Deep Active Learning through Self-Supervised Representations

Adithya Ganesh*
Stanford University
Stanford, CA
acganesh@stanford.edu

Anthony Vento*
Stanford University
Stanford, CA
avento@stanford.edu

Abstract

This work applies learned self-supervised representations to the problem of active learning. Specifically, we explore the question – given a dataset D and a fraction $0 < \alpha < 1$ of data that can be labeled, which subset of the data is most helpful to label to maximize image classification performance? We employ BYOL [8] to learn a self-supervised representation on the data that is unlabeled. Given a fixed ResNet18-BYOL model which produces image embeddings, we compare various policies to determine the best subset for labeling, including a random baseline, a K-means clusterwise sampling policy, loss-based sampling, and gradient-based sampling. We tested all of these approaches on three different datasets: STL-10, SVHN, and a synthetic biased CIFAR-10 dataset. Ranking six different samplers based on their classification accuracy, we found that loss-based variance sampling (LossVarSampler) performed best with an average rank of 2.2. Further, we found that KMeans-based sampling (KMeansSampler) performs well in a class imbalanced setting.

1. Introduction

In many machine learning contexts, we have access to large quantities of unlabeled data. However, labeling all the data can be prohibitively expensive and time consuming. Motivated by this, we explore approaches to determine the optimal subset of unlabeled data to label in order to yield maximum image classification performance. Our unique contribution is leveraging a learned self-supervised representation to guide our choice of examples to label.

For our experiments, we start with unlabeled image data, which we use to train a self-supervised BYOL-ResNet18 model. Once this completes, the next step is to fine-tune the model on a selected subset of the labeled training images. We explore various choices of samplers to address this problem of subset selection. Once we obtain labeled data, we then fine-tune the self-supervised model for image classification. Finally, we examine performance metrics from this supervised classification task to study different label selection strategies. This pipeline is shown in Figure 1.

*Equal contribution

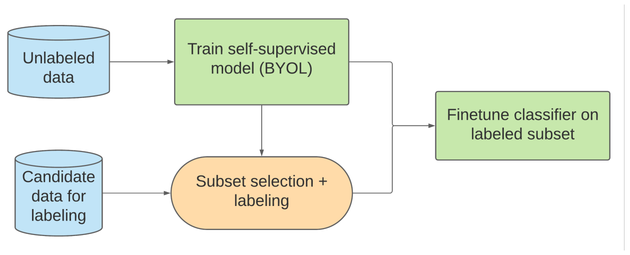

Unlabeled data
Train self-supervised model (BYOL)
Finetune classifier on labeled subset
Candidate data for labeling
Subset selection + labeling

Figure 1: Overall methodology of this work.

In this work, we evaluate algorithms to determine the best subset to label without feedback, which we refer to as "single-step" label selection. While it is possible to incorporate feedback to perform labeling in multiple steps [16]), we focus on the situation where feedback is unavailable for a few reasons. First, the single-step setting is simpler to study and analyze: requesting feedback requires an additional validation split of data and adds many parameters to experiments (e.g. the number of labels to obtain before requesting evaluation). Second, the single-step setting forms the basis for multi-step labeling policies, since the latter would need to apply a similar single-step sampler at each feedback iteration. Finally, a multi-step policy induces overhead computationally (e.g. by requiring multiple trained models) and logistically (e.g. requiring multiple rounds of communication with a labeling firm), so a single-step approach is preferred in many practical contexts.

While there have been many studies investigating self-supervised learning and active learning separately, this project combines both in a unique way that has not been significantly explored in the literature.

2. Related Work

Self-supervised learning is a branch of machine learning that aims to learn useful embeddings without any labeled examples [5]. The goal is to use the learned embeddings for downstream tasks (e.g. image classification or semantic segmentation) and obtain good performance. For example, a linear classifier can be applied to the embeddings learned through self-supervision in order to predict classes. This general procedure is referred to as semi-supervised learning.

One approach to self-supervised learning is to utilize a self-supervised learning pretext task such as image colorization or image completion [19, 15]. In these approaches, a model attempts to learn a model necessary to color the image from a grayscale version or fill in a blacked out region in the image.

However, current state of the art self-supervised models utilize contrastive learning. These include SimCLRv2 [3] and BYOL [8]. Contrastive learning tries to minimize the loss of the latent representations of two augmented views of the same image. Originally, SimCLR [2] proposed the idea of using negative samples when training where the model would attempt to minimize the similarity between the augmented views and the negative samples while maximizing the similarity of the two augmented views. SimCLRv2 extends on the idea of using contrastive learning with negative samples by making their network bigger and wider. BYOL's novelty is that negative samples are not necessary and applying the right training procedure will prevent the network from collapsing to a trivial solution (i.e. it will prevent making all latent representations the same).

Active learning is a domain of machine learning which explores labeling data on the fly. Ren et al.'s paper highlights different methods that can be applied in active learning [16]. Most active learning approaches assume that there is some subset of labeled data to begin and use that to extract features on the unlabeled dataset. These features are then used to determine which datapoints to label and the process is then repeated with the larger labeled dataset.

One common approach in active learning is to use a measure of model uncertainty to determine which images to label, (e.g. by choosing high uncertainty images to label) [1, 11]. Another class of approaches leverage distance measurements in a high dimensional space, which select examples to label that are furthest from some center point (e.g. Farthest First Active Learning [7]). A third approach is to use reinforcement learning [9], which can be applied successfully in contexts where feedback from the previous round is available.

In our approach, following that of BYOL [8], we assume zero labeled training samples to begin and employ self-supervised learning to train a model. Then, we extract features from the self-supervised model to determine which subset to label given a fixed budget (i.e. fraction α) using a single shot approach. Our focus is to combine top performing self-supervised models like BYOL with the problem of determining an optimal subset for labeling.

3. Methods

The overall methodology we followed is shown in Figure 1. We start by training a self-supervised model from unlabeled data. In this paper, we focus on BYOL [8], though this could be substituted with alternatives such as SimCLRv2 [3]. Then, given a remaining set of candidate data that can be labeled, we apply subset selection algorithms that leverage the learned self-supervised representation. Finally, we fine-tune a classifier on the newly obtained labeled data.

3.1. Self-supervised learning with BYOL

The basic self-supervised learning technique we use is BYOL, which learns a representation y_θ from unlabeled data [8]. BYOL consists of two neural networks, the online network and target network. The online network consists of three components: an encoder f_θ, a projector g_θ and a predictor q_θ, as shown in Figure 2.

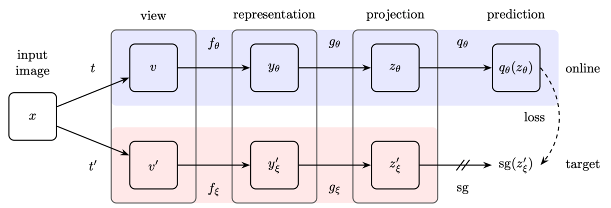


Figure 2: BYOL architecture from [8].

The target network has the same architecture as the online network, and provides regression targets to train the online network. In particular, its parameters ζ are an exponential moving average of the online parameters θ, that is

$$\zeta = \tau\zeta + (1-\tau)\theta,$$

for some time decay $\tau \in [0, 1]$. In this work, we used the ResNet18 architecture [10] for the online and target network.

Given an input image x, BYOL produces two *views*, which are defined as the result of applying two augmentations to the input image.

The loss encourages the representations output by the two networks to be similar for each view. Let $q_\theta(z_\theta)$ denote the prediction of the online network, and z'_ζ denote the projection of the target network. Further, let $\overline{q_\theta}(z_\theta)$ denote the normalized prediction of the online network. The contrastive loss is defined as the mean squared error between the normalized predictions and the target predictions.

$$L_{\theta,\zeta} = ||\overline{q}_\theta(z_\theta) - \overline{z}'_\zeta||_2^2 = 2 - 2 \cdot \frac{\langle q_\theta(z_\theta), z'_\zeta \rangle}{||q_\theta(z_\theta)||_2 \cdot ||z'_\zeta||_2}$$

During training, the loss is minimized with respect to θ only, but not ζ, as is indicated by "sg" (stop gradient) in Figure 2. Overall, at each training step, we will perform the following updates:

$$\theta = \text{optimizer}(\theta, \nabla_\theta L_{BYOL}, \eta)$$
$$\zeta = \tau\zeta + (1-\tau)\theta,$$

where optimizer can be replaced by any optimizer (e.g. Adam), and η is the learning rate.

3.2. Subset selection

In this work, we assume we have a fixed unlabeled dataset $X_{unlabeled}$ used for learning a representation. Then, we have a dataset of candidates $X_{candidates}$ that we can label and finetune the classifier. The main problem we are focused on is determining a subset of $X_{candidates}$ to label and fine-tune on, given an allowed budget of N labels (calculated from the fraction α of $X_{candidates}$). We hypothesize that leveraging the self-supervised embedding learned by the BYOL model can help determine the best subset to label.

In the following pseudocode listings, X_c is shorthand for $X_{candidates}$, N is the number of labels allowed, E is a matrix of BYOL embeddings obtained by computing forward passes on all of the data in X_c (when applicable), and M is the number of forward passes used to compute mean and variance statistics (when applicable).

3.2.1 RandSampler

A naive subset selection algorithm is random sampling. In this algorithm, we pick N examples at random to label. The procedure is shown in Algorithm 1.

Algorithm 1 RandSampler: Random label selection

```
1: procedure RANDSAMPLE($X_c$, $N$)
2:     idx ← RandomChoice(len($X_c$), $N$)
3:     XSubset ← $X_c$[idx]
4:     return XSubset
```

3.2.2 ClusterSampler

Another approach we considered is using clustering to inform the choice of examples to label. Specifically, we employ the procedure described in pseudocode in Algorithm 2.

We start by applying K-Means clustering [13] on the learned embedding vectors with 10 clusters (there were 10 classes for all our datasets). Then, we sample training examples from each cluster such that each cluster is represented with uniform probability. The intuition behind this sampler is that this allows us to query for "diversity" in our examples by ensuring that each cluster is equally represented in our labeled set in expectation.

Algorithm 2 KMeansSampler: uniform weight cluster sampling

```
1: procedure KMEANSAMPLE($X_c$, $E$, $N$)
2:     clusters ← KMeans(nclusters=10).fit(E)
3:     counts ← CountClusters(clusters)
4:     weights ← UniformProbabilityWeights(counts)
5:     idx ← WeightedRandomChoice(len($X_c$)), N, p=weights)
6:     XSubset ← $X_c[idx]$
7:     return XSubset
```

Here, UniformProbabilityWeights is a function that accepts the count of each cluster assigned to elements of E, and computes weights such that each cluster is sampled with uniform probability.

3.2.3 LossSampler

We also considered using the BYOL model's loss to determine the choice of examples of to label. Since the self-supervised loss $L_{\theta,\zeta}$ is stochastic, we tested choosing examples with the highest mean and variance of the loss, computed over $M = 5$ forward passes through X_c. This algorithm is shown in Algorithm 3.

The motivation behind this algorithm is choosing examples where the self-supervised embedding may be less confident or less consistent between runs, which might indicate that the model has not seen very many of a particular type of example before.

Algorithm 3 LossSampler: Sample high mean(loss) and var(loss) examples

```
1: procedure LOSSSAMPLE(model, $X_c$, $N$, $M$)
2:     lossHistory ← zeros($M$, len($X_c$))
3:     for i in [1..M] do
4:         for j in [1..len($X_c$)] do
5:             loss = model.Forward($X_c$[j])
6:             lossHistory[i][j] = loss
7:     lossMean ← PerExampleMean(lossHistory)
8:     lossVar ← PerExampleVar(lossHistory)
9:     idxMean ← argsort(lossMean, reverse=True)[:N]
10:    idxVar ← argsort(lossVar, reverse=True)[:N]
11:    XMeanSubset ← $X_c$[idxMean]
12:    XVarSubset ← $X_c$[idxVar]
13:    return XMeanSubset, XVarSubset
```

3.2.4 GradSampler

Finally, we considered choosing examples based on the gradient norm. Specifically, we considered the L2 norm of the vector obtained by concatenating the gradients of loss with respect to every parameter of the model (including the ResNet encoder and the BYOL embedding and projection layers). The algorithm is shown in Algorithm 4.

This is motivated by choosing examples that result in a large expected change to the model, which could indicate an example that has not been sufficiently encountered during training. In the below equation, the sum is over all parameters p in the model.

$$L_2 = \sqrt{\sum_{p=1}^{P} \left(\frac{\partial L_{\theta,\zeta}}{\partial p} \right)^2}$$

We implemented our subset selection algorithm based on the mean and variance of this norm, across $M = 5$ backward passes through all of X_c.

Algorithm 4 GradSampler: Sample based on norm of grad.

```
1:  procedure GRADSAMPLE(model, $X_c$, $N$, $M$)
2:      gradHistory ← zeros(M, len($X_c$))
3:      for i in [1..M] do
4:          for j in [1..len($X_c$)] do
5:              loss = model.Forward($X_c$[j])
6:              loss.Backward()
7:              gradNorm = L2Norm(GetAllGrads(model))
8:              gradHistory[i][j] = gradNorm
9:      gradMeans ← PerExampleMean(gradHistory)
10:     gradVars ← PerExampleVar(gradHistory)
11:     idxMean ← argsort(gradMeans, reverse=True)[:N]
12:     idxVar ← argsort(gradVars, reverse=True)[:N]
13:     XMeanSubset ← $X_c$[idxMean]
14:     XVarSubset ← $X_c$[idxVar]
15:     return XMeanSubset, XVarSubset
```

3.3. Fine-tuning from BYOL embeddings

Once we obtain the labeled subset X_{subset} and labels Y_{subset}, we start by computing E_{subset}, a matrix of BYOL embeddings obtained for each example in X_{subset}. Then, we fine-tune a final layer that maps these embeddings to the ground truth classes in our dataset. We considered the alternative of backpropagating through further layers of the model (e.g. through a portion of the ResNet18 encoder), but we obtained for the last-layer strategy as it was computationally cheaper, and has precedent in the linear evaluation procedure applied in [8]. Additionally, while fine-tuning more layers might produce better absolute accuracy, we are more interested in the relative accuracy between different subset selection algorithms.

4. Datasets

We conducted our experiments on three datasets: STL10, SVHN, and a biased CIFAR-10 dataset.

4.1. STL10

One dataset we used was Self-Taught Learning 10 (STL10) [4], a popular self-supervised and semi-supervised image classification dataset. This dataset consists of 100,000 unlabeled training images, 5,000 labeled training images, and 8,000 labeled test images, obtained from ImageNet. The image size is 96 × 96 × 3, and there are 10 image classes.

$X_{candidates}$ was set to be the "5000 labeled training images" previously mentioned. STL10 was chosen because the images have a wide diversity between classes.

4.2. SVHN

Another dataset we used was the Street View House Numbers (SVHN) dataset [14]. It consists of 73257 digits from house signs for training, 26032 digits for testing, and 531131 additional samples designated as "extra" training data. The image size is 32 × 32 × 3, and there are 10 classes for each digit.

$X_{candidates}$ was set to be the first 10000 images of the "extra" split. SVHN provides a different kind of dataset in that numbered digits have less visual diversity than STL10.

4.3. Biased CIFAR-10

As we were interested in exploring the effect of class imbalanced data on subset selection, we constructed a synthetic biased dataset from CIFAR-10. The regular CIFAR-10 dataset [12] consists of 60000 images in 10 classes, with 6000 images per class, split between 50000 for training and 10000 for testing.

We constructed a synthetic "biased CIFAR-10" dataset by making each class contain 10%, 20%, ... 100% of the original class size. For example, class 0 has 10% of the original size, class 1 has 20% of the original size, etc. Then, we do a stratified split where we set $X_{unlabeled}$ to be 24750 images, $X_{candidates}$ to be 2750 images, and X_{test} to be 5500 images. Class counts can be seen below.

Class	$X_{unlabeled}$	$X_{candidates}$	X_{test}
0	450	50	100
1	900	100	200
⋮	⋮	⋮	⋮
9	4500	500	1000

4.4. Preprocessing and Augmentation

We performed normalization so that the mean of the training set is 0 and the variance is 1. The mean and variance statistics were calculated using the unlabeled self-supervised images. We used the same mean and variance statistics to normalize test images during inference.

For data augmentation, we follow the set of image augmentations employed in SimCLR [2]. Specifically, we choose a random patch of the image, and resize it to the size of the input image with a random horizontal flip and color jitter and distortion. At the end of this we apply Gaussian blur to each patch and, as stated previously, we normalize across channels to zero mean and unit variance.

5. Experiments & Results

For the encoder of BYOL we use ResNet-18 [10]. For the projection and predictor, we use fully connected layers with embedding size 512 and projection size 256. We used the Adam optimizer with learning rate 3×10^{-4} and a batch size of 512. We trained 3 separate BYOL models for 100 epochs on the unlabeled dataset of STL10, SVHN, and Biased CIFAR-10 datasets. We chose these values after examining compute constraints and after examining initial loss values. Overall, we tested 108 different allowed label counts, across 6 different sampler algorithms and 3 different datasets, resulting in a total of 1944 distinct classification models trained.

To evaluate performance on the test datasets, we computed Top-1 accuracy, Top-3 accuracy, and average precision [6] for the rare class 0 for the biased CIFAR-10 experiments. We chose Top-1 and Top-k accuracy for $k = 3$ as these are standard metrics used in the image classification literature [10, 8]. Since we often use a very small subset of labels to fine-tune, we are more interested in relative differences in accuracy than absolute accuracy. Average precision was examined as it provides a useful class specific measure of performance for the rarest class 0 in Biased CIFAR-10. Furthermore, accuracy is typically a less useful signal for such class imbalanced situations.

5.1. Comparing BYOL to Logistic Regression + PCA

Table 1 compares the performance of the BYOL-ResNet18 classifier with that of LogisticRegression on PCA [18] features obtained from images directly (512 components), while we vary the number of labels used and the label selection policy.

The BYOL-ResNet18 classifier universally outperforms LogisticRegression on all dataset types by a large margin. This is unsurprising, but validates the fact that BYOL is learning useful representations through the self-supervised learning procedure.

5.2. Sampler Performance Comparison

Figure 3 shows a detailed comparison of Top-1 accuracy as we very the number of labels used. We noticed a few

Classifier	Sampler	Labels Used	Top-1 Acc
PCA-LR	RandSampler	0.6%	.197
BYOL-ResNet	RandSampler	0.6%	.363
PCA-LR	RandSampler	20%	.296
BYOL-ResNet	RandSampler	20%	.662
PCA-LR	Use all labels	100%	.319
BYOL-ResNet	Use all labels	100%	.717

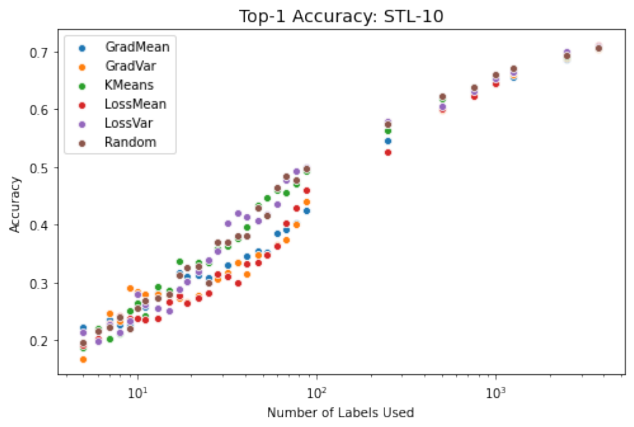
Table 1: Performance metrics comparing the BYOL-ResNet18 classifier to baseline on the STL-10 Dataset for various allowed labels.

trends. First off, the performance of all models increases as the number of labels increases – this is intuitive. Additionally, we found that the LossSamplerVar method generally performed near the top (purple color in scatter plots). A possible intuition for this is that a high variance in the stochastic self-supervised loss $L_{\theta,\zeta}$ suggests the model is less confident in a particular example subject to different augmentations. Thus, a ground truth label for such an example benefits classification performance. Finally, we note that simple random sampling of labels performs quite well once we have a relatively large number of labels (e.g. $>=$ 20%), while other samplers produce similar levels of performance. This can be attributed to the fact that a random sampling policy will converge to sampling uniformly from each class in expectation, which is likely a decent default for balanced datasets.

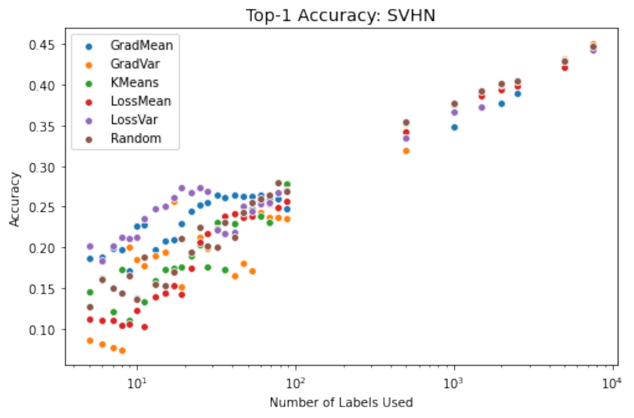
We rank each sampling algorithm across various choices of allowed labels. Performing this ranking in the low data regime across all datasets (linearly spaced from 5 examples to 90 examples), we found that LossBasedVar had the best average rank of 2.2. Random sampling had an average rank of 2.7, K-Means was 3.17, GradSamplerMean was 3.4, GradSamplerVar was 4.61, and LossSamplerMean was 5. We also found a similar trend when examining Top-3 accuracy. We present some of the metrics from Figure 3 in the appendix.

5.3. Rare Class Analysis

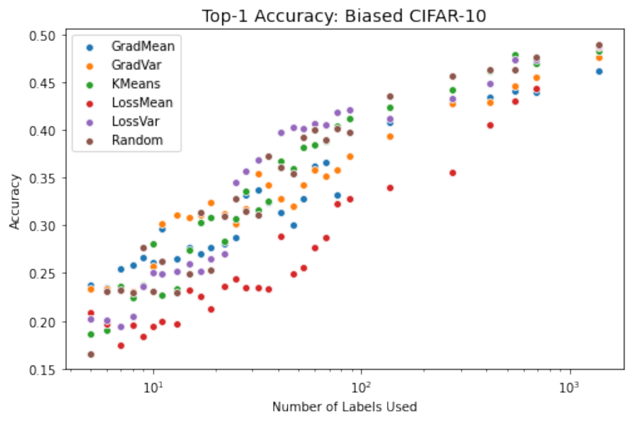
We were also interested in seeing how the sampling algorithm performs on the rare classes in the Biased CIFAR-10 dataset. We observed that K-Means sampling works relatively well in detecting rare classes in the low-data regime ($<$ 1% of labels), while random sampling sometimes does not. For example, Table 2 shows the average precision for the sampling strategies on Class 0 (the rarest class), where higher values are better. This could be explained by the fact that K-Means sampling policy deliberately samples outliers from diverse clusters. On the other hand, K-Means sampling was not the best for the balanced STL-10 and SVHN datasets, which supports the intuition that different samplers



(a) Top-1 Accuracy on the STL-10 dataset. In STL-10, there are 5000 possible candidates for labeling.



(b) Top-1 Accuracy on the STL-10 dataset. For SVHN, there are 10000 possible candidates for labeling.



(c) Top-1 Accuracy on the Biased CIFAR-10 dataset. In this dataset, there are 2750 possible candidates for labeling.

Figure 3: Top-1 Accuracies for different datasets, sampling strategies, and number of labels used.

Sampler	# of Examples	AP of class 0
RandSampler	20	0.021
LossSamplerVar	20	0.017
LossSamplerMean	20	0.039
KMeansSampler	20	0.024
GradSamplerVar	20	0.019
GradSamplerMean	20	0.018
RandSampler	40	0.040
LossSamplerVar	40	0.016
LossSamplerMean	40	0.034
KMeansSampler	40	0.047
GradSamplerVar	40	0.024
GradSamplerMean	40	0.018
RandSampler	80	0.040
LossSamplerVar	80	0.027
LossSamplerMean	80	0.028
KMeansSampler	80	0.055
GradSamplerVar	80	0.021
GradSamplerMean	80	0.033

Table 2: Performance metrics comparing the choice of various samplers on the rare class of biased CIFAR-10.

should be applied for different downstream classification tasks and different data distributions.

Qualitatively, we were interested in seeing which images from the rare class (if any) the subset algorithm would select for labeling. We found that, in some cases, the KMeansSampler was able to successfully pick the rare class 0 (airplanes) to be labeled. In general, we noticed that the KMeanSampler would pick up on standard commercial passenger jets. For example, when running the KMeansSampler on 30 training examples, the algorithm chose to label the two images shown in 4. Therefore, when evaluating the performance of the downstream semi-supervised classifier, we noticed that the model could classify airplanes that looked like commercial passenger jets, but struggled with variants that looked more like fighter jets (see Figure 5).

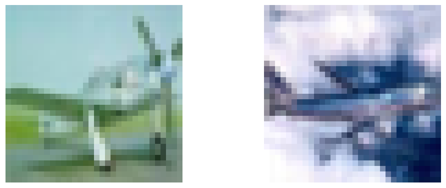


Figure 4: Candidates the KMeansSampler chose to label that were class 0, where we limited the sampler to select 30 images to label, despite class 0 consisting of only 1.8% of the dataset.

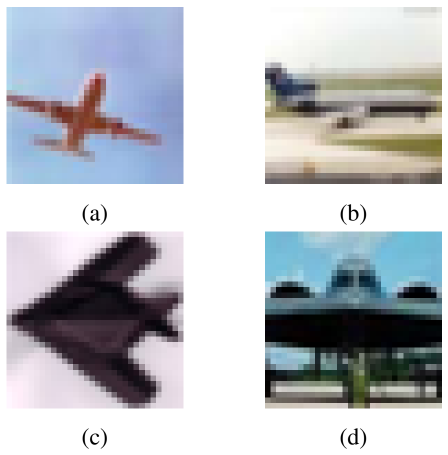


Figure 5: Biased-CIFAR 10 test examples. After fine tuning on the two images in Figure 4, obtained via KMeansSampler, the model was able to classify 5a and 5b but did not correctly classify 5c and 5d from class 0.

6. Discussion

6.1. Takeaways

Overall, we found that the LossVarSampler, KMeansSampler, and RandSampler policies worked the best, though their strengths appeared in different situations. LossVarSampler performed well in the balanced dataset case. This sampler finds examples where the self-supervised loss varies significantly between forward passes. We hypothesize this is useful because it identifies examples where the embedding changes when different augmentations are applied, which indicates lower confidence from the model.

RandSampler performed well in the balanced dataset case as the number of allowed labels gets larger (approximately $>20\%$), which has the advantage of being conceptually simple and computationally fastest. The KMeansSampler performed best when evaluated by downstream classification performance on a rare class on Biased CIFAR-10. This reflects the need to sample images that are "distant" in embedding space when rare classes are present.

The gradient-based samplers (GradMeanSample and GradVarSample) did not perform as well. One possible explanation is that we computed the norm of the gradient of the loss with respect to every parameter in the network. This might be too noisy of a signal, given that ResNet18 alone has around 11 million trainable parameters, and we add even more parameters via the BYOL layers and the fine-tuning layers. We hypothesize that taking the norm of the gradient of the loss with respect to more specific quantities of interest, such as the embedding vector, would provide a more useful signal to the subset selection algorithm.

6.2. Limitations

One limitation of this analysis is that we have not evaluated classification performance obtained by fine-tuning more layers from the ResNet18. We opted to fine-tune the final linear layer because it has been shown to work well [8] in practice and fine-tuning more layers would be much more computationally expensive, but the metrics presented might not reflect the maximum performance of the BYOL-ResNet18 model.

Additionally, as mentioned in the introduction, this work did not consider the effect of obtaining labels with "feedback" (e.g. by training a model and evaluating it on a validation set). While single-step label selection is often practically desirable and foundational to multi-step label selection, multi-step methods could leverage entirely different types of approaches (e.g. reinforcement learning).

7. Conclusion

Selecting examples to label is an important question in many practical scenarios, for instance when we have a large amount of unlabeled data or when labeling is very expensive. We have presented various sampling algorithms for performing this task and identified KMeansSampler and LossVarSampler as particularly promising. We analyzed the performance of each sampler on three diverse datasets, and found that KMeansSampler worker particularly well in the setting of extreme class imbalance, while LossVarSampler performed the best on average. Code is available here: https://github.com/acganesh/self-supervised-AL.

Future work could explore applying the methods presented in this paper to different datasets. One possibility is running these samplers on ImageNet, which has many more classes than the 10 classes in each of our datasets. The difficulty here is that even training BYOL on ImageNet is extremely computationally expensive: the BYOL model introduced in [8] was trained for 8 hours on 512 TPUs. This does not even consider the computational requirements of running each subset selection algorithm, which each have many parameters to tune. Another interesting avenue of exploration is considering label selection in much more long-tailed datasets. For instance, medical images might have a rare class ratio of far less than the 1.8% we examined in Biased CIFAR-10. More extreme "needle in a haystack" type problems might be needed to highlight the full potential of the sampling methods in this work (e.g. finding one example from millions).

8. Contributions & Acknowledgements

In general, we ensured that work was split evenly between the two of us. Both of us worked together to brainstorm project ideas, train models, and design and implement selection algorithms. In particular, Adithya worked on setting up BYOL training and fine-tuning, serializing embeddings and metrics, developing the loss and k-means algorithms, and creating precision and recall plots. Anthony worked on the dataset construction and biased dataset construction, training and fine tuning the BYOL model, developing the gradient based selection algorithm, and generating accuracy plots. We both split up writing this report.

We would like to thank Chris Waites, our CS231N project TA, for sharing useful suggestions. Additionally, we would like to thank the maintainers of PyTorch BYOL[17], which we modified and extended to fit our needs. Finally, we would like to thank the Stanford Research Computing Center with their assistance on GPU set up.

9. Appendix: Additional Metrics Table

Dataset	Sampler	Number of Examples	Top-1 Accuracy	Top-3 Accuracy
Biased CIFAR-10	RandSampler	20	0.329	0.504
Biased CIFAR-10	LossSamplerVar	20	0.323	0.480
Biased CIFAR-10	LossSamplerMean	20	0.227	0.367
Biased CIFAR-10	KMeansSampler	20	0.284	0.436
Biased CIFAR-10	GradSamplerVar	20	0.342	0.482
Biased CIFAR-10	GradSamplerMean	20	0.222	0.334
Biased CIFAR-10	RandSampler	137	0.436	0.615
Biased CIFAR-10	LossSamplerVar	137	0.412	0.569
Biased CIFAR-10	LossSamplerMean	137	0.340	0.510
Biased CIFAR-10	KMeansSampler	137	0.424	0.597
Biased CIFAR-10	GradSamplerVar	137	0.393	0.618
Biased CIFAR-10	GradSamplerMean	137	0.408	0.585
Biased CIFAR-10	RandSampler	550	0.464	0.649
Biased CIFAR-10	LossSamplerVar	550	0.473	0.655
Biased CIFAR-10	LossSamplerMean	550	0.431	0.614
Biased CIFAR-10	KMeansSampler	550	0.479	0.660
Biased CIFAR-10	GradSamplerVar	550	0.447	0.647
Biased CIFAR-10	GradSamplerMean	550	0.442	0.636
Biased CIFAR-10	Use all labels	2750	0.503	0.687
SVHN	RandSampler	100	0.274	0.431
SVHN	LossSamplerVar	100	0.268	0.408
SVHN	LossSamplerMean	100	0.255	0.397
SVHN	KMeansSampler	100	0.265	0.416
SVHN	GradSamplerVar	100	0.250	0.414
SVHN	GradSamplerMean	100	0.263	0.406
SVHN	RandSampler	500	0.354	0.522
SVHN	LossSamplerVar	500	0.334	0.498
SVHN	LossSamplerMean	500	0.342	0.503
SVHN	KMeansSampler	500	0.352	0.521
SVHN	GradSamplerVar	500	0.319	0.492
SVHN	GradSamplerMean	500	0.319	0.489
SVHN	RandSampler	2000	0.402	0.574
SVHN	LossSamplerVar	2000	0.399	0.570
SVHN	LossSamplerMean	2000	0.394	0.569
SVHN	KMeansSampler	2000	0.398	0.571
SVHN	GradSamplerVar	2000	0.401	0.576
SVHN	GradSamplerMean	2000	0.377	0.556
SVHN	Use all labels	10000	0.457	0.635
STL10	RandSampler	50	0.417	0.624
STL10	LossSamplerVar	50	0.444	0.619
STL10	LossSamplerMean	50	0.306	0.493
STL10	KMeansSampler	50	0.415	0.598
STL10	GradSamplerVar	50	0.400	0.593
STL10	GradSamplerMean	50	0.445	0.659
STL10	RandSampler	500	0.623	0.801
STL10	LossSamplerVar	500	0.605	0.784
STL10	LossSamplerMean	500	0.601	0.796
STL10	KMeansSampler	500	0.618	0.796
STL10	GradSamplerVar	500	0.598	0.793
STL10	GradSamplerMean	500	0.621	0.802
STL10	RandSampler	1000	0.661	0.831
STL10	LossSamplerVar	1000	0.654	0.830
STL10	LossSamplerMean	1000	0.644	0.825
STL10	KMeansSampler	1000	0.653	0.829
STL10	GradSamplerVar	1000	0.645	0.823
STL10	GradSamplerMean	1000	0.645	0.818
STL10	Use all labels	5000	0.716	0.869

Table 3: Detailed additional metrics, exploring the performance of various samplers for a broad range of allowed label counts.

References

[1] Nabiha Asghar, Pascal Poupart, Xin Jiang, and Hang Li. Deep active learning for dialogue generation. *arXiv preprint arXiv:1612.03929*, 2016. 2

[2] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 1597–1607. PMLR, 13–18 Jul 2020. 2, 5

[3] Ting Chen, Simon Kornblith, Kevin Swersky, Mohammad Norouzi, and Geoffrey Hinton. Big self-supervised models are strong semi-supervised learners. *arXiv preprint arXiv:2006.10029*, 2020. 2

[4] Adam Coates, Andrew Ng, and Honglak Lee. An analysis of single-layer networks in unsupervised feature learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 215–223. JMLR Workshop and Conference Proceedings, 2011. 4

[5] Carl Doersch, Abhinav Gupta, and Alexei A Efros. Unsupervised visual representation learning by context prediction. In *Proceedings of the IEEE international conference on computer vision*, pages 1422–1430, 2015. 2

[6] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338, 2010. 5

[7] Yonatan Geifman and Ran El-Yaniv. Deep active learning over the long tail. *arXiv preprint arXiv:1711.00941*, 2017. 2

[8] Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre H Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Daniel Guo, Mohammad Gheshlaghi Azar, et al. Bootstrap your own latent: A new approach to self-supervised learning. *arXiv preprint arXiv:2006.07733*, 2020. 1, 2, 4, 5, 7

[9] Manuel Haußmann, Fred A Hamprecht, and Melih Kandemir. Deep active learning with adaptive acquisition. *arXiv preprint arXiv:1906.11471*, 2019. 2

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 2, 5

[11] Tao He, Xiaoming Jin, Guiguang Ding, Lan Yi, and Chenggang Yan. Towards better uncertainty sampling: Active learning with multiple views for deep convolutional neural network. In *2019 IEEE International Conference on Multimedia and Expo (ICME)*, pages 1360–1365. IEEE, 2019. 2

[12] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009. 4

[13] Stuart Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137, 1982. 3

[14] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. 2011. 4

[15] Deepak Pathak, Philipp Krahenbuhl, Jeff Donahue, Trevor Darrell, and Alexei A Efros. Context encoders: Feature learning by inpainting. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2536–2544, 2016. 2

[16] Pengzhen Ren, Yun Xiao, Xiaojun Chang, Po-Yao Huang, Zhihui Li, Xiaojiang Chen, and Xin Wang. A survey of deep active learning. *arXiv preprint arXiv:2009.00236*, 2020. 1, 2

[17] Phil Wang. byol-pytorch: http://github.com/lucidrains/byol-pytorch/, 2020. 8

[18] Svante Wold, Kim Esbensen, and Paul Geladi. Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2(1-3):37–52, 1987. 5

[19] Richard Zhang, Phillip Isola, and Alexei A Efros. Colorful image colorization. In *European conference on computer vision*, pages 649–666. Springer, 2016. 2