Improving Fine-Tuning with Latent Cluster Correction

Cédric Ho Thanh
Prediction Science Laboratory, RIKEN, Japan
`cedric.hothanh@riken.jp`

Abstract

The formation of salient clusters in the latent spaces of a neural network (NN) during training strongly impacts its final accuracy on classification tasks. This paper proposes a novel fine-tuning method that boosts performance by improving the quality of these latent clusters, using the Louvain community detection algorithm and a specifically designed loss function. We present preliminary results that demonstrate that this process yields an appreciable accuracy gain for classical NN architectures fine-tuned on the CIFAR100 dataset.

1 Introduction

1.1 Background

Neural networks (NNs) are a family of machine learning models obtained by composition of simple, usually learnable operations called *layers*. In its simplest form, a NN f can be expressed as

$$f(x) = f_n\left(f_{n-1}\left(\cdots f_1(x)\cdots\right)\right), \tag{1.1}$$

where x is an input sample, and where $f_1, \ldots, f_k$ are f's layers, e.g. dense, convolutional, dropout, softmax, $\tanh$. Depending on the application, x can represent an abstract vector of numbers, an image, an electrocardiogram, etc. The present paper deals with *classifier NNs* which places an input x in a discrete category by assigning a predicted label $y_{\text{pred}} = f(x)$ to it.

A *latent representation* of x is any intermediate value $z = f_i\left(\cdots f_1(x)\cdots\right)$ for some $1 \leq i < n$, obtained while evaluating the NN on x. The space in which z lives, i.e. the output space of f_i, is called a *latent space*. Although the relationship between x and its predicted label $y_{\text{pred}} = f(x)$ can be well-understood (for example an image of a cat should have the label "cat"), the relationship between x, y, and the latent representations of x is more mysterious.

During training, a NN learns to extract features from the input x and represents them in its abstract latent spaces, shuffles them in a way that is (hopefully) beneficial by passing the representations through further transformation layers, and finally make a classification decision. However the precise way in which the NN decides to organize its latent spaces is not known and perhaps not even *knowable*. Indeed, popular NNs can have dozens of different latent spaces each spanning tens of thousands of dimensions!

Nonetheness, it is known that eventhough the dimension of a chosen latent space can be large, the actual (or *intrinsic*) number dimensions used by the NN can be much smaller, and some patterns emerge. Consider figure 1 representing the structures of several latent spaces [1] of an instance of ResNet-18 [HZRS16] trained on the CIFAR-10 dataset [Kri09].

[1]We shall use the expression "latent space" a little loosely to also refer to the *distribution* of latent representations in the latent space. So by "structure of the latent space" we actually refer to the patterns that emerge in the distribution of latent representations.

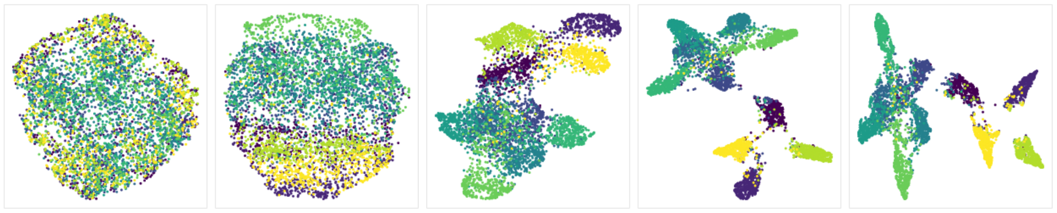

Figure 1: Latent representations of 5000 samples from the CIFAR-10 dataset passed through a pretrained instance of ResNet-18, after dimensionality reduction using UMAP [MHM20], coloured by class. The latent space sits deeper in the network as we go from left to right.

It is apparent that as we go deeper into the network (i.e. apply more and more layers to the input), the successive latent spaces become increasingly structured. Representations of samples from the same class, while initially scattered during the feature extraction phase, get increasingly clumped, or *clustered* together, while the clusters get increasingly *separated*.

With this in mind, a few natural questions emerge: 1. how does latent space structure (mainly separation and clustering) relate to the NN's performance? 2. how to measure latent space structure? 3. how to influence latent space structure during training to improve the NN's performance?

1.2 Related works

Separation and clustering in latent spaces has been investigated in several works.

First, [SMG$^+$21] introduce the *generalized discrimination distance* (GDV) which is perhaps the most direct way to measure separation and clustering of a set of samples. Through empirical study, they demonstrate that GDV is relevant for the study of latent spaces of multi-layer perceptrons (MLPs).

In [CCLS20], the authors place more emphasis in the study of the geometry (such as the dimension and radius) of the *object manifolds*, which are the manifolds defined by the latent representation of samples in a given class. They investigate the capacity of a NN to linearly separate these manifolds by computing a quantity known as the *system load*. Their empirical study shows that learning tends to increase separability and decrease the dimension and radius of object manifolds.

In [BLH$^+$19, LGO21], the authors try to actively influence the layout of the latent spaces by introducing the *graph smoothness loss*. That graph is constructed from the latent representations, where the vertices are the representation and the edges are weighted inversely exponentially by distance. By training a NN to exclusively minimize this graph smoothness loss, and then training a separate small classifier model (either a MLP or a support vector machine) on top of that, the authors achieve performances similar to NNs trained classically using cross-entropy loss. In [LGO21] specifically, the authors also use these separations methods for knowledge distillation [HVD15, RBK$^+$15] and to improve the robustness of classifier models against dataset corruption or adversarial attacks.

The works of [SPA$^+$21] go even further by designing NN architectures that are more amenable to latent separation. In their first approach, the authors create a model consisting of M parallel deep convolutional neural networks (DCNN), where M is the number of classes. Each is a binary classifier for the corresponding class tasked with clustering latent representations of its class away from samples from other classes. The second approach is more involved. First, a single DCNN is trained to maximize the *Grassmanian geodesic distance* (GGD) [ZZHJH18] between the subspaces of latent representation of each classes. Unlike traditional separation metrics that aim to minimize the distance between samples of the same class and/or maximize the distance between samples of different classes, the GGD merely encourages the network to allocate different linear latent subspaces for each class in such a way that the *principal angles* between these subspaces is as large as possible.

The aforementionned works, along with the present paper, only deal with classifier NNs. In [Dah18], the author applies these separation and clustering ideas to autoencoders (AEs). Instead of naively encouraging clusters in the latent space[2] of an AE, an extra NN, called *representation network* is attached to the encoder alongside the decoder. The role of the representation network is to embed latent representations to another space in which a clustering metric is applied.

1.3 Contribution

The present work follows the same idea as many of the papers above, namely encouraging clustering to improve performance. Our method, which we call *latent cluster correction* (LCC), uses a nearest-neighbor-based Louvain community detection algorithm to find these clusters (section 2.1), an assignment method to interpret their "intended" class (section 2.2), and a new loss function to encourage the formation of more salient and accurate clusters (section 2.3). Finally, we present some promising preliminary results in section 3.

1.3.1 Louvain clustering

Although the final latent structure of figure 1 seems simple enough to be solvable via an ensemble support vector machine scheme, one must keep in mind that the process of dimensionality reduction inevitably destroys a lot of information. A cluster in the 2D representation may in fact contain several clusters in the original high-dimensional latent space, and the relative position and shape of 2D clusters does not reflect the true latent geometry in general. Furthermore, empirical evidence suggests that a latent space may exhibit more clusters than there are true classes. For example, the right plot of figure 2 displays the latent representations (after dimensionality reduction) of a single class. Two significant clusters are visible. This separation originates from the presence of multiple (non-dimensionality-reduced) clusters among latent representations of this class. We can think of this phenomenon as the NN deciding to segregate samples in the same class based on some extracted features, only to ultimately assign the same label to them. Therefore, we argue that forcing the NN to make one cluster per class, as is the case in [BLH$^+$19, SPA$^+$21] among others, could be counterproductive.

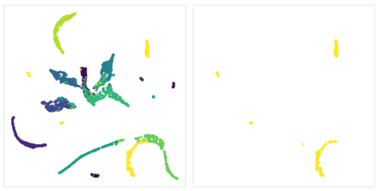

Figure 2: On the left: latent representations of 5000 samples from the Fashion MNIST dataset passed through a pretrained instance of AlexNet, after dimensionality reduction using UMAP, coloured by class. On the right: same, but only the representations belonging to the "Ankle boot" class are plotted.

In particular, the algorithm used for detecting the clusters should not have the expected number of clusters as a hyperparameter, which rules out k-means for example. In fact, the choice of algorithm and its hyperparameters should carry as few empirical biases as possible. For example, the clustering algorithm should be able to deal with non-linearly separable clusters, shouldn't produce outliers, should have few hyperparameters, should produce meaningful clusters even if the data is high-dimensional, etc. Table 1 compares a small selection of the most common clustering algorithms against our requirements. This list is far from exhaustive, and a more complete account can be found in [PHL04, AR14, MGL$^+$18].

Our method, kNN-Louvain, first constructs the k-nearest neighbors (kNN) graph of the sample set, and then applies the Louvain method to find *communities* in the graph, which correspond to clusters in sample space. Other flexible graph community detection algorithm exist and could be used, see [LF09] for a survey. Using these alternative methods is the subject of ongoing studies.

[2]When talking about AEs, the term "latent space" refers only to the space that sits between the encoder and decoder.

Table 1: A very small review of clustering algorithms.

Algorithm	Can deal with unknown number of clusters	Can deal with non-linearly separable clusters	Few parameters	Doesn't produce outliers
k-means	×	×	✓	✓
Gaussian mixture	×	×	✓	✓
BIRCH [ZRL96]	✓	×	✓	✓
CLIQUE [AGGR98]	✓	✓	✓	✓
OPTICS [ABKS99]	✓	✓	×	×
Spectral clustering [NJW01]	×	✓	×	✓
Affinity propagation [FD07]	✓	×	✓	✓
HDBSCAN [CMS13]	✓	✓	×	×
kNN-Louvain (used in this work)	✓	✓	✓	✓

2 The Louvain loss

The *Louvain loss* measures the quality and accuracy (*vis-à-vis* the true labels) of the clusters in a chosen latent space of a NN f. If $f(x) = f_n\left(f_{n-1}\left(\cdots f_1(x)\cdots\right)\right)$ and we selected an index $1 \leq i < n$, then the Louvain loss of f on a dataset or a batch (X, y) is computed in four steps:

1. compute the latent representations $Z = f_i\left(\cdots f_1(X)\cdots\right)$;
2. (section 2.1) compute the latent clusters of Z using our proposed *kNN-Louvain* method:
   (a) compute kNN graph G of Z;
   (b) find the Louvain communities of G which creates a new vector $\bar{y}$ of *Louvain labels*;
3. (section 2.2) find an optimal *one-to-many matching* between the labels in y and $\bar{y}$;
4. (section 2.3) based on this matching, find the *misclustered* latent samples and compute the loss term associated to each; the Louvain loss is the mean of all these terms.

2.1 The Louvain–Leiden algorithm

Given a graph $G = (V, E)$ with weighted edges, what is the best partition of V into *communities* so as to maximise the weighted connectivity within each community while minimizing it between communities? A popular approximation method known as the *Louvain algorithm* [BGLL08] seeks to maximize a surrogate objective called *modularity*, given by $Q = \frac{1}{2m}\sum_{i,j}\left(A_{i,j} - \frac{k_i k_j}{2m}\right)\delta_{i,j}$ where A is the adjacency matrix of G, $m = \frac{1}{2}\sum_{i,j} A_{i,j}$ is the sum of all the edges weight, $k_i = \sum_i A_{i,j}$ is the sum of the weights of all edges incident to node i, and $\delta_{i,j}$ is 1 if node i and j are in the same community, and 0 otherwise. The algorithm then produces a series of partitioned graphs $G = G^{(1)}, G^{(2)}, \ldots$ as follows:

1. (initialization) Start with the trivial partition where every node belongs to its own community.
2. (modularity optimization) For every edge $e = (i, j)$, we consider whether is it beneficial to remove node i from the community it currently belongs to and move it to the community that contains its neighbor j, say $C_j \subseteq V$. The benefit is measured by the *modularity gain* ΔQ, which is a quantity that depends on the connectivity within C_j, and between C_j and i. Node i is moved to the community of its neighbor j for which the modularity gain is the largest, provided it is positive. If all modularity gains are negative, node i does not move.
3. (aggregation) A new graph $G^{(n+1)}$ is built, where the nodes are the communities of $G^{(n)}$ produced by step 2. The adjacency matrix of $G^{(n+1)}$ is given by $A^{(n+1)}_{C,D} = \sum_{c\in C, d\in D} A^{(n)}_{c,d}$ if $C \neq D$, and $A^{(n+1)}_{C,C} = \frac{1}{2}\sum_{c,d\in C} A^{(n)}_{c,d}$.

These steps are repeated until no further improvement is possible, i.e. until a graph $G^{(N)}$ is produced for which no "node moving" in step 2. offers a positive modularity gain. At this point, nodes of

$G^{(N)}$ can be traced back to a partition of the nodes of G into disjoint communities. Unfortunately, the Louvain algorithm sometimes produces sparse communities, (i.e. for which the k_i''s are small even if $i \in C$). The *Leiden algorithm* [TWVE19] is an improvement that prevents this. It works by adding a step between step 1. and 2. that further partitions communities into subcommunities following a set complex rules. Then, at the begining of the next iteration, instead of starting with the trivial partition (that in which every node belongs to its own community), subcommunities of the same community are already grouped together. In the sequel, we refer to the Louvain–Leiden algorithm as simply "Louvain algorithm".

Let (Z, y) be a labeled dataset of latent representations, where $Z = (z_1, \dots, z_N)^t \in \mathbb{R}^{N \times d}$ (d is called the *latent dimension*), and $y \in \mathbb{N}^N$. The vector y is referred to as the vector of *true labels*.

The first step towards computing the Louvain loss of (Z, y) is to construct the k-nearest neighbor (kNN) graph G of Z, where k is a hyperparameter chosen in advance. The set of nodes of G is simply $\{z_i \mid 1 \leq i \leq N\}$, and nodes z_i and z_j are linked if z_i is among the kNNs of z_j or conversely.[3] In this case, the weight of the edge is the Euclidean distance $\|z_i - z_j\|$.

Then, running the Louvain algorithm on G produces a partition of Z into communities $C_1, C_2, \dots$, which gives rise to the vector of *Louvain labels* $\bar{y} \in \mathbb{N}^N$, where $\bar{y}_i = j$ if $z_i \in C_j$. It is expected that most of the time $y \neq \bar{y}$ even up to label permutation, and in fact, the two vectors might not even have the same number of distinct labels (see section 1.3.1 and figure 2).

2.2 True labels vs. Louvain labels

To establish a relationship between y and $\bar{y}$, we construct an *assignment* between the true labels and the Louvain labels. To disambiguate, let us denote the set of true labels (i.e. the distinct values of y) by $L_{\text{true}} \subseteq \mathbb{N}$, and the set of Louvain labels by $L_{\text{Louvain}} \subseteq \mathbb{N}$. Up to changing the name of the labels, we may assume that $L_{\text{true}} \cap L_{\text{Louvain}} = \varnothing$. An *one-to-many matching* between L_{true} and L_{Louvain} is simply a function $\alpha : L_{\text{Louvain}} \longrightarrow L_{\text{true}}$, possibly non bijective. We think of the Louvain label $l \in L_{\text{Louvain}}$ as "being matched" to the true label $\alpha(l)$. The assignment is optimal if the following value is maximized:

$$\sum_{t \in L_{\text{true}}} \left| \{z_n \mid y_n = t\} \cap \{z_n \mid \alpha(\bar{y}_n) = t\} \right|, \tag{2.1}$$

where $|-|$ refers to the cardinality of a set. In other words, the Louvain labels are assigned to true labels so as to minimize the overall discrepency between the set of samples with true labels $t \in L_{\text{true}}$ and the set of samples with Louvain label matched to t.

Such an assignment can be found by reformulating the objective function of equation (2.1) as a *discrete max-flow-max-weight problem*. A directed graph $F = (V, E)$ is constructed, where the underlying set of nodes is $V = \{\bot, \top\} \cup L_{\text{true}} \cup L_{\text{Louvain}}$. The edges and their weight and *capacity* are given by the following rules:

1. for all $t \in L_{\text{true}}$ and all $l \in L_{\text{Louvain}}$, there is an edge from t to l whose weight is $|\{z_n \mid y_n = t\} \cap \{z_n \mid \bar{y}_n = l\}|$ and whose capacity is 1;
2. for all $t \in L_{\text{true}}$, there is an edge from $\bot$ to t whose weight is 0 and whose capacity is infinite;
3. for all $l \in L_{\text{Louvain}}$, there is an edge from l to $\top$ whose weight is 0 and whose capacity is 1.

A discrete *flow* from $\bot$ to $\top$ is a function $g : E \longrightarrow \mathbb{N}$ such that:

1. for all node v except $\bot$ and $\top$, $\sum_{(w,v) \in E} g((w, v)) = \sum_{(v,w) \in E} g((v, w))$, i.e. the incoming flow of node v equals its outgoing flow;
2. for all edge $e \in E$, $g(e)$ is at most the capacity of e.

The flow is optimal if the quantity $\sum_e g(e) w_e$ is maximized, where w_e is the weight of edge e. In this case, it is possible to define an optimal one-to-many matching $\alpha : L_{\text{Louvain}} \longrightarrow L_{\text{true}}$ by matching $l \in L_{\text{Louvain}}$ to the de-facto unique true label $t \in L_{\text{true}}$ such that the edge (t, l) has a positive flow.

[3] An element is not considered to be one of its own neighbors.

2.3 The Louvain loss

At this stage, we dispose of a Louvain label vector $\bar{y}$ which regroups samples following an optimal kNN clustering scheme. We also constructed a many-to-one matching α which assigns a true label to each Louvain label.

We say that a sample z_i with true label y_i is *misclustered* if its Louvain label $\bar{y}_i$ is not assigned to its true label y_i, i.e. $\alpha(\bar{y}_i) \neq y_i$. The *Louvain loss* of the dataset of latent representations (Z, y) is defined as

$$\mathcal{L}_{\text{Louvain}} = \frac{1}{\sqrt{d}E} \sum_{z_i \text{ miscl.}} \left\| z_i - \frac{z_{j_1} + \cdots + z_{j_k}}{k} \right\| \tag{2.2}$$

where d is the latent dimension, E is the number of misclustered samples, and where $z_{j_1}, \ldots, z_{j_k}$ are the k nearest neighbots of z_i that are correctly clustered and in the same true class as z_i, i.e. $y_{j_p} = y_i$ and $\alpha(\bar{y}_{j_p}) = y_i$ for all $1 \leq p \leq k$. If there exist less than k such neighbors, then the contribution of z_i is 0 instead.

2.4 The approximate Louvain loss

The whole process of computing the "true" Louvain loss is computationally expensive, see section 4.1. To partially alleviate this, we propose an approximate Louvain loss that is slightly faster to compute. First, for each cluster C, choose a random sample in C that is correctly clustered, if such a sample exists. The *approximate Louvain loss* is defined as

$$\tilde{\mathcal{L}}_{\text{Louvain}} = \frac{1}{\sqrt{d}E} \sum_{z_i \text{ miscl.}} \| z_i - \bar{z}_i \| \tag{2.3}$$

where $\bar{z}_i$ is the closest target to z_i that is in the same true class as z_i. If such a target does not exist, the contribution of z_i is 0 instad.

3 Experiments

3.1 Setup

This preliminary study focuses on three model architectures, varying from small to somewhat large:

Model		Nb. of weights
TinyNet	[HWZ+20, LN21]	$\approx 2.06 \times 10^6$
ResNet-18	[HZRS16]	$\approx 11.7 \times 10^6$
VGG-11	[SZ15]	$\approx 132.9 \times 10^6$

The task at hand is to fine-tune them on the CIFAR-100 dataset [Kri09] using LCC.

The selected layers for latent cluster correction were either the last dense layer (which outputs the logits, also called the classifier *head*), or the second to last trainable layer. The Louvain loss weight w ranged over $\{1, 10^{-2}, 10^{-4}\}$. In all cases, the number of neighbors considered for clustering was $k = 2$, and training started with a *warmup* epoch, where the Louvain loss was not applied. Lastly, we studied the case where LCC is applied at every epoch (post-warmup) versus the case where it is applied only every five epochs.

Every training was performed using the Adam optimization algorithm [KB17] with a fixed learning rate of 5×10^{-5} and no weight decay.

3.2 Results

The best parameters and results are summarized in the tables below. The first row of every table corresponds to the baseline, which has been fine-tuned without LCC.

These results reveal that TinyNet and VGG-11, respectively the smallest and biggest of the three models, benefited little from LCC on average (with one notable exception for VGG-11). ResNet-18, which had highest baseline accuracy of the three, however, did consistently benefit from LCC.

Fine-tuning results for TinyNet

w	Int.	Layer	Acc.	Gain
			70.57%	
1	1	2nd. to last	69.13%	−1.44%
1	1	Head	70.86%	+0.29%
1	5	2nd. to last	70.75%	+0.18%
1	5	Head	70.89%	+0.32%
10^{-2}	1	2nd. to last	70.56%	−0.01%
10^{-2}	1	Head	70.57%	+0.0%
10^{-2}	5	2nd. to last	70.73%	+0.16%
10^{-2}	5	Head	70.98%	+0.41%
10^{-4}	1	2nd. to last	70.58%	+0.01%
10^{-4}	1	Head	70.84%	+0.27%
10^{-4}	5	2nd. to last	70.79%	+0.22%
10^{-4}	5	Head	71.04%	+0.47%

Fine-tuning results for VGG-11

w	Int.	Layer	Acc.	Gain
			67.82%	
1	1	2nd. to last	70.19%	+2.37%
1	1	Head	68.72%	+0.9%
1	5	2nd. to last	69.7%	+1.88%
1	5	Head	68.85%	+1.03%
10^{-2}	1	2nd. to last	67.35%	−0.47%
10^{-2}	1	Head	68.6%	+0.78%
10^{-2}	5	2nd. to last	66.67%	−1.15%
10^{-2}	5	Head	66.36%	−1.46%
10^{-4}	1	2nd. to last	68.62%	+0.8%
10^{-4}	1	Head	67.74%	−0.08%
10^{-4}	5	2nd. to last	66.99%	−0.83%
10^{-4}	5	Head	67.82%	+0.0%

Fine-tuning results for ResNet-18

w	Int.	Layer	Acc.	Gain
			73.21%	
1	1	2nd. to last	75.71%	+2.5%
1	1	Head	75.34%	+2.13%
1	5	2nd. to last	77.22%	+4.01%
1	5	Head	76.76%	+3.55%
10^{-2}	1	2nd. to last	75.45%	+2.24%
10^{-2}	1	Head	76.29%	+3.08%
10^{-2}	5	2nd. to last	77.79%	+4.58%
10^{-2}	5	Head	78.19%	+4.98%
10^{-4}	1	2nd. to last	75.66%	+2.45%
10^{-4}	1	Head	75.35%	+2.14%
10^{-4}	5	2nd. to last	77.37%	+4.16%
10^{-4}	5	Head	77.48%	+4.27%

4 Discussion

4.1 Computational cost

Latent cluster correction comes at a steep computational cost. First, at the begining of each training epoch (after warmup), the correction targets of misclustered latent representations have to be computed, which entails evaluating the whole dataset and keeping the latent representations Z in memory.

1. Build a nearest neighbor index on Z to construct the kNN graph. Using the KD-tree or ball-tree method, and assuming that the latent dimension is large (as is often the case, sadly), this has a time complexity of $\mathcal{O}(N)$ on average, where N is the number of samples.
2. Run the Louvain algorithm on that graph. Each iteration of the algorithm has a complexity of $\mathcal{O}(kN)$, but the number of iterations cannot be known in advance. Since every iteration produces a smaller graph, in the worst case, N iterations are needed for the algorithm to converge, which places the worst-case complexity at $\mathcal{O}(kN^2)$. However, it has been observed that the algorithm usually converges very quickly, giving it a $\mathcal{O}(kN)$ empirical complexity [LF09, LF14].
3. Solve a max-flow max-weight problem to match the Louvain labels to the true labels. In pactice, the cost of this is negligible compared to the other steps.
4. Then, for each true label $t \in L_{\text{true}}$, a new nearest neighbor index has to be build for the set of samples with that labels and that are also correctly clustered. If there are $n^{t,\checkmark}$ correctly clustered samples and $n^{t,\times}$ misclustered samples with true label t, then building the index has an average complexity of $\mathcal{O}(n^{t,\checkmark})$. The index then needs to be queried $n^{t,\times}$, which, again due to the high latent dimension, has a complexity of $\mathcal{O}(n^{t,\times})$ times. So across all labels, this step has a complexity of $\mathcal{O}(N)$.

It is important to note that for step 1., the whole dataset of latent prepresentations Z has to be in memory, which might be prohibitive for very large datasets, or when the latent dimension is extremely large. For step 4., technically, only the subdataset of samples with the same true label t has to be loaded at a given time in order to compute the targets of misclustered samples with true label t. All these costs compound if multiple layers are selected for simultaneous LCC.

On the other hand, if Z fits on the GPU, then the nearest neighbor index can be built and queried very quickly. This still adds a noticeable (albeit more palatable) overhead compared to regular training, however.

4.2 Choice of latent space(s)

The efficacy of LCC is expected to be highly dependent on the choice of latent space(s). In general, early layers are designed to extract features form the input samples, and the associated latent spaces are not expected to contain meaningful clusters. On the other hand, deeper layers are dedicated to classification rather than feature extraction, and it is expected that class-segregated clusters are forming. A preliminary empirical analysis as in figures 1 and 2 can help refine the choice of latent space(s).

4.3 Orthogonal correction

As noted in [CCLS20], latent representations not only form clusters, but organize along *object manifolds* of lower effective dimension. A misclustered representation z thus sits in an incorrect manifold $M \subseteq \mathbb{R}^d$. To remedy this, the Louvain loss pulls z along a vector $v = z - \frac{z_{j_1} + \cdots + z_{j_k}}{k}$ pointing towards some neighbors of z (see section 2.3). It could be advantageous to first project v onto $(T_z M)^\perp$, the subspace orthogonal to the tangent space of M at z, in order to accelerate the separation of z from M. If z sits in a dense enough region of M, approximating the tangent space $T_z M$ can be achieved using principal component analysis, considering only the k' closest neighbors of z for a fairly large k'.

4.4 ImageNet

The ImageNet dataset [DDS+09] has become the *de-facto* benchmark for image classification. It is significantly more varied than CIFAR-100 which we considered in this study, but it is also much larger. Considering the computational burden outlined in section 4.1, fine-tuning on ImageNet is not yet practical. Various algorithmic optimizations are under consideration as of the time of writing.

5 Conclusion

This paper presents a fine-tuning approach that operates on latent spaces by improving the quality and accuracy (in the sense of the assignment of section 2.2) of latent clusters. The only assumption underpinning our approach is the existence of such clusters, i.e. that during training, a NN tends to represent similar samples in close proximity, and that high density regions mostly contain representations from the same class. This phenomenon has been observed in every study about the structure of latent spaces. Unlike some of these studies however, we do not make any assumption on the geometry, overall separability, or even the number of such clusters.

We propose a new procedure which we call *latent cluster correction* (LCC) that aims to improve the quality of these clusters. The procedure consists of two steps: first, we use the Louvain community detection algorithm to find these clusters, and then apply a "nearest desirable neighbor" correction loss, which we call the *Louvain loss*.

Our preliminary results show that, on average, LCC brings noticeable classification accuracy improvements, but at a significant computational cost. We expect ongoing optimization efforts to reduce this cost and bring models with larger latent dimensions (such as vision transformers) and larger datasets (such as ImageNet) within reach.

Acknowledgments and Disclosure of Funding

We would like to thank our anonymous reviewers for their constructive feedback. We would also like to thank Prof. Jun Seita and Dr. Dorothy Ellis for their helpful guidance and discussions. This research was supported by RIKEN Pioneering Project "Prediction for Science".

References

[ABKS99] Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel, and Jörg Sander. OPTICS: Ordering points to identify the clustering structure. *ACM SIGMOD Record*, 28(2):49–60, June 1999.

[AGGR98] Rakesh Agrawal, Johannes Gehrke, Dimitrios Gunopulos, and Prabhakar Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, pages 94–105, Seattle Washington USA, June 1998. ACM.

[AR14] Charu C. Aggarwal and Chandan K. Reddy, editors. *Data Clustering: Algorithms and Applications*. Chapman & Hall/CRC Data Mining and Knowledge Discovery Series. Chapman and Hall/CRC, Boca Raton, 2014.

[BGLL08] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008, October 2008.

[BLH+19] Myriam Bontonou, Carlos Lassance, Ghouthi Boukli Hacene, Vincent Gripon, Jian Tang, and Antonio Ortega. Introducing Graph Smoothness Loss for Training Deep Learning Architectures. In *2019 IEEE Data Science Workshop (DSW)*, pages 160–164, Minneapolis, MN, USA, June 2019. IEEE.

[CCLS20] Uri Cohen, SueYeon Chung, Daniel D. Lee, and Haim Sompolinsky. Separability and geometry of object manifolds in deep neural networks. *Nature Communications*, 11(1):746, February 2020.

[CMS13] Ricardo J. G. B. Campello, Davoud Moulavi, and Joerg Sander. Density-Based Clustering Based on Hierarchical Density Estimates. In David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Madhu Sudan, Demetri Terzopoulos, Doug Tygar, Moshe Y. Vardi, Gerhard Weikum, Jian Pei, Vincent S. Tseng, Longbing Cao, Hiroshi Motoda, and Guandong Xu, editors, *Advances in Knowledge Discovery and Data Mining*, volume 7819, pages 160–172. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[CNL11] Adam Coates, Andrew Ng, and Honglak Lee. An analysis of single-layer networks in unsupervised feature learning. In Geoffrey Gordon, David Dunson, and Miroslav Dudík, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 215–223, Fort Lauderdale, FL, USA, April 2011. PMLR.

[Dah18] Paras Dahal. Learning Embedding Space for Clustering From Deep Representations. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 3747–3755, Seattle, WA, USA, December 2018. IEEE.

[DDS+09] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, Miami, FL, June 2009. IEEE.

[Den12] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[EBVJVD+17] Babak Ehteshami Bejnordi, Mitko Veta, Paul Johannes Van Diest, Bram Van Ginneken, Nico Karssemeijer, Geert Litjens, Jeroen A. W. M. Van Der Laak, and the CAMELYON16 Consortium, Meyke Hermsen, Quirine F Manson, Maschenka Balkenhol, Oscar Geessink, Nikolaos Stathonikos, Marcory Crf Van Dijk, Peter Bult, Francisco Beca, Andrew H Beck, Dayong Wang, Aditya Khosla, Rishab Gargeya, Humayun Irshad, Aoxiao Zhong, Qi Dou, Quanzheng Li, Hao Chen, Huang-Jing Lin, Pheng-Ann Heng, Christian Haß, Elia Bruni, Quincy Wong, Ugur Halici, Mustafa Ümit Öner, Rengul Cetin-Atalay, Matt Berseth, Vitali Khvatkov, Alexei Vylegzhanin, Oren Kraus, Muhammad Shaban, Nasir Rajpoot, Ruqayya Awan, Korsuk Sirinukunwattana, Talha Qaiser, Yee-Wah Tsang, David Tellez, Jonas Annuscheit, Peter Hufnagl, Mira Valkonen, Kimmo Kartasalo, Leena Latonen, Pekka Ruusuvuori, Kaisa Liimatainen, Shadi Albarqouni, Bharti Mungal, Ami George, Stefanie Demirci, Nassir Navab, Seiryo Watanabe, Shigeto Seno, Yoichi Takenaka, Hideo Matsuda, Hady Ahmady Phoulady, Vassili Kovalev, Alexander Kalinovsky, Vitali Liauchuk, Gloria Bueno, M. Milagro Fernandez-Carrobles, Ismael Serrano, Oscar Deniz, Daniel Racoceanu, and Rui Venâncio. Diagnostic Assessment of Deep Learning Algorithms for Detection of Lymph Node Metastases in Women With Breast Cancer. *JAMA*, 318(22):2199, December 2017.

[FD07] Brendan J. Frey and Delbert Dueck. Clustering by Passing Messages Between Data Points. *Science*, 315(5814):972–976, February 2007.

[HBDB18] Patrick Helber, Benjamin Bischke, Andreas Dengel, and Damian Borth. Introducing Eurosat: A Novel Dataset and Deep Learning Benchmark for Land Use and Land Cover Classification. In *IGARSS 2018 - 2018 IEEE International Geoscience and Remote Sensing Symposium*, pages 204–207, Valencia, July 2018. IEEE.

[HBDB19] Patrick Helber, Benjamin Bischke, Andreas Dengel, and Damian Borth. EuroSAT: A Novel Dataset and Deep Learning Benchmark for Land Use and Land Cover Classification. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 12(7):2217–2226, July 2019.

[HVD15] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the Knowledge in a Neural Network, March 2015.

[HWZ$^+$20] Kai Han, Yunhe Wang, Qiulin Zhang, Wei Zhang, Chunjing Xu, and Tong Zhang. Model Rubik's Cube: Twisting Resolution, Depth and Width for TinyNets, December 2020.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, Las Vegas, NV, USA, June 2016. IEEE.

[KB17] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization, January 2017.

[KDW$^+$21] Alexander Kolesnikov, Alexey Dosovitskiy, Dirk Weissenborn, Georg Heigold, Jakob Uszkoreit, Lucas Beyer, Matthias Minderer, Mostafa Dehghani, Neil Houlsby, Sylvain Gelly, Thomas Unterthiner, and Xiaohua Zhai. An image is worth 16x16 words: Transformers for image recognition at scale. 2021.

[Kri09] Alex Krizhevsky. Learning Multiple Layers of Features from Tiny Images. 2009.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[Kuh10] Harold W. Kuhn. The Hungarian Method for the Assignment Problem. In Michael Jünger, Thomas M. Liebling, Denis Naddef, George L. Nemhauser, William R. Pulleyblank, Gerhard Reinelt, Giovanni Rinaldi, and Laurence A. Wolsey, editors, *50 Years of Integer Programming 1958-2008*, pages 29–47. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[LF09] Andrea Lancichinetti and Santo Fortunato. Community detection algorithms: A comparative analysis. *Physical Review E*, 80(5):056117, November 2009.

[LF14] Andrea Lancichinetti and Santo Fortunato. Erratum: Community detection algorithms: A comparative analysis [Phys. Rev. E **80** , 056117 (2009)]. *Physical Review E*, 89(4):049902, April 2014.

[LGO21] Carlos Lassance, Vincent Gripon, and Antonio Ortega. Representing Deep Neural Networks Latent Space Geometries with Graphs. *Algorithms*, 14(2):39, January 2021.

[LMW$^+$22] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A ConvNet for the 2020s. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11966–11976, New Orleans, LA, USA, June 2022. IEEE.

[LN21] Hugo Larochelle and Neural Information Processing Systems Foundation, editors. *34th Conference on Neural Information Processing Systems (NeurIPS 2020): Online, 6-12 December 2020.* Number 33 in Advances in Neural Information Processing Systems. Curran Associates, Inc, Red Hook, NY, 2021.

[MG07] Jiří Matoušek and Bernd Gärtner. *Understanding and Using Linear Programming*. Universitext. Springer, Berlin ; New York, 2007.

[MGL$^+$18] Erxue Min, Xifeng Guo, Qiang Liu, Gen Zhang, Jianjing Cui, and Jun Long. A Survey of Clustering With Deep Learning: From the Perspective of Network Architecture. *IEEE Access*, 6:39501–39514, 2018.

[MHM20] Leland McInnes, John Healy, and James Melville. UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction, September 2020.

[NJW01] Andrew Y. Ng, Michael I. Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. In *Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic*, NIPS'01, pages 849–856, Cambridge, MA, USA, 2001. MIT Press.

[PHL04] Lance Parsons, Ehtesham Haque, and Huan Liu. Subspace clustering for high dimensional data: A review. *ACM SIGKDD Explorations Newsletter*, 6(1):90–105, June 2004.

[RBK$^+$15] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. FitNets: Hints for Thin Deep Nets, March 2015.

[SHZ$^+$18] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, Salt Lake City, UT, June 2018. IEEE.

[SMG$^+$21] Achim Schilling, Andreas Maier, Richard Gerum, Claus Metzner, and Patrick Krauss. Quantifying the separability of data classes in neural networks. *Neural Networks*, 139:278–293, July 2021.

[SPA$^+$21] Ali Sekmen, Mustafa Parlaktuna, Ayad Abdul-Malek, Erdem Erdemir, and Ahmet Bugra Koku. Robust feature space separation for deep convolutional neural network training. *Discover Artificial Intelligence*, 1(1):12, December 2021.

[SWY$^+$15] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, Boston, MA, USA, June 2015. IEEE.

[SZ15] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition, April 2015.

[TWVE19] V. A. Traag, L. Waltman, and N. J. Van Eck. From Louvain to Leiden: Guaranteeing well-connected communities. *Scientific Reports*, 9(1):5233, March 2019.

[Unk98] Unknown. Semeion Handwritten Digit, 1998.

[VLW$^+$18] Bastiaan S. Veeling, Jasper Linmans, Jim Winkens, Taco Cohen, and Max Welling. Rotation Equivariant CNNs for Digital Pathology, June 2018.

[XRV17] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-MNIST: A Novel Image Dataset for Benchmarking Machine Learning Algorithms, September 2017.

[ZRL96] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. BIRCH: An efficient data clustering method for very large databases. *ACM SIGMOD Record*, 25(2):103–114, June 1996.

[ZZHJH18] Jiayao Zhang, Guangxu Zhu, Robert W. Heath Jr., and Kaibin Huang. Grassmannian Learning: Embedding Geometry Awareness in Shallow and Deep Learning, August 2018.