Learning Object Representations by Mixing Scenes

Master Thesis

Lukas Zbinden
University of Fribourg

May 2019

Supervisor:
Prof. Dr. Paolo Favaro
Computer Vision Group, Institute of Computer Science
University of Bern, Switzerland

Abstract

In the digital age of ever increasing data amassment and accessibility, the demand for scalable machine learning models effective at refining the new oil is unprecedented. Unsupervised representation learning methods present a promising approach to exploit this invaluable yet unlabeled digital resource at scale. However, a majority of these approaches focuses on synthetic or simplified datasets of images. What if a method could learn directly from natural Internet-scale image data? In this thesis, we propose a novel approach for unsupervised learning of object representations by mixing natural image scenes. Without any human help, our method mixes visually similar images to synthesize new realistic scenes using adversarial training. In this process the model learns to represent and understand the objects prevalent in natural image data and makes them available for downstream applications. For example, it enables the transfer of objects from one scene to another. Through qualitative experiments on complex image data we show the effectiveness of our method along with its limitations. Moreover, we benchmark our approach quantitatively against state-of-the-art works on the STL-10 dataset. Our proposed method demonstrates the potential that lies in learning representations directly from natural image data and reinforces it as a promising avenue for future research.

Acknowledgements

First and foremost, I want to thank my supervisor, Prof. Dr. Paolo Favaro, for giving me the opportunity to write my thesis in his research group and in particular for devising the research idea, which turned out to be precisely what I was looking for. It has been a privilege to be able to immerse oneself in this interdisciplinary, intriguing and dynamic research field with seemingly endless potential. I thank him also for many motivating and inspiring discussions which pushed me to go another step further and improve the method under research. I am also indebted to the PhD students Qiyang Hu, Simon Jenni, Givi Meishvili and Adrian Wälchli for their support. Without their expertise I would not have achieved the progress presented in this work. Special thanks to Adrian Wälchli, Joel Niklaus and Ioannis Glampedakis for proof-reading the thesis, the constructive feedback was highly appreciated. Much appreciated was also the treat of coffee, cookies and chocolate along with a workplace offered by the Computer Vision Group (CVG) of the University of Bern. Finally, I want to thank my parents, Hans and Christine, for their continued support throughout my master studies.

Contents

Chapter 1

Introduction

1.1 Motivation

The dawn of deep learning in 2012 with the invention of the groundbreaking AlexNet neural network reignited the idea of an intelligent computer that is on a par with the human brain. After much disappointment in the late 20th century, the decade old research field of artificial intelligence and deep learning in particular was revived around the globe. While today research is still far away from the lofty aim of an artificial general intelligence machine, impressive achievements have been made continuously ever since the breakthrough. One research area in deep learning where progress has stood out in recent years is *computer vision*. The field of computer vision aims at equipping a machine with artificial eyesight to process and interpret visual signals such as images and video. For instance, a recent study demonstrates that the computer identifies skin cancer with a precision higher than that of most dermatologists [22]. Other significant applications such as autonomous driving or medical image analysis have emerged from the ability to understand visual data and presumably many more are yet to follow.

Within computer vision, one area of research focuses on the representation of visual data such as objects or features thereof (e.g. pose, color) in low dimensional space, the so-called latent, representation, feature or embedding space. Whereas an image resides in high dimensional space, the aim of *representation learning* in the context of computer vision is to learn a model that successfully transforms a given image to a representation residing in a lower dimensional space that is meaningful and captures the significant image features. This type of dimensionality reduction opens up a multitude of new possibilities for downstream applications that make use of latent space representations. Going yet a step further, some research imposes additional constraints on the representation to lay the groundwork for more fine-grained applications. This research area is called *disentangling factors*

of variation. A factor of variation represents an image object or image attribute in latent space. The aim is to learn a representation that is not only low dimensional and meaningful but exposes a structure consisting of disentangled, independent factors of variation. Disentangled and independent means when a particular image feature is changed (e.g. pose, color) only the corresponding factor of variation in the latent space changes while all the others remain unaffected. With such a structure at hand, *latent space interpolations* become possible where objects or attributes can be swapped, mixed or modified using arithmetic operations. Such interpolations often result in nonlinear changes in the output space. Latent space interpolations thus present a powerful technique for image and video processing. Previous works have been successful at representation learning and disentangling factors of variation on a variety of datasets mostly either preprocessed and aligned or even synthetic. Many of these methods use manual labeling that requires human effort and some methods use no labels at all.

In our work, we pose the question if it is possible to devise an unsupervised learning method effective at learning object representations directly from a natural, unprocessed dataset with several objects per image. Previous methods have been remarkable at achieving this task yet using simpler data mostly with one center-aligned object per image. Our aim, however, is to find a method that performs as well as previous works but learns solely from a collection of random images with everyday scenes in their natural context. These images come with a varying number of objects per sample and add an extra layer of difficulty to the learning task due to their inherent unstructured nature. More specifically, we focus on learning representations and disentangling factors of variation in an unsupervised manner using natural image data. In a world of ever increasing data it has become infeasible to process the vast amounts by hand and make sense of them using the generally better performing *supervised learning* approaches where the machine model is told the expected outcome for a given data sample. Here, *unsupervised learning* comes to the rescue as a technique that learns directly from unlabeled data without the limitation of supervised learning requiring human annotation and can therefore better exploit the tremendous potential that lies therein. To that end, we propose a novel machine learning model aimed at learning directly from Internet-scale unlabeled image data as motivated in Figure 1.1. Simply put, given two real images showing different scenes, the model generates a new realistic scene that mixes the content of the two inputs. For instance, the two scenes could be pictures taken in a park with several persons and dogs in each. The generated image will in turn also represent a scene in a similar looking park with people and dogs mixed together from both input scenes. The model extracts objects from both scenes and joins them in a new made-up scene. The mixing of the objects is done in the latent space based on the extracted object representations. We name the resulting architecture of our model *LORBMS* (Learning Object Rep-

resentations By Mixing Scenes).

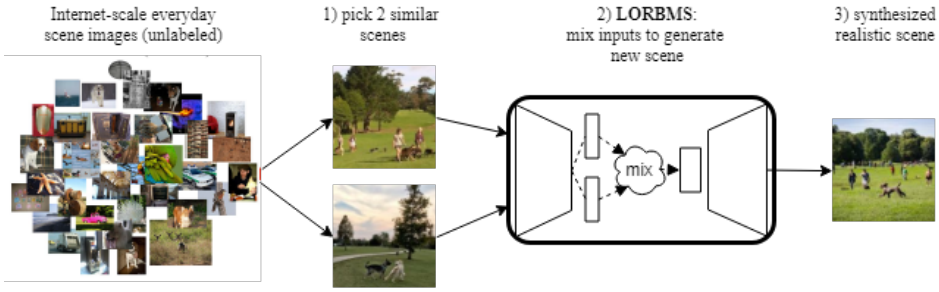


Figure 1.1: Motivation for the proposed LORBMS system.

By successfully mixing and generating realistic scenes, the proposed method learns to extract objects from an image into well structured representations. It is such a representation that can be of most value to downstream applications such as image search, classification, object detection, semantic segmentation, latent space interpolations and transfer learning tasks. This motivates our work.

Below we propose LORBMS, an unsupervised method to extract object representations and generate mixed scenes using natural data. We investigate the question whether the proposed model is able to learn object representations from a complex unstructured dataset where samples come with a varying number of objects. Our work builds on a previous method that successfully disentangles factors of variation in the MNIST, Sprites and celebA datasets. We present this groundwork in Section 2 along with other relevant prior works. In Section 3 we describe our method in detail. By both qualitatively and quantitatively evaluating the generated scenes along with other experimental results, we showcase the potential and limitations of our approach in Section 4.

1.2 Challenges

To achieve significant learning performance from natural image data is a difficult task. Considering the nature of our approach, we identify the following areas that will likely pose considerable challenges to the effectiveness of the method.

- **Unsupervised learning** Without any human annotation, the model has to find a way to learn from unlabeled complex image data solely and to generalize well to unseen data.
- **Learning object representations** The model should learn to extract low dimensional object representations that correlate well with the image input

and capture the most relevant semantic content thereof to leverage downstream applications. The training images contain a varying number of objects at different locations which will significantly increase the learning difficulty.

- **Disentangling factors of variation** Ideally, the learnt object representations should expose a structure that consists of disentangled factors of variation. Each factor would represent an object or an object attribute from image space.

- **Unaligned natural dataset** A dataset with aligned data is a desirable property for a machine learning model because the amount of image data variability the model has to cope with is considerably lower than in a natural dataset where objects occur almost randomly in any variation. It can be observed in many representation learning papers that methods most often use relatively simple or even synthetic datasets for experiments. For instance, celebA [54] is a face dataset where an image contains exactly one face which is often center aligned. In contrast, our model is applied to a natural dataset in which the data (e.g. objects) is inherently unaligned and comes with a high variability. Adding to the difficulty, the model makes the assumption that a given image contains up to four objects where each object is approximately aligned in one of the quadrants. The underlying dataset will therefore pose a substantial challenge to the model in the learning process.

- **Generalization** Given the challenges mentioned above, how well will the model generalize and perform on unseen data and other datasets, respectively? Ultimately we want a model which, once trained, works well at inference time on new inputs.

1.3 Contributions

The main contributions of our work are the following:

- A novel method called LORBMS to learn object representations from unlabeled data in a completely unsupervised manner.

- A novel visual similarity detection algorithm.

- An implementation of the proposed method.

- Engineering of neural architectures for the components of the model.

- An implementation of the proposed detection algorithm on top of the MS COCO dataset.
- Extensive hyperparameter tuning in search of an optimal model implementation.
- A qualitative and quantitative study of the capabilities and limitations of the proposed method.

Chapter 2

Preliminaries and Related Work

In this section, we first give a brief overview of neural networks. We start with multi-layer perceptrons and then focus on convolutional neural networks and autoencoders as two relevant types of neural networks. Secondly, we give an introduction to generative adversarial networks and a number of variations thereof. Thirdly, we review the two research areas this work is rooted in, namely *representation learning* in general and *unsupervised disentangling factors of variation* in particular. For the latter, we give an overview of previous major approaches along with their characteristics to establish the contribution and relevance of our work. Finally, we present a previous method as the groundwork on which our method is built.

2.1 Multi-layer perceptron (MLP)

The multi-layer perceptron (MLP) [70, 73] is a generic and probably the simplest type of feedforward neural network. The MLP consists of an input and an output layer and one or more hidden layers in between. A layer consists of a set of neurons. A neuron is the basic unit of a neural network and represents a mathematical function that receives one or more inputs and sums them to produce an output, the activation of that neuron. Each input is first multiplied by a weight belonging to the neuron. The set of weights of a neuron are learnt during training of the network by a process called *backpropagation* [51, 73]. Backpropagation is the mechanism by which the weights of every neuron in a network are updated for each set of training data (i.e. a *training batch* contains one or usually a multiple of two samples) such that the accuracy of the decision the network makes for that set of data is improved. Every neuron of each layer is connected with all neurons of the previous layer. Each layer of an MLP is what is called a *fully connected layer* and can be written as $y = \sigma(Wx + b)$ where W is a weight matrix, x the

input, b the bias and σ an activation function, also called nonlinearity. Each layer and each neuron, respectively, thus performs a dot product between the input and its weights and adds the bias. The result is passed to an activation function which introduces nonlinearity to the layer. In fact, the activation function is what makes a stack of fully connected layers exploitable in the first place. Without nonlinearity, all the fully connected layers in an MLP, with each layer being a linear function, simply represent a linear combination which is still a linear function in itself and could be collapsed into one single layer again. Therefore, the nonlinearity is what makes the multi-layer perceptron inherently nonlinear and gives it its expressive power. A single fully connected layer can also be used as a *single layer classifier*, a type of *linear classifier*. For instance, this is useful in transfer learning experiments as demonstrated later. An important characteristic of an MLP is stated by the universal approximation theorem [26]. It essentially says a feedforward neural network with at least one hidden layer, sufficient neurons and a suitable activation function can approximate any possible continuous function. So an MLP can be thought of as a *universal function approximator*. We can thus exploit the expressive power of neural networks to tackle problems and tasks yet unsolved and unachieved, respectively. The MLP is a typical example of a deep learning model and is usually trained using the stochastic gradient descent (SGD) optimization method.

A reason why we do not always go to an MLP to accomplish a task is their high demand in computational resources and memory resources as the size of the network grows (i.e. number of neurons). For instance, each layer, which is always a fully connected layer, has a fixed number of neurons each of which is connected to all neurons of the previous layer which results in a considerable amount of weights per neuron and equally many computations in the forward as well as the backward pass while training. While the architecture is simple in its structure it is inflexible in some respects. Every neurons' receptive field spans the entire input and it is not possible to limit the receptive field of a neuron (see Section 2.2.5). MLPs are great for approximating many nonlinear functions but not necessarily a good choice for computationally and memory intensive tasks such as image processing. For the latter case, a convolutional neural network (CNN) is a more suitable choice. We describe more specific types of neural networks such as CNNs, autoencoders and GANs in the following sections.

2.2 Convolutional neural network (CNN)

2.2.1 Evolution

Deep convolutional neural networks (CNNs), also called *convnets*, are among the most common type of neural networks. With respect to image processing and in comparison to MLP networks, CNNs have achieved greater success and are more time and space efficient as explained later. As a result, CNNs are ubiquitous in computer vision applications. In 1979, Fukushima [18] and in 1989, Le Cun *et al.* [13] laid significant groundwork for CNNs.[1] But it was not until 2012, when Krizhevsky *et al.* [41] released a CNN model which went on to win the annual *ImageNet* large-scale visual object recognition challenge (ILSVRC) [14] with an unparalleled accuracy and by a significant margin, that CNNs gained widespread research attention in fields such as object detection [69], image recognition [28, 41], semantic segmentation [7], instance segmentation [29] and image captioning [1]. CNNs as a key ingredient in the respective architectures have helped to achieve unprecedented state-of-the-art (SOTA) performances. And CNNs continue to do so today. A reason for the success of CNNs on image data could lie in their ability to act as automatic feature extractors, where a feature (or representation) is a condensed piece of information that represents an essential semantic content of an image. A feature is used as a core input for many downstream tasks. In contrast, strictly mathematical approaches to understanding image data have not been as successful. Probably because the rigidity of mathematics does not match as well with the inherent complexity of natural image data. CNNs as universal function approximators and feature extractors seem to have adapted much better to this type of data and tasks, respectively.

2.2.2 Architecture and design

The design of a CNN architecture is not a rigorously defined procedure but rather an art depending on the task to solve. For instance, given an image recognition task, a CNN is followed by a classifier often in the form of an MLP. In contrast, to obtain a generative model, a CNN would be followed by a decoder. A CNN architecture therefore is the result of an assembly of a number of individual components and layers, respectively. Those are used across a wide range of different types of CNNs. Considering Figure 2.1, we describe relevant CNN components along with their roles within the architecture next.

[1] This is not an account of the history of CNNs.

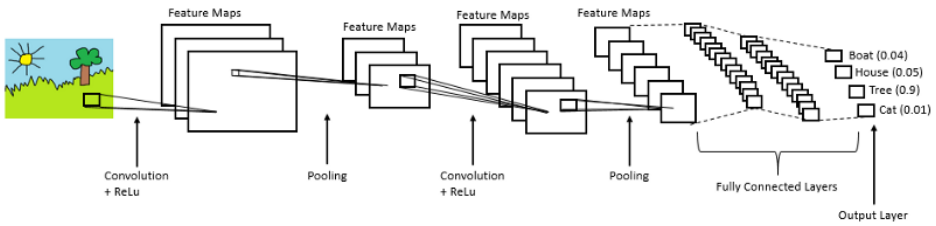


Figure 2.1: A simple CNN architecture for image recognition [65].

Convolutional (conv) layer Performs the convolution operation between its input, which is either the input image itself or the output of the previous layer, and its kernels. The kernels are the set of weights that belong to a layer. The result of the convolution is called the feature map or activation map and stores the information if and where a certain feature occured in the input. That is the intuition, the convolutional layer looks for a specific local structure and does so across the entire input. The orchestration of a number of convolutional layers will result in the ability of the CNN to detect higher level features such as objects. Formally, the 2D convolution at location (i, j) on an input I with C channels is defined as

$$Y(i,j) = \sum_{c,n,m} I(c, i \cdot s - n, j \cdot s - m) K(c, n, m), \tag{2.1}$$

where $K \in \mathbb{R}^{C \times N \times M}$ is the kernel with size $N \times M$ and s the stride. Note that a convolutional layer usually has multiple kernels which will result in an output $Y(k, i, j)$ where k is the number of kernels and the number of feature maps in the output, respectively. See Section 2.2.3 for more details.

Kernel/filter Holds the weights of a convolutional layer. The kernel is shared between all neurons of the convolutional layer unlike in a fully connected layer where each neuron has its own sets of weights. The number of kernels a layer has (a hyperparameter) depends on the chosen depth of the output volume such that the number of kernels equals the number of output channels. The kernel is applied to the input by using a dot product. One dot product results in a scalar which represents exactly one value in the resulting feature map. The size of the kernel is a hyperparameter and is often one, three or five. Intuitively, each kernel is used to learn to detect a certain feature across the input volume.

Stride Defines the step size of the kernel as it slides across the input volume. It is often one or two. With a stride greater than one, the spatial output dimension of the convolution will always be smaller than the input. Thus a convolutional layer can also be used to reduce the spatial dimension of an input as an alternative to the pooling layer.

Padding The padding controls the output size of the feature map of a convolutional layer. If the padding is 0 (in ML frameworks often denoted by *VALID*), the kernel stays inside the input dimension and the output size will be smaller. A padding greater than 0 (in ML frameworks often denoted by *SAME*) causes the input volume to be padded with 0s such that the output size of the convolution is the same as the input size.

Normalization layer Oftentimes a convolutional layer is followed by a normalization layer before the activation function. Normalization of the activations of a convolutional layer can help make the learning process of the network more effective. Common normalization types are *batch norm* [32], *instance norm* [78] and *spectral norm* [60]. Which type to use depends on the task at hand. Not always is the use of a normalization layer effective. For instance, in an encoder aiming at disentangling factors of variation, batch norm would be counterproductive as it normalizes activations across the training batch smoothing out too many details whereas instance norm works on one training example only and thus could be a better choice. See Figure 2.2. Batch norm, however, works great with CNNs for image classification.

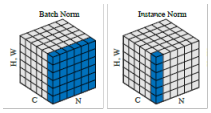


Figure 2.2: Batch vs. instance norm [80]. N is the batch size.

Non-linearity layer (activation function) A nonlinearity layer consists of an activation function which takes the output of a (convolutional) layer and processes it element-wise to produce the activation map[2]. The activation function essentially enables the CNN to learn a nonlinear function. Without it, a sequence of convolutional layers would collapse into one linear function and building deeper models would be pointless. As depicted in Figure 2.3, common activation functions are sigmoid, tanh, softmax and the rectified linear unit (ReLU), likely the most common. The softmax function does not take a single element as input but a vector of K real numbers and normalizes them to a probability distribution consisting of K probabilities that sum to one.It is often used as the last layer in a single-label classification network.

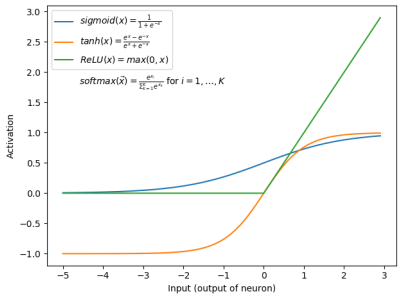


Figure 2.3: Common activation functions.

[2]The terms *activation map* and *feature map* are often used interchangeably in the literature.

Pooling layer The pooling layer's main responsibility is to reduce the spatial size of the activation maps. The idea is to drop irrelevant information so as to keep the most important activations only, thereby to focus on the "essence" of a feature and getting rid of the rest to ultimately perform tasks based on very compact but highly decisive information. Common types are max pooling and average pooling. The output of a pooling layer will be the same independently of where a specific feature is located inside its pooling region as the maximum or the average will be the same. This contributes to the translation-invariant property of convnets.[3] Note that the pooling operation also helps to build the abstraction hierarchy in the convnet as it drops irrelevant information, keeps the decisive information and thus makes the retained information more "abstract".

Fully connected layer As illustrated in Figure 2.1 the feature maps resulting from the convolutional layers are passed to a fully connected layer. In fact, the last three layers represent an MLP as described in Section 2.1 except that the input is not a vector but the activation maps (a 3D tensor) from the previous layer. The fully connected layers here function as a classifier that receive the high level features from the convolutional layers as input and output a class probability distribution representing the class predictions for the input image.

Output layer The last layer in a CNN (and any neural network) is called the output layer. As for a classification task like in Figure 2.1, it is a fully connected layer where the number of neurons equals the number of classes. For a multi-class (single-label) classification, the softmax function is then used to "squash" the output values to a probability distribution that sums to one.These predictions present the final output of the CNN. To measure the accuracy of the output, the *cross-entropy loss* is used together with the actual labels, the ground truth. The cross-entropy loss is defined as

$$\mathcal{L}_{ce}(p, y) = -\sum_{i=1}^{C} y_i \log(p_i), \tag{2.2}$$

where y is the ground truth, p the predictions of the network and C the number of classes. In case of a multi-label classification, the sigmoid function is used to compute an independent probability for each class. Thus each output value of the network is an independent prediction for a specific class. Here, to measure the

[3]Note that the convolution operation itself is not translation-invariant, but rather translation-*equivariant*. The output of the convolution translates in the same way as the input. For instance, when a dog moves spatially in the input, the corresponding activations in the output also move spatially. However, the convnet as a whole (with pooling etc.) allows one to build a function that is translation-invariant w.r.t. the input (e.g. an object classifier).

accuracy of the output, the *binary cross-entropy loss* is used for each predicted value separately. Hence $C = 2$ and the cross entropy loss becomes

$$\mathcal{L}_{bce}(p, y) = -(y \log(p) + (1 - y) \log(1 - p)). \tag{2.3}$$

For a regression task, in contrast, the CNN is used to predict a quantity and not a label. The output layer would consist of only one neuron and one value, respectively. For instance, a CNN is used to predict the age of a patient given a brain image. Other scenarios involving multiple neurons are possible too.

2.2.3 Inner workings of a convolutional layer

The convolutional layer is the main building block of a CNN. For simplicity we will only consider image data as input to a CNN. This implies that any input is three dimensional $h \times w \times c$ where h is the height of the image, w the width and c the number of channels, which for colored images is usually three or four and for grey images one. In subsequent layers, however, c can take on any number and represents the number of feature and activation maps, respectively, that a layer outputs. The convolutional layer receives as input a three dimensional tensor and produces as output a three dimensional tensor. Each dimension in the output can be different from the respective dimension in the input. When a convolution is applied, the spatial dimension of the tensor is often reduced depending on the kernel size and the stride. Simultaneously, the depth dimension c is often increased to compensate for the reduced spatial dimension. Therefore, the spatial input information lost in the convolution operation "lives on" in the output, namely in the depth dimension of the activation maps. A kernel of a layer looks for a distinct feature in the input. Upon detection, the kernel is activated by the feature and produces the activation map. Each activation map stores the occurrence of a distinct feature detected in the input tensor (e.g. an edge, a texture, an object) and carries that information to the subsequent layer. Consequently, the depth dimension should be large enough to account for a sufficient amount of distinct features to detect in the input. To showcase the inner workings of a convolutional layer i consider Figure 2.4. Layer i receives as input the output tensor of the previous layer $i - 1$ and produces itself an output which becomes the input to the subsequent layer $i + 1$. The variable c in the output volume refers to the number of output channels and activation maps, respectively. Each activation map represents the result of convolving (or sliding) each of the kernels (also filter) across each input channel of dimension $h \times w$ and captures the feature the kernel has learnt to detect. If a kernel detects edges, for instance, the activation map will only be nonzero if the input volume contains an edge that activates the kernel. The same kernel is convolved over the entire spatial dimension of the input volume

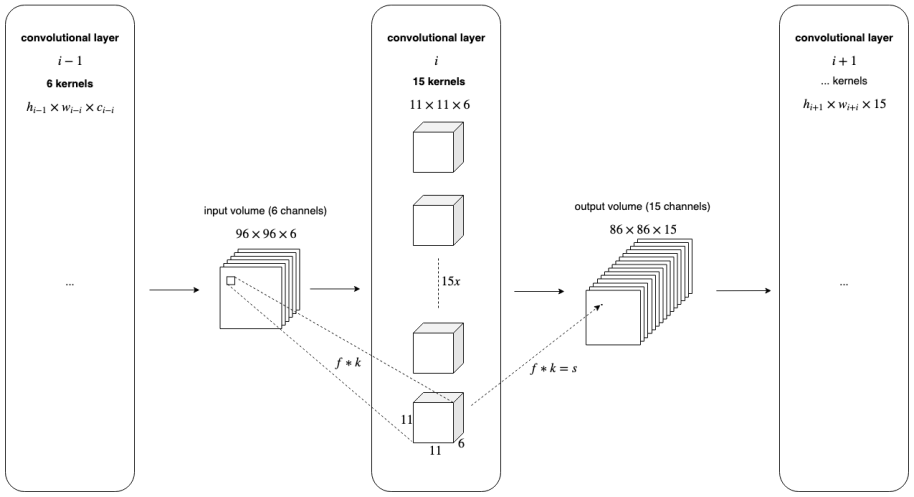


Figure 2.4: A convolutional layer i with its input and output volume. By convolving each of the 15 kernels of i across the input volume, the 15 activation maps (channels) of the output volume result. Each kernel has the same depth of six as the input volume. As depicted, one single convolution operation $f * k$ produces exactly one value in one activation map. f is the receptive field of s and equals the size of k. Best viewed on screen with zoom.

hence it will detect an edge irrespective of its spatial location (i.e. the translation-equivariant property of the convolutional layer). The fact that the kernel is reused means its weights are shared between the convolutions across an input. This is what lies behind the notion of "weight sharing" in a convolutional layer. The kernel weights are learnt and optimized, respectively, during training of the network. Note that the depth of the kernel (e.g. six in Figure 2.4) always equals the depth of the input volume because the convolution operation, that is, the dot product of the receptive field of the kernel in the input volume with the kernel,[4] always spans the entire depth dimension.

With each subsequent layer, the convnet can detect more elaborate features. In the first few layers of the network the features detected are typically low level such as edges or curves. As these features are combined and processed together in subsequent layers, the features learnt become more high level such as objects and even later represent abstract concepts such as faces, arms, legs [83].

A sequence of convolutional layers as in Figure 2.4 represents a series of linear functions which is again a linear function. In order to allow it to model nonlinear functions, each convolutional layer is followed by an activation function (e.g. ReLU). The activation function is what introduces the nonlinearity to the CNN,

[4]Technically, this is a cross-correlation, a function related to convolution and used instead in many neural network libraries.

analogous to the MLP.

Time and space complexity The convolutional layer is much more time and space efficient in comparison with a fully connected layer. In the convolutional layer each neuron is connected to only a few neurons of the previous layer (weight sharing) unlike a fully connected layer where each neuron is connected to all the neurons of the previous layer (no weight sharing). This impacts not only the number of computations (time) to perform in a forward or backward pass but also the number of weights to keep in memory (space).

2.2.4 AlexNet: case study of a seminal CNN

As a real world example, consider the convnet named *AlexNet* [41] in Figure 2.5. AlexNet consists of eight layers with weights where the first five are conv

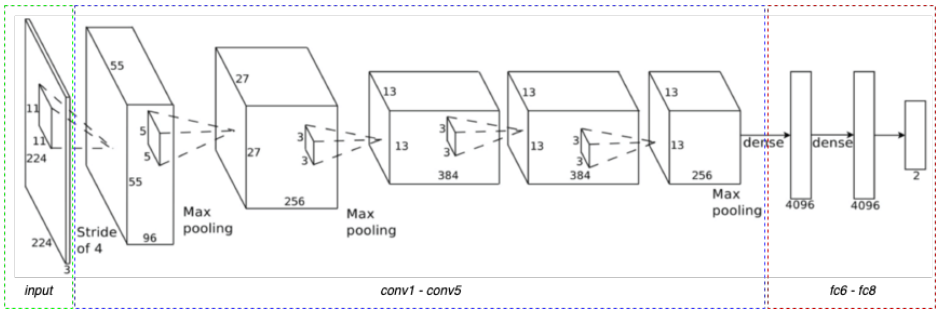


Figure 2.5: The AlexNet convnet [41].

layers (*conv1 - conv5*) followed by three fully connected layers (*fc6 - fc8*). The latter part is essentially an MLP classifier. AlexNet has 62.3 million trainable parameters of which only 3.7 million (6%) come from the convolutional layers. The *input* is an image of size $224 \times 224 \times 3$ (in fact, 227 pixels were used in [41] unlike stated in Figure 2.5). Across the five conv layers, the main part of the feature extraction with respect to the input takes place. In this process the spatial resolution of the input is reduced from 227×227 to 6×6 using kernels of decreasing size before the resulting $6 \times 6 \times 256$ activation map is passed to the classifier. To introduce nonlinearity to the network, all the eight layers are followed by the ReLU activation function except fc8 which is followed by the softmax function. Moreover, conv1 and conv2 are followed by normalization layers to help in generalization (however, they are not commonly used there anymore). To help reduce spatial resolution and as a measure against overfitting, max-pooling layers are employed after conv1, conv2 and conv5.

AlexNet as presented in Figure 2.5 is a nonlinear image classifier. Under the hood the network can also be considered a concatenation of a feature extractor

(conv1 - conv5) and a nonlinear classifier (fc6 - fc8). The feature extractor part, for instance, can be used for transfer learning. AlexNet is still widely used today as a well established and de facto standard convnet.

2.2.5 Receptive field and its implications

In fully connected networks, the output value of any neuron depends on the entire input to the network, that is all the pixels in case of an image input. In contrast, the output value of a neuron[5] in a CNN only depends on a region of the input image, the so called *receptive field* (RF) [45], or *field of view*, for that neuron. Only the region defined by the receptive field of a neuron impacts the output value of that neuron. Anything outside that region does not affect the outcome. Note that in case of an MLP the receptive field of any neuron always spans the entire input image. We assume the RF is square-shaped and the size of each side is given in pixels. The receptive field of a neuron can be considered with respect to the previous layer or with respect to the input image (or any of the previous layers). Considering only the previous layer, the receptive field of the current layer is equal to its filter size. It follows that only if the filter size of a layer is greater than 1×1 will the receptive field increase. Assuming that is the case for all layers, the network starts off with a very small receptive field that allows it to focus on low level features such as edges or curves. The next layer then looks at a combination of receptive fields from the previous layer which results in a larger receptive field. As the network goes deeper, the receptive field continues to increase and so each neuron of the next layer looks at yet a larger region of the input image and is therefore capable of learning more abstract features such that the network can eventually learn high level features such as objects or regions. In conclusion, with respect to the input image, the receptive field size of any layer in the CNN is a function of the number of previous layers along with the filter size and stride of each layer [49].

What needs to be ensured with respect to the receptive field when designing a convolutional network is that it covers the entire relevant region of the input image. What the relevant region is depends on the application. In case of a classification, object detection or a segmentation problem the receptive field of the CNN should span the entire input image. The size, therefore, should not be smaller than the object or part to detect, otherwise important information is left out. Likewise, the field can be defined to be as big as the input image or larger such that each output value in the last layer is affected by the whole input image. With such a large field it might be difficult for a CNN to disentangle multiple objects in the input image if

[5] A CNN neuron is equivalent to the dot product of a filter of a convolutional layer with the input tensor.

every pixel of the input image maps to every element of the representation. Thus it is crucial to consider the receptive field and its size, respectively, with respect to the problem at hand when designing a convolution network.

2.2.6 Unsupervised training of CNNs

CNNs are often trained in a supervised setting where human annotated labels are available to drive the learning process. A major challenge in the realm of unsupervised learning is the lack of such labels. In supervised learning the availability of labels as ground truth (usually by human annotation) allows us to define loss functions that measure the accuracy of the prediction with respect to the label and thereby providing an error signal to perform backpropagation with and realize the learning progress, respectively. Using labels we can easily train a CNN as a regression or classification task. In unsupervised learning, however, this is not as straightforward since ground truth is nonexistent. The task is still to train a CNN but in an unsupervised manner. A widespread method in unsupervised learning is clustering. A recent approach using clustering is presented in the work by Caron *et al.* [6]. Their novel *DeepCluster* technique is used to perform unsupervised training of CNNs as shown in Figure 2.6. In a first step, a (randomly initialized) convnet without the classifier is taken to produce the representations for all images from a large dataset by simple forward passing. Those representations are then grouped using the clustering algorithm k-means into k clusters. When comparing the images of the representations, the representations belonging to the same cluster will also be visually similar in the image space. Similar on a semantic level but not necessarily on a pixel level. Each of those clusters then becomes the pseudo-label for all the representations belonging to that cluster. In a second step, with the classifier appended to the convnet, the convnet is trained in a supervised manner using the pseudo-labels, cross-entropy loss and backpropagation. The convnet has to predict the correct "cluster" for a given image. After each epoch, this two step process is repeated for a number of times. What results is a trained convnet that is capable of extracting visual features from images and achieves notable performance in standard tasks such as classification and object detection.

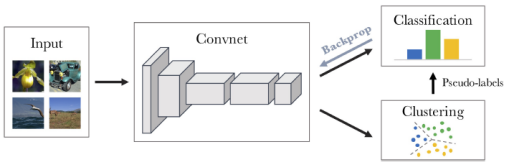


Figure 2.6: DeepCluster unsupervised training of CNNs [6].

2.3 Autoencoder

2.3.1 Overview

An autoencoder [3, 73] is a type of neural network and can be used for instance as a generative model, for representation learning or dimensionality reduction. An autoencoder consists of an encoder network f_θ and a decoder network g_θ as depicted in Figure 2.7. The encoder takes a high dimensional input instance x and encodes it into a latent space representation z. Its task is to encode as much of the relevant information from the input into the compressed z (while leaving the irrelevant information out) in order to allow the decoder to reconstruct the input x as $\hat{x}$ as faithfully as possible. A common objective function for an autoencoder is

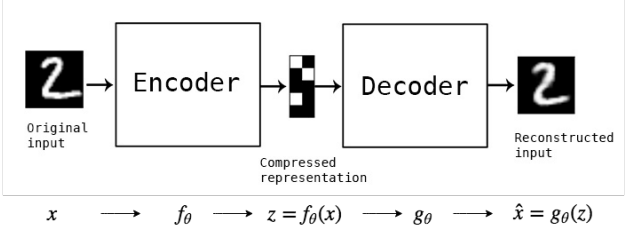


Figure 2.7: Overview of an autoencoder [10].

to minimize the pixel-wise reconstruction loss $\|x - \hat{x}\|^2$, also called squared error. Both networks are trained jointly using SGD. Note that this architecture does not enforce any kind of structure on the latent representation. Both the encoder and decoder networks can be implemented as MLPs or CNNs depending on the type of input data. For image data, CNNs are used normally.

Autoencoders are often used in unsupervised learning methods. An autoencoder learns in a self-supervised manner in which the labels (or ground truth) are generated from the input data. In a standard autoencoder the label is exactly the input because the autoencoder tries to reproduce the input as closely as possible. In this setting no supervision for learning is required.

Autoencoders are applied in many contexts. The encoder is often used as a *feature extractor* where the representation z being the "feature extract" is used for downstream tasks other than reproducing the input. The decoder can be used as a generative model whose input z' may be different from the z produced by the encoder to generate a new x'. Autoencoders play an important role not only in our work but also in the work by Hu *et al.* [27]. In particular, they make use of a *mixing autoencoder* which we describe next.

2.3.2 Mixing Autoencoder

A mixing autoencoder, a term introduced by Reed *et al.* [67] and coined in [27], refers to a neural network architecture with an autoencoder where multiple instances of the encoder are used in parallel to project a set of data points to latent space. An example is shown in Figure 2.8. All encoder instances share one set of weights. The latent space representations are then mixed in some way and finally projected back onto data space using the decoder. As with the encoder, multiple instances of the decoder with shared weights can exist. As a result of the latent space mixing, the output x_3 can mean any kind of mixture of the inputs x_1, x_2.

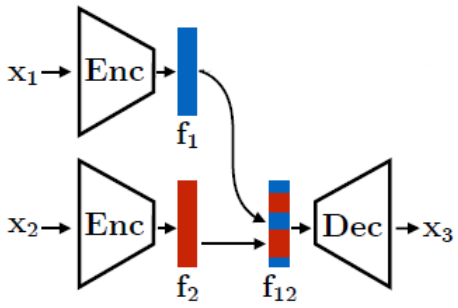


Figure 2.8: Schematic of a mixing autoencoder [27].

Mixing autoencoders provide a structural prior for disentangling latent representations and are used as an integral part of the respective method in [27, 67] as well as in our work.

2.4 Generative adversarial network (GAN)

2.4.1 Vanilla GAN

This type of neural network, the so-called Vanilla GAN, was introduced in 2014 by Goodfellow *et al.* [20] and has inspired a significant amount of research and progress ever since. It is called "the coolest idea in deep learning in the last 20 years" by AI pioneer Yann LeCun and "a significant and fundamental advance" by AI luminary Andrew Ng. The GAN defines a learning framework for synthetic image generation in which two neural networks, usually convolutional, namely the generator G (a generative model) and the discriminator D (a discriminative model), compete against each other in a two-player game. A schematic is illustrated in Figure 2.9. The framework also includes a dataset with real images. These images are used by the discriminator to learn to discriminate real

from fake images. Real images come from the dataset, fake images come from the generator. Further, the dataset is defined by an implicit probability data distribution p_{data} which all its samples follow. The generator's objective is to learn and approximate, respectively, the data distribution p_{data} and by that generate new, realistic images that look similar to samples from the dataset. Thus the generator implicitly defines a probability model distribution p_{model} as the distribution of the samples obtained from the generator given some input z, often a Gaussian noise. In other words, the discriminator learns to determine whether a sample is from p_{data} or p_{model} and the generator learns to invent samples from p_{model} that closely resemble samples from p_{data} and ultimately confuse D in its decision.

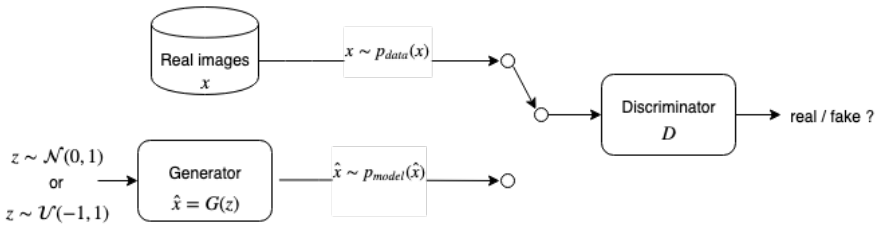


Figure 2.9: The GAN two-player game between generator and discriminator.

Even though so far GANs have received the most research on image data, the idea of learning a data distribution is more general and should lend itself to other types of input. Generally, at inference time only the generator is used to create new images. However, the discriminator can also serve other purposes such as a classifier [72] or as a feature extractor for transfer learning [35].

Minimax game The minimax game comes into play in the learning phase. The two models are simultaneously trained so that G learns to generate samples that are hard to classify correctly by D, while D learns to discriminate the samples generated by G. As training evolves, D helps G to produce even better samples. This setting can be viewed as a two-player game where both players (models) try to minimize their own loss and the final state of the game is the Nash equilibrium where both players cannot improve anymore. Ideally, when G is fully trained, it should not be possible for D to perform any better than randomly guessing. The two-player minimax game is formally defined as

$$\min_G \max_D V(D,G) = \mathbb{E}_{x \sim p_{data}(x)}\big[log\, D(x)\big] + \mathbb{E}_{z \sim p_z(z)}\big[\log(1 - D(G(z)))\big] \quad (2.4)$$

where z is a random noise. G tries to minimize $\log(1 - D(G(z))$ (or maximize $D(G(z))$) such that D gives a high probability for input $G(z)$, while D tries to maximize $\log(D(x)) + \log(1 - D(G(z))$ with a high probability for x and a low probability for $G(z)$. The two networks are trained in an alternating manner: for

every training batch, one network is fixed and the other is updated so that back-propagation happens in one network at a time exclusively. In an ideal convergence scenario, G has learnt to produce samples so realistic that D cannot discern them from real images and assigns probability 0.5 to every image (Nash equilibrium). In this state, D cannot teach G anymore how to improve, convergence has occured. Note that practically the expected value is the average over the training images x.

GAN loss functions Multiple formulations of the GAN loss functions exist [44]. In this work we focus on the non-saturating GAN loss introduced in [20] as the empirical evaluation in [44] suggests to favor this version when applying GANs to a new dataset. The hinge loss version has also shown good performance in [60] yet [44] concludes that it performs very similar to the non-saturating loss.

The discriminator loss function is

$$\mathcal{L}_D = -\mathbb{E}_{x\sim p_{data}(x)}\big[\log(D(x))\big] - \mathbb{E}_{\hat{x}\sim p_{model}(\hat{x})}\big[\log(1-D(\hat{x}))\big] \tag{2.5}$$

where p_{data} denotes the (true) data distribution and p_{model} the model distribution. It is equivalent to the binary cross entropy between the data and the model distributions of real and generated images (D has to discriminate between two classes).

The generator loss function is

$$\mathcal{L}_G = -\mathbb{E}_{\hat{x}\sim p_{model}(\hat{x})}\big[\log(D(\hat{x}))\big] \tag{2.6}$$

where the generator creates samples $\hat{x} = G(z)$ which the discriminator believes to be real $D(\hat{x}) \approx 1$ and thereby trying to approach $\log(1) = 0$.

Joint loss function Similar to our work, a number of other methods [27, 56, 63] use a joint loss function for the generator which is composed of two or more losses and oftentimes includes a reconstruction loss and an adversarial loss $\mathcal{L}_G = \lambda_{rec}\mathcal{L}_{rec} + \lambda_{adv}\mathcal{L}^G_{adv} + \lambda_i\mathcal{L}_i$ where i is an additional method specific loss. To look up the weightings in other works proves useful to find reasonable values more efficiently. For instance, [63] uses $\lambda_{rec} = 0.999$ and $\lambda_{adv} = 0.001$ on ImageNet whereas [27] deploys $\lambda_{rec} = 30$ and $\lambda_{adv} = 1$ on celebA.

2.4.2 Significant GAN works

Here we summarize significant types of GANs that have evolved from the original GAN formulation and which have influenced our work. See [24] for a comprehensive list of other GAN works.

DCGAN Radford *et al.* [66] introduces a class of CNNs called deep convolutional generative adversarial networks (DCGANs), that have certain architectural constraints, and demonstrates for the first time that GANs are a strong candidate

for purely unsupervised learning. The authors propose a more stable set of architectures for training generative adversarial networks based on convolutional layers and give evidence that adversarial networks learn good representations of images. Moreover, while the vanilla GAN could not yet produce visually realistic CIFAR-10 like images, DCGAN for the first time shows that it is possible to generate authentic images using a GAN. According to Ian Goodfellow, DCGAN defines a quasi-standard generator network architecture. Accordingly, DCGAN has become the de facto backbone architecture for many other subsequent types of GANs.

SNGAN Miyato *et al.* [60] presents a novel weight normalization technique for stabilizing GAN training, the *spectral normalization* (SN). They propose to add SN to the GAN discriminator. This normalization technique also helps GANs to prevent a collapse at the beginning of training and to produce more diverse images, which is significant because the lack of diversity in image generation is a problem for GANs. GANs with SN achieve better or comparative inception scores (IS), a popular metric for measuring the quality of generated images introduced by Salimans *et al.* [72]. In our work, we use SN to help stabilize the GAN training as discussed later.

TTUR Heusel *et al.* [23] propose the two time-scale update rule (TTUR) to help stabilize the training of GANs. In TTUR, the generator and discriminator have separate learning rates unlike the original GAN training. With TTUR, they show, the GAN training essentially converges to a local Nash equilibrium meaning it is beneficial for a good learning dynamic. We make use of TTUR in our method to help stabilize our DCGAN training. Moreover, they introduce the *Fréchet Inception Distance* (FID), another metric to measure the quality of generated images. FID measures the similarity of the generated images with a set of real images. The FID metric is consistent with human judgment and unlike the inception score, it has the advantage that it takes into account real images.

SAGAN Zhang *et al.* [84] introduces the self-attention mechanism to convolutional GANs. It helps both the generator and the discriminator to model relationships between widely separated spatial regions. The self-attention module is thus a nonlocal operation. It complements the nature of convolutions in that they look at only a small local region of the input (see Section 2.2.5). Their proposed SAGAN achieves best FID and IS results on the ImageNet dataset. Another finding is that while [60] proposes to add spectral normalization to the GAN discriminator, adding it to the GAN generator improves the training further. We adopt this finding in our method where we add spectral normalization to the generator (i.e. decoder) as well.

DNA-GAN In [82], Xiao *et al.* propose the supervised DNA-GAN by using an analogy with the DNA double helix structure, where different kinds of traits are encoded in different DNA pieces. In the same way, they argue that different

visual attributes in an image are controlled by different attribute-relevant parts in the latent representation of that image. Similar to our model, for two parallel input images, they use an encoder to retrieve their representations and a decoder to generate new images whose realism is judged by a discriminator (GAN). Additionally, they use labels to support the disentanglement in the latent space where image pairs are required to have different labels for a certain attribute (e.g. smiling or not). The respective part of a label in the latent representations of the image pair is then swapped to obtain two crossbreed representations and images, respectively. DNA-GAN is able to disentangle latent representations and allow for interpolations on datasets such as celebA.

Conditional GAN, PatchGAN Isola *et al.* [33] use a conditional GAN to achieve impressive image-to-image translation. A conditional GAN (cGAN) learns a mapping from an observed image x and a random noise vector z to an output image y, $G : \{x, z\} \rightarrow y$. Moreover, they introduce the PatchGAN discriminator which penalizes at the scale of image patches rather than the entire image. Using a reconstruction loss in addition to the GAN loss, they argue that using $L1$ encourages less blurring than $L2$ and that since the $L1$ loss enforces correctness at the low frequency level, it suffices to have a discriminator that pays attention only to high-frequencies and to the structure of local image patches, respectively. The PatchGAN classifier decides for each patch in an image if it is real or fake. In our method, we make use of the PatchGAN discriminator.

INFOGAN Chen *et al.* [8] describe InfoGAN, one of the first GANs to learn disentangled representations in an unsupervised manner. On top of the GAN framework, InfoGAN maximizes the mutual information (from information theory) between a subset of the latent factors and the generated images. InfoGAN can disentangle both discrete and continuous latent factors and be applied to more complicated datasets such as celebA. The results suggest that generative modelling enhanced with a mutual information cost is a fruitful approach for learning disentangled representations. INFOGAN disentangles latent factors such as digit shape and rotation on MNIST handwritten digits [46], pose and lighting on 3D faces [64] or hair style and emotion on celebA [54]. They use DCGAN as implementation for stable GAN training.

2.5 Representation learning

Representation learning [4] or feature learning[6] is a field within machine learning that aims at automatically discovering and extracting representations from raw

[6]The terms *representation*, *feature* and *feature vector* are often used interchangeably. We use *representation* throughout.

data[7] which can be used as input for many downstream tasks. While technically a representation can simply be a vector of numbers, semantically a representation is of good quality when it contains the discriminative information from the input data in an organized way and with redundant information removed. This often results in a lower dimensional representation. A good representation is *expressive*, meaning that it can express many concepts from the data space. Such a representation can leverage predictors or classifiers as they can extract decisive information more easily. In that sense, a good representation is one which makes the downstream task easier to solve. Yet the downstream task might be unknown at the time of learning the representation. Some key properties of a good representation are *distributed*, *invariant* and *disentangled factors of variation*.

Distributed representation What we aim at is a distributed representation that has the capacity to represent an exponential number of concepts as opposed to a local (non-distributed) representation in which only linearly many concepts fit. In a local representation, one neuron represents one thing (one-to-one relationship). In contrast, many neurons represent one concept in a distributed representation and a neuron participates in the representation of many concepts (many-to-many relationship, see [25]). It follows that a distributed representation also lends itself better to the notion of generalization. When a neural network encounters unseen data at inference time, it has a better chance of representing that input with a distributed representation as parts can be reused to represent something new. Neural networks such as autoencoders mostly use distributed representations by design.

Invariant representation A representation is invariant if changes in the input data that are irrelevant to the concept being represented do not affect the representation itself. In other words, the representation remains unchanged as long as the input data is transformed in a way that is unconnected to the represented concept. While intuitively this property makes sense, it is difficult to achieve in practice as it is a challenge already to even learn a proper representation of a concept. The invariance property is also related to the disentangling property described next.

Disentangled factors of variation An important assumption for representations of real world data is that there are multiple *explanatory factors* or *factors of variation* that underlie the data generating distribution. An additional assumption is that there exists a *hierarchical organization of explanatory factors* in which more abstract concepts to describe the real world are defined in terms of less abstract ones lower in the hierarchy. Therein also lies an advantage of using deep architectures as they can potentially learn more abstract factors at higher layers. More abstract has the advantage that it is more *invariant* (i.e. robust) to local changes in the input (e.g. a cat is still a cat no matter the surrounding scene or

[7] I.e. real world data such as images, video and other sensor data.

the change in lighting). Therefore, a representation that comprises highly abstract concepts becomes a highly nonlinear function of the raw input. To identify and disentangle the explanatory factors is an important objective of representation learning and is further detailed in Section 2.7. Achieving this objective, as [4] proposes, should result in a good representation that is significantly more robust to the complex variations in real world data.

With respect to real world data, attempts to extract representations with rigorous explicit algorithms (e.g. pure math) have not been as promising as approximative approaches with neural networks. Significant in using deep learning is that the system can automatically perform feature engineering, otherwise a labor-intensive hand-crafted process including feature extraction, feature importance and feature selection. This allows to construct new applications faster and paves the way for more progress towards an AI system that can understand the world around it.

The notion of a representation is usually associated with a latent vector such as in an autoencoder. Autoencoders provide a suitable framework for representation learning. In our work we use mixing autoencoders (see Section 2.3.2) to drive the learning of object representations. Representation learning can be carried out in supervised or unsupervised form. In unsupervised learning only unlabeled data is used. Our method is fully unsupervised.

Some examples of representation learning models are variational autoencoder (VAE) [40], β-VAE [59], UNIT [52], InfoGAN [8] and DNA-GAN [82].

2.6 Manifold learning and latent space interpolation

Manifold is an important concept in machine learning based on which concepts such as latent space, latent vector, representation and latent space interpolation can be better understood.

A *manifold* is a geometric surface (a connected region) in the latent space of a manifold learning algorithm. Mathematically, in Goodfellow *et al.* [19], a manifold is a set of points, associated with a neighborhood around each point and from any given point, the manifold locally appears to be a Euclidean space. As exemplified in Figure 2.10, a typical example of a manifold in three dimensions is the surface of the Earth which we locally perceive as a two dimensional space when in fact it lives in three dimensions. A manifold has a much smaller dimensionality than the input data space. This is exploited in, for instance, autoencoders as is the case with our model. A manifold represents the input data in a condensed space with an intrinsic coordinate system that can be traversed and whose coordinates (i.e. points) can be

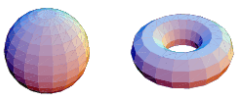


Figure 2.10: Examples of manifolds [74].

projected back onto input space. As a loose definition, a manifold consists of a connected set of points where each point is surrounded by a neighborhood of other points [19]. The existence of a neighborhood should allow for transformations to move on the manifold from one point to a neighboring one. However, the manifold, embedded in a higher-dimensional space, often uses only a small number of the data space dimensionality. Each dimension of the manifold corresponds to a local direction of variation. This implies when a dimension (or multiple) of a manifold latent vector is changed arithmetically (i.e. linearly) the right amount, a transformed latent vector will result that also lies on the manifold and that may yield nonlinear changes in the output image [38]. This is where latent space interpolation comes into play.

Manifold learning is based on the assumption that data probability mass is highly concentrated along a low-dimensional manifold (or a collection thereof) where all the valid inputs are projected to. This implies that most inputs from the data space yield latent space coordinates which lie outside a manifold and are therefore irrelevant. Interesting latent space interpolations and variations in the output, respectively, will occur only if we move along a direction that lies on the manifold or move from one manifold to another [19].

The *manifold hypothesis* states that the probability distribution over images, text strings, and audio that occur in real life is highly concentrated as well [4]. It is straightforward that an image created with pixel values uniformly distributed will unlikely result in a natural image. Also, it is likely that images resembling each other strongly in image space such as pairs of images with slightly rotated objects will also be reachable in latent space by traversing the corresponding manifold. The concept of latent space interpolation is about traversing a manifold.

Latent space interpolation is not a well defined operation. It relies on the vague notion of a "semantically meaningful combination" of latent vectors and also depends on the dataset. Essentially, a latent space interpolation is commonly implemented as a convex combination[8] of two latent vectors z_1, z_2 as follows: $\hat{x}_\alpha = g_\theta(\alpha z_1 + (1-\alpha)z_2)$ where $\alpha \in [0,1]$, g_θ is a decoder and $\hat{x}_\alpha$ the decoded output [5]. An ideal interpolation scenario with synthetic data is depicted in Figure 2.11.

Sample results of interpolations between the representations of natural images are shown in Figure 2.12.

[8]This is a linear combination of points where all coefficients are nonnegative and sum to 1.

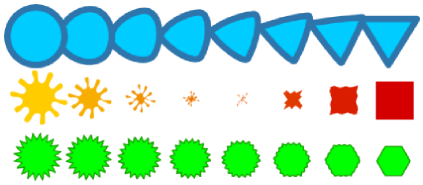


Figure 2.11: An ideal interpolation scenario [48]. As the latent vector of the original image on the left is interpolated and traversed on the manifold, respectively, the output image morphs into the original image on the right and vice versa.

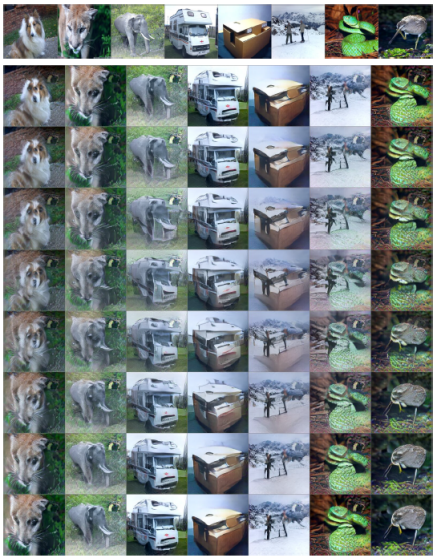


Figure 2.12: Interpolations between representations of real images [15]. The top row shows input images.

2.7 Disentangling factors of variation

Apart from being distributed and invariant, good representations should also capture factors of variation in the input space and disentangle (i.e. separate) them into different independent sections (or chunks, parts) of the representation [4]. A *factor of variation* (or *explanatory factor*) refers to a distinct and hidden source of influence with respect to the input data. A factor of variation explains part of the data. For instance, when analyzing an image of a car, the factors of variation include the position of the car, its color, and the angle and brightness of the sun [19].

One of the difficulties of a representation learning algorithm to disentangle factors of variation is when many of the factors influence a lot of the data observed. For instance, many natural images are composed of multiple objects with several

light sources and shadows overlapping in complex ways. Here the question arises how the objects can be disentangled from their shadows. An important measure to take, as [4] suggests, is to use as much data as possible to support learning representations that separate the various explanatory factors. In addition to using a large quantity of examples, deep learning is one of the foremost approaches to process complex natural data due to its ability to learn high-level abstract features from raw data by building complex concepts out of simpler concepts.

Further, it is important to distinguish between learning invariant representations and learning representations with disentangled explanatory factors. Invariant representations have reduced sensitivity to variation in the data that is irrelevant to the task at hand, yet internally the explanatory factors could all be represented in an entangled way. These representations may be intended for multiple tasks simultaneously which makes it difficult to learn them accordingly beforehand. Thus [4] concludes that the most robust approach to representation learning is to *disentangle as many factors as possible, discarding as little information about the data as is practical*.

A *disentangled representation* can be described as one which separates the factors of variation and explicitly represents the important attributes of the data [16]. In an ideal disentangled representation, each section capturing a factor of variation is invariant to changes in sections representing other explanatory factors.

Some applications for disentangled representations are [55]:

- directly build a predictive model instead of using the high dimensional input data
- use for learning of downstream tasks, transfer learning and few shot learning
- use for tasks involving latent space interpolations
- use when good interpretability is required

Ultimately, the goal is to learn representations that are invariant to irrelevant changes in the data [16].

Related work. Many works exist that have disentangled factors of variation in raw data. We present a number of significant previous contributions towards unsupervised disentangling next and relate them to our work. Note that many prior works on disentangling factors of variation are fully supervised. They use labels for all factors of variation to be disentangled. In contrast, our method is fully unsupervised with unlabeled data only. Furthermore, many previous methods work on synthetic datasets or datasets with aligned data, that is oftentimes one single centered object per image. Our method, in contrast, faces the challenge of disentangling factors of variation in natural, unaligned data. The related works

presented subsequently do not strongly connect with our method yet they have made significant contributions to the research field.

Cheung *et al.* [9] augment autoencoders (AE) with regularization terms during training. They use an unsupervised **cross-covariance penalty** (XCov) as a method to disentangle class-relevant signals (observed variables) from other factors in the latent variables along with a supervised cross-entropy loss. In case of MNIST they consider the class label as a high-level representation of its corresponding input. Their model learns a class invariant continuous latent representation z that encodes digit style (slant) whereas the observed variable $\hat{y}$ represents the digit itself. On the Toronto face dataset (TFD) dataset, the observed variable $\hat{y}$ encodes the facial expression while the autoencoder is able to retain the identity of the faces through latent variable z. Here the XCov penalty prevents facial expression label variation from 'leaking' into the latent representation.

Higgins *et al.* [59] introduce the **β-VAE** which is based on the variational autoencoder (VAE). β is an extra hyperparameter added to the standard VAE objective to encourage more disentangling in the latent representation. β-VAE is a method for unsupervised learning of disentangled representations. It is able to disentangle more latent factors than the basic VAE and in a similar capacity as the unsupervised INFOGAN. β-VAE achieves considerable disentanglement of multiple latent factors in various datasets including celebA, chairs [2] and faces.

Similar to β-VAE, Kumar *et al.* [43] propose **DIP-VAE**, another extension of the variational autoencoder for unsupervised learning of disentangled representations. Based on variational inference, they impose a regularizer on the VAE objective to encourage disentanglement. DIP-VAE outperforms β-VAE in disentangling latent factors as shown by quantitative metrics and visual latent traversals on the 2D shapes, celebA and chairs datasets.

Liu *et al.* [52] propose the unsupervised image-to-image translation framework **UNIT** based on VAEs and GANs (namely Coupled GANs). They try to map an image from one domain to a corresponding image in another domain (e.g. from day to night). UNIT assumes the existence of a shared latent space where corresponding images from different domains have the same latent representation. Based on that they build the UNIT framework to generate domain translated images. They show convincing results on tasks such as sunny to rainy, day to night, summery to snowy on a street images dataset and others. Unlike our method, the primary aim of this work is not disentangling latent representations but it shares a similar goal in being a generative model which, given two source images, produces a new image that is a mixture of the inputs.

Mathieu *et al.* [57] use a conditional generative model and labeled observations (images, audio) to extract latent representations that consist of specified and unspecified factors of variation. Similiar to UNIT, they use VAEs and GANs but with the aim of disentangling latent factors, like in our work. They show convinc-

ing interpolation results on MNIST, Sprites [67] and NORB [47] dataset.

Remez *et al.* [68] propose an instance segmentation architecture based on a Mask R-CNN and GAN discriminator that can **learn via cut-and-paste of image objects**. They use adversarial training to learn to generate segmentation masks for image objects. The realism of the resulting image with the pasted object is judged by a discriminator. Their approach is weakly supervised in that they use bounding boxes only for training (no supervision on the masks). The results on COCO [50] and Cityscapes [12] datasets show some accurate segmentation masks. This work presents a somewhat similar approach to ours in that both try to move objects around independently of the background. However, the two methods are clearly distinct approaches in that one works with segmentation and the other with mixing latent representations.

Eastwood and Williams [16] propose a **framework for the quantitative evaluation of disentangled representations** when the ground-truth factor of variation is available. The framework is limited to synthetic datasets. The authors state that reliable disentanglement is far from solved even in the restricted setting of the framework. Further, they state that there is at that time no quantitative benchmark for disentangled representation learning available. A more recent work [79] uses the Mutual Information Gap (MIG) metric to quantitatively evaluate disentanglement in representations. MIG requires the underlying factors of variation to be known and is therefore not applicable to evaluate an unsupervised method.

Watters *et al.* [79] present an architecture called **Spatial Broadcast Decoder** where they leverage a variational autoencoder. It aims at improving disentangling representations, reconstruction accuracy and generalization. Their key idea, as shown in Figure 2.13, is to use a different architecture for the decoder other than the common deconvolutional network. In fact, the decoder is designed of convolutional layers only. First, they broadcast (replicate) the latent space vector across the spatial dimensions and concatenate two channels to the third dimension. One channel contains the X coordinates and the other contains the Y coordinates (see Figure 2.13). Then a fully convolutional network with 1×1 stride is applied to reconstruct the input. With this architectural novelty they address the problem for a deconvolutional network to render an object at a particular position and argue that with the spatial broadcast decoder this becomes a simple task. Intuitively, the concatenated coordinate matrix should support the decoder in placin an object at the correct image location. Likewise, the coordinate sys-

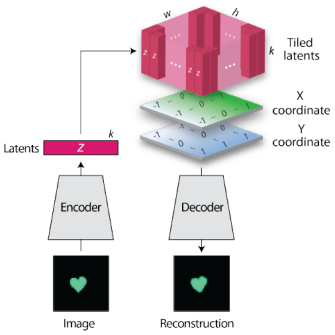


Figure 2.13: Spatial broadcast decoder [79].

tem encourages the encoder to encode and disentangle the location information of an object into the latent vector z such that the decoder can use this information together with the fixed coordinate system to determine the exact location in the output image. Put simply, if the encoder adds the location information as a (x, y) coordinate to the latent z, then the decoder can easily compute the location by matching that coordinate in the coordinate system. Experiments on synthetic datasets show encouraging results, however they do not demonstrate their method on more complex, natural data. The conclusion states that this decoder improves latent representations, most significantly for datasets with objects varying in position. The latter is an interesting feature as our method has to cope with non-aligned objects and the notion of object position is important, respectively. Say if the encoder in our system would explicitly encode the location information of an object into the latent representation, that could prove very beneficial for the task of mixing objects and scenes, respectively.

Greff *et al.* [21] describe the **binding problem**, a problem in the field of representation learning. It refers to ambiguities and interferences occuring in distributed representations of multiple objects from the same input. It can arise when multiple objects stemming from the same input are to be represented and disentangled at the same time. A proper disentanglement of objects might not be achieved. Where images contain only one object at a time the binding problem is avoided. They argue that the use of convolutions also mitigates the problem. Further, they propose an unsupervised method that explicitly models inputs as a composition of multiple objects. It dynamically binds features of different objects together and recovers these using the notion of mutual (pixel) predictability. Based on the latter, their framework uses clustering and a *denoising autoencoder* (DAE) to iteratively reconstruct the input and learn object representations, respectively. Even though they show promising results on binary images from artificial datasets (e.g. dSprites [58]), experiments on natural image data are left as future work. As Greff noted in a presentation, he would expect that the model would cluster by color but not by objects when exposed to a natural dataset.

In Table 2.1 we give an overview of a selection of prior works on disentangled representation learning. It shows that many of them work on simpler or even synthetic datasets.

2.7.1 Disentangling factors of variation by mixing them

Our method is based on the work of Hu *et al.* [27] and pushes its idea further. First we summarize the results of [77] and [76] as a theoretical background for [27] and then present the method itself.

Szabo *et al.* [77] identify two main challenges in disentangling factors of vari-

Method	Architecture	LT	Datasets
Reconstruction Clustering [21]	DAE, Clustering	us	Shapes, Bars, Corners, MNIST
β-VAE [59]	VAE	us	celebA, 3D chairs, 3D faces
DIP-VAE [43]	VAE	us	celebA, 3D chairs, 2D shapes
UNIT [52]	VAE, CoGAN	us	streets (own), celebA, SYNTHIA
Chen *et al.* [8]	InfoGAN	us	MNIST, 3D faces, celebA
Cheung *et al.* [9]	AE	ss	MNIST, TFD, Multi-PIE
DNA-GAN [82]	GAN	sv	celebA, Multi-PIE
Mathieu *et al.* [57]	VAE, GAN	sv, us	Sprites, MNIST, NORB

Table 2.1: Overview and characteristics of prior works on disentangled representation learning. (LT) learning type, (us) unsupervised, (ss) semi-supervised, (sv) supervised.

ation: the *shortcut problem* and the *reference ambiguity*. In the shortcut problem the model learns degenerate encodings in which all information is encoded only in one part of the representation, i.e. either in the component c for common attributes shared by both images or the component v for varying attributes. For an image pair there are two v and one c in latent space. The encoder maps a complete description of its input into vector N_v and the decoder completely ignores vector N_c. The second challenge, reference ambiguity, occurs when the attribute representation for an image is not guaranteed to follow the same interpretation on another image. In other words, the reference in which a factor is interpreted may depend on other factors which makes the attribute transfer[9] ambiguous. For example, the viewpoint angle of a vessel gets interpreted in two different ways depending on the boat type.

The two challenges identified in [77] are then addressed by Szabo *et al.* [76]. They introduce an adversarial weakly supervised training method that uses image triplets and fully tackles the shortcut problem by means of adversarial training. It includes a composite loss function consisting of an autoencoder loss and an adversarial loss. Used in conjunction, this function provably averts the shortcut problem. Concerning our work, this implies that the shortcut problem cannot occur in theory as our method is based also on a similar composite loss function. Furthermore, they analyze the reference ambiguity and prove that it is unavoidable when disentangling with only weak labels. The problem occurs when a decoder reproduces the data without satisfying the disentangling properties for the varying attribute v. Practically, this means not all the factors of variation can provably be disentangled from weakly labeled data (i.e. when only c is known for image pairs).

[9] i.e. the replacement of a feature chunk, which exposes an image attribute, by the same chunk from another image

Disentangling Factors of Variation by Mixing Them Hu *et al.* [27] present a novel unsupervised method to disentangle factors of variation without any data knowledge. The factors of variation correspond to image attributes such as the pose or color of objects. The disentangled representation consists of a fixed number of feature chunks where each chunk represents a factor of variation. The disentanglement is attained by a neural architecture including mixing autoencoders, a classifier and a GAN. A sequence of two mixing autoencoders is used to enforce disentanglement and invariance by mixing the feature chunks of input images x_1 and x_2 to generate a new image x_3, re-encode x_3 and using the obtained representation and its chunks, respectively, to reconstruct the representations of x_1 and x_2 to finally re-decode them for a reconstruction loss (see Figure 2.14). This two-time cycle of encoding and decoding helps establish the disentanglement and invariance by "channeling" each image attribute into a distinct feature chunk. Invariance means each image attribute is represented by one feature chunk which is invariant to changes in other chunks and for the decoding part affects only one image attribute. Invariance allows the system after disentangled encoding to create new mixes of factors of variation and decode them into new images semantically consistent with the dataset. The GAN establishes the adversarial training to ensure that the decoded images from the mixed features follow approximately the data generating distribution (cf. encoder Enc + decoder Dec + discriminator Dsc in Figure 2.14). To avoid the shortcut problem a classification constraint ensures that each feature chunk corresponds to a discernible image attribute such that degenerate encodings cannot occur (cf. classifier Cls in Figure 2.14).

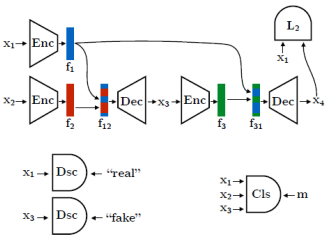


Figure 2.14: Architecture by Hu *et al.* [27].

Category	Theirs	Ours
Latent space	Attribute transfer	Object transfer
Representation mixing	Random mask	Scene mixing algorithm (Sec. 3.5)
Input	two images	five images (Sec. 3.4)
Data	Single object, aligned	Multiple object, unaligned
Dataset	Sprites, celebA	MS COCO
Classifier	Per feature match image	Per image match quadrant (Sec. 3.5)

Table 2.2: Differences between [27] and our method.

In the experiments they show qualitative results on two datasets including Sprites and celebA with attribute transfer of factors of variation such as pose, vest, hair color, leg (Sprites) and glasses, hair style, brightness, background (celebA).

Next we outline the differences between [27] and our method in Table 2.2. Notice that their system is designed for attribute transfer which disentangles and swaps attributes in aligned single object images whereas our system aims at object transfer that disentangles and swaps objects in unaligned natural image data. While the architecture as a whole is roughly the same in both works, the individual components differ substantially in their architecture as the challenges of a natural unaligned dataset call for exploration of other approaches.

Chapter 3

Learning Object Representations by Mixing Scenes

This chapter gives a full account of our method. We first develop and explain the underlying concept and continue with a formal definition thereof including the architecture. Next, we work out the loss functions of the model. A thorough description of the image processing algorithms our method is based on follows. We close the chapter by summarizing the implementation including relevant details.

3.1 Concept

As the main objective we want to build an unsupervised system that learns object representations by mixing objects from different scenes. The scenes and images, respectively, are to come from a unprocessed, unstructured natural dataset as it would occur in the real world. To this end, we want the system to identify and extract objects in images and represent them in a lower dimensional latent space in a disentangled and invariant state. By editing the latent space representation, we want to move objects around in a nonlinear way. That is the basic motivation for our system. Eventually, the edited representation is projected back to image space and by that a new scene is created, a mixed scene with a combination of objects from the input images. For instance, the system could extract an aircraft object from an airport, extract a lawn mower object in a public park, swap the two objects in the latent space and create two new mixed scenes with a lawn mower in an airport and an aircraft in a public park.

The system should therefore enable us to move objects around or swap them independently of their background and merge them into new scenes. We want to transfer objects between images simply by swapping the respective parts or chunks in the representation. Consequently, the chunks should be self-contained

and independent (i.e. disentangled) from each other. Ideally, the system should be able to cut-and-paste objects between different scenes where an object and its attributes (e.g. size) and its surrounding background are adapted in a nonlinear way in the target image. The function of the system should occur as a spatial object-to-object translation. The model should learn not only to identify objects in scenes but also to extract and merge them into different scenes with the result of generating a new scene. See Figure 3.1 for a schematic of the proposed system. While the synthesis of realistic mixed scenes is one goal, the ultimate ambition

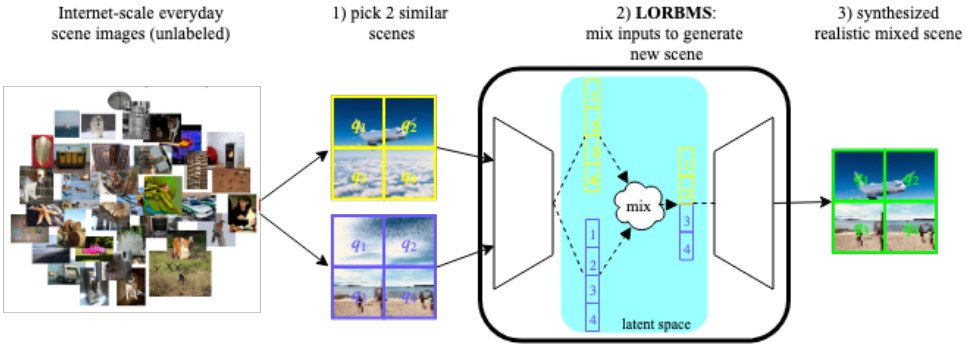


Figure 3.1: The proposed system moves objects around by mixing the latent space representations of the input images and thereby creates a new scene.

of our method is the object representation. An object representation is a high dimensional vector in the latent space and represents raw image data in a lower dimensional numeric form. The purpose in the end is to have a representation where a representation chunk (i.e. an object) can be taken from one representation and plugged into another and the image rendering adapts the content accordingly. We aim at an architecture where we can take images and mix the content in a selective way such that it gives us control over what we mix and eventually allows us to produce output images that correspond well with the mixing. Therefore, we want to move objects around such that if we move an object in the latent space we want to see it moved in the output image.

The model is built on the assumption that each of the quadrants of an image (i.e. spatial slots) contains at most one object as exemplified in Figure 3.2. We call it the *4 object assumption*. It is the strongest "inductive bias" of our model for inference on unseen data. The four object assumption confines the model to coarse object movements. The notion of coarseness depends of course on the size of the objects in an image. Note that the aim is not to segment objects but to move objects around wile making sure the result is realistic. Perhaps a parked car is copied onto a street but the car may not be in the right "placement". As the car is copied onto that street we expect the model to automatically put it in the right

placement (i.e. pose, size) with respect to the street. Or the system copies a person into a scene where the placement is further away, so in the target image the scale of the person has to be smaller. When a quadrant contains more than one object or just part of an object, the resulting mixed scene might suffer realism and not be as good as expected depending on what is mixed together. Whether the objects in an image belong to the same category or different categories is irrelevant to the model.

Figure 3.2: The four object assumption: at most one object per quadrant.

We expect that the system will face multiple key challenges in learning good object representations based on the task definition and assumptions made:

1. **Unaligned data.** Many prior works have shown that disentanglement works with aligned data. This means spatial location matters for disentangling factors of variation. As we use a natural dataset, the data is inherently non-aligned and spatial location of objects highly stochastic.

2. **Object size.** in most datasets used by other methods the objects have little variation in spatial size. Our system, however, faces natural data where objects take on any size in any spatial location.

3. **Proximity pixels.** An extra source of difficulty in extracting and merging an object into a new scene is its impact on pixels in its proximity such as shading or other partly occluded objects.

Essentially, our unsupervised model tries to discover structure in the dataset in order to learn its manifold. We then try to interpolate through it when mixing objects. We expect this to be a very difficult task as the model faces a natural dataset with a complex, high-dimensional data distribution and an accordingly complex manifold.

The input dataset is defined to be a natural dataset with a wide variety of objects and scenes. It should be as natural, unrefined and diverse as possible as

we want the model to understand the real world, the ultimate target for our method. Given that premise, we should mix objects only if they are sufficiently similar to support the learning process. We cannot mix images randomly. If we swap a person on a chair in a kitchen with a closeup view of a vegetable market stand, we cannot expect a realistic and meaningful outcome. The requirement thus is to mix meaningful or compatible pairs of objects only, then we should be able to see realistic output. We might expect at least to swap two different people, cars, buildings or animals with perhaps distinct poses and out should come a rendered image with that change. The method to determine compatible pairs of objects is detailed in Section 3.4.

Conceptually, the model learns the latent representation as a concatenation of four chunks where each chunk exposes one object from the input and represents a disentangled factor of variation, respectively. While Hu *et al.* [27] consider an *image attribute* as a factor of variation in single-object input data, our model views an *image object* as a factor of variation in multi-object input data.[1] The latent representation can also be considered as a *compositional representation* [79] consisting of components that can be recombined. Our model architecture is designed to encourage such representations. See Section 3.2 for more details.

In theory, given images with compatible pairs of objects and under the assumption that our system produces realistic mixed scene outputs, the model should learn robust object representations where the description of an object and ideally its attributes such as pose and scale are clearly disentangled (i.e. separated). As mentioned earlier, however, the notion of object attributes is not given explicit structural attention by the architecture of our system. The object itself is exposed in the form of a representation chunk. Therefore, a representation chunk will likely contain the object description and its attributes in an entangled form.

To increase the realism of the generated scenes the model uses adversarial training. Here we make the assumption that the generative network we employ is capable of learning the input data distribution of the dataset in place. Otherwise our method will not be able to approximate the data distribution with the mixed scenes. The proposed method is a generative model that is conditioned on the input dataset. We describe the model and its architecture next.

3.2 Model architecture

In this section, we formally present our method. A fundamental assumption is that a real world image I_{ref} contains four objects, ideally one in each quadrant (see Figure 3.2). The method considers I_{ref} to consist of quadrants $I_{ref}^{q1}, I_{ref}^{q2}, I_{ref}^{q3}, I_{ref}^{q4}$

[1] The disentangling of object attributes within disentangled objects would provide an interesting avenue for future work.

where each quadrant contains either one object or none. In addition to I_{ref}, the method receives four other "quadrant replacement" images $I_{q1}, I_{q2}, I_{q3}, I_{q4}$ where each represents a candidate for the replacement of the respective quadrant in I_{ref} (see Section 3.4 for more on "quadrant replacement" images). A replacement of one or more quadrants in an image is considered a mixing of two scenes with the result of a new scene. Note that the replacement of quadrants and mixing of scenes, respectively, exclusively takes place in the latent space of the model as is described later. These per-quadrant replacement images have been preselected in the algorithm outlined in Section 3.4 and underlie the same assumptions as I_{ref}. In total the model receives five input images simultaneously.

The model architecture is depicted in Figure 3.4. Its main components are:

- a GAN comprised of a generator G and a discriminator D for adversarial training
- an autoencoder including encoder Enc and decoder Dec for image generation
- a classifier Cls to encourage disentanglement within the representation of an image
- a latent space scene mixing algorithm (see Section 3.5)

The GAN and the adversarial training, respectively, is used to generate realistic images representing the mixed scenes. Note that the generator G is synonymous for the entire autoencoder. The autoencoder functions as the generator in the GAN framework. That means in training when G is updated during backpropagation, what actually gets updated is the encoder and the decoder. The GAN generator therefore is used twofold: Firstly, using Enc it encodes the input image into a disentangled representation that is then mixed with other representations according to the scene mixing algorithm. Secondly, given the mixed representation, the generator, using Dec, generates a new image I_{mix}. Ideally, I_{mix} represents a valid image according to the input data distribution. It is then evaluated by the discriminator as real or fake. Additionally, the classifier receives I_{mix} to decide for each quadrant whether its content originates from a replacement image or not and thereby provides a feedback to the autoencoder to what degree its quadrant disentanglement encoding and decoding is consistent with the output. Only if the encoder disentangles the quadrants in the latent space well can the decoder generate an image that represents a semantically correct scene where the respective quadrants can be reidentified by the classifier at the same location as in the original position.

The generator is composed of the encoder and the decoder. The encoder Enc maps an image A_{ref} into a latent space representation

$$Enc(A_{ref}) = [a_1, a_2, a_3, a_4] = f_{A_{ref}} \tag{3.1}$$

where $f_{A_{ref}}$ denotes the representation of image A_{ref} which is a concatenation of a fixed number of representation chunks a_i. See Figure 3.3 for a schematic representation. Each chunk represents the respective image quadrant q_i in the latent space. The variable a_i is meant to exclusively contain a lower dimensional representation of the object in q_i such that this chunk is invariant to modifications in the neighboring chunks. Hence a_i represents a disentangled factor of variation of the input image. At the same time it can be viewed as the DNA part of one quadrant of the output image.

Figure 3.3: The encoder maps image A_{ref} to representation $f_{A_{ref}}$. Chunk a_i represents the quadrant q_i in latent space.

By swapping some of the representation chunks we obtain a new, mixed representation $f_{I_{mix}} = [a_1, b_2, c_3, d_4]$ where a_1, b_2, c_3, d_4 represent quadrants from possibly different input images. Using decoder Dec, said representation is projected back into image space as

$$Dec(f_{I_{mix}}) = I_{mix}, \tag{3.2}$$

where I_{mix} represents a new, realistic scene containing a mixture of objects from one up to four images. See Section 3.5 for more details on how the mixing takes place.

The decoder is not only used to generate new images I_{mix}. It is also used to reconstruct images $\hat{A}_{ref} = Dec(Enc(A_{ref}))$ to facilitate a reconstruction constraint on the autoencoder to the help guide the learning process of the generative model (see Figure 3.4).

The reconstruction constraint on the autoencoder is twofold. The first constraint is based on $\hat{A}_{ref}$ just introduced. For the second, I_{mix} is encoded again as $Enc(I_{mix}) = \hat{f}_{I_{mix}}$ to obtain a reconstruction of its original representation, $f_{I_{mix}}$. The original representation of A_{ref}, $f_{A_{ref}}$, is then reconstructed as $\hat{f}_{A_{ref}}$ using as many chunks from $\hat{f}_{I_{mix}}$ as possible. In fact, as many as were used from $f_{A_{ref}}$ in the mixing algorithm to create $f_{I_{mix}}$. The remaining chunks are reused directly from $f_{A_{ref}}$. Finally, using the decoder, $\hat{f}_{A_{ref}}$ is decoded for a second image reconstruction $A'_{ref} = Dec(\hat{f}_{A_{ref}})$ on which the second reconstruction constraint is

imposed. Both constraints contribute an equally weighted training signal to the autoencoder.

To establish the adversarial training, I_{mix} and the original image A_{ref} are passed to the discriminator D. D decides on the realism of the fake image I_{mix}, thereby issuing a training signal to the autoencoder for improvement (generator loss). Additionally, D decides on the realism of the real image A and the fakeness of the fake image I_{mix} together, thereby issuing a training signal to itself (discriminator loss).

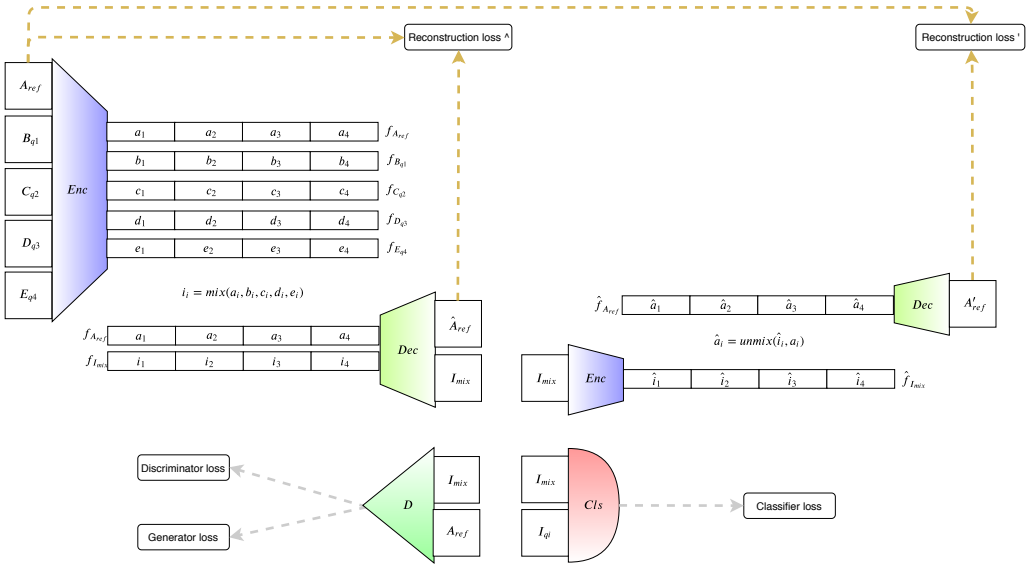


Figure 3.4: Schematic of our model architecture. The architecture includes the encoder Enc, the decoder Dec, the classifier Cls and the discriminator D. Same color denotes same network weights. Best viewed on screen with zoom.

Figure 3.4 shows the sequence of two mixing autoencoders ($Enc - Dec - Enc - Dec$) that helps to encourage disentanglement of image objects in latent space and data space alike. In latent space when the model encodes each image quadrant into a representation chunk, each chunk should be invariant to modifications in the other chunks. And in data space when the model decodes the representation into an image, each quadrant should represent the object its corresponding representation chunk contained. Intuitively, if we decode and re-encode the mixed representation $f_{I_{mix}}$, the resulting vector $\hat{f}_{I_{mix}}$ should preserve the original chunks

copied into it and keep the same order of chunks as in $f_{I_{mix}}$.

To further foster the disentanglement of factors of variation, a classification challenge is posed to a classifier Cls that receives a pair of images including the mix image I_{mix} along with one of the original "quadrant replacement" images I_{qi} with $i \in \{1, 2, 3, 4\}$. For each I_{qi}, the classifier has to decide if that image occurs in I_{mix} and if so in which quadrant. Therefore, if the autoencoder does a thorough job in encoding (and disentangling, respectively) the objects from the various input images and equally in decoding the mixture of the chunks to a new image, the classifier should be able to recognize the original objects in the generated, mixed scene. The classifier is depicted in more detail in Figure 3.5. Note that the chal-

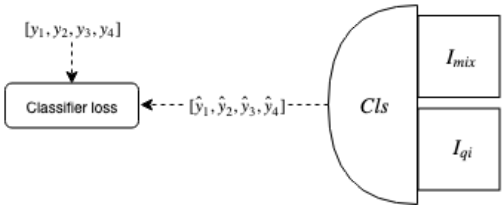


Figure 3.5: Cls decides if an image I_{q_i} occurs in I_{mix} and in which quadrant, where $y_i \in \{0, 1\}$ and $\hat{y}_i \in [0, 1]$.

lenge for the classifier is twofold. It not only has to decide if an image appears in I_{mix} at all but also in which quadrant. The classifier thus consists of four binary classifiers. It outputs four variables $[\hat{y}_1, \hat{y}_2, \hat{y}_3, \hat{y}_4] = \hat{y}$, one for each quadrant of I_{mix}, that decide if image I_{q_i} occurs in quadrant q_i. The output along with the ground truth $[y_1, y_2, y_3, y_4] = y$, i.e. the actual occurrence of the "quadrant replacement" images in I_{mix}, is then used to define the classifier loss.

In addition, the classifier also helps to avoid the shortcut problem [77] where one or more chunks of the representation are ignored by the model. The autoencoder cannot simply choose to ignore a representation chunk, say i_2 in Figure 3.4, since the classifier will not be able to locate the corresponding quadrant of image I_{q_2} in I_{mix} if i_2 was left unused by the autoencoder.

3.3 Loss functions

Our model consists of three loss functions that are minimized jointly during training. The parameters of the loss functions indicate which network weights are updated and trained by which loss function.

3.3.1 Generator loss

The encoder and decoder together as the generator G receive a composite loss that is a weighted sum of three losses: the reconstruction loss, the adversarial loss and the classifier loss.

Reconstruction loss. The reconstruction loss imposes a constraint on the autoencoder that steers it to produce images that closely match the originals $\hat{I}_{ref} \approx I_{ref}$ and is given by

$$\mathcal{L}_{rec}(\theta_{Enc}, \theta_{Dec}) = \mathbb{E}_{I_{ref} \sim p_{data}} \big[\|I_{ref} - \hat{I}_{ref}\|_p + \|I_{ref} - I'_{ref}\|_p \big], \tag{3.3}$$

where θ_{Enc}, θ_{Dec} are the network weights for the encoder and decoder, respectively, and $p = 1$ or $p = 2$. The well known deficiency of l_1 or l_2 as a loss function is blurry predictions that D can easily discriminate.

Adversarial loss. As in [27] and [56], to counteract this flaw we use an adversarial loss to guide G towards sharp and realistic predictions defined by

$$\mathcal{L}^G_{adv}(\theta_{Enc}, \theta_{Dec}) = \mathbb{E}_{I_j \sim p_{data}} \big[\mathcal{L}_{bce}(D(G(I_{ref}, I_{q1}, I_{q2}, I_{q3}, I_{q4})), 1) \big], \tag{3.4}$$

where $j \in \{ref, q1, q2, q3, q4\}$. $\mathcal{L}^G_{adv}$ is equivalent to the non-saturating GAN loss in Equation 2.6. Note that the adversarial loss is computed using the discriminator D. The generator's objective is to fool D into thinking that the fake images G generated are real.

Classifier loss. Thirdly, to encourage disentanglement in the latent space representation of the image objects, we impose the classifier constraint defined by

$$\mathcal{L}_{Cls}(\theta_{Cls}) = \mathbb{E}_{I_j \sim p_{data}} \big[\sum_j \lambda_j \mathcal{L}_{bce}(\hat{y}^j, y^j) \big], \tag{3.5}$$

where θ_{Cls} are the network weights for the classifier and $j \in \{q1, q2, q3, q4\}$, $\hat{y}^j$ the classifier prediction for I_j, $Cls(I_{mix}, I_j) = [\hat{y}^j_1, \hat{y}^j_2, \hat{y}^j_3, \hat{y}^j_4]$, and y^j the ground truth. λ_j is a coefficient but we use $\lambda_j = 1$ for $j \in \{q1, q2, q3, q4\}$. Cls thus consists of four binary classifiers, one for each quadrant of I_{mix}. Each classifier decides if a quadrant in I_{mix} was generated using the quadrant from the replacement image I_j or from I_{ref}. The main challenge for Cls is it does not know which image I_j it is given besides I_{mix} and therefore cannot simply remember what to predict. This loss can be minimized only if the disentanglement of the quadrants and image objects, respectively, as well as the decoding thereof works well.

Joint loss. To combine the three losses we introduce coefficients λ_{rec}, λ_{adv} and λ_{Cls}. By means of these we can optimize the tradeoff between original image reconstruction (3.3), sharpness and realism (3.4) and object disentanglement (3.5). Thus we get the joint loss on G as follows:

$$\mathcal{L}_G(\theta_{Enc}, \theta_{Dec}, \theta_{Cls}) = \lambda_{rec}\mathcal{L}_{rec} + \lambda_{adv}\mathcal{L}^G_{adv} + \lambda_{Cls}\mathcal{L}_{Cls} \tag{3.6}$$

3.3.2 Discriminator loss

The discriminator D with θ_{Dsc} as the network weights is trained with the standard GAN discriminator loss defined by

$$\mathcal{L}^D_{real}(\theta_{Dsc}) = \mathbb{E}_{I_{ref} \sim p_{data}} \big[\mathcal{L}_{bce}(D(I_{ref}), 1)\big] \tag{3.7}$$

$$\mathcal{L}^D_{fake}(\theta_{Dsc}) = \mathbb{E}_{I_j \sim p_{data}} \big[\mathcal{L}_{bce}(D(G(I_{ref}, I_{q1}, I_{q2}, I_{q3}, I_{q4})), 0)\big] \tag{3.8}$$

$$\mathcal{L}_D(\theta_{Dsc}) = \mathcal{L}^D_{real} + \mathcal{L}^D_{fake} \tag{3.9}$$

The discriminator's objective is to classify real images as real and images generated by G as fake, both as resolutely as possible.

3.3.3 Classifier loss

The classifier loss as defined in Equation 3.5 is used to train the classifier separately with weights fixed for both generator and discriminator. Moreover, as described above, the classifier loss contributes to the generator loss.

3.4 Finding visually similar images

We loosely define a "quadrant replacement" image as an image that has a quadrant which is very similar visually, as perceived by the human eye, to the same quadrant of another image. For instance, this could be a blue sky in the top right quadrant, or a baseball player in the bottom left quadrant. For our model the two quadrants can appear very similar semantically but should not be the same pixel-wise. Intuitively, we want to replace a quadrant with a pixel-wise different but semantically similar quadrant to give the model a challenge in generating a new image with different content. Yet at the same time this task should not be easy, thus the two quadrants must not be too close visually (i.e. on the pixel level).

In unsupervised learning there is no human intervention. There are no human annotated labels. Likewise we cannot rely on human labor to find and group similar images. The question arises how we can provide the model with meaningful "quadrant replacement" images since annotations in the dataset are inexistent and random selection works poorly. We therefore have to rely on another mechanism to help us detect visually similar images that satisfy the aforementioned requirements of a "quadrant replacement" image. To this end, we devise a novel visual similarity detection algorithm that leverages the clustering method DeepCluster [6].

Visual similarity detection algorithm As noted in Section 2.2.6, we exploit the fact that images whose representations are close in latent space will also look

similar in image space. The closer two features are in latent space the more visually similar two images will be in data space. "Visually similar" means both images contain an object of the same type such as a person, plane, dog, etc., yet the objects may differ substantially at the pixel level.

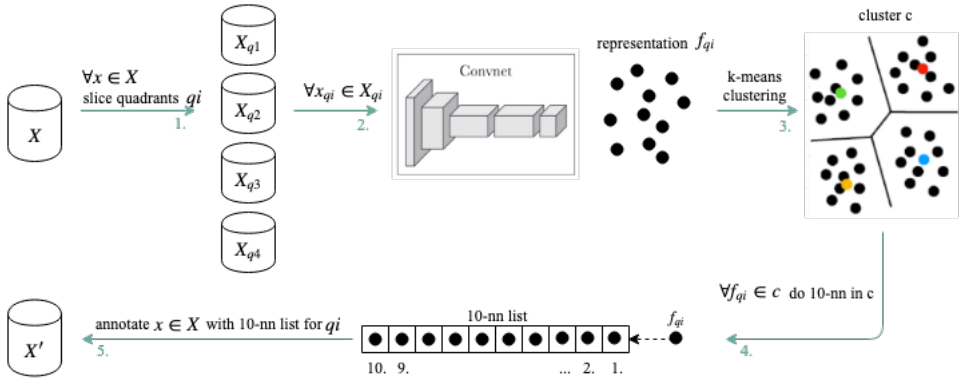


Figure 3.6: Schematic of the visual similarity detection algorithm.

Consider Figure 3.6. Given a dataset $X = \{x_1, x_2, \ldots, x_N\}$ of N images, the algorithm first slices every image from X into quadrants (1.) and creates four extra datasets $X_{q1}, X_{q2}, X_{q3}, X_{q4}$. Subsequently each dataset X_{qi} is processed separately. To produce the representations for all images from X_{qi}, the algorithm uses a convnet (2.) pretrained with DeepCluster [6]. The representations are then clustered using the *k*-means algorithm (3.) to obtain groups of representations. For each representation of a cluster, the ten nearest neighbors within that cluster are calculated (4.) using the *k*-nn algorithm with the $L2$ metric. Finally, each image in X is annotated (5.) with four lists containing the ids of the ten nearest images, one list per quadrant. A pseudocode for the algorithm is given in Appendix A.

In Figure 3.7 we show sample results as determined by the algorithm on the MS COCO [50] dataset. Note that the *visual similarity detection algorithm* is used only to preprocess and annotate the dataset, respectively, prior to the model training. It is not employed during training or test of the model. The list of the ten nearest image ids per quadrant is later utilized during training by the *latent space scene mixing algorithm* described next.

3.5 Latent space scene mixing algorithm

As a result of the clustering algorithm in Section 3.4, each I_{ref} from the dataset has four ordered lists of ten nearest neighbors (10NN-list), one for each quadrant I_{ref}^{qi} with $i \in \{1, 2, 3, 4\}$. The 10NN-list contains the ten images whose corresponding quadrant is closest to I_{ref}^{qi} in latent space. The *latent space scene mixing*

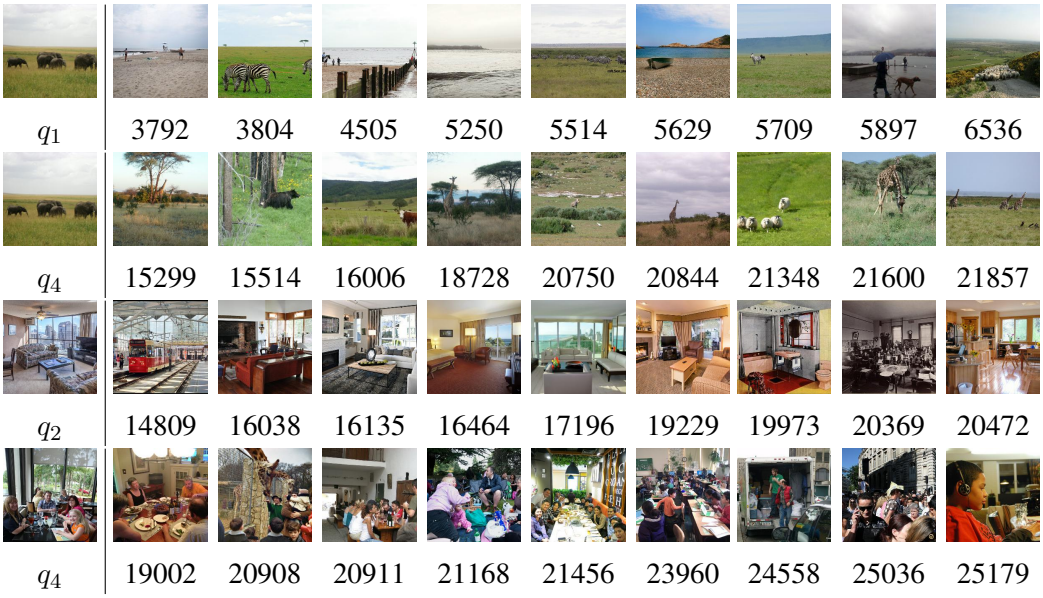


Figure 3.7: Sample results of the visual similarity detection algorithm. The leftmost column shows the reference image. The label q_i denotes the quadrant for which the 9 nearest neighbors are shown on the right. The numbers denote the L2 distance between the reference image quadrant q_i and the respective quadrant of an image on the right. All images belong to the MS COCO training dataset. Best viewed on screen with zoom.

algorithm described here takes place in the latent space of the autoencoder where the representations of the input images are available. This process is summarized in Algorithm 1.

The algorithm ensures the resulting mixed representation $f_{I_{mix}}$ has the following properties:

- at least one quadrant remains from I_{ref} (lines 6-7)
- at least one quadrant of I_{ref} is replaced (lines 9-10)
- only replacements which are "sufficiently similar" to the original occur (lines 12, 15)

Therefore, the number of replaced quadrants and representations, respectively, in $f_{I_{mix}}$ is always between one and three. In other words, the mixed scene will always be made up of between two and four images. See Figure 3.8 for visual depictions of mixed representations. Quadrants are replaced only at the same position in the image, i.e. a replacement involving I_{ref}^{qi} and I_{qj} where $i \neq j$ does not occur. Due

Input: a reference image I_{ref}, its representation $f_{I_{ref}}$, a threshold τ
Output: A mixed representation $f_{I_{mix}}$ from two to four images

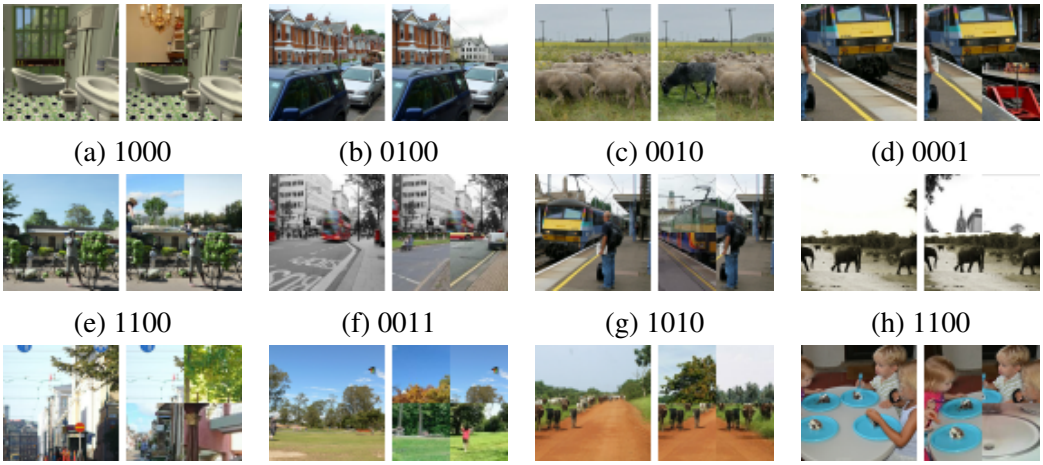
begin
  foreach *quadrant* I_{ref}^{qi} *of* I_{ref} **do**
    $I_{qi} \leftarrow$ choose replacement candidate uniformly from 10NN-list of I_{ref}^{qi}
  end
  foreach *quadrant* I_{ref}^{qi} *of* I_{ref} **do**
    if I_{qi} *has the greatest* $L2$ *distance among all replacement candidates* **then**
      select representation of quadrant I_{ref}^{qi} and ignore candidate I_{qi}
    end
    if I_{qi} *has the lowest* $L2$ *distance among all replacement candidates* **then**
      replace representation of quadrant I_{ref}^{qi} with representation of I_{qi}
    end
    if $L2$ *distance of* I_{qi} *is greater than* τ **then**
      select representation of quadrant I_{ref}^{qi} and ignore I_{qi}
    else
      replace representation of quadrant I_{ref}^{qi} with representation of I_{qi}
    end
  end
end

Algorithm 1: Latent space scene mixing algorithm

to the limitations of the algorithm there are also mixed scenes that do not match well semantically unlike the examples shown.

3.6 Implementation

As in [27] and [8], we use a GAN architecture derived from DCGAN [66]. This means both the generator and the discriminator use convolutional layers to process their input. The generator which is implemented as an autoencoder uses a handcrafted architecture as depicted in Table 3.1 and Table 3.3. For the reconstruction loss we choose the l_1-norm over l_2 as encouraged by [33]. We do not use dropout in any of the networks. The weights for the joint loss are $\lambda_{rec} = 0.904$, $\lambda_{adv} = 0.052$ and $\lambda_{Cls} = 0.044$. The system processes batches of images of size 64×64 and is trained on a dataset of 338,865 color images. The batch size is 12.



(a) 1000 (b) 0100 (c) 0010 (d) 0001

(e) 1100 (f) 0011 (g) 1010 (h) 1100

(i) 0111 (j) 1011 (k) 1101 (l) 1011

Figure 3.8: Examples of mixed feature vectors $f_{I_{mix}}$ in data space. In each example, left is I_{ref}, right is the image equivalent of the mixed feature vector $f_{I_{mix}}$. The binary mask denotes clockwise (0) original quadrant and (1) replacement quadrant, starting from the top left corner of the image. First row: 1 replaced quadrant, second row: 2 replaced quadrants, third row: 3 replaced quadrants.

For each iteration and batch, respectively, the generator is optimized twice in a row and the discriminator and the classifier once. The model is trained using the non-saturating GAN loss (see Section 2.4.1) where both the generator loss and the discriminator loss are minimized using the Adam [39] optimizer. The learning rate for the generator is 0.0002 and for the discriminator 0.0004 [23]. The training is performed on a Nvidia GeForce GTX TITAN X with 12 GB memory. The training time for 50 epochs is approximately 12 days. One reason for the extensive duration is that for each image in the batch the network has to load an additional four images (i.e. the quadrant replacement candidates) directly from the file system in a directory of 338,865 files, causing a lot of I/O processing. An alternative approach to this would be to store all the 40 quadrant replacement candidates together with the reference image in the tfrecord file and thereby to circumvent the slow I/O access into a large directory altogether. However, it is not clear whether this idea would indeed lead to faster processing as for each image in the batch an additional 40 images would be loaded eagerly but only four of them used. As for another but less significant reason, we conjecture the deep autoencoder with 89 convolutional layers consumes another non-negligible amount of time to perform backpropagation even though the number of parameters is only a fraction of the total network. Also, the use of resize convolution in the decoder is known to be slower than transposed convolution. But more significantly, an entire AlexNet is

Conv Layers	Building Block	# Feature Maps	Output Size
-	Input	-	$64 \times 64 \times 3$
-	CoordConv	-	$64 \times 64 \times 3$
1	3×3 conv, $stride = 1$	48	$64 \times 64 \times 48$
2-5	DB-E $\times$ 4	112	$64 \times 64 \times 112$
6	TD	112	$32 \times 32 \times 112$
7-11	DB-E $\times$ 5	192	$32 \times 32 \times 192$
12	TD	192	$16 \times 16 \times 192$
13-19	DB-E $\times$ 7	304	$16 \times 16 \times 304$
20	TD	304	$8 \times 8 \times 304$
21-30	DB-E $\times$ 10	464	$8 \times 8 \times 464$
31	TD	464	$4 \times 4 \times 464$
32-43	DB-E $\times$ 12	656	$4 \times 4 \times 656$
44	TD	656	$2 \times 2 \times 656$
45	1×1 conv, $stride = 1$	192	$2 \times 2 \times 192$
-	Reshape	-	1×768

Table 3.1: Encoder architecture adapted from *FC-DenseNet103* with 5 pools of dense blocks and growth rate $k = 16$. Image size is 64×64 pixels. See Table 3.2 for a definition of the building blocks.

Building Block	Definition
Dense Block Encoder (DB-E)	instance norm ReLU 3×3 conv, $stride = 1$
Transition Down (TD)	instance norm ReLU 1×1 conv, $stride = 1$ 2×2 max pool, $stride = 2$

Table 3.2: Building blocks in the encoder architecture.

used for the classifier. On top of it all, the generator (i.e. the autoencoder) is updated twice in each training iteration. Due to time constraints we have not profiled the network during training to find out more about the actual cause. A possible remedy could be the use of the $/dev/shm$ facility on Linux systems which is a directory that is stored in memory and therefore enables much faster I/O access. We tried to explore that option but its size at the time could not contain the entire dataset.

The encoder as shown in Table 3.1 uses dense blocks [28] and is inspired by the *FC-DenseNet103* model from Jégou *et al.* [37]. Using dense blocks solely yields a deep architecture that consists of convolutional and pooling layers without any fully connected layers. This saves parameters and memory. More importantly, we argue that this is beneficial for the disentanglement of objects as the spatial information is preserved by the convolutional as well as the pooling layers. A fully connected layer at the end would squash the spatial information of different objects and likely discourage disentanglement. To further encourage the encoder to disentangle objects along with spatial information, we prepend a CoordConv layer [53] which simply adds two channels along the depth dimension of the incoming tensor. The two channels represent hard-coded (x, y) Cartesian space coordinates. The CoordConv layer gives the subsequent convolutional layers a chance to know where they are in Cartesian space and may contribute to the conservation of spatial location information up to the encoded object representation.

Note in Table 3.1 the drastic increase in feature maps as the network becomes deeper. As argued in [28], the feature maps can be seen as the global state of the network. Each dense block layer contributes new state and the number of filters thus increases. The rate of increase is defined by the growth rate k. At the final layer, the encoder outputs the latent representation that consists of four representation chunks of size 192, each representing a distinct image quadrant and ideally an image object, respectively.

Our encoder has a receptive field of 660×660 pixels.[2] We also try with an encoder that has a small receptive field of 46 pixels with the intuition that in order to build good disentangled object representations the last layer of the encoder, that is the neurons in the last layer, should roughly only see one quarter of the input image, equivalent to one quadrant with one object as opposed to the entire image with four objects. However, the experiments conducted do not yield better visual results and we dismiss the idea (see Appendix D for other unsuccessful experiments).

The decoder architecture is shown in Table 3.3. The design rationale for the decoder is to be approximately symmetric to the encoder yet to give it more capacity to support the generation of a mixed scene image as this is likely the more difficult task than to just faithfully reconstruct the input image. Encouraged by evidence in [84], we add spectral normalization in the decoder for each convolutional layer (as opposed to only the discriminator as originally proposed). A peculiarity in the upsampling process is the use of the resize convolution [62] in place of the more frequent transposed convolution. Autoencoder calibration experiments we

[2] See `compute_receptive_field_from_graph_def` from the `tf.contrib` TensorFlow library for how to compute the receptive field.

conduct show that resize convolution in the decoder outperforms transposed convolution in terms of better reconstruction quality and higher peak signal-to-noise ratio (PSNR) values, respectively. See Section 3.6.1 for more details.

Conv Layers	**Building Block**	**# Feature Maps**	**Output Size**
-	Input	-	1×768
-	Reshape	-	$2 \times 2 \times 192$
1	TU	656	$4 \times 4 \times 656$
2-13	DB-D $\times$ 12	(848) 192	$4 \times 4 \times 192$
14	TU	464	$8 \times 8 \times 464$
15-24	DB-D $\times$ 10	(624) 160	$8 \times 8 \times 160$
25	TU	304	$16 \times 16 \times 304$
26-32	DB-D $\times$ 7	(416) 112	$16 \times 16 \times 112$
33	TU	192	$32 \times 32 \times 192$
34-38	DB-D $\times$ 5	(272) 80	$32 \times 32 \times 80$
39	TU	112	$64 \times 64 \times 112$
40-43	DB-D $\times$ 4	(176) 64	$64 \times 64 \times 64$
44	spectral norm 1×1 conv, $stride = 1$	3	$64 \times 64 \times 3$

Table 3.3: Decoder architecture adapted from *FC-DenseNet103* with 5 pools of dense blocks. Note that only the new feature maps resulting from the current dense block are passed to the next layer. See Table 3.4 for a definition of the building blocks.

Building Block	**Definition**
Dense Block Decoder (DB-D)	ReLU spectral norm 3×3 conv, $stride = 1$
Transition Up (TU)	spectral norm 3×3 resize conv, $stride = 2$

Table 3.4: Building blocks in the decoder architecture.

For the classifier, analogous to [27], we use AlexNet with batch normalization after each convolutional layer as described in Section 2.2.4. The two image inputs for the classifier are concatenated along the RGB channels as shown in Figure 3.5.

The discriminator is implemented as a PatchGAN [33] discriminator with a receptive field of 46 pixels. It is defined in Table 3.5. The discriminator does not use CoordConv [53] layers.

Conv Layers	**Building Block**	**# Feature Maps**	**Output Size**
-	Input	-	$64 \times 64 \times 3$
1	4×4 conv, $stride = 2$ LReLU	42	$32 \times 32 \times 42$
2	spectral norm 4×4 conv, $stride = 2$ LReLU	84	$16 \times 16 \times 84$
3	spectral norm 4×4 conv, $stride = 2$ LReLU	168	$8 \times 8 \times 168$
4	spectral norm 4×4 conv, $stride = 2$ LReLU	336	$4 \times 4 \times 336$
5	1×1 conv, $stride = 1$	1	$4 \times 4 \times 1$

Table 3.5: Our PatchGAN discriminator architecture with a receptive field of 46 pixels.

For the latent space scene mixing algorithm we use a threshold of $\tau = 16,000$ to allow for enough variation with respect to the quadrant replacement images.

In Table 3.6 we show the number of parameters of each network of our model. The generator is given more capacity than the discriminator for the demanding task of understanding and generating realistic images compared to the discriminator deciding on a yes/no question.

Regarding the visual similarity detection algorithm we use the k-means and k-nn implementations provided by Johnson *et al.* [36] and a pretrained AlexNet provided by [6] as the convnet to compute the representations.

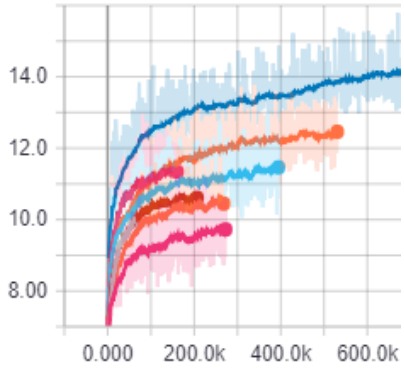
3.6.1 Autoencoder calibration

As an intermediate step in the design of the neural architecture for our method, we conduct more than 20 experiments to calibrate the autoencoder so as to improve faithfulness and realism of the reconstruction of the input image. To that

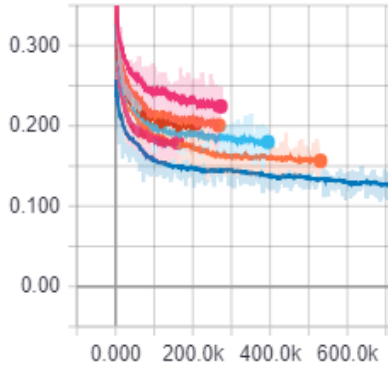
Network	Parameters
Discriminator	1,188,391
Generator	8,100,851
Encoder	2,800,512
Decoder	5,300,339
Classifier	45,355,044
Total	**54,644,286**

Table 3.6: Number of parameters of each network in our model.

end, we take the autoencoder (i.e. generator) and the discriminator without the classifier and train it on the $L1$ reconstruction and the GAN loss with varying neural architectures and hyperparameters. As shown in Figure 3.9, the autoencoder architecture with resize convolution and spectral normalization in the decoder and a latent representation size of 768 (cf. blue curve) outperforms all the other experiments that use transposed convolution among other parameters.



(a) PSNR (higher is better)



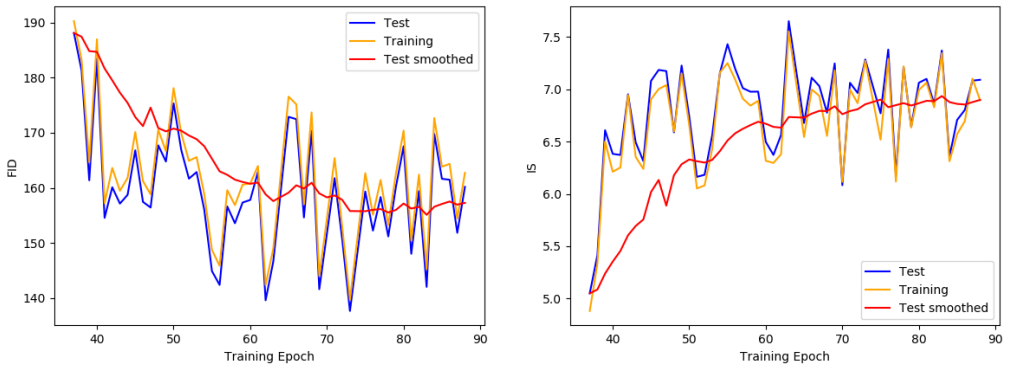
(b) $L1$ reconstruction loss (lower is better)

Figure 3.9: PSNR and $L1$ reconstruction loss vs. training step. The autoencoder with resize convolution and spectral normalization in the decoder (blue curve) achieves the best values. The other colors represent different experiments.

3.6.2 FID and IS training and test curves

In Figure 3.10 we plot the FID [23] and IS [72] curves on the training and the test set as a function of the training epoch. After every training epoch, similar to [38], we calculate the FIDs using 10,492 images drawn randomly from the test set and report the distance at every training epoch. Likewise, we use 10,482 images drawn randomly from the training set and report the distance at every training

epoch (see Table 4.1 regarding the datasets). As can be seen in the chart, the FID



(a) FID of generated images

(b) IS of generated images

Figure 3.10: FID and IS on COCO for our model as a function of the training epoch. FID: lower is better. IS: higher is better.

values are still comparably high (lower is better) with respect to the achievements of other models which lie in the range of 5-250. Likewise with the IS metric. We notice, however, that the FID and IS values decrease and increase, respectively, with the duration of the training which is a good sign that the model is learning. To perform the experiments we choose the model with the lowest FID.

Chapter 4

Experiments

In this section, we run experiments to assess the performance of our method in itself and with respect to other methods. We do this in three directions: Firstly, using our model in inference mode, we visually observe the quality of the generated images. Secondly, by means of various transfer learning experiments we quantitatively evaluate our model with respect to others. Thirdly, an ablation analysis demonstrates how the proposed full model performs. Next we describe the datasets used in the experiments.

4.1 Datasets

For the training and evaluation of our model we use various natural image datasets.

4.1.1 MS COCO

The Microsoft Common Objects in Context (COCO) dataset [50] is a large, natural multi-object image dataset for classification, object detection, segmentation, and captioning. As the name suggests, COCO images are taken from everyday scenes that add the "context" to the objects captured. COCO provides about 330,000 images in total of which more than 220,000 are labeled. On average, COCO contains 3.5 categories and 7.7 instances per image. In other words, most often an image contains more than three categories and more than seven objects (instances). Unlike the ImageNet dataset that contains one object per image, COCO is thus a reasonable choice for training of our method which assumes an image contains four objects, one per quadrant. Another interesting observation is only 10% of the images in COCO have only one category per image. This means only few images have all objects of the same type and therefore the images are

highly diverse, which makes for a complex, natural dataset and a difficult learning challenge for our method. We use the 2017 update of COCO which comes with 118,288 labeled training images, 5001 validation images and 40,670 test images. As our method is unsupervised, we do not use any labels. However, to ramp up the initial training experiments, we utilize the bounding box annotation to create datasets where each image has at least four bounding boxes (i.e. four objects). We thus create different datasets out of the available COCO data. The versions are listed in Table 4.1. We use data augmentation to increase the dataset size and support the learning process, respectively.

Source dataset	**Purpose**	**Id**	**Image constraints**	**Data augmentation**	**Size**
training	training	tr_v4	have at least four bounding boxes, size at least 200 pixels	horizontal flip	135,554
training	training	tr_v5	have at least four bounding boxes, size at least 200 pixels	horizontal flip, random crops with scaling	338,865
training	FID/IS	tr_v6	size at least 200 pixels	-	10,482
test	FID/IS, testing	te_v2	size at least 200 pixels	-	10,492

Table 4.1: COCO training and test datasets used for LORBMS.

Notice the creation of a dataset involves the algorithm described in Section 3.4 and listed in Algorithm 2. As indicated in Table 4.1 for dataset tr_v5, we augment each original image in four ways using horizontal flip and random crops with different scales as depicted in Figure 4.1. To some extent, these augmentations will increase the size of objects and move some objects entirely into one quadrant as desired by the method. However, objects spanning more than one quadrant can result as well. These augmentations essentially take an object and produce several versions of it. Intuitively, this should give the model a better chance to learn meaningful features as the same object is encountered multiple times during one training epoch and will hopefully encourage the learning process.

Figure 4.1: Augmentations on each original image including horizontal flip and random crops with scaling for training set tr_v5.

4.1.2 STL-10

The STL-10 [11] dataset is a single-object image recognition dataset for single-label classification. It is intended also for unsupervised representation learning and provides a set of 100,000 unlabeled images. We use it only for transfer learning experiments in a supervised setting. To that end, STL-10 provides 5000 labeled training images with 500 images per class and 8000 test images with 800 images per class. The ten classes are airplane, bird, car, cat, deer, dog, horse, monkey, ship, truck. Each image contains one object only which is roughly center aligned. The images are of size 96×96 pixels. STL-10 is similar to the CIFAR-10 dataset. However, CIFAR-10 comes with image size 32×32 only which is the main reason we use STL-10. STL-10 is a complex dataset with both high inter-class and intra-class diversity.

4.1.3 PASCAL VOC2012

The PASCAL visual object classes (VOC) dataset [17] is a realistic scenes dataset for multi-label classification, detection, and segmentation. The 2012 version comes with 11,540 labeled training images with 20 object classes. Unlike STL-10, PASCAL VOC images can have multiple classes per image and are thus suitable for multi-label classification tasks. The 20 classes are person, bird, cat, cow, dog, horse, sheep, aeroplane, bicycle, boat, bus, car, motorbike, train, bottle, chair, dining table, potted plant, sofa, tv/monitor. The images are of varying size, mostly larger than 200 pixels.

4.1.4 ImageNet ILSVRC2012

The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) 2012 dataset [71] is the basis for the ILSVRC, an annual computer vision competition. The dataset contains 1,281,167 labeled training images with 1,000 object categories for single-label image classification, object detection and segmentation. It comes with 50,000 validation images and 100,000 test images.

4.2 Qualitative Evaluation

A reliable method for qualitative evaluation of a generative model is visual inspection of the generated images on unseen data. The two subsequent experiments showcase our model generating mixed scenes.

4.2.1 Mixed scene renderings

Using our method in inference mode, we showcase renderings of mixed representations and scenes along with the corresponding input images in Figure 4.2. One row represents one forward pass through our model and thus one distinct example. The examples do not always meet the *4 object assumption* made in Section 3.1 that the reference image in column I_{ref} consists of up to four objects spread evenly across its quadrants. We can observe how the model copes with this misalignment in column I_{mix}. For more specific object transfer experiments in line with the model assumptions see Section 4.2.2.

To produce these results only the autoencoder of our method, that is the encoder and the decoder as described in Section 3.2, is used. Note that the column $I_{f_{I_{mix}}}$ exists solely for the purpose of visualizing the mixed representation $f_{I_{mix}}$. The renderings in column I_{mix} show that the model has learnt to take the mixed representation and create a scene that clearly correlates with the representation mixing. Moreover, the model fuses the quadrant boundaries as seen in column $I_{f_{I_{mix}}}$ smoothly. This demonstrates that the model adds a degree of realism to the rendering other than a simple reverse encoding of the representation as depicted in column $I_{f_{I_{mix}}}$. Even in case where the mixed representation does not meet the *4 object assumption* well, the model often manages to render a somewhat meaningful output. Yet as we show in Section 4.4, the FID values of these renderings are still relatively high compared to other generative models though many of them use simpler datasets. The realism of the renderings therefore still leaves room for improvement. See Appendix B for more examples.

4.2.2 Object transfer experiments

In this experiment we investigate visually the capability of the system to transfer objects from one scene to another. To that end, we take an image that contains an object and a second image as the background scene we want the object to transfer to. To implement the object transfer we use only a subset of our system including the autoencoder and feed it the two images along with a mask to define in which quadrants the transfer object is located and which quadrants to mix. For instance, as schematized in Figure 4.3, we take an image I_{obj} with a car object located in the bottom left quadrant together with a scene I_{ref} depicting a street and pass the pair along with the mask 0010 to the subsystem. Based on the representation mixing defined by the mask, the subsystem generates a new scene I_{mix} as shown on the right. Note that in this experiment we handpick the two input images from random sources by human judgment only and there is no *visual similarity detection algorithm* in place as described in Section 3.4. Therefore, in these samples we can observe how well our model generalizes to unseen data which

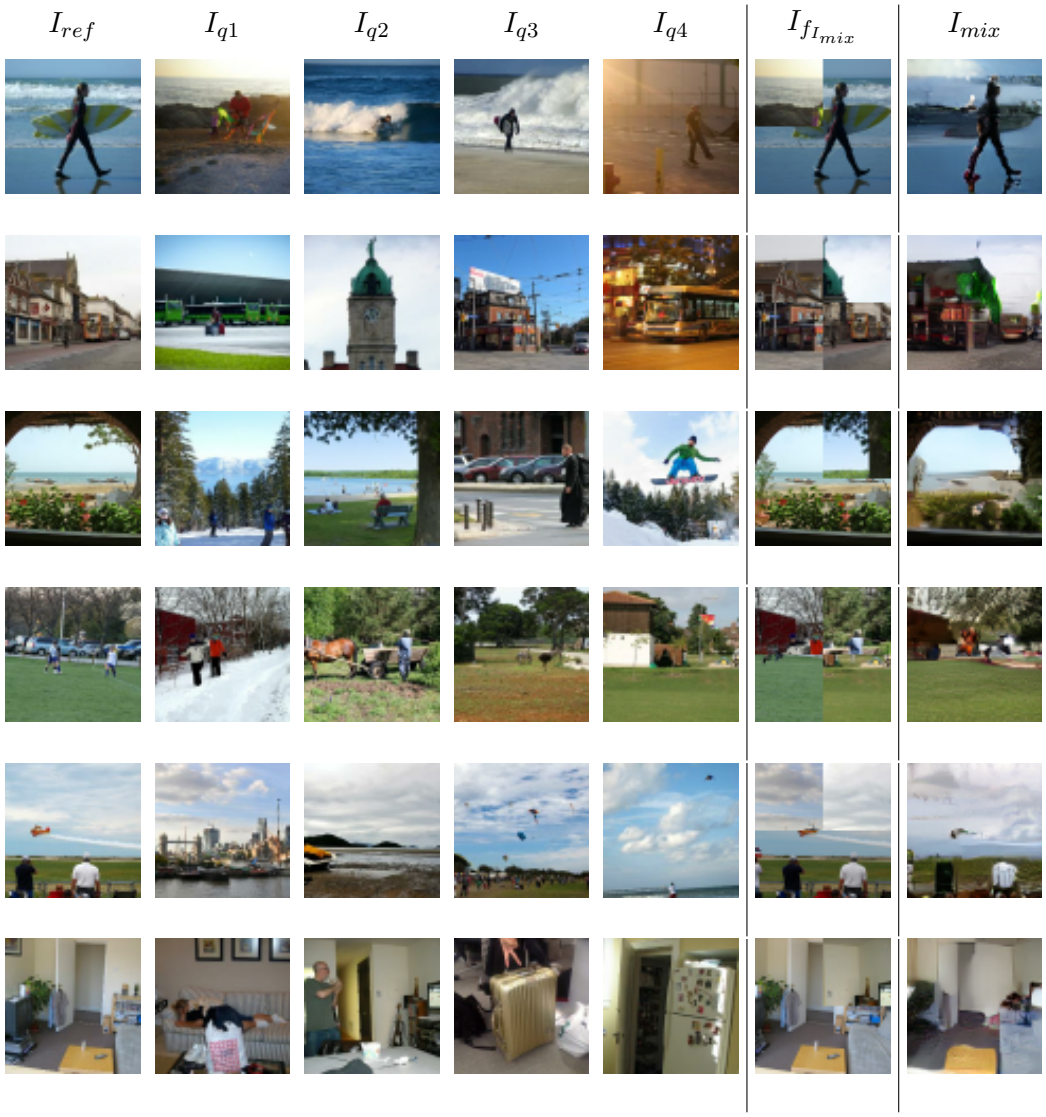


Figure 4.2: Mixed scene renderings shown in the rightmost column produced by our method with the MS COCO dataset at 64×64. Each row represents a forward pass through our model: The columns I_{ref}, I_{q1}, I_{q2}, I_{q3}, I_{q4} represent the input space while $I_{f_{I_{mix}}}$ shows the image equivalent of the mixed feature vector $f_{I_{mix}}$ and finally I_{mix} is the rendered output image.

otherwise might not have been selected by the training process at all.

In Table 4.2 we show object transfer renderings for a variety of handpicked objects and scenes. We observe that transferred objects do not always appear in the right pose. This shows that the system has no notion of object pose thus far. Further, the boundaries of the transferred objects are sometimes distorted when the

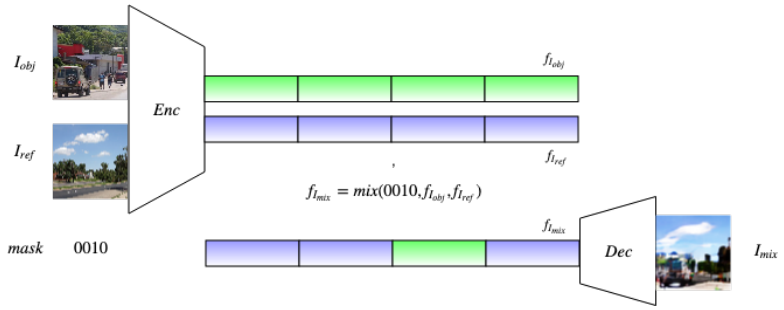


Figure 4.3: Object transfer of a car to a street scene using a subsystem of our method.

object consists of lighter colors than its surroundings. It appears that the system is better at handling objects with darker colors (i.e. keeping its boundaries) and good at distorting lighter colors such as drawing a sky. For more examples see Appendix C.

4.3 Transfer learning

As a form of quantitative evaluation we use the subsequent transfer learning experiments to compare our model with other SOTA works. In these experiments, we first pretrain our model using the proposed unsupervised training. We then transfer the model weights and freeze them before training additional weights on a target dataset to learn a new task. Finally, the evaluation is performed on test images of the target dataset.

4.3.1 Classifying STL-10 using LORBMS as feature extractor

A common method to evaluate the quality of representations[1] learnt in an unsupervised algorithm is described by Coates *et al.* [11] and applied for instance in [66], [5]. The idea is to use the representations as input to a one-layer linear classifier and train it on a supervised learning task. The performance of the model, after fitting it on top of the representations, is then evaluated. The rationale behind this evaluation procedure is that a simple classifier should achieve better performance given representations of good quality and consisting of well disentangled factors of variation. For the experiments with our method, we use either the encoder or the discriminator to map images to their representations. In case of the discriminator, we use the output of the penultimate layer as input to the classifier. The experiments are performed on the STL-10 [11] dataset with a

[1] e.g. the latent space vectors of an autoencoder or the output of a discriminator

Table 4.2: Object transfer renderings. An object spans one or two quadrants.

ten class single-label classification benchmark. To that end, analogous to [35], we follow the testing protocol as suggested by [11]. We train the classifier with the Adam [39] optimizer (default parameters) and a softmax cross entropy loss for 200 epochs on the ten predefined folds using the same data augmentations as in [35] (i.e. randomly resize the images and extract 96×96 crops) and report the average accuracy on the test set. We compare our model with other pretrained models including supervised learning and reference SOTA models. For comparison with the SOTA, note that our model was trained on 64×64 COCO image

patches and then used on STL-10 images at 96×96. This might account for a few percent in reduced performance. All results are presented in Table 4.3.

Model	**Accuracy**	**SD**	**Cls**
Random	10.0%	-	-
Random encoder	43.4%	±0.4	17,290
Random encoder (finetuned)	56.5%	±0.6	17,290
STL-10 encoder	**78.7%**	±0.1	17,290
PASCAL encoder	47.1%	±0.2	17,290
STL-10 AlexNet	60.9%	±0.1	40,970
ImageNet AlexNet	62.4%	±0.3	40,970
Jenni & Favaro [35] (frozen)	**76.9%**	±0.1	40,970
Swersky *et al.* [75]	70.1%	±0.6	-
Ours (discriminator)	**62.8%**	±0.3	120,970
Ours (discriminator finetuned)	62.6%	±0.2	120,970
Ours (encoder)	36.5%	±0.2	17,290
Ours (encoder finetuned)	54.2%	±0.2	17,290
Ours (knowledge transfer [61], discriminator)	**42.9%**	±0.3	40,970
Ours (knowledge transfer [61], encoder)	38.5%	±0.1	40,970

Table 4.3: Comparison of average test-set top-1 accuracy on STL-10 images with different models and other published results. The rightmost column indicates the number of parameters of the corresponding classifier.

For the *Random encoder* model we initialize the encoder with random weights and keep them fixed while training only the classifier. We unfreeze the encoder weights for the *Random encoder (finetuned)* experiment and train the encoder along with the classifier. The *STL-10 encoder* model uses supervised pretraining for both our encoder (initialized randomly) and the classifier on the STL-10 training dataset at 96×96 pixels for 150 epochs. The encoder is then frozen and the classifier retrained as described above. It unsurprisingly achieves better performance than our unsupervised model and the SOTA. For the *PASCAL encoder* pretraining we follow Krähenbühl *et al.* [42] and first train our encoder together with a single-layer classifier using the momentum optimizer with a sigmoid cross entropy loss for 80,000 iterations on the PASCAL VOC 2012 dataset at a resolution of 224×224 pixels. PASCAL VOC offers a 20-way multi-label classification task. The encoder is then frozen and transferred for the evaluation on STL-10. The performance is marginally above the *Random encoder* and suggests that pretraining on PASCAL VOC is not effective for transfer learning on STL-10. The *STL-10 AlexNet* experiment uses the same setting as the STL-10 encoder but with a standard AlexNet (conv1-conv5) as the feature extractor. Surprisingly, it does

not perform as well as our encoder. In the *ImageNet AlexNet* experiment we pretrain an entire AlexNet on ImageNet at 224×224 pixels for 60 epochs with a final ImageNet validation top-1 accuracy of 44.7%. Then we transfer the conv1-conv5 weights to the evaluation on STL-10 and achieve 62.4%. This indicates that pretraining on ImageNet is a better choice than on PASCAL.

Regarding the performance of our model pretrained with LORBMS, the result of 62.8% accuracy with the discriminator gives rise to conclude that our method has indeed learnt to extract features fairly well with respect to the SOTA. In contrast, however, the frozen encoder with 36.5% performs worse than essentially all other experiments. Also, the penultimate layer of the discriminator has a relatively large output of size $6 \times 6 \times 336$ (i.e. 336 activation maps of size 6×6 pixels) which results in a seven times bigger classifier than with the encoder (see rightmost column in Table 4.3). So the good performance of the discriminator could also be attributed to the classifier.

Note that the referenced SOTA have different architectures from ours. For instance, [35] uses an AlexNet (conv1-conv5) for the feature extraction which equals a model of 3.2M parameters. In contrast, our hand-engineered encoder has 2.8M parameters and our PatchGAN discriminator has 1.2M parameters. The numbers are therefore indicative only.

For a fairer comparison with the SOTA, we turn to the self-supervised learning framework by Noroozi *et al.* [61]. It presents a method for knowledge transfer between otherwise incompatible architectures which is the case between our encoder/discriminator and the AlexNet. The method works as depicted in Figure 4.4. Shown in (a) is the unsupervised training of our method LORBMS. In (b) we compute the representations of all images in the COCO dataset using either our encoder or discriminator as feature extractor and cluster them using k-means. As depicted in (c), the resulting clusters give a pseudo-label for each representation and image, respectively. In (d) we then train an AlexNet using the "pseudo-labeled" COCO dataset as single-label classification task. Finally, the trained AlexNet is frozen and the conv1-conv5 weights transferred to the STL-10 evaluation. In the case where our encoder is used for the feature extraction in (b), the result on STL-10 achieves 38.5%, slightly better than the direct counterpart yet significantly lower than the SOTA. Likewise, when we use the discriminator in (b), the result on STL-10 is only 42.9%. Considering these results as a fair comparison, it becomes evident that there is a significant gap

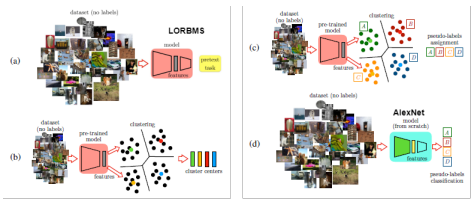


Figure 4.4: Knowledge transfer from the LORBMS model to the AlexNet model (adapted from [61]).

in STL-10 accuracy between our model and the SOTA.

In summary, the empirical evidence of the STL-10 experiment indicates that the representations extracted by our method are of limited quality when used on other input data. This suggests that the encoder in particular generalizes poorly considering the weak performance on the classification task. While the discriminator performs better, the results of our model suggest that what our method has learnt on COCO does not generalize well to extract features on other data. Although intuitively the STL-10 dataset could be viewed as an image subspace of the COCO dataset and thus a better result reasonable.

4.4 FID and IS scores

In Table 4.4 we list the FID and IS scores of our model. The model with the lowest FID of 137.7 occurs after epoch 73 and is the version we use for all experiments. The FID values are relatively high compared to other models that display a range of 5-250 is possible (e.g. [23, 38, 60, 84]).

	Mean	**SD**	**Best**
FID	158.5	$\pm$10.9	137.7
IS	6.9	$\pm$0.5	7.6

Table 4.4: FID and IS statistics for our model over the course of training.

4.5 Ablation analysis

The ablation analysis allows us to validate the model design if the full model indeed outperforms various partial models with one component removed. To the best of our knowledge, there exists yet no metric for the quantitative evaluation of disentangled representations from unsupervised learning when no labels are available. Therefore, to assess our model, we use the STL-10 transfer learning benchmark and the FID/IS metrics.

Essentially, we want to see if all of the components of our model contribute to the best performance. To that end, we consider the composite loss $\mathcal{L}_G$ from Equation 3.6, remove each component individually and train the model with our method. Using the resulting trained encoder for transfer learning, we test its performance on the STL-10 classification task as done in Section 4.3.1. Given the limited time budget to perform the ablation analysis and since the full training of our method is a lengthy process ($\sim$ 12 days), we train each model for ten epochs only ($\sim$ 2.6 days) and report its performance. The training is done with

the same parameters as in Section 3.6. We do without visual results as our final best model occurred only at epoch 73 and focus instead on quantitative metrics. Consequently, we compare the classification performance on STL-10 of each ablation experiment and compare the FID/IS scores as well to reflect on the generative capability of the respective model. The results are shown in Table 4.5 and visualized in Figure 4.5. Note that the achieved values are weak compared to our final model due to the short training duration.

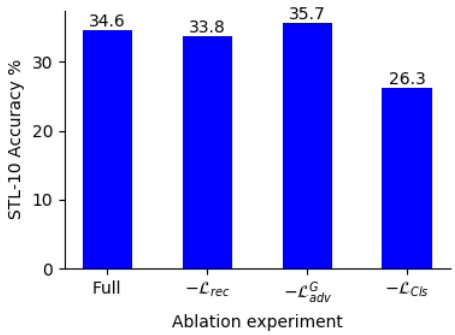
Ablation	**Accuracy**
Full	34.6%
$-\mathcal{L}_{Cls}$	33.8%
$-\mathcal{L}_{rec}$	26.3%
$-\mathcal{L}^{G}_{adv}$	35.7%

(a) STL-10 test set accuracy

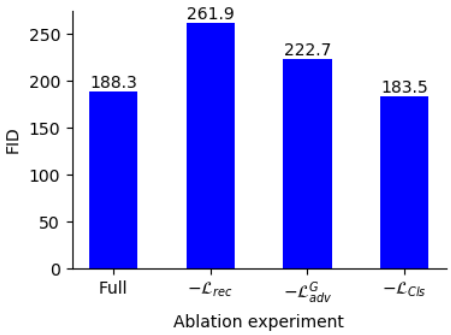
Ablation	**FID**	**IS**
Full	188.3	5.6
$-\mathcal{L}_{Cls}$	183.5	6.0
$-\mathcal{L}_{rec}$	261.9	3.2
$-\mathcal{L}^{G}_{adv}$	222.7	4.2

(b) FID and IS results

Table 4.5: Ablation analysis of our method after ten epochs of training. Each ablation experiment removes one component of the joint loss in Equation 3.6.



(a) STL-10 test set accuracy



(b) FID results (lower is better)

Figure 4.5: Visualization of the results in Table 4.5.

Given the STL-10 test set accuracies, we can see that the removal of the classifier loss or the reconstruction loss causes the accuracy to drop and that these components do have a positive effect on the usefulness of the learnt encoder as a feature extractor. Without the reconstruction loss, the performance even drops drastically. Surprisingly, when the adversarial loss is removed, the accuracy goes up. Without it, the learning task becomes simpler in that the generator has to focus only on image reconstruction and disentangling of the objects and image quadrants, respectively. This seems to positively impact the training of the encoder for transfer learning use, at least at this early stage of training.

We observe a somewhat different picture with the FID and IS results. These metrics are applied directly to the generated images of the various models and thus give a different perspective on the performance. The reported value is the average computed over the last three epochs. Again, overall the scores are high as the training was short. Surprisingly, the experiment without the classifier loss achieves a lower FID score than the full method. We conjecture that this is due to the early training stage of the model and that as the model matures with ongoing training the values might change. This could, however, indicate that the weight for the classifier loss was set too high (i.e. $\lambda_{Cls} = 0.044$) and that another experiment should be attempted with a much smaller weight. At the same time it is possible that the additional loss signal from the classifier as part of the joint loss also burdens the generator at the beginning of training. The drop in STL-10 accuracy does indicate, however, that the classifier loss accounts for to a better quality representation. The high FID value for the experiment without the reconstruction loss is not unexpected as the generator can produce anything that fools the discriminator yet is not necessarily a realistic image and one that follows the data distribution, respectively. Note that the IS results (higher is better) suggest the same interpretations as the FID values (lower is better).

The two ablation analysis together tentatively suggest that the full model achieves overall the best performance. However, considering the results after only ten epochs of training, it has not become fully apparent. A longer training period is needed to provide stronger evidence.

Chapter 5

Conclusions and Future Work

In this thesis, we have proposed a novel method LORBMS for unsupervised representation learning and disentangling factors of variation by mixing natural scenes. Unlike many prior methods that work on synthetic or simple image datasets, our method takes the bold approach to learn object representations directly from unprocessed natural image data. By means of a novel clustering algorithm for finding visually similar images, the method uses the matches to mix together new scenes in the latent space of its generator. These mixes are then rendered as realistic images using adversarial training. In this process, the method learns to disentangle image quadrants and objects in the latent space and thereby gains an understanding of the natural image data. We give evidence of this in qualitative evaluations where the model demonstrates the capability of rendering realistic scenes given unseen interpolated representations extracted from real scenes. However, as noticeable visually along with evidence from quantitative evaluations including the FID, the degree of realism achieved by the model still offers room for improvement. We believe that more experiments including the use of standard architectures, increased model capacity, different sets of data augmentations and a much larger, automated hyperparameter search could yield substantial advancements.

Moreover, as transfer learning experiments have indicated, the quality of the learnt representations has not achieved SOTA performance, particularly not when the encoder is used as feature extractor. It seems that the task of learning the complex manifold of unstructured image data and thereby extracting high quality representations is too difficult for the model at this point. We conjecture this is due to a significant degree to the high variance in the image data that makes it extremely difficult for the model to detect and learn low-level and high-level patterns as there is little repetition across samples. Contrasting with the celebA dataset, in each sample there is exactly one face somewhat center-aligned that the model can focus on and build upon. Such a repetitive pattern across our dataset is practically

nonexistent. A possible remedy could be to expand the data augmentation such that each image and object, respectively, is replicated in various ways to give the model more exposure to the same object.

Since a lot of time was dedicated to sequential hyperparameter search, it should be automated and parallelized as much as possible. The hurdle to take is the nonstandard architecture of our method, its implementation and the nonstandard set of hyperparameters that comes with it. A first step would be a random search where the hyperparameters are drawn from a distribution and the training happens in parallel where at the end of the run the best performing model is picked. However, this approach also requires a substantial effort around a nonstandard model which LORBMS is. An even more interesting approach to consider is the recently proposed method called *population based training* by Jaderberg *et al.* [34] which is based on random search but where sets of hyperparameters and their performances are compared repeatedly to the population during training and then exploited and built upon as training continues. Although for an efficient execution of this method many GPUs are required.

We also reconsider the complexity of the dataset used in the training of our method and the poor results achieved in the transfer learning experiments. To that end, it would be interesting to experiment with the idea of using multiple datasets of increasing complexity for training (assuming they exist). The training would commence with a simple dataset where the model could learn more easily, then the model would be transferred for continued training on a more difficult dataset, where it would be harder for the model to generate realistic scenes, and so on until the most complex dataset is reached and the model trained on. As the model is progressively trained on increasingly difficult datasets in a finetuning manner, it might be better prepared at the end to effectively exploit the target dataset and learn object representations of truly high quality.

An interesting avenue for exploration would be to question or enhance the approach to mixing scenes taken so far. As discussed earlier, the decision to split images into quadrants is a very coarse approach. One could try fundamentally different approaches to fragment the input image. For instance, unsupervised segmentation methods (e.g. Xia and Kulis [81]) could split images into a varying number of parts with varying size. And thus better capture objects and distinguish them from surrounding backgrounds. Using the segmentation mask one could devise a much more fine granular mixing algorithm where scenes are mixed according to segments and not quadrants.

Another interesting direction for future work is given by the idea that ultimately not only image objects should be disentangled but also the attributes thereof. Put simply, combine our model for instance with the model by Hu *et al.* [27] such that the learnt latent representation not only exposes disentangled objects as distinct entities but within each object and representation chunk, re-

spectively, also each object's attributes such as pose, color, lighting, etc. This idea, however, will presumably be very difficult to achieve considering how far our model has managed to even disentangle objects.

In terms of limitations, our system by design does not have an explicit notion of object location (or any other attributes). The system models an object in a coarse fashion where an instance is exposed in latent space as a single entity (as part of an image quadrant) and the location, which would be a valuable piece of information to expose explicitly, is somewhere hidden and nonaccessible in that entity. To explore ways so as to capture and expose the location of the image objects would be very interesting. Many prior works have shown that spatial location matters.

In more practical terms, the architecture uses a resolution of 64×64 pixels. The model could be trained on a higher resolution such as 128×128 pixels to observe if the performance increases when using a larger input size and more image details, respectively. In addition, with respect to the visual similarity detection algorithm, the value of k, i.e. the number of clusters for the k-means algorithm, could be reconsidered, explored further and its impact on the scene mixing algorithm better understood.

All in all, we believe our approach has shown that tapping into and learning directly from unstructured natural data lies within the realm of the possible and has tremendous potential for the future yet clearly remains a significant challenge for the research community of today.

List of Figures

List of Tables

Bibliography

[1] P. Anderson, X. He, C. Buehler, D. Teney, M. Johnson, S. Gould, and L. Zhang, *Bottom-up and top-down attention for image captioning and visual question answering*, 2017. eprint: arXiv:1707.07998 (page 13).

[2] M. Aubry, D. Maturana, A. Efros, B. Russell, and J. Sivic, "Seeing 3d chairs: Exemplar part-based 2d-3d alignment using a large dataset of cad models", in *CVPR*, 2014 (page 33).

[3] D. H. Ballard, "Modular learning in neural networks.", in *Proc. AAAI*, 1987, pp. 279–284 (page 22).

[4] Y. Bengio, A. Courville, and P. Vincent, *Representation learning: A review and new perspectives*, 2012. eprint: arXiv:1206.5538 (pages 27, 29–32).

[5] D. Berthelot, C. Raffel, A. Roy, and I. Goodfellow, *Understanding and improving interpolation in autoencoders via an adversarial regularizer*, 2018. eprint: arXiv:1807.07543 (pages 30, 64).

[6] M. Caron, P. Bojanowski, A. Joulin, and M. Douze, *Deep clustering for unsupervised learning of visual features*, 2018. eprint: arXiv:1807.05520 (pages 21, 48, 49, 56).

[7] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, *Encoder-decoder with atrous separable convolution for semantic image segmentation*, 2018. eprint: arXiv:1802.02611 (page 13).

[8] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel, *Infogan: Interpretable representation learning by information maximizing generative adversarial nets*, 2016. eprint: arXiv:1606.03657 (pages 27, 29, 36, 51).

[9] B. Cheung, J. A. Livezey, A. K. Bansal, and B. A. Olshausen, *Discovering hidden factors of variation in deep networks*, 2014. eprint: arXiv:1412.6583 (pages 33, 36).

[10] F. Chollet, *Building autoencoders in keras*, May 2016. [Online]. Available: https://blog.keras.io/building-autoencoders-in-keras.html (visited on 02/12/2019) (page 22).

[11] A. Coates, A. Ng, and H. Lee, "An analysis of single-layer networks in unsupervised feature learning", in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, vol. 15, Apr. 2011, pp. 215–223 (pages 61, 64, 65).

[12] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding", in *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016 (page 34).

[13] Y. L. Cun, B. Boser, J. S. Denker, R. E. Howard, W. Habbard, L. D. Jackel, and D. Henderson, "Advances in neural information processing systems 2", in, D. S. Touretzky, Ed., San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1990, ch. Handwritten Digit Recognition with a Back-propagation Network, pp. 396–404, ISBN: 1-55860-100-7. [Online]. Available: http://dl.acm.org/citation.cfm?id=109230.109279 (page 13).

[14] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database", in *CVPR09*, 2009 (page 13).

[15] A. Dosovitskiy and T. Brox, *Generating images with perceptual similarity metrics based on deep networks*, 2016. eprint: arXiv:1602.02644 (page 31).

[16] C. Eastwood and C. K. I. Williams, "A framework for the quantitative evaluation of disentangled representations", in *International Conference on Learning Representations*, 2018 (pages 32, 34).

[17] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes challenge: A retrospective", *International Journal of Computer Vision*, vol. 111, no. 1, pp. 98–136, Jan. 2015. DOI: 10.1007/s11263-014-0733-5 (page 61).

[18] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position", *Biological cybernetics*, vol. 36, pp. 193–202, Feb. 1980. DOI: 10.1007/BF00344251 (page 13).

[19] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org (pages 29–31).

[20] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, *Generative adversarial networks*, 2014. eprint: arXiv:1406.2661 (pages 23, 25).

[21] K. Greff, R. K. Srivastava, and J. Schmidhuber, *Binding via reconstruction clustering*, 2015. eprint: arXiv:1511.06418 (pages 35, 36).

[22] H. A. Haenssle, C. Fink, R. Schneiderbauer, F. Toberer, T. Buhl, A. Blum, A. Kalloo, A. B. H. Hassen, L. Thomas, A. Enk, L. Uhlmann, C. Alt, M. Arenbergerova, R. Bakos, A. Baltzer, I. Bertlich, A. Blum, T. Bokor-Billmann, J. Bowling, N. Braghiroli, R. Braun, K. Buder-Bakhaya, T. Buhl, H. Cabo, L. Cabrijan, N. Cevic, A. Classen, D. Deltgen, C. Fink, I. Georgieva, L. E. Hakim-Meibodi, S. Hanner, F. Hartmann, J. Hartmann, G. Haus, E. Hoxha, R. Karls, H. Koga, J. Kreusch, A. Lallas, P. Majenka, A. Marghoob, C. Massone, L. Mekokishvili, D. Mestel, V. Meyer, A. Neuberger, K. Nielsen, M. Oliviero, R. Pampena, J. Paoli, E. Pawlik, B. Rao, A. Rendon, T. Russo, A. Sadek, K. Samhaber, R. Schneiderbauer, A. Schweizer, F. Toberer, L. Trennheuser, L. Vlahova, A. Wald, J. Winkler, P. Wolbing, and I. Zalaudek, "Man against machine: diagnostic performance of a deep learning convolutional neural network for dermoscopic melanoma recognition in comparison to 58 dermatologists", *Ann. Oncol.*, vol. 29, no. 8, pp. 1836–1842, Aug. 2018 (page 6).

[23] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, "Gans trained by a two time-scale update rule converge to a local nash equilibrium", 2017. eprint: arXiv:1706.08500 (pages 26, 52, 57, 68).

[24] A. Hindupuravinash, *The gan zoo*, Nov. 2018. [Online]. Available: https://github.com/hindupuravinash/the-gan-zoo (visited on 04/23/2019) (page 25).

[25] G. Hinton, *Lecture 5: Distributed representations*. [Online]. Available: http://www.cs.toronto.edu/~bonner/courses/2014s/csc321/webpages/lectures.htm (visited on 04/29/2019) (page 28).

[26] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators", *Neural Networks*, vol. 2, no. 5, pp. 359 –366, 1989, ISSN: 0893-6080. DOI: https://doi.org/10.1016/0893-6080(89)90020-8. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0893608089900208 (page 12).

[27] Q. Hu, A. Szabó, T. Portenier, M. Zwicker, and P. Favaro, *Disentangling factors of variation by mixing them*, 2017. eprint: arXiv:1711.07410 (pages 22, 23, 25, 35, 37, 38, 42, 47, 51, 56, 72).

[28] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, *Densely connected convolutional networks*, 2016. eprint: arXiv:1608.06993 (pages 13, 54).

[29] Z. Huang, L. Huang, Y. Gong, C. Huang, and X. Wang, *Mask scoring r-cnn*, 2019. eprint: arXiv:1903.00241 (page 13).

[30] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, *Squeezenet: Alexnet-level accuracy with 50x fewer parameters and ¡0.5mb model size*, 2016. eprint: arXiv:1602.07360 (page 92).

[31] S. Iizuka, E. Simo-Serra, and H. Ishikawa, "Globally and locally consistent image completion", *ACM Trans. Graph.*, vol. 36, 107:1–107:14, 2017 (page 92).

[32] S. Ioffe and C. Szegedy, *Batch normalization: Accelerating deep network training by reducing internal covariate shift*, 2015. eprint: arXiv:1502.03167 (page 15).

[33] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, *Image-to-image translation with conditional adversarial networks*, 2016. eprint: arXiv:1611.07004 (pages 27, 51, 56, 92).

[34] M. Jaderberg, V. Dalibard, S. Osindero, W. M. Czarnecki, J. Donahue, A. Razavi, O. Vinyals, T. Green, I. Dunning, K. Simonyan, C. Fernando, and K. Kavukcuoglu, *Population based training of neural networks*, 2017. eprint: arXiv:1711.09846 (page 72).

[35] S. Jenni and P. Favaro, *Self-supervised feature learning by learning to spot artifacts*, 2018. eprint: arXiv:1806.05024 (pages 24, 65–67).

[36] J. Johnson, M. Douze, and H. Jégou, "Billion-scale similarity search with gpus", *arXiv preprint arXiv:1702.08734*, 2017 (page 56).

[37] S. Jégou, M. Drozdzal, D. Vazquez, A. Romero, and Y. Bengio, *The one hundred layers tiramisu: Fully convolutional densenets for semantic segmentation*, 2016. eprint: arXiv:1611.09326 (page 54).

[38] T. Karras, S. Laine, and T. Aila, *A style-based generator architecture for generative adversarial networks*, 2018. eprint: arXiv:1812.04948 (pages 30, 57, 68).

[39] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, 2014. eprint: arXiv:1412.6980 (pages 52, 65).

[40] D. P. Kingma and M. Welling, *Auto-encoding variational bayes*, 2013. eprint: arXiv:1312.6114 (page 29).

[41] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks", *Commun. ACM*, vol. 60, no. 6, pp. 84–90, May 2017, ISSN: 0001-0782. DOI: 10.1145/3065386. [Online]. Available: http://doi.acm.org/10.1145/3065386 (pages 13, 19).

[42] P. Krähenbühl, C. Doersch, J. Donahue, and T. Darrell, *Data-dependent initializations of convolutional neural networks*, 2015. eprint: arXiv:1511.06856 (page 66).

[43] A. Kumar, P. Sattigeri, and A. Balakrishnan, *Variational inference of disentangled latent concepts from unlabeled observations*, 2017. eprint: arXiv:1711.00848 (pages 33, 36).

[44] K. Kurach, M. Lucic, X. Zhai, M. Michalski, and S. Gelly, *The gan landscape: Losses, architectures, regularization, and normalization*, 2018. eprint: arXiv:1807.04720 (page 25).

[45] H. Le and A. Borji, *What are the receptive, effective receptive, and projective fields of neurons in convolutional neural networks?*, 2017. eprint: arXiv:1705.07049 (page 20).

[46] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition", *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998, ISSN: 0018-9219. DOI: 10.1109/5.726791 (page 27).

[47] Y. LeCun, Fu Jie Huang, and L. Bottou, "Learning methods for generic object recognition with invariance to pose and lighting", in *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, vol. 2, Jun. 2004, II–104 Vol.2. DOI: 10.1109/CVPR.2004.1315150 (page 34).

[48] R. Lerch, *Official inkscape tutorial: Using the interpolation extension*, Sep. 2013. [Online]. Available: https://inkscapetutorials.wordpress.com/2008/01/30/official-inkscape-tutorial-using-the-interpolation-extension/ (visited on 02/13/2019) (page 31).

[49] S. Li, *Calculating receptive field of cnn*, Feb. 2017. [Online]. Available: http://shawnleezx.github.io/blog/2017/02/11/calculating-receptive-field-of-cnn/ (visited on 02/11/2019) (page 20).

[50] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár, *Microsoft coco: Common objects in context*, 2014. eprint: arXiv:1405.0312 (pages 34, 49, 59).

[51] S. Linnainmaa, "The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors", Master's thesis, Univ. Helsinki, 1970 (page 11).

[52] M.-Y. Liu, T. Breuel, and J. Kautz, *Unsupervised image-to-image translation networks*, 2017. eprint: arXiv:1703.00848 (pages 29, 33, 36).

[53] R. Liu, J. Lehman, P. Molino, F. P. Such, E. Frank, A. Sergeev, and J. Yosinski, *An intriguing failing of convolutional neural networks and the coordconv solution*, 2018. eprint: arXiv:1807.03247 (pages 54, 56, 91).

[54] Z. Liu, P. Luo, X. Wang, and X. Tang, "Deep learning face attributes in the wild", in *Proceedings of International Conference on Computer Vision (ICCV)* (pages 9, 27).

[55] F. Locatello, S. Bauer, M. Lucic, S. Gelly, B. Schölkopf, and O. Bachem, *Challenging common assumptions in the unsupervised learning of disentangled representations*, 2018. eprint: arXiv:1811.12359 (page 32).

[56] M. Mathieu, C. Couprie, and Y. LeCun, *Deep multi-scale video prediction beyond mean square error*, 2015. eprint: arXiv:1511.05440 (pages 25, 47).

[57] M. Mathieu, J. Zhao, P. Sprechmann, A. Ramesh, and Y. LeCun, *Disentangling factors of variation in deep representations using adversarial training*, 2016. eprint: arXiv:1611.03383 (pages 33, 36).

[58] L. Matthey, I. Higgins, D. Hassabis, and A. Lerchner, *Dsprites: Disentanglement testing sprites dataset*, https://github.com/deepmind/dsprites-dataset/, 2017 (page 35).

[59] L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner, "Beta-vae: Learning basic visual concepts with a constrained variational framework", in *ICLR*, 2017 (pages 29, 33, 36).

[60] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida, *Spectral normalization for generative adversarial networks*, 2018. eprint: arXiv:1802.05957 (pages 15, 25, 26, 68).

[61] M. Noroozi, A. Vinjimoor, P. Favaro, and H. Pirsiavash, *Boosting self-supervised learning via knowledge transfer*, 2018. eprint: arXiv:1805.00385 (pages 66, 67).

[62] A. Odena, V. Dumoulin, and C. Olah, *Deconvolution and checkerboard artifacts*, Oct. 2016. [Online]. Available: https://distill.pub/2016/deconv-checkerboard/ (visited on 02/01/2019) (page 54).

[63] D. Pathak, P. Krahenbuhl, J. Donahue, T. Darrell, and A. A. Efros, "Context encoders: Feature learning by inpainting", 2016. eprint: arXiv:1604.07379 (page 25).

[64] P. Paysan, R. Knothe, B. Amberg, S. Romdhani, and T. Vetter, "A 3d face model for pose and illumination invariant face recognition", in *2009 Sixth IEEE International Conference on Advanced Video and Signal Based Surveillance*, Sep. 2009, pp. 296–301 (page 27).

[65] Prabhu, *Understanding of convolutional neural network (cnn) - deep learning*, Mar. 2018. [Online]. Available: https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148 (visited on 02/19/2019) (page 14).

[66] A. Radford, L. Metz, and S. Chintala, *Unsupervised representation learning with deep convolutional generative adversarial networks*, 2015. eprint: arXiv:1511.06434 (pages 25, 51, 64).

[67] S. E. Reed, Y. Zhang, Y. Zhang, and H. Lee, "Deep visual analogy-making", in *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds., Curran Associates, Inc., 2015, pp. 1252–1260. [Online]. Available: http://papers.nips.cc/paper/5845-deep-visual-analogy-making.pdf (pages 23, 34).

[68] T. Remez, J. Huang, and M. Brown, *Learning to segment via cut-and-paste*, 2018. eprint: arXiv:1803.06414 (page 34).

[69] S. Ren, K. He, R. Girshick, and J. Sun, *Faster r-cnn: Towards real-time object detection with region proposal networks*, 2015. eprint: arXiv:1506.01497 (page 13).

[70] F. Rosenblatt, *The Perceptron, a Perceiving and Recognizing Automaton Project Para*, ser. Report: Cornell Aeronautical Laboratory. Cornell Aeronautical Laboratory, 1957. [Online]. Available: https://books.google.ch/books?id=P_XGPgAACAAJ (page 11).

[71] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge", *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015. DOI: 10.1007/s11263-015-0816-y (page 61).

[72] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, *Improved techniques for training gans*, 2016. eprint: arXiv:1606.03498 (pages 24, 26, 57).

[73] J. Schmidhuber, "Deep learning in neural networks: An overview", *Neural Networks*, vol. 61, pp. 85–117, 2015. DOI: 10.1016/j.neunet.2014.09.003 (pages 11, 22).

[74] Z. Singer, *Manifolds in data science - a brief overview*, Jan. 2019. [Online]. Available: https://towardsdatascience.com/manifolds-in-data-science-a-brief-overview-2e9dde9437e5 (visited on 05/18/2019) (page 29).

[75] K. Swersky, J. Snoek, and R. P. Adams, "Multi-task bayesian optimization", in *Advances in Neural Information Processing Systems 26*, 2013, pp. 2004–2012 (page 66).

[76] A. Szabo, Q. Hu, T. Portenier, M. Zwicker, and P. Favaro, "Understanding degeneracies and ambiguities in attribute transfer", in *The European Conference on Computer Vision (ECCV)*, Sep. 2018 (pages 35, 36).

[77] A. Szabó, Q. Hu, T. Portenier, M. Zwicker, and P. Favaro, *Challenges in disentangling independent factors of variation*, 2017. eprint: arXiv:1711.02245 (pages 35, 36, 46).

[78] D. Ulyanov, A. Vedaldi, and V. Lempitsky, *Instance normalization: The missing ingredient for fast stylization*, 2016. eprint: arXiv:1607.08022 (page 15).

[79] N. Watters, L. Matthey, C. P. Burgess, and A. Lerchner, *Spatial broadcast decoder: A simple architecture for learning disentangled representations in vaes*, 2019. eprint: arXiv:1901.07017 (pages 34, 42, 91).

[80] Y. Wu and K. He, *Group normalization*, 2018. eprint: arXiv:1803.08494 (page 15).

[81] X. Xia and B. Kulis, *W-net: A deep model for fully unsupervised image segmentation*, 2017. eprint: arXiv:1711.08506 (page 72).

[82] T. Xiao, J. Hong, and J. Ma, *Dna-gan: Learning disentangled representations from multi-attribute images*, 2017. eprint: arXiv:1711.05415 (pages 26, 29, 36).

[83] J. Yosinski, J. Clune, A. Nguyen, T. Fuchs, and H. Lipson, *Understanding neural networks through deep visualization*, 2015. eprint: arXiv:1506.06579 (page 18).

[84] H. Zhang, I. Goodfellow, D. Metaxas, and A. Odena, *Self-attention generative adversarial networks*, 2018. eprint: arXiv:1805.08318 (pages 26, 54, 68, 92).

Appendix A

Visual similarity detection algorithm

The pseudocode of the algorithm is given in Algorithm 2.

```
   Input: a dataset X
   Output: dataset X' with nearest neighbor annotations
 1 begin
 2     foreach image x ∈ X do
 3         slice x into quadrants x_q1, x_q2, x_q3, x_q4
 4         create datasets x_q1 ∈ X_q1, x_q2 ∈ X_q2, x_q3 ∈ X_q3, x_q4 ∈ X_q4
 5     end
 6     foreach X_qi do
 7         compute representations (reps) for all images in X_qi
 8         cluster reps with k-means
 9         foreach cluster c do
10             foreach representation f ∈ c do
11                 compute 10 nearest reps tnf in c with k-nn and L2 distance
12                 create list tni of 10 nearest neighbor images using tnf
13                 annotate image x_qi belonging to f with tni
14             end
15         end
16     end
17     foreach image x in X do
18         foreach X_qi do
19             annotate x with list tni from corresponding x_qi
20         end
21     end
22 end
```

Algorithm 2: Pseudocode for the visual similarity detection algorithm.

Appendix B

Sheets of mixed scene renderings

In addition to Section 4.2.1, we show more test examples of mixed scene renderings using our model in Figures B.1 and B.2. Note that the mixes are done automatically by the latent space scene mixing algorithm and our model, respectively.

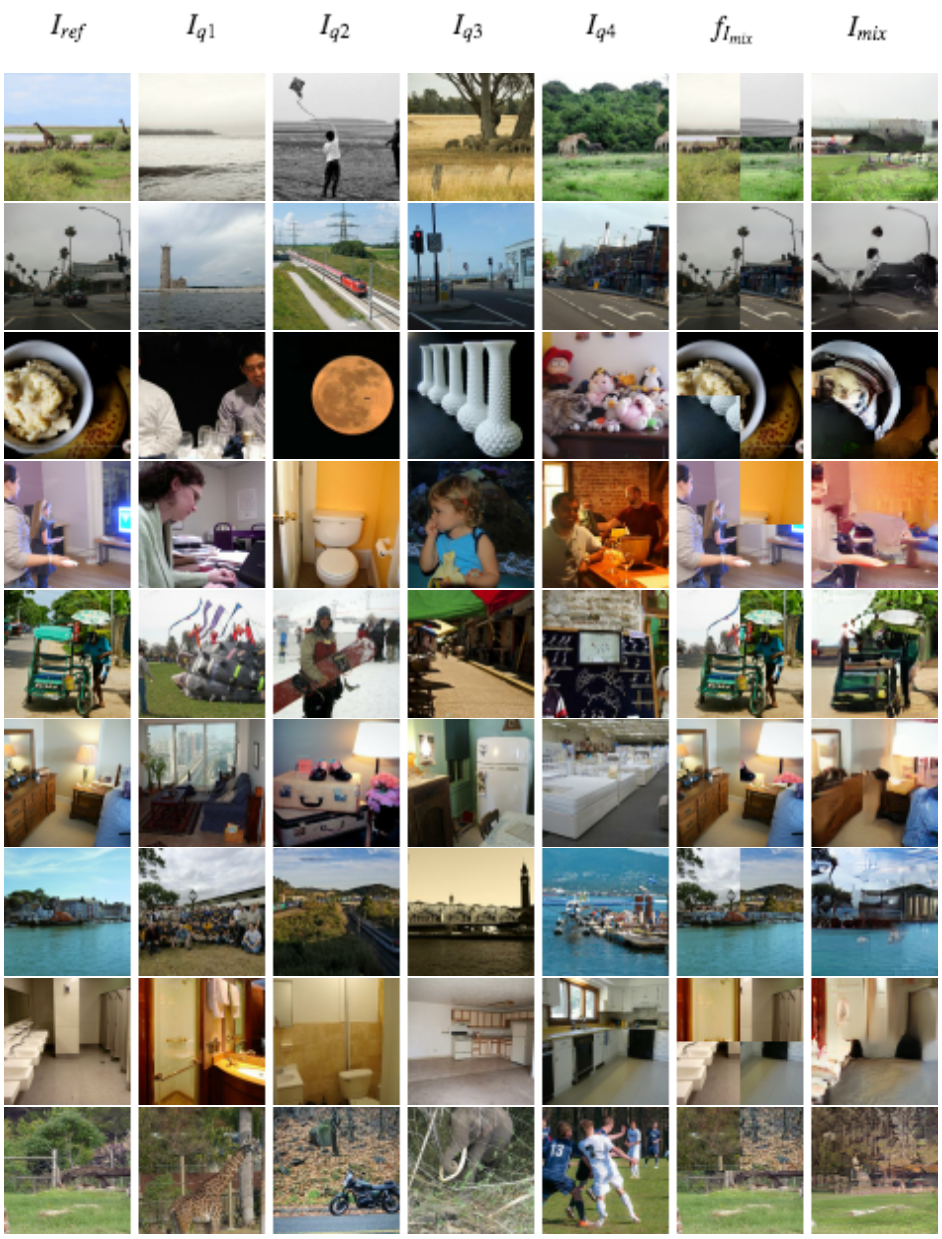


Figure B.1: (Sheet 1) Mixed scene renderings shown in the rightmost column produced by our method with the COCO dataset at 64×64 pixels. See Figure 4.2 for further explanations.

Figure B.2: (Sheet 2) Mixed scene renderings shown in the rightmost column produced by our method with the COCO dataset at 64×64 pixels. See Figure 4.2 for further explanations.

Appendix C

More object transfer experiments

Here we show more object transfer renderings using our model in Table C.1. The two examples at the bottom lack coherence between the perspectives in the images to mix and the model has difficulty in coping with that.

I_{obj}	I_{ref}	$mask$	I_{mix}
		1100	
		1100	
		1100	
		0010	
		1010	
		0101	

Table C.1: More object transfer renderings. An object spans one or two quadrants.

Appendix D

Experiments with non-significant results

"I have not failed. I've just found 10,000 ways that won't work."

—Thomas A. Edison

Here we list conducted experiments that did not produce convincing results or failed altogether but still offer a lesson to learn from.

D.1 Mixing scenes with 9 tiles and a random mask

At an earlier stage of the thesis, the model used a more simplistic approach to mix scenes. The model first split each training batch in half, then sliced each image into 9 equally sized tiles and encoded them individually. A randomly generated binary mask was used to mix each image and representation, respectively, from the first half of the training batch with the corresponding image (and representation) from the second half of the training batch. Thus the selection of the pair of images that were used for the mixing was pure coincidence. This procedure created a lot of meaningless scene mixes that were simply too difficult for the model to render realistically and learn from. We eventually abandoned this approach in search of a better scene mixing algorithm.

D.2 Mixing scenes with 4 tiles and least L2 distance

In this approach the model uses 4 tiles and quadrants, respectively, to do the mixing. Given a training batch, it computes the representations of all images. It then takes one image as a reference and computes the L2 distances between

its representation and the representations of the other images. Using a threshold, it replaces those tiles of the reference image where another image and its representations, respectively, is below the threshold and in addition ensures that at least one tile from the reference image remains. While this mixing approach performed somewhat better, the results were still too weak and it evidenced that mixing scenes only within the scope of one batch of training images was not sufficient to create meaningful mixes and to assist the model in learning, respectively.

D.3 A classifier task too simple

Before we redefined the classifier task including a two image input as shown in Figure 3.5, the classifier received six images as input. The reference image I_{ref}, the four quadrant replacement images I_{qi} and the generated image I_{mix}. For each quadrant in I_{mix}, the classifier had to decide the origin of that quadrant, i.e. whether I_{ref} was the source or the respective I_{qi}. A straightforward binary classification problem. As the progression of the classifier loss during training clearly indicated, this classifier task was too simple. The loss very quickly converged towards zero and the classifier had an easy job in making accurate decisions. For that reason we redefined the classification task to make it more challenging.

D.4 CoordConv discriminator

We use a CoordConv [53] layer as the first layer in our encoder architecture. Inspired by the experiments in [53], we also experimented with CoordConv layers in the discriminator. One experiment included a CoordConv layer in front of every convolutional layer in the discriminator together with a self-attention module. The training of that model clearly did not yield improved results, the generated images were of little quality and we dismissed the use of CoordConv layers in the discriminator altogether.

D.5 Spatial Broadcast Decoder

We implement the spatial broadcast decoder from Watters *et al.* [79] and replace our decoder with it. The rationale behind this experiment is that the spatial broadcast decoder introduces a notion of spatial location to the model which in theory should benefit the model for handling positional features in the input. According to the paper, this type of decoder should help the network to learn disentangled representations even though originally it is applied as the decoder in a

variational autoencoder (VAE). Nonetheless, we trained our method with the spatial broadcast decoder. However, with no exception, the model collapsed within the first few iterations and would not recover. For time reasons we then dismissed using this decoder.

D.6 Self-attention GAN

We tried to incorporate the self-attention module from SAGAN [84] into our networks with the rationale as described in Section 2.4.2. We added the module as an extra layer in the decoder and also in the discriminator. While we could not observe an improvement from the self-attention module in the decoder, the discriminator showed promising results. Therefore, we kept using the self-attention module in the discriminator as it contributed to the best performance at that time. It was not until we experimented with the PatchGAN [33] discriminator, which outperformed the self-attention version, that we dismissed it and instead used the PatchGAN version as our final discriminator.

D.7 Global local discriminator

As a measure to increase the quality of the generated scenes, we ran experiments with the global local discriminator as proposed by Iizuka *et al.* [31]. This discriminator consists of two separate sub-discriminators where the global one looks at the entire input (e.g. 64×64 pixels) and the local one considers only a random patch of the image (e.g. 32×32 pixels). Together they decide on the realism of the given image. Intuitively, the global local discriminator should help the generator increase the sharpness of the generated image as it not only considers the image as a whole but also focuses on local details (i.e. the patch). The experiment with the global local discriminator did not outperform the other models under test and thus we dismissed it. The lack of improvement with the global local discriminator could be attributed to the comparably small image resolution of 64×64 pixels that we were using for our model. It is likely that the global local discriminator would make more difference if the image resolution was doubled or even further increased.

D.8 A note on SqueezeNet

We take SqueezeNet [30] instead of AlexNet as the classifier network implementation with the aim to save memory consumption and in favor of a bigger batch size for faster training. In the training process however we noticed that

SqueezeNet would permanently make an assignment prediction of 0.5 even after the first iteration no matter the input whereas AlexNet would take up the training process as expected. A look into SqueezeNet's penultimate layer before the final average pooling layer revealed that its activation was all 0 after the first backpropagation already. We conjecture that with a cross entropy loss function where most ground truth values are 0 (i.e. actual assignments), SqueezeNet suffers from a lack of numerical stability around 0 and therefore collapses to 0 from which is does not recover. This behavior can be reproduced consistently with the test program `test_squeezenet_0_collapse.py`. We dismissed the idea of using SqueezeNet altogether.

Declaration of consent

on the basis of Article 30 of the RSL Phil.-nat. 18

Name/First Name: Zbinden Lukas

Registration Number: 01-195-346

Study program: Computer Science

Bachelor ☐ Master ☑ Dissertation ☐

Title of the thesis: Learning Object Representations by Mixing Scenes

Supervisor: Prof. Dr. Paolo Favaro

I declare herewith that this thesis is my own work and that I have not used any sources other than those stated. I have indicated the adoption of quotations as well as thoughts taken from other authors as such in the thesis. I am aware that the Senate pursuant to Article 36 paragraph 1 litera r of the University Act of 5 September, 1996 is authorized to revoke the title awarded on the basis of this thesis.

For the purposes of evaluation and verification of compliance with the declaration of originality and the regulations governing plagiarism, I hereby grant the University of Bern the right to process my personal data and to perform the acts of use this requires, in particular, to reproduce the written thesis and to store it permanently in a database, and to use said database, or to make said database available, to enable comparison with future theses submitted by others.

Place/Date

Signature