Unsupervised Visual Learning: An Empirical Study

Yiming Zuo
Carnegie Mellon University
yzuo@andrew.cmu.edu

Yunze Man
Carnegie Mellon University
yman@andrew.cmu.edu

Zhengyi Luo
Carnegie Mellon University
zluo2@andrew.cmu.edu

1 Introduction

Unsupervised learning is a fundamental and essential sub-field of machine learning that has attracted significant attention. In this study, we focus on unsupervised visual learning, where the center problem can be formulated as: how do we learn underlying patterns from image data that we can classify visual entities without using labels. Despite the the popularity of supervised learning, it requires large, clean, and carefully handcrafted datasets and unsupervised learning is becoming increasingly appealing due to its ability to learn form massive amount of unlabeled data. Traditional methods such as K-means[1] and Gaussian Mixture Models (GMM) have shown promising unsupervised classification performance on benchmark datasets like MNIST [17]. In recent years, benefiting from the development of deep learning and ever-growing availability of computational resources, more advanced unsupervised learning methods such as InfoGAN[5], Capsule Networks[10] and MoCo [8] are poposed.

What are the intuition behind these methods? Do they perform better compared to traditional methods and are they sensitive to the choice of hyper-parameters? Are they computationally expensive and affordable? In this study, we thoroughly analyze the pros and cons of classicial and state-of-the-art methods and try to answer these questions.

This study is structured as follows: in Sec. 2 we describe the two dataset we use. In Sec. 3 we provide a summary of the research background of unsupervised learning and in Sec. 4 we introduce related works in recent years. Methods are described in Sec.5 and experiments and results are shown in Sec.6. Finally we have discussion and analysis in Sec.7.

2 Data

We mainly study the performance of our models on two visual tasks: Optical Character Recognition (OCR) and 3D Object Instances Categorization. The OCR task is relatively simple and well studied and is used as a benchmark to fairly compare the performance of different methods. 3D Object Instances Categorization is a more advanced task to push the boundary of our studied method. We generate the data by ourselves for this task.

2.1 Optical Character Recognition (OCR)

One classic task of unsupervised learning is character classification, or generally Optical Character Recognition (OCR). MNIST [17] and USPS [13] are popular benchmarks for unsupervised learning methods. These tasks are comparatively easy and computationally inexpensive as the background is always clean and the image size is small. We use MNIST as our OCR dataset. MNIST contains images of handwritten digits (0-9) with ground truths. It has 60K training and 10K testing gray scale images of size 28×28 size on a background. We use this dataset as a benchmark for all of the methods we studied. MNIST is relatively well-known that we do not show sample data of it.

2.2 3D Object Instances Categorization

OCR mainly studies the ability of recognizing 2D entity recognition under 2D affine transforms. However, real world objects reside in the 3D world and are subjective to 3D transformations. Given this constraint on the OCR task, we proposes using a more difficult 3D entity dataset to study the effectiveness of unsupervised classification techniques for 3D objects subject to 3D transformations. We synthetically generate a new dataset that contains 10 different chairs under different viewpoints. Specifically, we rendered 10 instances of chairs from ShapeNetV2 dataset [4] under different rotations.

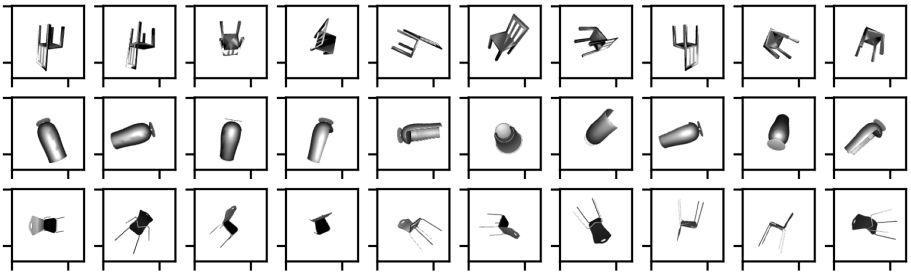

Figure 1: Samples from the ShapeNet-Rotation Dataset

Notice that only rotation is considered as traditional object classification task assume objects to be centered in the current image. For each instance we rendered 7000 images, amounting to 60K training and 10K testing images. Since each instance has a different color and can provide a strong cue for the classifier, we convert all the images to grey scale. Samples from the dataset can be found in Figure.1. For simplicity, we will call it ShapeNet in the following sections.

3 Background

By the time of midway report, we finished the related work searching and studied classic models on MNIST. Classical methods such as Gaussian Mixture Model (GMM) and K-means are simple yet effective unsupervised visual classification methods.

K-means clusters data points through iterative calculate centroids for images based on their euclidean distance in the pixel space. Each centroid is estimated through taking an average of all current data points inside the cluster.

Gaussian mixture model assumes pixels from an image are from multivariate Gaussian distribution, and automatically estimate these Gaussian clusters through Expectation Maximization (EM).

For the consistency of our study structure. Result on classic model are shown in Sec.6 on Table. 1 with newly studied models.

4 Related Works

In this section we will discuss the topics concerning the state-of-the-art methods used in this study.

Contrastive Learning. Contrastive learning [7] is built on the assumption that samples of the same class will have higher similarity and vise versa. In contrastive learning, people use contrastive loss to measure the similarity between sample pairs in the representation space on-the-fly during training. Different methods have been proposed to get pairs of samples with low similarity. For instance, [2, 7, 12, 9, 22, 25] use an end-to-end manner to get negative samples. They use samples in the current mini-batch as the dictionary and requires a large GPU memory size in order to get a sizeable set of negative samples. Another approach is to use a memory bank [24] that contains representation of all samples in the dataset. However, the representation of different samples in the memory bank are extracted by an encoder at different time stamp resulting in inconsistency. In MoCo, a queue is proposed as a dynamic dictionary to maintain past samples' representation, and a momentum update mechanism is proposed to address the inconsistency inside the training process.

Capsule Networks. Capsule networks, proposed in [10], structure hidden units of a neural network into groups, and each group is called a capsule. The key idea behind capsule networks is modeling visual entity's pose and pose invariant features through latent encoding. Each capsule contains a pose vector that represents the 2D or 3D pose of the visual entity, and also contains a feature vector that represents the visual appearance, identity, presence etc. Capsules networks can be viewed as a form of geometrical reasoning where the geometrical knowledge (poses) are explicitly disentangled from its visual appearance. Also, capsules can be grouped and stacked in a hierarchical order to jointly model a visual entity from low level visual features to high level features, simulating the hierarchical structure of objects. Though having been proposed for a decade, capsule networks are difficult to train. Recent advances in its training procedures such as [11] utilizes EM-routing to model part-part and part-object relationship, while [15] uses maximum likelihood and autoencoding.

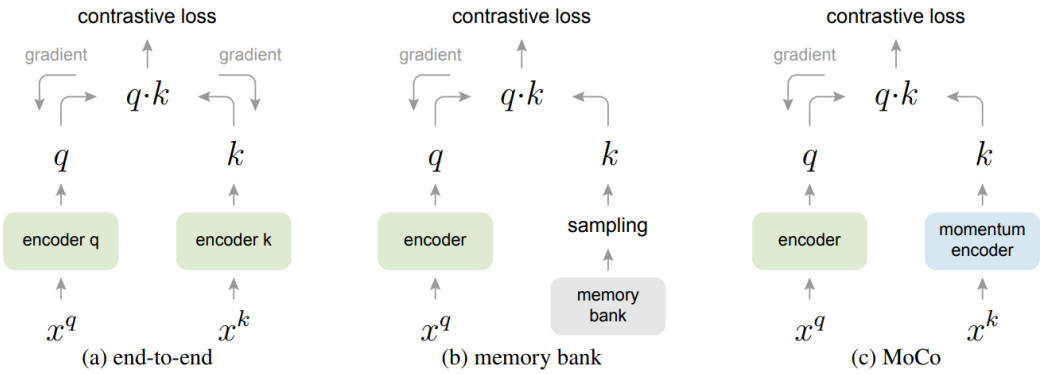

Figure 2: ***Comparison between three contrastive learning paradigm***. In the figure one pair of query and key are illustrated as examples. **(a).** The encoders for computing query and key representations are updated in ***end-to-end*** manner. **(b).** The key representation is sampled from a ***memory bank*** [24]. **(c).** ***MoCo*** samples key representation from a maintained dictionary queue, and the dictionary encoder is updated by momentum mechanism without back-propagation.

InfoGAN [6] introduced the Generative Adversarial Networks (GAN), a framework for training generative models using a minimax game. The goal is to learn a generator that can transform a noise variable $z \sim P_{noise}(z)$ into a sample $G(z)$, and a discriminator that can differentiate between generated samples and real samples. The generator is trained by playing against an adversarial discriminator network D that aims to distinguish between samples from the true data distribution and the generator's distribution. InfoGAN [5] uses an additional loss term to explicitly enforce maximum mutual information between latent code and generated sample. Please see more details in Sec.5.

Autoencoder is a type of artificial neural network used to learn efficient data encodings in an unsupervised manner [16]. To the best of our knowledge, autoencoder was first introduced in [3] for dimensionality reduction and data compression. It was shown to have superior results compared to traditional data compression methods such as principal component analysis (PCA). Autoencoder compress an input singal into a limited dimensional latent code, and then translate the latent code back to the input signal. More recent studies of autoencoder uses additional loss term to enforce sparsity of the latent code and a better representation [20]. The encoder and decoder neural network can also be designed as convolutional architecture as in [19] for parameter efficiency, which is especially suitable for image feature extraction.

5 Methods

5.1 Autoencoder

Autoencoder consists of a encoding network and a decoding network. Formally, Let X represent the input image, h the latent variable, E the encoder network, and D the decoder network. We have:$h = E(X)$ and $X' = D(h)$. The encoder and decoder can be a convoluational neural network, or a multi-layer percetron (fully connected network). Training of the network weights is done by using gradient descent methods with a reconstruction loss between the input and output such as Mean Square Error (MSE) loss: $L(X, X') = \frac{1}{M} \sum_{i=1}^{M} ||X_i - X'_i||_2^2$.

5.2 Momentum Contrast

Contrastive learning models what makes two objects similar or different, and use such similarity as self-supervision. If two samples are similar, then we want the encoding for these two samples to be similar as well.

Problem Modelling Contrastive learning can be thought of as a dictionary look-up task, in which we want to train an encoder to map similar samples into similar representations. Consider encoded samples $\{k_0, k_1, k_2, \ldots\}$ as dictionary, and q as a encoded query. With similarity measured by dot

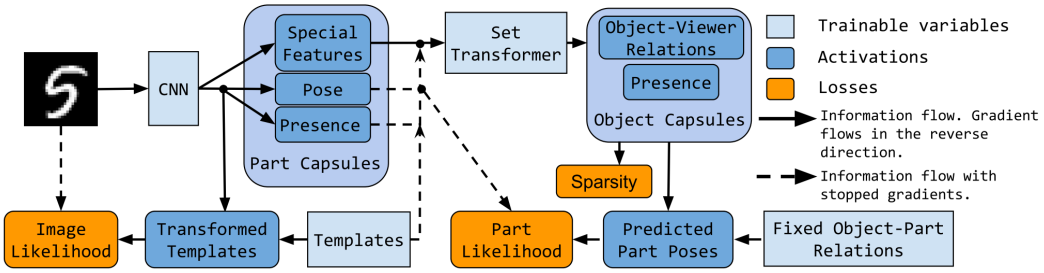

Figure 3: Architecture of Stacked Capsule Autoencoder

product, we use InfoNCE as our contrastive loss:

$$\mathcal{L}_q = -\log \frac{\exp\left(q \cdot k_+/\tau\right)}{\sum_{i=0}^{K} \exp\left(q \cdot k_i/\tau\right)} \tag{1}$$

In which τ is a temperature parameter as defined in [24]. Intuitively, this is equivalent to a $(K+1)$-way softmax-based classifier that tries to classify q as K_+. Specifically, we use an augmented mini-batch as positive samples K_+, and make those sampled from dictionary as negative samples.

Queue as Dictionary In order to sample negative data from a sufficiently big set of data, we maintain a queue of encoded past samples as our dictionary. As such, we decouple the size of dictionary with the size of mini-batch during training, allowing us to have a very large dictionary with limited GPU memory. It is different from the memory bank [24] in the sense that the queue is updated on-the-fly. And more importantly, a momentum-based update mechanism is used for the dictionary-side encoder.

Momentum Update Although introducing a queue as dictionary addresses the coupling issue between mini-batch size and dictionary size, it brings a problem that back-propagation won't work on the continuously updating queue. A straightforward idea is to copy the query-side encoder to the dictionary-side. However it will cause inconsistency between key representations. As such, a momentum update is used to address the issue. Formally, denoting the parameters of query-size encoder as θ_k and those of dictionary-side as θ_k, we update θ_k by:

$$\theta_{\mathrm{k}} \leftarrow \beta\theta_{\mathrm{k}} + (1-\beta)\theta_{\mathrm{q}} \tag{2}$$

Here $\beta \in [0, 1)$ denotes momentum coefficient. Intuitively, the momentum mechanism is calculating the dictionary-side encoder parameters with moving average of the keys. This momentum rule smooths the encoder update process and significantly reduces the representation space inconsistency between different keys in the dictionary.

In Sec. 6, we compares performance with different hyper-parameters and encoder updating mechanisms, in other words with or without momentum update.

5.3 Stacked Capsule Autoencoder

Stacked capsule autoencoder (SCAE) [15], divides a visual entities into *parts* and *objects*. Each *object* consists of multiple *parts*, and assembling visual appearance of *parts* will form a complete object. SCAE discovers visual entities through maximum likelihood estimation and auto-encoding as it is a form of unsupervised learning.

Part Capsule Autoencoder (PCAE) Part capsules represents a small visual entity such as a strokes for a character or a handle for a 3D chair. Part capsules is in charge of discovering visual parts from an image and infer the relative view-point for that part. Assume we have M part capsules in our model. Each part capsule contains a six-dimensional pose x_m (two rotations, two translations, scale and shear), a presence variable $d_m \in [0, 1]$, and an appearance/identity vector z_m. Given an input image y, part capsule encoder $\mathrm{h}^{\mathrm{enc}}$ encodes the it into M capsules $x_{1:M}, d_{1:M}, z_{1:M}$. Each special appearance/identity vector z_m will be used to map the part capsule to a color c_m and image template $T_m \in [0, 1]^{h_t \times w_t \times (c+1)}$ to be assembled to a full image.

$$\mathbf{x}_{1:M}, d_{1:M}, \mathbf{z}_{1:M} = \mathrm{h}^{\mathrm{enc}}(\mathbf{y}) \qquad \text{predict part capsule parameters} \tag{3}$$

$$c_m = \text{MLP}\left(\mathbf{z}_m\right) \quad \text{predict color of part capsule} \tag{4}$$

$$T_m = \text{MLP}\left(\mathbf{z}_m\right) \quad \text{predict template image of part capsule} \tag{5}$$

$$\widehat{T}_m = \text{TransformImage}\left(T_m, \mathbf{x}_m\right) \quad \text{apply affine transforms to image templates} \tag{6}$$

$$p^y_{m,i,j} \propto d_m \widehat{T}^a_{m,i,j} \quad \text{compute mixing probabilities} \tag{7}$$

$$p(\mathbf{y}) = \prod_{i,j} \sum_{m=1}^{M} p^y_{m,i,j} \mathcal{N}\left(y_{i,j} | \mathbf{c}_m \cdot \widehat{T}^c_{m,i,j}; \sigma_y^2\right) \quad \text{calculate the image likelihood} \tag{8}$$

The image is modelled as a spatial Gaussian mixture and the parameters of the networks is trained by maximizing the image likelihood of equation (8).

Object Capsule Autoencoder (OCAE) Object capsules assembles the part capsules into respective objects. The input to the object capsules encoder h^{caps} are the flattened out $x_{1:M}, d_{1:M}, z_{1:M}$ vectors from the part capsules. Assume there are K object capsules, then the output of the object capsule is object capsule pose (object view matrix) $\text{OV}_{1:K}$, object capsule feature vector c_k, and object presence probability $a_k \in [0, 1]$. From part capsule feature c_k, each part capsule uses a seperate multi-layer perceptron $\text{h}^{\text{part}}_{\text{k}}\left(\mathbf{c}_k\right)$ to decode c_k into object-part pose relationship matrix $\text{OP}_{k,1:N}$ (assume each object consists of $< N$ parts), conditional probability $a_{k,n} \in [0, 1]$ that given object a candidate part exists, and standard deviation $\lambda_{k,n}$. Finally, an object is modeled as a mixture of Gaussians from the part parameters. Each part capsule can belong to only one object capsule, and object-capsule likelihood is given by equation (13).

$$\text{OV}_{1:K}, \mathbf{c}_{1:K}, a_{1:K} = \text{h}^{\text{caps}}\left(x_{1:M}, d_{1:M}, z_{1:M}\right) \quad \text{predict object capsule parameters} \tag{9}$$

$$\text{OP}_{k,1:N}, a_{k,1:N}, \lambda_{k,1:N} = \text{h}^{\text{part}}_{\text{k}}\left(\mathbf{c}_k\right) \quad \text{decode candidate parameters from } c_k\text{'s} \tag{10}$$

$$V_{k,n} = \text{Ov}_k \, \text{OP}_{k,n} \quad \text{decode a part pose candidate,} \tag{11}$$

$$p\left(\mathbf{x}_m | k, n\right) = \mathcal{N}\left(\mathbf{x}_m | \mu_{k,n}, \lambda_{k,n}\right) \quad \text{turn candidates into mixture components} \tag{12}$$

$$p\left(\mathbf{x}_{1:M}, d_{1:M}\right) = \prod_{m=1}^{M} \left[\sum_{k=1}^{K} \frac{a_k a_{k,m}}{\sum_i a_i \sum_j a_{i,j}} p\left(\mathbf{x}_m | k, m\right) \right]^{d_m} \tag{13}$$

The OCAE network parameters are also trained to maximize the object-capsule likelihood in equation (13). Architecture of the networks can be found in Figure.3. For more details about training OCAE & PCAE, please refer to [15].

5.4 InfoGAN

InfoGAN [5] is an advanced method for learning interpretable representation learning in a fully unsupervised manner. It is an information-theoretic extension to the Generative Adversarial Network that is able to learn disentangled representations.

Method Based on Generative Adversarial Network (GAN) [6], InfoGAN uses an additional loss term to explicitly maximize the mutual information between the latent code and output image.

Formally, Let G denotes generator, D discriminator, c the latent code, $I(\cdot, \cdot)$ the mutual information and V the GAN loss [6], we have:

$$\min_G \max_D V_I(D, G) = V(D, G) - \lambda I(c; G(z, c)) \tag{14}$$

And the mutual information term is approximated by an additional neural network Q which shares most of the layers with the Discriminator, D.

Choice of the Latent Code It was shown that by providing the generator with an additional noise term which follows 10-class uniform categorical distribution (each corresponding to a digit class in MNIST dataset) and two parameter following continuous uniform distribution, the model is able to learn disentangled representation for digits, width and rotation. We follow the experiment setup described in the original paper and get interpretable generation results.

6 Experiments and Results

During our study, we uses prediction accuracy on the hold out test set as our metric for evaluating the unsupervised classification methods. To assign labels to the learned clusters, we employ majority vote in algorithm (1) using training labels to assign labels to each clusters, and use this cluster dictionary to assign labels for samples in the test set.

Algorithm 1 Assign Majority

Function `gen_cluster_dict(`*training_label, training_clusters*`):`
  for *cluster in training_clusters* **do**
    label_count = []
    for *sample in cluster* **do**
      true_label = training_label[sample]
      label_count[true_label] += 1
    end
    cluster_label = $\text{argmax}_{label}(label\_count)$
    cluster_label_dict[cluster] = cluster_label
  end

6.1 Implementation Details

Autoencoder Considering the resolution and computational cost, we use different implementation of Autoencoder for the two datasets. Details are described below.

For MNIST, we use fully-connected network architecture. Both the encoder and decoder has one hidden layer. Encoder feature size is $28 \times 28 \rightarrow 128 \rightarrow 64$ and decoder $64 \rightarrow 128 \rightarrow 28 \times 28$. We use MSE loss with l2 regularization. We use Adam [14] optimizer with learning rate 1e-3 and trained on the MNIST training set for 100 epochs.

For ShapeNet, we resize the original image to 64×64 and use fully-convolutional neural network (FCN) for both Encoder and Decoder. The encoder has two convolutional layers with kernel size 3 and stride 2 followed by ReLU activation and a maxpooling layer with down-sample rate of 2. The Decoder uses de-convolutional layers with corresponding size with respect to the Encoder. The flattened feature at bottleneck is 200 dimensional vector for each image. We use Adam optimizer with learning rate 1e-3 and trained on the MNIST training set for 30 epochs. It takes about 2 hours for the model to fully converge.We feed the bottleneck feature extracted by the autoencoder to a GMM model with assign_majoirty to get the final classification results.

Momentum Contrast Because of the difference in dataset complexity and size, different backbone networks are used for two dataset. The backbone networks are relatively small in order to save time for parameter tuning and ablation study.

For MNIST, we use a simple two-layer neural network. Two conv2d layers followed by a maxpooling and fully connected to a 1d representation vector. We use random rotation within 20 degrees and random cropping into 28×28 image to form positive samples.

For ShapeNet, we use a self-implemented AlexNet. Note that we use AdaptiveMaxpooling to allow customized size of input. For augmentation, we use random rotation within 20 degrees and random cropping into 112×112 size of image, given the original size of image is 128×128.

We use SGD as our optimizer. The initial learning rate is 0.01 with weight decay set to 0.0001. We use a mini-batch size of 100 in one single RTX2080Ti GPU. We train for 100 epoches with 5 mins for MNIST dataset and 16 hours for ShapeNet dataset.We feed the bottleneck feature extracted by the MoCo encoder to a GMM model with assign_majoirty to get the final classification results.

Stacked Capsule Autoencoders A convolutional encoder is used for part capsules and a set transformer encoder [18] is used for object capsules. MNIST and ShapeNet dataset share most of the model configuration, with MNIST has a input canvas size 40×40 and ShapeNet 128×128. The best performing model has 40 template images, 40 part capsules and 32 object capsules. The part capsule

have CNN architecture of: 2 layers of CNN with channel size 128 and stride of 2 plus a 2 layers of CNN with channel size 128 and stride of 1, with 3×3 kernels . For set transformer it uses one attention head of 16 hidden units and 256 output units with layer normalization. ReLU activation is used for all nonlinearities except for presence vector d_m, where sigmoid is used. RMSPropOptimizer is used for optimization. For implementation details please refer to [15]. The model for MNIST was trained for 300000 epoches over 5 days. The model for ShapeNet was for 6 days and reached around 4000 epoches. Classification on Capsule Network is performed by K-means clustering on object capsules into 10 clusters and bipartite matching based on training labels.

6.2 Results on Two Dataset

We report our experiment results on MNIST and ShapeNet in Table 1. For GMM and K-means, we report the best result using tuned parameters. For other models, refer to Sec.6.1 for details. For MNIST dataset, Capsule reports the best accuracy of 0.985. In terms of efficiency, MoCo is better than Capsule. It achieves comparative results in 5 minutes, which is 1000 times faster than Capsule. On ShapeNet, Capsule fails to converge with limited computational resources and training time, while MoCo still achieves a reasonable accuracy with acceptable training time.

Table 1: Results of different unsupervised learning methods on MNIST and ShapeNet.

	MNIST		**ShapeNet**	
	Accuracy	Training Time	Accuracy	Training Time
K-means (10 classes)	0.752	1 min	0.395	2 mins
GMM (130 components)	0.928	3 mins	0.747	1 hour
Auto-Encoder	0.912	20 mins	0.637	2 hours
InfoGan [1]	0.920	-	-	-
Capsule	**0.985**	5 days	0.228 [2]	-
MoCo	0.969	5 mins	**0.752**	16 hours

6.3 Ablation Study

Stacked Capsule Autoencoders: For Capsule networks, we tuned the number of part and object capsules for both of the datasets, and results can be found in table 2. From the result we can see that 40 part capsules and 32 object capsules are the optimal setting for both of the experiments.

Table 2: Results of different parameter of capsule networks.

Template #	Template Size	Capsule #	**Acc.**	
			MNIST	ShapeNet
30	30	16	0.9836	0.1508^2 (5019 epoch)
40	40	32	**0.985**	$\mathbf{0.2280}^2$ (4320 epoch)
60	60	40	0.9825	0.1146^2 (2068 epoch)

Momentum Contrast: For MoCo, we test our model with different queue length (dictionary size) and momentum factor on MNIST dataset. The result is shown in Table 3. We can see that with momentum 0.999 and queue length 64, the model can achieve best performance. Notice that Momentum $= 0$ will result in the non-convergence of MoCo. This prove that a momentum closer to 1 can smooth the update of dictionary-side encoder and stabilize the training. A possible reason for the performance drop with $m = 0.9999$ is that the encoder process too slowly and hasn't reach the optimal state. The dictionary size also should not be too large as it will amplify the inconsistency between different keys within the dictionary. In our case, 64 is a good trade-off for MNIST dataset.

7 Discussion and Analysis

Our results on MNIST show that all methods except for K-means can achieve comparative results, while Capsule networks achieve the best performance of 98.5% accuracy. However, comparing the training time, it is clear that MoCo has the best performance overall in this task and achieve a balance between computational efficiency and accuracy. Notice that GMM, though a non-deep-learning method, performed well in this task and achieved competitive accuracy. During our experiment

[1] Due to limited computational resources, we did not train InfoGAN from scratch but use pre-trained model

[2] Models have not yet converged after 6 days and partially trained result are shown here

Table 3: Ablation Study of MoCo on MNIST Dataset. '-' means the model doesn't converge.

Queue Length	16	32	64	128	256
Momentum = 0	-	-	-	-	-
Momentum = 0.9	0.9138	0.9511	0.9516	0.9531	0.9442
Momentum = 0.99	0.9298	0.9327	0.9623	0.9691	0.9536
Momentum = 0.999	0.9291	0.9429	**0.9698**	0.9487	0.9339
Momentum = 0.9999	0.9120	0.9380	0.9446	0.9367	0.9403

we notice that the accuracy of GMM models increase significantly as we increase the number of components, and the trend of accuracy gain does not stop at 130 components (due to computational limitation our maximum model have full diagonal matrix and 130 components). The strong performance of GMM with large number of components can be attributed to the characteristic of the MNIST dataset: the dataset contains only black and white digits and does not contains significant variation in terms of background and shape. Thus, a GMM model with significantly more components than number of classes can memorize all of the variations and successfully classify in the test set. However, in natural images with a variety of backgrounds, GMM may suffer from model capacity while MoCo and Autoencoder may benefit from CNN based feature extractors. InfoGAN is designed as a interpretable generative model and is not explicitly optimized for classification task, so its result is not as appealing as other state-of-the-art methods like Capsule.

On the more challenging dataset of rendered ShapeNet models, similar trend can be observed. failed to achieve a resonable training epoch during the time frame of this project, and only attain an accuracy slightly better than random guessing. It can be seen that Capsule networks still suffers from its long training time and model complexity. MoCo performed the best overall within a reasonable training time, while GMM still achieves similar results. GMM's success can be attributed similar reason under MNIST: this dataset features black and white images of rendered 3D chairs from ShapeNet under different viewpoints, and while having a much bigger image size (128 x 128 [ShapeNet] vs 40 x 40 [MNIST]), the dataset still suffers from lack of variation in its background. The dataset also suffers from lack of pixel variability, with the shapes rendered only under the same lighting and texture condition.

MoCo's strong performance overall demonstrated its effective in discovering visual entities under both 2D transformations and 3D transformations. Since we chose a relatively simple backbone for both of the dataset, the full power of MoCo has yet to be discovered. Though state-of-the-art methods usually rely on CNNs as feature extractor (auto-encoder, Capsules, InfoGAN), results from GMM shows that it is an efficient way for modeling appearance for the same entity. Notice that the drawbacks from the current ShapeNet dataset can be improved by render the 3D shapes to a randomrized background, as well as perform random image disturbing on the generated images (such as blur, add random noise, etc.). Result from such a dataset can provide additional insight into the ability of GMM and MoCo under a more diverse image setting.

7.1 Future Works

We have studied several state-of-the-art unsupervised learning methods and conducted experiment to compare them across two different datasets. The experiments, however, are still limited in their thoroughness. For instance, due to the limited computational resources, we only did ablation studies on several parameters that we think are most important. The choice of neural network backbones for MoCo is also conservative as a deeper network would consume more GPU memory and typically takes longer time to converge. In the future we would try to extend the analysis to include more variables so that we can search for the "best" model under multiple criteria.

On the other hand, though we have covered both 2D and 3D vision dataset, all of the methods we have explored does not utilize any form of 3D prior (capsule networks only uses 2D poses, while in general CNN models no 3D transformation). The natural next step would be to extend the current models by baking in 3D priors such as in [21] [23].

8 Acknowledgement

We would like to thank the professors, TAs (especially Himanshi Yadav) and students of 10701 who kindly reviewed our work and provided valuable suggestion to us.

References

[1] David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. Technical report, 2006.

[2] Philip Bachman, R Devon Hjelm, and William Buchwalter. Learning representations by maximizing mutual information across views. In *Advances in Neural Information Processing Systems*, pages 15509–15519, 2019.

[3] Pierre Baldi and Kurt Hornik. Neural networks and principal component analysis: Learning from examples without local minima. *Neural networks*, 2(1):53–58, 1989.

[4] Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. ShapeNet: An Information-Rich 3D Model Repository. Technical Report arXiv:1512.03012 [cs.GR], Stanford University — Princeton University — Toyota Technological Institute at Chicago, 2015.

[5] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *Advances in neural information processing systems*, pages 2172–2180, 2016.

[6] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[7] Raia Hadsell, Sumit Chopra, and Yann LeCun. Dimensionality reduction by learning an invariant mapping. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 2, pages 1735–1742. IEEE, 2006.

[8] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. *arXiv preprint arXiv:1911.05722*, 2019.

[9] Olivier J Hénaff, Ali Razavi, Carl Doersch, SM Eslami, and Aaron van den Oord. Data-efficient image recognition with contrastive predictive coding. *arXiv preprint arXiv:1905.09272*, 2019.

[10] Geoffrey E. Hinton, Alex Krizhevsky, and Sida D. Wang. Transforming auto-encoders. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 6791 LNCS(PART 1):44–51, 2011.

[11] Geoffrey E Hinton, Sara Sabour, and Nicholas Frosst. Matrix capsules with EM routing. In *International Conference on Learning Representations*, 2018.

[12] R Devon Hjelm, Alex Fedorov, Samuel Lavoie-Marchildon, Karan Grewal, Phil Bachman, Adam Trischler, and Yoshua Bengio. Learning deep representations by mutual information estimation and maximization. In *International Conference on Learning Representations*, 2019.

[13] Jonathan J. Hull. A database for handwritten text recognition research. *IEEE Transactions on pattern analysis and machine intelligence*, 16(5):550–554, 1994.

[14] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[15] Adam Kosiorek, Sara Sabour, Yee Whye Teh, and Geoffrey E Hinton. Stacked capsule autoencoders. In *Advances in Neural Information Processing Systems*, pages 15486–15496, 2019.

[16] Mark A Kramer. Nonlinear principal component analysis using autoassociative neural networks. *AIChE journal*, 37(2):233–243, 1991.

[17] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[18] Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 3744–3753, Long Beach, California, USA, 09–15 Jun 2019. PMLR.

[19] Jonathan Masci, Ueli Meier, Dan Cireşan, and Jürgen Schmidhuber. Stacked convolutional auto-encoders for hierarchical feature extraction. In *International conference on artificial neural networks*, pages 52–59. Springer, 2011.

[20] Andrew Ng et al. Sparse autoencoder.

[21] Thu Nguyen-Phuoc, Chuan Li, Lucas Theis, Christian Richardt, and Yong-Liang Yang. HoloGAN: Unsupervised learning of 3D representations from natural images. (Figure 1), 2019.

[22] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

[23] Nitish Srivastava, Hanlin Goh, and Ruslan Salakhutdinov. Geometric Capsule Autoencoders for 3D Point Clouds. 2019.

[24] Zhirong Wu, Yuanjun Xiong, Stella X Yu, and Dahua Lin. Unsupervised feature learning via non-parametric instance discrimination. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3733–3742, 2018.

[25] Mang Ye, Xu Zhang, Pong C Yuen, and Shih-Fu Chang. Unsupervised embedding learning via invariant and spreading instance feature. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6210–6219, 2019.