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

Development

GitLab Repository

branch

dev

MR

Planning

Training

MLflow

Results
Concept

“Sparse Label Learning”

→ only minority of training data labeled

→ Labeling is expensive!

→ Goal: Compare labelling strategies
Models

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

<table>
<thead>
<tr>
<th>Model</th>
<th>Architecture</th>
<th>Pretrained</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixmatch</td>
<td>WideResNet 28x2</td>
<td>No</td>
</tr>
<tr>
<td>SSL</td>
<td>WideResNet 50x2</td>
<td>No</td>
</tr>
<tr>
<td>Basic</td>
<td>WideResNet 50x2</td>
<td>No</td>
</tr>
<tr>
<td>Transfer</td>
<td>WideResNet 50x2</td>
<td>ImageNet</td>
</tr>
</tbody>
</table>
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.
FixMatch Approach: Pseudo-label + **Consistency regularization**

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

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>n</th>
<th>features</th>
<th>classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>40927</td>
<td>CIFAR_10</td>
<td>60,000</td>
<td>3073</td>
<td>10</td>
</tr>
<tr>
<td>41983</td>
<td>CIFAR-100</td>
<td>60,000</td>
<td>3073</td>
<td>100</td>
</tr>
<tr>
<td>41081</td>
<td>SVHN</td>
<td>99,289</td>
<td>3073</td>
<td>10</td>
</tr>
</tbody>
</table>

- 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]
Clustered sampling [coreset, kmeans]
Passive Learning (Vision Transformer features)

Clustered sampling [coreset, kmeans]

→ apparently not much better than random selection in most cases (some experiments missing)
We expected clustering (coreset/ kmeans) to at least do better than random? 

Possible Explanation: Unbalanced selection of class labels
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 baldly 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

<table>
<thead>
<tr>
<th>Parameters, euclidean dist.</th>
<th>№ of clusters</th>
<th>Noise pts.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eps 100, min_sample 2</td>
<td>Clusters 1</td>
<td>Noise pts 0</td>
</tr>
<tr>
<td>Eps 85, min_sample 2</td>
<td>Clusters 1</td>
<td>Noise pts 0</td>
</tr>
<tr>
<td>Eps 71.8, min_sample 2</td>
<td>Clusters 10</td>
<td>Noise pts 219</td>
</tr>
<tr>
<td>Eps 50, min_sample 2</td>
<td>Clusters 323</td>
<td>Noise pts 8024</td>
</tr>
</tbody>
</table>

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

→ 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

#Experiments run / model

#Experiments run / dataset

basic  ssl  transfer  fixmatch

Cifar10  Cifar100  SVHN
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 KNCC performance for conventional CNN models

<table>
<thead>
<tr>
<th>Feature Extractor</th>
<th>extraction layer</th>
<th>euclidean distance</th>
<th>cosine distance</th>
<th>#features</th>
<th>batch size / total</th>
</tr>
</thead>
<tbody>
<tr>
<td>raw cifar data</td>
<td>/</td>
<td>29.43</td>
<td>37.07</td>
<td>3072</td>
<td>100/ 10k</td>
</tr>
<tr>
<td>WideResNet_50x2</td>
<td>avg'pool</td>
<td>54.02</td>
<td>55.45</td>
<td>2048</td>
<td>100/ 10k</td>
</tr>
<tr>
<td>ResNet_50</td>
<td>avg'pool</td>
<td>56.78</td>
<td>58.00</td>
<td>2048</td>
<td>100/ 10k</td>
</tr>
<tr>
<td>VGG19</td>
<td>classifier fc1</td>
<td>51.89 (25.94)</td>
<td>53.24 (25.62)</td>
<td>4096</td>
<td>50/ 5k</td>
</tr>
<tr>
<td>VGG19</td>
<td>classifier fc2</td>
<td>52.39 (25.69)</td>
<td>52.98 (26.49)</td>
<td>4096</td>
<td>50/ 5k</td>
</tr>
<tr>
<td>GoogLeNet</td>
<td>avg'pool</td>
<td>45.45</td>
<td>48.19</td>
<td>1024</td>
<td>100 / 10k</td>
</tr>
</tbody>
</table>

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

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