. LeCun, "Vicreg: Variance-invariance-covariance regularization for self-supervised learning," *arXiv preprint arXiv:2105.04906*, 2021. 2

[10] K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick, "Momentum contrast for unsupervised visual representation learning," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 9729–9738. 2

[11] Sthalles, "Pytorch-byol." [Online]. Available: https://github.com/sthalles/PyTorch-BYOL 3

[12] google deepmind, "deepmind-research/byol." [Online]. Available: https://github.com/google-deepmind/deepmind-research/tree/master/byol 3

[13] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," in *NIPS Deep Learning and Representation Learning Workshop*, 2015. [Online]. Available: http://arxiv.org/abs/1503.02531 3

[14] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *Proceedings of 2016 IEEE Conference on Computer Vision and Pattern Recognition*, ser. CVPR '16. IEEE, Jun. 2016, pp. 770–778. [Online]. Available: http://ieeexplore.ieee.org/document/7780459 4, 5

[15] T. Salimans and D. P. Kingma, "Weight normalization: A simple reparameterization to accelerate training of deep neural networks," in *Advances in Neural Information Processing Systems*, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, Eds., vol. 29. Curran Associates, Inc., 2016. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2016/file/ed265bc903a5a097f61d3ec064d96d2e-Paper.pdf 4

[16] P. Bojanowski, "Dino github." [Online]. Available: https://github.com/facebookresearch/dino 5, 6

[17] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jégou, "Training data-efficient image transformers distillation through attention," 2021. 5

[18] D. Hendrycks and K. Gimpel, "Gaussian error linear units (gelus)," 2023. 5

[19] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255. 5

[20] Mohit Prabhushankar, K. Kokilepersaud, Yash-Yee Logan, S. T. Corona, G. AlRegib, and C. Wykoff, "Olives dataset: Ophthalmic labels for investigating visual eye semantics." [Online]. Available: https://zenodo.org/record/7105232 7

[21] K. Kokilepersaud, M. Prabhushankar, and G. AlRegib, "Clinical contrastive learning for biomarker detection," 2022. 7