Sparse Label Learning

Final Presentation
Group: team-3-for-the-win

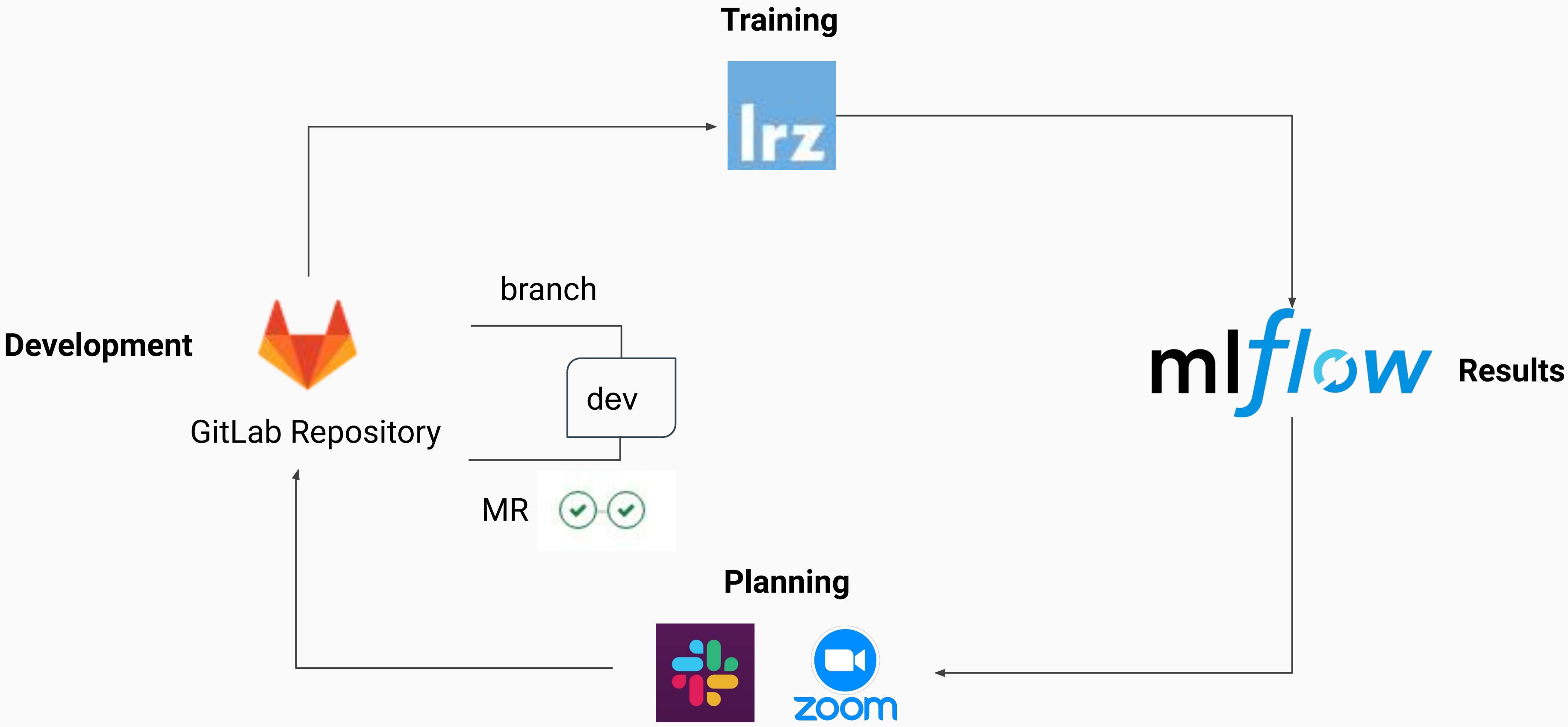
21/07/2021

Team: Pavel, Gabriel, Cristian, Elena, Max
Supervisors: Sandra Obermeier, Evgeniy Faerman

Overview

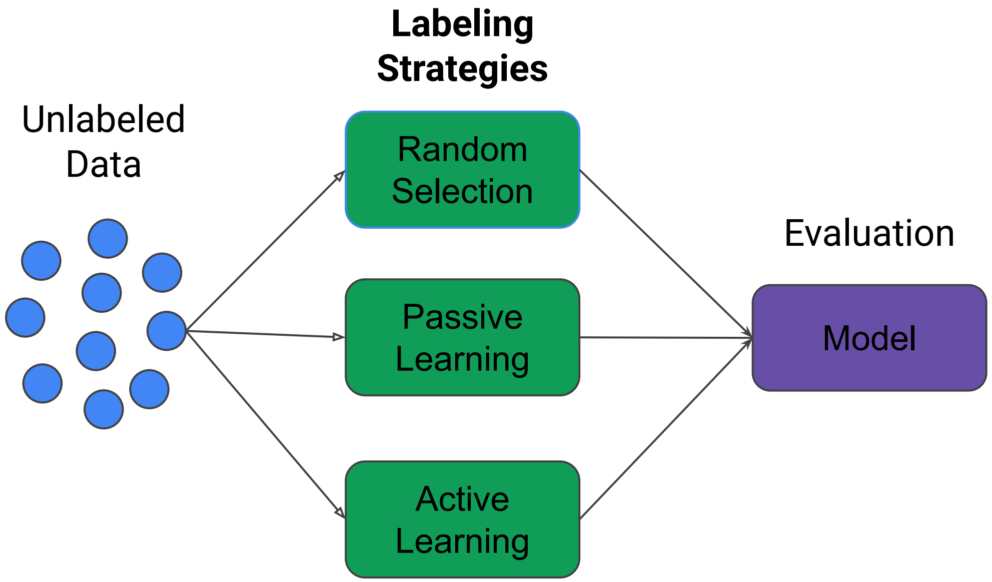
- Concept, Setup, Models, Datasets
- Passive Learning Concept (clustering methods / sampling strategies), Active Learning Concept (outlook)
- Training Results
- Feature Quality Study
- Conclusion & Future Work

Workflow



Training
Development
branch
dev
GitLab Repository
MR
Results
Planning

Concept



“Sparse Label Learning”

→ only minority of training data labeled

→ Labeling is expensive!

→ Goal: Compare labelling strategies

Models

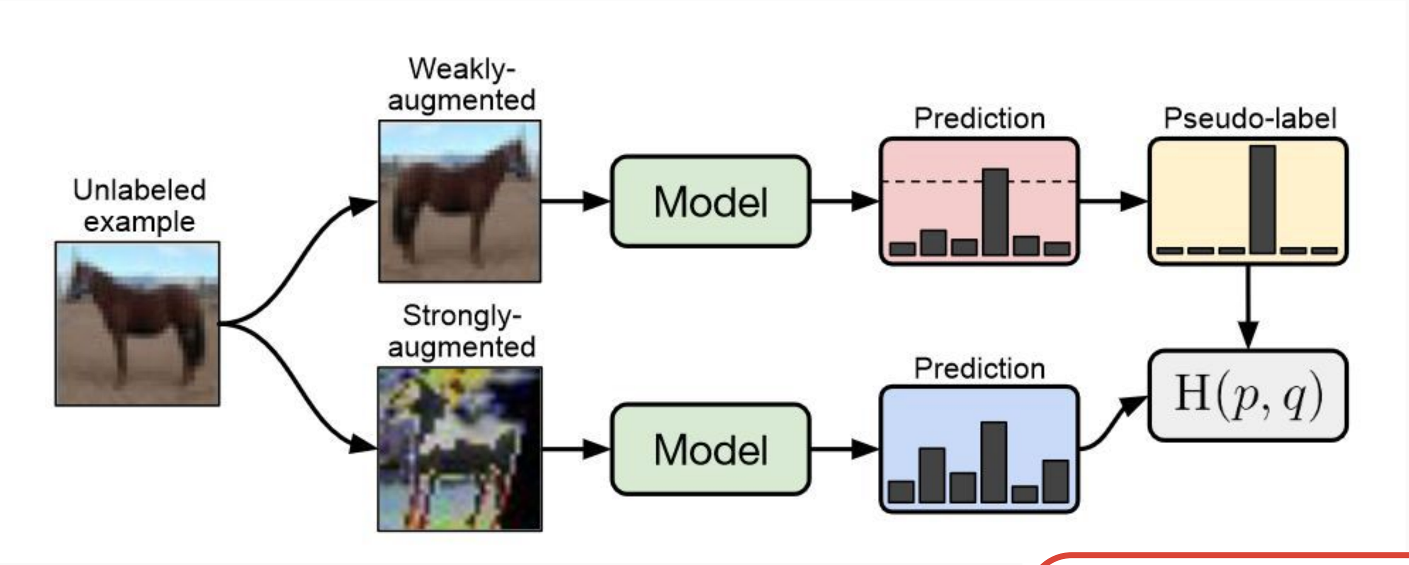
Model	Architecture	Pretrained
Fixmatch	WideResNet 28x2	No
SSL	WideResNet 50x2	No
Basic	WideResNet 50x2	No
Transfer	WideResNet 50x2	ImageNet

Hyperparameters

- Learning-rate: 0.03
- Optimizer: Adam
- Weight-decay: cosine-scheduler, 0.005 (0.01 for CIFAR100)
- Goal: Comparability

Models: Semi-supervised Learning

FixMatch Approach: Pseudo-label + **Consistency regularization**

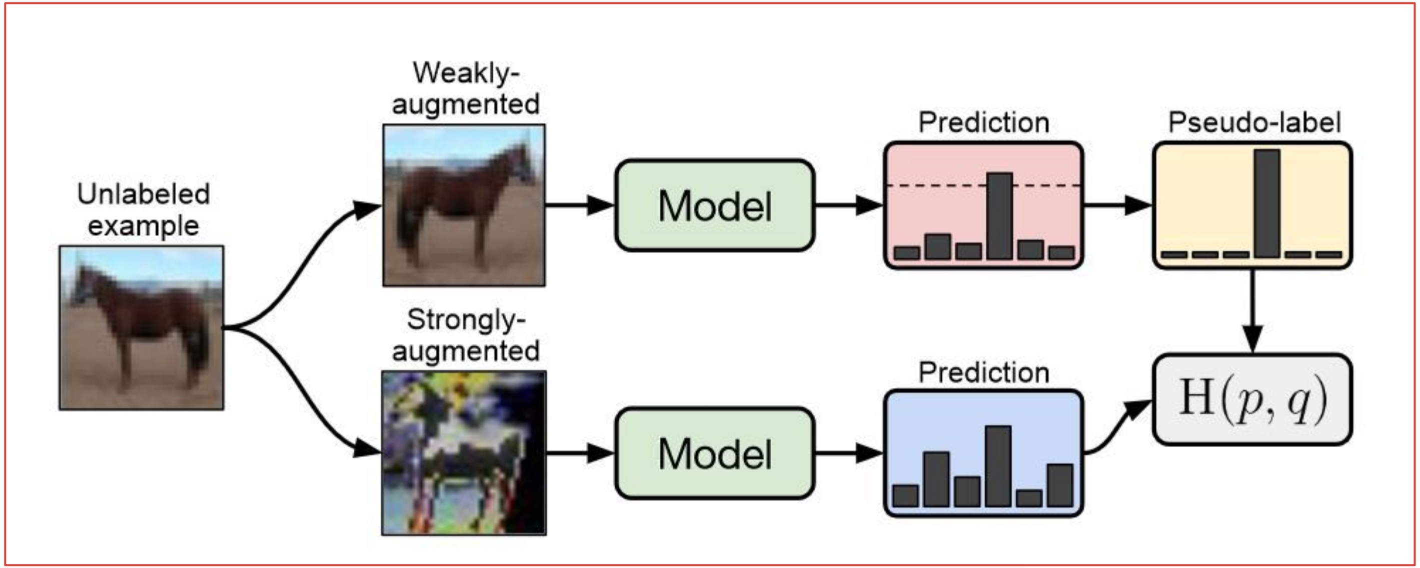


Main takeaway: SSL models can make use of unlabeled examples in addition to labeled data

$$\underbrace{\min_{\theta} \sum_{(x,y)\in X_L}^{n} \mathcal{L}_s(x,y,\theta)}_{\text{supervised loss}} + \alpha \underbrace{\sum_{(x)\in X_U}^{m} \mathcal{L}_u(x,\theta)}_{\text{unsupervised loss}}$$

Models: Semi-supervised Learning

FixMatch Approach: Pseudo-label + **Consistency regularization**

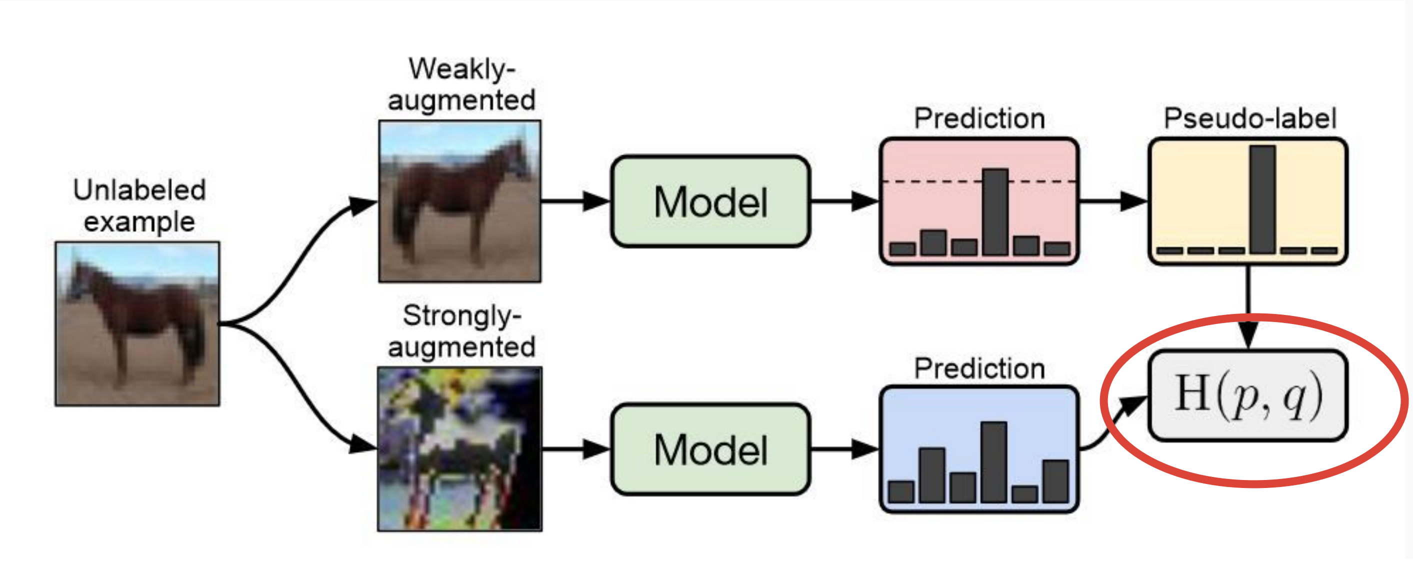


Main takeaway: SSL models can make use of unlabeled examples in addition to labeled data

$$\underbrace{\min_{\theta} \sum_{(x,y)\in X_L}^{n} \mathcal{L}_s(x,y,\theta)}_{\text{supervised loss}} + \alpha \underbrace{\sum_{(x)\in X_U}^{m} \mathcal{L}_u(x,\theta)}_{\text{unsupervised loss}}$$

Models: Semi-supervised Learning

FixMatch Approach: Pseudo-label + **Consistency regularization**



Main takeaway: SSL models can make use of unlabeled examples in addition to labeled data

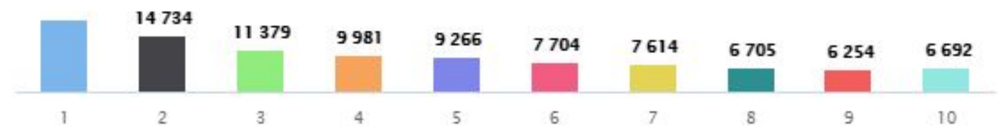
$$\underbrace{\min_{\theta} \sum_{(x,y)\in X_L}^{n} \mathcal{L}_s(x, y, \theta)}_{\text{supervised loss}} + \alpha \underbrace{\sum_{(x)\in X_U}^{m} \mathcal{L}_u(x, \theta)}_{\text{unsupervised loss}}$$

Datasets

ID	Name	n	features	classes
40927	CIFAR_10	60_000	3073	10

ID	Name	n	features	classes
41983	CIFAR-100	60_000	3073	100

ID	Name	n	features	classes
41081	SVHN	99289	3073	10



- Original
- Modified (10% of samples for 50% of classes)

Passive Learning

Recap:

- Labeling is expensive!
- Idea: Find certain subsets of labeled data that lead to best training result.
- Use unsupervised selection methods (like clustering) on extracted dataset features *before* training. Start training with only 10/100/300 points per class as #selected_samples.

Features: With different Neural Network pretrained on ImageNet, use feature extraction (representation at last layer) for each sample of our datasets.

Clustering Methods

- ***Kmeans***: simplest and popular unsupervised clustering
  - Move centroids to mean distance of assigned samples

- ***Dbscan & optics:*** density based clustering methods
  - minPts: The minimum number of points (a threshold) clustered together for a region to be considered dense.
  - eps (ε): A distance measure that will be used to locate the points in the neighborhood of any point.
  - 2 more parameters for optics : Core Distance and Reachability Distance

- ***Coreset:*** minimal set of data points (training samples) that allows the model to deliver approximately as good a performance as it would if the whole training data set was used.
  - We use **K-Center-Greedy** Coreset in our project, which has following steps:
    1.pick randomly one center
    2.choose next one which is furthest to the current centers as new center
    3.continue until all k centers are picked

Sampling strategies

Kmeans:

- Take top x sample from centroid (#centroids < #samples to label)
    i. Closest distance
    ii. Any #x from cluster randomly
- Closest sample to each centroid (#centroids == #samples to label) ←

Dbscan & optics:

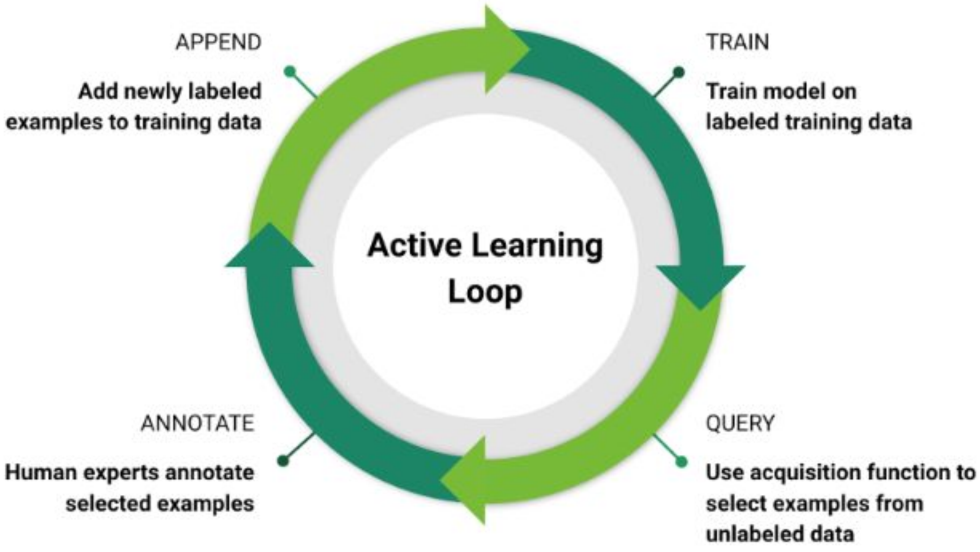
- Depending on #clusters := eps, minpts
- Take x from cluster randomly

Coreset:

- Take the output of the algorithm (k Centers) as the samples to label
- #centers == #points

Outlook: Active Learning

- Used when there is a huge amount of unlabelled data
- Model is trained on small amount of data and an acquisition function, which determines which data point to label next
- Annotate selected samples and add them to training set
- Train new model on the bigger training set

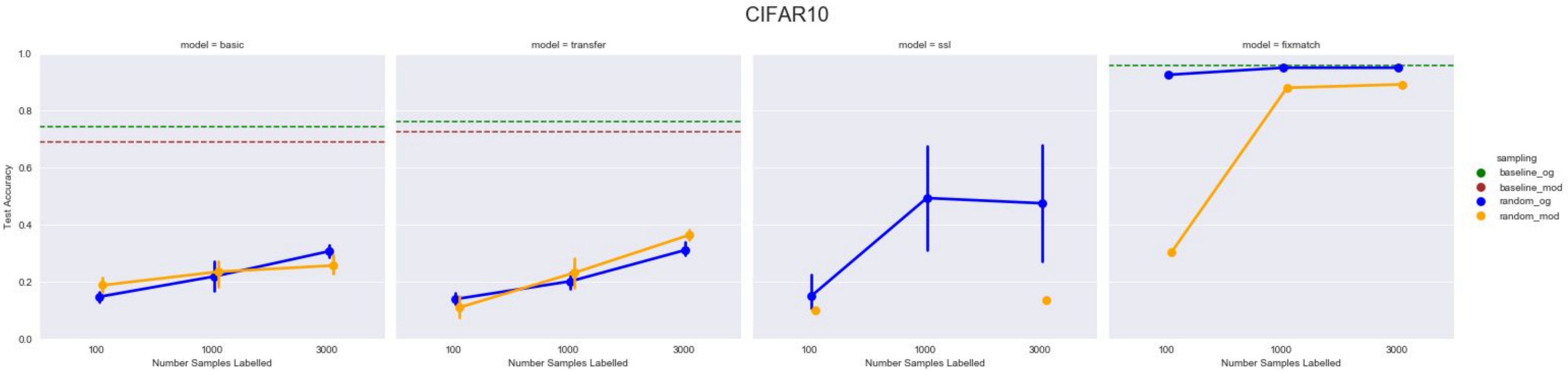


Active Learning Sampling Strategies

- **Pool-Based sampling:** there is a large pool of unlabelled data → we use the unlabelled training set
- Most informative instances are selected based on the **acquisition function** → first version with Random Sampling
- **Future Work** in acquisition functions:
  - Uncertainty Sampling used as informative measure
  - Acquisition function makes use of model's uncertainty
  - CNN with Dropout :
    - Bayesian Active Learning by Disagreement (**BALD**)

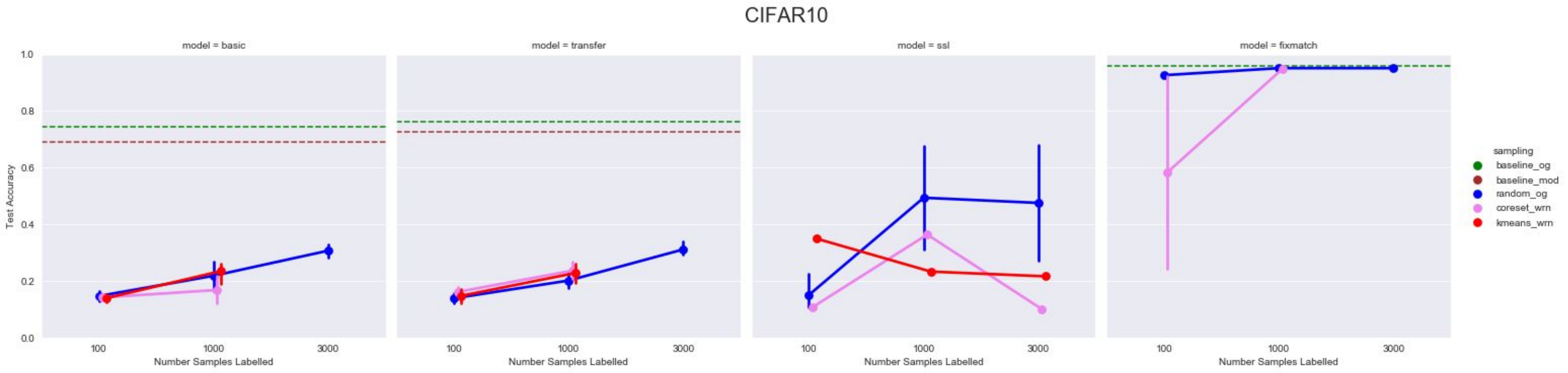
Training Results

Random sampling [original, modified]



Passive Learning (WRN50x2 features)

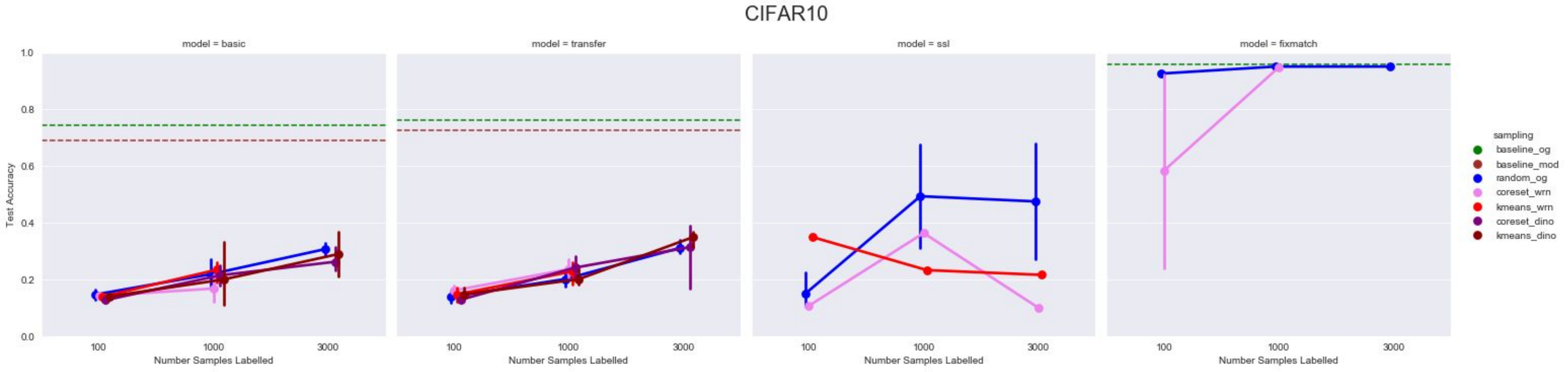
Clustered sampling [coreset, kmeans]



(some experiments missing)

Passive Learning (Vision Transformer features)

Clustered sampling [coreset, kmeans]

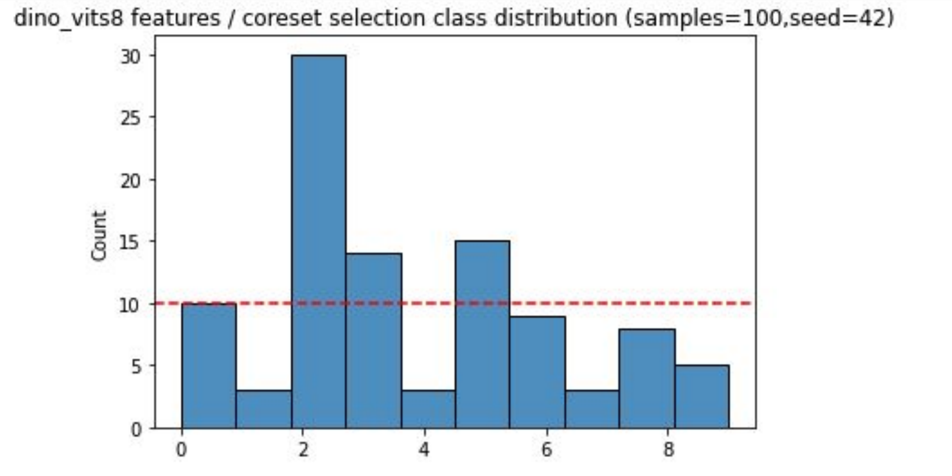


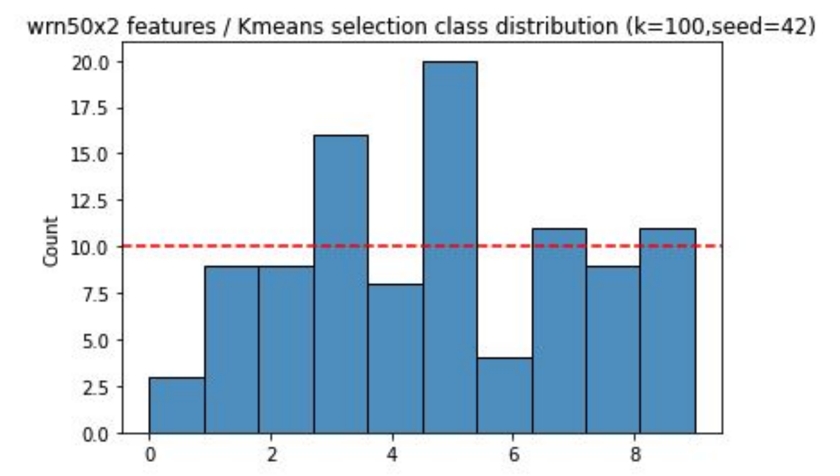
→ apparently not much better than random selection in most cases

(some experiments missing)

Clustering evaluation

We expected clustering (coreset/ kmeans) to at least do better than random?

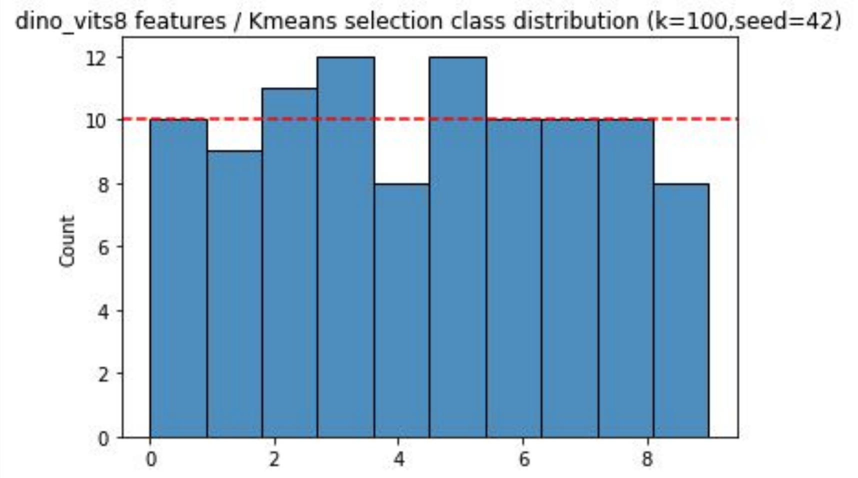


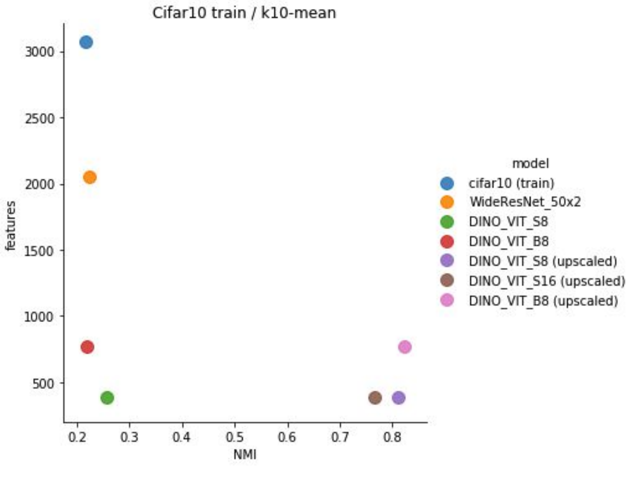


Possible Explanation: Unbalanced selection of class labels

Clustering evaluation

However, at least kmeans on the transformer weights *looked* promising ….

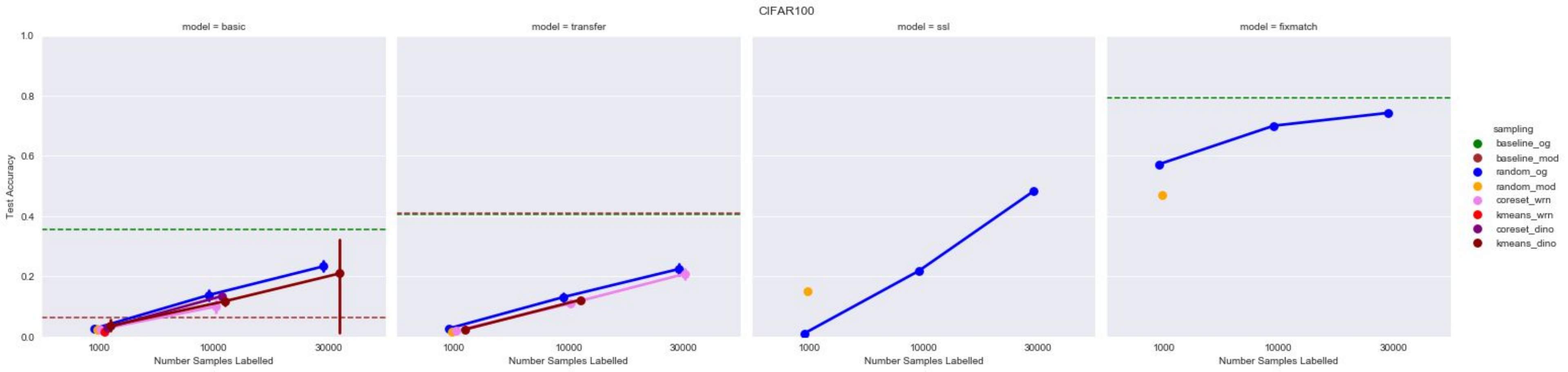




Seemingly balanced selection per class label, great NMI score (clustering assignments vs. true labels)

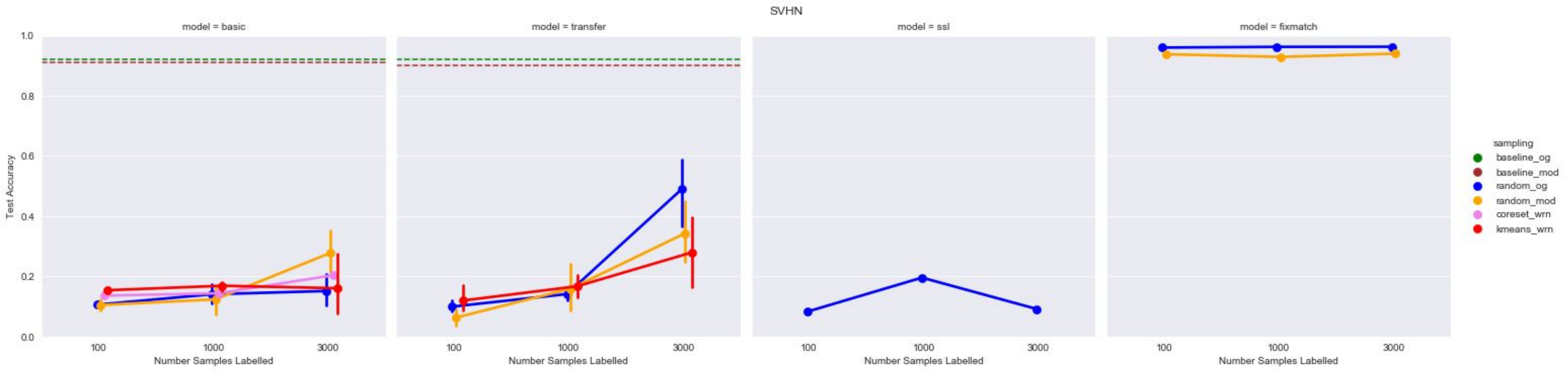
→ still didn't help notably with training

CIFAR100



(some experiments missing)

SVHN



(some experiments missing)

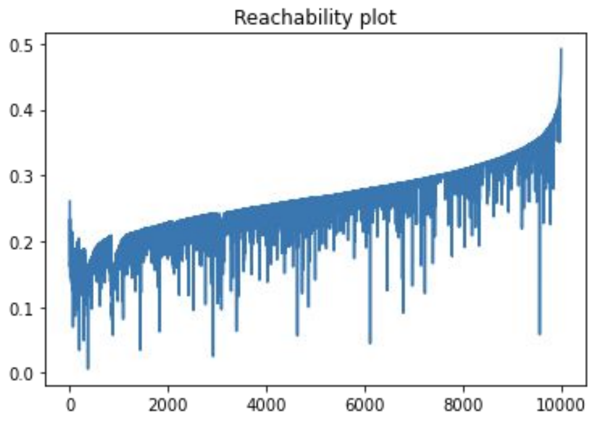
DBSCAN & OPTICS (didn’t work)

→ density based clustering does baldy with high-dimensional feature representations

What we tried (on 10k WRN50x2 representations of Cifar10):

- DBSCAN (default parameters):
  - *0 clusters*
  - *10.000 noise points*

- OPTICS (min-pts: 2)
  - Euclidean distance (301 clusters, 9339 npts)
  - Coside distance (950 clusters, 7748 npts) →



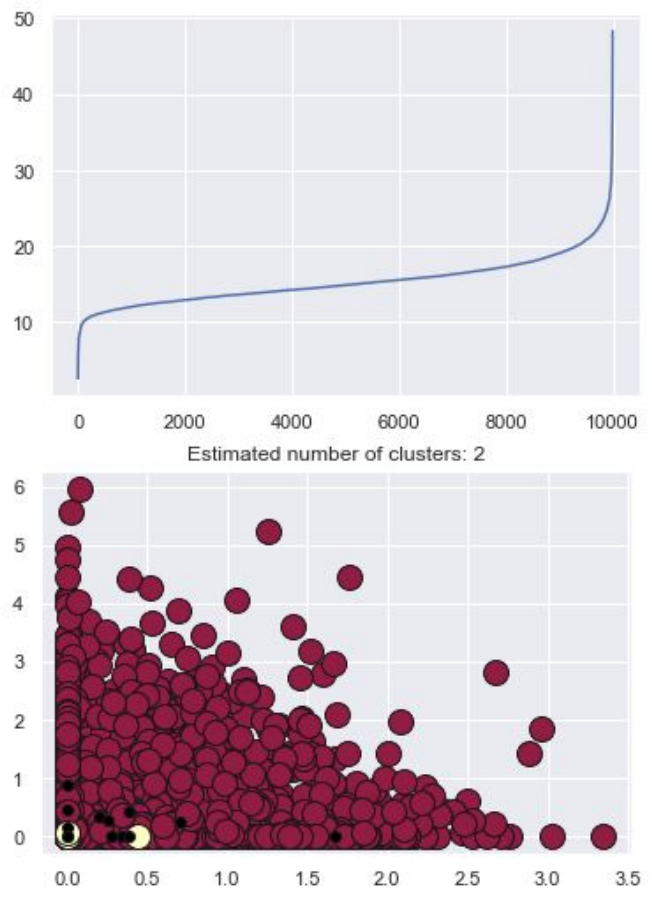
DBSCAN Elbow method

Elbow method:

- WRN50x2
- euclidean distance
- eps=35 min_points=2

Results

- Estimated number of clusters: 2
- Estimated number of noise points: 13
- Homogeneity: 0.001
- Completeness: 0.119
- V-measure: 0.001
- Adjusted Rand Index: 0.000
- Adjusted Mutual Information: 0.000
- Silhouette Coefficient: 0.432



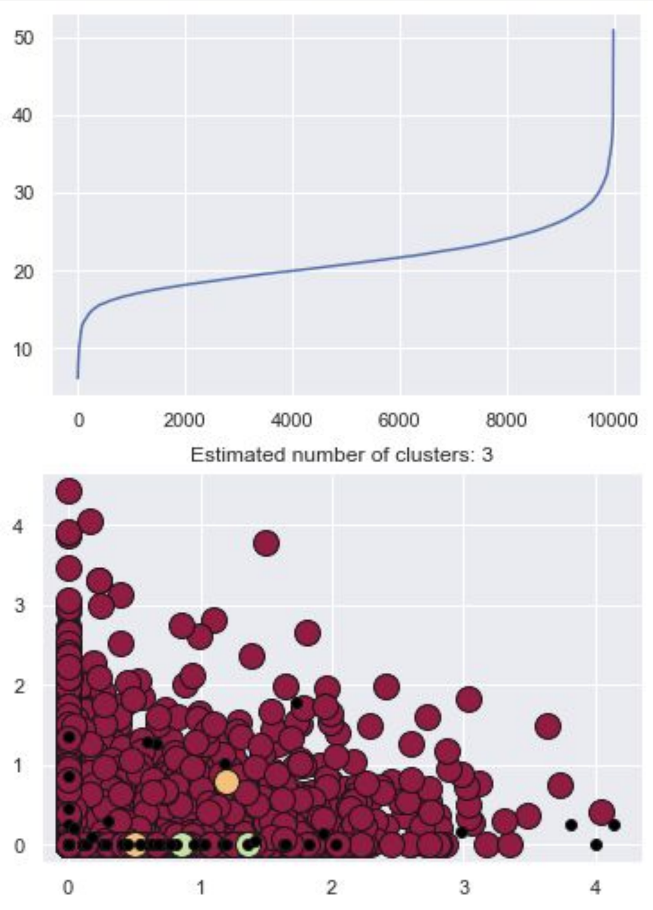
DBSCAN Elbow method

Parameters:

- RN50
- euclidean distance
- eps=35 min_points=2

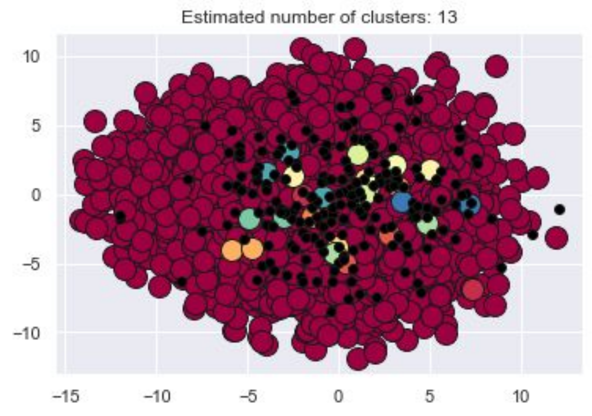
Results

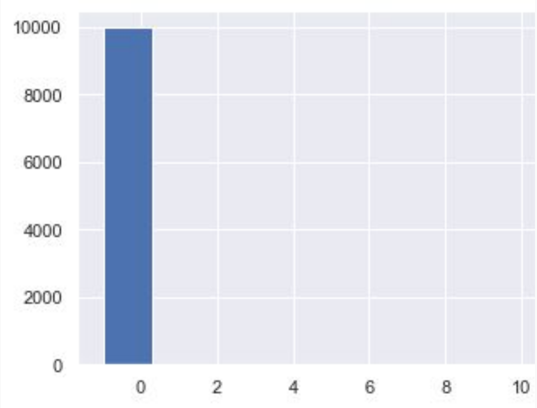
- Estimated number of clusters: 3
- Estimated number of noise points: 57
- Homogeneity: 0.001
- Completeness: 0.063
- V-measure: 0.002
- Adjusted Rand Index: 0.000
- Adjusted Mutual Information: 0.001
- Silhouette Coefficient: 0.219



DBSCAN on VITs8 representations

Parameters, euclidean dist.	№ of clusters	Noise pts.
Eps 100, min_sample 2	Clusters 1	Noise pts 0
Eps 85, min_sample 2	Clusters 1	Noise pts 0
Eps 71.8, min_sample 2	Clusters 10	Noise pts 219
Eps 50, min_sample 2	Clusters 323	Noise pts 8024

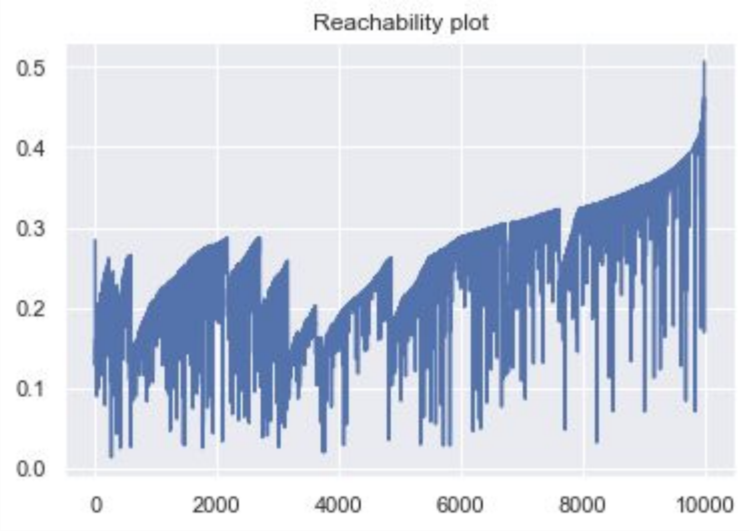




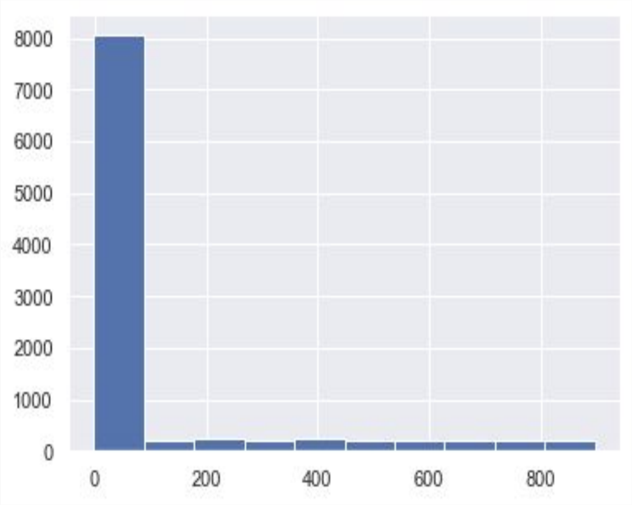
Cluster distribution (eps 71.8, minpts 2)

→ Tried cosine distance with elbow method. Seemingly much more reasonable clusterings, but very imbalanced actually

OPTICS on VITs8 representations



(Cosine distance)

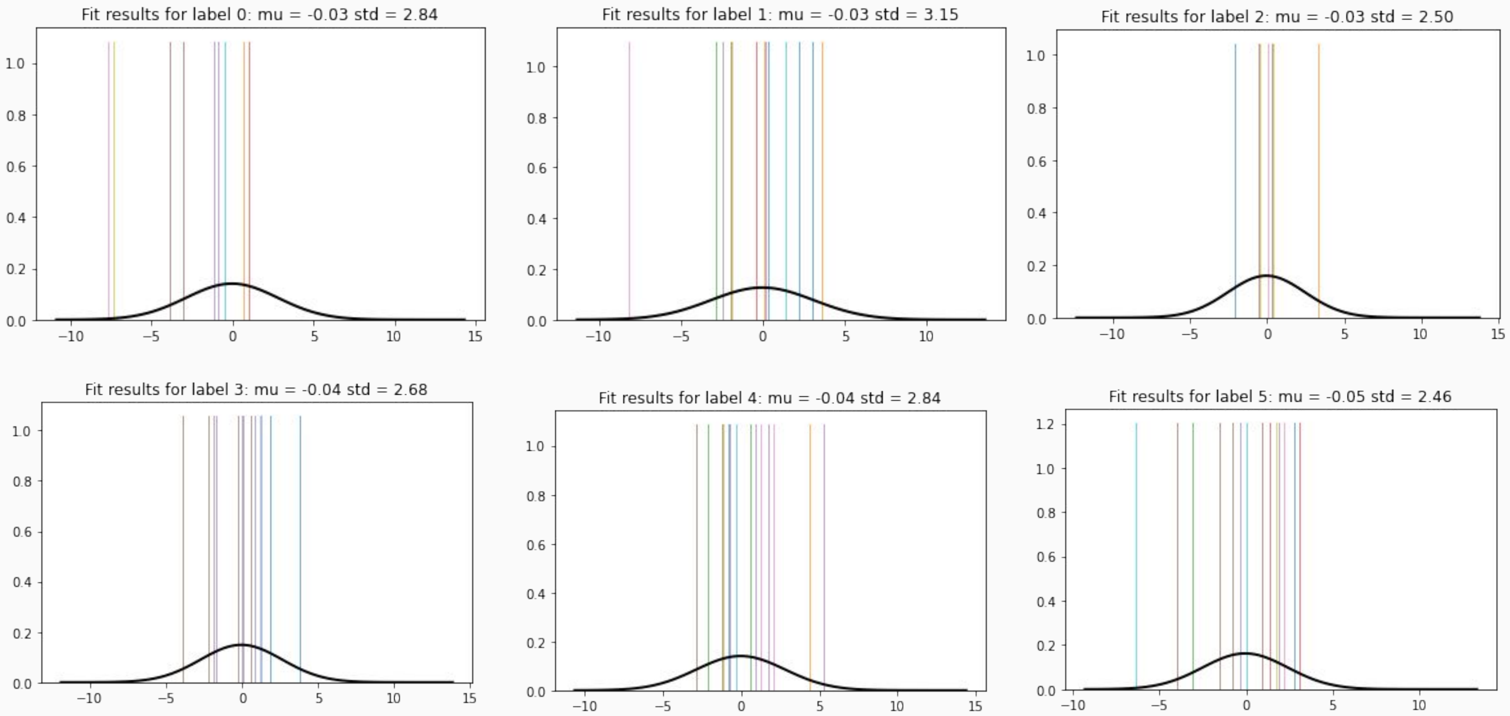


Class distribution, Cifar10

→ Best approach so far, although clusters still heavily skewed

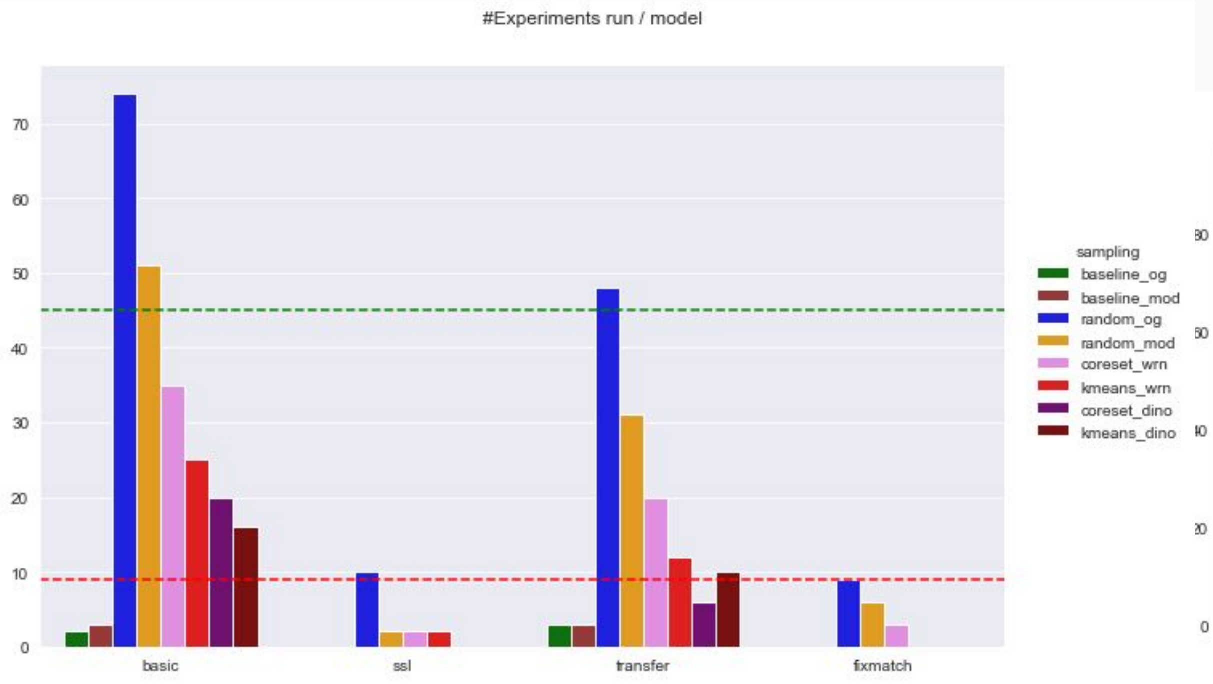
(No balanced sampling possible → no training)

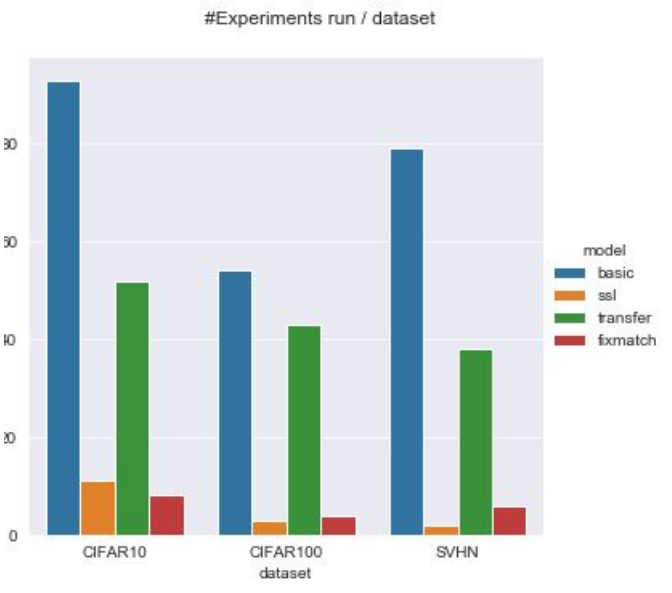
Why it's not working?



Try - other distances, - other [algorithms](), - other models

In total





basic ssl transfer fixmatch

Cifar10 Cifar100 SVHN

Feature Quality

→ Training performance dependent on 'quality' of selected representations.

→ Assumption: Using "non-descriptive" features does not help with selecting representative samples for passive/active learning.

Problem: How to know which model delivers quality features that we can use for selection/ clustering?

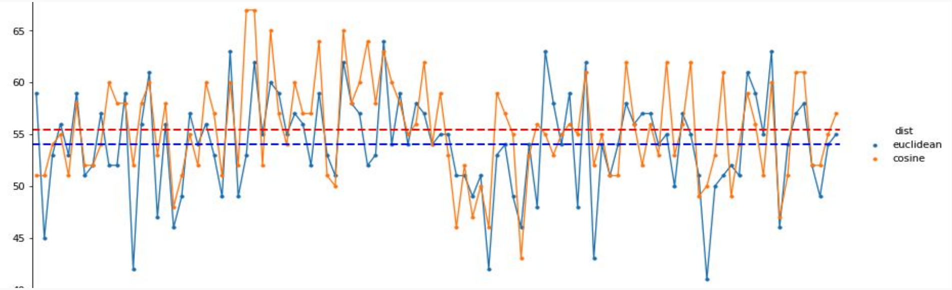
→ run K Nearest Neighbor Classifier on extracted features

(extract features of Cifar10 train and test, then use extracted_train to predict KNNC prediction for extracted_test)

Feature Quality

→ Average KNNC performance for conventional CNN models

Feature Extractor	extraction layer	euclidean distance	cosine distance	#features	batch size / total
raw cifar data	/	29.43	37.07	3072	100/ 10k
WideResNet_50x2	avg'pool	54.02	55.45	2048	100/ 10k
ResNet_50	avg'pool	56.78	58.00	2048	100/ 10k
VGG19	classifier fc1	51.89 (25.94)	53.24 (26.62)	4096	50/ 5k
VGG19	classifier fc2	52.39 (25.69)	52.98 (26.49)	4096	50/ 5k
GoogLeNet	avg'pool	45.45	48.19	1024	100 / 10k



(Example)
← k10NNC on WRN50x2 features
(Big Variance even across batches,
euclidean distance, cosine distance)

Feature Quality

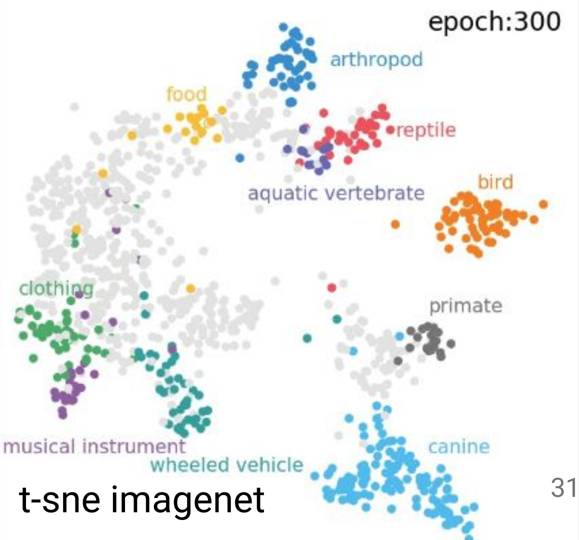
→ Can we do better?

→ *Emerging Properties in Self-Supervised Vision Transformers* (facebook)

the model learns a feature space that exhibits a very interesting structure. If we embed ImageNet classes using the features computed ***using DINO, we see that they organize in an interpretable way,*** *with similar categories landing near one another. This suggests that the model managed to connect categories based on visual properties*

→ extract features from VITs8 / VITs16 / VITb8
(s = ‘small’ = 23M params,   feature dimension 384,
  b = ‘big’   = 85M params, feature dimension 768)

Patchsizes 8x8 (bigger patches) or 16x16 (smaller patches)



t-sne imagenet

Feature Quality

→ After applying appropriate resizing to 244x244 (imagenet dimensions)

→ VIT yields very accurate NNC predictions (>90%, notably better than non VIT models and good NMI score for k-means clustering)

Feature Extractor	extraction layer	euclidean distance	cosine distance	#features	batch size / total
DINO_ResNet_50 (23M params)	avg'pool (our extr.)	62.06	64.70	2048	100/ 10k
DINO_VIT_S8	avg'pool (our extr.)	63.69	64.08	384	100/ **50k**
DINO_VIT_S8 (no norm)	avg'pool (our extr.)	59.86	60.12	384	100/ 50k
DINO_VIT_S16 (21M params)	avg'pool (our extr.)	62.27	62.75	384	100/ 50k
DINO_VIT_S16 (no norm)	avg'pool (our extr.)	60.97	62.37	384	100/ 50k
DINO_VIT_B8 (85M params)	avg'pool (our extr.)	59.68	60.05	768	100/ **25k**
DINO_VIT_S8 (upscaled)	avg'pool (our extr.)	96.06	96.22	384	100/ **50k**
DINO_VIT_S16 (upscaled)	avg'pool (our extr.)	92.90	92.76	384	100/ 10k
DINO_VIT_B8 (upscaled)	avg'pool (our extr.)	95.24	95.12	768	100/ 10k

(Scrum) Lessons learned

- Don’t underestimate time to set up environment (MLflow, CI runner)!
- Having team-member with overview (i.e., a scrum master) greatly helps with issue management and communication.
- Uniformity across code-classes helps with iterating features.
- Estimating time easier for smaller, well defined tasks.

Conclusion & Future Work

- Importance of sample selection shown, some approaches more impactful than others. Quality of sampling features are important
- SSL techniques very powerful in sparse settings
- Techniques show similar influence on all datasets

- Maybe compare with AutoEncoder features (even ones with inbuilt clustering loss or other passive/active learning helpers)