Unsupervised Constrained Clustering with Contrastive Learning Semester Project Report

Gökberk Özsoy
Supervisor: Laura Manduchi

June 1^{st} - September 30^{th} 2022

1 Introduction

1.1 Constrained Clustering

Clustering is the process of grouping similar samples together based on their feature representations. The user often has preferences about if particular pairs should be in the same or different clusters. Constrained clustering deals with incorporating these preferences into the clustering objective, which can be considered as a semi-supervised concept.

DC-GMM [MCM+21] is a generative model which assumes that the data is generated from a latent space in the form of Gaussian Mixture Model that is conditioned by user's clustering preferences. If exist, the dataset labels can be used as clustering preferences, which boosts the performance of the model, but makes it supervised.

In this project, we will work on making DC-GMM fully unsupervised, which can automatically understand similarity notions hidden in the given dataset without any external help. The preceding VaDE [JZT+16] paper, which is a special case of DC-GMM without any constraints is also fully unsupervised. However, our method differs from it by having constraints as in DC-GMM, but generating them in a self-supervised way using contrastive learning.

1.2 Contrastive Learning

Contrastive learning is the idea of learning close low dimensional representations for similar samples, and far for dissimilar ones. In early versions of it, contrastive [CHL05] and triplet [SKP15] losses were introduced where the aim was simply reducing or increasing the difference between two same labeled or different labeled samples, respectively. This setting is supervised, but we are interested in unsupervised versions.

SimCLR [CKNH20], and Barlow Twins [ZJM+21] are two examples to it. Former aims to maximize agreement between two differently augmented versions of the same image, while minimizing agreement between all other samples within a batch. We will be inspired from this idea to develop our solution. Latter aims to make cross-correlation matrix close to identity, which is computed by using features of two differently augmented samples in a batch. Both of these rely on high quality augmentations which help the underlying model capture most of the possible variations beyond the given dataset.

1.3 Research Steps

During the course of the project, we will adhere the following order:

- Evaluating original DC-GMM model on natural image datasets, namely STL-10, and CIFAR-10.
- Following SimCLR batch structure to generate random cannot-links and augmentation must-links, and evaluating performance on a simple model comparable with original paper
- Crafting advanced architectures such as convolutional autoencoders, and exploring their behaviour in constrained clustering setting
- Pretraining convolutional autoencoders with contrastive loss to increase their feature extraction capacity

2 Tasks

In this section, we mention about the ideas and techniques for making DC-GMM model operable in fully unsupervised case. Each task uses preceding one's findings, and tries to extend it. For each task, we will explain the contribution, evaluate its effect on the performance, and comment on the results. We often explain the necessary changes on the original codebase to be complete, and easy-to-follow.

2.1 Task 1: Feature Extraction with ResNet + Unsupervised DC-GMM

In DC-GMM paper, among the datasets the model was evaluated on, STL-10 dataset is useful for our research for being high dimensional, multiple class natural image dataset. See A.1. Both VaDE and DC-GMM use a pretrained ResNet-50 for extracting features. Then, the inputs to the model become these features which are encoded and decoded by 4 layer fully connected layers.

We keep ResNet-50 for feature extraction, but change the way pairwise prior information matrix W is created. Originally, this is done by sampling predefined number of pairs randomly, and deciding each pair's relation (must or cannot) by dataset labels (DataGenerator class). We do this by generating random cannot links, and must-links from augmentations following SimCLR paper (UnsupervisedDataGenerator class). In more detail, given a batch of images, we randomly pair each sample exclusively with another one as cannot-link. For must-links, we sample two augmentation pipelines, and obtain two different augmented versions of each sample in the batch. We used a relatively simplified version of the strategy used in SimCLR paper, which is random cropping followed by color distortion. See A.2. We also see that performance drops when using only cropping or only color distortion.

We observed that for a batch size of N, the training becomes stable with only N constraints, even though N^2 constraints are possible. In addition, number of must-links should be lower than the cannot-links. For example, for a batch size of 128, the cannot links count is 128, and must-link count is 50. This means we use a subset of samples in a batch for must-links. Both observations are sound with the original DC-GMM training.

In Table 1, we present the detailed evaluation of the original and contrastive models. 'Latent Dim.' is encoded size by encoder of the given sample. '# of Constraints' is pair count for original model. For ours, it is just a number indicating randomly selected sample count, where the selected samples are used for random cannot-link and augmented must-link generation. Note that, no label information exists in our case. 'Link Type' is 0 for both cannot and must-links, 1 for only must-links. '# of Must-links' is the hyperparameter we used to control must-to-cannot ratio. 'VaDE' is whether we opt out constrained loss (loss2ac) from the DC-GMM objective. 'Exp. Cnt' is total number of runs for corresponding experiment.

Experiment							Test Performance			
Type	Latent Dim.	# of Constraints	Link Type	# of Must-links	VaDE	Exp. Cnt	Accuracy	NMI	ARI	Pretrain Acc.
Original	10	1k	0	-	×	2	70.2±3.2	60.2±0.2	51.8±1.3	67.4±1.9
Original	10	6k	0	-	×	3	85.2±0.5	73.3±0.6	70.3±0.7	70.4±2.3
Original	10	6k	0	-	✓	3	52.7±0.4	46.6±0.3	31.4±0.7	67.2±2.6
Original	10	6k	1	-	×	3	80.9±0.1	77.6±0.4	71.6±0.4	67.0±1.2
Contrastive	10	1k	-	50	×	1	68.6±0.0	64.1±0.0	54.1±0.0	66.6±0.0
Contrastive	10	6k	-	50	×	5	77.5±3.6	71.7±0.7	64.5±1.6	68.0±1.9
Contrastive	10	6k	-	128	×	2	72.4±0.6	70.4±0.3	61.0±1.5	67.8±0.1
Contrastive	25	6k	-	50	×	1	61.4±0.0	63.6±0.0	50.5±0.0	67.1±0.0

Table 1: Performance evaluated on original metrics for STL-10 test set. See 2.1 for detailed explanation of experiment settings.

2^{nd} experiment of Original is the one reported on the DC-GMM paper. Decreasing number of constraints to 1k narrows down the subset size the model can learn the similarities, hence the performance drops. Opting out constrained loss reduces performance, as the model has to process each sample independently. Only using must-links slight lowers the metrics as the model cannot exclude different samples clustered together wrongly. At this point, our extension works nicely finding itself slightly lower place under the upper bound of supervised training. As we assign cannot-links randomly, there is % 10 chance that same labeled samples are used to repel each other. In addition, for must-links, huge variation exists within each class that we cannot reach via solely using augmentations. As mentioned earlier, must-to-cannot ratio is important, which is apparent from increasing must-link count to 128. Number of constraints has the same effect with the original one, as expected.

After this successful first attempt, we need to state that this model is not fully unsupervised as we used pretrained ResNet-50 for feature extraction. It is trained with labeled natural image dataset ImageNet, hence the extracted features implicitly hold similarities, which eases the job of DC-GMM. In the next task, we will investigate this further, and try to find a solution.

2.2 Task 2: Convolutional Autoencoders + Unsupervised DC-GMM

In this task, we need to deal with two objectives at the same time: high quality feature extraction, and unsupervised constrained clustering. We identify the challenges as following: a) if the above two objectives will show accordance during training, b) a strong encoder is needed for quality feature extraction, but it comes with the cost of doubling the parameter size due to its decoder counterpart, c) deep autoencoders are either newly emerging or not practical for most of the cases due to scarce number of publications.

First of all, we try a simple architecture (two VGG layers with 32 and 64 filters, both having 3x3 kernels) tested to be working in Hearth Echo dataset in the original paper for both CIFAR-10 and STL-10 datasets. In Table 2, 'VGG 2' under 'Model' setting shows the results. The metrics are low, but better than random guessing, so they indicate that some learning is happening. Furthermore, our 'Contrastive' method from Task 1 still works with a tiny margin lower than the upper bound 'Original'. These similar performances mean that autoencoder architecture search needs to be addressed first. For both types, we observe that

Experiment						Test Performance			
Dataset	Type	Model	Latent Dim.	Pretrain	Exp. Cnt	Accuracy	NMI	ARI	Pretrain Acc.
CIFAR-10	Original	VGG 2	10	✓	5	20.7±1.5	8.0±1.5	4.3±1.3	23.6±0.7
CIFAR-10	Original	VGG 2	10	×	1	14.3±0.0	3.2±0.0	1.1±0.0	-
CIFAR-10	Original	VGG 2	50	✓	2	10.0±0.0	0.0±0.0	0.0±0.0	22.9±0.2
CIFAR-10	Original	VGG 6	10	✓	2	17.7±0.3	4.9±0.5	2.3±0.1	22.8±0.4
CIFAR-10	Contrastive	VGG 2	10	✓	1	19.7±0.0	9.7±0.0	5.3±0.0	23.2±0.0
STL-10	Original	VGG 2	10	✓	2	20.4±1.2	10.0±1.6	4.4±1.2	23.9±0.2
STL-10	Original	VGG 2	10	×	2	11.3±0.0	0.3±0.1	0.0±0.0	-
STL-10	Original	VGG 6	10	✓	1	19.2±0.0	8.4±0.0	3.3±0.0	23.7±0.0
STL-10	Contrastive	VGG 2	10	✓	2	19.2±0.2	7.5±0.6	3.4±0.5	22.7±0.8
STL-10	Contrastive	VGG 6	10	✓	3	20.2±1.2	10.7±0.5	4.0±0.5	21.8±0.9

Table 2: Performance evaluated on original metrics for STL-10 test set. 'Model' means if we used 2 or 6 VGG layers. 'Pretrain' means if we pretrain the autoencoder before DC-GMM training. See 2.1 for detailed explanation of other experiment settings.

the pretraining has high importance on final metrics. However, for the models which include pretraining, pretraning accuracy is still very low compared to Task 1. 2 VGG layers might be weak to understand natural images, thus we decide to increase model depth.

After following best practices such as replacing max pooling with strides, adding batch normalization [cif], and increasing depth carefully while observing reconstruction quality, we determine to work with a model with 6 VGG layers A.3. For how the reconstruction quality changes with varying latent dimensions on CIFAR-10, please refer to A.5. In Table 2, 'VGG 6' under 'Model' setting shows the results with this new model. As apparent, the final metrics are very low and similar to 2 VGG layers. Furthermore, we still did not see any improvement on pretrainig accuracy, hence no correlation exists between clustering accuracy and reconstruction quality. We understood this by visualizing the t-SNE applied 2-dimensional latent spaces for both pretrained ResNet-50 and our best model in reconstruction quality for STL-10 dataset (Figure 1). One remedy to this might be regularizing the latent space during pretraining which will be mentioned in Task 3.

In addition to VGG model, we also implemented a ResNet autoencoder, which is ready to use in the codebase, but we did not see any superior performance over VGG, hence did not report it.

We conclude that it is suboptimal to train huge autoencoder which is able to extract pretrained ResNet-like features and constrained loss at the same time within DC-GMM schema. In addition, individual dataset sizes are low compared to huge ImageNet, which hampers the understanding of natural image dynamics.

2.2.1 The NaN Loss Problem

Throughout our experimentations for Task 1 and 2, we experienced that the training fails occasionaly because of NaN loss occuring as below:

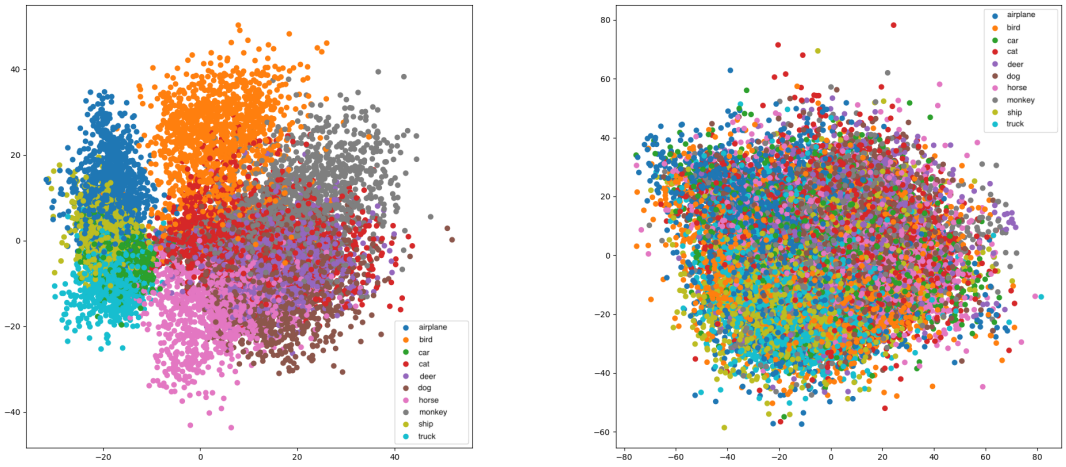

Figure 1: 2 dimensional t-SNE embedding space for STL-10 samples. Left: ResNet-50 encoder outputs. Right: Our best 6 layer VGG encoder outputs. Zoom in for details.

```
Epoch 1/500
162/162 - 239s - loss: nan - output_1_loss: 17371.5684 - output_4_accuracy_metric: 0.2866
- loss_1a: nan - loss_1b: nan - loss_1c: nan - loss_1d: nan - loss_2a: nan - loss_2b: nan
- loss_3: nan - loss_2a_c: nan - val_loss: nan
```

We observed this phenomena when training Task 1 original model with only cannot links for STL-10 dataset, Task 2 original and contrastive models both for VGG 2 and 6 for STL-10 dataset, Task 2 original and contrastive models for VGG 6 for CIFAR-10 dataset, and Task 2 no pretraining for both STL-10 and CIFAR-10 datasets. For other cases for CIFAR-10 dataset, we did not observe it. We can vaugly say that dataset and model complexity play roles in this problem.

We investigated the problem as follows: Since all sublosses becomes nan at the same time, source of the problem should be the same. Only encoder output mean and log variance hold this property. Thus we printed them after each iteration to see that log variance easily diverges to huge numbers after a few number of iterations. See A.4.

We identified the possible causes and proposed solutions as: a) minimizing loss 3 and loss 1b, maximize log z sigma directly. We multiplied these with a small number such as 0.01 to reduce their effect on training. b) reconstruction loss is computed over all pixels, hence increasing pixel count makes it huge compared to other sublosses, possibly making the training unstable. We multiplied it with different small numbers again. c) GMM mean and variance assigments during pretraining might be problematic, hence we initialized GMM with zero mean and unit variance. d) We reduced learning rate and batch size, and switch between cross entropy and mean squared error losses. None of these solved the increasing log variance problem. Hence we decided to cap it within [-1,1] range or not to use encoder output log z sigma and simply assign unit variance for each sample. This twist solves the problem naively, and in Table 2, all STL-10, and CIFAR-10 with 6 layer VGG experiments use it.

2.3 Task 3: Contrastive Pretraining

In this task, we work on regularizing the latent space during pretraining, which will help the autoencoder part of DC-GMM model to extract more distinctive features. This will in turn

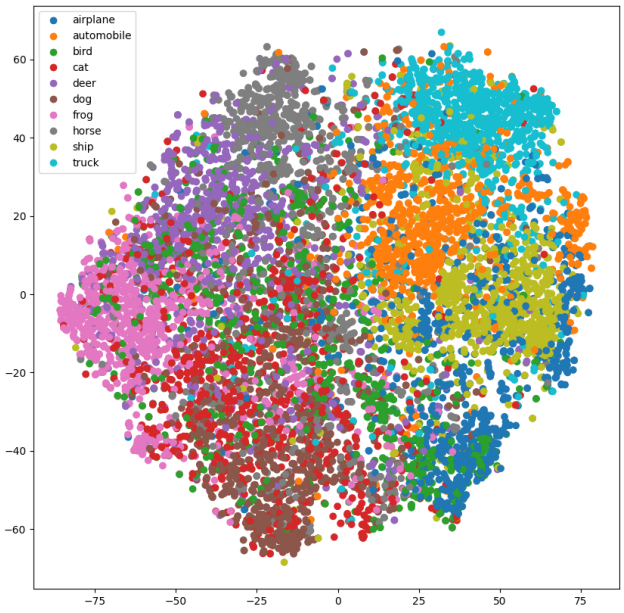

Figure 2: CIFAR-10 test set t-SNE applied 2 dimensional embeddings. Pretraining accuracy: 44%. Note that similar labels reside nearby such as 'truck' and 'automobile', 'cat' and 'dog', and 'deer' and 'horse'.

increase the pretraining accuracy, and ease the DC-GMM model's job.

For regularization of the latent space, we again apply the contrastive learning idea because we want to stay fully unsupervised. This means that we will have two objectives during pretraining: reconstruction loss and contrastive loss between latent representations of samples as outputted by the encoder part. To the best of our knowledge, this is the first attempt for integrating autoencoders with contrastive learning.

We implement the contrastive loss as defined in [CKNH20] in Section 2.1. We directly use their recommended transformations for CIFAR-10 dataset, explained in Appendix B.9. Still, our version is the vanilla version of their original idea because of the following changes: a) They use ResNet-50 as encoder, we need to use VGG 6 layer because of compute resources and decoder parameter doubling. b) They talk about batch sizes up to 8192, and epoch counts up to 1000. In addition, they have two fully connected layers of size 2048, but we will only have of size 128. Overall, we do not aim to mimic their results but slightly increase pretraining accuracy, thus these drawbacks should not hugely affect the main purpose.

Firstly, we check if an encoder with only contrastive loss can lead to higher pretraining accuracy. The answer is yes with the pretraining accuracy of 44%. The CIFAR-10 test set t-SNE applied 2 dimensional embeddings are visualized in Figure 2.

Secondly, we include decoder part and reconstruction loss. Reconstruction loss dominates contrastive one, so we multiply it by 0.001 to make their scale equal. This works smoothly as well with the pretraining accuracy of 38%. This decrease is expected because of the introduced trade-off. The CIFAR-10 test set t-SNE applied 2 dimensional embeddings and reconstructed samples are in Figure 3. Although we implemented the code for integrating this new pretraining regime with DC-GMM training, we could not conduct experiments for measuring DC-GMM performance because of the deadline of this report.

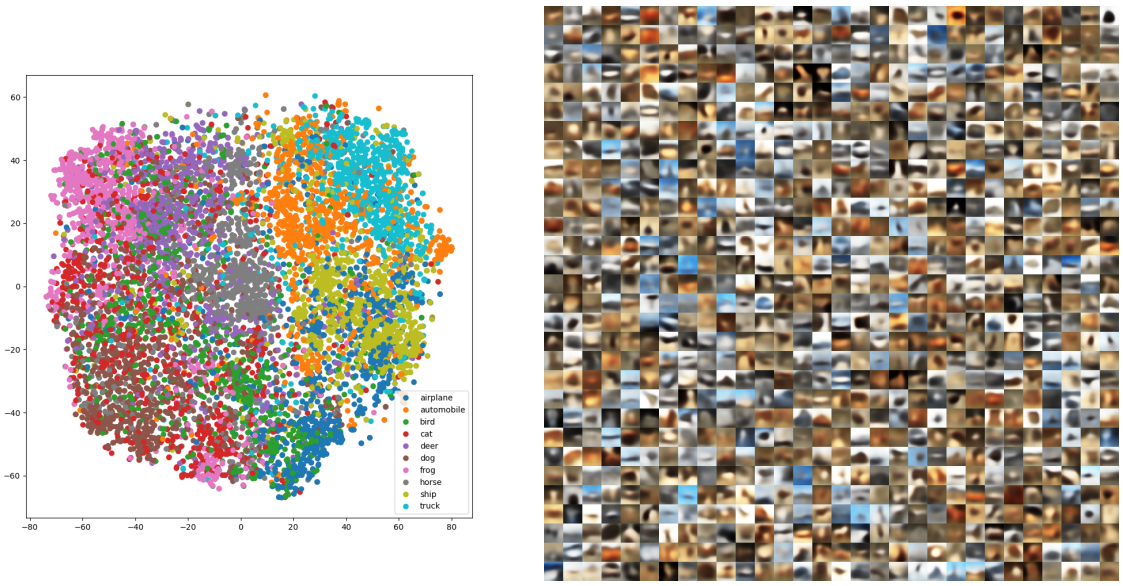

Figure 3: 2 dimensional t-SNE embedding space for CIFAR-10 samples. Pretraining accuracy: 38%. 100 reconstructed samples. Reconstruction quality is not high as in Task 2, but still enough for recovering general object shapes.

To conclude, we can say that contrastive learning regularizes the latent space as expected, and can be a new research direction on its own. For our case, employing it turns the pretraining the overwhelming part of the overall training, although being helpful to unsupervised constrained clustering.

3 Conclusion

In this project, we put effort for making the promising DC-GMM model fully unsupervised using contrastive learning. We believe that we were able to show this self-supervised way of creating preference matrix W is a powerful and intuitive approach. We acknowledge that training dynamics becomes much more complex when we want to turn overall model end-to-end using convolutional autoencoders. Limited number of publications exists in even only for convolutional autoencoder case, thus further integrating it with DC-GMM did not shine in terms of performance as in Task 1. We had to deal with frequently occuring NaN loss along the way, which was the most frustrating part to investigate and try to solve. With no time limit, we would investigate the effects of different latent dimensions, and richer data augmentation pipelines on the performance. Another exciting moment was when we see that the contrastive pretraining works as expected, even though we were out of time to see if the model maintain this better representations during DC-GMM training. Furthermore, currently our implementation is not general enough to run every natural image dataset easily. It only supports running Task 1 with STL-10, Task 2 with STL-10 and CIFAR-10, and Task 3 with CIFAR-10. This technical limitation beside other potential ideas for performance improvement mentioned above constitutes future work.

References

[CHL05] S. Chopra, R. Hadsell, and Y. LeCun. Learning a similarity metric discriminatively, with application to face verification. In *2005 IEEE Computer Society*

Conference on Computer Vision and Pattern Recognition (CVPR'05), volume 1, pages 539–546 vol. 1, 2005.

[cif] https://github.com/rtflynn/Cifar-Autoencoder.

[CKNH20] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey E. Hinton. A simple framework for contrastive learning of visual representations. *CoRR*, abs/2002.05709, 2020.

[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, 11–13 Apr 2011. PMLR.

[JZT+16] Zhuxi Jiang, Yin Zheng, Huachun Tan, Bangsheng Tang, and Hanning Zhou. Variational deep embedding: A generative approach to clustering. *CoRR*, abs/1611.05148, 2016.

[Kri09] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

[MCM+21] Laura Manduchi, Kieran Chin-Cheong, Holger Michel, Sven Wellmann, and Julia E. Vogt. Deep conditional gaussian mixture model for constrained clustering. *CoRR*, abs/2106.06385, 2021.

[SKP15] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. *CoRR*, abs/1503.03832, 2015.

[ZJM+21] Jure Zbontar, Li Jing, Ishan Misra, Yann LeCun, and Stéphane Deny. Barlow twins: Self-supervised learning via redundancy reduction. *CoRR*, abs/2103.03230, 2021.

A Appendix

A.1 Datasets

- The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images [Kri09].
- The STL10 dataset contains color images of 96-by-96 pixel size. The classes are airplane, bird, car, cat, deer, dog, horse, monkey, ship, truck. For each class, 500 training images, 800 test images exits, hence in total 13000 [CNL11].

A.2 Augmentation Details

Our simple transformation strategy for STL-10 dataset is as below. For CIFAR-10, random crop dimensions are (24,24) and resize dimensions are (32,32).

```
transformation = tf.keras.Sequential([
                tf.keras.layers.experimental.preprocessing.RandomCrop(height=72, width=72),
                tf.keras.layers.experimental.preprocessing.Resizing(height=96, width=96),
                tf.keras.layers.experimental.preprocessing.RandomFlip(mode='horizontal'),
                tf.keras.layers.experimental.preprocessing.RandomContrast(factor=0.4)
            ])
```

A.3 Experiment Settings and Model Architectures

For any model, we used symmetric encoder-decoder architecture, so we will only mention encoder network below. Details mentioned below can be seen in accompanying config.yml and model.py files.

For Task 1, ResNet-50 output size is 2048. It is fed to encoder consisting of fully connected layers with sizes 500, 500, 2000, and encoded dim. We used the following training hyperparameters: 'pretrain epochs': 10, 'pretrain batch size': 128, 'pretrain learning rate': 0.001, 'epochs': 500, 'batch size': 256, 'alpha': 10000.0, 'learning rate': 0.001, 'beta 1': 0.9, 'beta 2': 0.999, 'decay rate': 0.9, 'learning rate decay period': 20.

As VGG blocks, we use two types. VGGConvBlock is from the original paper with max-pool replaced with stride of 2, and VGGConvBlockAUX is our version which uses 'same' padding for not decreasing image dimensions, especially useful for CIFAR-10. All kernels are 3x3.

For Task 2, 2 layer VGG encoder consists of [VGGConvBlock(32), VGGConvBlock(64)] where 32 and 64 being filter size. 6 layer VGG encoder consists of [VGGConvBlockAUX(32), VGGConvBlockAUX(64), VGGConvBlock(128), VGGConvBlockAUX(128), VGGConvBlock(256), VGGConvBlockAUX(256)]. Different from above training hyperparameters: 'epochs': 20, 'batch size': 128 for CIFAR-10, 'batch size': 64 for STL-10 (these batch sizes are for 'Contrastive' model, double them for 'Original' model). Increasing epoch size does not increase performance metrics as in Task 1, so we kept it relatively small but enough for more experiment runs.

For Task 3, we use 7 layer VGG encoder which consists of [VGGConvBlockAUX(32), VGGConvBlockAUX(64), VGGConvBlock(128), VGGConvBlockAUX(128), VGGConvBlock(256), VGGConvBlockAUX(256), VGGConvBlockAUX(128)]. The pretraining hyperparameters are 'pretrain epochs': 20, 'pretrain batch size': 128, 'pretrain learning rate': 0.001. Note that we multiply reconstruction loss with 1e-3 to make its scale even with contrastive loss.

A.4 The NaN Loss Problem

Here we present some arbitrary samples' encoder outputted log z sigma values, that appear to diverge quickly in magnitude after just 3 iterations, which is the source of NaN loss problem. First 2 samples (rows) of a batch is shown, where the latent size (columns) is 10.

```
1st iteration
[-2.1377518  -0.17969762  0.32275033  1.329561    0.9209764   1.4660954
   0.31966978  0.7563938   2.189761    0.54003143]
[ 0.49579227 -0.40945256 -0.21772635 -0.38648158  0.4074375   0.50358987
   0.02598591  0.42695895  0.03382339 -0.37310562]

2nd iteration
[  3.1013741  -18.471605    -5.572601     3.2285595   -5.630305
  -16.088032     2.963475    -3.4507668    5.8110456    5.0744433 ]
[ -4.8788004   -8.28209    -13.336223    13.346958     4.4116197
  -22.498154   -12.632924     5.3262563   -1.0842781    3.336876  ]

3rd iteration
[ -15.491957    -16.422861    -20.24002      22.838223     -0.5999282
   -25.35648     -24.273048     -3.6824138   -14.7971735    15.021989  ]
[  -5.985685    -13.221239    -11.37963      16.51146      11.126913
   -12.69787     -18.128578      9.06949      -1.5507759    10.362943  ]

4th iteration
[nan nan nan nan nan nan nan nan nan nan]
[nan nan nan nan nan nan nan nan nan nan]
```

A.5 Convolutional Autoencoder Reconstruction Quality vs. Latent Dimension Experiments

Reconstructions in 4 are done with CIFAR-10 dataset. As apparent, the reconstruction quality gets better for increasing latent dimension, and near perfect at latent dimension of 750. We state that this 6 layer VGG architecture was not enough for STL-10 quality reconstruction. We tried to increase depth and filter sizes but faced LSF memory errors during pretraining. Even if we would manage to pretrain our deep model for STL-10 dataset, during DC-GMM training, we would need higher batch sizes for contrastive learning, which would create additional memory problems.

Figure 4: Reconstruction quality vs encoded size trade-off visualization of VGG 6 layers model for 900 CIFAR-10 images. Encoded size for upper left, upper right, lower left, and lower right are 10, 50, 500, and 750, respectively. Zoom in for details.