Autoencoder : Learning Features

------ Xiaorui HUO

Abstract

Representation learning is a set of techniques that allows a system to automatically discover the representations needed for feature detection or classification from raw data. This replaces manual feature engineering and allows a machine to both learn the features and use them to perform a specific task. In unsupervised representation learning, features are learned with unlabeled input data. Examples include dictionary learning, independent component analysis, autoencoders, matrix factorization and various forms of clustering[1]. In our project, we develop several different structures of autoencoders to acquire a set of feature points that describe objects, such as the shape, color, position, and then we analyze and explore the feature points in order to generate new similar samples. The learned feature points can be used in reinforcement learning which allows the robot to dynamically manipulate objects, including picking up a cup and pushing a chair.

Contents

1. Introduction

There are several impressive successes of applying reinforcement learning to real-world systems. However, one of the fundamental challenges in applying reinforcement learning to robotic manipulation tasks is the need to define objects with rather low dimensionality. Typically, manual feature engineering is used to create features that make machine learning algorithms work. But it is both difficult and expensive. So how could a robot automatically acquire low-dimensional features from raw images captured by a camera in current tasks? By learning low-dimensional features, it allows a robot to learn useful policies.

We propose several different structures of autoencoders to acquire low-dimensional features. There are four models, one of which is deep spatial autoencoder[2], which provides for data-efficient learning by minimizing the number of non-convolutional parameters in the encoder network. The other three autoencoders are based on VGG-16 architecture [3] which is pre-trained on ImageNet [4] for image classification.

In our experiments, we first train deep spatial autoencoder and VGG-16 autoencoder on dataset which picks three categories(tables, toys and balls) from ImageNet. These two models are evaluated by comparing the performance of reconstruction. Due to the complexity of images on ImageNet, usually containing multiple objects, we build simple dataset consisting of synthetic images. We then choose VGG-16 autoencoders and train them on our own dataset. Finally, we analyze the output of the encoder part to identify each feature point.

2. Related Work

There are numerous studies in the area of unsupervised feature learning and deep learning, covering advances in probabilistic models, autoencoders, manifold learning, and deep networks. Representation learning has been applied to control from high-dimensional visual input in several recent works. Most studies for representation learning have proposed autoencoders to learn a lower-dimensional latent space representation.

Finn et al. [2] use deep spatial autoencoder to acquire a state space. The architecture using for the encoder is based on a spatial softmax followed by an expectation operation, which produces a set of spatial features that correspond to points of maximal activation in the last convolutional layer. It can acquire a

representation from real-world images that is particularly well suited for high-dimensional continuous control. It is also data-efficient and can handle relatively small datasets.

Besides, Forestier et al. [5] use various unsupervised learning algorithms for the goal space learning component : AutoEncoders, Variational AE, Variational AE with Normalizing Flow, PCA. The architecture of AutoEncoder or Variational AE are based on VGG-16 architecture. It shows that representation learning algorithms can discover goal spaces that lead to exploration dynamics close to the one obtained using an engineered goal representation space. Other methods about autoencoders are mainly composed of convolutional layers and fully connected layers.

In our project, we apply deep spatial autoencoder[2] and VGG-16 autoencoders [5] to our dataset , and we train and evaluate these models on two datasets. At the same time, we analyze lower-dimensional latent space representation to generate new images.

3. Proposed Methods

We mainly introduce four methods, deep spatial autoencoder, VGG-16 autoencoder, sparse VGG-16 autoencoder and variational VGG-16 autoencoder. Next we provide details about each architecture.

3.1 Deep Spatial Autoencoder

For this architecture(shown in figure 1), the first part of encoder is a standard 3-layer convolutional neural network with rectified linear units. We compute the spatial features from the last convolutional layer by performing a “spatial soft arg-max” operation that determines the image-space point of maximal activation in each channel of conv3. Then it follows by an expectation operator that extracts the positions of the points of maximal activation in each channel. Finally, the decoder is simply a single linear (fully connected layer) mapping from the feature points to the down-sampled image.

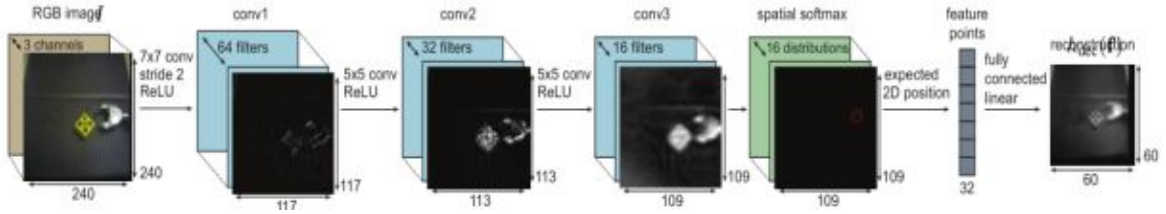


Figure 1 : The architecture for deep spatial autoencoder

In order to reconstruct the image instead of the down-sampled image, we add 4-layer transpose convolutional neural network with rectified linear units to the part of decoder for our project.

3.2 VGG-16 Autoencoder

As in our network architecture(shown in figure 2), the part of encoder is based on VGG-16 architecture [3] which is pre-trained on ImageNet [4] for image classification and each convolutional layer is followed by batch normalization and ReLU nonlinearities. The last fully connected layer forms the bottleneck of the autoencoder. We choose 32 as the dimension of the bottleneck, since the object has many features. The decoder consists of 3 fully connected layers followed by 5 convolution layers, finally reconstructing the 224*224*3 image. We initialize the part of encoder with weights trained on ImageNet, in addition to the bottleneck of the autoencoder.

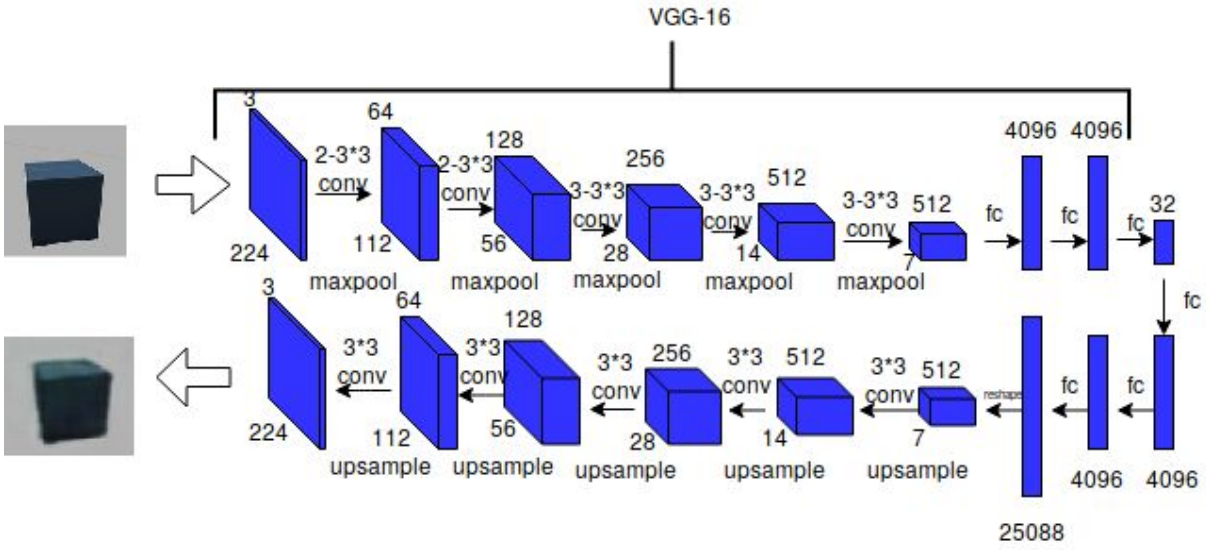


Figure 2 : The architecture for VGG-16 autoencoder

3.3 Sparse VGG-16 Autoencoder

For the previous VGG-16 Autoencoder, the representation is constrained by the hidden layer with the size 32. In such a situation, it is like an approximation of PCA (principal component analysis). But another way to constrain the representation to be compact is to add a sparsity constraint on the activity of the hidden representation, so fewer units would "fire" at a given time and the model can learn more useful features.

The overall structure of sparse VGG-16 autoencoder is the same as the above VGG-16 autoencoder. We only add a sparse constraint at the bottleneck(the last fully connected layer of the encoder) to obtain more useful feature points.

3.4 Variational VGG-16 Autoencoder

Variational autoencoders are a slightly more modern and interesting take on autoencoding. It's a type of autoencoder with added constraints on the encoded representations being learned. Instead of letting the neural network learn an arbitrary function, it learns the parameters of a probability distribution modeling the dataset. If we sample points from this distribution, we can generate new input data samples: a VAE is a "generative model". In practice, this model simplifies to an architecture very similar to an AE, differing only in the fact that the encoder f outputs the parameters μ and ά of a multivariate Gaussian distribution N(μ,diag(fi^2)) with diagonal covariance matrix, from which the representation z is sampled. Moreover, an extra term is added to the cost function, to condition the distribution of z in the representation space.

The structure of variational VGG-16 autoencoder is roughly the same as that of the above VGG-16 autoencoder, and its structure is as shown in the figure below.

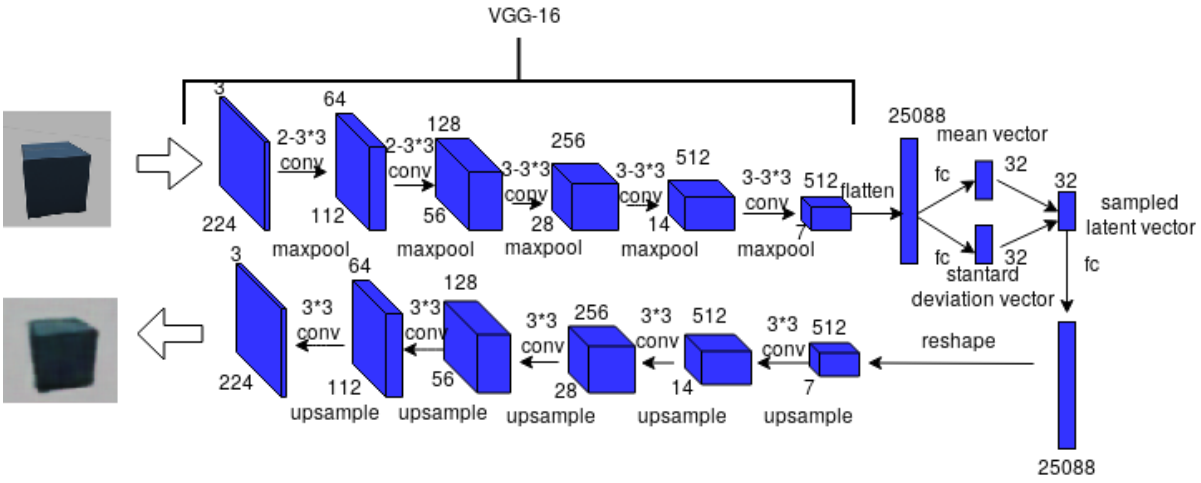


Figure 3 : The architecture for variational VGG-16 autoencoder

4. Experiments

We trained and evaluated our several methods on two datasets. The aim of these experiments was to determine which method was more appropriate to our case. We compared several methods and analyzed lower-dimensional latent space representation. In addition, we also generated new images by changing some nodes of the representation.

4.1 Models trained on ImageNet dataset

4.1.1 ImageNet dataset

ImageNet is a project about computer vision system recognition and is the largest dataset for image recognition in the world. So we decided to pick several kinds of images from ImageNet as our dataset. We downloaded three categories(tables, toys and balls) from ImageNet. Of course, some images don't fit very well and may contain multiple other objects, so we need to remove some images. In total we obtained 3000 images from ImageNet.

In order to train our model on this dataset, we also need to do some preprocessing on these images. For deep spatial autoencoder, these images were cropped to 240*240*3 pixels. For VGG-16 autoencoder, these images were cropped to 224*224*3 pixels as the input.

4.1.2 Results and Analysis

In this experiment, we trained and evaluated two models, deep spatial autoencoder and VGG-16 autoencoder.

For deep spatial autoencoder, we optimized the autoencoder using Adam, and the loss function used is mean square error. The batch size is 32 and we trained the autoencoder for 50 epochs. As for the way of evaluation, we put the original image into the model, let the image go through encoder and decoder, and then we observed the similarity of the two pictures. After training, we observed that the resulting image(shown in figure 4) is very blurred.

Input image Output image

Figure 4 : Result for deep spatial autoencoder

For VGG-16 autoencoder, we also optimized the autoencoder using Adam, but the loss function used is binary cross entropy. Since the loss function we originally used is mean square error, the resulting image(shown in figure 6) is still very blurred. The batch size is 32 and we trained the autoencoder for 50 epochs. Due to the large number of parameters in this model and memory for our computer, we only selected 1000 pictures for training. It took more than one hour to train this model. After training, we found that the result(shown in figure 6) is better than that of deep spatial autoencoder. Although the resulting image is blurred, we could see the outline of the object.

Figure 5 : Result for VGG-16 autoencoder(mean square error)

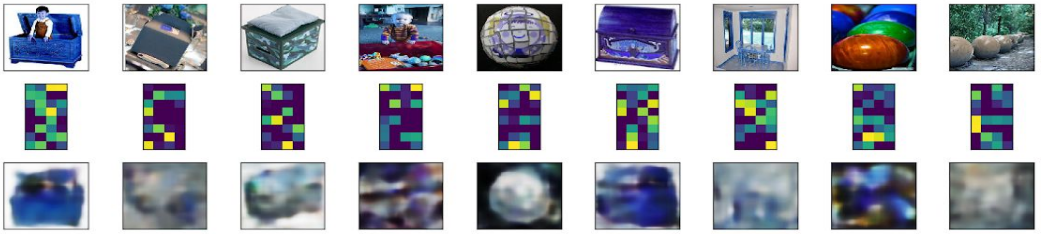


Figure 6 : Result for VGG-16 autoencoder(binary cross entropy)

By evaluating these two models, we could conclude that the performance of VGG-16 autoencoder is much better than that of deep spatial autoencoder, although the result obtained by VGG-16 autoencoder is also blurred. We considered that this situation may be caused by the complexity of image. There is only a single object in the picture, but the background is more complicated, and it is difficult for the model to obtain useful features. In the next experiments, we created a new dataset consisting of synthetic images. Models used are VGG-16 autoencoder, sparse VGG-16 autoencoder and variational VGG-16 autoencoder.

4.2 Models trained on synthetic dataset

In this experiment, we trained and evaluated three models(VGG-16 autoencoder, sparse VGG-16 autoencoder and variational VGG-16 autoencoder)on a small dataset, and then trained and evaluated on a big dataset. We also changed some nodes of representation to observe the result.

4.2.1 Models trained on small dataset

4.2.1.1 Small dataset

This small dataset consists of six objects, cardboard box, cinder block, coke can, grey tote, thin stick and wood cube. Each object has about 90 images, differing in orientation. In total, we created 552 images, of which 440 were used as training sets and 112 were used as test sets. Similarly, we do some preprocessing on these images. These images were cropped to 224*224*3 pixels as the input image.

4.2.1.2 Results and Analysis

A. Test model without training

Before we train the model, we first tried to test VGG-16 autoencoder (trained on ImageNet in the first experiment) directly on our small dataset. The result(shown in figure 7) is very vague, we can't see anything.

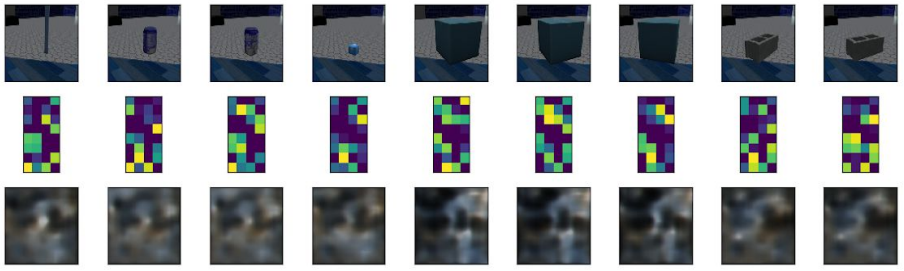
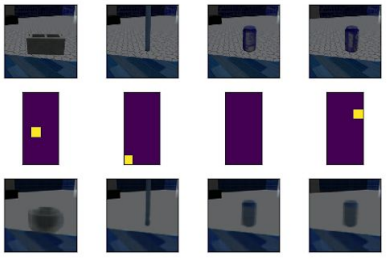


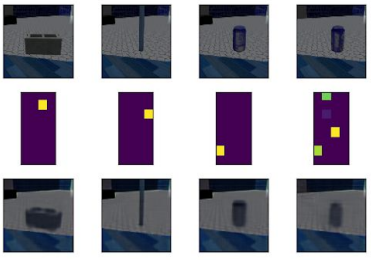
Figure 7 : Result for VGG-16 autoencoder(test on small dataset without traning)

B. Compare regularizer parameters for sparse VGG-16 autoencoder

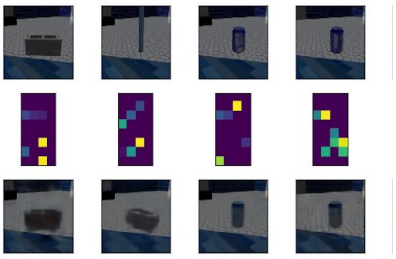
For sparse VGG-16 autoencoder, we regularize the autoencoder by using a sparsity constraint such that only a fraction of the nodes would have nonzero values, called active nodes. In Keras, this can be done by adding an activity_regularizer to the last fully connected layer of the encoder. This is typically a value in the range [0.001, 0.000001]. So we chose 10e-6, 15e-7, 10e-7, 10e-8 to evaluate the performance of the model. The result is shown in figure 8. We observed that if the parameter is too large like 10e-6, the result is bad, and if the parameter is too small like 10e-8, there are still many nonzero points for the representation. So we finally chose parameter 15e-7 for the next experiments.



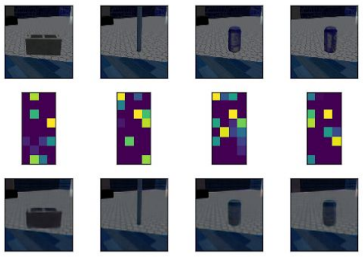
regularizer parameters:10e-6



regularizer parameters:15e-7



regularizer parameters:10e-7



regularizer parameters:10e-8

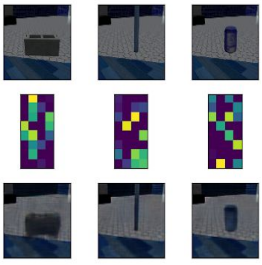
Figure 8 : Comparison with different regularizer parameters

C. Evaluate three models

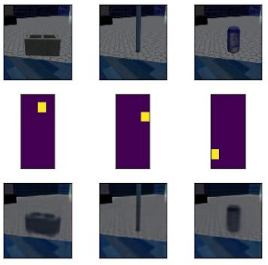
We evaluated three models, VGG-16 autoencoder, sparse VGG-16 autoencoder and variational VGG-16 autoencoder. The loss function of the first two models is binary cross entropy. For the third model, it contains two loss function, one is binary cross entropy, another is the KL divergence between the learned latent distribution and the prior distribution, acting as a regularization term. The results, shown in table 1, show that these three models achieve a similar reconstruction loss, but variational VGG-16 autoencoder is better than others in terms of the image reconstruction(shown in figure 9).

Architectures	Reconstruction Loss (test loss)
VGG-16 autoencoder	0.480
Sparse VGG-16 autoencoder	0.482
Variational VGG-16 autoencoder	0.479

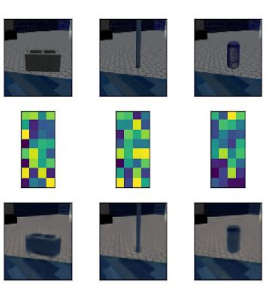
Table 1 : Comparison with different autoencoder architectures on small dataset



VGG-16 AE



Sparse VGG-16 AE



Variational VGG-16 AE

Figure 9 : Results for different autoencoder architectures on small dataset

D. Generate new images

For variational VGG-16 AE, it is a generative model, we can also use it to generate new images. So we randomly sampled 32 points from uniform distribution u(-1,1) as the representation, and then generated new samples similar but different from the training set data through the decoder. The result is as follows.

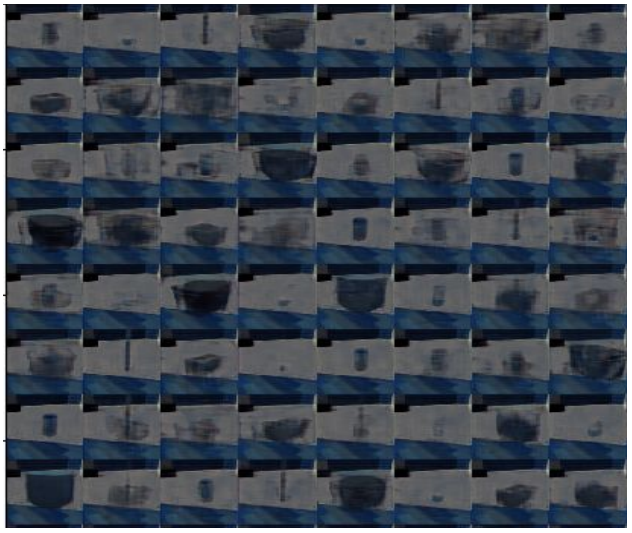


Figure 10 : Generate new samples

We also tried to change only a few points(2 or 4) for the representation in order to generate a new sample(it may change the shape,color or orientation), but we found that the result is still similar to the original one and hasn’t changed.

E. Sparse and Variational VGG-16 AE

We added a sparsity constraint on the activity of the mean layer for variational VGG-16 AE in order to reduce points of the representation(some points become zero). We first chose 15e-7 as the regularizer parameter, but the result shows that the number of nonzero points doesn’t decrease. Then we tried to choose a bigger regularizer parameter 10e-6, the result(shown in figure 11) is the same.

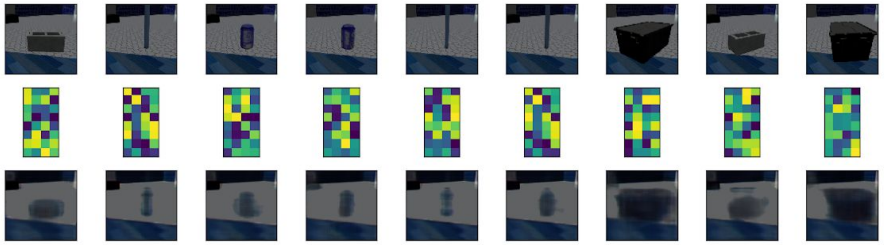


Figure 11 : Result for Sparse and Variational VGG-16 AE
(regularizer parameters:10e-6)

4.2.2 Models trained on big dataset

4.2.1.1 Big dataset

Considering that the number of images in the previous dataset is not enough, and there is only one different feature(orientation) for each object, so we decided to create a large dataset. The big dataset consists of six objects, cardboard box, cinder block, coke can, grey tote, thin stick and wood cube. Each object has about 300 images, differing in size, color, orientation, shape, scalar. In total, we created 1900 images, of which 1700 were used as training sets and 200 were used as test sets. Similarly, we do some preprocessing on these images. These images were cropped to 224*224*3 pixels as the input image.

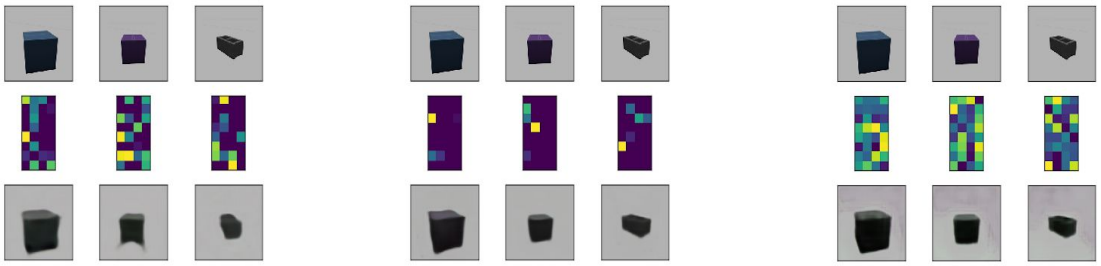
4.2.1.2 Results and Analysis

A. Evaluate three models

We evaluated three models, VGG-16 autoencoder, sparse VGG-16 autoencoder and variational VGG-16 autoencoder. Their loss function(binary_crossentropy) and optimizer(Adam or Rmsprop for variational VGG-16 autoencoder) are the same as previous experiments. The results, shown in table 2, show that these three models achieve a similar reconstruction loss, but it exists one problem with the resulting images reconstruction(shown in figure 12), there is no color.

Architectures	Reconstruction Loss (test loss)
VGG-16 autoencoder	0.591
Sparse VGG-16 autoencoder	0.593
Variational VGG-16 autoencoder	0.590

Table 2 : Comparison with different autoencoder architectures on big dataset

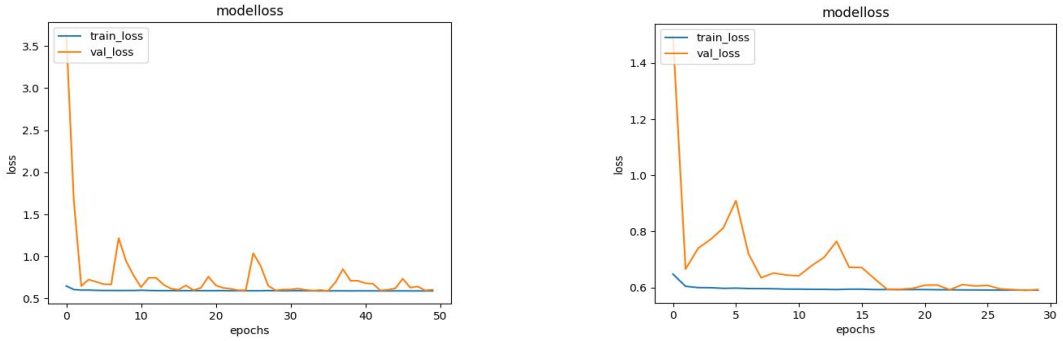


VGG-16 AE Sparse VGG-16 AE Variational VGG-16 AE

Figure 12 : Results for different autoencoder architectures on big dataset

B. Change loss function and learning rate

By observing the curves of train loss and validation loss, we found that the curves (shown in figure 13)have several fluctuation, so we decided to change the loss function and learning rate.



VGG-16 AE Sparse VGG-16 AE

Figure 13 : Model Loss

There are four combinations for loss function and the learning rate. We first tested VGG-16 autoencoder and the results(shown in figure 14 and table 3), shows that the model with a lower learning rate has a lower reconstruction loss, and the resulting images have color. But when we just changed the loss function, the resulting images are still without color.

Loss function and Learning rate	Reconstruction Loss (test loss)
(1) mean_squared_error + Adam(lr =0.001)	0.0067
(2) mean_squared_error + Adam(lr = 0.0001)	0.0015
(3) binary_crossentropy + Adam(lr = 0.001)	0.5914
(4) binary_crossentropy + Adam(lr = 0.0001)	0.5867

Table 3 : Comparison with different loss function and learning rate(VGG-16 AE)

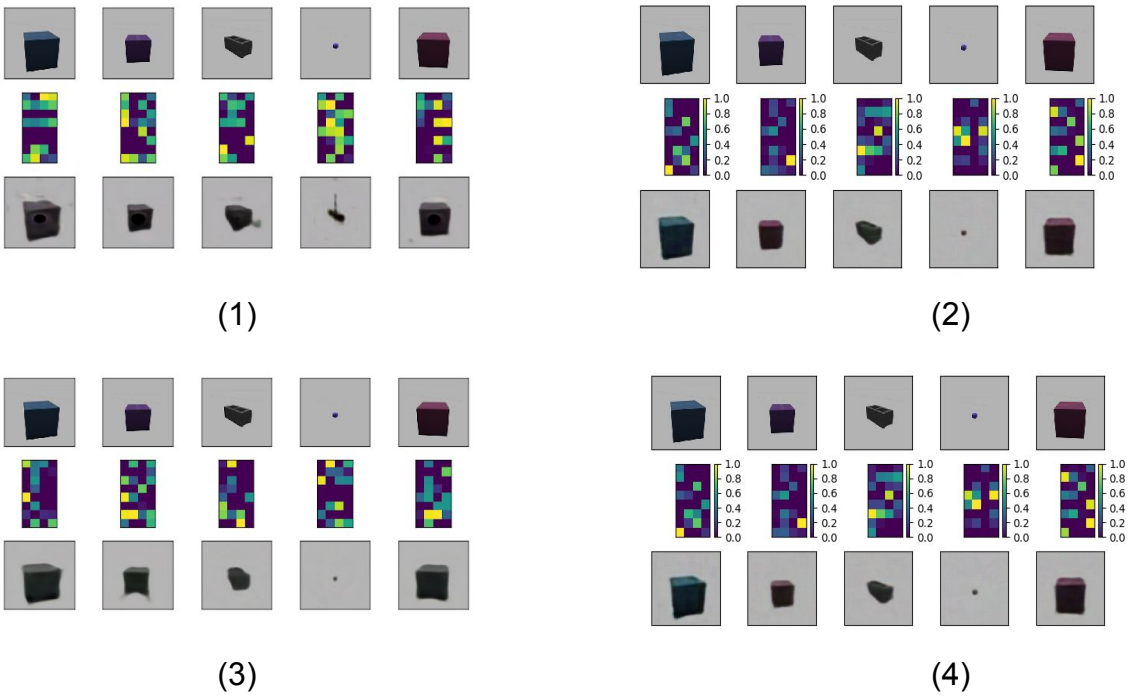
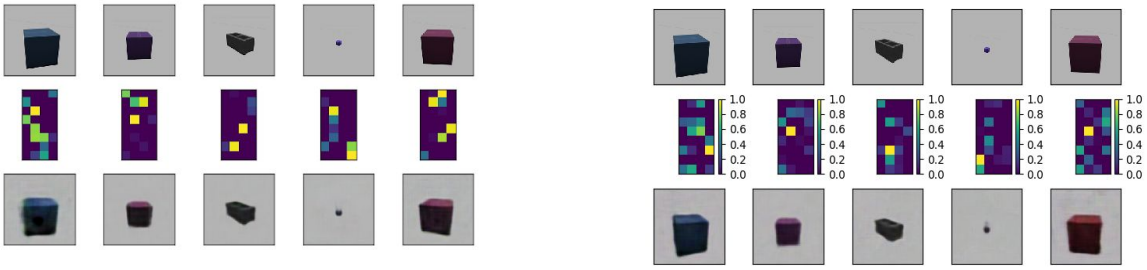


Figure 14 : Results for four combinations(VGG-16 AE)

We then tested sparse VGG-16 autoencoder, and since we only changed the loss function, we couldn't solve the problem about color, we tested it in two cases. The result(shown in table 4 and figure 15) shows that when we choose binary cross entropy as loss function, the result is better. While mean squared error is used as loss function, the orientation of the object is different from the original one.

Loss function and Learning rate	Reconstruction Loss (test loss)
(1)mean_squared_error + Adam(lr = 0.0001)	0.0028
(2)binary_crossentropy + Adam(lr = 0.0001)	0.5884

Table 4 : Comparison with different loss function and learning rate
(sparse VGG-16 AE)



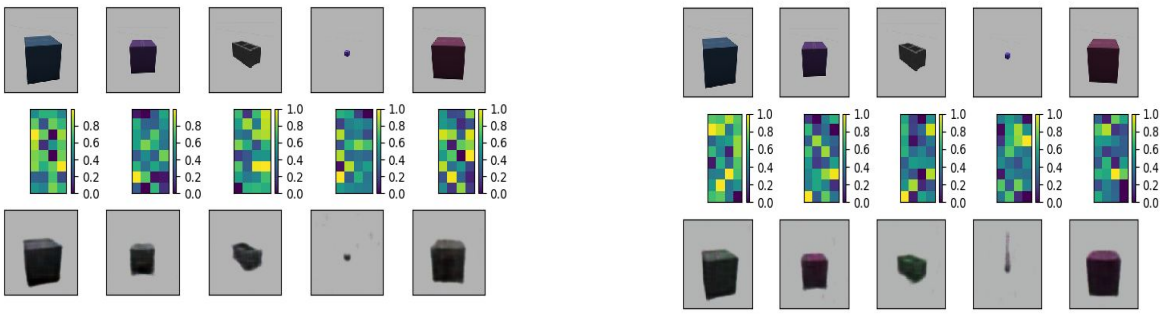
(1) (2)

Figure 15 : Results for two combinations(sparse VGG-16 AE)

Finally we tested variational VGG-16 autoencoder in two case. The result(shown in table 5 and figure 16) shows that when we choose binary as loss function and a lower learning rate(0.0001), the resulting image is with color, but the color of some images does not correspond to the original images. However, when the loss function is mean squared error and the learning rate is lower, the resulting image is still without color.

Loss function and Learning rate	Reconstruction Loss (test loss)
(1)mean_squared_error +Rmsprop(lr = 0.0001)	0.0025
(2)binary_crossentropy + Rmsprop(lr = 0.0001)	0.5876

Table 5 : Comparison with different loss function and learning rate
(varaitional VGG-16 AE)

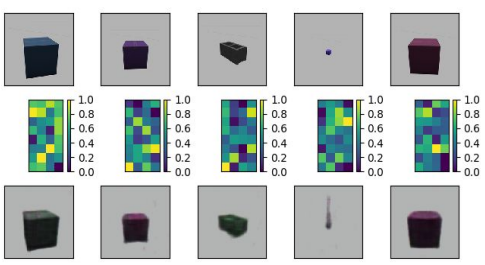


(1) (2)

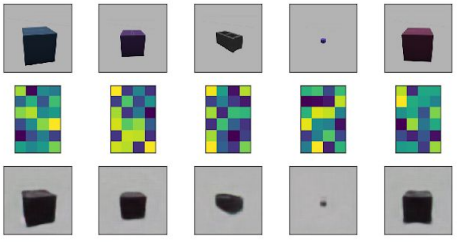
Figure 16 : Results for two combinations(variational VGG-16 AE)

C. Reduce dimension of the representation

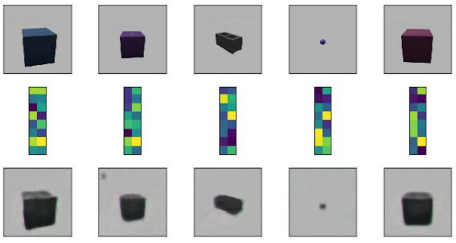
As far as the dimension of the representation is a bit more, we decided to reduce the dimension. So we tested variational VGG-16 AE autoencoder and the results are as follows. When we reduce the dimension of the representation, the resulting images lose the color.



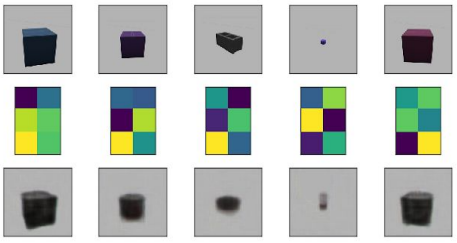
dimension : 32



dimension : 24



dimension : 16



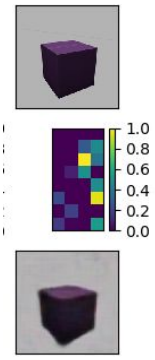
dimension : 6

Figure 17 : Results for different dimension of the representation (variational VGG-16 AE)

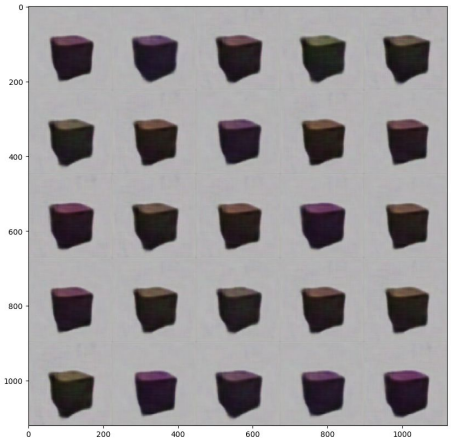
D. Change a few points of the representation

Considering that the number of nonzero points in the representation(the output of the encoder) is relatively fewer for sparse VGG-16 AE, so we decided to choose the decoder of sparse VGG-16 AE. By changing partial points in the representation to generate new pictures, maybe some features change.

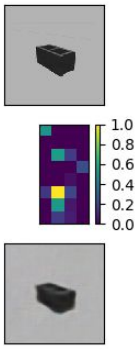
For each type of object in the test set, we chose four points with the highest mean or variance in the representation, and then we randomly sampled 32 points from uniform distribution u(0,4) to replace the original four points for one image. We tested each type of image and the results are as follows.



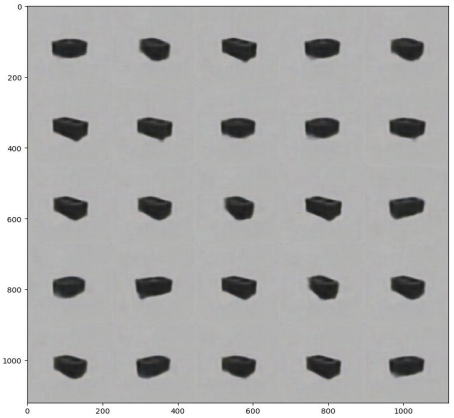
original image(cardboard box)



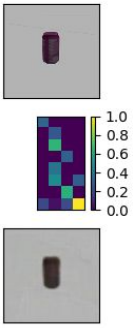
resulting image(the change of color)



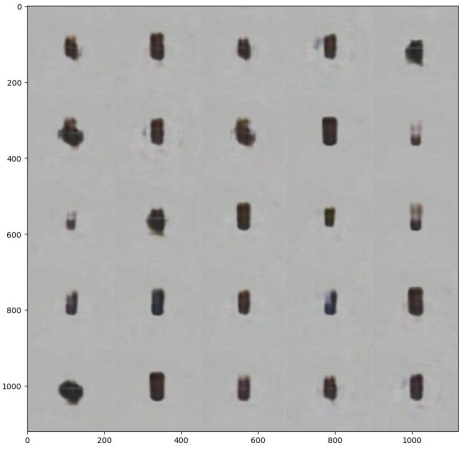
original image(cinder block)



resulting image(the change of orientation)



original image(coke can)



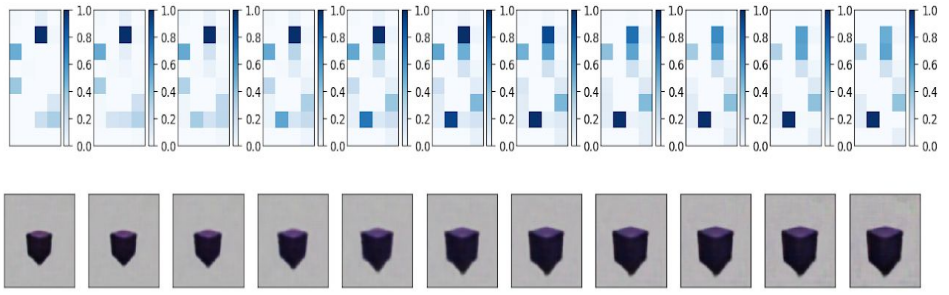
resulting image(the change of size)

Figure 18 : Results for changing four points in the representation

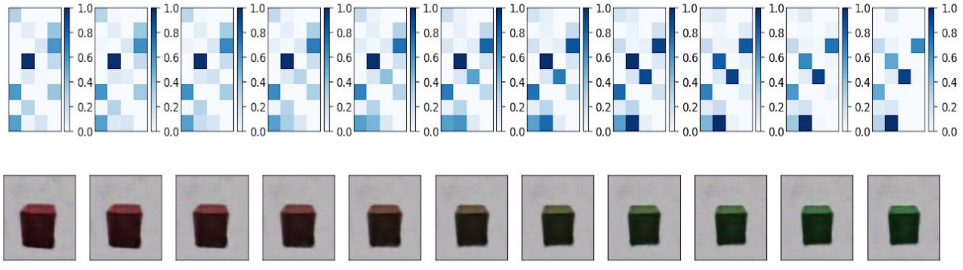
From these three examples we can observe that the four points changed correspond to some features(size,color,orientation) of the object. When the values of these points change, the features change accordingly.

E. Linear transformation for the representation

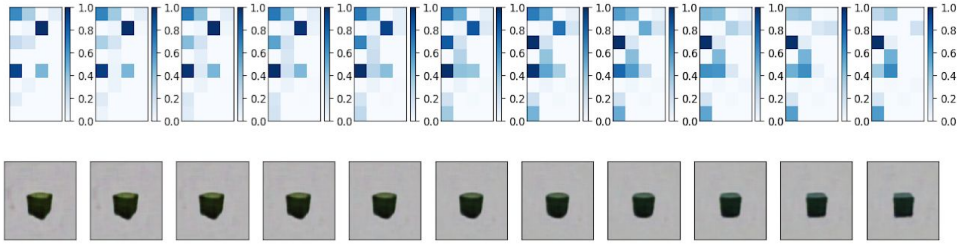
To further analyze the representation(the output of the encoder), we take two similar images for linear transformations. The two images differ only in one feature, such as size, color, or orientation. Assuming that the representation of the first image is L_1 and the representation of the second image is L_2, then the formula of linear transformations is $M = L_1 + a(L_2 - L_1)$, when $a = 0$, $M = L_1$; when $a = 1$, $M = L_2$. In the interval $[0, 1]$, we take the values every 0.1 , which are 0, 0.1, 0.2 ... 0.9, 1. These resulting M as the inputs go through the decoder of sparse VGG-16 AE to generate some similar images. The results are as follows.



size



color



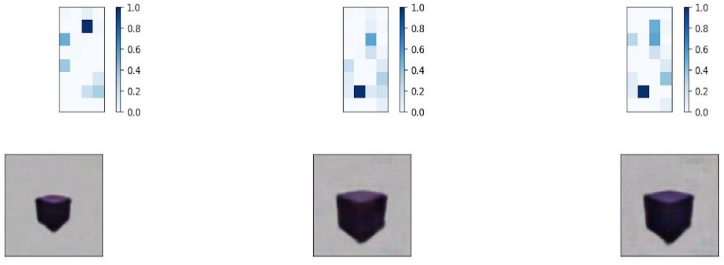
orientation

Figure 19 : Results for linear transformations(size,color, orientation)

From these examples we can see that the color and size gradually change as the value of the representation changes, but for orientation, as the value changes, the orientation change directly from the first image to the second image, there is no other new image.

We also drew the difference between the two images, where the representation is $M = \left|L_2 - L_1\right|$. The resulting M as the input go through the decoder of sparse VGG-16 AE to generate image. The results are as follows.

First image Difference Second image



size

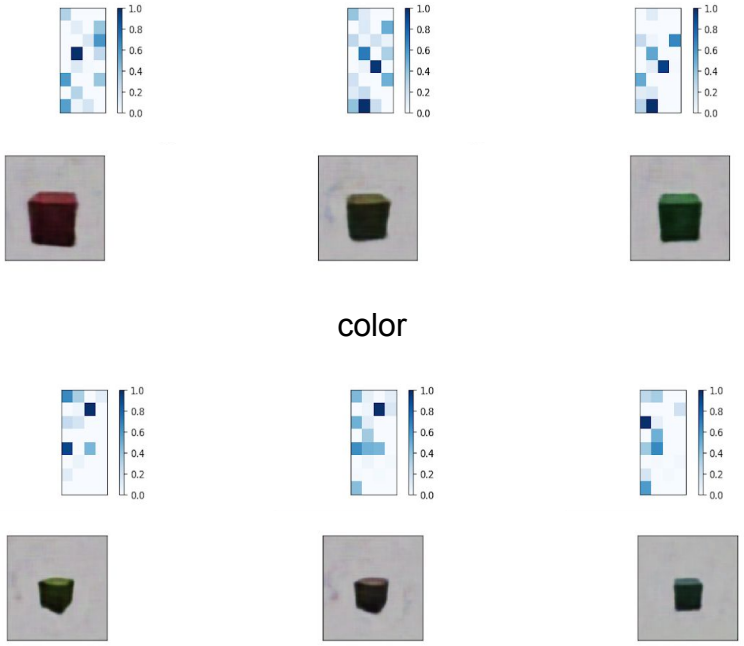
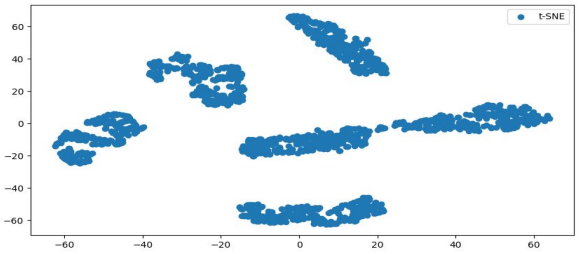


Figure 20 : Difference between two similar images(size,color, orientation)

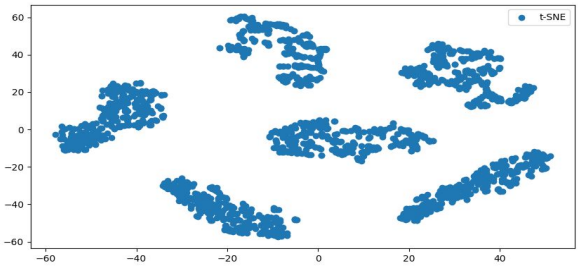
For the two examples of color and size, we can see that the difference between the representations of the two images produces a new different color or size. But for the example of orientation, it has not changed. Since there are too many points in the representation, we can't observe which point changed a feature(size,color, orientation).

F. Data dimensionality reduction and visualization---t-SNE

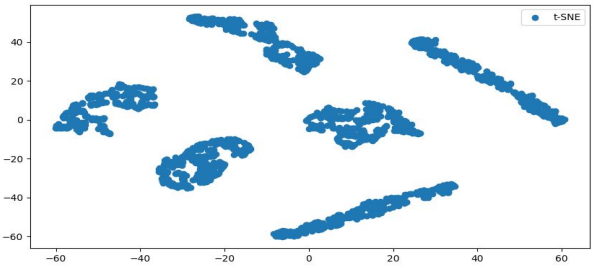
t-SNE is a nonlinear dimensionality reduction technique well-suited for embedding high-dimensional data for visualization in a low-dimensional space of two or three dimensions[6]. In our project, we choose 32 as the dimension of representation, which is a bit high. Therefore, we used the method of t-SNE to classify high-dimensional representations to observe the separability of data, and whether there are large intervals between images with different features. We tested on three models(VGG-16 autoencoder, sparse VGG-16 autoencoder and variational VGG-16 autoencoder). The results(shown in figure 21) show that dataset is divided into six blocks, which correspond to six types of objects. The performance of variational VGG-16 autoencoder) is better, since the division between the six types of dataset is obvious.



VGG-16 autoencoder



sparse VGG-16 autoencoder



variational VGG-16 autoencoder

Figure 21 : t-SNE on three models

5. Conclusion

We proposed four methods, deep spatial autoencoder, VGG-16 autoencoder, sparse VGG-16 autoencoder and variational VGG-16 autoencoder, to learn lower-dimensional representation.

In the first experiment, we trained two models on ImageNet dataset. The results show that the performance of VGG-16 autoencoder is better than that of deep spatial autoencoder. So in the next experiments, we mainly chose three kinds of VGG-16 autoencoder. After adjusting the parameters(regularizer, learning rate and loss function), training and testing for three models, our experiments have shown that the performance of variational VGG-16 autoencoder is the best, whose test loss is lower.

Furthermore, we analyzed the representation by changing some points, using linear transformation and t-SNE. The results show that some points in the representation correspond some features and we can change these points to generate new similar images, which differ in one feature(size, color, orientation). In addition, by the method of t-SNE, we observed there were six splits corresponding six types of objects, showing good separability of lower-dimensional representation.

Autoencoders have been successfully applied to dimensionality reduction and information retrieval tasks. Lower-dimensional representations can improve performance on many tasks, such as classification. In the future, we can take the learned lower-dimensional representation as input and use the classifier to judge the characteristics of the object, such as movable, rolling. Besides, we can add a linear layer on the top of the encoder to determine which point can change the feature.

Reference

[1] Feature learning, https://en.wikipedia.org/wiki/Feature_learning

[2] Chelsea Finn, Xin Yu Tan, Yan Duan, Trevor Darrell, Sergey Levine, Pieter Abbeel, Deep Spatial Autoencoders for Visuomotor Learning, 2016.

[3] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. CoRR, abs/1409.1556, 2014.

[4] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. IJCV, pages 1–42, 2014.

[5] Alexandre Péré, Sebastien Forestier, Olivier Sigaud, Pierre-Yves Oudeyer, Unsupervised learning of goal spaces for intrinsically motivated goal exploration, 2018.

[6] t-distributed stochastic neighbor embedding, https://en.wikipedia.org/wiki/T-distributed_stochastic_neighbor_embedding

README

AUTOENCODER

AutoEncoder - Keras implementation on ImageNet and synthetic dataset

Dependencies

- keras(TensorFlow backend)
- numpy, matplotlib, scipy, glob,cv2
- cuda 9.0 , cudnn 7.0.5

Usage

1) Manage dataset

You need to do some preprocessing on dataset. These images are cropped to 224*224*3 pixels as the input images.
You can run
src/prepocess.py :

- it crops the input images and resizes them
- example : *python3 prepocess.py -p 'dbcafe/x_train/" - d 224*
  -p means the path of dataset, -d means the dimension to crop

2) Train and Test your model

There are 5 models, you can choose one of them to train and test.
You can run
src/AE_deep_spatial.py :

- it creates, trains and test deep spatial autoencoder .
- After training, you can save your model in *save_model* directory, and then you can test your model, the result can be saved in *result_cafe* directory.
- example: *python3 AE_deep_spatial.py*

src/ ae.py :

- it creates, trains and test VGG-16 autoencoder .
- After training, you can save your model in *save_model* directory, and then you can test your model, the result can be saved in *result_cafe/AE/* directory.
- example: *python3 ae.py -d 32 -r 0.0001 -e 50 - p 'save_model/ae.h5'*
  -d means the dimension of latent space, -r means learning rate - e means training epochs, -p means the path for saving model.
- if you use the model trained and test the model, you can run
  python3 ae.py -m 'save_model/ae.h5'

-m means the path of model trained

src/ ae_sparse.py :

- it creates, trains and test sparse VGG-16 autoencoder .
- After training, you can save your model in *save_model* directory, and then you can test your model, the result can be saved in *result_cafe/AE_SPARSE/* directory.
- example: *python3 ae_sparse.py -d 32 -r 0.0001 -e 50 -re 15e-7 - p 'save_model/ae_sparse.h5'*

  -d means the dimension of latent space, -r means learning rate - e means training epochs, -re means the regularizer parameter -p means the path for saving model.
- if you use the model trained and test the model, you can run

  python3 ae_sparse.py -m 'save_model/ae_sparse.h5'

  -m means the path of model trained

src/ vae.py :

- it creates, trains and test variational VGG-16 autoencoder .
- After training, you can save the weights of model in *save_model* directory, the saved model can be used to analyse the latent distribution and to generate new images, and then you can test your model, the result can be saved in *result_cafe/VAE/* directory.
- example: *python3 vae.py -d 32 -r 0.0001 -e 50 - p 'save_model/vae.h5'*

  -d means the dimension of latent space, -r means learning rate - e means training epochs, -p means the path for saving the weights of model.
- if you load the weights of model and test the model, you can run

  python3 vae.py -m 'save_model/vae.h5'

  -m means the path of loading the weights of model.

src/ vae_sparse.py :

- it creates, trains and test sparse and variational VGG-16 autoencoder .
- After training, you can save the weights of model in *save_model* directory, the saved model can be used to analyse the latent distribution and to generate new images,and then you can test your model, the result can be saved in *result_cafe/VAE_SPARSE* directory.
- example: *python3 vae-sparse.py -d 32 -r 0.0001 -e 50 -re 15e-7 - p 'save_model/vae_sparse.h5'*

  -d means the dimension of latent space, -r means learning rate - e means training epochs, -re means the regularizer parameter, -p means the path for saving the weights of model.
- if you load the weights of model and test the model, you can run

  python3 vae_sparse.py -m 'save_model/vae_sparse.h5'

-m means the path of loading the weights of model.

3) Analysis

If you want to analyze the latent space for sparse VGG-16 autoencoder, you can run src/ sparse_decoder.py :

- it analyzes the latent space by changing four nodes and linear transformation.
- example: *python3 sparse_decoder.py 0 -p '.dbcafe/cardboard/'*
  0 means that you choose the method of changing four nodes, -p means the path of dataset, you can save the results in *result-cafe/generator/* directory
- example: *python3 sparse_decoder.py 1*
  1 means that you choose the method of linear transformation, you can save the results in *result-cafe/linear/* directory

Directory on the server greiner

/home/xhuo/AE/
- src(code)
  -- AE_deep_spatial.py
  -- ae_sparse.py
  -- sparse_decoder.py
  -- vae_sparse.py
  -- ae.py
  -- preprocess.py
  -- vae.py
- data(small synthetic dataset)
  -- x_train(training set)
  -- x_test(test set)
- dbcafe(big synthetic dataset)
  -- x_train(training set)
  -- x_test(test set)
  -- two(used for linear transformation)
  -- box(one kind of object used for changing four nodes)
  -- cardboard
  -- can
  -- cinder
  -- mailbox
- result(the testing results of different models on small synthetic dataset)
  -- AE
  -- VAE
  -- AE_SPARSE
- result_cafe(the testing results of different models big synthetic dataset)
  -- AE
  -- VAE
  -- AE_SPARSE
  -- VAE_SPARSE
  -- linear (results for linear transformation)
  -- generator(results for changing four nodes)
- save_model(saving model)