An Empirical Study on Clustering Pretrained Embeddings: Is Deep Strictly Better?

Anonymous Author(s)
Affiliation
Address
email

Abstract

Recent research in clustering face embeddings has found that unsupervised, shallow, heuristic-based methods—including k-means and hierarchical agglomerative clustering—underperform supervised, deep, inductive methods. While the reported improvements are indeed impressive, experiments are mostly limited to face datasets, where the clustered embeddings are highly discriminative or well-separated by class (Recall@1 above 90% and often near ceiling), and the experimental methodology seemingly favors the deep methods. We conduct an empirical study of 14 clustering methods on two popular non-face datasets—Cars196 and Stanford Online Products—and obtain robust, but contentious findings. Notably, deep methods are surprisingly fragile for embeddings with more uncertainty, where they underperform the shallow, heuristic-based methods. We believe our benchmarks broaden the scope of supervised clustering methods beyond the face domain and can serve as a foundation on which these methods could be improved.

Clustering—the process by which a set of items are semantically-partitioned into finite groups—has been employed in many domains of computing including machine learning, computer graphics, information retrieval, and bioinformatics. It has numerous applications including interpretability, compression, visualization, outlier detector, and zero-shot classification. While clustering methods have traditionally ingested raw features (e.g., pixels) as input, an alternative explored particularly in the face-verification literature is to use *deep embeddings*. Deep embeddings are latent-feature vectors discovered by neural networks [11, 22, 28, 29, 34, 35, 41, 43–46]. They form a natural space for clustering because they are learned with loss functions that encourage grouping of semantically-similar inputs [4, 6, 7, 13, 37, 38, 42] and explicit representation of task-relevant features [30, 31].

Face verification systems rely on large backbone networks that produce highly-discriminative embeddings of face images. The term *discriminative* refers to the network's ability to separate embeddings that belong to the same identity or class from those that do not, as measured by a distance function. Improvements to these systems come largely through increasing the available labeled data, but with an expensive annotation cost. To overcome the annotation challenge, recent work has successfully used clustering to pseudo-label [15, 28, 34, 41, 43, 44, 46]. Since the crux of the approach is the clustering step, as improved clustering leads to better pseudo-labels and thus improved embedding quality, research has shifted from unsupervised, shallow, heuristic-based clustering methods [3, 5, 8–10, 16, 17, 21–23, 25, 27, 29, 33, 35, 36, 40] to supervised, deep, inductive methods [11, 15, 28, 34, 41, 43–46], with claimed improvements upward of 20% over k-means [25] and hierarchical agglomerative clustering (HAC) [36], for example.

One might, and arguably should, expect the deep methods to generalize beyond the face domain to other visual domains, as an innumerable number of research articles attest to inter-domain generalization. Additionally, the prior work on clustering pretrained embeddings has begun to expand beyond the face domain to other datasets (e.g., DeepFashion [24]) showing similar, impressive results. However, the prior work has limitations. First, by focusing primarily on face datasets, the clustering

Submitted to 34th Conference on Neural Information Processing Systems (NeurIPS 2020). Do not distribute.

	Cars196			SOP		
	m	n	$\frac{m}{n}$	m	n	$\frac{m}{n}$
Backbone Train	$3,963$	49	81	$32,277$	$5,658$	6
Clustering Train	$4,091$	49	83	$27,265$	$5,658$	5
Validation	$4,058$	49	83	$32,734$	$5,658$	6
Test	$4,073$	49	83	$27,777$	$5,660$	5

Table 1: The dataset splits with number of instances, m, number of classes, n, and approximate number of instances per class, $\frac{m}{n}$, for Cars196 and SOP.

methods operate on embeddings that are highly discriminative and not representative of the many embedding spaces that could benefit from clustering. Deng et al. [7] show that both verification and Recall@1 [26] performance of the embeddings are well-above 90% and can even approach ceiling performance across a range of face datasets, including MegaFace [18]. We also confirm that the embeddings for DeepFashion [24]—one of the only non-face datasets explored—exhibit Recall@1 performance above 90%. Second, common face datasets (e.g., MS-Celeb1M [12] and MegaFace [18]) are deprecated and no longer publicly available, with progress reliant on using shared embeddings extracted from pretrained backbones. Third, similar to Musgrave et al. [26], we find methodological choices that may implicitly favor the recent, deep methods; these choices include the lack of a validation split and monitoring test performance for hyperparameter tuning and early stopping.

For the reasons above, we empirically studied methods for clustering pretrained embeddings. We focused on outlining an end-to-end, reproducible pipeline, including dataset curation, backbone training, and clustering. We then benchmarked 14 clustering methods—including two state-of-the-art, supervised, deep methods, GCN-VE [43] and STAR-FC [34]—on two popular, non-face datasets: Cars196 [20] and Stanford Online Products (SOP) [39]. We specifically chose to benchmark against Cars196 and SOP because they are: (1) popular in the embedding-learning literature [4, 26, 32], (2) extend beyond the face domain, and (3) produce embeddings that are significantly less discriminative.

Our results falsify conclusions that deep clustering methods strictly outperform shallow methods. On Cars196 and SOP, we show that the state-of-the-art, deep methods underperform methods such as k-means and HAC. We posit a relationship between Recall@1 performance of the embeddings and the benefits provided by deep clustering methods, where the improvements shown with deep methods only manifest for embedding spaces exhibiting strong retrieval performance. We affirm an additional, but unrelated conclusion that ℓ_2-normalization of embeddings prior to clustering leads to a stark improvement across all heuristic-based methods, exploiting the geometrical properties of embeddings learned with softmax cross-entropy [32].

1 Methodology

We consider two datasets for our experiments: Cars196 [20] and Stanford Online Products (SOP) [39]. Both datasets are partitioned into a *backbone train* split, used to train the backbone, a *clustering train* split, used only to train the deep clustering methods, a *validation* split, used for hyperparameter tuning and early stopping for the backbone and deep clustering methods, and a *test* split, used to fit the unsupervised clustering methods and evaluate all clustering methods. All splits have disjoint sets of classes. Table 1 has per-split statistics on the number of instances, number of classes, and approximate number of instances per class for both datasets.

We use a ResNet50 [14] for the backbone with an additional fully-connected layer added after global average pooling and prior to the classification head, which produces a 256D embedding. The model is trained with cosine softmax cross-entropy—based on findings from Scott et al. [32]—using the backbone-train split of each dataset, with Recall@1 monitored on the validation split for early stopping and hyperparameter tuning. A hyperparameter search was performed for each of the datasets. Additional details on training the backbone, including specifics on the architecture, data augmentation, and hyperparameters are included in Appendix E.1.

The clustering methodology varies between the supervised, deep methods (e.g., GCN-VE and STAR-FC) and the unsupervised, shallow methods (e.g., k-means), discussed below. Appendix C enumerates and details both classes of methods. Appendix E.2 contains additional methodological details, architecture details for GCN-VE and STAR-FC, and an enumeration of all hyperparameters and their tuned values across all clustering methods.

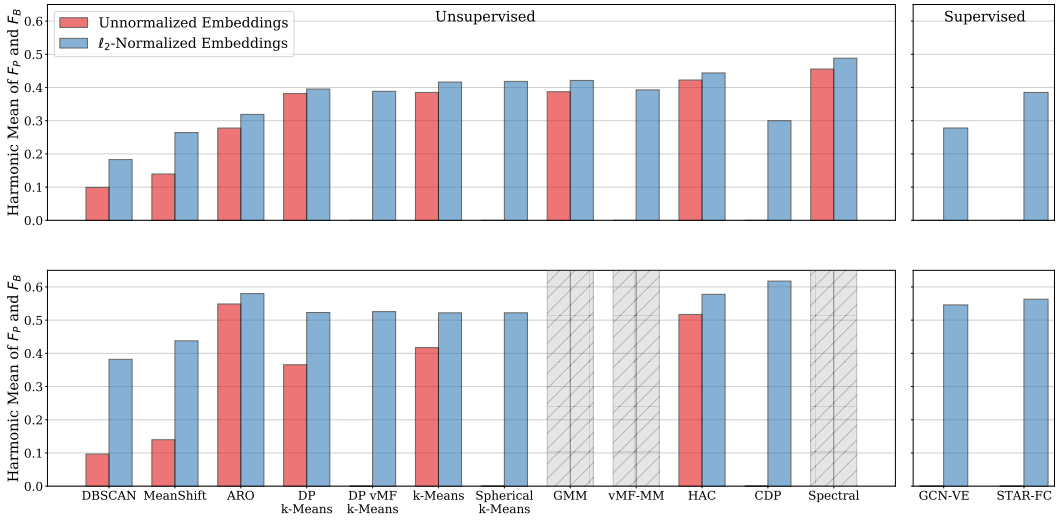

Figure 1: Harmonic mean of Pairwise (F_P) and BCubed (F_B) F-scores across clustering methods for Cars196 (top) and SOP (bottom). The left and right pane contain results for unsupervised and supervised clustering methods, respectively, and the red and blue bars indicate clustering of unnormalized and ℓ_2-normalized embeddings, respectively. GMM, vMF-MM, and Spectral could not be run on SOP due to runtime inefficiencies, indicated by the gray, hatched bars. The methods that omit a result for unnormalized embeddings assume ℓ_2-normalized embeddings, by default.

	Unnormalized Embeddings			ℓ_2-Normalized Embeddings		
	Clustering Train	Validation	Test	Clustering Train	Validation	Test
Cars196	0.67	0.76	0.76	0.69	0.80	0.79
SOP	0.72	0.70	0.73	0.75	0.74	0.77

Table 2: Recall@1 on the clustering-train, validation, and test splits of Cars196 and SOP for both unnormalized and ℓ_2-normalized embeddings.

For supervised clustering, we compute the 256D embeddings for the clustering-train, validation, and test splits. Methods are trained using the clustering-train embeddings from each dataset with the loss monitored on the validation split for early stopping and hyperparameter tuning. A hyperparameter search was performed for each of the datasets and we report results using the set of hyperparameter values associated with maximal clustering performance on the validation split. The model is then applied, inductively, on the test embeddings and final performance is reported.

In contrast, the unsupervised methods operate directly on the test embeddings. A hyperparameter search was also conducted for the unsupervised methods, however, we report results using the set of values associated with maximal clustering performance on the test split. We admit that optimizing for test performance appears to provide an unfair advantage to the unsupervised methods, however, the supervised methods: (1) have access to a split of data unavailable to the unsupervised methods, (2) have thousands of learnable parameters, and (3) have 12 and 16 hyperparameters for STAR-FC and GCN-VE, respectively, compared to at-most 4 hyperparameters for the unsupervised methods.

2 Experimental Results

Using the methodology outlined in Section 1, we conduct extensive experiments on Cars196 and Stanford Online Products (SOP) across 14 clustering methods. We measure clustering performance on the test split of both datasets. The clustering results focus on two metrics: Pairwise (F_P) [35] and BCubed (F_B) [1] F-scores. Because the metrics have different emphases—specifically, F_P emphasizes fidelity of larger clusters and F_B emphasizes fidelity of clusters proportional to their size—we report their harmonic mean when not reporting both, individually. Figure 1 shows clustering performance for Cars196 (top) and SOP (bottom) across 12 unsupervised methods and the

	Cars196		SOP	
	F_P	F_B	F_P	F_B
GCN-VE [43]	0.26	0.37	0.44	0.63
GCN-VE (ours)	0.22	0.37	0.47	0.65

Table 3: Comparison of Pairwise (F_P) and BCubed (F_B) F-scores for the open-source GCN-VE code and our reimplementation on Cars196 and SOP.

2 supervised methods. Appendix B contains tabulated results used to construct the figure, as well as additional metrics such as normalized mutual information. There are two key takeaways: (1) using ℓ_2-normalized embeddings (blue bars) consistently improves clustering performance, sometimes upward of 30%, and (2) the supervised, deep methods are outperformed by unsupervised, shallow methods and are not even among the top-3 performers, contrary to past work reporting consistent state-of-the-art performance. Among the unsupervised methods, there is no clear best-performing method, however, HAC performs strongly on both datasets. As a point of comparison, HAC outperforms GCN-VE and STAR-FC by 16% and 6% for Cars196, respectively, and 3% and 1% for SOP, respectively.

We posit a relationship between the embedding's discriminability, measured by Recall@1, and the likelihood of deep clustering methods outperforming baselines. Table 2 contains Recall@1 for the clustering-train, validation, and test splits of both datasets for unnormalized and ℓ_2-normalized embeddings. Consistent with Scott et al. [32], we find that ℓ_2-normalization strictly improves discrimination, as it removes intra-class variance. We emphasize that past conclusions on the deep methods are based on results from embeddings with Recall@1 greater than 0.9. Cars196 and SOP are more challenging benchmarks due to data limitations and thus exhibit Recall@1 less than 0.8. The clustering results on Cars196 and SOP highlight a downside of the deep methods, particularly that they become fragile in the presence of uncertainty (i.e., less discriminability) in the embedding space.

Due to the unexpected results for GCN-VE, we further investigated the method to verify our findings. We compared our reimplementation of GCN-VE to the open-source version, and Table 3 contains the results. We confirm that our reimplementation, within reasonable variance, matches the implementation from Yang et al. [43]. Additionally, Appendix D contains an ablation study of GCN-VE, where we find that one can recover the majority of GCN-VE's perform without any deep components.

3 Conclusions

Many of the innovations in supervised methods for clustering pretrained embeddings are tied to face verification. The methods were benchmarked primarily on face datasets and motivated by enlarging them via pseudo-labeling the vast amounts of unlabeled face data. Common face datasets (e.g., MS-Celeb1M [12] and MegaFace [18]) are deprecated and no longer publicly available, with progress reliant on using shared embeddings extracted from pretrained backbones. Additionally, the impressive improvements of supervised methods are bound to face embeddings, shown to have verification and Recall@1 performance well above 90% and in some cases near ceiling [7]. Finally, results of recent work may implicitly favor the novel, supervised methods based on methodological choices such as the lack of a validation split and hyperparameter tuning based on test performance.

Our goals in conducting an empirical study on clustering pretrained embeddings are to: (1) broaden the scope of supervised methods beyond faces, (2) present an end-to-end pipeline including backbone training along with results for benchmarking future methods, and (3) benchmark the robustness of supervised, deep methods on embeddings with less discriminability. We do so by presenting benchmarks on 14 clustering methods using two common embedding-learning datasets: Cars196 and SOP. Cars196 and SOP are not only from diverse visual domains, but have very different dataset statistics (Table 1) and embedding discriminability (Table 2) compared to common face datasets.

Our clustering results on Cars196 and SOP exemplify the fragility of deep clustering methods, where they underperform unsupervised, shallow methods such as HAC despite having thousands of learnable parameters and three times more hyperparameters. We attribute the surprising results to (1) our careful, systematic methodology, which corrects for past methodological choices such as not using a validation split and using the test split for hyperparameter tuning and early stopping, and (2) exploring datasets from diverse visual domains with both dataset statistics and embedding discriminability that differs significantly from the common face verification benchmarks. Additionally, we encourage practitioners working on embedding clustering to consider ℓ_2-normalization, as we find robust improvements to clustering, corroborating results from Scott et al. [32].

References

[1] Enrique Amigó, Julio Gonzalo, Javier Artiles, and Felisa Verdejo. A Comparison of Extrinsic Clustering Evaluation Metrics Based on Formal Constraints. *Information Retrieval*, 12(4), 2009.

[2] David Arthur and Sergei Vassilvitskii. k-Means++: The Advantages of Careful Seeding. In *ACM-SIAM Symposium on Discrete Algorithms*, 2007.

[3] Arindam Banerjee, Inderjit S. Dhillon, Joydeep Ghosh, and Suvrit Sra. Clustering on the Unit Hypersphere Using von Mises–Fisher Distributions. *Journal of Machine Learning Research*, 2005.

[4] Malik Boudiaf, Jérôme Rony, Imtiaz Masud Ziko, Eric Granger, Marco Pedersoli, Pablo Piantanida, and Ismail Ben Ayed. A Unifying Mutual Information View of Metric Learning: Cross-Entropy vs. Pairwise Losses. In *European Conference on Computer Vision*, 2020.

[5] Yizong Cheng. Mean Shift, Mode Seeking, and Clustering. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1995.

[6] Sumit Chopra, Raia Hadsell, and Yann LeCun. Learning a Similarity Metric Discriminatively, with Application to Face Verification. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2005.

[7] Jiankang Deng, Jia Guo, Niannan Xue, and Stefanos Zafeiriou. ArcFace: Additive Angular Margin Loss for Deep Face Recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2019.

[8] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *SIGKDD Conference on Knowledge Discovery and Data Mining*, 1996.

[9] Brendan J. Frey and Delbert Dueck. Clustering by Passing Messages Between Data Points. *Science*, 2007.

[10] Siddharth Gopal and Yiming Yang. von Mises–Fisher Clustering Models. In *International Conference on Machine Learning*, 2014.

[11] Senhui Guo, Jing Xu, Dapeng Chen, Chao Zhang, Xiaogang Wang, and Rui Zhao. Density-Aware Feature Embedding for Face Clustering. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2020.

[12] Yandong Guo, Lei Zhang, Yuxiao Hu, Xiaodong He, and Jianfeng Gao. MS-Celeb-1M: A Dataset and Benchmark for Large-Scale Face Recognition. In *European Conference on Computer Vision*, 2016.

[13] Raia Hadsell, Sumit Chopra, and Yann LeCun. Dimensionality Reduction by Learning an Invariant Mapping. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2006.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[15] Yue He, Kaidi Cao, Cheng Li, , and Chen Change Loy. Merge or Not? Learning to Group Faces via Imitation Learning. In *AAAI Conference on Artificial Intelligence*, 2018.

[16] Kurt Hornik, Ingo Feinerer, Martin Kober, and Christian Buchta. Spherical k-Means Clustering. In *Journal of Statistical Software*, 2012.

[17] Anil K. Jain. Data Clustering: 50 Years Beyond k-Means. In *Pattern Recognition Letters*, 2010.

[18] Ira Kemelmacher-Shlizerman, Steven M. Seitz, Daniel Miller, and Evan Brossard. The Megaface Benchmark: 1 Million Faces for Recognition at Scale. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[19] Thomas N. Kipf and Max Welling. Semi-Supervised Classification with Graph Convolutional Networks. In *International Conference on Learning Representations*, 2017.

[20] Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 3D Object Representations for Fine-Grained Categorization. In *IEEE International Conference on Computer Vision Workshops*, 2013.

[21] Brian Kulis and Michael I. Jordan. Revisiting k-Means: New Algorithms via Bayesian Non-parametrics. In *International Conference on Machine Learning*, 2012.

[22] Wei-An Lin, Jun-Cheng Chen, Carlos D Castillo, and Rama Chellappa. Deep Density Clustering of Unconstrained Faces. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[23] Ting Liu, Cory Jones, Mojtaba Seyedhosseini, and Tolga Tasdizen. A Modular Hierarchical Approach to 3D Electron Microscopy Image Segmentation. In *Journal of Neuroscience Methods*, 2014.

[24] Ziwei Liu, Ping Luo, Shi Qiu, Xiaogang Wang, and Xiaoou Tang. DeepFashion: Powering Robust Clothes Recognition and Retrieval with Rich Annotations. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[25] Stuart Lloyd. Least Squares Quantization in PCM. In *IEEE Transactions on Information Theory*, 1982.
[26] Kevin Musgrave, Serge Belongie, and Ser Nam Lim. A Metric Learning Reality Check. In *European Conference on Computer Vision*, 2020.
[27] Andrew Ng, Michael Jordan, and Yair Weiss. On Spectral Clustering: Analysis and an Algorithm. In *Advances in Neural Information Processing Systems*, 2001.
[28] Xuan-Bac Nguyen, Duc Toan Bui, Chi Nhan Duong, Tien D. Bui, and Khoa Luu. Clusformer: A Transformer Based Clustering Approach to Unsupervised Large-Scale Face and Visual Landmark Recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2021.
[29] Charles Otto, Dayong Wang, and Anil K. Jain. Clustering Millions of Faces by Identity. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2018.
[30] Karl Ridgeway. A Survey of Inductive Biases for Factorial Representation-Learning. *arXiv e-prints 1612.05299 cs.LG*, 2016.
[31] Florian Schroff, Dmitry Kalenichenko, and James Philbin. FaceNet: A Unified Embedding for Face Recognition and Clustering. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2015.
[32] Tyler R. Scott, Andrew C. Gallagher, and Michael C. Mozer. von Mises–Fisher Loss: An Exploration of Embedding Geometries for Supervised Learning. In *IEEE International Conference on Computer Vision*, 2021.
[33] David Sculley. Web-Scale k-Means Clustering. In *International Conference on World Wide Web*, 2010.
[34] Shuai Shen, Wanhua Li, Zheng Zhu, Guan Huang, Dalong Du, Jiwen Lu, and Jie Zhou. Structure-Aware Face Clustering on a Large-Scale Graph with 10^7 Nodes. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2021.
[35] Yichun Shi, Charles Otto, and Anil K. Jain. Face Clustering: Representation and Pairwise Constraints. In *IEEE Transactions on Information Forensics and Security*, 2018.
[36] Robin Sibson. SLINK: An Optimally Efficient Algorithm for the Single-Link Cluster Method. In *The Computer Journal*, 1973.
[37] Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical Networks for Few-Shot Learning. In *Advances in Neural Information Processing Systems 30*, 2017.
[38] Kihyuk Sohn. Improved Deep Metric Learning with Multi-Class N-Pair Loss Objective. In *Advances in Neural Information Processing Systems 29*, 2016.
[39] Hyun Oh Song, Yu Xiang, Stefanie Jegelka, and Silvio Savarese. Deep Metric Learning via Lifted Structured Feature Embedding. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.
[40] Julian Straub, Trevor Campbell, Jonathan P. How, and John W. Fisher. Small-Variance Nonparametric Clustering on the Hypersphere. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2015.
[41] Zhongdao Wang, Liang Zheng, Yali Li, and Shengjin Wang. Linkage Based Face Clustering via Graph Convolution Network. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2019.
[42] Kilian Q. Weinberger and Lawrence K Saul. Distance Metric Learning for Large Margin Nearest Neighbor Classification. *Journal of Machine Learning Research*, 2009.
[43] Lei Yang, Dapeng Chen, Xiaohang Zhan, Rui Zhao, Chen Change Loy, and Dahua Lin. Learning to Cluster Faces via Confidence and Connectivity Estimation. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2020.
[44] Lei Yang, Xiaohang Zhan, Dapeng Chen, Junjie Yan, Chen Change Loy, and Dahua Lin. Learning to Cluster Faces on an Affinity Graph. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2019.
[45] Jinxing Ye, Xioajiang Peng, Baigui Sun, Kai Wang, Xiuyu Sun, Hao Li, and Hanqing Wu. Learning to Cluster Faces via Transformer. *arXiv e-prints 2104.11502 cs.CV*, 2021.
[46] Xiaohang Zhan, Ziwei Liu, Junjie Yan, Dahua Lin, and Chen Change Loy. Consensus-Driven Propagation in Massive Unlabeled Data for Face Recognition. In *European Conference on Computer Vision*, 2018.
[47] Chunhui Zhu, Fang Wen, and Jian Sun. A Rank-Order Distance Based Clustering Algorithm for Face Tagging. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2011.

A Compute Resources

Each backbone was trained using a single NVIDIA Tesla T4, 16 CPUs, and 64 GB of RAM on Google Cloud Platform. Code was implemented using PyTorch v1.6.0 and Python 3.8.2 on Ubuntu 18.04.

For clustering, all methods had access to 16 CPUs, 96 GB of RAM, and a single NVIDIA Tesla T4, when appropriate. Code was implemented with PyTorch v1.6.0, scikit-learn v0.24.2, FAISS v1.7.1, and Python 3.8.2 on Ubuntu 18.04. Additionally, we note that both HAC and spectral clustering could be made faster by computing the nearest-neighbor subgraph on GPU instead of CPU.

B Cars196 and SOP Tabulated Results

The tables below contains tabulated clustering results for Cars196 and SOP. The F_P and F_B values are used to compute the harmonic means for ℓ_2-normalized embeddings presented in Figure 1.

Cars196							
	Adj. Rand Index	NMI	AMI	F_P	F_B	Time to Cluster (s)	Number of Clusters
DBSCAN	0.10	0.63	0.28	0.12	0.35	3	$2,425$
MeanShift	0.18	0.65	0.42	0.20	0.38	177	$1,407$
ARO	0.28	0.69	0.49	0.29	0.36	3	$1,415$
DP k-Means	0.37	0.69	0.61	0.38	0.41	10	403
DP vMF k-Means	0.36	0.67	0.61	0.38	0.40	12	155
k-Means	0.40	0.65	0.62	0.41	0.42	12	50
Spherical k-Means	0.40	0.65	0.62	0.41	0.42	12	50
GMM	0.40	0.66	0.62	0.41	0.43	62	50
vMF-MM	0.37	0.65	0.61	0.38	0.40	267	60
HAC	0.42	0.68	0.65	0.44	0.45	4	55
CDP	0.24	0.68	0.46	0.26	0.36	2	$1,686$
Spectral	**0.47**	0.70	**0.68**	**0.48**	**0.50**	99	50
GCN-VE	0.20	0.65	0.49	0.22	0.37	1140, 16	905
STAR-FC	0.36	**0.71**	0.53	0.37	0.40	341, 1	$1,624$

SOP							
	Adj. Rand Index	NMI	AMI	F_P	F_B	Time to Cluster (s)	Number of Clusters
DBSCAN	0.29	0.93	0.49	0.29	0.58	0.8	$17,801$
MeanShift	0.35	0.93	0.48	0.35	0.57	54	$18,096$
ARO	0.52	**0.94**	0.64	0.52	0.66	0.2	$9,915$
DP k-Means	0.45	0.93	0.59	0.45	0.63	13	$12,261$
DP vMF k-Means	0.45	0.93	0.59	0.45	0.63	3	$11,282$
k-Means	0.46	0.92	0.58	0.46	0.61	210	$7,750$
Spherical k-Means	0.45	0.93	0.59	0.45	0.62	242	$8,250$
HAC	0.52	0.93	0.64	0.52	0.65	4	$7,250$
CDP	**0.56**	**0.94**	**0.67**	**0.56**	**0.69**	0.1	$11,480$
GCN-VE	0.47	0.93	0.62	0.47	0.65	20, 13s	$8,808$
STAR-FC	0.50	0.93	0.61	0.50	0.65	8, 2s	$10,393$

Table 4: Clustering results for Cars196 and SOP. NMI, AMI, F_P, and F_B represent normalized mutual information, adjusted mutual information, Pairwise F-score, and BCubed F-score, respectively. The time to cluster for Cars196 and SOP are measured in seconds and minutes, respectively. The time to cluster for GCN-VE and STAR-FC contains the train time and test time separated by a comma. Boldface indicates the highest value for a metric.

C Related Work

C.1 Unsupervised Clustering

Clustering is normally characterized as an unsupervised learning task where methods leverage heuristics based on assumptions about the data-generation process or the structure of the data. Adopting the terminology of Jain [17], we discuss four common classes of heuristics: partition-based, density-based, hierarchical, and graph-theoretic.

Methods using partition-based heuristics [9, 16, 21, 25, 33, 40] have a representation of clusters and decide which data points belong to each cluster, in many cases, using a similarity or distance function. The most popular of these methods is k-means clustering [25] which estimates empirical cluster centers minimizing within-cluster variance. Other variants of k-means make assumptions about the geometry of the data, e.g., spherical k-means [16], make assumptions about the scale of the data, e.g., mini-batch k-means [33], or explore alternative formulations that remove the need for a predefined number of clusters, e.g., Dirichlet-process (DP) k-means [21, 40].

Density-based methods assume that clusters represent high-density regions in the feature space and are separated from other clusters by low-density regions. Popular density-based methods attempt to estimate cluster densities via expectation-maximization, e.g., the Gaussian mixture model (GMM) & the von Mises–Fisher mixture model (vMF-MM) [3], variational inference [10], Parzen windows, e.g., DBSCAN [8] & deep density clustering [22], and kernel density estimation, e.g., MeanShift [5].

Hierarchical methods form clusters by starting either with a singleton cluster containing all data points and iteratively splitting it (i.e., top-down or divisive) or with a cluster for each data point and iteratively merging them (i.e., bottom-up or agglomerative). The split or merge is decided greedily using a linkage rule until a criteria is satisfied such as the total number of clusters. We experiment with hierarchical agglomerative clustering (HAC) [36] and several popular linkage rules (single, complete, average, and Ward linkage), as well as approximate rank-order (ARO) [29] based on the rank-order linkage metric [47].

The final common class of heuristics use graph theoretics. These methods represent data points as a graph with edges weighted by pairwise similarities. One of the most popular methods, and the one we employ in our experiments, is spectral clustering [27]. Spectral clustering performs k-means clustering on the eigenvectors of the normalized Laplacian matrix constructed from the graph's affinity matrix.

C.2 Supervised Clustering

Recent research, instead of relying on heuristics, has proposed deep, supervised methods that inductively cluster. These methods rely on supervision that indicates which data points belong to the same cluster. By *inductive*, we refer to methods that learn a function, for example a function that predicts if two data points should be clustered together, which can then be applied to unseen data. This is in contrast to the unsupervised clustering methods which lack generalizability from one dataset to another. Generally, the idea is to leverage local and/or global structure in the feature space to learn what points belong to the same cluster or what pairs of points should be linked. Clusters can then be formed using simple algorithms such as connected-component labeling.

Consensus-driven propagation (CDP) [46] uses an ensemble of backbone networks to produce statistics across many k-nearest-neighbor affinity graphs and then trains a *mediator* network to predict links between data points from which clusters are formed. Wang et al. [41] improves link prediction by capturing structure in local affinity graphs directly with graph convolutional networks (GCNs) [19]. A series of GCN-methods followed that incorporate both local and global structure via density information [11] and multi-scale views [44], for example. Alternatives beyond GCNs have been proposed such as using self-attention via transformers [28, 45] and learning an agglomerative clustering policy with inverse reinforcement learning [15].

Our experiments include results from CDP, as well as two state-of-the-art GCN approaches: GCN-VE [43] and STAR-FC [34]. GCN-VE is a supervised approach that trains two GCNs. The first, GCN-V, estimates a confidence for each data point based on a supervised density measure. The second, GCN-E, constructs subgraphs based on the estimated confidences and predicts pairwise links. The links are used in a *tree-deduction* algorithm to construct final clusters. STAR-FC trains a single GCN

to predict pairwise links, but uses a structure-preserving sampling strategy to train the GCN with both local and global graph structure. During inference, links are predicted followed by *graph parsing and refinement* steps to construct clusters.

Consistent with past work, we do not train an ensemble and mediator for CDP, but rather use just the unsupervised *label-propagation* step to form clusters. Note that this removes all supervised components of CDP, which is why we classify it as an unsupervised method in Section 2.

D GCN-VE Ablation Study

	Cars196		SOP	
	F_P	F_B	F_P	F_B
Tree Deduction	0.13	0.36	0.44	0.63
+ ℓ_2-Norm	**0.25**	**0.37**	0.44	0.63
+ GCN-E	0.19	**0.37**	0.44	0.62
GCN-VE	0.22	**0.37**	**0.47**	**0.65**

Table 5: Ablation study comparing GCN-VE to simpler variants. The reported results are Pairwise (F_P) and BCubed (F_B) F-scores on Cars196 and SOP. Boldface indicates the highest value for a metric.

Due to the unexpected results of GCN-VE on Cars196 and SOP, we conducted an ablation study of the method to better understand which components contributed most to the final performance. In general, GCN-VE works by: (1) estimating a confidence for each embedding via the GCN-V graph convolutional network, (2) constructing subgraphs based on the estimated confidences, (3) estimating which edges in the subgraphs are valid connections via the GCN-E graph convolutional network, and (4) running a tree-deduction algorithm to form the final clusters.

We consider three ablated variants of GCN-VE, described below. All that is needed to form clusters is to run tree deduction on a k-nearest-neighbors subgraph with edges pruned by a distance threshold. We refer to this first ablation as *Tree Deduction*. Based on empirical analysis from Scott et al. [32], the embedding ℓ_2-norm is a measure of confidence, thus, we can replace the GCN-V network with the ℓ_2-norm of each embedding. Additionally, we can replace the GCN-E network simply by considering edges valid if they are connected to higher-confidence embeddings within a distance threshold, and run tree deduction as is. We refer to this second ablation as *Tree Deduction* + ℓ_2*-norm*. The third ablation, *Tree Deduction* + ℓ_2*-norm* + *GCN-E*, reintroduces GCN-E, but still uses ℓ_2-norm instead of the GCN-V network.

Table 5 measures the clustering performance of the three ablations against GCN-VE for Cars196 and SOP. Interestingly, one can recover the majority, if not all, of GCN-VE performance with the Tree Deduction + ℓ_2-norm variant, corroborating results from Scott et al. [32] on ℓ_2-norm conveying embedding confidence. Note that the Tree Deduction + ℓ_2-norm variant has no deep components. Additionally, the critical component of GCN-VE is not the GCN-E network, but GCN-V, as GCN-E adds no performance gain on top of Tree Deduction + ℓ_2-norm.

E Experimental Details

E.1 Backbone

The experimental details mimic Scott et al. [32] unless explicitly noted. For completeness, we recount the details below.

E.1.1 Architecture

The backbone network architecture is a ResNet50 [14]. The network is initialized using weights pretrained on ImageNet and all batch-normalization parameters are frozen. We remove the head of the architecture and add two fully-connected layers, with no activations, directly following the

global-average-pooling layer. The first fully-connected layer maps from 2048 units to 256 units, and the second fully-connected layer maps from 256 units to Y units, where Y is the number of classes in the backbone-training split of the dataset. The embedding dimensionality is thus 256 for all datasets. For Cars196 and SOP, $Y = 49$ and $Y = 5658$, respectively.

E.1.2 Dataset Augmentation

Cars196. Data augmentation during training includes: (1) resizing images to 256×256, (2) random jittering of brightness, contrast, saturation, and hue with factors in [0.7, 1.3], [0.7, 1.3], [0.7, 1.3], and [-0.1, 0.1], respectively, (3) cropping at a random location with random size between 16% and 100% of the input size, (4) resizing the final cropped images to 224×224, and (5) random horizontal flipping and z-score normalization. Data augmentation during validation and testing includes: (1) resizing images to 256×256, (2) center cropping to 224×224, and (3) z-score normalization. The mean and standard deviation for z-score normalization are the same values used for ImageNet. We match Boudiaf et al. [4] and sample batches randomly.

SOP. Data augmentation during training includes: (1) resizing images to 256×256, (2) cropping at a random location with random size between 16% and 100% of the input size with the aspect ratio randomly selected in [0.75, 1.33], (3) resizing the final cropped images to 224×224, and (4) random horizontal flipping and z-score normalization. Data augmentation during validation and testing includes: (1) resizing images to 256×256, (2) center cropping to 224×224, and (3) z-score normalization. The mean and standard deviation for z-score normalization are the same values used for ImageNet. We match Boudiaf et al. [4] and sample batches randomly.

E.1.3 Backbone Network Hyperparameters

Models are trained with SGD and either standard or Nesterov momentum. Based on results from Scott et al. [32], we use cosine softmax cross-entropy as the loss. The validation split's Recall@1 is monitored throughout training and if it does not improve for 15 epochs, the learning rate is cut in half. If the validation split's Recall@1 does not improve for 35 epochs, model training terminates early. The model parameters are saved for the epoch resulting in the highest validation Recall@1.

The inverse-temperature, β, is parameterized as $\beta = \exp(\tau)$ where $\tau \in \mathbb{R}$ is an unconstrained network parameter learned automatically via gradient descent. Unlike Scott et al. [32], τ is optimized using the same learning rate as all other network parameters.

The remaining hyperparameters for training the backbone network are:

- Learning rate
- ℓ_2 weight decay
- Momentum
- Nesterov momentum
- Initial value of τ

Hyperparameters were optimized with a grid search. The table below contains the hyperparameter values used for the model achieving the best validation Recall@1 for each dataset.

	Learning rate	ℓ_2 weight decay	Momentum	Nesterov momentum	Initial value of τ
Cars196	0.005	5×10^{-5}	0.0	False	0.0
SOP	0.005	5×10^{-5}	0.9	True	-2.773

Table 6: Backbone hyperparameter values for Cars196 and SOP.

E.2 Clustering

For clustering, the inputs are the 256D embeddings from the penultimate output of the trained backbone network. The embeddings are ℓ_2-normalized unless explicitly noted otherwise. To produce the embeddings for clustering, the images were augmented using the testing augmentation for each

dataset, which is: (1) resizing images to 256×256, (2) center cropping to 224×224, and (3) z-score normalization.

E.2.1 GCN-VE Details

We match Yang et al. [43] and use a one-layer graph convolutional network (GCN) [19] with mean aggregation [44] and ReLU activation for GCN-V, and a similar four-layer graph convolutional network for GCN-E. GCN-V maps the 512D output of the GCN through a fully-connected layer to 512 units, then through a PReLU activation, followed by a final fully-connected layer to a single output unit. The model is trained with mean-squared-error to match an empirically-estimated confidence based on the density of an embedding's neighborhood. GCN-E has identical structure to GCN-V except that it contains a four-layer GCN, the GCN output is 256D, and the first fully-connected layer has 256 units. GCN-E is trained with softmax cross-entropy to predict the likelihood that a pair of nearby embeddings belong to the same cluster. The GCN layers are initialized using Xavier-uniform for the weights and zero for the biases. The fully-connected layers are initialized using Kaiming-uniform for both weights and biases.

Both GCN-V and GCN-E are trained with SGD and standard momentum. If the validation split's loss does not decrease for 100 epochs, the learning rate is cut in half, and if it does not decrease for 250 epochs, model training terminates early. The model parameters are saved for the epoch resulting in the lowest validation loss.

E.2.2 STAR-FC Details

We match Shen et al. [34] and use a one-layer graph convolutional network (GCN) [19] with mean aggregation [44] and ReLU activation. The GCN output is 1024D which is then passed through a fully-connected layer to 512 units, then through a PReLU activation, followed by a final fully-connected layer to a single output unit. The model is trained with softmax cross-entropy to predict the likelihood that a pair of embeddings belong to the same cluster. The GCN layers are initialized using Xavier-uniform for the weights and zero for the biases. The fully-connected layers are initialized using Kaiming-uniform for both weights and biases.

STAR-FC is trained with SGD and standard momentum. If the validation split's loss does not decrease for 15 epochs, the learning rate is cut in half, and if it does not decrease for 35 epochs, model training terminates early. The model parameters are saved for the epoch resulting in the lowest validation loss.

E.2.3 Clustering Hyperparameters

For each clustering method, we list the hyperparameters and values for each dataset. All hyperparameters were optimized with a grid search.

k-Means with Random Initialization [25, 33]. The only hyperparameters are k, the number of clusters, and the minibatch size. A minibatch size of -1 indicates the batch size is equal to the full dataset size.

	k	Minibatch size
Cars196	40	-1
SOP	$8,250$	-1

Table 7: Hyperparameter values for k-means with random initialization.

k-Means with k-Means++ Initialization [2, 33]. The only hyperparameters are k, the number of clusters, and the minibatch size. A minibatch size of -1 indicates the batch size is equal to the full dataset size.

	k	Minibatch size
Cars196	50	-1
SOP	$7,750$	-1

Table 8: Hyperparameter values for k-means with k-means++ initialization.

Spherical k-Means with Random Initialization [16]. The only hyperparameter is k, the number of clusters.

	k
Cars196	45
SOP	$8,000$

Table 9: Hyperparameter values for spherical k-means with random initialization.

Spherical k-Means with k-Means++ Initialization [16]. The only hyperparameter is k, the number of clusters.

	k
Cars196	50
SOP	$8,250$

Table 10: Hyperparameter values for spherical k-means with k-means++ initialization.

Dirichlet-Process (DP) k-Means [21]. The hyperparameters are:

- λ, the cluster penalty
- Initialization strategy, either 'Global Centroid' or 'Random'
- Whether to do an online EM update

	λ	Initialization strategy	Online EM
Cars196	0.9	Global Centroid	True
SOP	0.85	Random	False

Table 11: Hyperparameter values for DP k-means.

Dirichlet-Process (DP) von Mises–Fisher (vMF) k-Means [40]. The hyperparameters are:

- λ, the cluster penalty
- Initialization strategy, either 'Global Centroid' or 'Random'
- Whether to do an online EM update

	λ	Initialization strategy	Online EM
Cars196	-0.55	Random	True
SOP	-0.38	Random	False

Table 12: Hyperparameter values for DP vMF k-means.

Gaussian Mixture Model (GMM). The hyperparameters are:

- k, the number of clusters
- Initialization strategy, either 'k-Means' or 'Random'
- Type of covariance, either 'Full' or 'Diagonal'

GMM was run only on Cars196 due to runtime inefficiencies.

	k	Initialization strategy	Covariance type
Cars196	50	k-Means	Diagonal

Table 13: Hyperparameter values for GMM.

von Mises–Fisher Mixture Model (vMF-MM) [3, 10]. The hyperparameters are:

- k, the number of clusters
- Initialization strategy, one of 'k-Means++' or 'Random', 'Random-Class', or 'Random-Orthonormal'
- Posterior type, either 'Hard' or 'Soft'

vMF-MM was run only on Cars196 due to runtime inefficiencies.

	k	Initialization strategy	Posterior type
Cars196	50	Random-Class	Soft

Table 14: Hyperparameter values for vMF-MM.

Hierarchical Agglomerative Clustering (HAC) [36]. The hyperparameters are:

- k, the number of clusters
- Affinity metric, one of 'Euclidean' or 'Cosine'
- Linkage criterion, one of 'Ward', 'Complete', 'Average', or 'Single'
- Number of neighbors for computing the nearest-neighbor affinity subgraph

	k	Affinity metric	Linkage criterion	Number of neighbors
Cars196	55	Euclidean	Ward	40
SOP	$7,250$	Euclidean	Ward	2

Table 15: Hyperparameter values for HAC.

Approximate Rank Order (ARO) [29]. The hyperparameters are:

- Number of neighbors for computing the nearest-neighbors subgraph
- Threshold on rank-order distances
- Distance metric, one of 'Euclidean' or 'Cosine'

Note that for ℓ_2-normalized embeddings, both Euclidean and cosine distance metrics produce identical results because the rank-order is unchanged.

	Number of neighbors	Threshold	Distance metric
Cars196	30	0.65	Euclidean
SOP	5	1.0	Euclidean

Table 16: Hyperparameter values for ARO.

Density-Based Spatial Clustering of Applications with Noise (DBSCAN) [8]. The hyperparameters are:

- ϵ, the maximum distance between two embeddings in the same neighborhood
- Minimum number of samples in neighborhood for core point
- Distance metric, either 'Euclidean' or 'Cosine'
- Whether a sparse matrix is used for nearest-neighbor subgraph

	ϵ	Minimum number of samples	Distance metric	Sparse matrix
Cars196	0.25	5	Cosine	False
SOP	0.66	2	Euclidean	False

Table 17: Hyperparameter values for DBSCAN.

MeanShift [5]. The hyperparameters are:

- Bandwidth for RBF kernel
- Minimum bin frequency
- Whether to cluster all embeddings including orphans

	Bandwidth	Minimum bin frequency	Cluster all embeddings
Cars196	0.75	5	True
SOP	0.65	1	False

Table 18: Hyperparameter values for MeanShift.

Spectral Clustering [27]. The hyperparameters are:

- k, the number of clusters
- Method for constructing the affinity matrix
- Number of neighbors for computing the nearest-neighbor affinity subgraph
- Number of components for the spectral embedding

Spectral clustering was run only on Cars196 due to runtime inefficiencies.

	k	Affinity	Number of neighbors	Number of components
Cars196	50	Nearest neighbor subgraph	20	50

Table 19: Hyperparameter values for spectral clustering.

Consensus-Driven Propagation (CDP) [46]. The hyperparameters are:

- Number of neighbors for computing the nearest-neighbor subgraph

- Threshold on cosine similarity in the nearest-neighbor subgraph for pruning edges
- Threshold step size
- Max cluster size

	Number of neighbors	Threshold	Threshold step size	Max cluster size
Cars196	2	0.6	0.01	320
SOP	5	-0.2	0.05	10

Table 20: Hyperparameter values for CDP.

Tree Deduction [43]. The hyperparameters are the number of neighbors for computing the nearest-neighbor subgraph and a threshold on cosine similarity for pruning edges during tree deduction.

	Number of neighbors	Threshold
Cars196	2	0.75
SOP	1	0.6

Table 21: Hyperparameter values for tree deduction.

Tree Deduction with Embedding ℓ_2-Norm Confidence. The hyperparameters are the number of neighbors for computing the nearest-neighbor subgraph and a threshold on cosine similarity for pruning edges during tree deduction.

	Number of neighbors	Threshold
Cars196	20	0.6
SOP	3	0.65

Table 22: Hyperparameter values for tree deduction with embedding ℓ_2-norm confidence.

Embedding ℓ_2-Norm Confidence + GCN-E + Tree Deduction. The hyperparameters are:

- Number of neighbors for computing the adjacency matrix (N)
- Threshold on cosine similarity for pruning edges in the adjacency matrix (τ_1)
- Ignore ratio (IR)
- Threshold on cosine similarity for pruning edges during tree deduction (τ_2)
- Number of units in the hidden layers of the network (H)
- Learning rate (LR)
- Momentum (MOM)
- ℓ_2 weight decay (ℓ_2)
- Dropout probability (D)

	N	τ_1	IR	τ_2	H	LR	MOM	ℓ_2	D
Cars196	60	0.0	0.1	0.6	512	0.01	0.9	1×10^{-5}	0.2
SOP	5	0.0	0.7	0.7	512	0.01	0.9	1×10^{-5}	0.0

Table 23: Hyperparameter values for embedding ℓ_2-norm confidence + GCN-E + tree deduction.

GCN-VE [43]. GCN-VE has two sub-networks, GCN-V and GCN-E. We present the hyperparameters associated with each sub-network independently.

The hyperparameters for GCN-V are:

- Number of neighbors for computing the adjacency matrix (N)
- Threshold on cosine similarity for pruning edges in the adjacency matrix (τ)
- Number of units in the hidden layers of the network (H)
- Learning rate (LR)
- Momentum (MOM)
- ℓ_2 weight decay (ℓ_2)
- Dropout probability (D)

	N	τ	H	LR	MOM	ℓ_2	D
Cars196	10	0.0	512	0.01	0.9	1×10^{-5}	0.0
SOP	2	0.0	512	0.1	0.9	1×10^{-5}	0.0

Table 24: Hyperparameter values for GCN-E.

The hyperparameters for GCN-E are:

- Number of neighbors for computing the adjacency matrix (N)
- Threshold on cosine similarity for pruning edges in the adjacency matrix (τ_1)
- Ignore ratio (IR)
- Threshold on cosine similarity for pruning edges during tree deduction (τ_2)
- Number of units in the hidden layers of the network (H)
- Learning rate (LR)
- Momentum (MOM)
- ℓ_2 weight decay (ℓ_2)
- Dropout probability (D)

	N	τ_1	IR	τ_2	H	LR	MOM	ℓ_2	D
Cars196	200	0.0	0.0	0.7	512	0.01	0.9	1×10^{-5}	0.0
SOP	5	0.0	0.7	0.8	512	0.01	0.9	1×10^{-5}	0.0

Table 25: Hyperparameter values for GCN-E.

STAR-FC [34]. The hyperparameters are:

- Number of neighbors for computing the adjacency matrix (N)
- Threshold on cosine similarity for pruning edges in the adjacency matrix (τ)
- Number of seed clusters (SC)
- Number of nearest clusters (NC)
- Number of random clusters (RC)
- Random node proportion (RNP)
- Prune threshold (P)
- Intimacy threshold (I)
- Number of units in the hidden layers of the network (H)
- Learning rate (LR)

- Momentum (MOM)
- ℓ_2 weight decay (ℓ_2)

	N	τ	SC	NC	RC	RNP	P	I	H	LR	MOM	ℓ_2
Cars196	10	0.0	4	6	10	0.9	0.15	0.6	512	0.1	0.9	1×10^{-5}
SOP	3	0.0	4	500	250	0.9	0.5	0.7	512	0.1	0.9	1×10^{-5}

Table 26: Hyperparameter values for STAR-FC.